

DSA – Formação Cientista de Dados

7. Machine Learning

7.1. Introdução

O que é Aprendizado de Máquina?

O que é Aprendizado?

Aprendizado é a capacidade de se adaptar, modificar e melhorar seu comportamento e suas respostas, sendo portanto uma das propriedades mais importantes dos seres ditos inteligentes, sejam eles humanos ou não.

Características do Aprendizado:

- Adaptação
- Correção
- Otimização
- Representação
- Interação

Há grande semelhança entre o processo de aprendizado de seres humanos e através de algoritmos de Machine Learning!

Já podemos então definir Aprendizado de Máquina!

Machine Learning é um subcampo da Inteligência Artificial que permite dar aos computadores a habilidade de aprender sem que sejam explicitamente programados para isso!

Machine Learning ou Aprendizado de Máquina é um método de análise de dados que automatiza o desenvolvimento de modelos analíticos. Usando algoritmos que aprendem iterativamente a partir de dados, o aprendizado de máquina permite que os computadores encontrem insights ocultos sem serem explicitamente programados para procurar algo específico.

Tipos de Aprendizagem:

- Supervisionada
- Não Supervisionada
- Semi Supervisionada
- Aprendizagem Por Reforço
- Deep Learning

Definição de Inteligência:

- Dotado de inteligência, capaz de compreender, esperto, habilidoso
- Faculdade de conhecer, de aprender, de conceber, de compreender: a inteligência distingue o homem do animal.

Definição de Inteligência Artificial:

- Conjunto de teorias e de técnicas empregadas com a finalidade de desenvolver máquinas capazes de simular a inteligência humana.
- A Inteligência Artificial é uma área de estudos da computação que se interessa pelo estudo e criação de sistemas que possam exibir um comportamento inteligente e realizar tarefas complexas com um nível de competência que é equivalente ou superior ao de um especialista humano.

Por que Machine Learning Está Transformando o Mundo?

Algoritmos de aprendizagem de máquina, aprendem a induzir uma função ou hipótese capaz de resolver um problema a partir de dados que representam instâncias do problema a ser resolvido.

Exemplos onde o aprendizado de máquina está sendo aplicado com sucesso:

- Reconhecimento de Voz
- Previsão de Doenças
- Diagnóstico de Câncer
- Detecção de Fraudes em Cartões de Crédito
- Carros Autônomos
- Detecção de Anomalias
- Resultados de pesquisa na Web
- Previsão de falhas em equipamento
- Anúncios em tempo real em páginas da web e dispositivos móveis
- Análise de sentimento baseada em texto
- Filtragem de spams no e-mail
- Detecção de invasão na rede
- Pontuação de crédito e próximas melhores ofertas

Machine Learning não está transformando nosso mundo; Machine Learning já transformou o nosso mundo.

The Dark Side of Big Data

Recomendação de leitura:

Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy

Processo de Aprendizagem

Problema de Negócio ↔ Análise de Dados → Preparação dos Dados → Modelagem → Avaliação → Deploy

Tipos de Aprendizagem

Aprendizagem Supervisionada

É o termo usado sempre que o programa é “treinado” sobre um conjunto de dados pré-definido.

Os algoritmos de aprendizado supervisionado fazem previsões com base em um conjunto de exemplos.

Fluxo:

Dados de Treino → Algoritmo de Machine Learning → Modelo Preditivo

Novos Dados → Modelo Preditivo → Previsões

Tipos de Aprendizagem Supervisionada:

- Classificação
- Regressão

Aprendizagem Não Supervisionada

Termo usado quando um programa pode automaticamente encontrar padrões e relações em um conjunto de dados.

O objetivo de um algoritmo de aprendizado não supervisionado é organizar os dados de alguma forma ou descrever sua estrutura.

Este tipo de aprendizado, assemelha-se aos métodos que nós seres humanos usamos para descobrir se certos objetos ou eventos são da mesma classe.

Tipos de Aprendizagem Não Supervisionada:

- Clustering

Os exemplos mais comuns são o K-Means, o Singular Value Decomposition (SVD) e o Principal Component Analysis (PCA).

Aprendizagem Por Reforço

Reinforcement Learning é similar ao que chamamos de aprender por tentativa e erro.

Fluxo:

Agente → (Ação) → Ambiente → (Recompensa e/ou Estado) → Agente → (Ação) → ...

No aprendizado por reforço, o algoritmo escolhe uma ação em resposta a cada ponto de dados.

O aprendizado por reforço é comum em robótica, em que o conjunto de leituras do sensor, em um ponto no tempo, é um ponto de dados e o algoritmo deve escolher a próxima ação do robô.

A ideia básica é simplesmente capturar os aspectos mais importantes do problema real que um agente de aprendizado enfrenta durante a interação com o ambiente para alcançar uma meta.

Aprendizagem Supervisionada

Classificação

Podemos representar a realidade e toda sua complexidade através de funções matemáticas.

Classificação é o processo de identificar a qual conjunto de categorias uma nova observação pertence, com base em um conjunto de dados de treino contendo observações (ou instâncias) cuja associação é conhecida. Exemplo: determinar o diagnóstico de uma doença em um paciente, observando as características similares em outros grupos de pacientes.

Regressão

Um estudo de regressão busca, essencialmente, associar uma variável Y (denominada variável resposta ou variável dependente) a uma outra variável X (denominada variável explanatória ou variável independente).

Como a Regressão pode ser usada?

- Investigação Científica
- Relações Causais
- Identificação de Padrões

IMPORTANTE: Correlação Não Implica Causalidade! Só porque (A) acontece juntamente com (B) não significa que (A) causa (B).

A regressão pode ter:

- Relacionamento Linear Positivo
- Relacionamento Linear Negativo
- Sem Relacionamento Linear

Aprendizagem Não Supervisionada

Clusterização

Algoritmos de Aprendizagem Não Supervisionada (Categoria: Algoritmo):

- Algoritmos Baseados em Centroides: K-means, Gaussian Mixture Model, Fuzzy c-mean
- Algoritmos Baseados em Conectividade: Algoritmos hierárquicos
- Algoritmos Baseados em Densidade: DBSCAN, Optics
- Probabilísticos: LDA
- Redução de Dimensionalidade: tSNE, PCA, KPCA
- Redes Neurais / Deep Learning: Autoencoders

Como Selecionar o Algoritmo Ideal para Cada Problema?

São muitos os algoritmos de Machine Learning:

- Árvores de decisão
- Random Forests

- Descoberta de associações e sequência
- Boosting e bagging de gradiente
- Máquinas de vetores de suporte
- Redes neurais
- Mapeamento de nearest-neighbor
- Cluster k-means
- Mapas auto-organizáveis
- Técnicas de otimização de busca local (por ex., algoritmos genéticos)
- Maximização da expectativa
- Análise Multivariada - Adaptive regression splines
- Redes Bayesianas
- Kernel para estimativa de densidade
- Análise de componentes principais
- Decomposição do valor singular
- Deep Learning

Antes de selecionar um deles, é necessário avaliar questões de:

- Precisão
- Tempo de Treinamento
- Número de Recursos
- Tipo de Problema a ser Resolvido
- Número de Parâmetros
- Linearidade

Comparação entre os principais algoritmos:

- Classificação Binária (2 classes):

Algoritmo	Tempo de Treinamento	Precisão	Linearidade
Regressão Logística	Moderado	Moderado	Moderado
Árvore de Decisão	Moderado	Moderado	N/A
Random Forest	Moderado	Alto	N/A
Redes Neurais	Alto	Moderado	N/A
SVM	Alto	Moderado	Moderado
Métodos Bayesianos	Moderado	Moderado	Moderado

- Classificação Multiclasse (mais de 2 classes):

Algoritmo	Tempo de Treinamento	Precisão	Linearidade
Regressão Logística	Alto	Moderado	Alto
Árvore de Decisão	Alto	Alto	N/A
Random Forest	Alto	Alto	N/A
Redes Neurais	Alto	Alto	N/A
SVM	Alto	Alto	Alto

- Regressão:

Algoritmo	Tempo de Treinamento	Precisão	Linearidade
-----------	----------------------	----------	-------------

Linear	Moderado	Moderado	Alto
Árvore de Decisão	Moderado	Alto	N/A
Random Forest	Alto	Alto	N/A
Redes Neurais	Alto	Alto	N/A
Poisson	Alto	Alto	Alto

- Não Supervisionados:

Algoritmo	Tempo de Treinamento	Precisão	Linearidade
K-Means	Alto	Moderado	Alto
PCA	Alto	Alto	Alto

Como Tratar a Mudança de Versão de Software

Vou explicar aqui uma questão importante, já que temos alunos que não vieram da área de TI.

Oportunidade de aprendizado. Leiam com atenção.

Como profissionais de dados, precisamos compreender que a tecnologia evolui. E em Ciência de Dados evolui ainda mais rápido. É algo que precisamos incorporar ao nosso trabalho e dia a dia.

Todo software tem versão e release. Uma nova versão costuma trazer diversas mudanças, enquanto um novo release costuma ser correção de bugs e falhas de segurança ou pequenas mudanças. Por exemplo:

A Linguagem R recebeu recentemente uma nova versão, passando da versão 3.x para a versão 4.x.

Uma mudança de release seria uma atualização de 3.5 para 3.6 (nesse caso chamado de major release) ou 3.6.1 para 3.6.2 (nesse caso chamado de minor release), por exemplo.

Isso vale praticamente para qualquer software, incluindo linguagens de programação. Python, Java, Hadoop, Spark, etc...

Com ferramentas open-source, a evolução é muito rápida e 2 ou 3 releases costumam ser lançados por ano, com uma nova versão a cada 1, 1.5 ou 2 anos, em geral.

Quando muda a versão, como aconteceu recentemente com a Linguagem R, vários pacotes deixam de funcionar. Por quê? Porque o pacote não suporta alguma mudança na nova versão.

Como resultado, ficamos em um período "nebuloso" por 3 a 4 meses, até que os desenvolvedores migrem seus pacotes para funcionar com a nova versão. É assim desde plugins do Wordpress, até apps nos smartphones. E aí entra onde quero chegar.

Não podemos migrar o curso para a nova versão da Linguagem R agora, pois vários pacotes ainda não foram migrados e não funcionarão e ao manter a versão atual, alguns pacotes já migrados podem não funcionar. E como fazemos então?

Toda solução open-source tem o chamado "archive", onde mantém TODAS as versões do software desde a versão 1. Todas as versões e releases ficam no archive.

Quando eu acesso o site principal do software e a versão não está disponível, eu sigo esses passos:

1- Abro o Google

2- Digito (por exemplo): spark archive download

3- Baixo a versão do meu interesse

Na maioria dos casos é o primeiro ou segundo link da busca. E então instalo a versão que eu preciso.

No caso da Linguagem R por exemplo, eu ainda estou usando a versão 3.6.3. Só migrarei para a versão 4.x quando o período “nebuloso” a que me referi acima passar, dentro de 3 a 4 meses do lançamento. Estamos agora no segundo mês.

Mas e se um pacote foi migrado e parou de funcionar? O que eu faço? Simples: busco a versão anterior e instalo offline. Sigo esses passos:

1- Pesquiso no Google, por: nome_pacote r package archive

2- Acesso o archive e faço download da versão que preciso (arquivo zip)

3- E então instalo assim:

```
install.packages('caminho_para_o_pacote/package.zip', repos = NULL)
```

Ou seja, instalo a partir do arquivo com a versão anterior. Mas tem que colocar repos = NULL, para que o instalador não tente buscar no repositório ativo da Linguagem R.

Essa dica também é boa para quem usa Linguagem R em empresas que bloqueiam o download de pacotes (caso de muitos alunos). Apenas verifique com a TI se não há problema em fazer a instalação na máquina.

O problema é se o pacote tiver muitas dependências. Ai, tem que repetir o procedimento para cada pacote.

Essa é a regra para qualquer software open-source. Em Python é bem mais fácil, pois basta informar a versão ao usar o pip, por exemplo:

```
pip install tensorflow==2.2.0 (sinal de igual duas vezes mesmo)
```

Já pensamos em criar na DSA o nosso repositório com as versões dos softwares que usamos nos cursos, para tentar reduzir os chamados de suporte. Mas eu sou contra isso, pois acho que podemos orientar e ensinar aos alunos um pouco mais do que vieram buscar, já que estão aqui para aprender e são todos profissionais. O Suporte DSA está orientado a explicar ao aluno o problema da mudança de versão e ensinar como resolver o problema, pois isso será útil no futuro quando o aluno estiver usando a solução no dia a dia.

Lidar com mudança de versão de software faz parte do trabalho, pois será realidade com a qual teremos que conviver. Ou será que alguém ainda usa Windows 98?

Estou colocando este texto em um manual em pdf no Capítulo 1 de todos os cursos da Formação Cientista de Dados, que é onde os alunos costumam ter mais dificuldade com as mudanças de versões.

7.2. Algoritmos de Machine Learning e Modelos Preditivos

Introdução

Dados + Análise = Valor

Modelos Descritivos:

- Quantos clientes perdemos nos últimos 3 meses?
- As fraudes aumentaram ou diminuiram no último ano?

Modelos Preditivos:

- Quantos clientes podemos conquistar nos próximos 3 meses?
- As fraudes aumentarão ou diminuirão no próximo ano?

Modelo Preditivo é uma função matemática que, aplicada a uma massa de dados, consegue identificar padrões ocultos e prever o que poderá ocorrer.

- Aprendizagem Supervisionada
 - Métodos Baseados em Instância
 - Métodos Probabilísticos
- Aprendizagem Não Supervisionada
 - Métodos Baseados em Procura
 - Métodos Baseados em Otimização

A construção de bons modelos preditivos implica o domínio de um conjunto de metodologias e conceitos sem os quais a qualidade poderá ser afetada.

O objetivo da análise preditiva é ir além das estatísticas e mostrar, através dos dados coletados, uma melhor visão do que vai acontecer no futuro. Assim é possível coletar insights que levarão a decisões melhores.

A Importância do Modelo Preditivo

Suponhamos uma operadora de telefonia móvel. Um dos principais problemas de empresas deste segmento é a taxa de perda de clientes ou churn rate.

Agregando ao modelo regras de negócio, como agrupar clientes por rentabilidade, a operadora pode fazer ofertas diferenciadas para evitar a desconexão.

Quanto aos dados, estes devem ser:

- Dados em Volume Adequado
- Dados Válidos

A escolha do modelo é importante e diversas variáveis devem ser consideradas.

Criar iniciativas de Big Data Analytics, não é simplesmente adquirir tecnologias.

Identifique com a maior precisão possível o problema de negócio. Quanto mais precisa a pergunta, mais precisa será a resposta e, portanto, maior o valor da resposta.

Mas não superestime o valor da predição. Mesmo em uma sociedade cada vez mais data-driven, a intuição muitas vezes é necessária.

Tenha dados em volume e qualidade adequados. Sem qualidade, o volume não tem valor.

E não subestime o desafio da implementação. Não basta ter apenas a tecnologia, é necessário expertise (conhecimento do negócio, tecnologia, modelagem) para fazer a coisa acontecer.

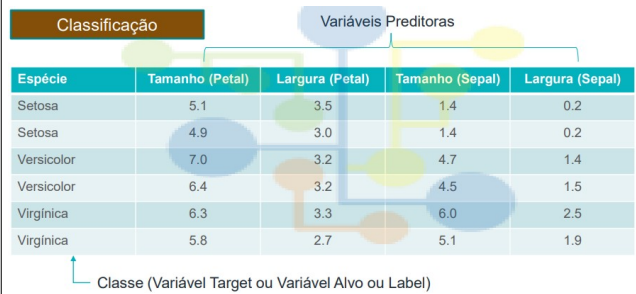
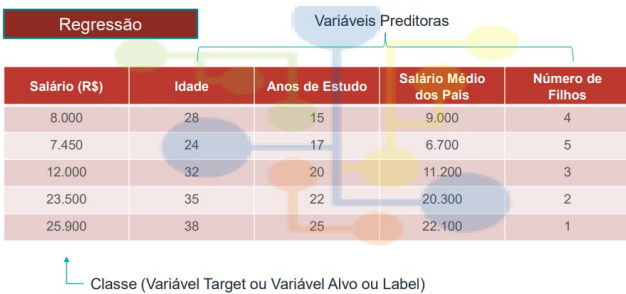
O que é um Modelo Preditivo?

Modelo Preditivo é uma função matemática, aproximada, encontrada através do treinamento com dados e que permite fazer previsões.

O modelo pode empregar uma equação linear simples, ou pode ser uma rede neural complexa.

Como o Modelo Preditivo é Construído?

O objetivo do aprendizado de máquina é aprender a aproximação da função f que melhor representa a relação entre os atributos de entrada (chamadas variáveis preditoras) com a variável target.

Classificação					Regressão				
									
Espécie	Tamanho (Petal)	Largura (Petal)	Tamanho (Sepal)	Largura (Sepal)	Salário (R\$)	Idade	Anos de Estudo	Salário Médio dos Pais	Número de Filhos
Setosa	5.1	3.5	1.4	0.2	8.000	28	15	9.000	4
Setosa	4.9	3.0	1.4	0.2	7.450	24	17	6.700	5
Versicolor	7.0	3.2	4.7	1.4	12.000	32	20	11.200	3
Versicolor	6.4	3.2	4.5	1.5	23.500	35	22	20.300	2
Virginica	6.3	3.3	6.0	2.5	25.900	38	25	22.100	1
Virginica	5.8	2.7	5.1	1.9					

Principais Métodos de Aprendizagem

Os principais métodos de aprendizagem são:

- Métodos Baseados em Instância
- Métodos Probabilísticos
- Métodos Baseados em Procura
- Métodos Baseados em Otimização

Aprendizagem Baseada em Instância

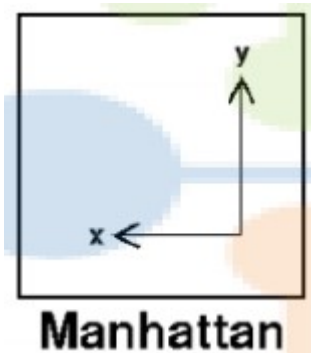
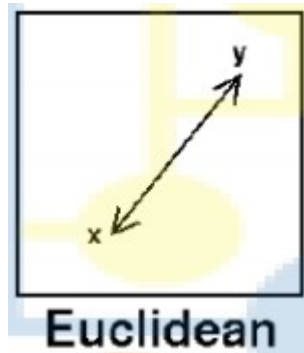
A aprendizagem consiste somente em armazenar os exemplos de treinamento!

O conceito base por trás deste método é que os dados tendem a estar concentrados em uma mesma região no espaço de entrada.

Para compensar as diferenças de unidade entre os atributos contínuos, temos que normalizá-los para ficar no intervalo [0,1].

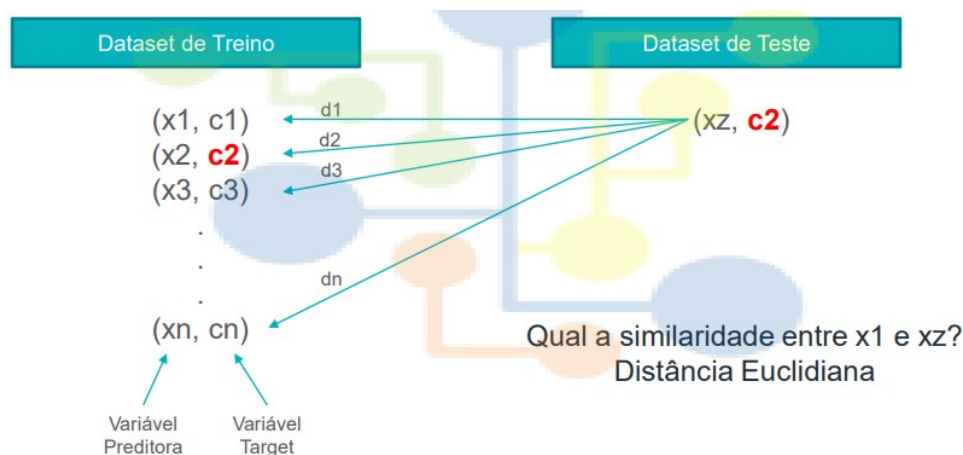
Métodos de aprendizagem baseados em instâncias assumem que as instâncias podem ser representadas como pontos em um espaço Euclidiano.

Outras Medidas de Distância:

Manhattan	Euclidean
	

- Correlação de Pearson – Coeficiente de correlação usado em estatística. Muito usado em bioinformática.
- Similaridade de Cosseno – Cosseno do ângulo entre os vetores. Usado para classificação de textos e outros dados de alta dimensão.
- Distância de edição – Usado para medir distância entre strings. Usado em classificação de textos e bioinformática.

Exemplo de como ocorre a aprendizagem através de método baseado em instância:



Dataset de Treino, Dataset de Teste, Variável Preditora e Variável Target.

Características dos Métodos Baseados em Instância:

- Constroem aproximações da função alvo para cada instância de teste diferente.
- Os métodos de aprendizagem baseados em instâncias são métodos não paramétricos.
- Uma desvantagem é o alto custo para classificação. Toda computação ocorre no momento da classificação.

- Ao contrário das outras abordagens, não ocorre a construção de um modelo de classificação explícito.
- Novos exemplos são classificados com base na comparação direta e similaridade aos exemplos de treinamento.
- Treinamento pode ser fácil, apenas memoriza exemplos.
- Teste pode ser intenso computacionalmente pois requer comparação com todos os exemplos de treinamento.
- Métodos baseados em instância favorecem a similaridade global e não a simplicidade do conceito.

KNN K – Nearest Neighbours (Lazy)

Objetos relacionados ao mesmo conceito são semelhantes entre si.

Regra do KNN: Classificar xz atribuindo a ele o rótulo representado mais frequentemente dentre as k amostras mais próximas e utilizando um esquema de votação.

Aprendizagem Baseada em Métodos Probabilísticos

Os métodos probabilísticos bayesianos assumem que a probabilidade de um evento A, que pode ser uma classe, dado em um evento B, não depende apenas da relação entre A e B, mas também da probabilidade de observar A independentemente de B.

Decisões ótimas podem ser tomadas com base nessas probabilidades conjuntamente com os dados observados e fornece uma solução quantitativa ponderando a evidência e suportando hipóteses alternativas. Fornece a base para algoritmos de aprendizagem que manipulam probabilidades bem como outros algoritmos que não manipulam probabilidades explicitamente.

Os Métodos Probabilísticos são relevantes por dois motivos:

1. Fornecem algoritmos de aprendizagem práticos:

- Aprendizagem Naïve Bayes
- Aprendizagem de Redes Bayesianas
- Combinam conhecimento a priori com os dados observados

2. Fornecem uma estrutura conceitual útil:

Fornecem a “norma de ouro” (regra do menor erro possível) para avaliar outros algoritmos de aprendizagem.

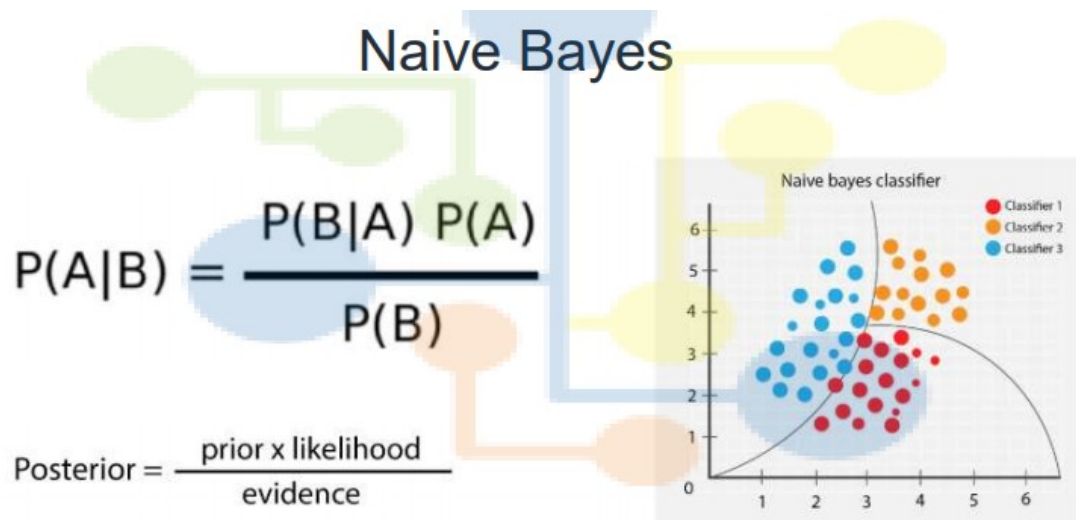
Cada exemplo de treinamento pode decrementar ou incrementar a probabilidade de uma hipótese ser correta.

Conhecimento a priori pode ser combinado com os dados observados para determinar a probabilidade de uma hipótese.

Métodos Bayesianos podem acomodar hipóteses que fazem previsões probabilísticas. (Ex: Este paciente tem uma chance de 93% de se recuperar)

Novas instâncias podem ser classificadas combinando a probabilidade de múltiplas hipóteses ponderadas pelas suas probabilidades.

O classificador Naïve Bayes é baseado na suposição simplificadora de que os valores dos atributos são condicionalmente independentes dado o valor alvo.



Aprendizagem Baseada em Procura

Árvores de Decisão

Árvores de decisão também podem ser representadas como conjuntos de regras SE-ENTÃO-SENÃO (IF-THEN-ELSE).

Árvores de decisão classificam instâncias ordenando as árvores acima (ou abaixo), a partir da raiz até alguma folha.

Entropia

A Física usa o termo entropia para descrever a quantidade de desordem associada a um sistema.

A incerteza ou impureza em um nó pode ser medida através da Entropia. Se todos os exemplos são da mesma classe, então a entropia assume valor mínimo. Se todas as classes têm o mesmo número de exemplos então a entropia assume o valor máximo.

Entropia mede o nível de certeza que temos sobre um evento.

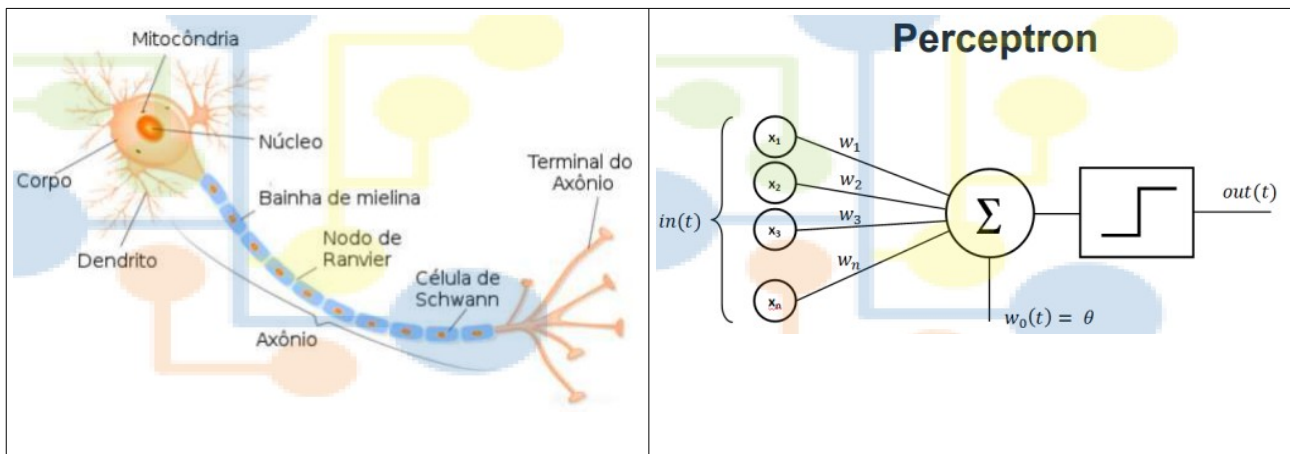
Ganho de Informação mede a efetividade de um atributo em classificar um conjunto de treinamento.

Aprendizagem Baseada em Otimização

Redes Neurais Artificiais

O cérebro humano é a máquina mais incrível do Planeta Terra.

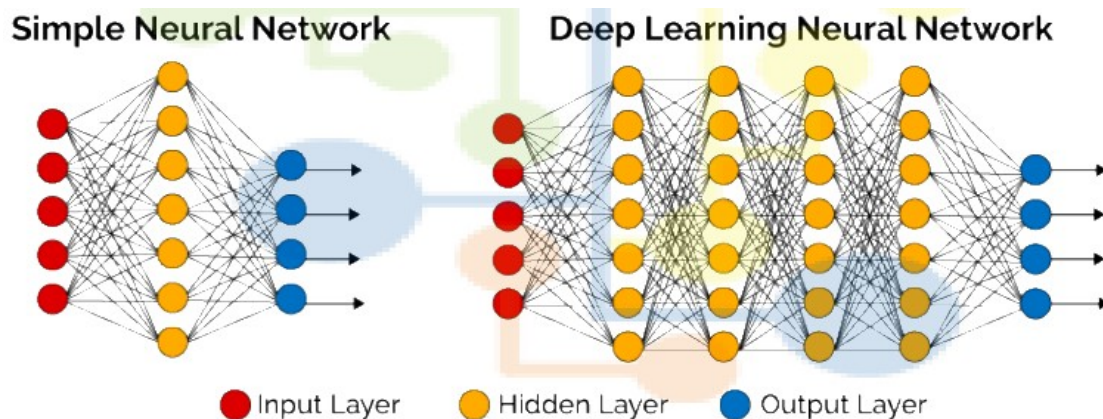
Representação de um Neurônio Biológico	Representação de um Neurônio Matemático
--	---



Uma estrutura de Redes Neurais Artificiais RNA's possui:

Entradas, Pesos, Neurônios Intermediários, Neurônios de Saída e as Saídas

A diferença entre uma Rede Neural Simples e uma Rede Neural Deep Learning é que a última possui mais de uma camada intermediária escondida.



Tipos de Redes Neurais Artificiais (informação a ser validada):

- Deep Learning
- Redes Neurais Convolucionais (CNN – Convolutional Neural Networks)
- Redes Neurais Recorrentes (RNN – Recurrent Neural Networks)
- Rede de Kohonen
- Rede de Hopfield

SVM (Support Vector Machines) ou Máquinas de Vetores de Suporte

As máquinas de vetores de suporte são consideradas uma outra categoria das redes neurais. Elas são compostas por Vetores de Suporte (pontos críticos), Margem máxima e Hiperplano separador ótimo.

Vetores de suporte são simplesmente as coordenadas de observação individual. Support Vector Machine é uma fronteira (hiperplano) que melhor segrega as duas classes.

A SVM é uma técnica de aprendizado estatístico, baseada no princípio da Minimização do Risco Estrutural (SRM) e pode ser usada para resolver problemas de classificação ou de regressão.

Aprendizagem Não Supervisionada Clusterização

Agrupar todos os clientes de sua locadora de automóveis em 10 grupos com base em seus hábitos de compra e usar uma estratégia separada para os clientes em cada um desses 10 grupos. Isso é Clusterização.

Agrupamento (Clustering) é a tarefa de dividir a população ou pontos de dados em um número de grupos de tal forma que os pontos de dados nos mesmos grupos são mais semelhantes a outros pontos de dados no mesmo grupo do que aqueles em outros grupos. Em palavras simples, o objetivo é segregar grupos com traços semelhantes e atribuí-los em clusters.

Medidas de Distância:

- Distância euclidiana: considera a distância entre dois elementos X_i e X_j no espaço p -dimensional;
- Distância “city-block”: corresponde à soma das diferenças entre todos os p atributos de dois elementos X_i e X_j , não sendo indicada para os casos em que existe uma correlação entre tais atributos.

Normalização: variáveis com mesmo peso

- Min-Max para um atributo f
- Z-score
- Desvio absoluto médio

Tipos de Clustering:

- Hard Clustering
- Soft Clustering

Tipos de Algoritmos de Clustering:

- Modelos de Conectividade
- Modelos Centróide
- Modelos de Distribuição
- Modelos de Densidade

Métodos Utilizados para Clusterização:

As heurísticas existentes para a solução de problemas de clusterização podem ser classificadas, de forma geral, em métodos **hierárquicos** e métodos de **particionamento**.

Aprendizagem com Métodos Ensemble

Isso é Método Ensemble: Unimos as saídas de diferentes modelos para encontrar a melhor resposta para o problema.

Passo a passo dos métodos ensemble (desde o dataset original):

Passo 1 – Criação de Múltiplos Datasets (amostras)

Passo 2 – Criação de Múltiplos Modelos

Passo 3 – Combinação dos Múltiplos Modelos

Temos duas abordagens principais com Métodos Ensemble:

- Bootstrap Aggregation ou Bagging
 - Bagged CART
 - Random Forest
- Boosting
 - C5.0
 - Stochastic Gradient Boosting
 - AdaBoost

Métodos Ensemble – Combinação de Preditores

Métodos Ensemble permitem aumentar consideravelmente o nível de precisão nas previsões do modelo preditivo.

Quando falamos de Métodos Ensemble, estamos falando na verdade na combinação de preditores, combinação de modelos. A ideia principal dos algoritmos que utilizam método ensemble é a de utilizar a saída final de diferentes preditores para agregá-los em um único preditor. Basicamente, ensemble é uma técnica de aprendizagem supervisionada para combinar vários modelos fracos a fim de produzir um modelo forte. Um Modelo Ensemble funciona melhor, quando agrupamos modelos com baixa correlação. Um bom exemplo de como os métodos ensemble são comumente usados para resolver problemas de ciência de dados é o algoritmo Random Forest, bastante usando em seleção de variáveis.

Os dois algoritmos mais populares de combinação de preditores são Bagging e Boosting, mas existe ainda o Voting ensemble.

Usamos **Bootstrap Aggregating (Bagging)** para construção de múltiplos modelos (normalmente do mesmo tipo) a partir de diferentes subsets no dataset de treino.

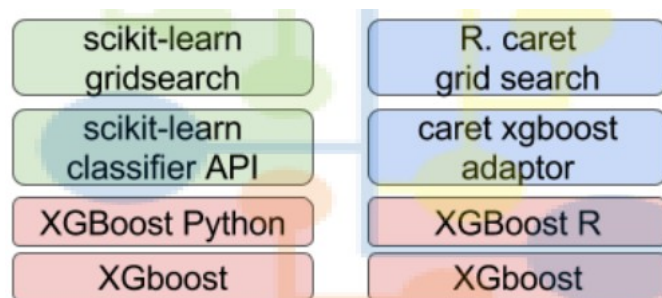
Bootstraps (amostras diferentes da base de dados que são usadas para aprender hipóteses diferentes)

Usamos **Boosting** para construção de múltiplos modelos (normalmente do mesmo tipo), onde cada modelo aprende a corrigir os erros gerados pelo modelo anterior, dentro da sequência de modelos criados.

O termo "Boosting" refere-se a uma família de algoritmos que converte modelos fracos em um modelo forte

- AdaBoost (Adaptive Boosting)
 - AdaBoost.M1
 - AdaBoost.M2
 - AdaBoost.R
 - Adaboost.R2
 - AdaBoost.RT
 - Boosting Correlation Improvement (BCI)

- Entrada: $(x_1, y_1), \dots, (x_m, y_m)$ x = vetor de características $y = \{-1, +1\}$
- Boosting para Problemas de Regressão – AdaBoost.R
 - $f(x)$
 $g(x)$
 $f(x_i) = g(x_i), \forall x_i \in X$
- Gradient Boosting
 - Gradient Boosting = Gradient Descent + Boosting
 GBM = Gradient Boosting Method
 - Função de Perda: $y = ax + b + e$
- XGBoost – eXtreme Gradient Boosting



Já o método **Voting** é usado na construção de múltiplos modelos (normalmente de tipos diferentes) e estatísticas simples (como a média) são usadas para combinar as previsões. O Voting Ensemble é um dos métodos Ensemble mais simples. Este método cria dois ou mais modelos separados a partir do dataset de treino. O Classificador Voting então utiliza a média das previsões de cada submodelo para fazer as previsões em novos conjuntos de dados. As previsões de cada submodelo podem receber pesos, através de parâmetros definidos manualmente ou através de heurística. Existem versões mais avançadas do Voting, em que o modelo pode aprender o melhor peso a ser atribuído aos submodelos. Isso é chamado Stacked Aggregation.

Por que utilizar Métodos Ensemble?

- Razões Estatísticas
- Grandes volumes de dados
- Pequenos volumes de dados
- Dividir e Conquistar
- Seleção de modelo
- Diversidade

Redução de Dimensionalidade

E esse aumento no volume de dados não é apenas em volume e de forma vertical, mas ocorre também na horizontal, com o aumento do número de dimensões ou atributos das bases de dados.

O termo dimensionalidade é atribuído ao número de características de uma representação de padrões, ou seja, a dimensão do espaço de características.

- Extração de Atributos: Principal Component Analysis, Multidimensional Scaling e o FastMap.
- Seleção de Atributos: Algoritmos de aprendizado de máquina (Random Forest), cálculo de dimensão fractal e wrapper

7 Técnicas para Redução da Dimensionalidade:

1. Missing Values Ratio
2. Low Variance Filter
3. High Correlation Filter
4. Random Forests / Ensemble Trees
5. Forward Feature Construction
6. Backward Feature Elimination
7. Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

Cada componente resultante é uma combinação linear de n atributos. Cada componente principal é uma combinação de atributos presentes no dataset.

O PCA precisa ser alimentado com dados normalizados. Utilizar o PCA em dados não normalizados pode gerar resultados inesperados.

Esta técnica pode ser utilizada para geração de índices e agrupamento de indivíduos.

A análise de componentes principais é associada à ideia de redução de massa de dados, com menor perda possível da informação.

Em termos gerais a PCA busca reduzir o número de dimensões de um dataset, projetando os dados em um novo plano.

Modelos Lógicos, Geométricos e Probabilísticos

Outra importante Classificação de Modelos Preditivos:

- Lógicos
- Geométricos
- Probabilísticos

Os modelos são a parte central da implementação de qualquer projeto de Machine Learning. Um modelo descreve os dados observados em um sistema e faz previsões. Os modelos são o resultado da aplicação de algoritmos a conjuntos de dados. Os modelos são então aplicados a novos conjuntos de dados e as previsões são feitas. Existe uma grande quantidade de algoritmos e variações e além das categorias de modelos que vimos ao longo deste capítulo, existe uma outra divisão: Modelos Lógicos, Geométricos e Probabilísticos.

Modelos lógicos são modelos mais algorítmicos por natureza e nos ajudam a derivar um conjunto de regras a partir da execução iterativa dos algoritmos. As árvores de decisão são um exemplo de algoritmo que geram modelos lógicos.

Os modelos geométricos usam conceitos de geometria como linhas, planos e distâncias. Esses modelos normalmente operam ou podem operar com grandes conjuntos de dados. Normalmente utilizamos transformação linear para comparar diferentes métodos criados a partir desses modelos. Um bom exemplo é o SVM.

E temos ainda os modelos probabilísticos que são modelos que empregam técnicas estatísticas. Esses modelos são baseados na estratégia que define o relacionamento entre duas variáveis. Este relacionamento pode ser derivado com alto grau de certeza, uma vez que envolve processos sendo executados em background. Em muitos casos um subset de dados (como uma amostra) pode ser considerado para processamento. Um exemplo desse modelo é o Naive Bayes.

Como você deve ter percebido ao longo deste módulo, existe uma grande quantidade de modelos preditivos que empregam uma grande variedade de técnicas diferentes. A melhor forma de aprender Machine Learning é a experimentação. Teste o maior número possível de algoritmos, faça experimentos, aplique o mesmo algoritmo a diferentes tipos dados, crie variações dos algoritmos, combine diferentes técnicas. Essa é a melhor forma de aprender a trabalhar com Machine Learning.

Deep Learning – Contexto

O interesse pela Aprendizagem de Máquina (Machine Learning) explodiu na última década. O mundo a nossa volta está passando por uma transformação e vemos uma interação cada vez maior das aplicações de computador com os seres humanos. Softwares de detecção de spam, sistemas de recomendação, marcação em fotos de redes sociais, assistentes pessoais ativados por voz, carros autônomos, smartphones com reconhecimento facial e muito mais.

E o interesse por Machine Learning se mostra ainda mais evidente pelo número cada vez maior de conferências, meetups, artigos, livros, cursos, buscas no Google e profissionais e empresas procurando compreender o que é e como usar aprendizagem de máquina, embora muitos ainda confundem o que podem fazer com o que desejam fazer. Não há como ficar indiferente a esta revolução trazida pela aprendizagem de máquina e, segundo o Gartner, até 2020 todos os softwares corporativos terão alguma funcionalidade ligada a Machine Learning.

Fundamentalmente, Machine Learning é a utilização de algoritmos para extrair informações de dados brutos e representá-los através de algum tipo de modelo matemático. Usamos então este modelo para fazer inferências a partir de outros conjuntos de dados. Existem muitos algoritmos que permitem fazer isso, mas um tipo em especial vem se destacando, as redes neurais artificiais.

As redes neurais artificiais não são necessariamente novas, existem pelo menos desde a década de 1950. Mas durante várias décadas, embora a arquitetura desses modelos tivesse evoluído, ainda faltavam ingredientes que fizessem os modelos realmente funcionar. E esses ingredientes surgiram quase ao mesmo tempo. Um deles você já deve ter ouvido: Big Data. O volume de dados, gerado em variedade e velocidade cada vez maiores, permite criar modelos e atingir altos níveis de precisão. Mas ainda falta um ingrediente. Faltava! Como processar grandes modelos de Machine Learning com grandes quantidades de dados? As CPUs não conseguiam dar conta do recado.

Foi quando os gamers e sua afeição por poder computacional e gráficos perfeitos, nos ajudaram a encontrar o segundo ingrediente: Programação Paralela em GPUs. As unidades de processamento gráfico, que permitem realizar operações matemáticas de forma paralela, principalmente operações com matrizes e vetores, elementos presentes em modelos de redes neurais artificiais, formaram a tempestade perfeita, que permitiu a evolução na qual nos encontramos hoje: Big Data + Processamento Paralelo + Modelos de Aprendizagem de Máquina = Inteligência Artificial.

A unidade fundamental de uma rede neural artificial é um nó (ou neurônio matemático), que por sua vez é baseado no neurônio biológico. As conexões entre esses neurônios matemáticos também foram inspiradas em cérebros biológicos, especialmente na forma como essas conexões se desenvolvem ao longo do tempo com “treinamento”. Em meados da década de 1980 e início da década de 1990, muitos avanços importantes na arquitetura das redes neurais artificiais ocorreram. No entanto, a quantidade de tempo e dados necessários para obter bons resultados retardou a adoção e, portanto, o interesse foi arrefecido, com o que ficou conhecido como AI Winter (Inverno da IA).

No início dos anos 2000, o poder computacional expandiu exponencialmente e o mercado viu uma “explosão” de técnicas computacionais que não eram possíveis antes disso. Foi quando o aprendizado profundo (Deep Learning) emergiu do crescimento computacional explosivo dessa década como o principal mecanismo de construção de sistemas de Inteligência Artificial, ganhando muitas competições importantes de aprendizagem de máquina. O interesse por Deep Learning não para de crescer e hoje vemos o termo aprendizado profundo sendo mencionado com frequência cada vez maior e soluções comerciais surgindo a todo momento.

Aprendizagem Profunda ou Deep Learning, é uma subárea da Aprendizagem de Máquina, que emprega algoritmos para processar dados e imitar o processamento feito pelo cérebro humano. Não se preocupe se alguns termos mais técnicos não fizerem sentido agora. Todos eles serão estudados no capítulo sobre Deep Learning.

Deep Learning usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída. Todas as camadas entre as duas são referidas como camadas ocultas. Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação.

A aprendizagem profunda é responsável por avanços recentes em visão computacional, reconhecimento de fala, processamento de linguagem natural e reconhecimento de áudio. O aprendizado profundo é baseado no conceito de redes neurais artificiais, ou sistemas computacionais que imitam a maneira como o cérebro humano funciona.

A extração de recursos é outro aspecto da Aprendizagem Profunda. A extração de recursos usa um algoritmo para construir automaticamente “recursos” significativos dos dados para fins de treinamento, aprendizado e compreensão. Normalmente, o Cientista de Dados, ou Engenheiro de IA, é responsável pela extração de recursos.

O aumento rápido e o aparente domínio do aprendizado profundo sobre os métodos tradicionais de aprendizagem de máquina em uma variedade de tarefas tem sido surpreendentes de testemunhar e, às vezes, difícil de explicar. Deep Learning é uma evolução das Redes Neurais, que por sua vez possuem uma história fascinante que remonta à década de 1940, cheia de altos e baixos, voltas e

reviravoltas, amigos e rivais, sucessos e fracassos. Em uma história digna de um filme dos anos 90, uma ideia que já foi uma espécie de patinho feio floresceu para se tornar a bola da vez.

Consequentemente, o interesse em aprendizagem profunda tem disparado, com cobertura constante na mídia popular. A pesquisa de aprendizagem profunda agora aparece rotineiramente em revistas como Science, Nature, Nature Methods e Forbes apenas para citar alguns. O aprendizado profundo conquistou Go, aprendeu a dirigir um carro, diagnosticou câncer de pele e autismo, tornou-se um falsificador de arte mestre e pode até alucinar imagens fotorrealistas.

Os primeiros algoritmos de aprendizagem profunda que possuíam múltiplas camadas de características não-lineares podem ser rastreados até Alexey Grigoryevich Ivakhnenko (desenvolveu o Método do Grupo de Manipulação de Dados) e Valentin Grigor'evich Lapa (autor de Cybernetics and Forecasting Techniques) em 1965, que usaram modelos finos mas profundos com funções de ativação polinomial os quais eles analisaram com métodos estatísticos. Em cada camada, eles selecionavam os melhores recursos através de métodos estatísticos e encaminhavam para a próxima camada. Eles não usaram Backpropagation para treinar a rede de ponta a ponta, mas utilizaram mínimos quadrados camada-por-camada, onde as camadas anteriores foram independentemente instaladas em camadas posteriores (um processo lento e manual).

No final da década de 1970, o primeiro inverno da IA começou, resultado de promessas que não poderiam ser mantidas. O impacto desta falta de financiamento limitou a pesquisa em Redes Neurais Profundas e Inteligência Artificial. Felizmente, houve indivíduos que realizaram a pesquisa sem financiamento.

As primeiras “redes neurais convolutivas” foram usadas por Kunihiko Fukushima. Fukushima concebeu redes neurais com múltiplas camadas de agrupamento e convoluções. Em 1979, ele desenvolveu uma rede neural artificial, chamada Neocognitron, que usava um design hierárquico e multicamadas. Este design permitiu ao computador “aprender” a reconhecer padrões visuais. As redes se assemelhavam a versões modernas, mas foram treinadas com uma estratégia de reforço de ativação recorrente em múltiplas camadas, que ganhou força ao longo do tempo. Além disso, o design de Fukushima permitiu que os recursos importantes fossem ajustados manualmente aumentando o “peso” de certas conexões.

Muitos dos conceitos de Neocognitron continuam a ser utilizados. O uso de conexões de cima para baixo e novos métodos de aprendizagem permitiram a realização de uma variedade de redes neurais. Quando mais de um padrão é apresentado ao mesmo tempo, o Modelo de Atenção Seletiva pode separar e reconhecer padrões individuais deslocando sua atenção de um para o outro (o mesmo processo que usamos em multitarefa). Um Neocognitron moderno não só pode identificar padrões com informações faltantes (por exemplo, um número 5 desenhado de maneira incompleta), mas também pode completar a imagem adicionando as informações que faltam. Isso pode ser descrito como “inferência”.

O Backpropagation, o uso de erros no treinamento de modelos de Deep Learning, evoluiu significativamente em 1970. Foi quando Seppo Linnainmaa escreveu sua tese de mestrado, incluindo um código FORTRAN para Backpropagation. Infelizmente, o conceito não foi aplicado às redes neurais até 1985. Foi quando Rumelhart, Williams e Hinton demonstraram o Backpropagation em uma rede neural que poderia fornecer representações de distribuição “interessantes”. Filosoficamente, essa descoberta trouxe à luz a questão dentro da psicologia cognitiva de saber se a compreensão humana depende da lógica simbólica (computacionalismo) ou de representações

distribuídas (conexão). Em 1989, Yann LeCun forneceu a primeira demonstração prática de Backpropagation no Bell Labs. Ele combinou redes neurais convolutivas com Backpropagation para ler os dígitos “manuscritos” (assunto do próximo capítulo). Este sistema foi usado para ler o número de cheques manuscritos.

Porém, tivemos neste período o que ficou conhecido como segundo Inverno da IA, que ocorreu entre 1985-1990, que também afetou pesquisas em Redes Neurais e Aprendizagem Profunda. Vários indivíduos excessivamente otimistas haviam exagerado o potencial “imediato” da Inteligência Artificial, quebrando as expectativas e irritando os investidores. A raiva era tão intensa, que a frase Inteligência Artificial atingiu o status de pseudociência. Felizmente, algumas pessoas continuaram trabalhando em IA e Deep Learning, e alguns avanços significativos foram feitos. Em 1995, Dana Cortes e Vladimir Vapnik desenvolveram a máquina de vetor de suporte ou Support Vector Machine (um sistema para mapear e reconhecer dados semelhantes). O LSTM (Long-Short Term Memory) para redes neurais recorrentes foi desenvolvido em 1997, por Sepp Hochreiter e Juergen Schmidhuber.

O próximo passo evolutivo significativo para Deep Learning ocorreu em 1999, quando os computadores começaram a se tornar mais rápidos no processamento de dados e GPUs (unidades de processamento de gráfico) foram desenvolvidas. O uso de GPUs significou um salto no tempo de processamento, resultando em um aumento das velocidades computacionais em 1000 vezes ao longo de um período de 10 anos. Durante esse período, as redes neurais começaram a competir com máquinas de vetor de suporte. Enquanto uma rede neural poderia ser lenta em comparação com uma máquina de vetor de suporte, as redes neurais ofereciam melhores resultados usando os mesmos dados. As redes neurais também têm a vantagem de continuar a melhorar à medida que mais dados de treinamento são adicionados.

Em torno do ano 2000, apareceu o problema conhecido como Vanishing Gradient. Foi descoberto que as “características” aprendidas em camadas mais baixas não eram aprendidas pelas camadas superiores, pois nenhum sinal de aprendizado alcançou essas camadas. Este não era um problema fundamental para todas as redes neurais, apenas aquelas com métodos de aprendizagem baseados em gradientes. A origem do problema acabou por ser certas funções de ativação. Uma série de funções de ativação condensavam sua entrada, reduzindo, por sua vez, a faixa de saída de forma um tanto caótica. Isso produziu grandes áreas de entrada mapeadas em uma faixa extremamente pequena. Nessas áreas de entrada, uma grande mudança será reduzida a uma pequena mudança na saída, resultando em um gradiente em queda. Duas soluções utilizadas para resolver este problema foram o pré-treino camada-acamada e o desenvolvimento de uma memória longa e de curto prazo.

Em 2001, um relatório de pesquisa do Grupo META (agora chamado Gartner) descreveu os desafios e oportunidades no crescimento do volume de dados. O relatório descreveu o aumento do volume de dados e a crescente velocidade de dados como o aumento da gama de fontes e tipos de dados. Este foi um apelo para se preparar para a investida do Big Data, que estava apenas começando.

Em 2009, Fei-Fei Li, professora de IA em Stanford na Califórnia, lançou o ImageNet e montou uma base de dados gratuita de mais de 14 milhões de imagens etiquetadas. Eram necessárias imagens marcadas para “treinar” as redes neurais. A professora Li disse: “Nossa visão é que o Big Data mudará a maneira como a aprendizagem de máquina funciona. Data drives learning.”. Ela acertou em cheio!

Até 2011, a velocidade das GPUs aumentou significativamente, possibilitando a formação de redes neurais convolutivas “sem” o pré-treino camada por camada. Com o aumento da velocidade de computação, tornou-se óbvio que Deep Learning tinha vantagens significativas em termos de eficiência e velocidade. Um exemplo é a AlexNet, uma rede neural convolutiva, cuja arquitetura ganhou várias competições internacionais durante 2011 e 2012. As unidades lineares retificadas foram usadas para melhorar a velocidade.

Também em 2012, o Google Brain lançou os resultados de um projeto incomum conhecido como The Cat Experiment. O projeto de espírito livre explorou as dificuldades de “aprendizagem sem supervisão”. A Aprendizagem profunda usa “aprendizagem supervisionada”, o que significa que a rede neural convolutiva é treinada usando dados rotulados. Usando a aprendizagem sem supervisão, uma rede neural convolucional é alimentada com dados não marcados, e é então solicitada a busca de padrões recorrentes.

O Cat Experiment usou uma rede neural distribuída por mais de 1.000 computadores. Dez milhões de imagens “sem etiqueta” foram tiradas aleatoriamente do YouTube, mostradas ao sistema e, em seguida, o software de treinamento foi autorizado a ser executado. No final do treinamento, um neurônio na camada mais alta foi encontrado para responder fortemente às imagens de gatos. Andrew Ng, o fundador do projeto, disse: “Nós também encontramos um neurônio que respondeu fortemente aos rostos humanos”. A aprendizagem não supervisionada continua a ser um campo ativo de pesquisa em Aprendizagem Profunda.

Atualmente, o processamento de Big Data e a evolução da Inteligência Artificial são ambos dependentes da Aprendizagem Profunda. Com Deep Learning podemos construir sistemas inteligentes e estamos nos aproximando da criação de uma IA totalmente autônoma. Isso vai gerar impacto em todas as partes da sociedade e aqueles que souberem trabalhar com a tecnologia, serão os líderes desse novo mundo que se apresenta diante de nós.

Caso queira se aprofundar no estudo de Deep Learning, acompanhe um projeto fascinante mantido pela DSA gratuitamente, o Deep Learning Book: <http://deeplearningbook.com.br/>

Deep Learning

- CNN Convolutional Neural Networks

O nome “Rede Neural Convolucional” indica que a rede aplica uma operação matemática denominada convolução. Convolução é um tipo especializado de operação linear. Redes convolucionais são redes neurais simples que utilizam convolução no lugar de multiplicação geral de matrizes em pelo menos uma das camadas.

- RNN Recurrent Neural Networks

Deep Learning é a técnica de aprendizado baseada em Multi-Camadas (Redes Neurais Densas).

Exemplos de utilização de Deep Learning são: o Reconhecimento de Voz e Processamento de Linguagem Natural.

Muitos modelos probabilísticos são difíceis de treinar devido à dificuldade de realizar inferência.

Deep Learning está revolucionando a indústria aeroespacial.

Simulação e Otimização

Podemos entender a simulação como um processo amplo que engloba não apenas a construção do modelo, mas todo o método experimental que se segue.

Simulação implica na modelagem de um processo ou sistema, de tal forma que o modelo imite as respostas do sistema real em uma sucessão de eventos que ocorrem ao longo do tempo.

- Um dispositivo para compreensão de um problema;
- Um meio de comunicação para descrever a operação de um sistema;
- Uma ferramenta de análise para determinar elementos críticos e estimar medidas de desempenho;
- Uma ferramenta de projeto para avaliar problemas e propor soluções;
- Um sistema de planejamento de operações para trabalhos, tarefas e recursos;
- Um mecanismo de controle;
- Uma ferramenta de treinamento;
- Uma parte do sistema para fornecer informações online, projeções de situações e suporte à decisão.

Alguns conceitos importantes:

- Variáveis
- Variáveis de Estado
- Entidade
- Atributo
- Recurso
- Processos
- Tempo de Simulação
- Filas
- Eventos

A simulação é a técnica empregada durante a fase de construção do modelo preditivo.

Pode ser necessário a simulação de uma grande variedade de alternativas e a criação de modelos preditivos pode gerar bons resultados.

Machine Learning nos ajuda a simular um determinado evento através de dados e com isso, prever o comportamento futuro.

Modelos Determinísticos x Modelos Estocásticos

Modelo Computacional é um programa de computador cujas variáveis apresentam o mesmo comportamento dinâmico e estocástico do sistema real que representa.

- Determinístico: Resultado do modelo é pré-determinado em função dos dados de entrada
- Estocástico: Resultado do modelo não depende somente dos dados de entrada, mas também de outros fatores, normalmente aleatórios. Isso requer um modelo probabilístico.

Quando uma variável de entrada de um sistema é aleatória, a variável de saída também será aleatória, no entanto, o sistema pode ter comportamento determinístico ou ser representado por um modelo determinístico.

Modelos determinísticos e estocásticos podem ser combinados para resolver problemas que requerem muitas alternativas diferentes, tais como:

- Busca na Web e Extração de Informação
- Desenvolvimento de Novos Medicamentos
- Prever o Comportamento do Mercado Financeiro
- Compreender o Comportamento de Clientes
- Criação de Robôs

Otimização

Aprendizado = Representação + Avaliação + Otimização

Ao contrário da simulação onde existe incerteza associada com os dados de entrada, na otimização nós temos não somente acesso aos dados, mas também as informações de dependências e relacionamentos entre os atributos dos dados.

Generalização → Principal Objetivo na Construção do Modelo Preditivo

Passos para Seleção de Modelo:

1. Definir Espaço de Parâmetro
2. Definir Configurações de Validação Cruzada
3. Definir Métrica
4. Treinar, Avaliar e Comparar

Estudo de Caso 1 – Features Engineering com Variáveis Categóricas

Arquivo EstudoCaso1.R:

```
# Estudo de Caso 1 - Features Engineering com Variáveis Categóricas na Prática

# Definindo o diretório de trabalho
# setwd("~/Dropbox/DSA/EstudoCaso1")
# getwd()

# Os modelos de aprendizado de máquina têm dificuldade em interpretar dados categóricos;
# Mas a engenharia de recursos nos permite re-contextualizar nossos dados categóricos para
# melhorar o rigor de nossos modelos de Machine Learning. A engenharia de recursos também
# fornece camadas adicionais de perspectiva para a análise de dados. A grande questão que as
# abordagens de engenharia de recursos resolvem é: como utilizar meus dados de maneiras
# interessantes e inteligentes para torná-los muito mais úteis?

# A engenharia de recursos não trata de limpar dados, remover valores nulos ou outras tarefas
# semelhantes (isso é Data Wrangling); a engenharia de recursos tem a ver com a alteração de
# variáveis para melhorar a história que elas contam.

# Vejamos alguns exemplos de tarefas de engenharia de atributos!

# Para este estudo de caso usaremos este dataset com dados bancários de usuários:

# Dataset: http://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank.zip

# Carregando os dados
dataset_bank <- read.table("bank/bank-full.csv", header = TRUE, sep = ";")
View(dataset_bank)
```


Exemplo 1 - Criação de Nova Coluna

Muitas vezes, quando você usa dados categóricos como preditores, pode achar que alguns dos
níveis dessa variável têm uma ocorrência muito escassa ou que os níveis das variáveis são
seriamente redundantes.

Qualquer decisão que você tome para começar a agrupar os níveis de variáveis deve ser
estrategicamente orientada. Um bom começo aqui para ambas as abordagens é a função table() em R.
table(dataset_bank\$job)

A ideia seria identificar a ocorrência de um nível com poucos registros ou alternativamente
compartimentos que parecem mais indicativos do que os dados estão tentando informar.

Às vezes, uma tabela é um pouco mais difícil de ingerir; portanto, jogar isso em um gráfico
de barras pode ser mais fácil.

```
library(dplyr)
library(ggplot2)
```

```
dataset_bank %>%
  group_by(job)%>%
  summarise(n = n())%>%
  ggplot(aes(x = job, y = n))+
  geom_bar(stat = "identity")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

Para esse estudo de caso, digamos que realmente queremos entender a profissão (job) de acordo
com o uso da tecnologia em uma determinada função. Nesse caso, começaríamos a classificar
cada uma das profissões em nível médio, alto e baixo em termos de uso de tecnologia.

Uma função que você pode usar é a mutate do dplyr é muito útil quando você está reatribuindo
muitos níveis diferentes de uma variável, em vez de usar alguma função ifelse aninhada.
Essa função também é muito útil ao converter variáveis numéricas em dados categóricos.

```
dataset_bank <- dataset_bank %>%
  mutate(technology_use =
    case_when(job == 'admin' ~ "medio",
              job == 'blue-collar' ~ "baixo",
              job == 'entrepreneur' ~ "alto",
              job == 'housemaid' ~ "baixo",
              job == 'management' ~ "medio",
              job == 'retired' ~ "baixo",
              job == 'self-employed' ~ "baixo",
              job == 'services' ~ "medio",
              job == 'student' ~ "alto",
              job == 'technician' ~ "alto",
              job == 'unemployed' ~ "baixo",
              job == 'unknown' ~ "baixo"))
```

```
View(dataset_bank)
```

Como você pode ver acima, criamos um novo campo chamado technology_use e atribuímos a cada
um valor de acordo com seu uso de tecnologia. Tenho certeza de que você poderia argumentar
tarefas diferentes para cada uma delas, mas para este estudo de caso é suficiente.

Agora vamos revisar rapidamente esse novo campo.
table(dataset_bank\$technology_use)

Vamos colocar isso em percentual
round(prop.table(table(dataset_bank\$technology_use)),2)

A distribuição deve depender do que você está tentando entender. Digamos que a granularidade
do trabalho foi muito maior e tivemos vários jobs relacionados a marketing, analista de marketing,
gerente de marketing digital etc.
Aproveite a tabela e os gráficos de barras para obter uma melhor
classificação de níveis das variáveis.

Exemplo 2 - Variáveis Dummies

A coluna default representa se um usuário entrou ou não no cheque especial.
Em vez de deixar os níveis da variável padrão como "sim" e "não",
codificaremos como uma variável fictícia (dummy).

Uma variável dummy é a representação numérica de uma variável categórica.
Sempre que o valor padrão for sim, codificaremos para 1 e 0 caso contrário.
Para duas variáveis de nível mutuamente exclusivas, isso elimina a necessidade de uma

```

# coluna adicional, pois está implícito na primeira coluna.
dataset_bank <- dataset_bank %>%
  mutate(defaulted = ifelse(default == "yes", 1, 0))

View(dataset_bank)

# Exemplo 3 - One-Hot Encoding

# Falamos sobre a criação de uma única coluna como uma variável dummy, mas devemos falar
# sobre a codificação one-hot.

# Uma codificação One-Hot é efetivamente a mesma coisa que fizemos no item anterior, mas para variáveis
# de muitos níveis em que a coluna possui 0s em todas as linhas, exceto onde o valor corresponde
# à nova coluna, que seria 1.

# Seria algo assim:

# 0000000001 - indica um valor
# 0000000010 - indica outro valor

library(caret)
?dummyVars
dmy <- dummyVars(" ~ .", data = dataset_bank)
bank.dummies <- data.frame(predict(dmy, newdata = dataset_bank))
View(bank.dummies)

# Acima, carregamos o pacote caret, executamos a função dummyVars para todas as variáveis e,
# em seguida, criamos um novo dataframe, dependendo das variáveis codificadas identificadas.
# Vamos dar uma olhada na nova tabela:
View(dataset_bank)
str(bank.dummies)
View(bank.dummies)

# Não incluímos todas as colunas, e você pode ver que isso deixou a coluna idade (age) no formato original.

# Exemplo 4 - Combinando Recursos ou Cruzamento de Recursos

# O cruzamento de recursos é onde você combina diversas variáveis.
# Às vezes, a combinação de variáveis pode produzir um desempenho preditivo que executa o
# que eles poderiam fazer isoladamente.

# Assim, podemos fazer um agrupamento por duas variáveis por exemplo, com a devida contagem:

dataset_bank %>%
  group_by(job, marital) %>%
  summarise(n = n())

# Uma visualização disso geralmente é muito mais fácil de interpretar
dataset_bank %>%
  group_by(job, marital) %>%
  summarise(n = n()) %>%
  ggplot(aes(x = job, y = n, fill = marital))+
  geom_bar(stat = "identity", position = "dodge") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Uma avaliação que geralmente é muito mais fácil de interpretar
dmy <- dummyVars( ~ job:marital, data = dataset_bank)
bank.cross <- predict(dmy, newdata = dataset_bank)
View(bank.cross)

# Lembre-se de que, ao combinar diversas variáveis, você pode ter alguns desses novos valores
# muito esparsos. Revise as saídas e se necessário aplique alguma outra técnica mencionada anteriormente.

# Conclusão

# Existem muitos métodos adicionais que podem ser usados para variáveis numéricas e combinações de
# numérico e categórico; podemos usar o PCA, entre outras coisas, para melhorar o poder preditivo das
# variáveis explicativas.

```

Quizz

Os modelos Generativos estimam uma função de densidade de probabilidade condicionada à uma classe. Um exemplo de teste é classificado na classe que maximiza a probabilidade a posteriori. Recebem este nome, pois as funções de probabilidade condicionada podem ser utilizadas para gerar dados sintéticos. O Naive Bayes é um exemplo clássico desse tipo de modelo.

O problema central de aprendizagem de máquina é induzir funções gerais a partir de exemplos de treinamento específicos e muitos métodos de aprendizagem constroem uma descrição geral e explícita da função alvo a partir de exemplos de treinamento. Os métodos de aprendizagem baseados em Instâncias simplesmente armazenam os exemplos de treinamento. A generalização é feita somente quando uma nova instância tiver que ser classificada.

O Naïve Bayes é um dos métodos de aprendizagem mais práticos. Normalmente é usado quando há disponibilidade de um conjunto de treinamento grande ou moderado e os atributos que descrevem as instâncias são condicionalmente independentes dada a classificação. É bastante utilizado em Diagnósticos médicos e classificação de documentos de texto.

As árvores de decisão podem ser usadas para problemas de classificação ou regressão. Os modelos em árvores são designados árvores de decisão no caso de problemas de classificação e árvores de regressão nos problemas de regressão. Seja em árvores de decisão, seja em árvores de regressão, a interpretação dos modelos assim como os algoritmos de indução das árvores são muito semelhantes e por isso usa-se mais amplamente o termo árvore de decisão.

A entropia da informação, no caso do aprendizado de máquina, visto que ela pode e é usada em outras áreas, mede a impureza de um determinado conjunto de dados. Em outras palavras mede a dificuldade que se tem para saber qual a classificação de cada registro dentro do conjunto de dados.

7.3. Como Funciona a Aprendizagem de Máquina

Processo de Aprendizagem – Generalização

Um componente chave do processo de aprendizagem é a generalização.

Se um algoritmo de Machine Learning não for capaz de generalizar uma função matemática que faça previsões sobre novos conjuntos de dados, ele não está aprendendo nada e sim memorizando os dados, o que é bem diferente.

E para poder generalizar a função que melhor resolve o problema, os algoritmos de Machine Learning se baseiam em 3 componentes:

- Representação
- Avaliação
- Otimização

Os algoritmos de aprendizagem possuem diversos parâmetros internos.

Nenhum algoritmo único ou uma combinação de algoritmos é 100% preciso o tempo todo.

Big Data é uma grande mistura de dados. Um bom algoritmo de Machine Learning deve ser capaz de distinguir os sinais e mapear as funções alvo de forma eficiente.

Cost Function → Nível de erro

Underfitting x Overfitting

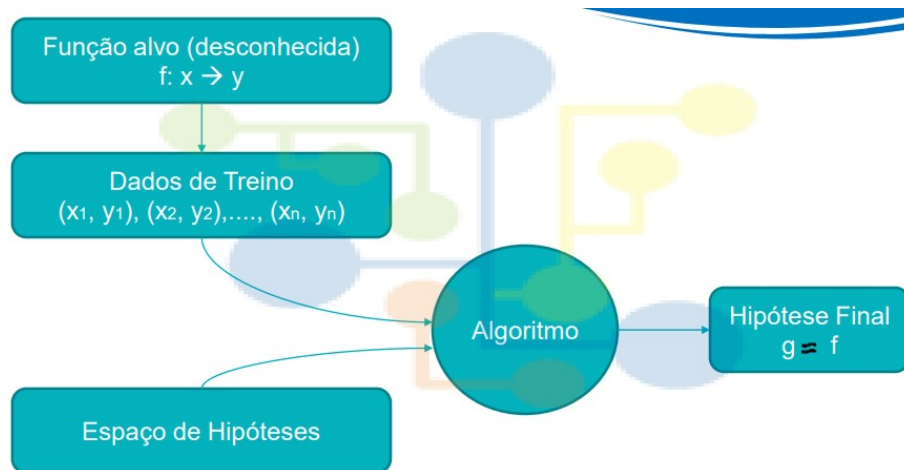
Elementos do Processo de Aprendizagem

- Um padrão existe
- Não há um único modelo matemático que explique esse padrão
- Dados estão disponíveis

Componentes do Processo de Aprendizagem

Componentes do Processo de Aprendizagem:

- Input – x {Dados do cliente}
- Output – y {Decisão → Crédito: Sim/Não}
- Função alvo – $f: x \rightarrow y$ {Representação do relacionamento} {Fórmula matemática desconhecida}
- Dados – $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ {Dados históricos}
- Hipótese – $g: x \rightarrow y$ {Faz parte do espaço de hipóteses do algoritmo}



- Espaço de Hipóteses
 $H = \{h\} \quad g \in H$
- Algoritmo de Aprendizagem

Modelo de Aprendizagem

Espaço de Hipóteses	+	Algoritmo de Aprendizagem	=	Modelo de Aprendizagem
Redes Neurais Support Vector Machines		Back Propagation Programação Quadrática		

O Espaço de Hipóteses contém os recursos com os quais podemos trabalhar. O Algoritmo de Aprendizagem recebe os dados e navega pelo Espaço de Hipóteses a fim de encontrar a melhor hipótese que gera o resultado desejado.

Qual o Critério Usado Para Definir a Previsão do Modelo?

Input $\rightarrow X = (x_1, x_2, \dots, x_d)$

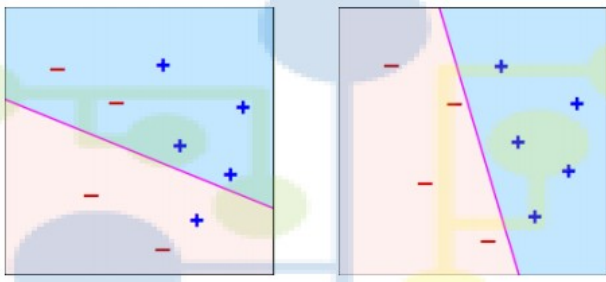
Weight (Peso):

$$\sum_{i=1}^d w_i x_i$$

Fórmula que Define as Hipóteses no Espaço de Hipóteses:

$$h(\mathbf{x}) = \text{sign} \left(\left(\sum_{i=1}^d w_i x_i \right) - \text{threshold} \right)$$

As diferentes combinações weight/threshold vão formar diferentes hipóteses.



As duas linhas em rosa nas imagens acima representam os modelos. Os dois modelos classificam os dados, mas um faz isso melhor do que o outro. Nosso trabalho, como Cientistas de Dados, é encontrar o melhor modelo possível.

O Que Acontece Quando o Modelo Comete Um Erro de Classificação?

Algoritmo de Aprendizagem	$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$
Dados de Treino	$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
Erro de Classificação	$\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n$
Ajuste	$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$

Se os dados forem linearmente separáveis, o algoritmo fará diversas iterações até encontrar a linha que realmente separa as duas classes.

Cost Function (Função de Custo)

Aprendizagem Supervisionada:

Coleção de vetores de atributos (Entrada) $\{x_i\}, i = 1, n$

Coleção de respostas observadas (Saída) $\{y_i\}, i = 1, n$

Durante o treinamento, construímos uma área de respostas (espaço de hipóteses) $h(x)$

Como sabemos se os valores de $h(x)$ são bons ou ruins?

Cost Function: Descreve quão bem resposta na área de respostas (espaço de hipóteses) se encaixa no conjunto de dados que está sendo analisado.

$$J(y_i, h(x_i)) \rightarrow \text{Valores Observados e Valores Previstos}$$

A Cost Function é um número que melhor representa a relação entre valores observados e valores previstos. Em outras palavras: é a diferença entre o que deveria ser previsto pelo modelo e aquilo que ele realmente previu!

O objetivo do algoritmo de ML é aprender um modelo que minimize os erros.

Valores menores da Cost Function significam um melhor "fit"

Um dos objetivos em Machine Learning é construir $h(x)$ de modo que o valor de J seja minimizado.

Em problemas de regressão, $h(x)$ é normalmente interpretada diretamente como a resposta a ser prevista.

Comparando uma previsão contra o seu valor real, usando uma cost function, determinamos o nível de erro do modelo.

Por ser uma formulação matemática, a cost function expressa o nível de erro em uma forma numérica. A cost function transmite o que é realmente importante e significativo para seus propósitos com o algoritmo de aprendizagem.

Gradiente Descendente / Máximo e Mínimo Local

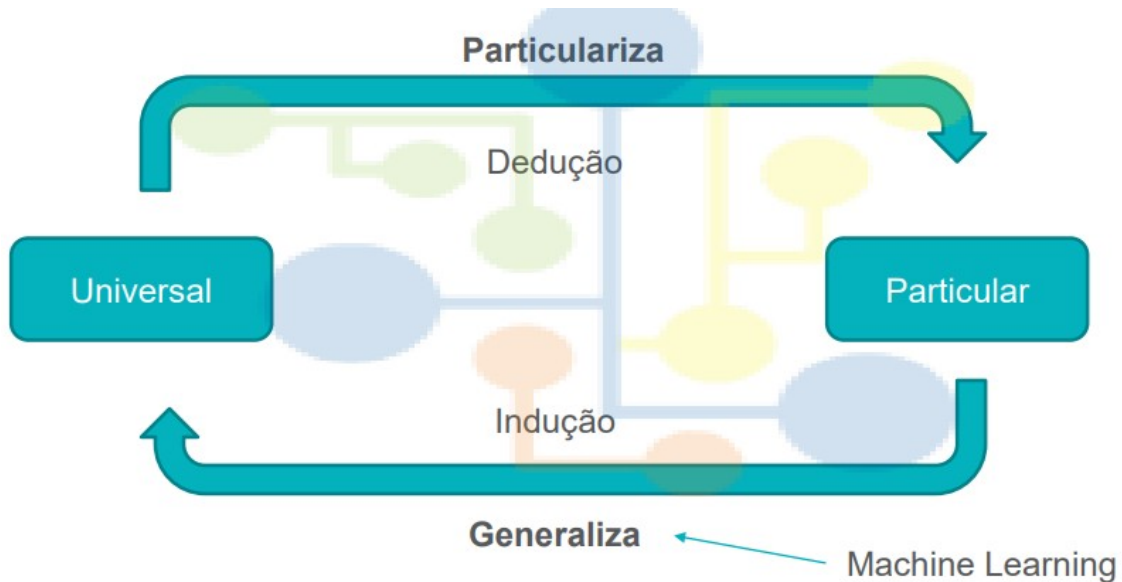
Gradiente Descendente é ideal quando os parâmetros não podem ser calculados analiticamente (por exemplo, usando álgebra linear) e devem ser pesquisados por um algoritmo de otimização.

```
# <font color="blue">Data Science Academy</font>
# <font color="blue">Matemática Para Machine Learning</font>
## Máximo e Mínimo Local
As funções matemáticas podem ter "colinas e vales": locais onde atingem um valor mínimo ou máximo.
# ->
from IPython.display import Image
Image("images/func1.png")
## Global (ou Absoluto) Máximo e Mínimo
O máximo ou mínimo em toda a função é chamado de máximo ou mínimo "Absoluto" ou "Global".
Existe apenas um máximo global (e um mínimo global), mas pode haver mais de um máximo ou mínimo local.
# ->
from IPython.display import Image
Image("images/func2.png")
# ->
from IPython.display import Image
Image("images/gradiente1.png")
# ->
from IPython.display import Image
Image("images/gradiente2.png")
## Exemplo
Encontre os mínimos locais da função  $y = (x + 5)^2$  começando do ponto  $x = 3$ .
Sabemos a resposta apenas olhando para o gráfico.  $y = (x + 5)^2$  atinge seu valor mínimo quando  $x = -5$  (ou seja, quando  $x = -5$ ,  $y = 0$ ). Portanto,  $x = -5$  é o mínimo local e global da função.
# ->
from IPython.display import Image
Image("images/grafico.png")
### Aplicando Gradiente Descendente
# ->
# O algoritmo inicia com o parâmetro  $x=3$ 
cur_x = 3
# Learning rate
rate = 0.01
# Define quando parar o algoritmo
# Parar o loop quando a diferença entre os valores de x em 2 iterações consecutivas for menor que 0,000001
# ou quando o número de iterações exceder 10 mil
precision = 0.000001
# Inicializa o contador do passo anterior
previous_step_size = 1
# Número máximo de iterações
max_iters = 10000
# Contador de iterações
iters = 0
# Gradiente da função
df = lambda x: 2*(x+5)
# ->
while previous_step_size > precision and iters < max_iters:
    # Armazena o valor corrente de x em prev_x
    prev_x = cur_x
    # Aplica o Gradient descent
    cur_x = cur_x - rate * df(prev_x)
    # Altera o valor de x
    previous_step_size = abs(cur_x - prev_x)
    # Incrementa o número de iterações
    iters = iters + 1
    # Imprime as iterações
```

```
print("Iteration", iters, "\nValor de x igual a ", cur_x)
print("\nO mínimo local da função ocorre em: ", cur_x)
# Fim
```

Generalização

Aprendizagem Supervisionada $\rightarrow Y = f(X)$



Universal \rightarrow (Particulariza / Dedução) \rightarrow Particular \rightarrow (Generaliza / Indução) \rightarrow Universal

Generalização refere-se a quão bem os conceitos aprendidos por um modelo de aprendizado de máquina se aplicam a exemplos específicos não vistos pelo modelo durante o processo de aprendizado.

Overfitting x Underfitting

Overfitting e underfitting são as duas maiores causas de mau desempenho dos algoritmos de aprendizado de máquina.

Overfitting (High variance/Alta variância)	Underfitting (High bias/Alto viés)	Good Balance (Low bias, low variance)
<p>High variance</p> <p>overfitting</p>	<p>High bias</p> <p>underfitting</p>	<p>Low bias, low variance</p> <p>Good balance</p>

Overfitting é o problema mais comum e pode ser resolvido com a regularização. Por exemplo, a poda (pruning) da árvore de decisão para evitar overfitting.

Underfitting é um problema menos comum e que pode ser resolvido (em geral) usando mais dados. Outra ação para resolução desse problema é a mudança do algoritmo.

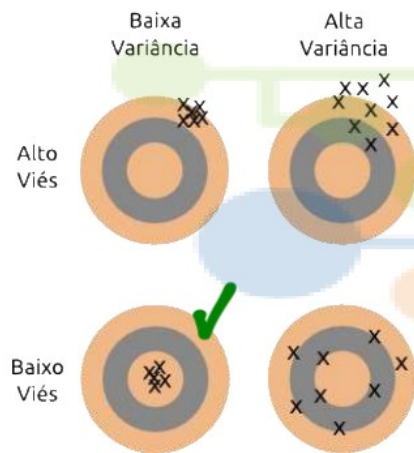
Good Fit é o que se busca. Trata-se de um modelo balanceado, que consegue fazer as previsões, mesmo com uma pequena taxa de erros. Para conseguir o good fit, pode-se usar reamostragem, conjuntos de dados de validação e early-stopping (parar de treinar o modelo num ponto de equilíbrio).

Bias (Viés) e Variância

Separando o erro de generalização em viés (bias) e variância (variance)

Viés é a tendência do modelo aprender consistentemente uma generalização incorreta.

Variância é a tendência de se aprender fatos aleatórios independentemente do sinal real.



Fórmula de Erro de Previsão de um Modelo:

$$E[(y - \hat{f}(x))^2] = Bias[\hat{f}(x)]^2 + Var[\hat{f}(x)] + \sigma^2$$

Fórmula de Bias / Fórmula de Viés:

$$Bias[\hat{f}(x)] = E[\hat{f}(x) - f(x)]$$

Fórmula de Variance / Fórmula de Variância:

$$Var[\hat{f}(x)] = E[\hat{f}(x)^2] - E[\hat{f}(x)]^2$$

Utilizar um modelo complexo que é capaz de reduzir consideravelmente o erro de previsão no dataset de treino, mas ao mesmo tempo não é tão generalizável a ponto de apresentar um bom resultado no dataset de teste, gera baixo viés e alta variância. Ou seja, modelos mais complexos têm viés baixo mas variância alta (overfitting).

Utilizar um modelo simples que é bem generalizável, mas não reduz consideravelmente o erro de previsão no dataset de treino, gera alto viés e baixa variância. Ou seja, modelos mais simples têm viés alto mas variância baixa (underfitting).

Sendo assim, é necessário fazer um tradeoff (achar um equilíbrio).

A tarefa essencial de previsão é selecionar um modelo que se aproxime do ponto mínimo da curva de erro do dataset de teste.

Um Pouco Mais Sobre Bias (Viés) e Variância

Você já deve ter percebido que construir modelos de Machine Learning eficientes não é tarefa simples. Como se não bastassem todas as técnicas de préprocessamento, mais todas as opções de configuração e otimização do algoritmo, temos ainda os erros e problemas gerados durante a fase de aprendizagem e isso sem falar nas questões de armazenamento dos dados. E o viés e variância são mais 2 desses problemas com os quais precisamos lidar. Quando construímos um modelo que generaliza incorretamente particularidades dos dados, temos o problema conhecido como overfitting.

Quando um modelo tem 100% de acerto no dataset de treino e 50% no dataset de teste, enquanto ele deveria ter 75% de acerto em ambos os conjuntos, dizemos que este modelo sofre de overfitting. Uma forma ilustrativa de se compreender tal fenômeno, é separando o erro de generalização em viés (bias) e variância (variance).

Viés é a tendência de o modelo aprender consistentemente uma generalização incorreta (por exemplo, pela simplicidade do modelo). O viés é a distância entre a média do conjunto de estimativas e o único parâmetro a ser estimado.

Variância é a tendência de se aprender fatos aleatórios independentemente do sinal real. A variância é simplesmente o valor esperado dos desvios quadrados de amostragem. Ele é usado para indicar quão distante, em média, o conjunto de estimativas está do valor esperado das estimativas.

Um modelo muito complexo, tem alta variância por ser capaz de aprender padrões que possam não ser reais. Isso faz com que, de forma contra intuitiva, os modelos mais complexos não sejam sempre a melhor alternativa e isso traz o desafio adicional de buscar o modelo que não seja tão complexo, mas, ainda assim, generalizável.

Normalmente, modelos mais simples tem viés alto mas variância baixa (underfitting), enquanto que modelos mais complexos tem viés baixo mas variância alta (overfitting). Mas o que significa um modelo ser mais complexo? Em geral, a complexidade de um modelo aumenta conforme o número de variáveis preditoras aumenta e conforme a capacidade do modelo de captar relações não lineares e interações entre as variáveis preditoras aumenta.

No caso de inúmeras variáveis preditoras, até mesmo um modelo de regressão linear pode se tornar complexo demais, exigindo técnicas de regularização como LASSO ou Regressão Ridge, para diminuir a possibilidade de overfitting.

O Teste de Turing

O Teste de Turing foi criado nos anos 50 por Alan Turing, um dos pais da ciência da computação e da inteligência artificial. O objetivo do teste é descobrir se uma inteligência artificial é inteligente a ponto de enganar um humano, fazendo-o acreditar que se trata de uma pessoa respondendo às suas perguntas, todas feitas e respondidas com texto. Se 30% dos humanos consultados acreditarem que se trata de outro humano, a máquina passa no teste de Turing.

Para contornar o problema da falta de definição precisa para “inteligência artificial”, Alan Turing propôs um teste capaz de determinar se uma máquina demonstra ou não inteligência (artificial), baseado no seguinte argumento:

Não sabemos definir precisamente o que é inteligência e, conseqüentemente, não podemos definir o que é inteligência artificial. Entretanto, embora não tenhamos uma definição de inteligência, podemos assumir que o ser humano é inteligente. Portanto, se uma máquina fosse capaz de se comportar de tal forma que não pudéssemos distingui-la de um ser humano, essa máquina estaria demonstrando algum tipo de inteligência que, nesse caso, só poderia ser inteligência artificial.

O Teste de Turing testa a capacidade de uma máquina exibir comportamento inteligente equivalente a um ser humano, ou indistinguível deste. No teste original, um jogador humano entra em uma conversa, em linguagem natural, com outro humano e uma máquina projetada para produzir respostas indistinguíveis de outro ser humano. Todos os participantes estão separados um dos outros. Se o juiz não for capaz de distinguir com segurança a máquina do humano, diz-se que a máquina passou no teste. O teste não verifica a capacidade de dar respostas corretas para as perguntas, mas sim o quão próximas as respostas são das respostas dados por um ser humano típico. A conversa é restrita a um canal de texto, como um teclado e uma tela para que o resultado não dependa da capacidade da máquina de renderizar áudio.

O teste foi introduzido por Alan Turing em seu artigo de 1950 "Computing Machinery and Intelligence", que começa com as palavras: "Eu proponho considerar a questão "As máquinas podem pensar?". Já que "pensar" é difícil de definir, Turing preferiu "trocar a pergunta por outra, a qual está relacionada à anterior e é expressa em palavras menos ambíguas". A nova pergunta de Turing foi: "Há como imaginar um computador digital que faria bem o 'jogo da imitação?". Turing acreditava que esta questão poderia ser respondida. No restante do artigo, ele argumenta contra as principais objeções a proposta que "máquinas podem pensar" o cientista afirmou ainda que, se um computador fosse capaz de enganar um terço de seus interlocutores, fazendo-os acreditar que ele seria um ser humano, então estaria pensando por si próprio.

O Teste de Turing foi representado fielmente no cinema com o filme "Ex Machina" lançado em 2015 e disponível no Netflix.

Quizz

O trabalho do algoritmo é utilizar uma série de técnicas e tentar descobrir qual função matemática gera a saída esperada. O algoritmo pode descobrir que mais de uma função resolve o problema e neste caso ele aplica um score, uma pontuação a cada função, para que ele tenha condições de selecionar a melhor opção entre todas.

Se um algoritmo de Machine Learning não for capaz de generalizar uma função matemática que faça previsões sobre novos conjuntos de dados, ele não está aprendendo nada e sim memorizando os dados, o que é bem diferente.

Os algoritmos de aprendizagem possuem diversos parâmetros internos (valores separados em vetores e matrizes) que funcionam como referência para o algoritmo, permitindo que o mapeamento ocorra e as características analisadas sejam conectadas. As dimensões e tipo de parâmetros internos delimitam o tipo de funções-alvo que um algoritmo pode aprender. O engine de otimização no algoritmo muda os valores iniciais dos parâmetros durante a aprendizagem para representar a função-alvo.

Quando um modelo tem 100% de acerto no dataset de treino e 50% no dataset de teste, enquanto ele deveria ter 75% de acerto em ambos os conjuntos, dizemos que este modelo sofre de overfitting. Ou seja, o modelo aprendeu demais no treinamento e não consegue fazer previsões em novos dados.

Viés (bias) é a tendência do modelo aprender consistentemente uma generalização incorreta (por exemplo, pela simplicidade do modelo). O viés é a distância entre a média do conjunto de estimativas e o único parâmetro a ser estimado.

7.4. Machine Learning – Regressão

O Que é Regressão?

Uma variável independente x , explica a variação em outra variável, que é chamada variável dependente y . Este relacionamento existe em apenas uma direção:

variável independente (x) \rightarrow variável dependente (y)

Análise de regressão é uma metodologia estatística que utiliza a relação entre duas ou mais variáveis quantitativas de tal forma que uma variável possa ser predita a partir de outra.

Variáveis Dependente e Independente /

O Que Representa a Correlação? /

Correlação Não Implica Causalidade /

Formalizando a Regressão Linear Simples (Um Pouco de Matemática) /

Solução de uma Regressão Linear Simples /

Método dos Mínimos Quadrados /

Minimizando a Soma Geral do erro ao Quadrado /

Erro Padrão da Estimativa /

Coeficiente de Determinação /

Interpretando Um Modelo de Regressão com StatsModels

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
```

```
# <font color='blue'>Capítulo 4 - Regressão</font>
```

```
### O que é Regressão?
```

Vamos começar o módulo compreendendo o que é Regressão.

Em diversos problemas das áreas médica, biológica, industrial, química, finanças, engenharia entre outras, é de grande interesse verificar se duas ou mais variáveis estão relacionadas de alguma forma. Para expressar esta relação é muito importante estabelecer um modelo matemático. Esse tipo de modelagem ajuda a entender como determinadas variáveis influenciam outra variável, ou seja, verifica como o comportamento de uma(s) variável(is) pode mudar o comportamento de outra. Vamos estudar os principais conceitos ligados à regressão.

A regressão linear ajuda a prever o valor de uma variável desconhecida (uma variável contínua) com base em um valor conhecido. Uma aplicação poderia ser: “Qual é preço de uma casa com base em seu tamanho?”

O preço é o valor que queremos prever, com base no tamanho da casa.

Para resolver esse problema, teríamos que buscar dados históricos de tamanho e preço de casa, treinar um modelo, aprender a relação matemática entre os dados e então fazer a previsão de preços com base em outros tamanhos de casa. Dado que estamos analisando o histórico para estimar um novo preço, ele se torna um problema de regressão. O fato de preço e tamanho estarem linearmente relacionados (quanto maior o tamanho da casa) o torna um problema de regressão linear.

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/regression1.png')
```

```
### Variáveis Dependente e Independente
```

Uma variável dependente é o valor que estamos prevendo e uma variável independente é a variável que estamos usando para prever uma variável dependente. Por exemplo, a temperatura é diretamente proporcional ao número de sorvetes comprados. À medida que a temperatura aumenta, o número de sorvetes comprados também aumenta. Aqui a temperatura é a variável independente e, com base nela, o número de sorvetes comprados (a variável dependente) é previsto.

Uma variável independente x , explica a variação em outra variável, que é chamada variável dependente y . Este relacionamento existe em apenas uma direção: variável independente (x) \rightarrow variável dependente (y)

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/regression2.png')
```

```
### O Que Representa a Correlação?
```

Do exemplo anterior, podemos notar que as compras de sorvete estão diretamente correlacionadas (ou seja, elas se movem na mesma direção) com a temperatura.

Neste exemplo, a correlação é positiva: à medida que a temperatura aumenta, as vendas de sorvete aumentam. Em outros casos, a correlação pode ser negativa: por exemplo, as vendas de um item podem aumentar à medida que o preço do item diminui.

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/correlation1.jpg')
```

```
### Correlação Não Implica Causalidade
```

No entanto, intuitivamente, podemos dizer com confiança que a temperatura não é controlada pela venda de sorvetes, embora o inverso seja verdadeiro. Isso traz à tona o conceito de causalidade, qual evento influencia outro evento. A temperatura influencia as vendas de sorvete - mas não vice-versa.

Análise de regressão é uma metodologia estatística que utiliza a relação entre duas ou mais variáveis quantitativas de tal forma que uma variável

possa ser predita a partir de outra, mas isso não implica causalidade. Ou seja, porque existe correlação entre duas variáveis não significa que uma é a causa da outra!

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/correlation2.png')
```

```
### Tipos de Modelos de Regressão
```

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/regression3.png')
```

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/tipos.jpg')
```

```
### Formalizando a Regressão Linear Simples
```

Agora que temos os termos básicos definidos, vamos nos aprofundar nos detalhes da regressão linear.

Uma regressão linear simples é representada pela equação abaixo:

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/formula-regression.png')
```

Onde:

* y é a variável dependente que estamos prevendo.

* x é a variável independente.

* a é o termo de viés.

* b é a inclinação da variável (o peso atribuído à variável independente).

Y e X são as variáveis dependente e independente respectivamente. Vamos focar nos coeficientes (a e b na equação anterior).

Começamos com o coeficiente a, também chamado de viés ou bias. Considere o exemplo:

Queremos estimar o peso de um bebê pela idade do bebê em meses. Assumiremos que o peso de um bebê depende exclusivamente de quantos meses ele tem. O bebê tem 3 kg ao nascer e seu peso aumenta a uma taxa constante de 0,75 kg por mês. No final de um ano (12 meses), o gráfico do peso do bebê seria assim:

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/regression4.png')
```

No gráfico, o peso do bebê começa em 3 (a, o viés) e aumenta linearmente em 0,75 (b, a inclinação) a cada mês. Observe que, um termo de viés é o valor da variável dependente quando todas as variáveis independentes são 0.

A inclinação (ou slope) de uma linha é a diferença entre as coordenadas x e y nos dois extremos da linha e no comprimento da linha. No exemplo anterior, o valor da inclinação (b) é o seguinte:

(Diferença entre as coordenadas y nos dois extremos) / (Diferença entre as coordenadas x nos dois extremos)

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/slope.png')
```

```
### Solução de uma Regressão Linear Simples
```

Vimos um exemplo simples de como a saída de uma regressão linear simples pode parecer (resolvendo viés e inclinação). Vamos agora encontrar uma maneira mais generalizada de gerar uma linha de regressão. O conjunto de dados fornecido é o seguinte:

```
# ->
```

```
import pandas as pd
```

```
df = pd.read_csv('dados/pesos.csv')
```

```
# ->
```

```
print(df)
```

Dado que estamos estimando o peso do bebê com base em sua idade, a regressão linear pode ser construída da seguinte maneira:

$y = a + bx$

Vamos resolver o problema a partir dos primeiros registros. Vamos supor que o conjunto de dados tenha apenas 2 pontos de dados. A modelagem ficaria assim:

Cálculo para o primeiro mês:

* $y = a + b \cdot x$

* $3.44 = a + b \cdot (0)$

* $a = 3.44$

Cálculo para o segundo mês:

* $y = a + b \cdot x$

* $4.39 = a + b \cdot (1)$

* $4.39 = 3.44 + b \cdot (1)$

* $b = 4.39 - 3.44$

* $b = 0.95$

As linhas anteriores já representam o treinamento do modelo e fomos capazes de prever os valores de a e b, nesse caso, $a = 3.44$ e $b = 0.95$. Isso é o que o modelo aprende durante o treinamento!

Se aplicarmos os valores de a e b nos demais pontos de dados restantes acima, conseguimos ter como resultado exatamente o valor de y. No entanto, isso provavelmente não seria o caso na prática, porque a maioria dos dados reais não tem uma relação assim tão perfeita. Por isso precisamos treinar o modelo com mais dados e encontrar uma formação mais genérica.

```
### Método dos Mínimos Quadrados
```

No cenário anterior, vimos que os coeficientes são obtidos usando apenas dois pontos de dados do conjunto total de dados - ou seja, não consideramos a maioria das observações na elaboração de valores ótimos de a e b. Para evitar deixar de fora a maioria dos pontos de dados durante a construção da equação, podemos modificar o objetivo para minimizar o erro quadrático geral (mínimos quadrados comuns) em todos os pontos de dados.

```
# ->
```

```
from IPython.display import Image
```

```
Image('imagens/minimos-quadrados1.png')
```

```
### Minimizando a Soma Geral do Erro ao Quadrado
```

O Método dos Mínimos Quadrados é o método de computação matemática pelo qual se define a reta de regressão. Esse método definirá uma reta que minimizará a soma das distâncias ao quadrado entre os pontos plotados (X, Y) e a reta (que são os valores previstos de Y').

O erro ao quadrado geral é definido como a soma da diferença ao quadrado entre os valores reais e previstos de todas as observações. A razão pela

qual consideramos o valor do erro ao quadrado e não o valor real do erro é que não queremos erro positivo em alguns pontos de dados compensando erros negativos em outros pontos de dados.

Por exemplo, um erro de +5 em três pontos de dados compensa um erro de -5 em três outros pontos de dados, resultando em um erro geral de 0 entre os seis pontos de dados combinados. O erro quadrático converte o erro -5 dos três últimos pontos de dados em um número positivo, para que o erro quadrático geral se torne $6 \times 5^2 = 150$. Isso levanta uma questão: por que devemos minimizar o erro quadrático geral? O princípio é o seguinte:

1. O erro geral é minimizado se cada ponto de dados individual for previsto corretamente.

2. Em geral, a superpredição em 5% é tão ruim quanto a subpredição em 5%, portanto, consideramos o erro ao quadrado.

->

```
from IPython.display import Image
```

```
Image('imagens/minimos-quadrados2.png')
```

Pelo método dos mínimos quadrados calculam-se os parâmetros “a” e “b” da reta que minimiza estas distâncias ou as diferenças (ou o erro) entre Y e Y’. Esta reta é chamada de reta de regressão. Para que a soma dos quadrados dos erros tenha um valor mínimo, devem-se aplicar os conceitos de cálculo diferencial com derivadas parciais. Como as incógnitas do problema são os coeficientes “a” e “b” estrutura-se um sistema de duas equações. Assim aplicando os conceitos acima referidos monta-se o sistema de equações normais que permitirá extrair os valores de a e b.

A reta de regressão que se obtém através do método dos mínimos quadrados é apenas uma aproximação da realidade, ela é um modo útil para indicar a tendência dos dados. Mas até que ponto a reta de regressão obtida é útil para avaliar a realidade? Duas medidas podem indicar o quanto útil ou aproximado da realidade é a reta:

* Erro padrão da estimativa

* Coeficiente de determinação

Erro Padrão da Estimativa

O erro padrão da regressão (S), também conhecido como erro padrão da estimativa, representa a distância média em que os valores observados

“caem” da linha de regressão. Convenientemente, ele mostra o quão errado o modelo de regressão

está, em média, usando as unidades da variável de resposta. Valores menores são melhores porque indica que as observações estão mais próximas da linha ajustada.

S mede a precisão das previsões do modelo. O erro padrão da estimativa é usado junto com o R Squared (Coeficiente de Determinação) na seção de ajuste da maioria dos resultados estatísticos. Ambas as medidas fornecem uma avaliação numérica de quão bem um modelo se ajusta aos dados da amostra. No entanto, existem diferenças entre as duas estatísticas.

O erro padrão da regressão fornece a medida absoluta da distância típica em que os pontos de dados “caem” da linha de regressão. S está nas unidades da variável dependente.

O R Squared fornece a medida relativa da porcentagem da variação da variável dependente explicada pelo modelo. O R Squared pode variar de 0 a 100%.

Coeficiente de Determinação

O coeficiente de determinação (R Squared) deve ser interpretado como a proporção de variação total da variável dependente Y que é explicada pela variação da variável independente X.

O coeficiente de determinação é igual ao quadrado do coeficiente de correlação. Assim a partir do valor do coeficiente de determinação podemos obter o valor do coeficiente de correlação. O coeficiente de determinação é sempre positivo, enquanto que o coeficiente de correlação pode admitir valores entre -1 e +1. Valor igual a 1 indica perfeito relacionamento positivo, enquanto valor igual a -1 indica perfeito relacionamento negativo.

Valores próximos de zero indicam que não há correlação.

O coeficiente de determinação indica o quanto a reta de regressão explica o ajuste da reta, enquanto que o coeficiente de correlação deve ser usado como uma medida de força da relação entre as variáveis.

Para mensurarmos o poder explicativo de um determinado modelo de regressão, ou o percentual de variabilidade da variável Y que é explicado pelo comportamento de variação das variáveis preditoras, precisamos entender alguns importantes conceitos.

Soma Total dos Quadrados (STQ ou SST) – Mostra a variação em Y em torno da própria média.

Soma dos Quadrados de Regressão (SQR) – Oferece a variação de Y considerando as variáveis X utilizadas no modelo.

Soma dos Quadrados dos Resíduos (SQU ou SSE) – Variação de Y que não é explicada pelo modelo elaborado.

STQ = SQR + SQU

->

```
from IPython.display import Image
```

```
Image('imagens/r2.png')
```

R2 é a fração da variância da amostra de Yi explicada (ou prevista) pelas variáveis preditoras. Para um modelo de regressão simples, esta medida mostra quanto do comportamento da variável Y é explicado pelo comportamento de variação da variável X, sempre lembrando que não existe, necessariamente, uma relação de causa e efeito entre as variáveis X e Y. Para um modelo de regressão múltipla, esta medida mostra quanto do comportamento da variável Y é explicado pela variação conjunta das variáveis X consideradas no modelo.

O coeficiente de ajuste R2 não diz aos analistas se uma determinada variável explicativa é estatisticamente significativa e se esta variável é a causa verdadeira da alteração de comportamento da variável dependente.

->

```
import pandas as pd
```

```
import statsmodels.formula.api as smf
```

```
# Carregando o dataset
```

```
df = pd.read_csv('dados/pesos.csv')
```

```
# Criando o Modelo de Regressão
```

```
estimativa = smf.ols(formula = 'Peso ~ Idade', data = df)
```

```
# Treinando o Modelo de Regressão
```

```
modelo = estimativa.fit()
```

```
# Imprimindo o resumo do modelo
```

```
print(modelo.summary())
```

```
# Fim
```

```
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Tipos de Modelos de Regressão Linear

São 2 tipos de Regressão Linear principais:

- Simples
 - 1 Variável Dependente Y
 - 1 Variável Independente X
- Múltiplo
 - 1 Variável Dependente Y
 - 2 ou + Variáveis Independentes X, Xi

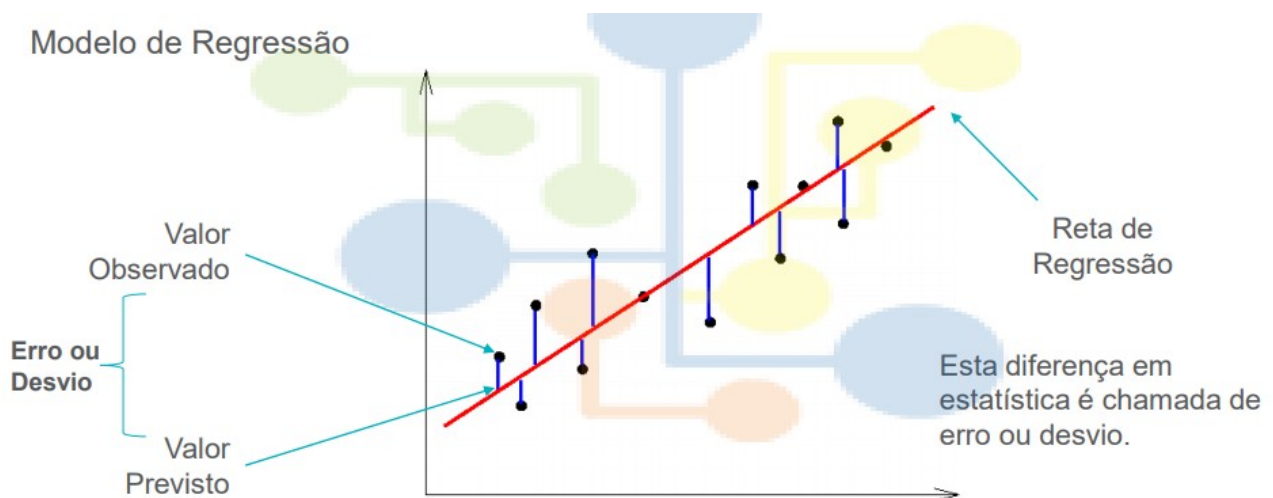
Assim, a análise de regressão compreende quatro tipos básicos de modelos:

- Linear Simples
- Não Linear Simples
- Linear Múltiplo
- Não Linear Múltiplo

Qual o objetivo em se determinar a relação entre duas variáveis?

- Prever a população futura de uma cidade simulando a tendência de crescimento da população no passado
- Produtividade (Y) de uma área agrícola é alterada quando se aplica certa quantidade (X) de fertilizante sobre a terra

Diagrama de Dispersão (Scatter Plot):



Valor observado, valor previsto, erro ou desvio, reta de regressão.

Fórmula: $\hat{y} = a + bx$

Onde:

\hat{y} = valor previsto de y dado um valor para x

x = variável independente

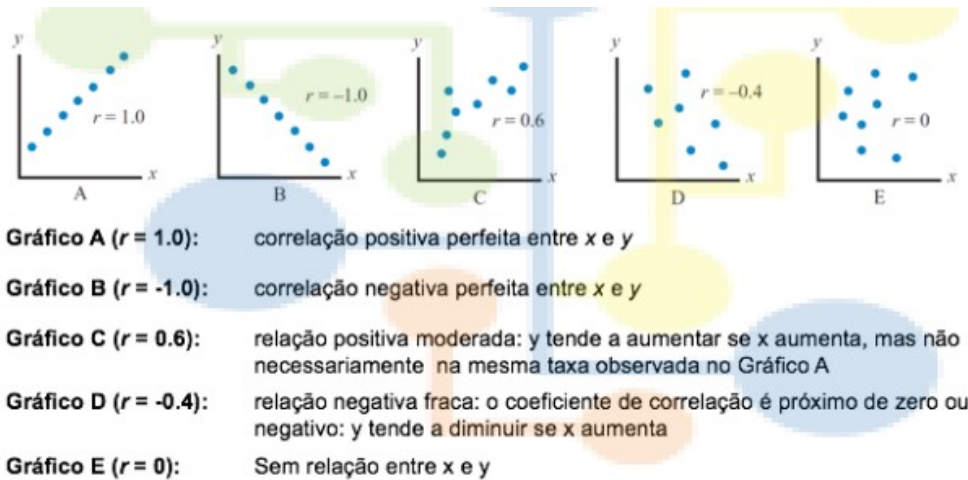
a = ponto onde a linha intercepta o eixo y

b = inclinação da linha reta

Método dos Mínimos Quadrados

Esse método definirá uma reta que minimizará a soma das distâncias ao quadrado entre os pontos plotados (X, Y) e a reta (que são os valores previstos de X', Y').

Coeficiente de Correlação



Os valores de r variam entre **-1.0** (uma forte relação negativa) até **+1.0**, uma forte relação positiva.

O coeficiente de determinação indica o quanto a reta de regressão explica o ajuste da reta, enquanto que o coeficiente de correlação deve ser usado como uma medida de força da relação entre as variáveis.

- (SST – Sum Square Total) Soma Total dos Quadrados (STQ) – Mostra a variação em Y em torno da própria média.
- (SSR – Sum Square Regression) Soma dos Quadrados de Regressão (SQR) – Oferece a variação de Y considerando as variáveis X utilizadas no modelo.
- (SSE – Sum Square Error) Soma dos Quadrados dos Resíduos (SQU) – Variação de Y que não é explicada pelo modelo elaborado.

$$STQ = SQR + SQU$$

Se o SSR é alto e o SSE é baixo, o Modelo de Regressão explica bem a variação nas previsões

Se o SSR é baixo e o SSE é alto, o Modelo de Regressão não explica bem a variação nas previsões

- SSR = medida da variação que pode ser explicada
- SSE = medida da variação que não pode ser explicada
- SST = medida da variação total

Coeficiente de Ajuste R²

$$R^2 = \frac{SQR}{SQR + SQU} = \frac{SQR}{STQ}$$

O coeficiente de ajuste R² não diz aos analistas se uma determinada variável explicativa é estatisticamente significativa e se esta variável é a causa verdadeira da alteração de comportamento da variável dependente.

Avaliando o Modelo de Regressão

Típicos problemas que podem ser resolvidos com Regressão:

- Quantos computadores serão vendidos no próximo mês?
- Quantas pessoas vão acessar nosso web site na próxima semana?
- Qual o salário de uma pessoa de acordo com a performance escolar?
- Qual o total de vendas relacionado ao número de seguidores em redes sociais?

Interpretando Modelos de Regressão Linear Simples e Múltipla

- Teste F de Significância Global
 - O modelo é útil para prever (alvo)?
- Testes de Significância Individuais
 - Quais variáveis estão relacionadas com (alvo)?
- Coeficientes R^2 e R^2 Ajustado
 - Qual percentual de variabilidade é explicado pelas variáveis usadas no modelo?
- Coeficientes
 - Valores que compõe a equação.

Regras Gerais

- Modelo é útil para prever o *preço*, se o valor-p do teste F é menor que 0,05.
- Há evidências de que uma variável está relacionada com o valor previsto, se o valor-p for menor que 0,05.
- O R^2 indica quanto da variabilidade de y é explicado pelas variáveis preditoras. Pode ser necessário incluir mais variáveis no modelo para aumentar este coeficiente.
- O objetivo da regressão é encontrar os coeficientes que permitem construir a equação de regressão e fazer as previsões.

Interpretando o Valor-p

Data Science é uma área multidisciplinar, que emprega conceitos de diversas áreas diferentes.

Data Science e Estatística NÃO são a mesma coisa, mas a Estatística fornece ferramentas importantes principalmente para interpretabilidade dos modelos de Machine Learning.

O **valor-p** é amplamente usado para interpretar modelos de regressão ou mesmo quando empregamos análise estatística aos dados.

Ponto de Vista Estatístico:

Os conceitos de valor-p e nível de significância são aspectos importantes dos testes de hipóteses e métodos estatísticos, como regressão. No entanto, eles podem ser um pouco difíceis de entender, especialmente para iniciantes, e uma boa compreensão desses conceitos pode ajudar bastante no entendimento do aprendizado de máquina.

Vamos imaginar o seguinte exemplo:

Considere dois grupos dentro de uma determinada população: um grupo de controle e um grupo experimental. O grupo experimental é uma amostra aleatória retirada da população sobre a qual um experimento será realizado e, em seguida, será comparada com o grupo de controle. A diferença nos grupos é definida em termos de uma estatística de teste, como o teste t de Student (por exemplo, uma empresa deseja saber se seu produto é comprado mais por homens ou mulheres).

Precisamos definir dois termos adicionais: uma **hipótese nula significa que não há diferença entre os dois grupos**, enquanto a **hipótese alternativa significa que há uma diferença estatisticamente significativa entre os dois grupos**.

Assumiremos que a hipótese nula é verdadeira, ou seja, não há diferença entre dois grupos. Em seguida, o experimento é realizado no grupo experimental. Em seguida, é verificado se há algum efeito significativo no grupo ou não.

Agora vamos considerar a importância do valor-p. Precisamos calcular a probabilidade de que o efeito no grupo seja atribuído ao acaso. Se você repetir o experimento repetidamente no mesmo tamanho de amostra para o grupo experimental, qual porcentagem de tempo você vê uma diferença no grupo experimental por acaso?

O valor-p é usado para avaliar de fato a força das hipóteses nula e alternativa. Os valores-p são números decimais entre 0 e 1, que servem como referência probabilística para pesar a hipótese.

Às vezes, o valor-p também é expresso como uma porcentagem.

Um valor-p maior que 0,05 significa que, em mais de 1/20 das vezes, o experimento não mostra diferença entre os dois grupos. O valor 0,05 é normalmente usado como referência e é conhecido como nível de significância (α).

Em um problema de regressão, você deseja que o valor-p seja muito menor que 0,05 para a variável ser considerada uma variável significativa.

Normalmente, um pequeno valor-p ($<0,05$) sugere que a hipótese nula deve ser rejeitada, enquanto um grande valor-p ($> 0,05$) indica que a hipótese nula deve não deve ser rejeitada devido à falta de evidências contra ela.

Valores iguais ou próximos a 0,05 sugerem que o Cientista de Dados deve tomar a decisão por si mesmo!

Ponto de Vista em Data Science:

Um pequeno valor-p indica que é improvável observar uma associação tão substancial entre o preditor e a resposta devido ao acaso, na ausência de qualquer associação real entre o preditor e a resposta.

Consequentemente, se vemos um pequeno valor-p podemos deduzir que há uma associação entre o preditor e a resposta. Isso significa que rejeitamos a hipótese nula, ou seja, afirmamos que existe um relacionamento entre as duas variáveis se o valor-p for pequeno o suficiente.

No caso de um grande valor-p deduzimos que não há uma associação entre o preditor e a resposta!

O valor-p representa a chance ou a probabilidade do efeito (ou da diferença) observada entre as variáveis ser devido ao acaso, e não aos fatores que estão sendo estudados.

Regularização

O modelo de regressão utiliza as variáveis explanatórias para explicar a variabilidade da variável resposta!

Mas o que acontece quando o número de variáveis explanatórias é muito grande?

A técnica de mínimos quadrados, nesta situação, pode não permitir previsões com precisão e nem permitir uma interpretação ideal para o modelo.

Isso significa que muitas variáveis seriam ajustadas e o modelo ficaria super estimado, com uma variância infinita, sendo inviável o método dos mínimos quadrados.

Temos basicamente 3 métodos que nos auxiliam quando o número de variáveis é maior que o número de observações:

- Seleção de um subconjunto de coeficientes
- Reduzir o valor dos coeficientes (Regularização)
- Reduzir a dimensão

Uma regressão com diversos coeficientes regressores torna o modelo como um todo muito mais complexo e pode tirar características de interpretabilidade

Métodos de Regularização – Shrinkage Methods (Métodos de Encolhimento):

- Ridge Regression – A Ridge Regression é um método de regularização do modelo que tem como principal objetivo suavizar atributos que sejam relacionados uns aos outros e que aumentam o ruído no modelo (multicolinearidade).
- LASSO Regression (Least Absolute Shrinkage and Selection Operator) – O LASSO tem o mesmo mecanismo de penalização dos coeficientes com um alto grau de correlação entre si, mas que usa o mecanismo de penalizar os coeficientes de acordo com o seu valor absoluto.

Regressão Logística

A regressão logística é uma técnica estatística que tem como objetivo modelar, a partir de um conjunto de observações, a relação “logística” entre uma variável resposta e uma série de variáveis explicativas numéricas (contínuas, discretas) e/ou categóricas.

A regressão logística é amplamente usada em ciências médicas e sociais, e tem outras denominações, como modelo logístico, modelo logit, e classificador de máxima entropia.

Na Regressão Logística, a variável resposta é binária:

1 → acontecimento de interesse (sucesso)

0 → acontecimento complementar (insucesso)

Transformação logit:

The diagram illustrates the derivation of the logit transformation through three steps, connected by arrows:

$$g(x) = \ln\left(\frac{\pi(x)}{1 - \pi(x)}\right)$$
$$g(x) = \ln\left(\frac{\frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}}{1 - \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}}\right) = \ln\left(\frac{\frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}}{\frac{1}{1 + e^{\beta_0 + \beta_1 x}}}\right)$$
$$g(x) = \ln(e^{\beta_0 + \beta_1 x}) = \beta_0 + \beta_1 x$$

An arrow labeled "Logaritmo" points to the final equation.

Regressão Logística é útil para modelar a probabilidade de um evento ocorrer como função de outros fatores. É um modelo linear generalizado que usa como função de ligação a função logit.

A regressão logística é utilizada em áreas tais como:

- Em medicina, permite por exemplo determinar os fatores que caracterizam um grupo de indivíduos doentes em relação a indivíduos saudáveis.
- Na área de seguros, permite encontrar frações de clientes que sejam sensíveis a determinada política securitária em relação a um dado risco particular.
- Em instituições financeiras, pode detectar os grupos de risco para a subscrição de um crédito.
- Em econometria, permite explicar uma variável discreta, como por exemplo as intenções de voto em atos eleitorais.

Regressão Vantagens e Desvantagens

Tipos de regressão:

- Simple Linear Regression
- Multiple Linear Regression
- Ridge Regression
- Lasso Regression

- Logistic Regression
- Polynomial Regression
- Stepwise Regression
- Elastic Net Regression

Não coloque a culpa no modelo, se você não entregar o que ele está esperando.

Vantagens da Regressão:

- Previsão do Futuro
- Apoio à Tomada de Decisão
- Correção de Erros
- Novos Insights

Importantes Desvantagens da Regressão:

- Apenas consideram relacionamento linear
- Toma como base a média da variável dependente
- Sensível a Outliers
- Regressão Linear assume que os dados são independentes

Regressão – Exercício – Solução

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 4 - Regressão</font>
#### Exercício:
Parte 1: Desenvolva o código necessário para a fórmula básica da regressão linear simples, calculando os coeficientes.
Parte 2: Use o modelo para fazer previsões.
O dataset abaixo contém dados sobre medidas da cabeça de seres humanos e o peso do cérebro. Seu trabalho é criar um modelo de regressão linear
simples que receba uma medida como entrada e faça a previsão do peso do cérebro!
# ->
# Imports
import numpy as np
import pandas as pd
# ->
# Carregando os dados
data = pd.read_csv('dados/pesos2.csv')
data.head()
# ->
# Definindo variáveis x e y
X = data['Head Size'].values
Y = data['Brain Weight'].values
# ->
# Calculando os Coeficientes
# Média de x e y
mean_x = np.mean(X)
mean_y = np.mean(Y)
# Número total de valores
n = len(X)
# Usando a fórmula para calcular a e b
numer = 0
denom = 0
for i in range(n):
    numer += (X[i] - mean_x) * (Y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2
b = numer / denom
a = mean_y - (b * mean_x)
# Imprimindo os coeficientes
print(a, b)
# ->
# Fazendo previsões
```

```
# y = a + bx
y = 325.57 + (0.26 * 4450)
print("O peso do cérebro é:", y)
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Padronização de Dados

Muitos algoritmos de aprendizado de máquina fazem suposições sobre os dados e geralmente é uma boa ideia preparar os dados para melhor expor a estrutura do problema aos algoritmos de aprendizado de máquina que você pretende usar, sendo esta uma parte importante da etapa de pré-processamento de dados, quando trabalhamos com Data Science.

Você quase sempre precisa pré-processar seus dados. É um passo praticamente obrigatório.

Uma dificuldade é que algoritmos diferentes fazem suposições diferentes sobre os dados e podem exigir transformações diferentes. Além disso, quando você segue todas as regras e prepara seus dados, às vezes os algoritmos podem oferecer melhores resultados sem o pré-processamento. Não há fórmulas mágicas, pois tudo depende dos dados e os dados sempre mudam.

Geralmente, recomendamos a criação de muitos modos de exibição e transformações diferentes dos dados e, em seguida, exercitar um punhado de algoritmos em cada bloco do seu conjunto de dados. Isso ajudará você a identificar quais transformações de dados podem ser melhores para expor a estrutura do problema em geral. Vamos listar agora as 4 diferentes técnicas de pré-processamento de dados para aprendizado de máquina, pois esta é uma das maiores dúvidas de quem começa em Data Science.

Vamos considerar um problema de classificação binária em que todos os atributos são numéricos (representando dados qualitativos e quantitativos) e possuem escalas diferentes. É um ótimo exemplo de dados que pode se beneficiar do pré-processamento.

1. Aplicar Escala aos dados

Quando seus dados são compostos de atributos com escalas variáveis, muitos algoritmos de aprendizado de máquina podem se beneficiar do reescalonamento dos atributos para que todos tenham a mesma escala.

Muitas vezes isso é conhecido como normalização (embora não seja o termo ideal) e os atributos são frequentemente redimensionados no intervalo entre 0 e 1. Isso é útil para algoritmos de otimização usados no núcleo de algoritmos de aprendizado de máquina como o gradiente descendente. Também é útil para algoritmos que pesam entradas como regressão e redes neurais e algoritmos que usam medidas de distância como K-Nearest Neighbors (KNN).

2. Padronização dos Dados

A padronização é uma técnica útil para transformar atributos com uma distribuição gaussiana e diferentes médias e desvios padrão para uma distribuição Gaussiana padrão com uma média de 0 e um desvio padrão de 1.

É mais adequado para técnicas que pressupõem uma distribuição gaussiana nas variáveis de entrada e funcionam melhor com dados reescalados, como regressão linear, regressão logística e análise discriminante linear.

3. Normalização dos Dados

A normalização refere-se ao reescalonamento de cada observação (linha) para ter um comprimento de 1 (chamado de norma unitária em álgebra linear).

Esse pré-processamento pode ser útil para conjuntos de dados esparsos (muitos zeros) com atributos de escalas variadas ao usar algoritmos que ponderam valores de entrada, como redes neurais e algoritmos que usam medidas de distância, como K-Nearest Neighbors (KNN).

4. Binarização dos Dados

Você pode transformar seus dados usando um limite binário. Todos os valores acima do limite são marcados como 1 e todos iguais ou inferiores são marcados como 0.

Isso é chamado de binarizar seus dados ou limitar seus dados. Pode ser útil quando você tem probabilidades que você deseja tornar valores nítidos para leitura e interpretação. Também é útil quando na engenharia de recursos adicionamos novos recursos que indicam algo significativo.

Correlação e Causalidade

Só porque (A) acontece juntamente com (B) não significa que (A) causa (B).

Para analisar relação de causalidade, é necessário investigação adicional em função de diferentes cenários que podem ocorrer:

1. (A) causa realmente (B);
2. (B) pode ser a causa de (A);
3. Um terceiro fator (C) pode ser causa tanto de (A) como de (B);
4. Pode ser uma combinação das três situações anteriores: (A) causa (B) e ao mesmo tempo (B) causa também (A);
5. A correlação pode ser apenas uma coincidência, ou seja, os dois eventos não têm qualquer relação para além do fato de ocorrerem ao mesmo tempo. (Se estivermos falando de um estudo científico, utilizar uma amostra grande ajuda a reduzir a probabilidade de coincidência).

Então como se determina a causalidade?

Depende sobretudo da complexidade do problema, mas a verdade é que a causalidade dificilmente poderá ser determinada com certeza absoluta.

Daí que em ciência já está subentendido que não existem verdades absolutas e que todas as teorias estão abertas a revisão face a novas evidências. No entanto, muitos erros podem ser evitados se tivermos mais cuidado com as conclusões precipitadas.

Utilizando o método científico é possível muitas vezes estabelecer uma relação de causa-efeito com uma segurança confortável. O que acaba por ter mais importância no final é a reprodutibilidade da relação causa-efeito e a possibilidade de fazer previsões corretas sobre eventos futuros.

A indústria do tabaco não pode continuar a alegar que a correlação entre o tabaco e o cancro de pulmão não implica necessariamente causalidade porque existe uma montanha de evidências científicas a favor da relação causa-efeito. Já o movimento anti-vacinação não possui quaisquer evidências concretas que suportem a afirmação de que as vacinas causam autismo. É aí que reside a diferença fundamental.

Modelo de Regressão Linear Simples / Modelo de Regressão Linear Múltipla / Regularização - Regressão Lasso e Ridge

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 5 - Regressão</font>
# Regressão Linear Simples
## Definindo o Problema de Negócio
Nosso objetivo é construir um modelo de Machine Learning que seja capaz de fazer previsões sobre a taxa média de ocupação de casas na região de Boston, EUA, por proprietários. A variável a ser prevista é um valor numérico que representa a mediana da taxa de ocupação das casas em Boston. Para cada casa temos diversas variáveis explanatórias. Sendo assim, podemos resolver este problema empregando Regressão Linear Simples ou Múltipla.
## Definindo o Dataset
Usaremos o Boston Housing Dataset, que é um conjunto de dados que tem a taxa média de ocupação das casas, juntamente com outras 13 variáveis que podem estar relacionadas aos preços das casas. Esses são os fatores como condições socioeconômicas, condições ambientais, instalações educacionais e alguns outros fatores semelhantes. Existem 506 observações nos dados para 14 variáveis. Existem 12 variáveis numéricas em nosso conjunto de dados e 1 variável categórica. O objetivo deste projeto é construir um modelo de regressão linear para estimar a taxa média de ocupação das casas pelos proprietários em Boston.
# ->
from IPython.display import Image
Image('imagens/boston.png')
Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/housing/
1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per 10,000
11. PTRATIO: pupil-teacher ratio by town
12. B:  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. TARGET: Median value of owner-occupied homes in $1000's
# ->
# Carregando o Dataset Boston Houses
from sklearn.datasets import load_boston
boston = load_boston()
# Carregando Bibliotecas Python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
## Análise Exploratória
# ->
# Convertendo o dataset em um dataframe com Pandas
dataset = pd.DataFrame(boston.data, columns = boston.feature_names)
dataset['target'] = boston.target
# ->
dataset.head(5)
# ->
# Calculando a média da variável de resposta
valor_medio_esperado_na_previsao = dataset['target'].mean()
# ->
valor_medio_esperado_na_previsao
# ->
# Calculando (simulando) o SSE
# O SSE é a diferença ao quadrado entre o valor previsto e o valor observado.
# Considerando que o valor previsto seja igual a média, podemos considerar que
# y = média da variável target (valores observados).
# Estamos apenas simulando o SSE, uma vez que a regressão ainda não foi criada e os valores previstos
# ainda não foram calculados.
squared_errors = pd.Series(valor_medio_esperado_na_previsao - dataset['target'])**2
SSE = np.sum(squared_errors)
print('Soma dos Quadrados dos Erros (SSE): %01.f % SSE)' % SSE)
# ->
# Histograma dos erros
# Temos mais erros "pequenos", ou seja, mais valores próximos à média.
hist_plot = squared_errors.plot('hist')
```

Para Regressão Linear Simples usaremos como variável explanatória a variável RM que representa o número médio de quartos nas casas.

```
# ->
# Função para calcular o desvio padrão
def calc_desvio_padrao(variable, bias = 0):
    observations = float(len(variable))
    return np.sqrt(np.sum((variable - np.mean(variable))**2) / (observations - min(bias, 1)))
# ->
# Imprimindo o desvio padrão via fórmula e via NumPy da variável RM
print('Resultado da Função: %0.5f Resultado do Numpy: %0.5f' % (calc_desvio_padrao(dataset['RM']), np.std(dataset['RM'])))
# ->
# Funções para calcular a variância da variável RM e a correlação com a variável target
def covariance(variable_1, variable_2, bias = 0):
    observations = float(len(variable_1))
    return np.sum((variable_1 - np.mean(variable_1)) * (variable_2 - np.mean(variable_2))) / (observations - min(bias, 1))
def standardize(variable):
    return (variable - np.mean(variable)) / np.std(variable)
def correlation(var1, var2, bias = 0):
    return covariance(standardize(var1), standardize(var2), bias)
# ->
# Compara o resultado das nossas funções com a função pearsonr do SciPy
from scipy.stats.stats import pearsonr
print('Nossa estimativa de Correlação: %0.5f' % (correlation(dataset['RM'], dataset['target'])))
print('Correlação a partir da função pearsonr do SciPy: %0.5f' % pearsonr(dataset['RM'], dataset['target'])[0])
# ->
# Definindo o range dos valores de x e y
x_range = [dataset['RM'].min(), dataset['RM'].max()]
y_range = [dataset['target'].min(), dataset['target'].max()]
# ->
# Plot dos valores de x e y com a média
scatter_plot = dataset.plot(kind = 'scatter', x = 'RM', y = 'target', xlim = x_range, ylim = y_range)
# Cálculo da média
meanY = scatter_plot.plot(x_range, [dataset['target'].mean(), dataset['target'].mean()], '--', color = 'red', linewidth = 1)
meanX = scatter_plot.plot([dataset['RM'].mean(), dataset['RM'].mean()], y_range, '--', color = 'red', linewidth = 1)
## Regressão Linear com o StatsModels
https://www.statsmodels.org/stable/index.html
# ->
# Importando as funções
import statsmodels.api as sm
# ->
# Gerando X e Y. Vamos adicionar a constante ao valor de X, gerando uma matrix.
y = dataset['target']
X = dataset['RM']
# ->
# Esse comando adiciona os valores dos coeficientes à variável X (o bias será calculado internamente pela função)
X = sm.add_constant(X)
# ->
X.head()
# ->
# Criando o modelo de regressão
modelo = sm.OLS(y, X)
# Treinando o modelo
modelo_v1 = modelo.fit()
# ->
print(modelo_v1.summary())
# ->
print(modelo_v1.params)
# ->
# Gerando os valores previstos
valores_previstos = modelo_v1.predict(X)
valores_previstos
# ->
# Fazendo previsões com o modelo treinado
RM = 5
Xp = np.array([1, RM])
print("Se RM = %01.f nosso modelo prevê que a mediana da taxa de ocupação é %0.1f" % (RM, modelo_v1.predict(Xp)))
### Gerando um ScatterPlot com a Linha de Regressão
# ->
# Range de valores para x e y
x_range = [dataset['RM'].min(), dataset['RM'].max()]
y_range = [dataset['target'].min(), dataset['target'].max()]
# ->
# Primeira camada do Scatter Plot
scatter_plot = dataset.plot(kind = 'scatter', x = 'RM', y = 'target', xlim = x_range, ylim = y_range)
# Segunda camada do Scatter Plot (médias)
meanY = scatter_plot.plot(x_range, [dataset['target'].mean(), dataset['target'].mean()], '--', color = 'red', linewidth = 1)
meanX = scatter_plot.plot([dataset['RM'].mean(), dataset['RM'].mean()], y_range, '--', color = 'red', linewidth = 1)
# Terceira camada do Scatter Plot (linha de regressão)
regression_line = scatter_plot.plot(dataset['RM'], valores_previstos, '-', color = 'orange', linewidth = 2)
```

```

# ->
# Gerando os resíduos
residuos = dataset['target'] - valores_previstos
residuos_normalizados = standardize(residuos)
# ->
# ScatterPlot dos resíduos
residual_scatter_plot = plt.plot(dataset['RM'], residuos_normalizados, 'bp')
plt.xlabel('RM')
plt.ylabel('Resíduos Normalizados')
mean_residual = plt.plot([int(x_range[0]), round(x_range[1], 0)], [0, 0], '-', color = 'red', linewidth = 3)
upper_bound = plt.plot([int(x_range[0]), round(x_range[1], 0)], [3, 3], '--', color = 'red', linewidth = 2)
lower_bound = plt.plot([int(x_range[0]), round(x_range[1], 0)], [-3, -3], '--', color = 'red', linewidth = 2)
plt.grid()
## Regressão Linear com Scikit-Learn
https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LinearRegression.html
# ->
from sklearn import linear_model
# ->
# Cria o objeto
modelo_v2 = linear_model.LinearRegression(normalize = False, fit_intercept = True)
# ->
# Define os valores de x e y
num_observ = len(dataset)
X = dataset['RM'].values.reshape((num_observ, 1)) # X deve sempre ser uma matriz e nunca um vetor
y = dataset['target'].values # y pode ser um vetor
# ->
type(X)
# ->
# Número de dimensões de X (matriz)
np.ndim(X)
# ->
print(X)
# ->
type(y)
# ->
# Número de dimensões de y (vetor)
np.ndim(y)
# ->
print(y)
# ->
# Treinamento do modelo - fit()
modelo_v2.fit(X, y)
# ->
# Imprime os coeficientes
print(modelo_v2.coef_)
print(modelo_v2.intercept_)
# ->
# Imprime as previsões
print(modelo_v2.predict(X))
# ->
# Fazendo previsões com o modelo treinado
RM = 5
# Xp = np.array(RM)
Xp = np.array(RM).reshape(-1, 1)
print("Se RM = %01.f nosso modelo prevê que a mediana da taxa de ocupação é %0.1f" % (RM, modelo_v2.predict(Xp)))
### Comparação StatsModels x ScikitLearn
# ->
from sklearn.datasets import make_regression
HX, Hy = make_regression(n_samples = 1000000, n_features = 1, n_targets = 1, random_state = 101)
# ->
%%time
sk_linear_regression = linear_model.LinearRegression(normalize=False, fit_intercept=True)
sk_linear_regression.fit(HX, Hy)
# ->
%%time
sm_linear_regression = sm.OLS(Hy, sm.add_constant(HX))
sm_linear_regression.fit()
## Cost Function de um Modelo de Regressão Linear
O objetivo da regressão linear é buscar a equação de uma linha de regressão que minimize a soma dos erros ao quadrado, da diferença entre o valor observado de y e o valor previsto.
Existem alguns métodos para minimização da Cost Function tais como: Pseudo-inversão, Fatorização e Gradient Descent.
# ->
from IPython.display import Image
Image('imagens/formula-regressao.png')
# ->
from IPython.display import Image
Image('imagens/formulas.png')
## Por Que Usamos o Erro ao Quadrado?

```

```

# ->
# Definindo 2 conjuntos de dados
import numpy as np
x = np.array([9.5, 8.5, 8.0, 7.0, 6.0])
# ->
# Função para cálculo da Cost
Function
def squared_cost(v, e):
    return np.sum((v - e) ** 2)
# ->
# A função fmin() tenta descobrir o valor do somatório mínimo dos quadrados
from scipy.optimize import fmin
xopt = fmin(squared_cost, x0 = 0, xtol = 1e-8, args = (x,))
# ->
print ('Resultado da Otimização: %0.1f' % (xopt[0]))
print ('Média: %0.1f' % (np.mean(x)))
print ('Mediana: %0.1f' % (np.median(x)))
# ->
def absolute_cost(v,e):
    return np.sum(np.abs(v - e))
# ->
xopt = fmin(absolute_cost, x0 = 0, xtol = 1e-8, args = (x,))
# ->
print ('Resultado da Otimização: %0.1f' % (xopt[0]))
print ('Média %0.1f' % (np.mean(x)))
print ('Mediana %0.1f' % (np.median(x)))
## Minimizando a Cost Function
Minimizando a Cost Function com Pseudo-Inversão
# ->
from IPython.display import Image
Image('imagens/formula-cost-function.png')
# ->
from IPython.display import Image
Image('imagens/formula-pseudo.png')
# ->
# Definindo x e y
num_observ = len(dataset)
X = dataset['RM'].values.reshape((num_observ, 1)) # X deve ser uma matriz
Xb = np.column_stack((X, np.ones(num_observ)))
y = dataset['target'].values # y pode ser um vetor
# ->
# Funções para matriz inversa e equações normais
def matriz_inversa(X, y, pseudo = False):
    if pseudo:
        return np.dot(np.linalg.pinv(np.dot(X.T, X)), np.dot(X.T,y))
    else:
        return np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T,y))
def normal_equations(X,y):
    return np.linalg.solve(np.dot(X.T,X), np.dot(X.T,y))
# ->
# Imprime os valores
print (matriz_inversa(Xb, y))
print (matriz_inversa(Xb, y, pseudo = True))
print (normal_equations(Xb, y))
## Aplicando o Gradiente Descendente
# ->
from IPython.display import Image
Image('imagens/gradient-descent.png')
# ->
from IPython.display import Image
Image('imagens/formula-gradient-descent1.png')
# ->
# Alfa é chamado de taxa de aprendizagem
from IPython.display import Image
Image('imagens/formula-gradient-descent2.png')
# ->
# Definindo x e y
observations = len(dataset)
X = dataset['RM'].values.reshape((observations,1))
X = np.column_stack((X,np.ones(observations)))
y = dataset['target'].values
# ->
import random
# Valores randômicos para os coeficientes iniciais
def random_w( p ):
    return np.array([np.random.normal() for j in range(p)])
# Cálculo da hipótese (valor aproximado de y)
def hypothesis(X,w):

```

```

    return np.dot(X,w)
# Cálculo da função de perda (loss)
def loss(X,w,y):
    return hypothesis(X,w) - y
# Cálculo do erro
def squared_loss(X,w,y):
    return loss(X,w,y)**2
# Cálculo do gradiente
def gradient(X,w,y):
    gradients = list()
    n = float(len( y ))
    for j in range(len(w)):
        gradients.append(np.sum(loss(X,w,y) * X[:,j]) / n)
    return gradients
# Atualização do valor dos coeficientes
def update(X,w,y, alpha = 0.01):
    return [t - alpha*g for t, g in zip(w, gradient(X,w,y))]
# Otimização do modelo
def optimize(X,y, alpha = 0.01, eta = 10**-12, iterations = 1000):
    w = random_w(X.shape[1])
    path = list()
    for k in range(iterations):
        SSL = np.sum(squared_loss(X,w,y))
        new_w = update(X,w,y, alpha = alpha)
        new_SSL = np.sum(squared_loss(X,new_w,y))
        w = new_w
        if k >= 5 and (new_SSL - SSL <= eta and new_SSL - SSL >= -eta):
            path.append(new_SSL)
            return w, path
        if k % (iterations / 20) == 0:
            path.append(new_SSL)
    return w, path
# ->
# Definindo o valor de alfa
# Alfa é chamado de taxa de aprendizagem
alpha = 0.048
# Otimizando a Cost Function
w, path = optimize(X, y, alpha, eta = 10**-12, iterations = 25000)
# ->
# Imprimindo o resultado
print ("Valor Final dos Coeficientes: %s" % w)
# ->
# Imprimindo o resultado
print ("Percorrendo o Caminho do Gradiente em que o erro ao quadrado era:\n\n %s" % path)
# ->
from IPython.display import Image
Image('imagens/gradient-descent.png')
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Data Science Academy - Machine Learning

Capítulo 5 - Regressão

Regressão Linear Múltipla

Definindo o Problema de Negócio

Nosso objetivo é construir um modelo de Machine Learning que seja capaz de fazer previsões sobre a taxa média de ocupação de casas na região de Boston, EUA, por proprietários. A variável a ser prevista é um valor numérico que representa a mediana da taxa de ocupação das casas em Boston. Para cada casa temos diversas variáveis explanatórias. Sendo assim, podemos resolver este problema empregando Regressão Linear Simples ou Múltipla.

Definindo o Dataset

Usaremos o Boston Housing Dataset, que é um conjunto de dados que tem a taxa média de ocupação das casas, juntamente com outras 13 variáveis que podem estar relacionadas aos preços das casas. Esses são os fatores como condições socioeconômicas, condições ambientais, instalações educacionais e alguns outros fatores semelhantes. Existem 506 observações nos dados para 14 variáveis. Existem 12 variáveis numéricas em nosso conjunto de dados e 1 variável categórica. O objetivo deste projeto é construir um modelo de regressão linear para estimar a taxa média de ocupação das casas pelos proprietários em Boston.

->

from IPython.display import Image

Image('imagens/boston.png')

Dataset: <https://archive.ics.uci.edu/ml/machine-learning-databases/housing/>

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-residential acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per 10,000

```

11. PTRATIO: pupil-teacher ratio by town
12. B:  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. TARGET: Median value of owner-occupied homes in $1000's
# ->
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.datasets import load_boston
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
import statsmodels.formula.api as smf
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
# ->
# Carregando o dataset
boston = load_boston()
dataset = pd.DataFrame(boston.data, columns = boston.feature_names)
dataset['target'] = boston.target
# ->
# Gerando número de observações e variáveis
observations = len(dataset)
variables = dataset.columns[:-1]
# ->
# Coletando x e y
X = dataset.iloc[:, :-1]
y = dataset['target'].values
# ->
# Variáveis explanatórias
X.head()
# ->
# Variável target
y
## Usando Múltiplos Atributos com StatsModels
# ->
Xc = sm.add_constant(X)
modelo = sm.OLS(y, Xc)
modelo_v1 = modelo.fit()
# ->
modelo_v1.summary()
#### Matriz de Correlação
# ->
# Gerando a matriz
X = dataset.iloc[:, :-1]
matriz_corr = X.corr()
print (matriz_corr)
# ->
# Criando um Correlation Plot
def visualize_correlation_matrix(data, hurdle = 0.0):
    R = np.corrcoef(data, rowvar = 0)
    R[np.where(np.abs(R) < hurdle)] = 0.0
    heatmap = plt.pcolor(R, cmap = mpl.cm.coolwarm, alpha = 0.8)
    heatmap.axes.set_frame_on(False)
    heatmap.axes.set_yticks(np.arange(R.shape[0]) + 0.5, minor = False)
    heatmap.axes.set_xticks(np.arange(R.shape[1]) + 0.5, minor = False)
    heatmap.axes.set_xticklabels(variables, minor = False)
    plt.xticks(rotation=90)
    heatmap.axes.set_yticklabels(variables, minor = False)
    plt.tick_params(axis = 'both', which = 'both', bottom = 'off', top = 'off', left = 'off', right = 'off')
    plt.colorbar()
    plt.show()
# ->
# Visualizando o Plot
visualize_correlation_matrix(X, hurdle = 0.5)
## Avaliando a Multicolinearidade
## Autovalores (Eigenvalues) e Autovetores (Eigenvectors)
Uma forma ainda mais automática de detectar associações multicolineares (e descobrir problemas numéricos em uma inversão de matriz) é usar autovetores. Explicados em termos simples, os autovetores são uma maneira muito inteligente de recombina a variância entre as variáveis, criando novos recursos acumulando toda a variância compartilhada. Tal recombinação pode ser obtida usando a função NumPy linalg.eig, resultando em um vetor de autovalores (representando a quantidade de variância recombina para cada nova variável) e autovetores (uma matriz nos dizendo como as novas variáveis se relacionam com as antigas).
# ->

```

```

# Gerando eigenvalues e eigenvectors
corr = np.corrcoef(X, rowvar = 0)
eigenvalues, eigenvectors = np.linalg.eig(corr)
Depois de extrair os autovalores, imprimimos em ordem decrescente e procuramos qualquer elemento cujo valor seja próximo de zero ou pequeno em comparação com os outros. Valores próximos a zero podem representar um problema real para equações normais e outros métodos de otimização baseados na inversão matricial. Valores pequenos representam uma fonte elevada, mas não crítica, de multicolinearidade. Se você detectar qualquer um desses valores baixos, anote a posição no vetor (lembre-se que os índices em Python começam por zero). O menor valor está na posição 8. Vamos buscar a posição 8 no autovetor.
# ->
print (eigenvalues)
Usando a posição do índice na lista de autovalores, podemos encontrar o vetor específico nos autovetores que contém as variáveis carregadas, ou seja, o nível de associação com os valores originais. No eigenvector, observamos valores nas posições de índice 2, 8 e 9, que estão realmente em destaque em termos de valor absoluto.
# ->
print (eigenvectors[:,8])
Agora nós imprimimos os nomes das variáveis para saber quais contribuem mais com seus valores para construir o autovetor. Associamos o vetor de variáveis com o eigenvector.
# ->
print (variables[2], variables[8], variables[9])
Tendo encontrado os culpados da multicolinearidade, o que devemos fazer com essas variáveis? A remoção de algumas delas é geralmente a melhor solução.
## Gradiente Descendente
# ->
# Gerando os dados
observations = len(dataset)
variables = dataset.columns
### Feature Scaling
Podemos aplicar Feature Scaling através de Padronização ou Normalização. Normalização aplica escala aos dados com intervalos entre 0 e 1. A Padronização divide a média pelo desvio padrão para obter uma unidade de variância. Vamos usar a Padronização (StandardScaler) pois nesse caso esta técnica ajusta os coeficientes e torna a superfície de erros mais "tratável".
# ->
# Aplicando Padronização
standardization = StandardScaler()
Xst = standardization.fit_transform(X)
original_means = standardization.mean_
original_std = standardization.scale_
# ->
# Gerando X e Y
Xst = np.column_stack((Xst,np.ones(observations)))
y = dataset['target'].values
# ->
import random
import numpy as np
def random_w( p ):
    return np.array([np.random.normal() for j in range(p)])
def hypothesis(X,w):
    return np.dot(X,w)
def loss(X,w,y):
    return hypothesis(X,w) - y
def squared_loss(X,w,y):
    return loss(X,w,y)**2
def gradient(X,w,y):
    gradients = list()
    n = float(len( y ))
    for j in range(len(w)):
        gradients.append(np.sum(loss(X,w,y) * X[:,j]) / n)
    return gradients
def update(X,w,y, alpha = 0.01):
    return [t - alpha*g for t, g in zip(w, gradient(X,w,y))]
def optimize(X,y, alpha = 0.01, eta = 10**-12, iterations = 1000):
    w = random_w(X.shape[1])
    path = list()
    for k in range(iterations):
        SSL = np.sum(squared_loss(X,w,y))
        new_w = update(X,w,y, alpha = alpha)
        new_SSL = np.sum(squared_loss(X,new_w,y))
        w = new_w
        if k>=5 and (new_SSL - SSL <= eta and new_SSL - SSL >= -eta):
            path.append(new_SSL)
            return w, path
        if k % (iterations / 20) == 0:
            path.append(new_SSL)
    return w, path
# ->
# Imprimindo o resultado
alpha = 0.01
w, path = optimize(Xst, y, alpha, eta = 10**-12, iterations = 20000)
print ("Coeficientes finais padronizados: " + ', '.join(map(lambda x: "%.04f" % x, w)))

```

```

# ->
# Desfazendo a Padronização
unstandardized_betas = w[:-1] / original_stds
unstandardized_bias = w[-1]-np.sum((original_means / original_stds) * w[:-1])
# ->
# Imprimindo o resultado
print ("%8s: %8.4f" % ('bias', unstandardized_bias))
for beta,varname in zip(unstandardized_betas, variables):
    print ("%8s: %8.4f" % (varname, beta))
## Importância dos Atributos
# ->
# Criando um modelo
modelo = linear_model.LinearRegression(normalize = False, fit_intercept = True)
# ->
# Treinando o modelo com dados não padronizados (em escalas diferentes)
modelo.fit(X,y)
# ->
# Imprimindo os coeficientes e as variáveis
for coef, var in sorted(zip(map(abs, modelo.coef_), dataset.columns[:-1]), reverse = True):
    print ("%6.3f" %s" % (coef,var))
# ->
# Padronizando os dados
standardization = StandardScaler()
Stand_coef_linear_reg = make_pipeline(standardization, modelo)
# ->
# Treinando o modelo com dados padronizados (na mesma escala)
Stand_coef_linear_reg.fit(X,y)
# ->
# Imprimindo os coeficientes e as variáveis
for coef, var in sorted(zip(map(abs, Stand_coef_linear_reg.steps[1][1].coef_), dataset.columns[:-1]), reverse = True):
    print ("%6.3f" %s" % (coef,var))
### Usando o R Squared
# ->
modelo = linear_model.LinearRegression(normalize = False, fit_intercept = True)
# ->
def r2_est(X,y):
    return r2_score(y, modelo.fit(X,y).predict(X))
# ->
print ('Coeficiente R2: %0.3f' % r2_est(X,y))
# ->
# Gera o impacto de cada atributo no R2
r2_impact = list()
for j in range(X.shape[1]):
    selection
    = [i for i in range(X.shape[1]) if i!=j]
    r2_impact.append(((r2_est(X,y) - r2_est(X.values[:,selection],y)), dataset.columns[j]))
for imp, varname in sorted(r2_impact, reverse = True):
    print ("%6.3f" %s' % (imp, varname))
### Fazendo Previsões com o Modelo de Regressão Linear Múltipla
# ->
# Imports
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
# ->
# Carregando o dataset
boston = load_boston()
dataset = pd.DataFrame(boston.data, columns = boston.feature_names)
dataset['target'] = boston.target
# ->
# Formato do Dataset
print("Boston housing dataset tem {} observações com {} variáveis cada uma.".format(*dataset.shape))
# ->
dataset.head()
# ->
# Coletando x e y
# Usaremos como variáveis explanatórias somente as 4 variáveis mais relevantes
X = dataset[['LSTAT', 'RM', 'DIS', 'PTRATIO']]
y = dataset['target'].values
# ->
X.head()
# ->
y
# ->
# Divisão em dados de treino e de teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)

```



```
# ->
# Cria o modelo
modelo = LinearRegression(normalize = False, fit_intercept = True)
# ->
# Treina o modelo
modelo_v2 = modelo.fit(X_train, y_train)
# ->
# Calcula a métrica R2 do nosso modelo
r2_score(y_test, modelo_v2.fit(X_train, y_train).predict(X_test))
# ->
# Produz a matriz com os novos dados de entrada para a previsão
LSTAT = 5
RM = 8
DIS = 6
PTRATIO = 19
# Lista com os valores das variáveis
dados_nova_casa = [LSTAT, RM, DIS, PTRATIO]
# Reshape
Xp = np.array(dados_nova_casa).reshape(1, -1)
# Previsão
print("Taxa Média de Ocupação Para a Casa:", modelo_v2.predict(Xp))
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 5 - Regressão</font>
# Regularização - Regressão Lasso e Ridge
```

Existem 2 métodos principais para regularização de um modelo de regressão linear: Regressão LASSO e Regressão Ridge. A Regressão Ridge é basicamente um modelo de regressão linear regularizado. O parâmetro λ é um escalar que também deve ser aprendido, usando um método chamado validação cruzada (cross validation).

Matematicamente, a Regressão Ridge estima uma função de regressão múltipla definida como:

```
# ->
from IPython.display import Image
Image('imagens/ridge.png')
```

A Regressão Ridge aplica restrição nos coeficientes (w). O termo de penalidade (λ) regulariza os coeficientes de forma que, se os coeficientes assumem valores grandes, a função de otimização é penalizada. Portanto, a Regressão Ridge reduz os coeficientes e ajuda a reduzir a complexidade e a multicolinearidade do modelo. Quando $\lambda \rightarrow 0$, a função de custo se torna semelhante à função de custo de regressão linear. Ou seja, diminuindo a restrição (λ baixo) nos recursos, o modelo se parecerá com o modelo de regressão linear.

A Regressão LASSO é um método de seleção de variáveis e encolhimento para modelos de regressão linear. O objetivo da Regressão LASSO é obter o subconjunto de preditores que minimiza o erro de previsão para uma variável de resposta quantitativa. O algoritmo faz isso impondo uma restrição nos parâmetros do modelo que faz com que os coeficientes de regressão de algumas variáveis converjam em direção a zero. Variáveis com um coeficiente de regressão igual a zero após o processo de restrição são excluídas do modelo. Portanto, quando você trabalhar um modelo de regressão, pode ser útil fazer uma Regressão LASSO para prever quantas variáveis seu modelo deve conter. Isso garante que seu modelo não seja excessivamente complexo e evita que o modelo se ajuste demais, o que pode resultar em um modelo tendencioso e ineficiente.

A única diferença da Regressão LASSO para a Regressão Ridge é que o termo de regularização está em valor absoluto. Mas essa diferença tem um enorme impacto. O método LASSO supera a desvantagem da Regressão Ridge, punindo não apenas os altos valores dos coeficientes β , mas definindo-os como zero se não forem relevantes. Portanto, você pode acabar com menos recursos incluídos no modelo em relação ao que começou, o que é uma grande vantagem.

Matematicamente, a Regressão LASSO estima uma função de regressão múltipla definida como:

```
# ->
from IPython.display import Image
Image('imagens/lasso.png')
### Dataset
https://www.kaggle.com/c/house-prices-advanced-regression-techniques/
http://jse.amstat.org/v19n3/decock.pdf
# ->
# Imports
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr
from sklearn.linear_model import LinearRegression, Ridge, LassoCV
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
# ->
# Carregando os dados
train = pd.read_csv('dados/treino.csv')
test = pd.read_csv("dados/teste.csv")
# ->
# Shape dos dados de treino
train.shape
# ->
train.head(10)
```

```

# ->
# Visualizando todos os dados
all_data = pd.concat((train.loc[:, 'MSSubClass': 'SaleCondition'],
                      test.loc[:, 'MSSubClass': 'SaleCondition']))
all_data.head(10)
# ->
# Pré-Processamento dos dados
new_price = {"price": train["SalePrice"], "log(price + 1)": np.log1p(train["SalePrice"])}
prices = pd.DataFrame(new_price)
matplotlib.rcParams['figure.figsize'] = (8.0, 5.0)
prices.hist()
# ->
# Log transform da variável target e remoção dos valores NA
train["SalePrice"] = np.log1p(train["SalePrice"])
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index
skewed_feats
# ->
# Aplicação das transformações a todos os dados e nova divisão em treino e teste
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
all_data = pd.get_dummies(all_data)
all_data = all_data.fillna(all_data.mean())
# Nova divisão em dados de treino e de teste
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y_train = train.SalePrice
# ->
# Função para calcular o RMSE
def rmse_cv(modelo):
    rmse = np.sqrt(-cross_val_score(modelo,
                                     X_train,
                                     y_train,
                                     scoring = "neg_mean_squared_error",
                                     cv = 5))
    return(rmse)
## Modelo de Regressão Linear Múltipla (sem regularização)
# ->
# Criando um modelo
modelo_lr = LinearRegression(normalize = False, fit_intercept = True)
# ->
# Treinando o modelo com dados não padronizados (em escalas diferentes)
modelo_lr.fit(X_train, y_train)
# ->
# Erro médio do modelo
rmse_cv(modelo_lr).mean()
## Modelo de Regressão Ridge
# ->
# Cria o modelo LASSO
modelo_ridge = Ridge()
# ->
# Cross Validation para encontrar os melhores valores dos parâmetros do modelo Ridge
cross_val_score(modelo_ridge,
                 X_train,
                 y_train,
                 scoring = "neg_mean_squared_error",
                 cv = 5)
# ->
# Calcula o erro do modelo
rmse_ridge = np.sqrt(-cross_val_score(modelo_ridge,
                                       X_train,
                                       y_train,
                                       scoring = "neg_mean_squared_error",
                                       cv = 5))
rmse_ridge
# ->
# Valores de alpha para testar com o modelo Ridge
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
# ->
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean() for alpha in alphas]
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validação")
plt.xlabel("Alpha")
plt.ylabel("RMSE")
# ->
# Erro médio do modelo
cv_ridge.mean()

```

```

## Modelo de Regressão LASSO
# ->
# Cria o modelo LASSO
modelo_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y_train)
# ->
# Erro médio do modelo
rmse_cv(modelo_lasso).mean()
# ->
# Coeficientes LASSO
coef = pd.Series(modelo_lasso.coef_, index = X_train.columns)
coef.head()
# ->
# Coeficientes LASSO mais relevantes e menos relevantes para o modelo
imp_coef = pd.concat([coef.sort_values().head(10), coef.sort_values().tail(10)])
matplotlib.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind = "barh")
plt.title("Coeficientes no Modelo LASSO")
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Implementando Regressão Logística em Linguagem R

```

# Regressão Logística
# Previsão e Detecção de Risco de Crédito

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap05/R")
getwd()

# Instalando os pacotes
install.packages("caret")
install.packages("ROCR")
install.packages("e1071")

# Carregando os pacotes
library(caret)
library(ROCR)
library(e1071)

# Carregando o dataset em um dataframe
credito_dataset <- read.csv("credito_dataset_final.csv", header = TRUE, sep = ",")
head(credito_dataset)
summary(credito_dataset)
str(credito_dataset)
View(credito_dataset)

##### Pré-processamento #####

# Transformando variáveis em fatores
to.factors <- function(df, variables){
  for (variable in variables){
    df[[variable]] <- as.factor(df[[variable]])
  }
  return(df)
}

# Normalização
scale.features <- function(df, variables){
  for (variable in variables){
    df[[variable]] <- scale(df[[variable]], center = T, scale = T)
  }
  return(df)
}

# Normalizando as variáveis
numeric.vars <- c("credit.duration.months", "age", "credit.amount")
credito_dataset_scaled <- scale.features(credito_dataset, numeric.vars)

# Variáveis do tipo fator
categorical.vars <- c("credit.rating", 'account.balance', 'previous.credit.payment.status',
  'credit.purpose', 'savings', 'employment.duration', 'installment.rate',

```

```

'marital.status', 'guarantor', 'residence.duration', 'current.assets',
'other.credits', 'apartment.type', 'bank.credits', 'occupation',
'dependents', 'telephone', 'foreign.worker')

# Aplicando as conversões ao dataset
credito_dataset_final <- to.factors(df = credito_dataset_scaled, variables = categorical.vars)
head(credito_dataset_final)
summary(credito_dataset_final)
View(credito_dataset_final)

# Preparando os dados de treino e de teste
indexes <- sample(1:nrow(credito_dataset_final), size = 0.6 * nrow(credito_dataset_final))
train.data <- credito_dataset_final[indexes,]
test.data <- credito_dataset_final[-indexes,]
class(train.data)
class(test.data)

# Separando os atributos e as classes
test.feature.vars <- test.data[,-1]
test.class.var <- test.data[,1]
class(test.feature.vars)

# Construindo o modelo de regressão logística
formula.init <- "credit.rating ~ ."
formula.init <- as.formula(formula.init)
help(glm)
modelo_v1 <- glm(formula = formula.init, data = train.data, family = "binomial")

# Visualizando os detalhes do modelo
summary(modelo_v1)

# Fazendo previsões e analisando o resultado
previsoes <- predict(modelo_v1, test.data, type = "response")
previsoes <- round(previsoes)
View(previsoes)

# Confusion Matrix
confusionMatrix(table(data = previsoes, reference = test.class.var), positive = '1')

# Feature Selection
formula <- "credit.rating ~ ."
formula <- as.formula(formula)
control <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
model <- train(formula, data = train.data, method = "glm", trControl = control)
importance <- varImp(model, scale = FALSE)

# Plot
plot(importance)

# Construindo um novo modelo com as variáveis selecionadas
formula.new <- "credit.rating ~ account.balance + credit.purpose + previous.credit.payment.status + savings + credit.duration.months"
formula.new <- as.formula(formula.new)
modelo_v2 <- glm(formula = formula.new, data = train.data, family = "binomial")

# Visualizando o novo modelo
summary(modelo_v2)

# Prevendo e Avaliando o modelo
previsoes_new <- predict(modelo_v2, test.data, type = "response")
previsoes_new <- round(previsoes_new)

# Confusion Matrix
confusionMatrix(table(data = previsoes_new, reference = test.class.var), positive = '1')

# Avaliando a performance do modelo

# Plot do modelo com melhor acurácia
modelo_final <- modelo_v2
previsoes <- predict(modelo_final, test.feature.vars, type = "response")
previsoes_finais <- prediction(previsoes, test.class.var)

# Função para Plot ROC
plot.roc.curve <- function(predictions, title.text){
  perf <- performance(predictions, "tpr", "fpr")
  plot(perf,col = "black",lty = 1, lwd = 2,
    main = title.text, cex.main = 0.6, cex.lab = 0.8,xaxs = "i", yaxs = "i")
  abline(0,1, col = "red")
}

```

```

auc <- performance(predictions,"auc")
auc <- unlist(slot(auc, "y.values"))
auc <- round(auc,2)
legend(0.4,0.4,legend = c(paste0("AUC: ",auc)), cex = 0.6, bty = "n", box.col = "white")

}

# Plot
par(mfrow = c(1, 2))
plot.roc.curve(previsoes_finais, title.text = "Curva ROC")

# Fazendo previsões em novos dados

# Novos dados
account.balance <- c(1, 4, 3)
credit.purpose <- c(4, 2, 3)
previous.credit.payment.status <- c(3, 3, 2)
savings <- c(2, 3, 2)
credit.duration.months <- c(15, 12, 8)

# Cria um dataframe
novo_dataset <- data.frame(account.balance,
                           credit.purpose,
                           previous.credit.payment.status,
                           savings,
                           credit.duration.months)

View(novo_dataset)

# Separa variáveis explanatórias numéricas e categóricas
new.numeric.vars <- c("credit.duration.months")
new.categorical.vars <- c('account.balance', 'previous.credit.payment.status',
                        'credit.purpose', 'savings')

# Aplica as transformações
novo_dataset_final <- to.factors(df = novo_dataset, variables = new.categorical.vars)
str(novo_dataset_final)

novo_dataset_final <- scale.features(novo_dataset_final, new.numeric.vars)
str(novo_dataset_final)

View(novo_dataset_final)

# Previsões
?predict
previsao_novo_cliente <- predict(modelo_final, newdata = novo_dataset_final, type = "response")

```

Quizz

A regressão linear ajuda a prever o valor de uma variável desconhecida (uma variável contínua) com base em um valor conhecido.

Uma variável dependente é o valor que estamos prevendo e uma variável independente é a variável que estamos usando para prever uma variável dependente.

O Método dos Mínimos Quadrados é o método de computação matemática pelo qual se define a reta de regressão. Esse método definirá uma reta que minimizará a soma das distâncias ao quadrado entre os pontos plotados (X, Y) e a reta (que são os valores previstos de Y’).

O erro ao quadrado geral é definido como a soma da diferença ao quadrado entre os valores reais e previstos de todas as observações. A razão pela qual consideramos o valor do erro ao quadrado e não o valor real do erro é que não queremos erro positivo em alguns pontos de dados compensando erros negativos em outros pontos de dados.

A regressão linear múltipla trabalha com uma variável explanatória e uma variável target.

(FALSO!!!)

Modelos de regressão são modelos matemáticos que relacionam o comportamento de uma variável Y com outra X. A variável X é a variável independente da equação enquanto Y é a variável dependente das variações de X.

Caso a variável resposta seja uma variável categórica, ou seja, a variável apresenta como possíveis realizações uma qualidade (ou atributo) e não mais uma mensuração, utilizamos o Modelo de Regressão Linear. **(FALSO!!!)**

Regressão é o processo matemático pelo qual derivamos os parâmetros “a” e “b” de uma função. Estes parâmetros determinam as características da função que relaciona ‘Y’ com ‘X’ e que no caso do modelo linear se representa por uma reta chamada de reta de regressão.

A Soma dos Quadrados dos Resíduos é a variação de Y que não é explicada pelo modelo elaborado.

Se as previsões forem realmente boas, o valor do SSE será baixo.

7.6. Classificação com K-Nearest Neighbours (KNN)

Conhecendo o Algoritmo KNN

- 1- Dados de Treinamento
- 2- Definir a métrica para cálculo da distância
- 3- Definir o valor de K (número de vizinhos mais próximos que serão considerados pelo algoritmo)

Cálculo da distância entre o exemplo desconhecido e o outros exemplos do conjunto de treinamento.

Considera-se o voto majoritário entre os rótulos de classe dos K vizinhos mais próximos.

Distância Euclidiana:

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

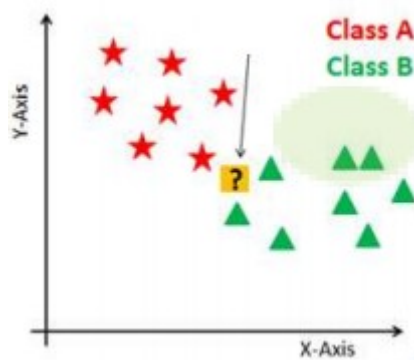
(Fórmula da Distância Euclidiana)

Outras formas de medir a distância:

- Distância Manhattan
- Distância de Minkowsky
- Distância de Hamming

KNN e Estrutura de Células de Voronoi

A pergunta que queremos responder é: a qual dos 2 grupos o ponto amarelo com a interrogação pertence?



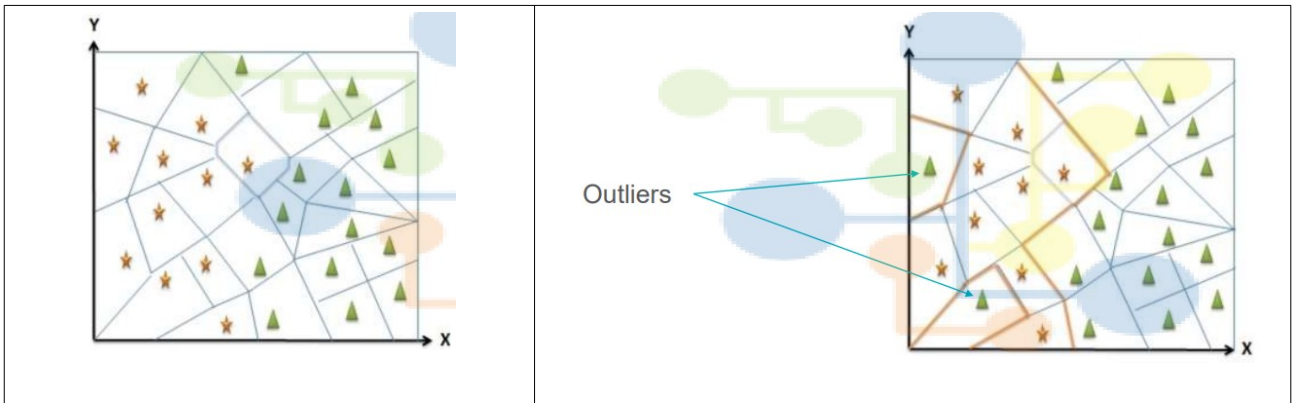
O algoritmo KNN utiliza medidas de distância a fim de comparar os pontos de dados.

Se usamos um valor de K igual a 3, o KNN vai comparar o novo ponto de dado com os 3 vizinhos mais próximos em cada classe e então fazer uma votação.

Célula Voronoi

O tamanho de cada célula é determinado pelo número de exemplos disponíveis no dataset de treino.

Estrutura de célula de Voronoi



Um aspecto interessante dessa separação por células é que existe uma fronteira que forma a separação entre as classes de dados.

Como funciona o KNN:

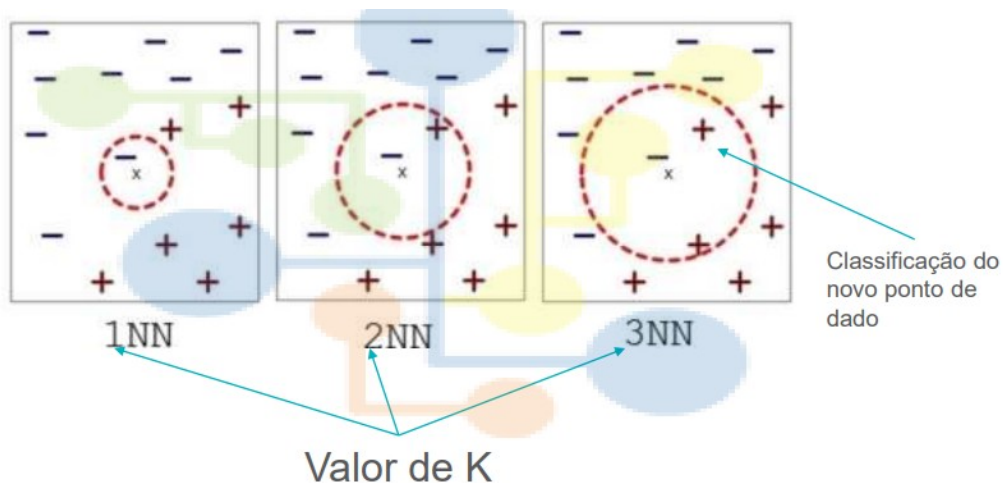
$$\{X_i, Y_i\}$$

X_i : Atributos (variáveis preditoras)

Y_i : Classe (variável target)

Como classificar um novo ponto de dado X :

- 1- A distância é computada entre X e X_i para cada valor de X_i .
- 2- É escolhido o k -vizinho mais próximo X_{in} e sua respectiva classe.
- 3- Retorna-se o valor de y mais frequente na lista $y_{i1}, y_{i2}, \dots, y_{in}$.

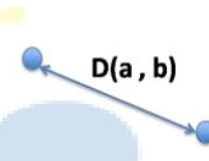


Existem diversas medidas de distância disponíveis e vamos discutir aqui algumas das mais comuns. O principal propósito da medida de distância é identificar os dados que são similares e que não são similares.

Medidas de Distância Matemática

Assim como o valor de K, a medida de distância influencia diretamente a performance de modelos criados com o KNN.

- Distância Euclidiana:

$$D(a, b) = \sqrt{\sum_{i=1}^n (b_i - a_i)^2}$$


- Distância de Hamming:

$$D_H = \sum_{i=1}^k |x_i - y_i|$$
$$x = y \Rightarrow D = 0$$
$$x \neq y \Rightarrow D = 1$$

- Distância de Minkowski:

$$\left(\sum_{i=1}^k (|x_i - y_i|)^q \right)^{1/q}$$

- Distância Manhattan:

$$\sum_{i=1}^k |x_i - y_i|$$

As distâncias Manhattan e Euclidiana são as mais comuns.

A precisão da classificação utilizando o algoritmo KNN depende fortemente do modelo de dados. Na maioria das vezes os atributos precisam ser normalizados para evitar que as medidas de distância sejam dominadas por um único atributo.

O algoritmo KNN é uma variação do algoritmo NN.

KNN Vantagens e Desvantagens

Vantagens do KNN:

- Rápido treinamento
- Capaz de aprender funções complexas

- Não perde/desperdiça informação
- Bastante flexível
- Em alguns casos pode apresentar bons resultados

Desvantagens do KNN:

- Classificar um exemplo desconhecido pode ser um processo computacionalmente complexo, pois requer um cálculo de distância para cada exemplo de treinamento
- Lento para realizar uma consulta
- A precisão da classificação pode ser severamente degradada pela presença de ruído ou características irrelevantes
- Não constrói um modelo de classificação

E Se Der Empate na Votação do KNN?

O KNN é um daqueles algoritmos que são muito simples de entender, mas que funcionam incrivelmente bem na prática. Também é surpreendentemente versátil e suas aplicações variam da detecção de padrões à geometria computacional. Muitos profissionais aprendem o algoritmo e não o utilizam, embora o KNN possa tornar algumas tarefas de construção de modelos preditivos muito simples. E talvez você não saiba, mas o KNN é considerado um dos 10 melhores algoritmos de mineração de dados.

KNN é um algoritmo de aprendizagem preguiçoso (lazy) não paramétrico. Essa é uma definição bastante concisa. Quando você diz que uma técnica é não paramétrica, isso significa que não faz nenhuma suposição sobre a distribuição de dados. Isso é bastante útil, pois no mundo real, a maioria dos dados não obedece às suposições teóricas típicas (por exemplo, misturas gaussianas, linearmente separáveis, etc.).

É também um algoritmo preguiçoso. O que isto significa é que ele não usa os pontos de dados de treinamento para fazer qualquer generalização. Em outras palavras, não há uma fase de treinamento explícita ou, quando há, ela é mínima. Isso permite que a fase de treinamento seja bastante rápida. A falta de generalização significa que o KNN mantém todos os dados de treinamento. Mais exatamente, todos os dados de treinamento são necessários durante a fase de teste. (Bem, isso é um exagero, mas não muito longe da verdade). Isso é um contraste com outras técnicas como SVM onde você pode descartar todos os “não vetores de suporte” sem qualquer problema. A maioria dos algoritmos preguiçosos - especialmente KNN - toma a decisão com base em todo o conjunto de dados de treinamento (ou no melhor dos casos, um subconjunto deles).

A dicotomia é bastante óbvia aqui - Há uma fase de treinamento inexistente ou mínima, mas uma fase de teste bastante intensa em termos de computação. O custo é em termos de tempo e memória. Mais tempo pode ser necessário, quando no pior caso, todos os pontos de dados podem se tornar pontos de decisão. Mais memória é necessária, pois precisamos armazenar todos os dados de treinamento.

O KNN assume que os dados estão em um espaço de características (feature space). Mais exatamente, os pontos de dados estão em um espaço métrico. Os dados podem ser escalares ou possivelmente vetores multidimensionais. Uma vez que os pontos estão no espaço de características, eles têm uma noção de distância - Esta não precisa ser necessariamente distância euclidiana, embora seja o comumente usado.

Cada um dos dados de treinamento consiste em um conjunto de vetores e labels de classe associada a cada vetor. No caso mais simples, será + ou - (para classes positivas ou negativas). Mas KNN, pode trabalhar igualmente bem com número arbitrário de classes.

Também definimos um único número "k". Esse número decide quantos vizinhos (onde os vizinhos são definidos com base na métrica de distância) influenciam a classificação. Este é geralmente um número ímpar se o número de classes é 2. Se $k = 1$, então o algoritmo é simplesmente chamado de algoritmo do vizinho mais próximo. De acordo com o valor definido para k, o KNN fará a comparação com o número k de vizinhos mais próximos. Ocorre normalmente um processo de votação, de modo a garantir que o novo ponto de dado seja classificado de acordo com a maioria de vizinhos similares.

Mas o que acontece quando esta votação dá empate? O que acontece se não houver um vencedor claro na votação da maioria? Por exemplo. Todos os k vizinhos mais próximos são de classes diferentes, ou para $k = 4$ existem 2 vizinhos da classe A e 2 vizinhos da classe B? O que acontece se não for possível determinar exatamente k vizinhos mais próximos por que há mais vizinhos que têm a mesma distância? Por exemplo. Para a lista de distâncias $(x_1; 2)$, $(x_2; 3.5)$, $(x_3; 4.8)$, $(x_4; 4.8)$, $(x_5; 4.8)$, $(x_6; 9.2)$ não seria possível determinar $k = 3$ ou $k = 4$ vizinhos mais próximos, porque os vizinhos x_3 , x_4 e x_5 todos têm mesma distância.

Ao trabalharmos com KNN é preciso compreender que ele não é um algoritmo estritamente matemático, mas um simples classificador / regressor baseado em uma intuição - a função alvo não muda muito quando os argumentos não mudam muito. Ou, em outras palavras, a função alvo é localmente quase constante. Com essa suposição, você pode estimar o valor da função alvo em qualquer ponto, por uma média (possivelmente ponderada) dos valores de k pontos mais próximos.

Tendo isso em mente, você pode perceber que não há uma regra pré-definida sobre o que fazer quando não há um vencedor claro na votação por maioria. Você pode sempre usar um k ímpar ou usar alguma ponderação. Outra opção é reduzir o valor de k em uma unidade, até atingir o valor ideal.

No caso dos vizinhos x_3 , x_4 e x_5 do exemplo acima, que estão à mesma distância do ponto de interesse, você pode usar apenas dois ou usar todos os 5. Novamente, tenha em mente KNN não é um algoritmo derivado de análise matemática complexa, mas apenas uma intuição simples. Cabe a você, Cientista de Dados, decidir como quer lidar com esses casos especiais.

Quando se trata de ponderação, você baseia seu algoritmo na intuição de que a função alvo não muda muito quando os atributos não mudam muito. Podemos dar pesos maiores para pontos que estão mais perto do ponto de interesse. Uma boa ponderação seria, por exemplo:

$$(1 / x - y)^2$$

Ou qualquer outra que seja relativamente grande quando a distância é pequena e relativamente pequena quando a distância entre os pontos é grande (Inversa de alguma função métrica contínua).

A intuição geral é que a função alvo varia de forma diferente em diferentes direções (ou seja, suas diferentes derivadas parciais são de magnitude diferente), portanto, seria sábio, em algum sentido, mudar as métricas / ponderação de acordo com essa intuição. Este truque geralmente melhora o desempenho do KNN, requerendo uma customização do algoritmo implementado nas principais linguagens de programação como R ou Python.

Mas então como encontrar o valor ideal de k , quando houver empate? Você provavelmente precisará testar diferentes grupos de hiperparâmetros e ver quais funcionam bem em algum conjunto de validação. Se você tem recursos computacionais (computador com bastante memória e processador) e deseja chegar aos parâmetros corretos automaticamente em um bom conjunto de hiperparâmetros, há uma ideia recente de usar processos gaussianos para otimização sem derivada nessa configuração.

Encontrar o conjunto de hiperparâmetros (isto é, que minimizam o erro nos dados de validação), pode ser visto como um problema de otimização. Infelizmente, nessa configuração, não podemos obter o gradiente da função que tentamos otimizar (que é o que geralmente queremos fazer, executar a descida de gradiente ou alguns métodos mais avançados). Os processos gaussianos podem ser usados nesta configuração, para encontrar conjuntos de hiperparâmetros, que têm grandes chances, de se comportarem melhor do que os melhores encontrados até o momento. Assim, você pode iterativamente executar o algoritmo com um conjunto de hiperparâmetros, então pergunte ao processo gaussiano para qual deles seria melhor tentar em seguida e assim por diante.

Ou seja, é mais fácil reduzir o valor de k em uma unidade até encontrar o valor ideal!!!

Classificação KNN em Python / Exercício Cap 06

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 6 - Classificação KNN</font>
O KNN é um dos algoritmos mais simples para Machine Learning, sendo um algoritmo do tipo "lazy", ou seja, nenhuma computação é realizada no
dataset até que um novo ponto de dado seja alvo de teste.
#### Classificação KNN em Python - Definindo Um Problema Para Classificação Multiclasse
Faremos previsões de dígitos escritos à mão no dataset mnist. Esse é um exemplo de classificação multiclasse, pois nosso modelo terá que prever
uma entre 10 saídas possíveis para cada registro (dígitos de 0 a 10).
http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\_digits.html
# ->
from IPython.display import Image
Image('imagens/digitos.png')
# ->
from IPython.display import Image
Image('imagens/flatten.png')
# ->
from IPython.display import Image
Image('imagens/frutas.png')
#### Classificação KNN em Python - Carregando e Explorando o Dataset
# ->
# Carrega os pacotes
import numpy as np
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
# ->
# Carrega o dataset
digitos = datasets.load_digits()
# ->
digitos
# ->
# Visualizando algumas imagens e labels
images_e_labels = list(zip(digitos.images, digitos.target))
for index, (image, label) in enumerate(images_e_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap = plt.cm.gray_r, interpolation='nearest')
    plt.title('Label: %i' % label)
# ->
# Gera X e Y
X = digitos.data
```

```

Y = digitos.target
# ->
# Formato de X e Y
print(X.shape, Y.shape)
#### Classificação KNN em Python - Pré-Processamento e Normalização
# ->
# Divisão em dados de treino e de teste
X_treino, testeData, Y_treino, testeLabels = train_test_split(X, Y, test_size = 0.30, random_state = 101)
# ->
# Divisão dos dados de treino em dados de treino e dados de validação
treinoData, validData, treinoLabels, validLabels = train_test_split(X_treino,
                                                                    Y_treino,
                                                                    test_size = 0.1,
                                                                    random_state = 84)
# ->
# Imprimindo o número de exemplos (observações) em cada dataset
print("Exemplos de Treino: {}".format(len(treinoLabels)))
print("Exemplos de Validação: {}".format(len(validLabels)))
print("Exemplos de Teste: {}".format(len(testeLabels)))
# ->
# Normalização dos dados pela Média
# Cálculo da média do dataset de treino
X_norm = np.mean(X, axis = 0)
# Normalização dos dados de treino e de teste
X_treino_norm = treinoData - X_norm
X_valid_norm = validData - X_norm
X_teste_norm = testeData - X_norm
# ->
# Shape dos datasets
print(X_treino_norm.shape, X_valid_norm.shape, X_teste_norm.shape)
#### Classificação KNN em Python - Testando o Melhor Valor de K
# ->
# Range de valores de k que iremos testar
kVals = range(1, 30, 2)
# ->
# Lista vazia para receber as acurácias
acuracias = []
# ->
# Loop em todos os valores de k para testar cada um deles
for k in kVals:
    # Treinando o modelo KNN com cada valor de k
    modeloKNN = KNeighborsClassifier(n_neighbors = k)
    modeloKNN.fit(treinoData, treinoLabels)
    # Avaliando o modelo e atualizando a lista de acurácias
    score = modeloKNN.score(validData, validLabels)
    print("Com valor de k = %d, a acurácia é = %.2f%%" % (k, score * 100))
    acuracias.append(score)
# ->
# Obtendo o valor de k que apresentou a maior acurácia
i = np.argmax(acuracias)
print("O valor de k = %d alcançou a mais alta acurácia de %.2f%% nos dados de validação!" % (kVals[i],
                                                                    acuracias[i] * 100))
#### Classificação KNN em Python - Construção e Treinamento do Modelo KNN
# ->
# Criando a versão final do modelo com o maior valor de k
modeloFinal = KNeighborsClassifier(n_neighbors = kVals[i])
# ->
# Treinamento do modelo
modeloFinal.fit(treinoData, treinoLabels)
#### Classificação KNN em Python - Previsões com Dados de Teste e Avaliação do Modelo
# ->
# Previsões com os dados de teste
predictions = modeloFinal.predict(testeData)
# ->
# Performande do modelo nos dados de teste
print("Avaliação do Modelo nos Dados de Teste")
print(classification_report(testeLabels, predictions))
# ->
# Confusion Matrix do Modelo Final
print("Confusion matrix")
print(confusion_matrix(testeLabels, predictions))
# ->
# Fazendo previsões com o modelo treinado usando dados de teste
for i in np.random.randint(0, high=len(testeLabels), size=(5,)):
    # Obtém uma imagem e faz a previsão
    image = testeData[i]
    prediction = modeloFinal.predict([image])[0]
    # Mostra as previsões

```

```

imgdata = np.array(image, dtype='float')
pixels = imgdata.reshape((8,8))
plt.imshow(pixels,cmap='gray')
plt.annotate(prediction,(3,3),bbox={'facecolor':'white'},fontsize=16)
print("Eu acredito que esse dígito seja: {}".format(prediction))
plt.show()
#### Classificação KNN em Python - Previsões em Novos Dados com o Modelo Treinado
# ->
# Definindo um novo dígito (dados de entrada)
novoDigito = [0., 0., 0., 8., 15., 1., 0., 0., 0., 0., 0., 12., 14.,
              0., 0., 0., 0., 3., 16., 7., 0., 0., 0., 0., 0.,
              6., 16., 2., 0., 0., 0., 0., 0., 7., 16., 16., 13., 5.,
              0., 0., 0., 15., 16., 9., 9., 14., 0., 0., 0., 3., 14.,
              9., 2., 16., 2., 0., 0., 0., 7., 15., 16., 11., 0.]

# ->
# Normalizando o novo dígito
novoDigito_norm = novoDigito - X_norm
# ->
# Fazendo a previsão com o modelo treinado
novaPrevisao = modeloFinal.predict([novoDigito_norm])
# ->
# Previsão do modelo
imgdata = np.array(novoDigito, dtype='float')
pixels = imgdata.reshape((8,8))
plt.imshow(pixels, cmap='gray')
plt.annotate(novaPrevisao,(3,3), bbox={'facecolor':'white'},fontsize=16)
print("Eu acredito que esse dígito seja: {}".format(novaPrevisao))
plt.show()
# ->
# Definindo um novo dígito (dados de entrada)
novoDigito = [0., 0., 0., 1., 11., 9., 0., 0., 0., 0., 0., 7., 16.,
              13., 0., 0., 0., 4., 14., 16., 9., 0., 0., 0., 10.,
              16., 11., 16., 8., 0., 0., 0., 0., 3., 16., 6., 0.,
              0., 0., 0., 0., 3., 16., 8., 0., 0., 0., 0., 5.,
              16., 10., 0., 0., 0., 0., 2., 14., 6., 0., 0.]

# ->
# Normalizando o novo dígito
novoDigito_norm = novoDigito - X_norm
# ->
# Fazendo a previsão com o modelo treinado
novaPrevisao = modeloFinal.predict([novoDigito_norm])
# ->
# Previsão do modelo
imgdata = np.array(novoDigito, dtype='float')
pixels = imgdata.reshape((8,8))
plt.imshow(pixels, cmap='gray')
plt.annotate(novaPrevisao,(3,3), bbox={'facecolor':'white'},fontsize=16)
print("Eu acredito que esse dígito seja: {}".format(novaPrevisao))
plt.show()
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Data Science Academy - Machine Learning

Capítulo 6 - Exercício

Neste exercício você vai praticar suas habilidades de pesquisador, fundamental para quem pretende trabalhar como Cientista de Dados. Seu trabalho será desenvolver o algoritmo KNN usando apenas linguagem Python e Numpy, sem o uso de frameworks (como Scikit-Learn). Você poderá usar o paper abaixo como referência para montar o algoritmo.

Paper de referência do KNN: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.815&rep=rep1&type=pdf>

Seu problema de negócio é a classificação de plantas em 3 categorias. No dataset fornecido, cada planta possui 4 variáveis preditoras representando características da planta e uma variável representando a classe. Seu algoritmo KNN deve prever a classe de uma nova planta uma vez que as 4 características sejam fornecidas.

Para ajudar você, parte do código já está sendo fornecido. Analise e estude o código. Você deve incluir sua solução no espaço indicado por "Escrever solução aqui" nas células abaixo.

A solução será apresentada no capítulo seguinte!

->

Imports

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

->

Carregando o dataset

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'classe']

iris_data = pd.read_csv('dados/iris.data', names = names)

iris_data.head()

->

Separando variáveis preditoras e variável target

X = iris_data.iloc[:,4].values

y = iris_data.iloc[:,4]

```

# Labels da variável target
target_class = pd.get_dummies(iris_data['classe']).columns
target_names = np.array(target_class)
# ->
# Convertendo as classes para valores numéricos correspondentes
y = y.replace(target_names[0], 0)
y = y.replace(target_names[1], 1)
y = y.replace(target_names[2], 2)
y = np.array(y)
# ->
# Separando os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 33)
print(X_train.shape, y_train.shape)
# ->
# Função para calcular a distância euclidiana
def distancia_euclidiana(att1, att2):
    dist = 0
    for i in range(len(att1)):
        dist += pow((att1[i] - att2[i]),2)
    return np.sqrt(dist)
# ->
# Algoritmo KNN
def KNN(array, k):
    # "Escrever solução aqui"
# ->
# Avaliando o modelo
y_test_pred = KNN(X_test, 5)
y_test_prediction = np.asarray(y_test_pred)
# ->
# Calculando a acurácia
acc = y_test - y_test_prediction
err = np.count_nonzero(acc)
acuracia = ((len(y_test) - err) / len(y_test)) * 100
acuracia
# ->
# Fazendo previsões para 5 novas plantas com K igual a 3
previsoes = KNN([[6.7,3.1,4.4,1.4],[4.6,3.2,1.4,0.2],[4.6,3.2,1.4,0.2],[6.4,3.1,5.5,1.8],[6.3,3.2,5.6,1.9]], 3)
previsoes
# ->
# Fazendo previsões para 5 novas plantas com K igual a 5
previsoes = KNN([[6.7,3.1,4.4,1.4],[4.6,3.2,1.4,0.2],[4.6,3.2,1.4,0.2],[6.4,3.1,5.5,1.8],[6.3,3.2,5.6,1.9]], 5)
previsoes
# Fim
### Obrigada - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Classificação KNN em R

```

# Classificação KNN em R

# Classificação KNN em R - Definindo Um Problema Para Classificação Binária

# Prevendo o resultado do índice S&P (The Standard & Poor's 500) do
# American stock market index (NYSE or NASDAQ)

# https://rdrr.io/cran/ISLR/man/Smarket.html

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap06/R")

# Instalando os pacotes
install.packages("ISLR")
install.packages("caret")
install.packages("e1071")

# Carregando os pacotes
library(ISLR)
library(caret)
library(e1071)

# Definindo o seed
set.seed(300)

```

```

# Classificação KNN em R - Carregando e Explorando o Dataset
?Smarket
summary(Smarket)
str(Smarket)
head(Smarket)
View(Smarket)

# Split do dataset em treino e teste
?createDataPartition
indxTrain <- createDataPartition(y = Smarket$Direction, p = 0.75, list = FALSE)
View(indxTrain)
dados_treino <- Smarket[indxTrain,]
dados_teste <- Smarket[-indxTrain,]
class(dados_treino)
class(dados_teste)

# Verificando a distribuição dos dados originais e das partições
prop.table(table(Smarket$Direction)) * 100
prop.table(table(dados_treino$Direction)) * 100

# Correlação entre as variáveis preditoras
descrCor <- cor(dados_treino[,names(dados_treino) != "Direction"])
descrCor

# Classificação KNN em R - Normalização dos Dados (Center e Scale)

# A transformação de "scale" calcula o desvio padrão para um atributo e divide
# cada valor por esse desvio padrão.

# A transformação "center" calcula a média de um atributo e a subtrai de cada valor.

# Função de Normalização
scale.features <- function(df, variables){
  for (variable in variables){
    df[[variable]] <- scale(df[[variable]], center = T, scale = T)
  }
  return(df)
}

# Removendo a variável target dos dados de treino e teste
numeric.vars_treino <- colnames(treinoX <- dados_treino[,names(dados_treino) != "Direction"])
numeric.vars_teste <- colnames(testeX <- dados_teste[,names(dados_teste) != "Direction"])

# Aplicando normalização às variáveis preditoras de treino e teste
dados_treino_scaled <- scale.features(dados_treino, numeric.vars_treino)
dados_teste_scaled <- scale.features(dados_teste, numeric.vars_teste)
View(dados_treino_scaled)
View(dados_teste_scaled)

# Classificação KNN em R - Construção e Treinamento do Modelo
set.seed(400)
?trainControl
?train

# Arquivo de controle
ctrl <- trainControl(method = "repeatedcv", repeats = 3)

# Criação do modelo
knn_v1 <- train(Direction ~ .,
  data = dados_treino_scaled,
  method = "knn",
  trControl = ctrl,
  # preProcess = c("center", "scale"), # Isso é a normalização
  tuneLength = 20)

# Classificação KNN em R - Avaliação do Modelo

# Modelo KNN
knn_v1

# Número de Vizinhos x Acurácia
plot(knn_v1)

# Fazendo previsões
knnPredict <- predict(knn_v1, newdata = dados_teste_scaled)
knnPredict

```



```

# Criando a Confusion Matrix
confusionMatrix(knnPredict, dados_teste$Direction)

# Classificação KNN em R - Aplicando Outras Métricas

# Arquivo de controle
ctrl <- trainControl(method = "repeatedcv",
  repeats = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary)

# Treinamento do modelo
knn_v2 <- train(Direction ~ .,
  data = dados_treino_scaled,
  method = "knn",
  trControl = ctrl,
  metric = "ROC",
  # preProcess = c("center","scale"), # Isso é a normalização
  tuneLength = 20)

# Modelo KNN
knn_v2

# Número de Vizinhos x Acurácia
plot(knn_v2, print.thres = 0.5, type="S")

# Fazendo previsões
knnPredict <- predict(knn_v2, newdata = dados_teste_scaled)

# Criando a Confusion Matrix
confusionMatrix(knnPredict, dados_teste$Direction)

# Previsões com novos dados

# Preparando dados de entrada
Year = c(2006, 2007, 2008)
Lag1 = c(1.30, 0.09, -0.654)
Lag2 = c(1.483, -0.198, 0.589)
Lag3 = c(-0.345, 0.029, 0.690)
Lag4 = c(1.398, 0.104, 1.483)
Lag5 = c(0.214, 0.105, 0.589)
Volume = c(1.36890, 1.09876, 1.231233)
Today = c(0.289, -0.497, 1.649)

novos_dados = data.frame(Year, Lag1, Lag2, Lag3, Lag4, Lag5, Volume, Today)
novos_dados
str(novos_dados)
class(novos_dados)

# Normalizando os dados

# Extraindo os nomes das variáveis
nomes_variaveis <- colnames(novos_dados)
nomes_variaveis

# Aplicando a função
novos_dados_scaled <- scale.features(novos_dados, nomes_variaveis)
novos_dados_scaled
str(novos_dados_scaled)
class(novos_dados_scaled)

# Fazendo previsões
knnPredict <- predict(knn_v2, newdata = novos_dados_scaled)
cat(sprintf("\n Previsão de \"%s\" é \"%s\"\n", novos_dados$Year, knnPredict))

```

Quizz

Todo o processo de classificação será influenciado pelo valor de K, definido por você, Cientista de Dados. A verdade é que o valor de K, além do impacto na performance do modelo KNN, é que vai definir o sucesso ou fracasso de um modelo criado com este algoritmo.

Vantagens do KNN: Rápido treinamento; Flexibilidade; Pode aprender funções complexas; O processo computacional é complexo (Por conta da medida de distância que precisa ser executada para cada classificação, o processo computacional do KNN é complexo).

Se o valor de K for muito alto, o algoritmo KNN pode resultar em um modelo com baixa precisão. Se o valor de K for muito baixo, o modelo será muito sensível a outliers, gerando classificações incorretas. O objetivo é definir um valor de k que gere o modelo mais generalizável possível.

O principal propósito da medida de distância é identificar os dados que são similares e que não são similares. Assim como o valor de K, a medida de distância influencia diretamente a performance de modelos criados com o KNN.

O KNN é um dos poucos algoritmos de Machine Learning que não depende de normalização nos dados. **(FALSO!!!)**

7.7. Classificação com Naive Bayes

Conhecendo o Naive Bayes

Classificação

A classificação consiste no processo de encontrar, através de aprendizado de máquina, um modelo ou função que descreva diferentes classes de dados.

Exemplos:

- Detecção de SPAM
- Organização automática de e-mails
- Identificação de páginas com conteúdo adulto
- Detecção de expressões e sentimentos

O Classificador Naive Bayes é um classificador probabilístico, baseado na aplicação do teorema de Bayes, com hipótese de independência entre os atributos.

Aplicações do Algoritmo Naive Bayes:

- Previsões multi-classes
- Classificação de textos/Filtragem de spam/Análise de sentimento
- Previsões em tempo real (boa performance com cálculos probabilísticos)
- Sistema de Recomendação

São 3 tipos de Naive Bayes no Scikit-Learn:

- Gaussian – considera que os dados estão dentro de uma distribuição normal
- Multinomial – usado para contagem de variáveis discretas
- Bernoulli – também conhecido como binomial, é útil se os vetores de dados são binários (0 e 1)

A Teoria da Probabilidade

Probabilidade é o estudo sobre experimentos que, mesmo realizados em condições bastante parecidas, apresentam resultados que não são possíveis de prever.

Estudamos Probabilidade com a intenção de prever as possibilidades de ocorrência de uma determinada situação ou fato.

Experimento Aleatório

Um experimento é considerado aleatório quando suas ocorrências podem apresentar resultados diferentes. Um exemplo disso acontece ao lançarmos uma moeda que possua faces distintas, sendo uma cara e outra coroa. O resultado desse lançamento é imprevisível, pois não há como saber qual a face que ficará para cima.

Espaço Amostral

O espaço amostral (S) determina as possibilidades possíveis de resultados. No caso do lançamento de uma moeda o conjunto do espaço amostral é dado por: $S = \{\text{cara, coroa}\}$, isso porque são as duas únicas respostas possíveis para esse experimento aleatório.

Evento

Na probabilidade a ocorrência de um fato ou situação é chamado de evento. Sendo assim, ao lançarmos uma moeda estamos estabelecendo a ocorrência do evento. Temos então que, qualquer subconjunto do espaço amostral deve ser considerado um evento. Um exemplo pode acontecer ao lançarmos uma moeda três vezes, e obtermos como resultado do evento o seguinte conjunto:

$$E = \{\text{Cara, Coroa, Cara}\}$$

Razão de Probabilidade

A razão de probabilidade é dada pelas possibilidades de um evento ocorrer levando em consideração o seu espaço amostral. Essa razão, que é uma fração, é igual ao número de elementos do evento (numerador) sobre o número de elementos do espaço amostral (denominador).

Considere os seguintes elementos:

E é um evento.

$n(E)$ é o número de elementos do evento.

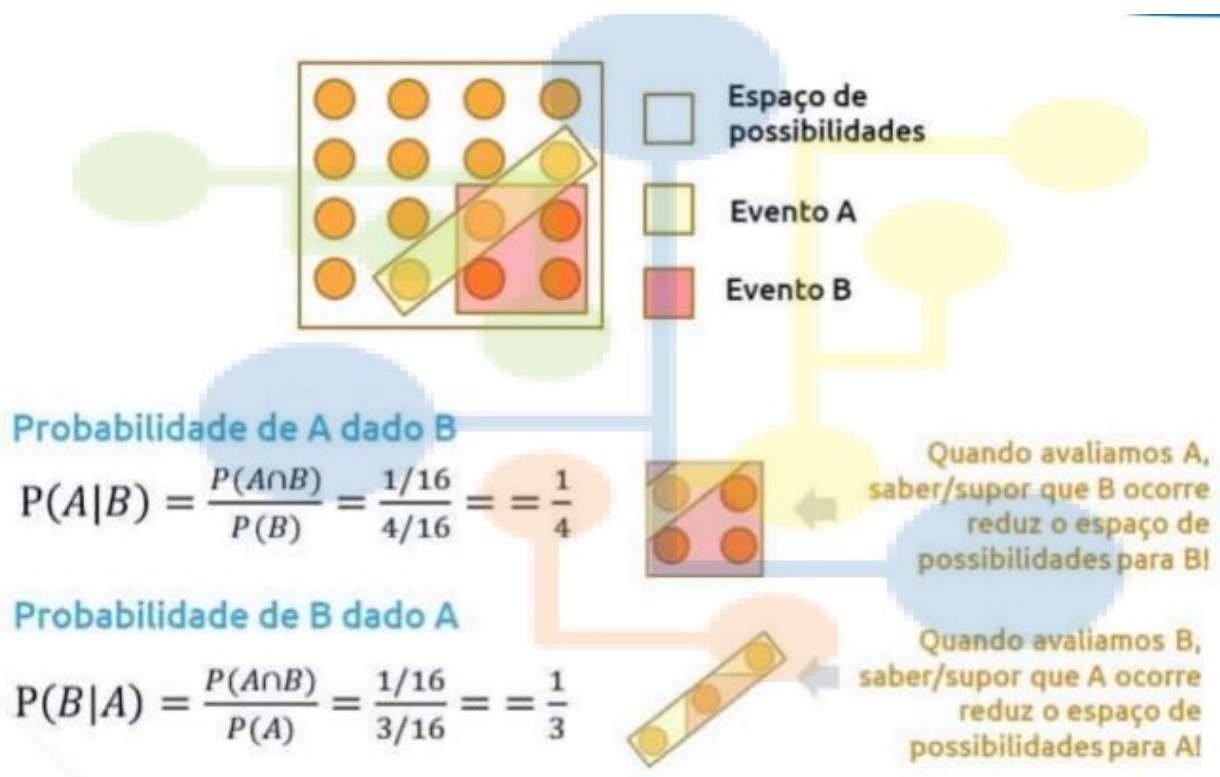
S é espaço amostral.

$n(S)$ é a quantidade de elementos do espaço amostral.

Valor mínimo de probabilidade é zero; Valor máximo é um.

Probabilidade Condicional

Probabilidade de A, dado a ocorrência de B:



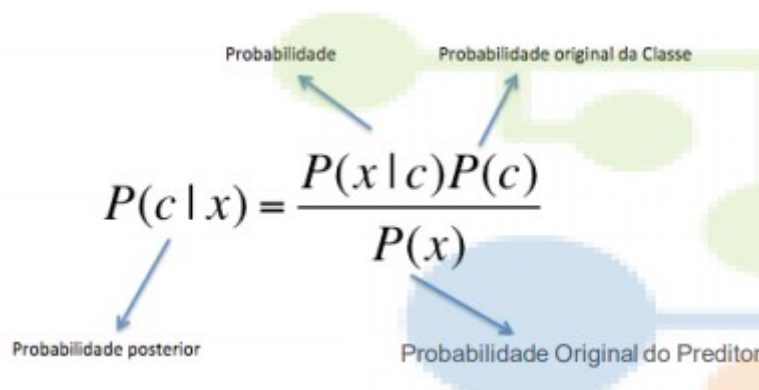
Teorema de Bayes

A regra de Bayes mostra como alterar as probabilidades a priori tendo em conta novas evidências de forma a obter probabilidades a posteriori.

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

$P(x)$ e $P(c)$ são as probabilidades a priori de x e c .

$P(c|x)$ e $P(x|c)$ são as probabilidades a posteriori de c condicional a x e de x condicional a c respectivamente.



O diagrama apresenta a equação $P(c | x) = \frac{P(x | c)P(c)}{P(x)}$ com setas azuis apontando para cada termo e suas respectivas descrições em português:

- Uma seta aponta de $P(c | x)$ para o texto "Probabilidade posterior".
- Uma seta aponta de $P(x | c)$ para o texto "Probabilidade".
- Uma seta aponta de $P(c)$ para o texto "Probabilidade original da Classe".
- Uma seta aponta de $P(x)$ para o texto "Probabilidade Original do Predictor".

- $P(c | x)$ é a probabilidade posterior da classe (c , alvo) dada o predictor (x , atributos).
- $P(c)$ é a probabilidade original da classe.
- $P(x | c)$ é a probabilidade do predictor dada a classe.
- $P(x)$ é a probabilidade original do predictor.

Como funciona o Teorema de Bayes?

Em teoria da probabilidade o Teorema de Bayes mostra a relação entre uma probabilidade condicional e a sua inversa.

Passo 1: Converter o conjunto de dados em uma tabela de frequência (contagem)

Passo 2: Criar tabela de Probabilidade para encontrar as probabilidades de cada ocorrência e de cada combinação.

Passo 3: Usamos a equação do Teorema de Bayes para calcular a probabilidade posterior para cada classe. A classe com maior probabilidade posterior é o resultado da previsão.

Problema da Frequência Zero

- Probabilidade correspondente será zero!
- Probabilidade a posteriori será também zero!

A ideia principal é que a probabilidade de um evento A dado um evento B (i.e. a probabilidade de alguém ter câncer de mama sabendo, ou dado, que a mamografia deu positivo para o teste) depende não apenas do relacionamento entre os eventos A e B (i.e., a precisão, ou exatidão, da mamografia), mas também da probabilidade marginal (ou "probabilidade simples") da ocorrência de cada evento.

Construindo um Classificador de Spam em R

```
# Classificação com Naive Bayes
# Filtrando mensagens de Spam via SMS
# http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning/Cap07/R")

# Pacotes
install.packages("slam")
install.packages("tm")
install.packages("SnowballC")
install.packages("wordcloud")
install.packages("gmodels")
library(tm)
library(SnowballC)
library(wordcloud)
library(e1071)
library(gmodels)

# Carregando os dados
dados <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)

# Examinando a estrutura dos dados
str(dados)

# Convertendo para fator
dados$type <- factor(dados$type)

# Examinando a estrutura dos dados
str(dados$type)
table(dados$type)

# Construindo um Corpus
dados_corpus <- VCorpus(VectorSource(dados$text))

# Examinando a estrutura dos dados
print(dados_corpus)
inspect(dados_corpus[1:2])

# Ajustando a estrutura
as.character(dados_corpus[[1]])
lapply(dados_corpus[1:2], as.character)

# Limpeza do Corpus com tm_map()
?tm_map
dados_corpus_clean <- tm_map(dados_corpus, content_transformer(tolower))

# Diferenças entre o Corpus inicial e o Corpus após a limpeza
as.character(dados_corpus[[1]])
as.character(dados_corpus_clean[[1]])

# Outras etapas de limpeza
dados_corpus_clean <- tm_map(dados_corpus_clean, removeNumbers) # remove números
dados_corpus_clean <- tm_map(dados_corpus_clean, removeWords, stopwords()) # remove stop words
dados_corpus_clean <- tm_map(dados_corpus_clean, removePunctuation) # remove pontuação

# Criando uma função para substituir ao invés de remover pontuação
removePunctuation("hello...world")
replacePunctuation <- function(x) { gsub("[[:punct:]]+", " ", x) }
replacePunctuation("hello...world")

# Word stemming
?wordStem
wordStem(c("learn", "learned", "learning", "learns"))

# Aplicando Stem
dados_corpus_clean <- tm_map(dados_corpus_clean, stemDocument)

# Eliminando espaço em branco
dados_corpus_clean <- tm_map(dados_corpus_clean, stripWhitespaces)

# Examinando a versão final do Corpus
```

```

lapply(dados_corpus[1:3], as.character)
lapply(dados_corpus_clean[1:3], as.character)

# Criando uma matriz esparsa document-term
?DocumentTermMatrix
dados_dtm <- DocumentTermMatrix(dados_corpus_clean)

# Solução alternativa 2 - cria uma matriz esparsa document-term direto a partir do Corpus
dados_dtm2 <- DocumentTermMatrix(dados_corpus, control = list(tolower = TRUE,
                                                             removeNumbers = TRUE,
                                                             stopwords = TRUE,
                                                             removePunctuation = TRUE,
                                                             stemming = TRUE))

# Solução alternativa 3 - usando stop words customizadas a partir da função
dados_dtm3 <- DocumentTermMatrix(dados_corpus, control = list(tolower = TRUE,
                                                             removeNumbers = TRUE,
                                                             stopwords = function(x) { removeWords(x, stopwords()) },
                                                             removePunctuation = TRUE,
                                                             stemming = TRUE))

# Comparando os resultados
dados_dtm
dados_dtm2
dados_dtm3

# Criando datasets de treino e de teste
dados_dtm_train <- dados_dtm[1:4169, ]
dados_dtm_test <- dados_dtm[4170:5559, ]

# Labels (variável target)
dados_train_labels <- dados[1:4169, ]$type
dados_test_labels <- dados[4170:5559, ]$type

# Verificando se a proporção de Spam é similar
prop.table(table(dados_train_labels))
prop.table(table(dados_test_labels))

# Word Cloud
wordcloud(dados_corpus_clean, min.freq = 50, random.order = FALSE)

# Frequência dos dados
sms_dtm_freq_train <- removeSparseTerms(dados_dtm_train, 0.999)
sms_dtm_freq_train

# Indicador de Features para palavras frequentes
findFreqTerms(dados_dtm_train, 5)

# save frequently-appearing terms to a character vector
sms_freq_words <- findFreqTerms(dados_dtm_train, 5)
str(sms_freq_words)

# Criando subsets apenas com palavras mais frequentes
sms_dtm_freq_train <- dados_dtm_train[, sms_freq_words]
sms_dtm_freq_test <- dados_dtm_test[, sms_freq_words]

# Converte para fator
convert_counts <- function(x) {
  print(x)
  x <- ifelse(x > 0, "Yes", "No")
}

# apply() converte counts para colunas de dados de treino e de teste
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)

str(sms_test)

# Treinando o modelo
?naiveBayes
nb_classifier <- naiveBayes(sms_train, dados_train_labels)

# Avaliando o modelo
sms_test_pred <- predict(nb_classifier, sms_test)

# Confusion Matrix
CrossTable(sms_test_pred,
           dados_test_labels,

```

```

prop.chisq = FALSE,
prop.t = FALSE,
prop.r = FALSE,
dnn = c('Previsto', 'Observado'))

# Melhorando a performance do modelo aplicando suavização laplace
nb_classifier_v2 <- naiveBayes(sms_train, dados_train_labels, laplace = 1)

# Avaliando o modelo
sms_test_pred2 <- predict(nb_classifier_v2, sms_test)

# Confusion Matrix
CrossTable(sms_test_pred2,
  dados_test_labels,
  prop.chisq = FALSE,
  prop.t = FALSE,
  prop.r = FALSE,
  dnn = c('Previsto', 'Observado'))

# Nota: Para fazer novas previsões com o modelo treinado, gere uma nova massa de dados
# e use a função predict().

```

Gaussian Naive Bayes em Python / Multinomial Naive Bayes em Python / Bernoulli Naive Bayes em Python / Construindo um Modelo Classificador Naive Bayes em Python

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 7 - Naive Bayes</font>
http://scikit-learn.org/stable/modules/naive\_bayes.html
### Gaussian Naive Bayes (GaussianNB) - Scikit-Learn
Ao lidar com dados contínuos, uma suposição típica é que os valores contínuos associados a cada classe são distribuídos de acordo com uma
distribuição gaussiana (distribuição normal).
http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.GaussianNB.html
http://i.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf
### Gaussian Naive Bayes - Exemplo 1
# ->
# Gaussian Naive Bayes - Exemplo 1
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
# Dataset
iris = datasets.load_iris()
# Classificador
clf = GaussianNB()
# Modelo
modelo = clf.fit(iris.data, iris.target)
# Previsões
y_pred = modelo.predict(iris.data)
# Imprime o resultado
print("Total de Observações: %d - Total de Previsões Incorretas : %d"
      % (iris.data.shape[0], (iris.target != y_pred).sum()))
### Gaussian Naive Bayes - Exemplo 2
# ->
# Gaussian Naive Bayes - Exemplo 2
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
# Dataset
dataset = datasets.load_iris()
# Classificador
clf = GaussianNB()
# Modelo
modelo = clf.fit(dataset.data, dataset.target)
print(modelo)
# Previsões
observado = dataset.target
previsto = modelo.predict(dataset.data)
# Sumário
print(metrics.classification_report(observado, previsto))
print(metrics.confusion_matrix(observado, previsto))
### Gaussian Naive Bayes - Exemplo 3
Machine Learning and Data Mining for Astronomy - http://www.astroml.org/

```


Obs: É necessário instalar o LaTeX de acordo com a versão do seu sistema operacional.

```
# ->
!pip install astroML
# ->
# Gaussian Naive Bayes - Exemplo 3
import numpy as np
from sklearn.naive_bayes import GaussianNB
from astroML.plotting import setup_text_plots
import matplotlib.pyplot as plt
from matplotlib import colors
%matplotlib inline
setup_text_plots(fontsize = 8, usetex = True)
# Criando massa de dados
np.random.seed(0)
mu1 = [1, 1]
cov1 = 0.3 * np.eye(2)
mu2 = [5, 3]
cov2 = np.eye(2) * np.array([0.4, 0.1])
# Concatenando
X = np.concatenate([np.random.multivariate_normal(mu1, cov1, 100),
                    np.random.multivariate_normal(mu2, cov2, 100)])
y = np.zeros(200)
y[100:] = 1
# Criação do Modelo
clf = GaussianNB()
clf.fit(X, y)
# Previsões
xlim = (-1, 8)
ylim = (-1, 5)
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 71), np.linspace(ylim[0], ylim[1], 81))
Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:, 1].reshape(xx.shape)
# Plot dos resultados
fig = plt.figure(figsize = (5, 3.75))
ax = fig.add_subplot(111)
ax.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.binary, zorder = 2)
ax.contour(xx, yy, Z, [0.5], colors = 'k')
ax.set_xlim(xlim)
ax.set_ylim(ylim)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
plt.show()
### Gaussian Naive Bayes - Exemplo 4
# ->
# Gaussian Naive Bayes - Exemplo 4
import numpy as np
from random import random
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
import pylab as pl
import matplotlib
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
%matplotlib inline
# ->
# Massa de dados representando 3 classes
leopardo_features = [(random() * 5 + 8, random() * 7 + 12) for x in range(5)]
urso_features = [(random() * 4 + 3, random() * 2 + 30) for x in range(4)]
elefante_features = [(random() * 3 + 20, (random() - 0.5) * 4 + 23) for x in range(6)]
# X
x = urso_features + elefante_features + leopardo_features
# Y
y = ['urso'] * len(urso_features) + ['elefante'] * len(elefante_features) + ['leopardo'] * len(leopardo_features)
# ->
# Plot dos dados
fig, axis = plt.subplots(1, 1)
# Classe 1
urso_weight, urso_height = zip(*urso_features)
axis.plot(urso_weight, urso_height, 'ro', label = 'Ursos')
# Classe 2
elefante_weight, elefante_height = zip(*elefante_features)
axis.plot(elefante_weight, elefante_height, 'bo', label = 'Elefantes')
# Classe 3
leopardo_weight, leopardo_height = zip(*leopardo_features)
axis.plot(leopardo_weight, leopardo_height, 'yo', label = 'Leopardos')
# Eixos
axis.legend(loc = 4)
```

```

axis.set_xlabel('Peso')
axis.set_ylabel('Altura')
# Plot
plt.show()
# ->
# Criando o Modelo com os dados de treino
clf = GaussianNB()
clf.fit(x, y)
# Criando dados de teste
new_xses = [[2, 3], [3, 31], [21, 23], [12, 16]]
# Previsões
print (clf.predict(new_xses))
print (clf.predict_proba(new_xses))
# ->
def plot_classification_results(clf, X, y, title):
    # Divide o dataset em treino e teste
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
    # Fit dos dados com o classificador
    clf.fit(X_train, y_train)
    # Cores para o gráfico
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
    h = .02 # step size in the mesh
    # Plot da fronteira de decisão.
    # Usando o meshgrid do NumPy e atribuindo uma cor para cada ponto
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Previsões
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    # Resultados em cada cor do plot
    Z = Z.reshape(xx.shape)
    pl.figure()
    pl.pcolormesh(xx, yy, Z, cmap=cmap_light)
    # Plot dos pontos de dados de treino
    pl.scatter(X_train[:, 0], X_train[:, 1], c = y_train, cmap = cmap_bold)
    y_predicted = clf.predict(X_test)
    score = clf.score(X_test, y_test)
    pl.scatter(X_test[:, 0], X_test[:, 1], c = y_predicted, alpha = 0.5, cmap = cmap_bold)
    pl.xlim(xx.min(), xx.max())
    pl.ylim(yy.min(), yy.max())
    pl.title(title)
    return score
# ->
xs = np.array(x)
ys = [0] * len(urso_features) + [1] * len(elefante_features) + [2] * len(leopardo_features)
# ->
score = plot_classification_results(clf, xs, ys, "Classificação Multiclasse")
print ("Acurácia de Classificação: %s" % score)
### Multiclassificação com Naive Bayes
# ->
#Gaussian Naive Bayes - Exemplo 5
import numpy as np
from sklearn import datasets
# Dataset
iris = datasets.load_iris()
# Imprimindo as 3 classes do dataset
print(np.unique(iris.target))
# Split dos dados
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.4)
# Shape
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
# Classificador
clf = GaussianNB()
# Resultado
plot_classification_results(clf, X_train[:, :2], y_train, "Classificação Multiclasse")
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 7 - Naive Bayes</font>
http://scikit-learn.org/stable/modules/naive_bayes.html
## Multinomial Naive Bayes - Scikit-Learn
O classificador Multinomial Naive Bayes é adequado para classificação com variáveis discretas (por exemplo, contagens de palavras para a classificação de texto). A distribuição multinomial normalmente requer contagens de entidades inteiras. No entanto, na prática, contagens fracionadas como tf-idf também podem funcionar.
http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

```

```

### Classificador de Notícias
http://qwone.com/~jason/20Newsgroups/
# ->
# Imports
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
# ->
# Definindo as categorias
# (usando apenas 4 de um total de 20 disponível para que o processo de classificação seja mais rápido)
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
# ->
# Treinamento
twenty_train = fetch_20newsgroups(subset = 'train', categories = categories, shuffle = True, random_state = 42)
# ->
# Classes
twenty_train.target_names
# ->
len(twenty_train.data)
# ->
# Visualizando alguns dados (atributos)
print("\n".join(twenty_train.data[0].split("\n")[:3]))
# ->
# Visualizando variáveis target
print(twenty_train.target_names[twenty_train.target[0]])
# ->
# O Scikit-Learn registra os labels como array de números, a fim de aumentar a velocidade
twenty_train.target[:10]
# ->
# Visualizando as classes dos 10 primeiros registros
for t in twenty_train.target[:10]:
    print(twenty_train.target_names[t])
### Bag of Words (Saco de Palavras)
# ->
# Tokenizing
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
count_vect.vocabulary_.get(u'algorith')
X_train_counts.shape
# ->
# De ocorrências a frequências - Term Frequency times Inverse Document Frequency (Tfidf)
tf_transformer = TfidfTransformer(use_idf = False).fit(X_train_counts)
X_train_tf = tf_transformer.transform(X_train_counts)
X_train_tf.shape
# ->
# Mesmo resultado da célula anterior, mas combinando as funções
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape
# ->
# Criando o modelo Multinomial
clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)
# ->
# Previsões
docs_new = ['God is love', 'OpenGL on the GPU is fast']
X_new_counts = count_vect.transform(docs_new)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)
predicted = clf.predict(X_new_tfidf)
for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, twenty_train.target_names[category]))
# ->
# Criando um Pipeline - Classificador Composto
# vectorizer => transformer => classifier
text_clf = Pipeline([('vect', CountVectorizer()),
                    ('tfidf', TfidfTransformer()),
                    ('clf', MultinomialNB()),
                    ])
# ->
# Fit
text_clf = text_clf.fit(twenty_train.data, twenty_train.target)
# ->

```

```

# Acurácia do Modelo
twenty_test = fetch_20newsgroups(subset = 'test', categories = categories, shuffle = True, random_state = 42)
docs_test = twenty_test.data
predicted = text_clf.predict(docs_test)
np.mean(predicted == twenty_test.target)
# ->

# Métricas
print(metrics.classification_report(twenty_test.target, predicted, target_names = twenty_test.target_names))
# ->

# Confusion Matrix
metrics.confusion_matrix(twenty_test.target, predicted)
# ->

# Parâmetros para o GridSearchCV
parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
              'tfidf__use_idf': (True, False),
              'clf__alpha': (1e-2, 1e-3),
              }
# ->

# GridSearchCV
gs_clf = GridSearchCV(text_clf, parameters, n_jobs = -1)
# ->

# Fit
gs_clf = gs_clf.fit(twenty_train.data[:400], twenty_train.target[:400])
# ->

# Teste
twenty_train.target_names[gs_clf.predict(['God is love'])[0]]
# ->

# Score
gs_clf.best_score_
# ->

# Parâmetros utilizados
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs_clf.best_params_[param_name]))
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 7 - Naive Bayes</font>
http://scikit-learn.org/stable/modules/naive_bayes.html
## Bernoulli Naive Bayes - Scikit-Learn
Assim como MultinomialNB, o classificador BernoulliNB é adequado para dados discretos. A diferença é que enquanto MultinomialNB trabalha com contagens de ocorrência o BernoulliNB é projetado para recursos binários / booleanos.
http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html
O RandomTreesEmbedding fornece uma maneira de mapear dados para uma representação de dados altamente dimensionais e esparsos, o que pode ser benéfico para problemas de classificação. O mapeamento é completamente sem supervisão e muito eficiente.
Este exemplo visualiza as partições dadas por várias árvores e mostra como a transformação também pode ser usada para redução de dimensionalidade não-linear ou classificação não-linear.
Pontos que são vizinhos muitas vezes compartilham a mesma folha de uma árvore e, portanto, compartilham grandes partes de sua representação hash. Isto permite separar dois círculos concêntricos simplesmente com base nos componentes principais dos dados transformados com SVD (Singular Value Decomposition).
Em espaços de alta dimensão, os classificadores lineares conseguem frequentemente uma excelente precisão. Para dados binários esparsos, o BernoulliNB é particularmente adequado. Os gráficos abaixo comparam o limite de decisão obtido pelo modelo BernoulliNB no espaço transformado com as florestas ExtraTreesClassifier aprendidas nos dados originais.
# ->
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.ensemble import RandomTreesEmbedding, ExtraTreesClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.naive_bayes import BernoulliNB
%matplotlib inline
# Gerando uma massa de dados
X, y = make_circles(factor = 0.5, random_state = 0, noise = 0.05)
# Usando o RandomTreesEmbedding para transformar dados
hasher = RandomTreesEmbedding(n_estimators = 10, random_state = 0, max_depth = 3)
X_transformed = hasher.fit_transform(X)
# Visualizando o resultado após a redução da dimensionalidade com SVD
svd = TruncatedSVD(n_components = 2)
X_reduced = svd.fit_transform(X_transformed)
# Criando um Classificador Naive Bayes e aplicando aos dados transformados
nb = BernoulliNB()
nb.fit(X_transformed, y)
# Criando um Classificador ExtraTreesClassifier para comparação
trees = ExtraTreesClassifier(max_depth = 3, n_estimators = 10, random_state = 0)
trees.fit(X, y)
# Plot
fig = plt.figure(figsize=(9, 8))
ax = plt.subplot(221)

```

```

ax.scatter(X[:, 0], X[:, 1], c=y, s=50)
ax.set_title("Variáveis Target Originais (2d)")
ax.set_xticks()
ax.set_yticks()
ax = plt.subplot(222)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, s=50)
ax.set_title("Dados reduzidos com SVD (2d) a partir dos dados transformados (%dd)" % X_transformed.shape[1])
ax.set_xticks()
ax.set_yticks()
# Plotando o espaço de decisão original e aplicando as cores no MeshGrid
h = .01
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
# Transforma o Grid usando RandomTreesEmbedding
transformed_grid = hasher.transform(np.c_[xx.ravel(), yy.ravel()])
y_grid_pred = nb.predict_proba(transformed_grid)[:, 1]
ax = plt.subplot(223)
ax.set_title("Naive Bayes nos Dados Transformados")
ax.pcolormesh(xx, yy, y_grid_pred.reshape(xx.shape))
ax.scatter(X[:, 0], X[:, 1], c=y, s=50)
ax.set_ylim(-1.4, 1.4)
ax.set_xlim(-1.4, 1.4)
ax.set_xticks()
ax.set_yticks()
# Transformando o Grid usando ExtraTreesClassifier
y_grid_pred = trees.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
ax = plt.subplot(224)
ax.set_title("ExtraTrees Predictions")
ax.pcolormesh(xx, yy, y_grid_pred.reshape(xx.shape))
ax.scatter(X[:, 0], X[:, 1], c=y, s=50)
ax.set_ylim(-1.4, 1.4)
ax.set_xlim(-1.4, 1.4)
ax.set_xticks()
ax.set_yticks()
plt.tight_layout()
plt.show()
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 7 - Naive Bayes</font>
### Construindo um Classificador Naive Bayes em Python
Estamos construindo um classificador Naive Bayes com BernoulliNB e MultinomialNB em Python. Não usaremos as funções do Scikit-learn.
# ->
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.datasets import fetch_20newsgroups
import logging
import sys
from time import time
from math import *
from matplotlib import pyplot as pl
from matplotlib.backends.backend_pdf import PdfPages
# ->
class MyBernClassifier():
    def __init__(self, smooth = 1):
        self._smooth = smooth
        self._feat_prob = []
        self._class_prob = []
        self._Ncls = []
        self._Nfeat = []
    def train(self, X, y):
        print ("Treinando Bernoulli NB...")
        count_each_class = {}
        feature_count = {}
        alpha = self._smooth
        temp = []
        temp.append(np.unique(y))
        self._Ncls.append(temp[0].size) # Número total de classes
        self._Nfeat.append(X[0].size) # Número total de features
        for i in range(y.size):
            if y[i] in feature_count:
                continue
            else:
                feature_count[y[i]] = [0 for w in range (X[i].size)]
        # Conta os atributos para cada classe através do treinamento ou
        # conta a ocorrência de cada classe através do treinamento

```

```

for i in range (y.size):
    if y[i] in count_each_class:
        count_each_class[y[i]] +=1
    else:
        count_each_class[y[i]] = 1
    for j in range(X[i].size):
        feature_count[y[i]][j] += X[i][j]
# Calcula probabilidades de classe e atributos para cada classe
for cls in feature_count:
    num = (self._smooth+count_each_class[cls])
    din = (y.size+(self._Ncls[0]*self._smooth))
    self._class_prob.append((num/float(din)))
    ar = np.array([])
    for j in range(X[i].size):
        num = (feature_count[cls][j] + self._smooth)
        din = (count_each_class[cls]+(2*self._smooth))
        ar = np.append(ar,(num/float(din)))
    self._feat_prob.append(ar)
def predict(self, X):
    print ("Fazendo Previsões com Bernoulli NB...")
    Y_predict = np.array([])
    for i in X:
        neg_log_prob = 0
        minimum_neg_log_prob = 9999999999999999
        category = 0
        for cls in range(self._Ncls[0]):
            neg_log_prob = -log(self._class_prob[cls])
            for j in range(self._Nfeat[0]):
                if (i[j])!=0:
                    neg_log_prob -= log(1-self._feat_prob[cls][j])
                else:
                    neg_log_prob -= log(self._feat_prob[cls][j])
            if minimum_neg_log_prob>neg_log_prob:
                category=cls
                minimum_neg_log_prob=neg_log_prob
        Y_predict=np.append(Y_predict,category)
    return Y_predict
# ->
class MyMultinomialBayesClassifier():
    def __init__(self, smooth = 1):
        self._smooth = smooth
        self._feat_prob = []
        self._class_prob = []
        self._class_neg_prob = []
        self._Ncls = []
        self._Nfeat = []
    def train(self, X, y):
        print ("Treinando Multinomial NB...")
        count_each_class = {}
        feature_count = {}
        for i in range(y.size):
            if y[i] in feature_count:
                continue
            else:
                feature_count[y[i]] = [0 for w in range (X[i].size)]
        for i in range (y.size):
            if y[i] in count_each_class:
                count_each_class[y[i]] +=1
            else:
                count_each_class[y[i]] = 1
            for j in range(X[i].size):
                feature_count[y[i]][j] += X[i][j]
        alpha = self._smooth
        temp = []
        temp.append(np.unique(y))
        self._Ncls.append(temp[0].size)
        self._Nfeat.append(X[0].size)
        self._class_prob.append(count_each_class)
        self._feat_prob.append(feature_count)
    def predict(self, X):
        print ("Fazendo Previsões com Multinomial NB...")
        Y_predict = np.array([])
        # Calcula o total de classes para os dados de treino
        total_train_count = 0
        for key in self._class_prob[0]:
            total_train_count += self._class_prob[0][key]
        for i in X:
            neg_log_prob = 0

```

```

minimum_neg_log_prob=999999999999999
category = 0
for cls in self._feat_prob[0]:
    Ny = sum(self._feat_prob[0][cls])
    neg_log_prob = -log((self._class_prob[0][cls]+1)/float(total_train_count+(self._Ncls[0]*self._smooth)))
    for j in range(self._Nfeat[0]):
        if (i[j])==0:
            continue
        for itere in range (i[j]):
            num = (self._smooth+self._feat_prob[0][cls][j])
            din = (Ny+(self._Nfeat[0]*self._smooth))
            neg_log_prob -= log(num/float(din))
    if minimum_neg_log_prob>neg_log_prob:
        category=cls
        minimum_neg_log_prob=neg_log_prob
    Y_predict=np.append(Y_predict,category)
return Y_predict
# ->
# Define as classes que serão usadas no processo de classificação
categories = [
    'alt.atheism',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space',
]
remove = ('headers', 'footers', 'quotes')
# ->
# Carrega os dados
data_train = fetch_20newsgroups(subset = 'train', categories = categories, shuffle = True, random_state = 42, remove = remove)
data_test = fetch_20newsgroups(subset = 'test', categories = categories, shuffle = True, random_state = 42, remove = remove)
print('Dados Carregados!')
# ->
# Treino e Teste
y_train, y_test = data_train.target, data_test.target
# ->
print("Extraíndo as features do dataset de treino usando o count vectorizer")
t0 = time()
# ->
# Binary = true for Bernoulli NB
vectorizer = CountVectorizer(stop_words = 'english', binary = True)
X_train = vectorizer.fit_transform(data_train.data).toarray()
X_test = vectorizer.transform(data_test.data).toarray()
feature_names = vectorizer.get_feature_names()
# ->
# For Bernoulli NB, Binary = true, train for one default smooth value alpha = 1
print ('-----')
print ('Tempo esperado para execução do modelo Bernoulli NB é 180 seg')
ta = time()
alpha = 1
clf = MyBernClassifier(alpha)
clf.train(X_train,y_train)
y_pred = clf.predict(X_test)
tb = time()
print ("Para o modelo Bernoulli NB: " + 'alpha = %f, accuracy = %f' %(alpha, np.mean((y_test - y_pred)==0)))
print ("Tempo total para treinar e prever com o modelo Bernoulli: " + str(tb-ta))
print ('-----')
### Esta célula pode levar horas para ser executada!
# ->
# Bernoulli Naive bayes: Alpha Vs accuracy
acc = []
alp = []
for alpha in [float(j) / 100 for j in range(1, 101, 1)]:
    print ('-----')
    ta = time()
    clf = MyBernClassifier(alpha)
    clf.train(X_train,y_train)
    y_pred = clf.predict(X_test)
    acc.append(np.mean((y_test-y_pred)==0))
    alp.append(alpha)
    tb = time()
    print ("Tempo de treinamento: " + str(tb-ta) + " acurácia, alpha is: " + str(np.mean((y_test-y_pred)==0)) + "," + str(alpha))
# Plotting
with PdfPages('Bernoulli.pdf') as pdf:
    pl.plot(alp,acc,marker='.', linestyle = '-', color = 'r')
    pl.ylabel('Acurácia',color='g')
    pl.xlabel('Alpha',color='g')
    pl.title('Plot Alpha Vs Acurácia para Bernoulli NB',color = 'r')
    pdf.savefig()

```

```

    pl.close()
# Print
print ("Acurácia máxima do modelo Bernoulli NB is: " + str(max(acc)))
print ("com valor correspondente para alpha de:      " + str(alp[(acc.index(max(acc)))]))
# ->
# Binary = false para Multinomial NB
print ("Extraindo dados com Binary = False para Multinomial NB")
vectorizer = CountVectorizer(stop_words = 'english', binary = False)
X_train = vectorizer.fit_transform(data_train.data).toarray()
X_test = vectorizer.transform(data_test.data).toarray()
feature_names = vectorizer.get_feature_names()
print ("Tempo total esperado para execução do Multinomial NB é 90 seg")
ta = time()
alpha = 1
clf1 = MyMultinomialBayesClassifier(alpha)
clf1.train(X_train,y_train)
y_pred = clf1.predict(X_test)
print ("Para modelo Multinomial NB:  " + 'alpha = %f accuracy = %f' %(alpha, np.mean((y_test-y_pred)==0)))
tb = time()
print ("Tempo total para treinar e prever o modelo Multinomial: " + str(tb-ta))
print ('-----')
### Esta célula pode levar horas para ser executada!
# ->
# Multinomial Naive bayes: Alpha Vs accuracy
acc = []
alp = []
for alpha in [float(j) / 100 for j in range(1, 101, 1)]:
    print ('-----')
    ta = time()
    clf1 = MyMultinomialBayesClassifier(alpha)
    clf1.train(X_train,y_train)
    y_pred1 = clf1.predict(X_test)
    acc.append(np.mean((y_test-y_pred1)==0))
    alp.append(alpha)
    tb = time()
    print ("Tempo de Treinamento: " + str(tb-ta) + " acurácia, alpha é: " + str(np.mean((y_test-y_pred1)==0)) + "," + str(alpha))
    #print ('alpha=%f accuracy = %f' %(alpha, np.mean((y_test-y_pred1)==0)))
# Plotting
with PdfPages('multinomial.pdf') as pdf:
    pl.plot(alp,acc,marker = '.', linestyle = '-', color = 'r')
    pl.ylabel('Acurácia',color = 'g')
    pl.xlabel('Alpha',color = 'g')
    pl.title('Plot Alpha Vs Acurácia para Multinomial NB',color = 'r')
    pdf.savefig()

    pl.close()
# Max Accuracy and Corresponding alpha.
print ("Acurácia máxima para o modelo Multinomial NB is: " + str(max(acc)))
print ("com valor correspondente alpha de:      " + str(alp[(acc.index(max(acc)))]))
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Solução de Exercício

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Solução Exercício do Capítulo 6</font>
Neste exercício você vai praticar suas habilidades de pesquisador, fundamental para quem pretende trabalhar como Cientista de Dados. Seu trabalho será desenvolver o algoritmo KNN usando apenas linguagem Python e Numpy, sem o uso de frameworks (como Scikit-Learn). Você poderá usar o paper abaixo como referência para montar o algoritmo.
Paper de referência do KNN: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.815&rep=rep1&type=pdf
Seu problema de negócio é a classificação de plantas em 3 categorias. No dataset fornecido, cada planta possui 4 variáveis preditoras representando características da planta e uma variável representando a classe. Seu algoritmo KNN deve prever a classe de uma nova planta uma vez que as 4 características sejam fornecidas.
Para ajudar você, parte do código já está sendo fornecido. Analise e estude o código. Você deve incluir sua solução no espaço indicado por "Escrever solução aqui" nas células abaixo.
A solução será apresentada no capítulo seguinte!
# ->
# Imports
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# ->
# Carregando o dataset
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'classe']
iris_data = pd.read_csv('dados/iris.data', names = names)

```



```

iris_data.head()
# ->
# Separando variáveis preditoras e variável target
X = iris_data.iloc[:,4].values
y = iris_data.iloc[:,4]
# Labels da variável target
target_class = pd.get_dummies(iris_data['classe']).columns
target_names = np.array(target_class)
# ->
# Convertendo as classes para valores numéricos correspondentes
y = y.replace(target_names[0], 0)
y = y.replace(target_names[1], 1)
y = y.replace(target_names[2], 2)
y = np.array(y)
# ->
# Separando os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 33)
print(X_train.shape, y_train.shape)
# ->
# Função para calcular a distância euclidiana
def distancia_euclidiana(att1, att2):
    dist = 0
    for i in range(len(att1)):
        dist += pow((att1[i] - att2[i]),2)
    return np.sqrt(dist)
# ->
# Algoritmo KNN
def KNN(array, k):
    # Array para o resultado final
    resultado = []
    # Loop por todos os elementos do array recebido como entrada
    for i in range(len(array)):
        valor = array[i]
        # Votação
        def vote(item):
            val = []
            for i in range(len(knn)):
                temp = item[i][1]
                val.append(temp)
            class_val = max(set(val), key = val.count)
            return class_val
        # Aplicando a função de distância aos dados
        distance = []
        for j in range(len(X_train)) :
            # Calcula a distância de cada ponto de dado de entrada (array) para cada ponto de dado de treino
            euclidean_distance = distancia_euclidiana(valor, X_train[j])
            # Cria uma lista contendo a distância calculada e o valor do label do dado de treino em j
            temp = [euclidean_distance, y_train[j]]
            # Adiciona o item anterior à lista de distâncias
            distance.append(temp)
        # Ordena
        distance.sort()
        # Obtém o valor de k para os vizinhos mais próximos
        knn = distance[:k]
        # Faz a votação
        resultado.append(vote(knn))
    return resultado
# ->
# Avaliando o modelo
y_test_pred = KNN(X_test, 5)
y_test_prediction = np.asarray(y_test_pred)
# ->
# Calculando a acurácia
acc = y_test - y_test_prediction
err = np.count_nonzero(acc)
acuracia = ((len(y_test) - err) / len(y_test)) * 100
acuracia
# ->
# Fazendo previsões para 5 novas plantas com K igual a 3
previsoes = KNN([[6.7,3.1,4.4,1.4],[4.6,3.2,1.4,0.2],[4.6,3.2,1.4,0.2],[6.4,3.1,5.5,1.8],[6.3,3.2,5.6,1.9]], 3)
previsoes
# ->
# Fazendo previsões para 5 novas plantas com K igual a 5
previsoes = KNN([[6.7,3.1,4.4,1.4],[4.6,3.2,1.4,0.2],[4.6,3.2,1.4,0.2],[6.4,3.1,5.5,1.8],[6.3,3.2,5.6,1.9]], 5)
previsoes
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Melhorando o Poder de Classificação do Modelo Naive Bayes

Se os atributos contínuos não têm distribuição normal, devemos usar a transformação ou métodos diferentes para convertê-los em distribuição normal.

Remova variáveis correlacionadas. Os atributos altamente correlacionados podem levar a um excesso de importância de uma característica, reduzindo a capacidade de generalização do modelo.

Classificadores Naive Bayes têm opções limitadas para ajuste de parâmetros, tais como como alfa = 1 para suavização, `fit_prior = [Verdade | Falso]` para aprendizagem a partir de probabilidades anteriores. Nós recomendamos focar no pré-processamento de dados e seleção de atributos.

Você pode querer aplicar alguma técnica ensemble como “bagging” e “boosting”, mas na prática esses métodos não ajudariam, pois a finalidade destes métodos é reduzir a variância. Naive Bayes não tem variância para minimizar.

Vantagens e Desvantagens de Modelos Naive Bayes

Vantagens de Modelos Naive Bayes:

- É fácil e rápido para prever o conjunto de dados da classe de teste. Também tem um bom desempenho na previsão de classes múltiplas.
- Quando a suposição de independência prevalece, um classificador Naive Bayes tem melhor desempenho em comparação com outros modelos como regressão logística, e você precisa de menos dados de treinamento.
- O desempenho é bom em caso de variáveis categóricas de entrada em comparação a variáveis numéricas. Para variáveis numéricas, assume-se a distribuição normal (curva de sino, que é uma suposição forte).

Desvantagens de Modelos Naive Bayes:

- Se a variável categórica tem uma categoria (no conjunto de dados de teste) que não foi observada no conjunto de dados de treinamento, então o modelo irá atribuir uma probabilidade de 0 (zero) e não será capaz de fazer uma previsão. Isso é muitas vezes conhecido como “Zero Frequency”. Para resolver esse problema, podemos usar a técnica de “suavização” (smoothing). Uma das técnicas mais simples de “suavização” (smoothing) é a chamada estimativa de Laplace.
- Uma limitação do Naive Bayes é a suposição de preditores independentes. Na vida real, é quase impossível ter um conjunto de indicadores que sejam completamente independentes.

Quizz

A classificação consiste no processo de encontrar, através de aprendizado de máquina, um modelo ou função que descreva diferentes classes de dados.

Um classificador Naive Bayes é um classificador probabilístico simples baseado na aplicação do teorema de Bayes com hipóteses de independência entre seus atributos. Em termos simples, um classificador Naive Bayes assume que a presença ou ausência de uma característica particular, não está relacionado com a presença ou ausência de qualquer outro elemento, tendo em conta a variável target, ou seja, a classe.

Problema da Frequência Zero é quando um determinado valor de atributo não aparece na base de treinamento, mas aparece no exemplo de teste.

Em modelos criados com Naive Bayes, o desempenho é bom em caso de variáveis categóricas de entrada em comparação com variáveis numéricas. Para variáveis numéricas, assume-se a distribuição normal (curva de sino, que é uma suposição forte).

Uma limitação do Naive Bayes é a suposição de preditores independentes. Na vida real, é quase impossível ter um conjunto de indicadores que sejam completamente independentes.

7.8. Decision Tree, Random Forest e Métodos Ensemble

O Que São Árvores de Decisão?

Podemos usar o RandomForest para seleção de atributos, ou seja, podemos usar árvores de decisão não apenas para modelos de ML em si, mas também para aplicar técnicas de feature selection a fim de preparar nosso dataset para outros algoritmos de ML.

Ao criarmos árvores de decisão, podemos ter árvores com muitos “galhos e folhas” e em algum momento teremos que parar a construção da árvore ou fazer ajustes reduzindo o número de pontos de decisão no modelo preditivo.

Árvores de Decisão Definimos um conjunto de regras e para cada regra há uma decisão que precisa ser tomada.

Componentes da Árvore de Decisão:

- Raiz (Root) – Nó Raiz – Atributo
- Ramo (Branch) – Valores que o Atributo pode assumir
- Nó (Node) – Condição – Checagem do valor do atributo
- Leaf (Folha) – Nó Folha (Ponto de Decisão) – Variável Target

Árvores de Decisão podem ser usadas para problemas de:

- Classificação – Árvore de Classificação
- Regressão – Árvore de Regressão

Considerações na Construção de Árvores de Decisão:

O segredo é saber como dividir os atributos:

Qual atributo deve ser usado para iniciar a árvore?

Qual deve ser o atributo seguinte?

Quando parar de construir ramos na árvore (para evitar overfitting)?

Métodos para seleção de atributos:

- Ganho de Informação e Entropia
- Índice de Gini (Gini Index)
- Taxa de Ganho (Gain Ratio)

Vantagens e Desvantagens das Árvores de Decisão

Vantagens:

- Útil em exploração de dados: A árvore de decisão é uma das formas mais rápidas de identificar as variáveis mais significativas e a relação entre duas ou mais variáveis. Com a ajuda de árvores de decisão, podemos criar novas variáveis/características que tenham melhores condições de prever a variável alvo.
- Fácil de entender: A visualização de uma árvore de decisão torna o problema fácil de compreender, mesmo para pessoas que não tenham perfil analítico. Não requer nenhum

conhecimento estatístico para ler e interpretar. Sua representação gráfica é muito intuitiva e permite relacionar as hipóteses também facilmente.

- Menor necessidade de limpar dados: Requer menos limpeza de dados em comparação com outras técnicas de modelagem. Até um certo nível, não é influenciado por pontos fora da curva “outliers” nem por valores faltantes (“missing values”).
- Não é restrito por tipos de dados: Pode manipular variáveis numéricas e categóricas.
- Método não paramétrico: A árvore de decisão é considerada um método não paramétrico. Isto significa que as árvores de decisão não pressupõem a distribuição do espaço nem a estrutura do classificador.

Desvantagens:

- Sobreajuste (“Overfitting”): Sobreajuste é uma das maiores dificuldades para os modelos de árvores de decisão. Este problema é resolvido através da definição de restrições sobre os parâmetros do modelo e da poda (pruning).
- Não adequado para variáveis contínuas: ao trabalhar com variáveis numéricas contínuas, a árvore de decisão perde informações quando categoriza variáveis em diferentes categorias.

Ganho de Informação, Entropia, Índice Gini e Pruning

As árvores de decisão têm desfrutado de muita popularidade por causa de seu algoritmo intuitivo. Sua saída é facilmente traduzida em regras e, portanto, é bastante compreensível pelos seres humanos (diferente de modelos como SVM e Redes Neurais, consideradas caixas pretas).

Processo de Aprendizado dos Algoritmos de Árvore de Decisão:

Usando uma amostra de observações como ponto de partida, o algoritmo identifica as regras que geraram as classes de saída (em problemas de classificação) ou valores numéricos (em problemas de regressão), e divide a matriz de entrada em partições menores, até que o processo acione uma regra de finalização.

Dados → Regras → Partições → (Regra de Parada) → Decisão

Greedy Search (Busca Gananciosa ou Gulosa)

O algoritmo procura maximizar o passo atual sem olhar para o passo seguinte, a fim de alcançar uma otimização global.

Greedy Search utiliza uma heurística estimada $h(n)$.

S → Estado inicial

G1, G2 → Objetivo

Entre as medidas mais utilizadas nos algoritmos de árvores de decisão, temos:

- Índice Gini
- Ganho de Informação
- Redução de Variância

Ross Quinlan criou um algoritmo de árvore de decisão baseado no ganho de informação, chamado ID3. Em seguida, esse algoritmo foi atualizado para C4.5 e C5.0.

Ross Quinlan → (ID3) → C4.5 → C5.0

Como definir o nó raiz e como realizar a divisão do conjunto de dados?

- Estratégia Gulosa (Greedy Selection)

Ao se trabalhar com Estratégia Gulosa, é necessário a medida da “impureza” do nó:



As principais formas de se obter a medida de impureza do nó são:

- Entropia
- Índice de Gini
- Erro de Classificação
- Divisão baseada em atributos nominais
 - Divisão Binária
 - Divisão Múltipla
- Divisão baseada em atributos contínuos
 - Decisão Binária
 - Discretização
 - Estática
 - Dinâmica

Entropia

Entropia é a medida da incerteza nos dados.

Ganho de Informação é a redução da Entropia.

Nos algoritmos ID3, C4.5 e C5.0, o nó raiz é escolhido com base em quanto do total da Entropia é reduzido, se aquele nó é escolhido Isso é chamado de Ganho de Informação!

Ganho de Informação = Entropia do sistema antes da divisão – Entropia do sistema após a divisão

$$E = - \sum_{i=1}^m p_i \log_2(p_i)$$
$$E_A = \sum_{i=1}^v \frac{D_i}{D} E(D_i)$$

Esta metodologia (Entropia) é aplicada para computar o ganho de informação para todos os

atributos. É escolhido o atributo com o mais alto ganho de informação. Isso é testado para cada nó a fim de escolher o melhor nó.

Índice de Gini

O Índice de Gini é usado para medir a probabilidade de dois itens aleatórios pertencerem à mesma classe.

A medida de Gini de um nó é a soma dos quadrados das proporções das classes.

O Índice de Gini diz: se selecionarmos dois itens de uma população aleatoriamente, então eles devem ser da mesma classe e a probabilidade para isto é 1 se a população é pura.

O Índice de Gini é usado como regra de parada para construção de uma árvore de decisão.

O que são as regras de parada (Stopping Rules)?

Regras de parada são limites para expansão de uma árvore.

A decisão de fazer divisões estratégicas afeta a precisão de uma árvore de decisão. As regras de parada são diferentes para árvores de classificação e de regressão.

As medidas mais utilizadas para construir todo o algoritmo de árvore de decisão, considerando as regras de parada:

- Índice Gini
- Qui-quadrado
- Ganho de Informação
- Redução de Variância

Pruning – Poda da Árvore

Mesmo com regras de parada, é possível que uma árvore de decisão fique muito grande.

Eliminando alguns nós ou galhos, é possível tornar o algoritmo menos propenso a overfitting.

- A árvore de decisão é concluída antes que uma classificação perfeita dos dados de treinamento seja alcançada.
- Ocorre o excesso de ajuste nos dados gerando um modelo e, em seguida, a árvore é podada (Pruning) para se tornar generalizável.

E como definir o tamanho correto da árvore?

- Usar um conjunto de validação – divisão do conjunto de dados original pode ser feita em 3 partes: treino, validação e teste. Usar um conjunto de dados de validação permite ajustar o modelo durante sua elaboração, antes da fase de teste.
- Usar métodos probabilísticos – verificar se a poda de um determinado nó, tem a probabilidade de produzir qualquer melhoria, no resultado final do modelo.

O classificador de árvore de decisão do Scikit-Learn não suporta atualmente o Pruning. Pacotes avançados como o XGBoost adotaram a poda de árvores em sua implementação. Mas a biblioteca rpart em R, fornece uma função para Pruning.

Compreendendo a Heurística ID3

As árvores de decisão são conhecidas como uma das mais poderosas e amplamente utilizadas técnicas de modelagem de aprendizado de máquina. As árvores de decisão naturalmente podem induzir regras que podem ser usadas para classificação de dados ou para realizar previsões.

O que são Heurísticas?

Heurísticas são soluções baseadas em algum tipo de conhecimento prévio sobre as propriedades dos dados, na procura de uma boa solução (mas não necessariamente a melhor). A pesquisa por heurísticas é uma pesquisa realizada por meio da quantificação de proximidade a um determinado objetivo.

Diz-se que se tem uma boa (ou alta) heurística se o objeto de avaliação está muito próximo do objetivo; dizemos que a heurística é ruim (ou baixa) se o objeto avaliado estiver muito longe do objetivo. A palavra heurística significa descobrir (e que deu origem também ao termo Eureka!, que vc já deve ter ouvido muitas e muitas vezes). Heurísticas de construção, tais como o método guloso (que vimos nos vídeos anteriores), são aquelas onde uma ou mais soluções são construídas elemento a elemento, seguindo algum critério heurístico de otimização, até que se tenha uma solução viável; O algoritmo ID3 possui uma heurística de árvore de decisão.

Este algoritmo segue os seguintes passos:

- 1- Começa com todos os exemplos de treino.
- 2- Escolhe o teste (atributo) que melhor divide os exemplos, ou seja, agrupa exemplos da mesma classe ou exemplos semelhantes. Nesta etapa, para achar o melhor atributo é necessário encontrar a entropia para cada atributo possível naquele nó. Isso é feito pelo algoritmo. Atributo com maior ganho de informação é selecionado para ser raiz da árvore de decisão.
- 3- Para o atributo escolhido, é criado um nó filho para cada valor possível do atributo. O próximo passo na heurística ID3 é calcular o ganho de informação para cada atributo que pode ser selecionado como nó na árvore. Essencialmente é apenas calcular a entropia de todo o conjunto de dados e diminuir este da entropia do sub-conjunto particionado para tal atributo (aquela fórmula que mostramos a você nos vídeos anteriores). Este processo é feito para cada atributo do conjunto de dados, e o atributo com o maior ganho de informação será o selecionado para o próximo nó da árvore.
- 4- Transporta os exemplos para cada filho considerando o valor do filho.
- 5- Repete o procedimento para cada filho não "puro". Um filho é puro quando cada atributo X tem o mesmo valor em todos os exemplos.

E como o algoritmo sabe o melhor atributo a escolher? Através do Ganho de Informação e Entropia!! A heurística ID3 usa o conceito de entropia para formular o ganho de informação na escolha de um atributo particular para ser o próximo nó na árvore de decisão.

A Heurística ID3

A física usa o termo entropia para descrever a quantidade de desordem associada a um sistema. Na teoria da informação, este termo tem um significado semelhante - ele mede o grau de desordem de um conjunto de dados. A heurística ID3 usa este conceito para encontrar o próximo melhor atributo de um dado para ser utilizado como nó de uma árvore de decisão.

Logo, a ideia por trás do algoritmo ID3 é achar um atributo que reduza em maior valor a entropia de um conjunto de dados, assim reduzindo a aleatoriedade - dificuldade de previsão - da variável que define classes. Seguindo esta heurística, você estará essencialmente encontrando o melhor atributo para classificar os registros (de acordo com a redução da quantidade de informação necessária para descrever a partição dos dados que foram divididos) a fim de que os mesmos tenham utilidade máxima (exemplos são da mesma classe).

Espaço de Hipóteses do ID3

O ID3 busca no espaço de hipóteses alguma que seja adequada para representar o conjunto de treinamento. Esse espaço de hipóteses é formado por um conjunto de todas possíveis árvores de decisão. O ID3 começa com árvore vazia e progressivamente elabora hipóteses até chegar em uma árvore de decisão.

A busca por hipóteses é guiada pelo Ganho de Informação dos atributos. Como ID3 não mantém todas hipóteses consistentes com o conjunto de treinamento, o ID3 não tem a habilidade de determinar quantas árvores de decisão alternativas são consistentes com os dados de treinamento. O ID3 também não realiza backtracking na busca, ou seja, uma vez que tenha selecionado um atributo, não reavalia a árvore de decisão formada. O ID3 emprega todos os dados de treinamento em cada passo da busca por hipóteses e toma decisões estatísticas em cada passo. A abordagem do ID3 privilegia árvores mais curtas em relação às mais longas observadas e atributos de maior Ganho de Informação mais próximos do topo ou raiz da árvore.

O ID3 deu origem a outros algoritmos semelhantes, os quais descrevemos abaixo e que serão estudados mais adiante.

ID3 (Iterative Dichotomizer 3) - O ID3 é um algoritmo usado para gerar árvores de decisão. Os atributos do conjunto de dados devem ser obrigatoriamente categóricos.

C4.5 - Um algoritmo para geração de árvores de decisão, sucessor do algoritmo ID3. O algoritmo C4.5 considera atributos numéricos e categóricos.

C5.0 - O C5.0 foi inicialmente uma versão comercial, que aliás ainda é comercializado (link na seção de links úteis), mas que teve o código aberto para a comunidade open-source. Também uma evolução do ID3.

CART (Classification and Regression Trees) - Técnica não-paramétrica que produz árvores de classificação ou regressão, dependendo se as variáveis são categóricas ou numéricas, respectivamente.

Construindo um Árvore de Decisão em R /

Pruning da Árvore de Decisão em R /

Estudo de Caso 2-Construindo um Modelo de Decisão p/ Risco Crédito

```
# Criando Árvore de Decisão com o pacote rpart

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap08/R")

# Criando um dataframe
?expand.grid
```

```

clima <- expand.grid(Tempo = c("Ensolarado", "Nublado", "Chuvoso"),
  Temperatura = c("Quente", "Ameno", "Frio"),
  Humidade = c("Alta", "Normal"),
  Vento = c("Fraco", "Forte"))

# Visualizando o dataframe
View(clima)

# Vetor para selecionar as linhas
response <- c(1, 19, 4, 31, 16, 2, 11, 23, 35, 6, 24, 15, 18, 36)

# Gerando um vetor do tipo fator para a Variável target
play <- as.factor(c("Não Jogar", "Não Jogar", "Não Jogar", "Jogar", "Jogar", "Jogar", "Jogar", "Jogar", "Jogar", "Jogar", "Não Jogar", "Jogar",
"Jogar", "Não Jogar"))

# Dataframe final
tennis <- data.frame(clima[response,], play)
View(tennis)

# Carregando o pacote
install.packages("rpart")
library(rpart)

# Criando o modelo
?rpart
?rpart.control
tennis_tree <- rpart(play ~ .,
  data = tennis,
  method = "class",
  parms = list(split = "information"),
  control = rpart.control(minsplit = 1))

# Visualizando o ganho de informação para cada atributo
tennis_tree

# Gerando o plot
install.packages("rpart.plot")
library(rpart.plot)

# Plot
?prp
prp(tennis_tree, type = 0, extra = 1, under = TRUE, compress = TRUE)

# Interpretando a árvore de decisão

# Para ler os nós da árvore, basta iniciar a partir do nó superior, que corresponde aos dados de treinamento original e, em seguida,
# começar a ler as regras. Observe que cada nó tem duas derivações: O ramo esquerdo significa que a regra superior é verdadeira
# e a direita significa que ela é falsa.

# À esquerda da primeira regra, você vê uma regra terminal importante (uma folha terminal), em um círculo, indicando
# um resultado positivo, Jogar, que você pode ler como jogar tênis = Verdadeiro. Os números sob a folha terminal mostram
# quatro exemplos afirmando que esta regra é "yes" e zero afirmando "no".

# Considere o atributo "Vento" que pode ter os valores "Fraco" ou "Forte". Calcula-se então a entropia para cada um desses
# valores e depois a diferença entre a entropia do atributo vento e a soma das entropias de cada um dos valores associados
# ao atributo, multiplicado pela proporção de exemplos particionados de acordo com o valor (separados de um lado os exemplos
# com Vento = "Fraco" e do outro lado Vento = "Forte").

# Frequentemente, as regras de árvore de decisão não são imediatamente utilizáveis, e você precisa interpretá-las antes do uso.
# No entanto, eles são claramente inteligíveis (e muito melhor do que um coeficiente de vetores de valores).

# Fazendo previsões com o modelo

# Dados
clima <- expand.grid(Tempo = c("Ensolarado", "Nublado", "Chuvoso"),
  Temperatura = c("Quente", "Ameno", "Frio"),
  Humidade = c("Alta", "Normal"),
  Vento = c("Fraco", "Forte"))

# Vetor para selecionar as linhas
response <- c(2, 20, 3, 33, 17, 4, 5)

# Novos dados
novos_dados <- data.frame(clima[response,])
View(novos_dados)

# Previsões

```

```
?predict  
predict(tennis_tree, novos_dados)
```

```
# Criando um árvore de decisão a partir do dataset titanic
```

```
# Obs: Caso tenha problemas com a acentuação, consulte este link:  
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
```

```
# Definindo o diretório de trabalho  
getwd()  
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap08/R")
```

```
# Gerando o dataset  
data(Titanic, package = "datasets")
```

```
# Criando o dataframe  
dataset <- as.data.frame(Titanic)  
View(dataset)
```

```
# Carregando o pacote  
install.packages("rpart")  
library(rpart)
```

```
# Criando o modelo  
titanic_tree <- rpart(Survived ~ Class + Sex + Age,  
                      data = dataset,  
                      weights = Freq,  
                      method = "class",  
                      parms = list(split = "information"),  
                      control = rpart.control(minsplit = 5))
```

```
titanic_tree
```

```
# Aplicando o Prune  
?prune  
pruned_titanic_tree <- prune(titanic_tree, cp = 0.02)  
pruned_titanic_tree
```

```
# Carregando o pacote rpart.plot  
install.packages("rpart.plot")  
library(rpart.plot)
```

```
# Imprimindo a árvore antes e depois do Prune
```

```
# Antes do Pruning  
prp(titanic_tree, type = 0, extra = 1, under = TRUE, compress = TRUE)
```

```
# Depois do Pruning  
prp(pruned_titanic_tree, type = 0, extra = 1, under = TRUE, compress = TRUE)
```

```
# Estudo de Caso 2 - Construindo Um Modelo de Decisão Para Risco de Crédito
```

```
# Obs: Caso tenha problemas com a acentuação, consulte este link:  
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
```

```
# Definindo o diretório de trabalho  
getwd()  
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap08/R")
```

```
# Calculando a entropia de duas classes  
-0.60 * log2(0.60) - 0.40 * log2(0.40)
```

```
# Gerando a curva de Entropia  
curve(-x * log2(x) - (1 - x) * log2(1 - x), col = "red", xlab = "x", ylab = "Entropy", lwd = 4)
```

```
# Identificando o risco de crédito  
credit <- read.csv("dados/credito.csv")  
str(credit)  
View(credit)
```

```
# Verificando 2 atributos do cliente  
table(credit$checking_balance)  
table(credit$savings_balance)
```

```
# Verificando as características do crédito  
summary(credit$months_loan_duration)  
summary(credit$amount)
```

```

# Variável target
table(credit$default)

# Usando sample para construir os dados de treino e de teste
set.seed(123)
train_sample <- sample(1000, 900)

# Split dos dataframes
credit_train <- credit[train_sample, ]
credit_test <- credit[-train_sample, ]

# Verificando a proporção da variável target
prop.table(table(credit_train$default))
prop.table(table(credit_test$default))

# Construindo um modelo
install.packages("C50")
library(C50)
?C5.0

# Criando e visualizando o modelo
credit_model <- C5.0(credit_train[-17], credit_train$default)
credit_model

# Informações detalhadas sobre a árvore
summary(credit_model)

# Avaliando a performance do modelo
credit_pred <- predict(credit_model, credit_test)

# Confusion Matrix para comparar valores observados e valores previstos
install.packages("gmodels")
library(gmodels)

# Criando a Confusion Matrix
?CrossTable
CrossTable(credit_test$default,
  credit_pred,
  prop.chisq = FALSE,
  prop.c = FALSE,
  prop.r = FALSE,
  dnn = c('Observado', 'Previsto'))

# Melhorando a performance do modelo

# Aumentando a precisão com 10 tentativas
credit_boost10 <- C5.0(credit_train[-17], credit_train$default, trials = 10)
credit_boost10
summary(credit_boost10)

# Score do modelo
credit_boost_pred10 <- predict(credit_boost10, credit_test)

# Confusion Matrix
CrossTable(credit_test$default,
  credit_boost_pred10,
  prop.chisq = FALSE,
  prop.c = FALSE,
  prop.r = FALSE,
  dnn = c('Observado', 'Previsto'))

# Dando pesos aos erros

# Criando uma matriz de dimensões de custo
matrix_dimensions <- list(c("no", "yes"), c("no", "yes"))
names(matrix_dimensions) <- c("Previsto", "Observado")
matrix_dimensions

# Construindo a matriz
error_cost <- matrix(c(0, 1, 4, 0), nrow = 2, dimnames = matrix_dimensions)
error_cost

# Aplicando a matriz a árvore
?C5.0
credit_cost <- C5.0(credit_train[-17], credit_train$default, costs = error_cost)

# Score do modelo

```

```
credit_cost_pred <- predict(credit_cost, credit_test)
```

```
# Confusion Matrix
CrossTable(credit_test$default,
  credit_cost_pred,
  prop.chisq = FALSE,
  prop.c = FALSE,
  prop.r = FALSE,
  dnn = c('Observado', 'Previsto'))
```

Construindo uma Árvore de Decisão em Python / Classificador Random Forest em Python / Regressor Random Forest em Python

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 9 - Decision Tree em Python</font>
# <font color='blue'>Entropia e Índice Gini</font>
# ->
!pip install pydot
# ->
!pip install graphviz
# ->
#!pip install graphviz
### Talvez seja necessário executar esse comando (CMD) para o windows
#!conda install python-graphviz
# Documentação em http://www.graphviz.org
# ->
# Importando os módulos
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import pydot
import graphviz
# ->
# Criando o dataset
instancias = [
  {'Melhor Amigo': False, 'Especie': 'Cachorro'},
  {'Melhor Amigo': True, 'Especie': 'Cachorro'},
  {'Melhor Amigo': True, 'Especie': 'Gato'},
  {'Melhor Amigo': True, 'Especie': 'Gato'},
  {'Melhor Amigo': False, 'Especie': 'Gato'},
  {'Melhor Amigo': True, 'Especie': 'Gato'},
  {'Melhor Amigo': True, 'Especie': 'Gato'},
  {'Melhor Amigo': False, 'Especie': 'Cachorro'},
  {'Melhor Amigo': True, 'Especie': 'Gato'},
  {'Melhor Amigo': False, 'Especie': 'Cachorro'},
  {'Melhor Amigo': False, 'Especie': 'Cachorro'},
  {'Melhor Amigo': False, 'Especie': 'Gato'},
  {'Melhor Amigo': True, 'Especie': 'Gato'},
  {'Melhor Amigo': True, 'Especie': 'Cachorro'}
]
# ->
# Transformando o Dicionário em DataFrame
df = pd.DataFrame(instancias)
# ->
df
# ->
# Preparando os dados de treino e de teste
X_train = [[1] if a else [0] for a in df['Melhor Amigo']]
y_train = [1 if d == 'Cachorro' else 0 for d in df['Especie']]
labels = ['Melhor Amigo']
# ->
print(X_train)
# ->
print(y_train)
# ->
# Construindo o Classificador Baseado em Entropia
# http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
modelo_v1 = DecisionTreeClassifier(max_depth = None,
  max_features = None,
  criterion = 'entropy',
  min_samples_leaf = 1,
```

```

min_samples_split = 2)
# ->
# Apresentando os dados ao Classificador
modelo_v1.fit(X_train, y_train)
# ->
# Definindo o nome do arquivo com a árvore de decisão
arquivo = 'Users/dmpm/Dropbox/DSA/MachineLearning2.0/Cap09/Python/tree_modelo_v1.dot'
# ->
# Gerando o gráfico da árvore de decisão
export_graphviz(modelo_v1, out_file = arquivo, feature_names = labels)
with open(arquivo) as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
# ->
!dot -Tpng tree_modelo_v1.dot -o tree_modelo_v1.png
# ->
# Construindo o Classificador Baseado no índice Gini
# http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
modelo_v2 = DecisionTreeClassifier(max_depth = None,
                                   max_features = None,
                                   min_samples_leaf = 1,
                                   min_samples_split = 2)
# ->
# Apresentando os dados ao Classificador
modelo_v2.fit(X_train, y_train)
# ->
# Definindo o nome do arquivo com a árvore de decisão
arquivo = 'Users/dmpm/Dropbox/DSA/MachineLearning2.0/Cap09/Python/tree_modelo_v2.dot'
# ->
# Gerando o gráfico da árvore de decisão
export_graphviz(modelo_v2, out_file = arquivo, feature_names = labels)
with open(arquivo) as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
# ->
!dot -Tpng tree_modelo_v2.dot -o tree_modelo_v2.png
#### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 9 - Random Forest</font>
## Criando uma Decision Tree
# ->
import numpy as np
import pandas as pd
import sklearn as sk
from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# ->
# Carrega o dataset
irisData = pd.read_csv("dados/iris_data.csv")
# ->
# Visualiza as primeiras linhas
print(irisData.head())
# ->
# Resumo estatístico
print(irisData.describe())
# ->
# Correlação
print(irisData.corr())
# ->
# Atributos e Variável target
features = irisData[["SepalLength", "SepalWidth", "PetalLength", "PetalWidth"]]
targetVariables = irisData.Class
# ->
# Gera os dados de treino
featureTrain, featureTest, targetTrain, targetTest = train_test_split(features,
                                                                        targetVariables,
                                                                        test_size = .2)
# ->
?train_test_split
# ->
# Criação do modelo

```

```

clf = DecisionTreeClassifier()
# ->
print(clf)
# ->
modelo = clf.fit(featureTrain, targetTrain)
previsoes = modelo.predict(featureTest)
# ->
print (confusion_matrix(targetTest, previsoes))
# ->
# Obs: Dependendo da versão do Scikit-Learn, a acurácia pode ser diferente
print (accuracy_score(targetTest, previsoes))
## Random Forest Classifier - I
# ->
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_digits
from sklearn.preprocessing import scale
%matplotlib inline
# ->
# Gera o dataset
digitos = load_digits()
# ->
# Aplica Escala nos dados
data = scale(digitos.data)
# ->
data
# ->
data.shape
# ->
# Obtém número de observações e número de atributos
n_observ, n_features = data.shape
# ->
n_observ
# ->
n_features
# ->
# Obtém os labels
n_digits = len(np.unique(digitos.target))
labels = digitos.target
# ->
labels
# ->
# Cria o classificador
# http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
clf = RandomForestClassifier(n_estimators = 10)
Os 4 principais parâmetros em Modelos de Random Forest são:
n_estimators - quanto maior, melhor!
max_depth - o padrão é 'none' e nesse caso árvores completas são criadas. Ajustando esse parâmetro pode ajudar a evitar overfitting.
max_features - diferentes valores devem ser testados, pois este parâmetro impacta na forma como os modelos RF distribuem os atributos pelas árvores.
criterion - define a forma como o algoritmo fará a divisão dos atributos e a classificação dos nós em cada árvore.
# ->
# Construção do modelo
clf = clf.fit(data, labels)
# ->
clf
# ->
# Estamos apenas "simulando" os dados de teste
scores = clf.score(data, labels)
# ->
print(scores)
# ->
# Extraíndo a importância
importances = clf.feature_importances_
indices = np.argsort(importances)
# ->
# Obtém os índices
ind=[]
for i in indices:
    ind.append(labels[i])
# ->
# Plot da Importância dos Atributos
plt.figure(1)
plt.title('Importância dos Atributos')
plt.barh(range(len(indices)), importances[indices], color = 'b', align = 'center')
plt.yticks(range(len(indices)), ind)
plt.xlabel('Importância Relativa')

```

```

plt.show()
## Random Forest Classifier - II
# ->
!pip install treeinterpreter
# ->
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from treeinterpreter import treeinterpreter as ti
from sklearn.datasets import load_iris
# ->
# Carrega o dataset
iris = load_iris()
# ->
# Cria o classificador
rf = RandomForestClassifier(max_depth = 4)
# ->
# Obtém os índices a partir do comprimento da variável target
idx = list(range(len(iris.target)))
# ->
# Randomiza o índice
np.random.shuffle(idx)
# ->
# Cria o modelo
rf.fit(iris.data[idx][:100], iris.target[idx][:100])
# ->
?rf.predict_proba
# ->
# Obtém as instâncias (exemplos ou observações) e retorna as probabilidades
# Obs: Como estamos trabalhando com randomização os resultados na sua máquina podem ser ligeiramente diferentes!
instance = iris.data[idx][100:101]
print(rf.predict_proba(instance))
# ->
prediction, bias, contributions = ti.predict(rf, instance)
print("Previsões", prediction)
print("Contribuição dos Atributos:")
for item, feature in zip(contributions[0], iris.feature_names):
    print(feature, item)
## Random Forest Regressor
### Random Forest Classifier - composto por árvores de decisão de classificação
### Random Forest Regressor - composto por árvores de decisão de regressão
http://www.boardgamegeek.com/
Nosso dataset possui registros de 81.312 Games Boards como esse: http://www.boardgamegeek.com/boardgame/167791/terraforming-mars
### Colunas no dataset:
https://github.com/ThaWeatherman/scrapers/tree/master/boardgamegeek
name – name of the board game.
playingtime – the playing time (given by the manufacturer).
minplaytime – the minimum playing time (given by the manufacturer).
maxplaytime – the maximum playing time (given by the manufacturer).
minage – the minimum recommended age to play.
usersRated – the number of users who rated the game.
averageRating – the average rating given to the game by users. (0-10)
totalWeights – Number of weights given by users. Weight is a subjective measure that is made up by BoardGameGeek.
It's how "deep" or involved a game is. Here's a full explanation.
averageWeight – the average of all the subjective weights (0-5).
# ->
# Import
import pandas
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
%matplotlib inline
# ->
# Carregando o dataset
games = pandas.read_csv("dados/games_data.csv")
# ->
# Imprimindo o nome das colunas
print(games.columns)
# ->
print(games.shape)
# ->
# Histograma com a média de avaliações
plt.hist(games["average_rating"])
plt.show()
# ->
# Visualizando as observações com rating igual a 0
games[games["average_rating"] == 0]

```



```

# ->
# Retornando a primeira linha do subset do dataframe, onde o índice é igual a 0.
print(games[games["average_rating"] == 0].iloc[0])
# ->
# Retornando a primeira linha do subset do dataframe, onde o índice é maior que 0.
print(games[games["average_rating"] > 0].iloc[0])
# ->
# Removendo as linhas sem avaliação de usuários.
games = games[games["users_rated"] > 0]
# ->
# Removendo linhas com valores missing
games = games.dropna(axis = 0)
# ->
# Histograma com a média de avaliações
plt.hist(games["average_rating"])
plt.show()
# ->
# Correlação
games.corr()["average_rating"]
# ->
# Obtém todas as colunas do dataframe
colunas = games.columns.tolist()
# ->
# Filtra as colunas e remove as que não são relevantes
colunas = [c for c in colunas if c not in ["bayes_average_rating", "average_rating", "type", "name"]]
# ->
# Preparando a variável target, a que será prevista
target = "average_rating"
# ->
# Gerando os dados de treino
df_treino = games.sample(frac = 0.8, random_state = 101)
# ->
# Seleciona tudo que não está no dataset de treino e armazena no dataset de teste
df_teste = games.loc[~games.index.isin(df_treino.index)]
# ->
# Shape dos datasets
print(df_treino.shape)
print(df_teste.shape)
# ->
# Criando um Regressor
reg_v1 = LinearRegression()
# ->
# Fit the model to the training data.
modelo_v1 = reg_v1.fit(df_treino[colunas], df_treino[target])
# ->
modelo_v1
# ->
# Fazendo previsões
previsoes = modelo_v1.predict(df_teste[colunas])
# ->
# Computando os erros entre valores observados e valores previstos
mean_squared_error(previsoes, df_teste[target])
# ->
# Criando um regressor Random Forest
reg_v2 = RandomForestRegressor(n_estimators = 100, min_samples_leaf = 10, random_state = 101)
# ->
# Criando o modelo
modelo_v2 = reg_v2.fit(df_treino[colunas], df_treino[target])
# ->
# Fazendo previsões
previsoes = modelo_v2.predict(df_teste[colunas])
# ->
# Computando o erro
mean_squared_error(previsoes, df_teste[target])
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Métodos Ensemble

Existe uma grande variedade de métodos para induzir modelos a partir de dados. Mas qual algoritmo irá ter o melhor desempenho para um determinado problema?

Com o objetivo de responder esta pergunta, Tom Michel, autor de um dos mais famosos livros sobre Machine Learning (que você encontra na seção de bibliografia), abordou esta questão executando um estudo com 23 algoritmos diferentes e construindo uma árvore de decisão para prever o melhor algoritmo a ser usado, dadas as propriedades de um dataset.

Embora o estudo estivesse inclinado em direção a árvores de decisão – pois 9 dos 23 algoritmos eram de árvores de decisão, o estudo revelou informações úteis, chegando a seguinte conclusão: a melhor forma de aumentar a acurácia de um modelo é juntar modelos diferentes a fim de criar um modelo final mais preciso. A isso deu-se o nome de Método Ensemble. Ensemble é uma palavra francesa que significa agrupamento.

Métodos Ensemble são uma categoria de Algoritmos de Machine Learning, que podem ser usados tanto em aprendizagem supervisionada quanto não supervisionada!

Construir um ensemble consiste em dois passos:

- 1- Construir diversos modelos.
- 2- Combinar suas estimativas.

Pode-se gerar modelos diferentes, variando por exemplo os pesos das observações, valores de dados, parâmetros, subconjuntos de variáveis ou aplicando diferentes técnicas de pré-processamento. A combinação pode ser feita por meio de votação, mas normalmente a combinação é feita por meio de pesos das estimativas dos modelos. Métodos Ensemble permitem aumentar consideravelmente o nível de precisão nas suas previsões

Aqui estão os principais modelos de métodos ensemble:

- Random Forest

O Random Forest é um método ensemble que combina diversas árvores de decisão e adiciona um componente estocástico para criar mais diversidade entre as árvores de decisão.

- Bagging

O Bagging, ou Bootstrap Aggregation, aplica técnicas de bootstrap (reamostragem) no dataset de treino a fim de construir diversas árvores de decisão e depois obtém a melhor estimativa por votação entre as árvores ou pelo peso das estimativas

- AdaBoost

O Adaboost constói diversos modelos de forma iterativa, variando o peso nos exemplos no dataset de treino, penalizando exemplos com muitos erros e reduzindo o peso daqueles com estimativa incorreta ou menos precisa.

- Gradient Boosting

O Gradient Boosting é uma extensão do AdaBoost com uma variedade de funções de erro para classificação e regressão.

Estado da Arte em Machine Learning

Métodos Ensemble têm uma técnica eficaz e poderosa para alcançar alta precisão em soluções supervisionadas e não supervisionadas. Diferentes modelos são eficientes e funcionam muito bem quando aplicados de forma isolada.

Os métodos ensemble permitem combinar modelos concorrentes para formar uma espécie de comitê, e tem havido muita pesquisa nesta área com um bom grau de sucesso. Os sistemas de recomendação e os aplicativos de mineração de texto baseados em fluxo, usam amplamente métodos ensemble.

Um princípio amplamente discutido em Machine Learning, é que acurácia e simplicidade do modelo levam ao estado da arte em análise preditiva. Mas conseguir os 2 normalmente é muito difícil, sendo um trade-off: um modelo mais complexo é mais flexível, mas consequentemente mais suscetível ao overfitting e provavelmente não vai generalizar bem em novos conjuntos de dados. Modelos mais simples podem não conseguir realizar o aprendizado de forma efetiva, embora reduzam o tempo total de treinamento.

Técnicas de regularização tem sido utilizadas como forma de se atingir o estado da arte em Machine Learning, aplicando uma função de erro que penaliza a complexidade do modelo. A regularização é apontada como uma das principais razões da performance muito superior de modelos de métodos ensemble.

Tem havido muitos estudos independentes realizados em grupos de aprendizagem supervisionados e não supervisionados. O tema comum observado é que muitos modelos diferentes, quando reunidos, fortaleceram os modelos fracos e geram um melhor desempenho global. Aprendizagem baseada em métodos ensemble é apenas uma das muitas categorias de modelos de aprendizagem.

Ensemble, em geral, significa um grupo de coisas que são geralmente vistos como um todo. Os conjuntos seguem uma abordagem de divisão e conquista usada para melhorar o desempenho.

E temos três categorias principais:

- Bagging é usado para construção de múltiplos modelos (normalmente do mesmo tipo) a partir de diferentes subsets no dataset de treino.
- Boosting é usado para construção de múltiplos modelos (normalmente do mesmo tipo), onde cada modelo aprende a corrigir os erros gerados pelo modelo anterior, dentro da sequência de modelos criados.
- Voting é usado para construção de múltiplos modelos (normalmente de tipos diferentes). Estatísticas simples (como a média) são usadas para combinar as previsões.

Os métodos ensemble são, comprovadamente, poderosos métodos para melhorar a precisão e robustez de soluções supervisionadas, semi-supervisionadas e não supervisionadas. O conceito é simples. A partir do dataset original, geramos diferentes subsets que irão alimentar diferentes modelos individuais. Cada modelo individual, gera uma estimativa que serão combinadas para gerar um resultado final. Mais a frente vamos discutir as técnicas envolvidas em cada uma destas etapas.

Mas é importante ressaltar, que esta regra se aplica aos dados de treino. Uma vez criado o modelo final a partir do método ensemble, normalmente apenas o modelo final é aplicado ao dataset de teste, gerando assim a avaliação final do modelo.

Métodos Ensemble – Bagging /

Métodos Ensemble – Extremely Randomized Trees (Extra Trees) /

Métodos Ensemble – Adaboost Para Prever Doenças do Coração /

Parâmetros x Hiperparâmetros /

Otimização dos Hiperparâmetros com Randomized Search /

Grid Search x Randomized Search para Estimação de

Hiperparâmetros /

Construindo um Classificador Gradient Boosting em Python /

Construindo um Regressor Gradient Boosting em Python /

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 9 - Métodos Ensemble</font>
## Bagging
Bagging é usado para construção de múltiplos modelos (normalmente do mesmo tipo) a partir de diferentes subsets no dataset de treino.
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html
Um classificador Bagging é um meta-estimador ensemble que faz o fit de classificadores base, cada um em subconjuntos aleatórios do conjunto de
dados original e, em seguida, agrega suas previsões individuais (por votação ou por média) para formar uma previsão final.
Tal meta-estimador pode tipicamente ser usado como uma maneira de reduzir a variância de um estimador (por exemplo, uma árvore de decisão),
introduzindo a randomização em seu procedimento de construção e fazendo um ensemble (conjunto) a partir dele.
# ->
# Imports
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_digits
from sklearn.preprocessing import scale
from sklearn.model_selection import cross_val_score
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# ->
# Carga de dados
digits = load_digits()
# ->
import matplotlib.pyplot as plt
%matplotlib inline
plt.gray()
plt.matshow(digits.images[5])
plt.show()
# ->
# Pré-processamento
# Coloca todos os dados na mesma escala
data = scale(digits.data)
# ->
# Variáveis preditoras e variável target
X = data
y = digits.target
# ->
# Construção do Classificador
bagging = BaggingClassifier(KNeighborsClassifier(), max_samples = 0.5, max_features = 0.5)
# ->
bagging
# ->
?cross_val_score
# ->
# Score do modelo
scores = cross_val_score(bagging, X, y)
# ->
# Média do score
mean = scores.mean()
# ->
print(scores)
# ->
print(mean)
## Extremely Randomized Trees (ExtraTrees)
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# ->
# Imports
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_digits
from sklearn.preprocessing import scale
```

```

from sklearn.model_selection import cross_val_score
# ->
# Carregando os dados
digits = load_digits()
# ->
# Pré-processamento
data = scale(digits.data)
# ->
# Variáveis preditoras e variável target
X = data
y = digits.target
# ->
# Cria o classificador com uma árvore de decisão
clf = DecisionTreeClassifier(max_depth = None, min_samples_split = 2, random_state = 0)
scores = cross_val_score(clf, X, y)
mean = scores.mean()
print(scores)
print(mean)
# ->
# Cria o classificador com Random Forest
clf = RandomForestClassifier(n_estimators = 10, max_depth = None, min_samples_split = 2, random_state = 0)
scores = cross_val_score(clf, X, y)
mean = scores.mean()
print(scores)
print(mean)
# ->
# Cria o classificador com Extra Tree
clf = ExtraTreesClassifier(n_estimators = 10, max_depth = None, min_samples_split = 2, random_state = 0)
scores = cross_val_score(clf, X, y)
mean = scores.mean()
print(scores)
print(mean)
## Adaboost
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
Um classificador AdaBoost é um meta-estimador que começa ajustando um classificador no conjunto de dados original e depois ajusta cópias
adicionais do classificador no mesmo conjunto de dados, mas onde os pesos das instâncias classificadas incorretamente são ajustados para que os
classificadores subsequentes se concentrem mais em casos difíceis.
# ->
# Import
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import fetch_openml
from sklearn.model_selection import cross_val_score
# ->
# Carregando os dados
# https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch\_openml.html
# https://archive.ics.uci.edu/ml/datasets/Heart+Disease
heart = fetch_openml("heart")
# ->
heart
# ->
X = heart.data
y = np.copy(heart.target)
y[y == -1] = 0
# ->
X
# ->
y
# ->
# Datasets de treino e de teste
X_test, y_test = X[189:], y[189:]
X_train, y_train = X[:189], y[:189]
# ->
# Construindo o estimador base
estim_base = DecisionTreeClassifier(max_depth = 1, min_samples_leaf = 1)
# ->
# Construindo a primeira versão do modelo Adaboost
ada_clf_v1 = AdaBoostClassifier(base_estimator = estim_base,
                                learning_rate = 0.1,
                                n_estimators = 400,
                                algorithm = "SAMME")
# ->
# Treinamento do modelo
ada_clf_v1.fit(X_train, y_train)
# ->
# Score
scores = cross_val_score(ada_clf_v1, X_test, y_test)

```

```

print(scores)
means = scores.mean()
print(means)
# ->
# Construindo a segunda versão do modelo Adaboost
ada_clf_v2 = AdaBoostClassifier(base_estimator = estim_base,
                                learning_rate = 1.0,
                                n_estimators = 400,
                                algorithm = "SAMME")
# ->
# Treinamento do modelo
ada_clf_v2.fit(X_train, y_train)
# ->
# Score
scores = cross_val_score(ada_clf_v2, X_test, y_test)
print(scores)
means = scores.mean()
print(means)
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 9 - Otimização dos Parâmetros com Randomized Search</font>

```

Todo modelo de Machine Learning possui parâmetros que permitem a customização do modelo. Esses parâmetros também são chamados de hiperparâmetros.

Em programação os algoritmos de Machine Learning são representados por funções e cada função possui os parâmetros de customização, exatamente o que chamamos de hiperparâmetros.

É comum ainda que as pessoas se refiram aos coeficientes do modelo (encontrados ao final do treinamento) como parâmetros.

Parte do nosso trabalho como Cientistas de Dados é encontrar a melhor combinação de hiperparâmetros para cada modelo.

Em Métodos Ensemble esse trabalho é ainda mais complexo, pois temos os hiperparâmetros do estimador base e os hiperparâmetros do modelo ensemble, conforme este exemplo abaixo:

```

* Estimador base:
estim_base = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=5, p=2,
weights='uniform')
* Modelo Ensemble:
BaggingClassifier(base_estimator=estim_base,
                  bootstrap=True, bootstrap_features=False, max_features=0.5,
                  max_samples=0.5, n_estimators=10, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)

```

Extremely Randomized Forest

Modelo padrão, com hiperparâmetros escolhidos manualmente.

```

# ->
# Imports
import pandas as pd
import numpy as np
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# ->
# Carrega o dataset
data = pd.read_excel('dados/credit.xls', skiprows = 1)
# ->
print(data)
# ->
# Variável target
target = 'default payment next month'
y = np.asarray(data[target])
# ->
# Variáveis preditoras
features = data.columns.drop(['ID', target])
X = np.asarray(data[features])
# ->
# Divisão de dados de treino e de teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 99)
# ->
# Classificador
clf = ExtraTreesClassifier(n_estimators = 500, random_state = 99)
# ->
# Treinamento do Modelo
clf.fit(X_train, y_train)
# ->
# Score

```



```

        return_train_score=True)
start = time()
random_search.fit(X, y)
print("RandomizedSearchCV executou em %.2f segundos para %d candidatos a parâmetros do modelo."
      % ((time() - start), n_iter_search))
# Imprime as combinações dos parâmetros e suas respectivas médias de acurácia
random_search.cv_results_
# ->
# Grid Search
# Usando um grid completo de todos os parâmetros
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "min_samples_leaf": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}
# Executando o Grid Search
grid_search = GridSearchCV(clf, param_grid = param_grid, return_train_score = True)
start = time()
grid_search.fit(X, y)
print("GridSearchCV executou em %.2f segundos para todas as combinações de candidatos a parâmetros do modelo."
      % (time() - start))
grid_search.cv_results_
#### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 9 - Gradient Boosting</font>
Gradient Boosting ou Gradient Boosted Regression Trees (GBRT) é uma técnica de aprendizagem estatística não-paramétrica usada para problemas de classificação e regressão.
Gradient Boosting = Gradient Descent + Boosting.
http://deeplearningbook.com.br/aprendizado-com-a-descida-do-gradiente/
Basicamente 3 etapas são realizadas na construção do modelo:
1- Gera um regressor
2- Computa o erro residual
3- Aprende a prever o resíduo
# ->
from IPython.display import Image
Image(url = 'imagens/GBRT1.png')
# ->
from IPython.display import Image
Image(url = 'imagens/GBRT2.png')
## Como funciona o Gradient Boosting:
1 - Realiza um conjunto de previsões (y)
2- Calcula o erro das previsões (j)
3- Tenta ajustar y reduzindo o erro (através de alpha)
4- Para cada estimador base, é estimado o gradiente da função de perda
5- Estimadores subsequentes estimam o erro residual dos estimadores anteriores
6- Aplica o Gradient Descent para reduzir j
7- Soma os resultados dos estimadores, dando peso a cada passo de acordo com o valor de alfa
http://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting
## Gradient Boosting Classifier
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
# ->
# Imports
from sklearn.datasets import make_hastie_10_2
from sklearn.ensemble import GradientBoostingClassifier
# Define dados para x e y
X, y = make_hastie_10_2(random_state = 0)
X_train, X_test = X[:2000], X[2000:]
y_train, y_test = y[:2000], y[2000:]
# Cria o classificador
clf = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0, random_state = 0)
# Treina o classificador
clf.fit(X_train, y_train)
# Calcula a acurácia (score)
clf.score(X_test, y_test)
## Gradient Boosting Regressor
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
# ->
# Imports
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.datasets import make_friedman1
from sklearn.ensemble import GradientBoostingRegressor
# Define dados para x e y
X, y = make_friedman1(n_samples = 1200, random_state = 0, noise = 1.0)
X_train, X_test = X[:200], X[200:]
y_train, y_test = y[:200], y[200:]

```



```
# Cria o regressor
est = GradientBoostingRegressor(n_estimators = 100, learning_rate = 0.1, max_depth = 1, random_state = 0, loss = 'ls')
# Treina o regressor
est.fit(X_train, y_train)
# Calcula o erro médio ao quadrado
mean_squared_error(y_test, est.predict(X_test))
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 9 - Gradient Boosting</font>
## Construindo Um Classificador Gradient Boosting em Python
# ->
# Imports
from sklearn.datasets import make_hastie_10_2
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
# ->
# Gerando o conjunto de dados
X, y = make_hastie_10_2(n_samples = 5000)
X_train, X_test, y_train, y_test = train_test_split(X, y)
# ->
X_train
# ->
y_train
# ->
# Cria o classificador
est = GradientBoostingClassifier(n_estimators = 200, max_depth = 3)
# ->
# Cria o modelo
est.fit(X_train, y_train)
# ->
# Previsões das classes (labels)
pred = est.predict(X_test)
# ->
# Score nos dados de teste (Acurácia)
acc = est.score(X_test, y_test)
print('Acurácia: %.4f' % acc)
# ->
# Previsão das probabilidades das classes
est.predict_proba(X_test)[0]
# ->
# Classificador - Método Ensemble (Gradient Boosting Classifier)
est
# ->
# Estimador Base
est.estimators_[0, 0]
### Parâmetros mais importantes:
Número de árvores de regressão (n_estimators)
Profundidade de cada árvore (max_depth)
loss function (loss)
## Construindo Um Regressor Gradient Boosting em Python
### Visualizando os Dados e a Linha de Regressão Proposta
# ->
import numpy as np
from sklearn.model_selection import train_test_split
%matplotlib inline
%pylab inline
FIGSIZE = (11, 7)
# Aproximação da função (linha de regressão ideal)
def reg_line(x):
    return x * np.sin(x) + np.sin(2 * x)
# Gerando dados de treino e de teste
def gen_data(n_samples = 200):
    # Gera a massa de dados aleatórios
    np.random.seed(15)
    X = np.random.uniform(0, 10, size = n_samples)[:, np.newaxis]
    y = reg_line(X.ravel()) + np.random.normal(scale = 2, size = n_samples)
    # Divide em treino e teste
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 3)
    return X_train, X_test, y_train, y_test
# Construção dos datasets
X_train, X_test, y_train, y_test = gen_data(100)
# Dados para a linha de regressão
x_plot = np.linspace(0, 10, 500)
# Plot dos dados
def plot_data(alpha = 0.4, s = 20):
    # Cria a figura
```

```

fig = plt.figure(figsize = FIGSIZE)
# Gera o plot
gt = plt.plot(x_plot, reg_line(x_plot), alpha = alpha)
# Plot dos dados de treino e de teste
plt.scatter(X_train, y_train, s = s, alpha = alpha)
plt.scatter(X_test, y_test, s = s, alpha = alpha, color = 'red')
plt.xlim((0, 10))
plt.ylabel('y')
plt.xlabel('x')
# Formatação
annotation_kw = {'xycoords': 'data', 'textcoords': 'data', 'arrowprops': {'arrowstyle': '->', 'connectionstyle': 'arc'}}
# Plot
plot_data()
# Azul - Treino
# vermelho - Teste
#### Plot de 2 Árvores com Diferentes Profundidades
# ->
from sklearn.tree import DecisionTreeRegressor
plot_data()
# Árvores de decisão com max-depth = 1
est = DecisionTreeRegressor(max_depth = 1).fit(X_train, y_train)
plt.plot(x_plot, est.predict(x_plot[:, np.newaxis]), label = 'max_depth=1', color = 'g', alpha = 0.9, linewidth = 3)
# Árvores de decisão com max-depth = 3
est = DecisionTreeRegressor(max_depth = 3).fit(X_train, y_train)
plt.plot(x_plot, est.predict(x_plot[:, np.newaxis]), label = 'max_depth=3', color = 'g', alpha = 0.7, linewidth = 1)
# Posição ds legenda
plt.legend(loc = 'upper left')
#### Aplicando o Gradient Boosting Regressor
# ->
from itertools import islice
from sklearn.ensemble import GradientBoostingRegressor
plot_data()
# Regressor GBRT
est = GradientBoostingRegressor(n_estimators = 1000, max_depth = 1, learning_rate = 1.0)
# Modelo
est.fit(X_train, y_train)
ax = plt.gca()
first = True
# Passos através das previsões à medida que adicionamos mais árvores
for pred in islice(est.staged_predict(x_plot[:, np.newaxis]), 0, est.n_estimators, 10):
    plt.plot(x_plot, pred, color = 'r', alpha = 0.2)
    if first:
        ax.annotate('Alto Viés - Baixa Variância',
                    xy = (x_plot[x_plot.shape[0] // 2], pred[x_plot.shape[0] // 2]),
                    xytext = (4, 4),
                    **annotation_kw)
        first = False
# Previsões
pred = est.predict(x_plot[:, np.newaxis])
plt.plot(x_plot, pred, color = 'r', label = 'GBRT max_depth=1')
ax.annotate('Baixo Viés - Alta Variância',
            xy = (x_plot[x_plot.shape[0] // 2], pred[x_plot.shape[0] // 2]),
            xytext = (6.25, -6),
            **annotation_kw)
# Posição da legenda
plt.legend(loc = 'upper left')
#### Diagnosticando Se o Modelo Sofre de Overfitting
# ->
def deviance_plot(est, X_test, y_test, ax=None, label="", train_color="#2c7bb6", test_color="#d7191c", alpha=1.0, ylim = (0, 10)):
    n_estimators = len(est.estimators_)
    test_dev = np.empty(n_estimators)
    for i, pred in enumerate(est.staged_predict(X_test)):
        test_dev[i] = est.loss_(y_test, pred)
    if ax is None:
        fig = plt.figure(figsize = FIGSIZE)
        ax = plt.gca()
    ax.plot(np.arange(n_estimators) + 1, test_dev, color = test_color, label = 'Teste %s' % label, linewidth = 2, alpha = alpha)
    ax.plot(np.arange(n_estimators) + 1, est.train_score_, color = train_color, label = 'Treino %s' % label, linewidth = 2, alpha = alpha)
    ax.set_ylabel('Erro')
    ax.set_xlabel('Número de Estimadores Base')
    ax.set_ylim(ylim)
    return test_dev, ax
# Aplica a função aos dados de teste para medir o overfitting do nosso modelo (est)
test_dev, ax = deviance_plot(est, X_test, y_test)
ax.legend(loc = 'upper right')
# Legendas
ax.annotate('Menor nível de erro no dataset de Teste',
            xy = (test_dev.argmax() + 1, test_dev.min() + 0.02),

```

```

        xytext = (150, 3.5),
        **annotation_kw)
ann = ax.annotate("", xy = (800, test_dev[799]), xycoords = 'data',
        xytext = (800, est.train_score_[799]), textcoords = 'data',
        arrowprops = {'arrowstyle': '<->'})
ax.text(810, 3.5, 'Gap Treino-Teste')
## Regularização (Evitar Overfitting)
1- Alterar a estrutura da árvore
2- Shrinkage
3- Stochastic Gradient Boosting
### Alterando a Estrutura da Árvore
Alterando o parâmetro min_samples_leaf garantimos um número maior de amostras por folha.
# ->
def fmt_params(params):
    return ", ".join("{}={}".format(key, val) for key, val in params.items())
fig = plt.figure(figsize = FIGSIZE)
ax = plt.gca()
for params, (test_color, train_color) in [({}, ('#d7191c', '#2c7bb6')), ({'min_samples_leaf': 3}, ('#fdae61', '#abd9e9'))]:
    est = GradientBoostingRegressor(n_estimators = 1000, max_depth = 1, learning_rate = 1.0)
    est.set_params(**params)
    est.fit(X_train, y_train)
    test_dev, ax = deviance_plot(est,
                                X_test,
                                y_test,
                                ax = ax,
                                label = fmt_params(params),
                                train_color = train_color,
                                test_color = test_color)
ax.annotate('Alto Viés', xy = (900, est.train_score_[899]), xytext= ( 600, 3), **annotation_kw)
ax.annotate('Baixa Variância', xy = (900, test_dev[899]), xytext = (600, 3.5), **annotation_kw)
plt.legend(loc = 'upper right')
### Shrinkage
Reduz o aprendizado de cada árvore reduzindo a learning_rate.
# ->
fig = plt.figure(figsize = FIGSIZE)
ax = plt.gca()
for params, (test_color, train_color) in [({}, ('#d7191c', '#2c7bb6')), ({'learning_rate': 0.1}, ('#fdae61', '#abd9e9'))]:
    est = GradientBoostingRegressor(n_estimators = 1000, max_depth = 1, learning_rate = 1.0)
    est.set_params(**params)
    est.fit(X_train, y_train)
    test_dev, ax = deviance_plot(est,
                                X_test,
                                y_test,
                                ax = ax,
                                label = fmt_params(params),
                                train_color = train_color,
                                test_color = test_color)
ax.annotate('Requer mais árvores', xy = (200, est.train_score_[199]), xytext=(300, 1.75), **annotation_kw)
ax.annotate('Menor erro no dataset de teste', xy = (900, test_dev[899]), xytext=(600, 1.75), **annotation_kw)
plt.legend(loc = 'upper right')
### Stochastic Gradient Boosting
Cria subsamples do dataset de treino antes de crescer cada árvore.
Cria subsamples dos atributos antes de encontrar o melhor split node (max_features). Funciona melhor se houver grande volume de dados.
# ->
fig = plt.figure(figsize=FIGSIZE)
ax = plt.gca()
for params, (test_color, train_color) in [({}, ('#d7191c', '#2c7bb6')), ({'learning_rate': 0.1, 'subsample': 0.7}, ('#fdae61', '#abd9e9'))]:
    est = GradientBoostingRegressor(n_estimators = 1000, max_depth = 1, learning_rate = 1.0, random_state = 1)
    est.set_params(**params)
    est.fit(X_train, y_train)
    test_dev, ax = deviance_plot(est,
                                X_test,
                                y_test,
                                ax = ax,
                                label = fmt_params(params),
                                train_color=train_color,
                                test_color=test_color)
ax.annotate('Menor Taxa de Erro no Dataset de Teste', xy = (400, test_dev[399]), xytext = (500, 3.0), **annotation_kw)
plt.legend(loc = 'upper right', fontsize='small')
### Tuning dos Hiperparâmetros com Grid Search
# ->
from sklearn.model_selection import GridSearchCV
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)
# Grid de parâmetros
param_grid = {'learning_rate': [0.1, 0.01, 0.001],
              'max_depth': [4, 5, 6],

```

```

        'min_samples_leaf': [3, 4, 5],
        'subsample': [0.3, 0.5, 0.7],
        'n_estimators': [400, 700, 1000, 2000, 3000]
    }
# Regressor
est = GradientBoostingRegressor()
# Modelo criado com GridSearchCV
gs_cv = GridSearchCV(est, param_grid, scoring =
    'neg_mean_squared_error', n_jobs = 4).fit(X_train, y_train)
# Imprime os melhores parâmetros
print('Melhores Hiperparâmetros: %r' % gs_cv.best_params_)
### Recria o Modelo com os Melhores Parâmetros
# ->
est.set_params(**gs_cv.best_params_)
est.fit(X_train, y_train)
# Plot
plot_data()
plt.plot(x_plot, est.predict(x_plot[:, np.newaxis]), color = 'r', linewidth = 2)
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

eXtreme Gradient Boosting (XGBoost) em R

```

# eXtreme Gradient Boosting (XGBoost)

# https://xgboost.readthedocs.io/en/latest/

# O XGBoost é uma biblioteca otimizada de aumento de gradiente distribuído, projetada
# para ser altamente eficiente, flexível e portátil. Ele implementa algoritmos de
# aprendizado de máquina sob a estrutura Gradient Boosting. O XGBoost fornece um
# aumento de árvore paralelo (também conhecido como GBDT, GBM) que resolve muitos
# problemas de Ciência de Dados de maneira rápida e precisa. O mesmo código é executado
# no ambiente distribuído (Hadoop, SGE, MPI) e pode resolver problemas com dados de
# bilhões de registros.

# Amplamente usado nas competições do Kaggle.

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap09/R")

# Neste exemplo, pretendemos prever se um cogumelo pode ser comido ou não!

# Pacotes
install.packages("xgboost")
install.packages("Ckmeans.1d.dp")
install.packages("DiagrammeR")
require(xgboost)
require(Ckmeans.1d.dp)
require(DiagrammeR)

# Datasets
# https://archive.ics.uci.edu/ml/datasets/mushroom
?agaricus.train
data(agaricus.train, package = 'xgboost')
data(agaricus.test, package = 'xgboost')

# Coletando subsets de treino e de teste
dados_treino <- agaricus.train
dados_teste <- agaricus.test

# Resumo dos dados
str(dados_treino)

# Visualizando as dimensões
dim(dados_treino$data)
dim(dados_teste$data)

# Visualizando os dados
View(dados_treino)
View(dados_teste)

```

```

# Classes a serem previstas
class(dados_treino$data)[1]
class(dados_treino$label)

# Construindo o modelo
?xgboost
modelo_v1 <- xgboost(data = dados_treino$data,
  label = dados_treino$label,
  max.depth = 2,
  eta = 1,
  nthread = 2,
  nround = 2,
  objective = "binary:logistic")

# Imprimindo o modelo
modelo_v1

# Matriz Densa
?xgb.DMatrix
dtrain <- xgb.DMatrix(data = dados_treino$data, label = dados_treino$label)
dtrain
class(dtrain)

# Modelo baseado em matriz densa
modelo_v2 <- xgboost(data = dtrain,
  max.depth = 2,
  eta = 1,
  nthread = 2,
  nround = 2,
  objective = "binary:logistic")

# Imprimindo o modelo
modelo_v2

# Criando um modelo e imprimindo as etapas realizadas
modelo_v3 <- xgboost(data = dtrain,
  max.depth = 2,
  eta = 1,
  nthread = 2,
  nround = 2,
  objective = "binary:logistic",
  verbose = 2)

# Imprimindo o modelo
modelo_v3

# Fazendo previsões
pred <- predict(modelo_v3, dados_teste$data)

# Tamanho do vetor de previsões
print(length(pred))

# Previsões
print(head(pred))

# Transformando as previsões em classificação
prediction <- as.numeric(pred > 0.5)
print(head(prediction))

# Erros
err <- mean(as.numeric(pred > 0.5) != dados_teste$label)
print(paste("test-error = ", err))

# Criando 2 matrizes
dtrain <- xgb.DMatrix(data = dados_treino$data, label = dados_treino$label)
dtest <- xgb.DMatrix(data = dados_teste$data, label = dados_teste$label)

# Criando uma watchlist
watchlist <- list(train = dtrain, test = dtest)
watchlist

# Criando um modeo
?xgb.train
modelo_v4 <- xgb.train(data = dtrain,
  max.depth = 2,
  eta = 1,
  nthread = 2,
  nround = 2,

```

```
        watchlist = watchlist,
        objective = "binary:logistic")

# Obtendo o label
label = getinfo(dtest, "label")

# Fazendo previsões
pred <- predict(modelo_v4, dtest)

# Erro
err <- as.numeric(sum(as.integer(pred > 0.5) != label))/length(label)
print(paste("test-error = ", err))

# Criando a Matriz de Importância de Atributos
importance_matrix <- xgb.importance(model = modelo_v4)
print(importance_matrix)

# Plot
xgb.plot.importance(importance_matrix = importance_matrix)

# Dump
xgb.dump(modelo_v4, with_stats = T)

# Plot do modelo
xgb.plot.tree(model = modelo_v4)

# Salvando o modelo
xgb.save(modelo_v4, "xgboost.model")

# Carregando o modelo treinado
bst2 <- xgb.load("xgboost.model")

# Fazendo previsões
pred2 <- predict(bst2, dados_teste$data)
pred2
```

Quizz

Para que uma decisão ocorra, o fluxo começa na raiz, que é o ponto de partida. Existem então as “condition checks” ou condições de checagem que vão determinar o próximo passo do fluxo e são chamadas de nós ou nodes. A decisão propriamente dita ocorre nas folhas.

Na construção das árvores de decisão, os atributos vão sendo agrupados em partições e as regras vão sendo identitcadas, à medida que o algoritmo aprende sobre os relacionamentos que geraram a saída apresentada nos dados de treinamento.

Poderá haver mais de uma árvore para um mesmo conjunto de dados.

Entropia é a medida da incerteza nos dados.

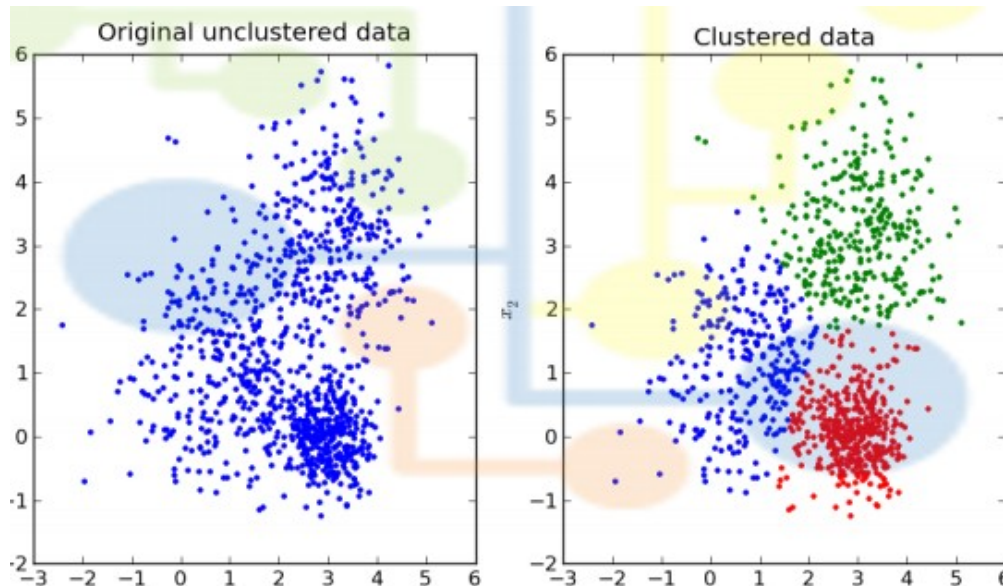
Ganho de Informação é a redução da Entropia.

7.10. Aprendizagem Não Supervisionada – Clustering

Clustering (Clusterização ou Agrupamento)

Clustering ou Agrupamento é o processo de buscar padrões nos dados, não muito diferente do que é feito pelos seres humanos.

O principal objetivo da técnica de Clustering é encontrar grupos semelhantes ou homogêneos em dados, que são chamados clusters (ou grupos).



Clustering é uma atividade de aprendizagem não supervisionada que divide automaticamente os dados em clusters ou grupos de itens semelhantes. O algoritmo faz essa classificação sem saber previamente como a classificação seria. Como nem sequer sabemos o que estamos procurando, o clustering é usado para a descoberta de conhecimento, ao invés de fazer previsões.

O agrupamento é guiado pelo princípio de que os itens dentro de um cluster devem ser muito semelhantes entre si, mas muito diferentes entre clusters.

Quando Usamos Clustering?

- Segmentação de clientes em grupos com demografia semelhante ou padrões de compra para campanhas de marketing direcionadas.
- Detecção de comportamento anômalo, como intrusões de rede não autorizadas, identificando padrões de uso fora dos clusters conhecidos.
- Simplificação de grandes conjuntos de dados agrupando características com valores semelhantes em um número menor de categorias homogêneas.

O Processo de Agrupamento

Em uma k-clusterização (como o K-Means, por exemplo), o número total de diferentes formas de agrupamento de n elementos de um conjunto em k clusters, equivale à função $N(n, k)$ abaixo:

$$N(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n$$

Outro aspecto a ser considerado em relação ao problema de clusterização é como medir o quanto um elemento é similar a outro e, assim, identificar se ambos devem estar contidos em um mesmo cluster ou não.

$$d(X_i, X_j) = \left[\sum_{l=1}^p (x_{il} - x_{jl})^2 \right]^{\frac{1}{2}}$$

Fórmula para cálculo da distância euclidiana

Quanto menor for a distância entre um par de elementos, maior é a similaridade entre eles.

Mas existem problemas de clusterização em que a distância não pode ser utilizada, ou não é conveniente que seja utilizada, como medida de similaridade, tendo em vista que os valores dos atributos não são escalares.

Os rótulos de classe obtidos de um classificador sem supervisão não possuem significado intrínseco.

Se você começar com dados sem rótulo ou seja sem variável target, você pode usar o clustering para criar rótulos de classe. A partir daí, você pode aplicar um algoritmo supervisionado, como árvores de decisão para encontrar os preditores mais importantes dessas classes! Isso é chamado de aprendizagem semi-supervisionada.

Tipos de Clustering

Clusterização:

- Hierárquica
 - Single Link
 - Complete Link
- Particional
 - Métrica Euclidiana → K-means
 - Teoria de Grafos
 - Mistura de Densidades → Expectation Maximization

Clustering Hierárquico

Agglomerative Clustering (bottom-up) | Divisive Clustering (top-down)

Para o agrupamento hierárquico, os pontos de dados reais não são necessários. Somente a matriz de medidas de distância é suficiente, pois o agrupamento é feito com base nas distâncias.

- 1- Comece com clusters como $S1 = \{X1\}$, $S2 = \{X2\}$... $S_m = \{X_m\}$.
- 2- Encontre um conjunto dos clusters mais próximos e mescle-os em um único cluster.
- 3- Repita o passo 2 até que o número de aglomerados formados seja igual a um número definido.

Clustering Particional

Uma vez que cada cluster formado é mutuamente exclusivo, nunca pode haver uma relação hierárquica entre os clusters.

Clustering Hierárquico x Clustering Particional

- Suposições Básicas
- Velocidade
- Processamento
- Parâmetros de Entrada
- Definição e Uso

A qualidade do agrupamento depende do algoritmo escolhido, da função de distância e da aplicação. Diz-se que a qualidade de um modelo de cluster é superior, quando a distância **inter-cluster** é maximizada e a distância **intra-cluster** é minimizada.

Algoritmo K-Means

O algoritmo K-Means atribui cada um dos n exemplos de dados a um dos k clusters, onde k é um número que foi determinado previamente.

- 1- O algoritmo atribui exemplos a um conjunto inicial de k clusters.
- 2- Atualiza as atribuições ajustando os limites de cluster de acordo com os exemplos que estão no cluster.

O algoritmo k-means começa por escolher k pontos no espaço de atributos para servir como centros de cluster.

K-Means++

Em 2007, foi introduzido um algoritmo chamado k-means++, que propõe um método alternativo para a seleção dos centros de cluster iniciais.

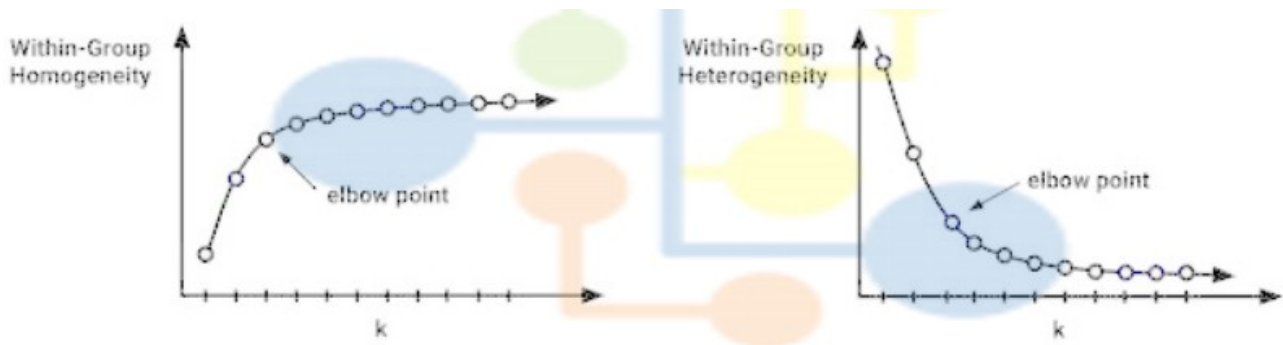
Escolhendo o Número Adequado de Clusters

Muito provavelmente, você terá um conhecimento a priori sobre os verdadeiros agrupamentos e você pode aplicar essa informação para escolher o número de clusters.

Às vezes, o número de clusters é definido por requisitos de negócios ou a motivação para a análise.

Sem qualquer conhecimento prévio, uma regra geral sugere que k seja igual à raiz quadrada de $(n / 2)$, onde n é o número de exemplos no conjunto de dados.

Método de Elbow



Lab – Segmentação de Consumidores Para Campanhas Customizadas de Marketing em R

```
# Lab - Segmentação de Consumidores Para Campanhas Customizadas de Marketing

# Objetivo: Encontrar segmentos de consumidores para campanhas personalizadas de Marketing.

# Clustering com K-Means
# https://cran.r-project.org/web/views/Cluster.html

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap10/R")

# Pacotes
install.packages("factoextra")
install.packages("cluster")
install.packages("fpc")
install.packages("NbClust")
install.packages("clValid")
install.packages("clustertend")
library(factoextra)
library(cluster)
library(fpc)
library(NbClust)
library(clValid)
library(magrittr)
library(clustertend)

# Carregando os dados
dados_clientes_v1 <- read.csv("dados/dados_clientes.csv")
str(dados_clientes_v1)
names(dados_clientes_v1)
View(dados_clientes_v1)
summary(dados_clientes_v1)

# Análise Exploratória

# Tabela de proporção do sexo dos clientes
tabela_sexos = table(dados_clientes_v1$Sexo)
tabela_sexos
table(dados_clientes_v1$Sexo, useNA = "ifany")

# Buscando valores missing para variáveis relacionadas a idade
summary(dados_clientes_v1$Idade)

# Buscando a média de idade
mean(dados_clientes_v1$Idade)

# Barplot de proporção do sexo dos clientes
barplot(tabela_sexos,
  main = "Proporção de Sexo dos Clientes",
```

```

    ylab = "Total",
    xlab = "Sexo",
    col = rainbow(2),
    legend = rownames(tabela_sexos))

# Histograma com a distribuição de frequência das idades dos clientes
hist(dados_clientes_v1$Idade,
     col = "blue",
     main = "Histogramas de Idades",
     xlab = "Idade",
     ylab = "Frequência",
     labels = TRUE)

# Boxplot para análise descritiva da idade
boxplot(dados_clientes_v1$Idade,
        col = 3,
        main = "Boxplot Para Análise Descritiva da Idade")

# Histograma com a distribuição de frequência do salário mensal
names(dados_clientes_v1)
summary(dados_clientes_v1$Salario_Mensal_Milhar)
hist(dados_clientes_v1$Salario_Mensal_Milhar,
     col = "#660033",
     main = "Histograma de Salário Mensal",
     xlab = "Salário Mensal",
     ylab = "Frequência",
     labels = TRUE)

# Análise da Pontuação dos Clientes
summary(dados_clientes_v1$Pontuacao_Gasto)

boxplot(dados_clientes_v1$Pontuacao_Gasto,
        horizontal = TRUE,
        col = "#990000",
        main = "BoxPlot Para Análise Descritiva da Pontuação do Cliente")

hist(dados_clientes_v1$Pontuacao_Gasto,
     main = "Histograma de Pontuação de Gasto",
     xlab = "Pontuação de Gasto",
     ylab = "Frequência",
     col = "#6600cc",
     labels = TRUE)

# Pré-Processamento dos Dados

# Pré-processando e analisando os dados independente do sexo
dados_clientes_v2 <- dados_clientes_v1[,-c(1,2)]
head(dados_clientes_v2)

# Padronizando as variáveis
dados_clientes_v2_scaled = scale(dados_clientes_v2)
head(dados_clientes_v2_scaled)

# Avaliando a Tendência de Cluster
# Estatística Hopkins para o conjunto de dados
# Valores > .5 significam que o dataset não é "clusterizável"
# Valores < .5 significam que o dataset é "clusterizável"
# Quanto mais próximo de zero, melhor.

set.seed(123)
?hopkins
hopkins(dados_clientes_v2_scaled, n = nrow(dados_clientes_v2_scaled)-1)

# Valor = 0.3137406 indica que o dataset é "clusterizável".

# K-means - Determinando o Número Ideal de Clusters

# Pacote NbClust: 30 índices para determinar o número de clusters em um conjunto de dados.
# Se index = 'all' - Executa 30 índices para determinar o número ideal de clusters.
# Se index = "silhouette" - Usa uma medida para estimar a diferença entre clusters.
# Um valor de silhueta mais alto é preferido para determinar o número ideal de clusters

# Opção 1:
?NbClust
num_clusters_opt1 <- NbClust(dados_clientes_v2_scaled,

```

```

        distance = "euclidean",
        min.nc = 2,
        max.nc = 15,
        method = "kmeans",
        index = "silhouette")

num_clusters_opt1$Best.nc
num_clusters_opt1$All.index

# Opção 2:
num_clusters_opt2 <- NbClust(dados_clientes_v2_scaled,
                             distance = "euclidean",
                             min.nc = 2,
                             max.nc = 15,
                             method = "kmeans",
                             index = "all")

# Um método recomenda 6 clusters e o outro recomenda 2. Usaremos 4 clusters.

# Criação do Modelo K-Means e Análise de Cluster
?kmeans
modelo <- kmeans(dados_clientes_v2_scaled, 4)
print(modelo)

# Verificando o tamanho dos clusters
modelo$size

# Verificando o centro dos clusters
modelo$centers

# Aplicando o ID dos clusters ao dataframe original
modelo$cluster
dados_clientes_v1$Cluster <- modelo$cluster
View(dados_clientes_v1)

# Média de idade por cluster
?aggregate
aggregate(data = dados_clientes_v1, Idade ~ Cluster, mean)

# Média de salário por cluster
aggregate(data = dados_clientes_v1, Salario_Mensal_Milhar ~ Cluster, mean)

# Média de pontuação de gasto por cluster
aggregate(data = dados_clientes_v1, Pontuacao_Gasto ~ Cluster, mean)

# Média de idade e salário por cluster
aggregate(data = dados_clientes_v1, cbind(Idade, Salario_Mensal_Milhar) ~ Cluster, mean)

# Total de pessoas por sexo e por cluster
with(dados_clientes_v1, table(Sexo, Cluster))

# Plot do Modelo
plot(dados_clientes_v2_scaled, col = modelo$cluster, pch = 15)

# Melhorando a Visualização
?eclust
cluster_viz <- eclust(dados_clientes_v2_scaled, "kmeans", k = 4, nstart = 25, graph = FALSE)

# Visualizando Clusters K-Means
?fviz_cluster
fviz_cluster(cluster_viz, geom = "point", ellipse.type = "norm")

# Outra opção de visualização
?clusplot
par(c(1,1))
clusplot(dados_clientes_v1, dados_clientes_v1$Cluster, color=TRUE, shade=TRUE, labels=5, lines=0)

```

Implementando o Algoritmo K-Means em Linguagem Python / Gerando Labels Para os Clusters e Identificando os Dados por Cluster / Density-Based Spatial Clustering of Applications with Noise (DBSCAN) / Análise de Cluster com Mean Shift / Mini-Projeto 1 – Agrupando Clientes Por Consumo de Energia

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 10 - K-Means</font>
http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
## Algoritmo K-Means em Python
# ->
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab
from sklearn.cluster import KMeans
from sklearn.metrics import homogeneity_completeness_v_measure
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
# ->
# Sample Generator
from sklearn.datasets.samples_generator import make_blobs
# ->
# Definindo manualmente os centros
centers = [[1, 1], [1, -1], [-1, -1], [-1, 1]]
# ->
# Gerando os dados
X, y = make_blobs(n_samples = 1000, centers = centers, cluster_std = 0.5, random_state = 101)
# ->
# Plot
plt.scatter(X[:,0], X[:,1], c = y, edgecolors = 'none', alpha = 0.9)
plt.show()
# ->
# Gera os clusters e cria o Plot dos Clusters nas Células de Voronoi
pylab.rcParams['figure.figsize'] = (10.0, 8.0)
for n_iter in range(1, 5):
    # Cria o classificador e constrói o modelo com os dados de entrada definidos nas células anteriores
    modelo = KMeans(n_clusters = 4, max_iter = n_iter, n_init = 1, init = 'random', random_state = 101)
    modelo.fit(X)
    # Plot
    plt.subplot(2, 2, n_iter)
    h = 0.02
    xx, yy = np.meshgrid(np.arange(-3, 3, h), np.arange(-3, 3, h))
    Z = modelo.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.imshow(Z,
               interpolation = 'nearest',
               cmap = plt.cm.Accent,
               extent = (xx.min(), xx.max(), yy.min(), yy.max()),
               aspect = 'auto',
               origin = 'lower')
    # Inertia = Soma das distâncias das amostras para o seu centro de agrupamento mais próximo.
    # Iteration = Número de iterações definido pelo parâmetro n_iter definido acima
    plt.scatter(X[:,0], X[:,1], c = modelo.labels_, edgecolors = 'none', alpha = 0.7)
    plt.scatter(modelo.cluster_centers_[0,0], modelo.cluster_centers_[0,1], marker = 'x', color = 'r', s = 100, linewidths = 4)
    plt.title("iteration=%s, inertia=%s" % (n_iter, int(modelo.inertia_)))
plt.show()
# ->
# Diferenças nos clusters de acordo com os valores de K
pylab.rcParams['figure.figsize'] = (10.0, 4.0)
# Gerando nova massa de dados
X, _ = make_blobs(n_samples = 1000, centers = 3, random_state = 101)
for K in [2, 3, 4]:
    # Criando o classificador e construindo o modelo
    modelo = KMeans(n_clusters = K, random_state = 101)
    y_pred = modelo.fit_predict(X)
    plt.subplot(1, 3, K-1)
    plt.title("K-means, K=%s" % K)
    plt.scatter(X[:, 0], X[:, 1], c = y_pred, edgecolors = 'none')
    plt.scatter(modelo.cluster_centers_[0,0], modelo.cluster_centers_[0,1], marker = 'x', color = 'r', s = 100, linewidths = 4)
plt.show()
## Métricas de Clusterização
```

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_completeness_v_measure.html

```
# ->
# Checando a homogeneidade dentro do cluster
pylab.rcParams['figure.figsize'] = (6.0, 4.0)
# Definindo a massa de dados
centers = [[1, 1], [1, -1], [-1, -1], [-1, 1]]
X, y = make_blobs(n_samples = 1000, centers = centers, cluster_std = 0.5, random_state = 101)
# Lista com os valores de K
valores_k = range(2, 10)
# Lista para receber as métricas
HCVs = []
# Um resultado de cluster satisfaz a homogeneidade se todos os seus clusters contiverem apenas pontos de
# dados que são membros de uma única classe.
# Um resultado de cluster satisfaz a completude se todos os pontos de dados que são membros
# de uma determinada classe são elementos do mesmo cluster.
# Ambas as pontuações têm valores positivos entre 0,0 e 1,0, sendo desejáveis valores maiores.
# Medida V é a média entre homogeneidade e completude.
for K in valores_k:
    # Criando o classificador e fazendo previsões sobre o cluster para cada ponto de dados
    y_pred = KMeans(n_clusters = K, random_state = 101).fit_predict(X)
    # Calculando as métricas
    HCVs.append(homogeneity_completeness_v_measure(y, y_pred))
plt.plot(valores_k, [el[0] for el in HCVs], 'b', label = 'Homogeneidade')
plt.plot(valores_k, [el[1] for el in HCVs], 'g', label = 'Completude')
plt.plot(valores_k, [el[2] for el in HCVs], 'r', label = 'Medida V')
plt.ylim([0, 1])
plt.xlabel("Valor de K")
plt.ylabel("Score")
plt.legend(loc = 4)
plt.show()
# ->
# Inertia = Soma das distâncias das amostras para o seu centro de agrupamento mais próximo.
# Lista de valores de K
Ks = range(2, 10)
# Lista para as métricas
valores_metrica = []
# Loop por diferentes modelos com diferentes valores de K
for K in Ks:
    modelo = KMeans(n_clusters = K, random_state = 101)
    modelo.fit(X)
    valores_metrica.append(modelo.inertia_)
plt.plot(Ks, valores_metrica, 'o-')
plt.xlabel("Valor de K")
plt.ylabel("Inertia")
plt.show()
#### Fim
#### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Data Science Academy - Machine Learning

Capítulo 10 - Gerando Labels Para os Clusters e Identificando os Dados Por Cluster

Identificando os Clusters

Na aprendizagem não supervisionada, os dados não possuem um label e cabe ao algoritmo descobrir automaticamente como os dados se agrupam. Isso é feito pelas medidas de distância dos pontos de dados para os centróides. Não há uma classificação final e cabe a você definir o que cada cluster representa, a partir da observação do resultado do algoritmo. Abaixo um exemplo de código sobre como realizar essa classificação a partir da saída do algoritmo.

```
# ->
# Imports
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import pylab as pl
# ->
# Carregando o dataset
iris = load_iris()
# ->
# Visualizando todo o conteúdo
# Perceba que os dados possuem apenas atributos de entrada e não possuem classificação como no dataset iris original
# https://archive.ics.uci.edu/ml/datasets/Iris
iris
# ->
# Visualizando o tipo de objeto dos dados
type(iris.data)
# ->
# Visualizando as 20 primeiras linhas
iris.data[1:20,]
# ->
# Criando o modelo
```

```

kmeans = KMeans(n_clusters = 3, random_state = 111)
kmeans.fit(iris.data)
# ->
kmeans.labels_
# ->
# Criando o Cluster Map
cluster_map = pd.DataFrame(iris.data)
cluster_map['cluster'] = kmeans.labels_
# ->
cluster_map
# ->
# Filtrando os Dados Pelo Cluster
cluster_map[cluster_map.cluster == 2]
Para poder visualizar os dados, vamos reduzir a dimensionalidade do dataset. O objeto pca_2d permite visualizar os dados com duas dimensões (2 componentes).
# ->
# Reduzindo a dimensionalidade
pca = PCA(n_components = 2).fit(iris.data)
# ->
# Aplicando o PCA
pca_2d = pca.transform(iris.data)
# ->
pca_2d
# ->
pca_2d.shape
# ->
pca_2d.shape[0]
# ->
# Gerando "labels" para os resultados dos clusters
for i in range(0, pca_2d.shape[0]):
    if kmeans.labels_[i] == 0:
        c1 = pl.scatter(pca_2d[i,0],pca_2d[i,1], c='r', marker='+')
    elif kmeans.labels_[i] == 1:
        c2 = pl.scatter(pca_2d[i,0],pca_2d[i,1], c='g', marker='o')
    elif kmeans.labels_[i] == 2:
        c3 = pl.scatter(pca_2d[i,0],pca_2d[i,1], c='b', marker='*')
    pl.legend([c1, c2, c3],[ 'Cluster 0', 'Cluster 1', 'Cluster 2'])
    pl.title('Clusters K-means com Iris dataset em 3 clusters')
pl.show()
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color="blue">Data Science Academy - Machine Learning</font>
# <font color="blue">Capítulo 10 - DBSCAN</font>
http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) é um algoritmo de clustering popular usado como uma alternativa ao K-Means, em análise preditiva. Ele não requer que você defina o número de clusters. Mas em troca, você tem que ajustar dois outros parâmetros. O Agrupamento Espacial Baseado em Densidade de Aplicações com Ruído (DBSCAN) é um algoritmo de agrupamento de dados proposto por Martin Ester, Hans-Peter Kriegel, Jörg Sander e Xiaowei Xu em 1996. Trata-se de um algoritmo de agrupamento baseado em densidade: dado um conjunto de pontos em algum espaço, agrupa pontos que estão intimamente próximos (pontos com muitos vizinhos próximos), e marcando como outliers pontos que estão sozinhos em regiões de baixa densidade (vizinhos que estão muito longe). DBSCAN é um dos algoritmos de agrupamento mais comuns e também mais citados na literatura científica.
Em 2014, o algoritmo foi premiado com o prêmio de teste de tempo na principal conferência de mineração de dados, KDD.
A implementação do Scikit-Learn fornece um padrão para os parâmetros eps e min_samples, mas que geralmente devem ser ajustados. O parâmetro eps é a distância máxima entre dois pontos de dados a serem considerados na mesma vizinhança. O parâmetro min_samples é a quantidade mínima de pontos de dados em um bairro (neighborhood) a ser considerado um cluster.
Uma vantagem do DBSCAN sobre o K-Means é que DBSCAN não é restrito a um número de conjuntos de clusters durante a inicialização. O algoritmo determinará um número de aglomerados com base na densidade de uma região. Tenha em mente, no entanto, que o algoritmo depende dos parâmetros eps e min_samples para descobrir qual a densidade de cada cluster.
Como o algoritmo DBSCAN tem um conceito interno de ruído, é comumente usado para detectar valores anômalos nos dados - por exemplo, atividade fraudulenta em cartões de crédito, e-commerce ou reivindicações de seguros.
# ->
# Imports
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
%matplotlib inline
# ->
# Gerando os dados
X, y = make_moons(n_samples = 200, noise = 0.05, random_state = 0)
plt.scatter(X[:,0], X[:,1])
plt.tight_layout()
plt.show()
# ->
# Construção do modelo
modelo = DBSCAN(eps = 0.2, min_samples = 5, metric = 'euclidean')
Parâmetros do DBScan
* eps, padrão = 0.5

```

A distância máxima entre duas amostras para uma ser considerada como na vizinhança da outra. Este não é um limite máximo para as distâncias de pontos dentro de um cluster. Este é o parâmetro DBSCAN mais importante para escolher adequadamente seu conjunto de dados e função de distância.

* min_samples, padrão = 5

O número de amostras (ou peso total) em uma vizinhança para um ponto a ser considerado como um ponto central. Isso inclui o próprio ponto.

```
# ->
# Fit do modelo
y_db = modelo.fit_predict(X)
# ->
# Plot
plt.scatter(X[y_db==0,0], X[y_db==0,1], c = 'blue', marker = 'o', s = 40, label = 'Cluster 1')
plt.scatter(X[y_db==1,0], X[y_db==1,1], c = 'red', marker = 's', s = 40, label = 'Cluster 2')
plt.legend()
plt.tight_layout()
plt.show()
### DBScan com Dataset Iris
# ->
# Imports
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
%matplotlib inline
# ->
# Carregando os dados
iris = load_iris()
# ->
# Primeira versão do modelo
dbscan_v1 = DBSCAN()
# ->
# Fit
dbscan_v1.fit(iris.data)
# ->
# Labels
dbscan_v1.labels_
# ->
# Reduzindo a Dimensionalidade
pca = PCA(n_components = 2).fit(iris.data)
# ->
# Fit
pca_2d = pca.transform(iris.data)
# ->
# Plot
for i in range(0, pca_2d.shape[0]):
    if dbscan_v1.labels_[i] == 0:
        c1 = plt.scatter(pca_2d[i,0], pca_2d[i,1], c = 'r', marker = '+')
    elif dbscan_v1.labels_[i] == 1:
        c2 = plt.scatter(pca_2d[i,0], pca_2d[i,1], c = 'g', marker = 'o')
    elif dbscan_v1.labels_[i] == -1:
        c3 = plt.scatter(pca_2d[i,0], pca_2d[i,1], c = 'b', marker = '*')
plt.legend([c1, c2, c3], ['Cluster 1', 'Cluster 2', 'Noise'])
plt.title('DBSCAN Gerou 2 Clusters e Encontrou Noise')
plt.show()
# ->
# Segunda versão do modelo
dbscan_v2 = DBSCAN(eps = 0.8, min_samples = 4, metric = 'euclidean')
dbscan_v2.fit(iris.data)
# ->
# Plot
for i in range(0, pca_2d.shape[0]):
    if dbscan_v2.labels_[i] == 0:
        c1 = plt.scatter(pca_2d[i,0], pca_2d[i,1], c = 'r', marker = '+')
    elif dbscan_v2.labels_[i] == 1:
        c2 = plt.scatter(pca_2d[i,0], pca_2d[i,1], c = 'g', marker = 'o')
    elif dbscan_v2.labels_[i] == -1:
        c3 = plt.scatter(pca_2d[i,0], pca_2d[i,1], c = 'b', marker = '*')
plt.legend([c1, c2, c3], ['Cluster 1', 'Cluster 2', 'Noise'])
plt.title('DBSCAN Gerou 2 Clusters e Encontrou Noise')
plt.show()
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Data Science Academy - Machine Learning

Capítulo 10 - Mean Shift

<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html>

https://docs.opencv.org/4.2.0/d7/d00/tutorial_meanshift.html

O Mean Shift é uma técnica não-paramétrica de análise de espaço de características para localizar os máximos de uma função de densidade. Pode ser usado para análise de cluster, visão computacional e processamento de imagem. Foi originalmente proposto em 1975.

Em contraste com o algoritmo K-Means, a saída do Mean Shift não depende de quaisquer suposições explícitas sobre a forma da distribuição de pontos de dados, o número de clusters ou qualquer forma de inicialização aleatória.

Uma formulação geral do algoritmo Mean Shift pode ser desenvolvida através da consideração de núcleos (Kernels) de densidade. Esta abordagem é muitas vezes referida como estimativa de densidade de kernel - um método para estimativa de densidade que muitas vezes converge mais rapidamente e que também gera uma boa estimativa contínua para a função de densidade.

Essencialmente, o Mean Shift trata o problema de agrupamento supondo que todos os pontos dados representam amostras de alguma função de densidade de probabilidade, com regiões de alta densidade de amostra correspondendo aos máximos locais desta distribuição. Para encontrar esses máximos locais, o algoritmo funciona permitindo que os pontos se atraiam, por meio do que poderia ser considerado uma espécie de "força gravitacional de curto alcance". Permitindo que os pontos gravitem para áreas de maior densidade, pode-se mostrar que eles eventualmente se unificarão em uma série de pontos, próximos aos máximos locais da distribuição. Esses pontos de dados que convergem para os mesmos máximos locais são considerados membros do mesmo cluster.

Exemplo 1

->

Imports

import numpy as np

from sklearn.cluster import MeanShift, estimate_bandwidth

from sklearn.datasets.samples_generator import make_blobs

import matplotlib.pyplot as plt

from itertools import cycle

%matplotlib inline

->

Gera massa de dados

centers = [[1, 1], [-.75, -1], [1, -1], [-3, 2]]

X, _ = make_blobs(n_samples = 10000, centers = centers, cluster_std = 0.6)

->

Cria o modelo

bandwidth = Comprimento da Interação entre os exemplos, também conhecido como a largura de banda do algoritmo.

bandwidth = estimate_bandwidth(X, quantile = .1, n_samples = 500)

Cria o modelo

modelo_v1 = MeanShift(bandwidth = bandwidth, bin_seeding = True)

Treina o modelo

modelo_v1.fit(X)

->

Coleta os labels, centróides e número de clusters

labels = modelo_v1.labels_

cluster_centers = modelo_v1.cluster_centers_

n_clusters_ = labels.max()+1

->

Plot

plt.figure(1)

plt.clf()

colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')

for k, col in zip(range(n_clusters_), colors):

my_members = labels == k

cluster_center = cluster_centers[k]

plt.plot(X[my_members, 0], X[my_members, 1], col + '.')

plt.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor = col, markeredgecolor = 'k', markersize = 14)

plt.title('Número Estimado de Clusters: %d' % n_clusters_)

plt.show()

Exemplo 2

->

Imports

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import style

from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import MeanShift

from sklearn.datasets.samples_generator import make_blobs

style.use("ggplot")

->

Gera os dados

centers = [[1,1],[5,5],[3,10]]

X, _ = make_blobs(n_samples = 500, centers = centers, cluster_std = 1)

->

Visualiza os dados

plt.scatter(X[:,0], X[:,1])

plt.show()

->

Criação do modelo

modelo_v2 = MeanShift()

->

Fit

modelo_v2.fit(X)

->

Coletando labels, centróides e número de clusters

labels = modelo_v2.labels_

cluster_centers = modelo_v2.cluster_centers_

n_clusters_ = len(np.unique(labels))

```

# ->
# Print
print(cluster_centers)
print("Número Estimado de Clusters:", n_clusters_)
# ->
# Cores
colors = 10*['r','g','b','c','k','y','m']
# ->
# Plot
for i in range(len(X)):
    plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize = 10)
plt.scatter(cluster_centers[:,0], cluster_centers[:,1], marker = "x", color = 'k', s = 150, linewidths = 5, zorder = 10)
plt.show()
# ->
# Plot 3d
# Centróides e massa de dados
centers = [[1,1,1],[5,5,5],[3,10,10]]
X, _ = make_blobs(n_samples = 500, centers = centers, cluster_std = 1.5)
# Modelo
modelo_v3 = MeanShift()
modelo_v3.fit(X)
# Extraíndo labels, centróides e número de clusters
labels = modelo_v3.labels_
cluster_centers = modelo_v3.cluster_centers_
n_clusters_ = len(np.unique(labels))
# Cores
colors = 10*['r','g','b','c','k','y','m']
# Plot
# Área de plotagem
fig = plt.figure()
# Gráfico 3d
ax = fig.add_subplot(111, projection = '3d')
# Adiciona os pontos de dados ao gráficos
for i in range(len(X)):
    ax.scatter(X[i][0], X[i][1], X[i][2], c = colors[labels[i]], marker = 'o')
# Adiciona os centróides ao gráfico
ax.scatter(cluster_centers[:,0], cluster_centers[:,1], cluster_centers[:,2],
           marker = "x",
           color = 'k',
           s = 150,
           linewidths = 5,
           zorder = 10)
plt.show()
### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 10 - Mini-Projeto - Agrupando Clientes Por Consumo de Energia</font>
### Solução
A partir de dados de consumo de energia de clientes, nosso trabalho é agrupar os consumidores por similaridade a fim de compreender o comportamento dos clientes e sua relação com o consumo de energia.
Você deve executar as seguintes tarefas:
1- Tratar os valores ausentes nos dados.
2- Coletar uma amostra de 1% dos dados para criar o modelo de clusterização com o K-Means.
3- Encontrar o melhor valor de K para esse conjunto de dados.
4- Criar o modelo K-Means usando o valor de K encontrado no item 3.
5- Criar um Meshgrid para visualização de todos os clusters.
6- Visualizar os centróides.
7- Calcular o Silhouette Score.
8 - Calcular a média de consumo de energia por cluster (usar a coluna Global_active_power para o cálculo da média).
Dica: Para o item 8, considere a amostra criada no item 2.
# ->
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from scipy.spatial.distance import cdist, pdist
from sklearn.metrics import silhouette_score
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
# ->
# Carregando os dados

```

```

dataset = pd.read_csv('dados/household_power_consumption.txt', delimiter = ';', low_memory = False)
# ->
dataset.head()
# ->
dataset.shape
# ->
dataset.dtypes
# ->
# Checando se há valores missing
dataset.isnull().values.any()
# ->
# Remove os registros com valores NA e remove as duas primeiras colunas (não são necessárias)
dataset = dataset.iloc[0:, 2:9].dropna()
# ->
dataset.head()
# ->
# Checando se há valores missing
dataset.isnull().values.any()
# ->
# Obtém os valores dos atributos
dataset_atrib = dataset.values
# ->
dataset_atrib
# ->
# Coleta uma amostra de 1% dos dados para não comprometer a memória do computador
amostra1, amostra2 = train_test_split(dataset_atrib, train_size = .01)
# ->
amostra1.shape
# ->
# Aplica redução de dimensionalidade
pca = PCA(n_components = 2).fit_transform(amostra1)
# ->
# Determinando um range de K
k_range = range(1,12)
# ->
# Aplicando o modelo K-Means para cada valor de K (esta célula pode levar bastante tempo para ser executada)
k_means_var = [KMeans(n_clusters = k).fit(pca) for k in k_range]
# ->
# Ajustando o centróide do cluster para cada modelo
centroids = [X.cluster_centers_ for X in k_means_var]
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html
# ->
# Calculando a distância euclidiana de cada ponto de dado para o centróide
k_euclid = [cdist(pca, cent, 'euclidean') for cent in centroids]
dist = [np.min(ke, axis = 1) for ke in k_euclid]
# ->
# Soma dos quadrados das distâncias dentro do cluster
soma_quadrados_intra_cluster = [sum(d**2) for d in dist]
# ->
# Soma total dos quadrados
soma_total = sum(pdist(pca)**2)/pca.shape[0]
# ->
# Soma dos quadrados entre clusters
soma_quadrados_inter_cluster = soma_total - soma_quadrados_intra_cluster
# ->
# Curva de Elbow
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(k_range, soma_quadrados_inter_cluster/soma_total * 100, 'b*-')
ax.set_ylim((0,100))
plt.grid(True)
plt.xlabel('Número de Clusters')
plt.ylabel('Percentual de Variância Explicada')
plt.title('Variância Explicada x Valor de K')
# ->
# Criando um modelo com K = 8
modelo_v1 = KMeans(n_clusters = 8)
modelo_v1.fit(pca)
# ->
# Obtém os valores mínimos e máximos e organiza o shape
x_min, x_max = pca[:, 0].min() - 5, pca[:, 0].max() - 1
y_min, y_max = pca[:, 1].min() + 1, pca[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
Z = modelo_v1.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# ->
# Plot das áreas dos clusters

```

```

plt.figure(1)
plt.clf()
plt.imshow(Z,
            interpolation = 'nearest',
            extent = (xx.min(), xx.max(), yy.min(), yy.max()),
            cmap = plt.cm.Paired,
            aspect = 'auto',
            origin = 'lower')

# ->
# Plot dos centróides
plt.plot(pca[:, 0], pca[:, 1], 'k.', markersize = 4)
centroids = modelo_v1.cluster_centers_
inert = modelo_v1.inertia_
plt.scatter(centroids[:, 0], centroids[:, 1], marker = 'x', s = 169, linewidths = 3, color = 'r', zorder = 8)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
# ->
?silhouette_score
# ->
# Silhouette Score
labels = modelo_v1.labels_
silhouette_score(pca, labels, metric = 'euclidean')
# ->
# Criando um modelo com K = 10
modelo_v2 = KMeans(n_clusters = 10)
modelo_v2.fit(pca)
# ->
# Obtém os valores mínimos e máximos e organiza o shape
x_min, x_max = pca[:, 0].min() - 5, pca[:, 0].max() + 1
y_min, y_max = pca[:, 1].min() + 1, pca[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
Z = modelo_v2.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# ->
# Plot das áreas dos clusters
plt.figure(1)
plt.clf()
plt.imshow(Z,
            interpolation = 'nearest',
            extent = (xx.min(), xx.max(), yy.min(), yy.max()),
            cmap = plt.cm.Paired,
            aspect = 'auto',
            origin = 'lower')

# ->
# Plot dos centróides
plt.plot(pca[:, 0], pca[:, 1], 'k.', markersize = 4)
centroids = modelo_v2.cluster_centers_
inert = modelo_v2.inertia_
plt.scatter(centroids[:, 0], centroids[:, 1], marker = 'x', s = 169, linewidths = 3, color = 'r', zorder = 8)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
# ->
# Silhouette Score
labels = modelo_v2.labels_
silhouette_score(pca, labels, metric = 'euclidean')
Criando o Cluster Map com os clusters do Modelo V1 que apresentou melhor Silhouette Score.
# ->
# Lista com nomes das colunas
names = ['Global_active_power', 'Global_reactive_power', 'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']
# ->
# Cria o cluster map
cluster_map = pd.DataFrame(amostra1, columns = names)
cluster_map['Global_active_power'] = pd.to_numeric(cluster_map['Global_active_power'])
cluster_map['cluster'] = modelo_v1.labels_
# ->
cluster_map
# ->
# Calcula a média de consumo de energia por cluster
cluster_map.groupby('cluster')['Global_active_power'].mean()
#### Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>

```

Algoritmos Genéricos

Os Algoritmos Genéticos (AGs) foram propostos por John H. Holland como sendo algoritmos de busca de propósito geral, com características de busca estocástica, busca de múltiplos pontos e busca paralela. Em seu trabalho, Holland estava mais interessado na evolução dos indivíduos de uma população, em uma tentativa de explicar os processos adaptativos em sistemas naturais para desenvolver sistemas artificiais baseados nestes processos, do que em resolver problemas de otimização. Atualmente, os Algoritmos Genéticos são usados em sistemas de Inteligência Artificial.

Algoritmos Genéticos são inspirados na evolução biológica, baseados na teoria da evolução de Darwin. É uma das várias técnicas da inteligência computacional de busca, muito eficiente, com interferência humana quase nula, que tem como objetivo obter soluções próximas da melhor solução possível, sendo indicadas somente para problemas difíceis como, por exemplo, os denominados NP difíceis. Algoritmos Genéticos são técnicas heurísticas, na verdade o processo nada mais é do que uma competição, onde os indivíduos mais aptos a sobrevivem. Porém pode ocorrer algo que faça com que a melhor geração desapareça como, por exemplo, uma “doença”, fazendo assim com que os indivíduos não tão aptos se tornem os melhores.

Para manter a semelhança, são usados nos sistemas artificiais, os mesmos termos que são usados na genética, portanto a estrutura corresponde a cadeias de caracteres (cromossomos), onde os caracteres (genes) situam-se em determinadas posições (locus) e com valores determinados (alelos). O indivíduo (genótipo) tem sua estrutura decodificada (fenótipo) e partir deste obtêm uma avaliação da função de desempenho. Ligados aos indivíduos estão os termos como população, reprodução, cruzamento e mutação.

O comportamento dos Algoritmos Genéticos corresponde a uma analogia com o comportamento dos indivíduos de uma população na natureza. Considerando uma população de indivíduos, estes competem entre si por diferentes recursos disponíveis no seu meio ambiente (habitat), como água, comida e abrigo. Cada um destes indivíduos possui características externas (fenótipo), relacionadas à sua constituição genética (genótipo), que os diferem entre si em relação à adaptação ao meio ambiente em que vivem. Esta adaptação afeta diretamente a capacidade de sobrevivência por período suficiente para se reproduzirem pelo acasalamento. Através do acasalamento, as características genéticas dos dois indivíduos envolvidos são combinadas e transmitidas para a prole. Dessa forma as gerações futuras possuem uma grande probabilidade de serem formadas por indivíduos com as características necessárias para um maior tempo de vida, em relação às gerações anteriores – a este processo é dado o nome de evolução natural. Para facilitar a descrição e utilização dos AGs, a terminologia utilizada na Biologia é adotada naturalmente.

O “fenótipo” de um indivíduo é obtido a partir da sua submissão a uma função que irá avaliar a qualidade do seu “código genético” e, dessa forma, corresponde às suas chances de gerar descendentes. Esta função, chamada de função de aptidão, é uma codificação da função-objetivo do problema e define a qualidade de cada indivíduo em relação ao problema modelado. Assim como na evolução natural, num Algoritmo Genético deve haver maiores chances de que os códigos genéticos dos indivíduos mais aptos sejam transmitidos para as gerações futuras através do processo seleção “natural” e reprodução.

Uma característica importante de um Algoritmo Genético é a utilização dos “operadores genéticos” sobre os indivíduos da população para que possam ser exploradas diferentes áreas do espaço de busca evitando, assim, uma convergência do algoritmo para uma solução ótima local. A combinação

entre partes do código genético de diferentes indivíduos (através do operador de cruzamento) e a realização de pequenas alterações genéticas (através do operador mutação) permitem a exploração de novas características, que podem corresponder a uma evolução dos indivíduos. Dessa forma, a população de indivíduos tende a convergir para uma combinação de características dos indivíduos que seja ideal para o problema em questão – a solução ótima. Este mecanismo de evolução natural de soluções permite que os Algoritmo Genéticos possam ser utilizados para a solução de quase todos os problemas de otimização. Beasley, Bull e Martin afirmam que “Se o AG foi implementado corretamente, a população irá evoluir ao longo de sucessivas gerações de tal forma que a aptidão do melhor indivíduo e do indivíduo médio em cada geração será incrementada em direção ao ótimo global. A convergência é a progressão em direção à uniformidade crescente. “

Em anexo você encontra um exemplo de construção do modelo de Algoritmo Genético em R.

Referências:

Algoritmos Genéticos: Estudo, novas técnicas e aplicações.

http://www.cpdee.ufmg.br/~joao/TesesOrientadas/VAS1997_1.pdf

Adaptation in Natural and Artificial Systems

<https://mitpress.mit.edu/books/adaptation-natural-and-artificial-systems>

Genetic Algorithms in Search, Optimization, and Machine Learning

https://www.amazon.com.br/Genetic-Algorithms-Optimization-Machine-Learning/dp/0201157675/ref=sr_1_1?ie=UTF8&qid=1481436333&sr=8-1&keywords=Genetic+Algorithms+in+search+%2C+optimization+and+machine+learning.+Addison-Wesley

An Overview of Genetic Algorithms: Part 2, Research Topics

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.3053&rep=rep1&type=pdf>

Avaliação de operadores de algoritmos genéticos em otimização multidimensional

<http://repositorio.unesp.br/handle/11449/88880>

```
# Algoritmos Genéticos em R

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap10/R")

# genalg: R Based Genetic Algorithm
# https://cran.r-project.org/web/packages/genalg/index.html
install.packages("genalg")
library(genalg)
library(ggplot2)

# Dataset e constraint de peso
dataset <- data.frame(item = c("canivete", "arroz", "batatas", "cebolas", "mochila", "corda", "compasso"),
                      survivalpoints = c(10, 20, 15, 2, 30, 10, 30),
                      weight = c(1, 5, 10, 1, 7, 5, 1))
weightlimit <- 20

# Antes de criar o modelo, temos de configurar uma função de avaliação.
# A função de avaliação avaliará os diferentes indivíduos (cromossomos) da população sobre o valor da sua
# configuração genética.

# Um indivíduo pode, por exemplo, ter a seguinte configuração de gene: 1001100.
```

```

# Cada número nesta cadeia binária representa se deve ou não levar um item consigo.
# Um valor de 1 refere-se a colocar o item específico na mochila enquanto um 0 se refere a deixar o item em casa.
# Dado o exemplo de configuração do gene, tomamos os seguintes itens;
chromosome = c(1, 0, 0, 1, 1, 0, 0)

# Acima, atribuímos um valor à configuração do gene de um determinado cromossomo.
# Isso é exatamente o que a função de avaliação faz.

# O algoritmo genalg tenta otimizar para o valor mínimo.
# Portanto, o valor é calculado como acima e multiplicado com -1.
# Uma configuração que excede a restrição do peso retorna um valor de 0 (um valor mais alto também pode ser atribuído).
dataset[chromosome == 1, ]

# Podemos verificar para qual quantidade de pontos de sobrevivência esta configuração atende.
cat(chromosome %*% dataset$survivalpoints)

# Função de Avaliação
evalFunc <- function(x) {
  current_solution_survivalpoints <- x %*% dataset$survivalpoints
  current_solution_weight <- x %*% dataset$weight

  if (current_solution_weight > weightlimit)
    return(0) else return(-current_solution_survivalpoints)
}

# Em seguida, escolhemos o número de iterações, projetamos e executamos o modelo.
iter = 100
GAmode <- rbgabin(size = 7, popSize = 200, iters = iter, mutationChance = 0.01, elitism = T, evalFunc = evalFunc)
cat(summary(GAmode, echo=TRUE))

# A melhor solução encontrada foi 1 1 0 1 1 1 1
# Isto nos leva a levar os seguintes itens conosco em nossa viagem para a natureza.
solution = c(1, 1, 0, 1, 1, 1, 1)
dataset[solution == 1, ]

# Isso, por sua vez, nos dá o número total de pontos de sobrevivência.
cat(paste(solution %*% dataset$survivalpoints, "/", sum(dataset$survivalpoints)))

# Plot
plot(GAmode)

```

Quiz

Algoritmos de Clustering é o tipo de algoritmo permite analisar dados e gerar regras internas para prever se um cliente X pertence a um grupo A, B ou C.

Clustering é uma atividade de aprendizagem não supervisionada que divide automaticamente os dados em clusters ou grupos de itens semelhantes.

Os métodos de cluster podem ser usados para:

- Segmentação de clientes em grupos com demografia semelhante ou padrões de compra para campanhas de marketing direcionadas.
- Detecção de comportamento anômalo, como intrusões de rede não autorizadas, identificando padrões de uso fora dos clusters conhecidos.
- Simplificação de grandes conjuntos de dados agrupando características com valores semelhantes em um número menor de categorias homogêneas.

O método de Elbow tenta avaliar como a homogeneidade ou heterogeneidade dentro dos clusters muda para vários valores de k. Dessa forma, espera-se que a homogeneidade dentro dos aglomerados diminua à medida que novos aglomerados sejam adicionados. **(FALSO!!!)**

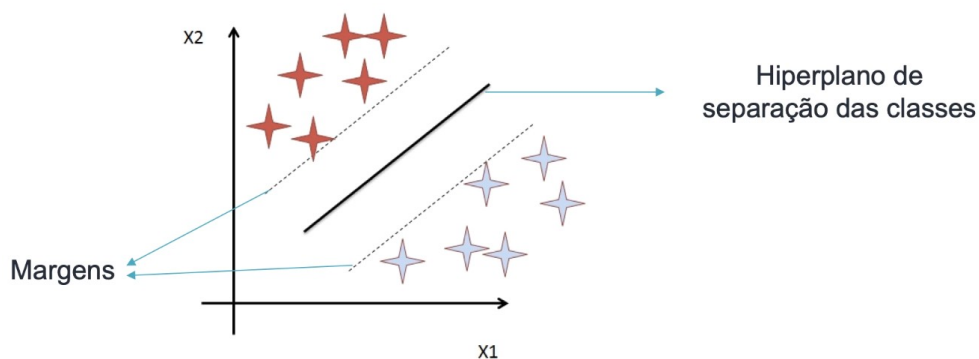
Uma vantagem do algoritmo do K-Means é UtilizaR princípios simples que podem ser explicados em termos não-estatísticos e altamente flexível.

7.11. Classificação e Regressão com Support Vector Machines (SVMs)

O Que São SVMs?

São modelos que geralmente oferecem um nível de precisão maior.

O objetivo principal do SVM é encontrar/calcular o hiperplano de separação das classes, que se encontra no meio das margens (que são os limites dos pontos de cada uma das classes).

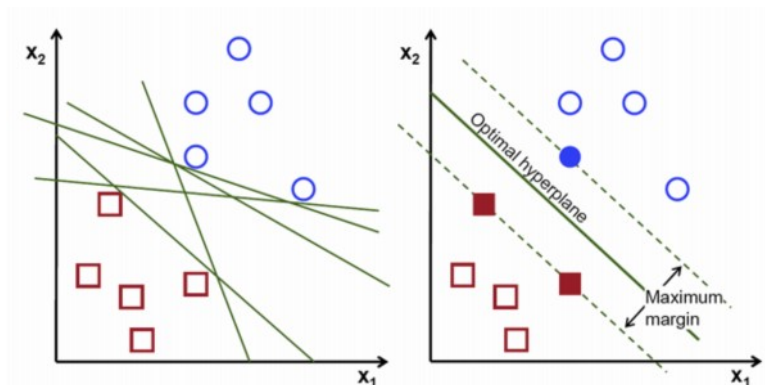


Principais Características das SVM's:

- Em caso de outliers o modelo SVM busca a melhor forma possível de classificação e, se necessário, desconsidera o outlier;
- É um classificador criado para fornecer separação linear;
- Funciona muito bem em domínios complicados, em que existe uma clara margem de separação;
- Não funciona bem em conjuntos de dados muito grandes, pois o tempo de treinamento é muito custoso;
- Não funciona bem em conjuntos de dados com grande quantidade de ruídos;
- Se as classes estiverem muito sobrepostas deve-se utilizar apenas evidências independentes.

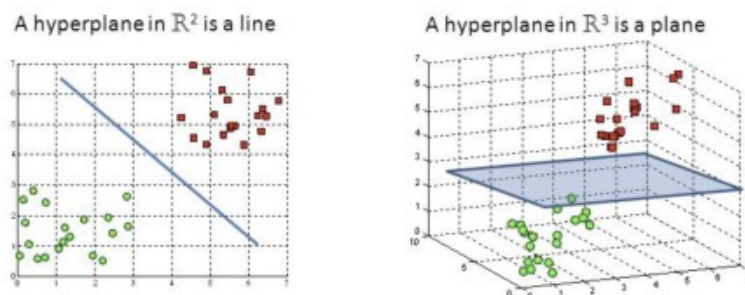
O Que São Vetores de Suporte?

O objetivo do algoritmo da máquina de vetores de suporte (SVM – Support Vector Machine) é encontrar um hiperplano em um espaço N-dimensional (N - o número de recursos ou atributos) que classifica distintamente os pontos de dados.

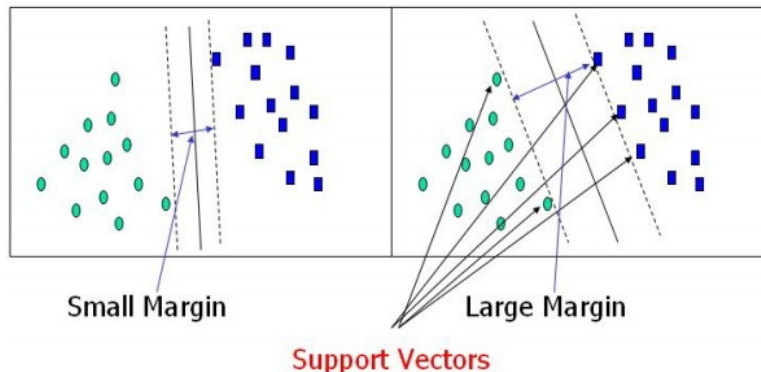


Para separar as duas classes de pontos de dados, existem muitos hiperplanos possíveis que podem ser escolhidos. Nosso objetivo é encontrar um hiperplano com a margem máxima, ou seja, a distância máxima entre os pontos de dados das duas classes. A maximização da distância da margem fornece um limite para que os pontos de dados futuros possam ser classificados com mais confiança.

Hiperplanos são limites de decisão que ajudam a classificar os pontos de dados. A dimensão do hiperplano depende do número de recursos. Se o número de recursos de entrada for 2, o hiperplano será apenas uma linha. Se o número de recursos de entrada for 3, o hiperplano se tornará um plano bidimensional. Torna-se difícil imaginar quando o número de recursos excede 3.



Os vetores de suporte são pontos de dados que estão mais próximos do hiperplano e influenciam a posição e a orientação do hiperplano. Usando esses vetores de suporte, maximizamos a margem do classificador. A exclusão dos vetores de suporte alterará a posição do hiperplano. Esses são os pontos que nos ajudam a criar nosso modelo SVM.



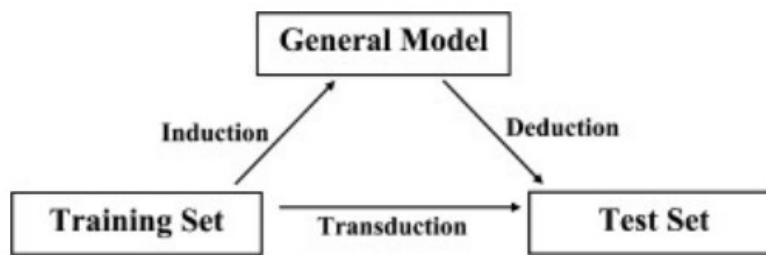
Teoria do Aprendizado Estatístico

Vamos começar fazendo uma breve revisão do processo de aprendizagem de máquina e na sequência vamos estudar a Teoria do Aprendizado Estatístico.

O processo de aprendizado de uma rede neural pode se dar de duas formas: aprendizagem supervisionada e aprendizagem não supervisionada.



As técnicas de Machine Learning empregam um princípio de inferência denominado indução, no qual obtém-se conclusões genéricas a partir de um conjunto particular de exemplos (que são os dados de treinamento). O aprendizado indutivo pode ser dividido em dois tipos principais: supervisionado e não-supervisionado.



No aprendizado supervisionado tem-se a figura de um professor externo, o qual apresenta o conhecimento do ambiente por conjuntos de exemplos na forma: dados de entrada, dados da saída desejada. O algoritmo de ML extrai a representação do conhecimento a partir desses exemplos. O objetivo é que a representação gerada seja capaz de produzir saídas corretas para novas entradas não apresentadas previamente.

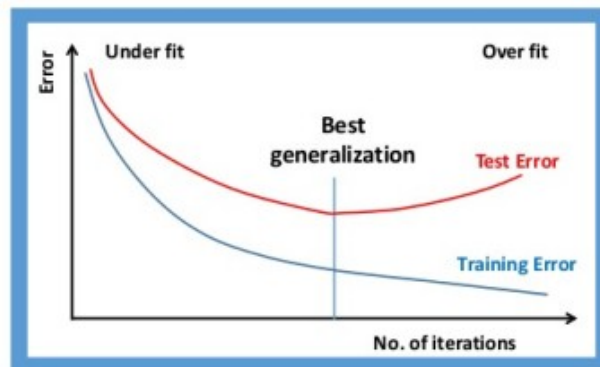
No aprendizado não-supervisionado não há a presença de um professor, ou seja, não existem exemplos rotulados. O algoritmo de Machine Learning aprende a representar (ou agrupar) as entradas submetidas segundo uma medida de qualidade. Essas técnicas são utilizadas principalmente quando o objetivo for encontrar padrões ou tendências que auxiliem no entendimento dos dados.

Um requisito importante para as técnicas de Machine Learning é que elas sejam capazes de lidar com dados imperfeitos, denominados ruídos. Muitos conjuntos de dados apresentam esse tipo de caso, sendo alguns erros comuns a presença de dados com rótulos e/ou atributos incorretos. A técnica de Machine Learning deve idealmente ser robusta a ruídos presentes nos dados, procurando não fixar a obtenção dos classificadores sobre esse tipo de caso. Deve-se também minimizar a influência de outliers no processo de indução. Os outliers são exemplos muito distintos dos demais presentes no conjunto de dados. Esses dados podem ser ruídos ou casos muito particulares, raramente presentes no domínio dos novos dados que serão apresentados ao modelo.

Os conceitos referentes à geração de um classificador a partir do aprendizado supervisionado são representados de forma simplificada neste diagrama abaixo. Tem-se nessa figura um conjunto com n dados. Cada dado x_i possui m atributos, ou seja, $x_i = (x_{i1}, \dots, x_{im})$. As variáveis y_i representam as classes. A partir dos exemplos e as suas respectivas classes, o algoritmo de ML extrai um classificador. Pode-se considerar que o modelo gerado fornece uma descrição compacta dos dados fornecidos. A obtenção de um classificador por um algoritmo de Machine Learning a partir de uma amostra de dados também pode ser considerada um processo de busca. Procura-se, entre todas as hipóteses que o algoritmo é capaz de gerar a partir dos dados, aquela com melhor capacidade de descrever o domínio em que ocorre o aprendizado. Para estimar a taxa de predições corretas ou incorretas (também denominadas taxa de acerto e taxa de erro, respectivamente) obtidas por um classificador sobre novos dados, o conjunto de exemplos é, em geral, dividido em dois subconjuntos disjuntos: de treinamento e de teste. O subconjunto de treinamento é utilizado no aprendizado do conceito e o subconjunto de teste é utilizado para medir o grau de efetividade do conceito aprendido na predição da classe de novos dados.

Outro importante conceito empregado em Machine Learning é o de generalização de um classificador, definida como a sua capacidade de prever corretamente a classe de novos dados. No caso em que o modelo se especializa nos dados utilizados em seu treinamento, apresentando uma baixa taxa de acerto quando confrontado com novos dados, tem-se a ocorrência de um superajustamento (overfitting). É também possível induzir hipóteses que apresentem uma baixa taxa de acerto mesmo no subconjunto de treinamento, configurando uma condição de subajustamento (underfitting). Essa situação pode ocorrer, por exemplo, quando os exemplos de treinamento disponíveis são pouco representativos ou quando o modelo obtido é muito simples.

Generalization



Depois desta breve revisão sobre os conceitos de Aprendizagem de máquina, vamos estudar outro importante conceito por trás de outros algoritmos de Machine Learning. Vamos começar definindo o que é a Teoria do Aprendizado Estatístico, que é a teoria por trás de vários algoritmos de Machine Learning em aprendizagem supervisionada!

Considere f um classificador e F o conjunto de todos os classificadores que um determinado algoritmo de Machine Learning pode gerar. Esse algoritmo, durante o processo de aprendizado, utiliza um conjunto de treinamento T , composto de n pares (x_i, y_i) , para gerar um classificador particular $\hat{f} \in F$.

Considere, por exemplo, o conjunto de treinamento desta figura abaixo. O objetivo do processo de aprendizado é encontrar um classificador que separe os dados das classes “círculo” e “triângulo”. As funções ou hipóteses consideradas são ilustradas na figura por meio das bordas, também denominadas fronteiras de decisão, traçadas entre as classes.



Na imagem da Figura a, tem-se uma hipótese que classifica corretamente todos os exemplos do conjunto de treinamento, incluindo dois possíveis ruídos. Por ser muito específica para o conjunto de treinamento, essa função apresenta elevada suscetibilidade a cometer erros quando confrontada

com novos dados. Esse caso representa a ocorrência de um superajustamento do modelo aos dados de treinamento (Overfitting).

Um outro classificador poderia desconsiderar pontos pertencentes a classes opostas que estejam muito próximos entre si. A ilustração da c representa essa alternativa. A nova hipótese considerada, porém, comete muitos erros, mesmo para casos que podem ser considerados simples. Tem-se assim a ocorrência de um subajustamento, pois o classificador não é capaz de se ajustar mesmo aos exemplos de treinamento (Underfitting).

Um meio termo entre as duas funções descritas é representado na figura b. Esse preditor tem complexidade intermediária e classifica corretamente grande parte dos dados, sem se fixar demasiadamente em qualquer ponto individual (Good Fit).

A Teoria de Aprendizado Estatístico estabelece condições matemáticas que auxiliam na escolha de um classificador particular \hat{f} a partir de um conjunto de dados de treinamento. Essas condições levam em conta o desempenho do classificador no conjunto de treinamento e a sua complexidade, com o objetivo de obter um bom desempenho também para novos dados do mesmo domínio. Na aplicação da Teoria de Aprendizado Estatístico, assume-se inicialmente que os dados do domínio em que o aprendizado está ocorrendo são gerados de forma independente e identicamente distribuída de acordo com uma distribuição de probabilidade $P(x, y)$, que descreve a relação entre os dados e os seus rótulos.

O erro (também denominado risco) esperado de um classificador f para dados de teste pode então ser quantificado por esta Equação. O risco esperado mede então a capacidade de generalização.

$$R(f) = \int c(f(\mathbf{x}), y) dP(\mathbf{x}, y)$$

Nesta Equação, $c(f(\mathbf{x}), y)$ é uma função de custo relacionando a previsão $f(\mathbf{x})$ quando a saída desejada é y . Essa função retorna o valor 0 se \mathbf{x} é classificado corretamente e 1 caso contrário. Infelizmente, não é possível minimizar o risco esperado, apresentado na Equação, diretamente, uma vez que em geral a distribuição de probabilidade $P(x, y)$ é desconhecida. Tem-se unicamente a informação dos dados de treinamento. Normalmente utiliza-se o princípio da indução para inferir uma função \hat{f} que minimize o erro sobre esses dados e esperase que esse procedimento leve também a um menor erro sobre os dados de teste.

O risco empírico de f , fornecido pela Equação 2, mede o desempenho do classificador nos dados de treinamento, por meio da taxa de classificações incorretas obtidas em T . Esse processo de indução com base nos dados de treinamento conhecidos, constitui o princípio de minimização do risco empírico.

$$R_{emp}(f) = \frac{1}{n} \sum_{i=1}^n c(f(\mathbf{x}_i), y_i)$$

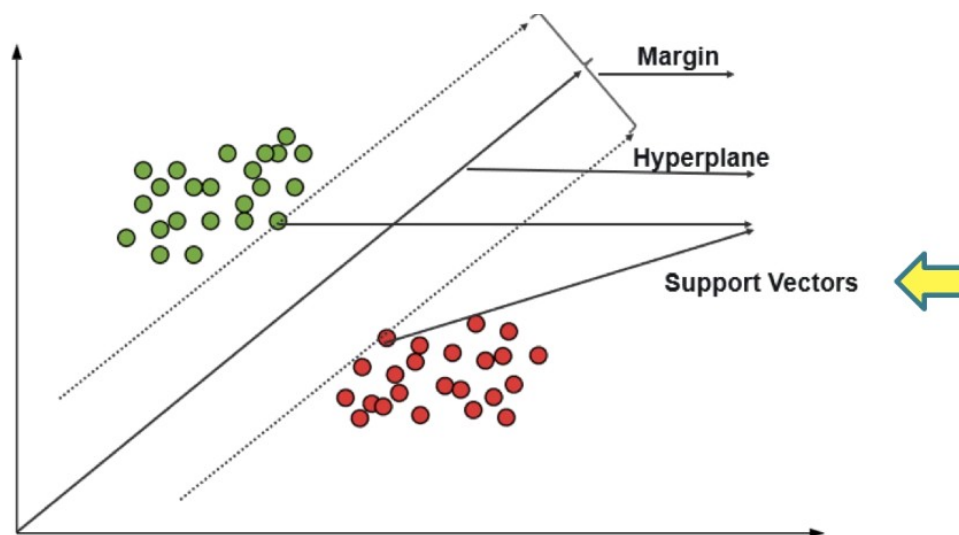
Embora a minimização do risco empírico possa levar a um menor risco esperado, nem sempre isso ocorre. Considere, por exemplo, um classificador que memoriza todos os dados de treinamento e gera classificações aleatórias para outros exemplos. Embora seu risco empírico seja nulo, seu risco esperado é 0,5. Nessa direção, a Teoria do Aprendizado Estatístico provê diversos limites no risco esperado de uma função de classificação, os quais podem ser empregados na escolha do classificador. São esses limites estabelecidos pela Teoria do Aprendizado Estatístico, sobre os quais alguns modelos SVM se baseiam.

Funcionamento das SVMs

Support Vector Machines (SVM's) são modelos de aprendizagem supervisionada, que possuem algoritmos de aprendizagem que analisam dados e reconhecem padrões, utilizados para classificação e análise de regressão.

Funcionamento do Modelo SVM para dados linearmente separáveis.

Support Vectors são os pontos de dado de cada classe que estão mais próximos um do outro.



O algoritmo recebe os dados, encontra os vetores de suporte, calcula as margens e encontra o melhor hiperplano.

Uma Dose de Matemática

O produto escalar entre dois vetores (dot product) mostra como os vetores são "semelhantes". Se os vetores representam pontos no seu conjunto de dados, o produto escalar informa se eles são semelhantes ou não.

Mas, em alguns (muitos) casos, o produto escalar não é a melhor métrica de similaridade.

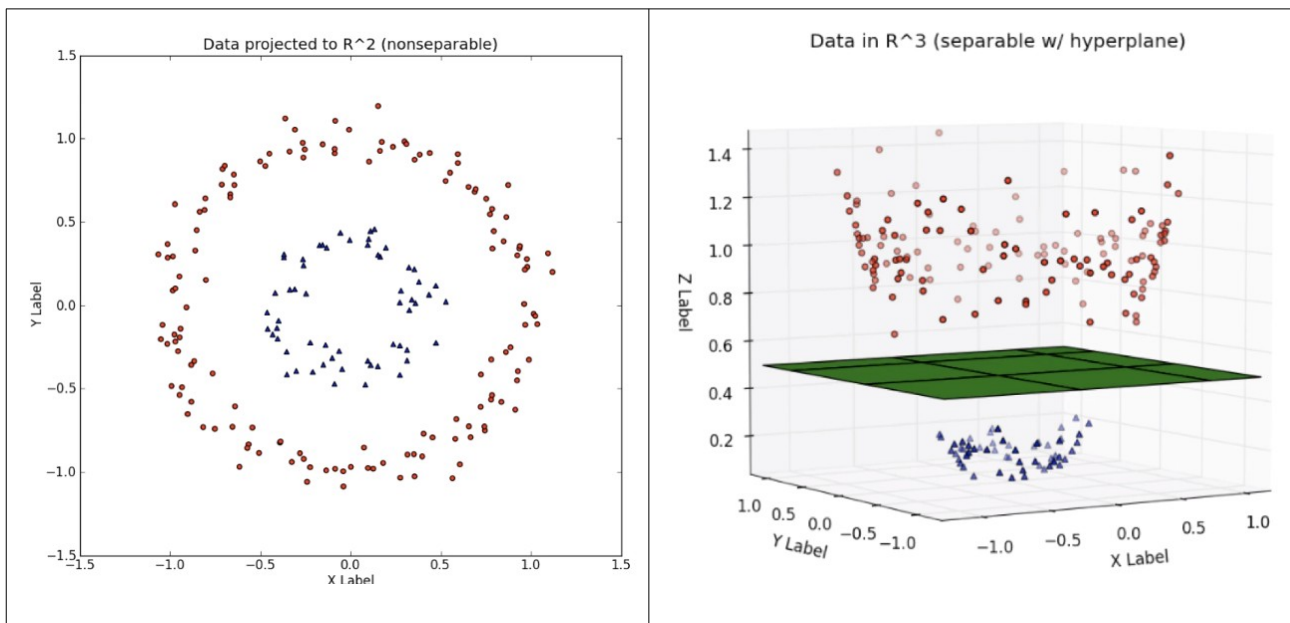
Por exemplo:

Talvez os pontos com produto escalar baixo sejam semelhantes por outras razões. Você pode ter itens de dados que não estão bem representados como pontos ou pode não haver separação linear.

Então, em vez de usar o produto escalar, você usa um "kernel", que é apenas uma função que recebe dois pontos e fornece uma medida de sua similaridade. O SVM aplica esse conceito que é denominado Truque do Kernel (Kernel Trick).

Funcionamento do Modelo SVM para dados NÃO linearmente separáveis.

A função de Kernel aplica uma operação matemática em dados não linearmente separáveis, que muda a dimensão desses dados (elevando a dimensão). A partir dessa nova dimensão é possível traçar um plano separador das classes (hiperplano).



SVM's com Margens Rígidas x SVM's com Margens Flexíveis

As máquinas de vetores de suporte (chamadas SVMs) são um algoritmo de aprendizado supervisionado que pode ser usado para problemas de classificação e regressão como classificação de vetores de suporte (SVC) e regressão de vetores de suporte (SVR).

Os pontos mais próximos ao hiperplano são chamados de pontos do vetor de suporte e a distância dos vetores do hiperplano é chamada de margem.

A intuição básica a ser desenvolvida aqui é que quanto mais pontos SV adicionais, do hiperplano, maior a probabilidade de classificar corretamente os pontos em suas respectivas regiões ou classes. Os pontos SV são muito críticos na determinação do hiperplano porque se a posição dos vetores muda, a posição do hiperplano é alterada.

Tecnicamente, esse hiperplano também pode ser chamado de hiperplano de maximização de margem.

Margens Rígidas

Se os pontos são linearmente separáveis, apenas o nosso hiperplano é capaz de distinguir entre eles e se algum erro for introduzido (outliers por exemplo), não será possível separá-los. Esse tipo de SVM é chamado SVM de Margem Rígida (já que temos restrições muito rígidas para classificar corretamente cada ponto de dados).

Margens Flexíveis

Basicamente, consideramos que os dados são linearmente separáveis e isso pode não ser o caso no cenário da vida real. Precisamos de uma atualização para que nossa função possa pular alguns valores discrepantes e poder classificar pontos quase linearmente separáveis. Por esse motivo, apresentamos uma nova variável Slack (ξ) chamada ξ_i .

Se $\xi_i = 0$, os pontos podem ser considerados corretamente classificados. Senão, se $\xi_i > 0$, pontos são classificados incorretamente.

Portanto, se $\xi_i > 0$ significa que ξ_i (variáveis) está na dimensão incorreta, podemos pensar em ξ_i como um termo de erro associado a ξ_i (variável).

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

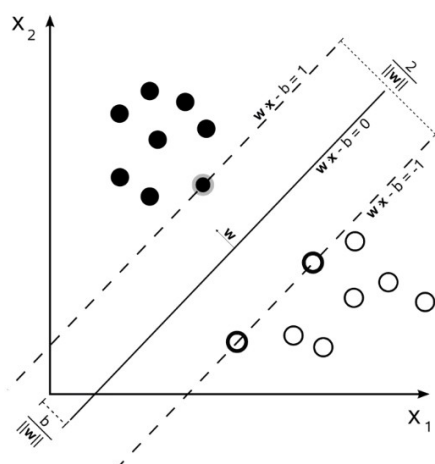
Resumindo

A margem rígida é aquela que separa claramente os pontos positivos e negativos.

A margem flexível também é chamada SVM linear “barulhenta”, pois inclui alguns pontos classificados incorretamente.

Parâmetro de Regularização C

Parâmetro de Regularização C



Distância Mínima Entre os VS:

$$\begin{aligned} \min_{w,b} \quad & \frac{\|w\|^2}{2} + C \sum_{i=1}^N \xi^{(i)}, \\ \text{s. t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi^{(i)}, \quad \forall i \in \{1, \dots, N\} \\ & \xi^{(i)} \geq 0, \quad \forall i \in \{1, \dots, N\} \end{aligned}$$

Maximizar a Distância Mínima (Otimização):

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^N \alpha^{(i)} - \sum_{i=1}^N \sum_{j=1}^N \left(y^{(i)} \alpha^{(i)} \phi(x^{(i)})^T \phi(x^{(j)}) y^{(j)} \alpha^{(j)} \right) \\ \text{s. t.} \quad & 0 \leq \alpha^{(i)} \leq C, \end{aligned}$$

O parâmetro de regularização C no Modelo SVM é responsável pelo treinamento do modelo com hiperplano de margem flexível ou rígida.

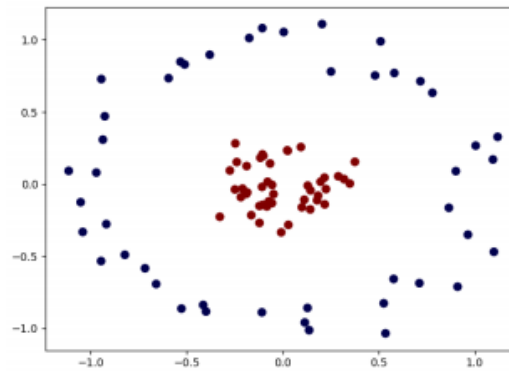
Quanto maior o valor de C menor a margem do hiperplano selecionada para o treinamento de um modelo.

Quanto menor o valor de C maior a margem do hiperplano escolhida para o treinamento de um modelo.

Para obter resultados de classificação mais precisos (menos amostras classificadas incorretamente), é necessário selecionar C com grande valor.

Tipos de Kernel

Você consegue traçar uma linha reta que divida claramente as duas classes abaixo?



Você já sabe que não! Os pontos vermelhos e azuis não podem ser separados por uma linha reta, pois estão distribuídos aleatoriamente e, na realidade, é assim que a maioria dos dados de problemas da vida real são encontrados.

No aprendizado de máquina, um "kernel" geralmente é usado para se referir ao truque do kernel, um método de usar um classificador linear para resolver um problema não linear. Isso implica transformar dados linearmente inseparáveis, em dados linearmente separáveis. A função do kernel é aplicada a cada instância de dados para mapear as observações não lineares originais em um espaço de maior dimensão no qual elas se tornam separáveis.

Em vez de definir uma série de recursos (como faz o KNN), você define uma única função do kernel para calcular a semelhança entre as classes. Você fornece esse kernel, juntamente com os dados e rótulos ao algoritmo de aprendizado, e o resultado é um classificador.

Diferentes tipos de kernels podem ser usados, tais como:

- Kernel Polinomial
- Kernel Gaussiano
- Kernel Gaussiano de Função Radial (RBF)
- Kernel Sigmoidal
- Kernel Tangente Hiperbólico
- Kernel Linear

Os kernels são funções aplicadas aos dados para permitir a separação de dados originalmente não separáveis. De todos os tipos de kernels, o RBF e o Linear são os mais importantes e aplicados em diversas situações e merecem nossa atenção nas aulas seguintes. Também é possível criar seu próprio Kernel (sua própria função de separação de dados).

Aqui você encontra a lista das funções de Kernel suportadas pelo Scikit-Learn em Python:

<https://scikit-learn.org/stable/modules/svm.html#kernel-functions>

Kernel RBF x Kernel Linear x Kernel Polinomial

As funções de Kernel são usadas para mapear o conjunto de dados original (linear / não linear) em um espaço dimensional mais alto, com vista a torná-lo linear.

A fórmula abaixo representa a definição padrão para classificar um ponto de dado (o que é usado por diversos algoritmos):

$$g(\vec{X}) = \vec{W}^T \vec{X} + b$$

Mas essa definição padrão pode não funcionar com dados linearmente inseparáveis. Nós podemos então adicionar uma função para permitir a separação dos dados, conforme mostrado na fórmula abaixo:

$$g(\vec{X}) = \vec{W} \cdot \vec{f}(\vec{X}) + b$$

A função $f(x)$ fornece superfícies de decisão não linear implícitas para os dados originais. A $f(x)$ é uma função chamada Kernel.

O Kernel Linear, Polinomial e RBF (ou Gaussiano) são simplesmente diferentes opções para estabelecer o limite de decisão do hiperplano entre as classes.

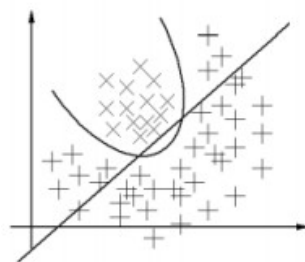
Geralmente, os Kernels Lineares e Polinomiais consomem menos tempo e fornecem menos precisão do que os Kernels RBF.

O Kernel Linear é o que você esperaria, um modelo linear. O RBF usa curvas normais em torno dos pontos de dados e as soma para que o limite de decisão possa ser definido por um tipo de condição de topologia, como curvas em que a soma está acima de um valor de 0,5.

Em termos matemáticos, nós temos:

Kernel Linear:	Kernel Polinomial	Kernel RBF:
$K(X, Y) = X^T Y$	$K(X, Y) = (\gamma \cdot X^T Y + r)^d, \gamma > 0$	$K(X, Y) = \exp(-\ X - Y\ ^2 / 2\sigma^2)$

Observe que em todos os casos o que temos é uma transformação aplicada aos dados.



Formalmente, uma "função de kernel" é qualquer função que satisfaça a condição de Mercer. Uma função, $k(x, y)$, satisfaz a condição de Mercer se, para todas as funções quadradas e integráveis $f(x)$,

$$\iint k(x, y) f(x) f(y) dx dy \geq 0$$

Esta condição é satisfeita por produtos internos (dot products):

$$\vec{W}^T \vec{X} = \sum_{d=1}^D w_d x_d$$

As funções de Kernel fornecem um espaço implícito de recurso. Isso nos permitirá usar Kernels para espaços dimensionais infinitos, além de dados não numéricos e simbólicos!

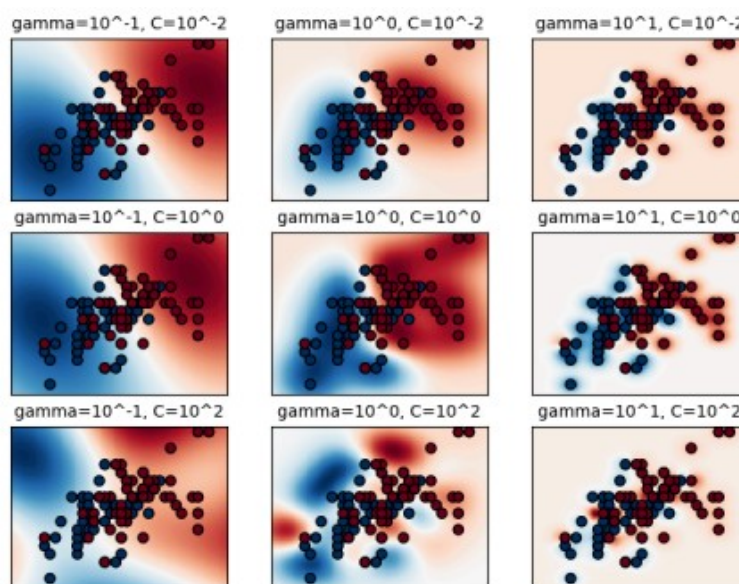
Parâmetros C, Gamma e Kernel RBF

O Kernel RBF é o mais amplamente usado ao treinar modelos SVM e dois Hiperparâmetros nos ajudam a ajustar o modelo ideal: C e gama.

Intuitivamente, o parâmetro gama define até que ponto a influência de um único exemplo de treinamento alcança, com valores baixos significando 'longe' e valores altos significando 'próximos'. Os parâmetros gama podem ser vistos como o inverso do raio de influência das amostras selecionadas pelo modelo como vetores de suporte.

O parâmetro C negocia a classificação correta dos exemplos de treinamento contra a maximização da margem da função de decisão. Para valores maiores de C, uma margem menor será aceita se a função de decisão for melhor na classificação correta de todos os pontos de treinamento. Um C mais baixo incentivar uma margem maior, portanto, uma função de decisão mais simples, ao custo da precisão do treinamento. Em outras palavras, “C” se comporta como um parâmetro de regularização no SVM.

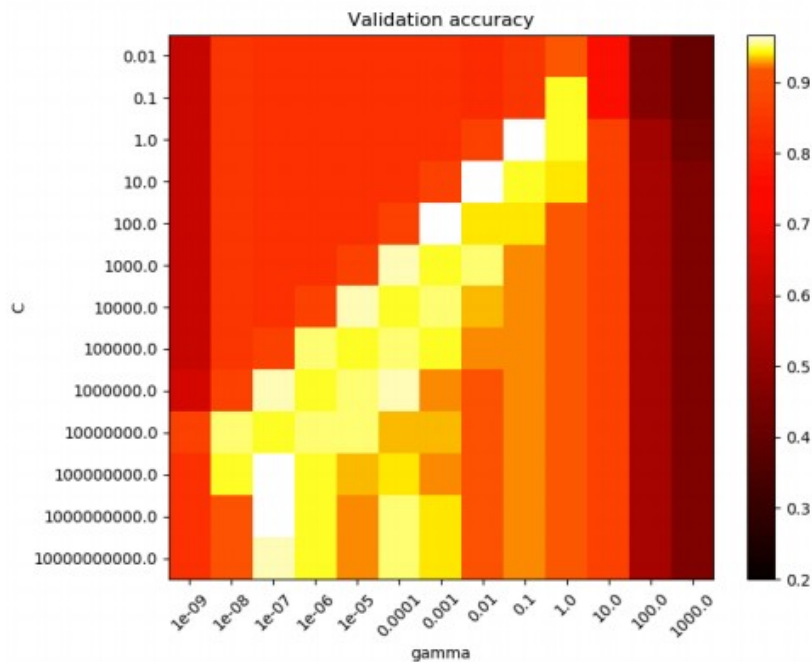
Observe os gráficos abaixo extraídos da documentação do SVM no Scikit-Learn:



O primeiro gráfico é uma visualização da função de decisão para uma variedade de valores de parâmetros em um problema de classificação simplificado, envolvendo apenas 2 recursos de entrada e 2 classes-alvo possíveis (classificação binária). Observe que esse tipo de gráfico não é possível para problemas com mais recursos ou classes de destino.

O segundo gráfico abaixo é um mapa de calor da precisão da validação cruzada do classificador em função de C e gama. Neste exemplo, exploramos uma grade relativamente grande para fins de

ilustração. Na prática, uma grade logarítmica geralmente é suficiente. Se os melhores parâmetros estiverem nos limites da grade, eles poderão ser estendidos nessa direção em uma pesquisa subsequente.



Observe que o gráfico do mapa de calor possui uma barra de cores especial com um valor de ponto médio próximo aos valores de pontuação dos modelos com melhor desempenho, para facilitar a diferenciação entre eles em um piscar de olhos.

O comportamento do modelo é muito sensível ao parâmetro gama. Se gama é muito grande, o raio da área de influência dos vetores de suporte inclui apenas o próprio vetor de suporte e nenhuma quantidade de regularização com C será capaz de impedir o ajuste excessivo.

Quando o valor de gama é muito pequeno, o modelo fica muito restrito e não pode capturar a complexidade ou a "forma" dos dados. A região de influência de qualquer vetor de suporte selecionado incluiria todo o conjunto de treinamento. O modelo resultante se comportará de maneira semelhante a um modelo linear com um conjunto de hiperplanos que separam os centros de alta densidade de qualquer par de duas classes.

Para valores intermediários, podemos ver no segundo gráfico que bons modelos podem ser encontrados na diagonal de C e gama. Modelos suaves (valores gama mais baixos) podem ser mais complexos, aumentando a importância de classificar cada ponto corretamente (valores C maiores), daí a diagonal dos modelos com bom desempenho.

Classificação Multiclasse com SVM

```
# Classificação Multiclasse com SVM - Prevendo Gastos com Cartão de Crédito em 3 Categorias
```

```
# Obs: Caso tenha problemas com a acentuação, consulte este link:
```

```
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
```

```
# Definindo o diretório de trabalho
```

```
getwd()
```

```

setwd("~/Dropbox/DSA/MachineLearning2.0/Cap11/R")

#### Definido o Problema de Negócio ####

# A identificação e a capacidade de classificar os clientes com base nos gastos sempre foram uma área de
# interesse para instituições bancárias e empresas de cartão de crédito. É um aspecto importante no
# gerenciamento de relacionamento com o cliente e ajuda a aumentar a receita com clientes existentes. Várias
# tentativas foram feitas a esse respeito. Os emissores de cartões de crédito tradicionalmente têm como alvo
# os consumidores usando informações sobre seus comportamentos e dados demográficos.

# Nosso trabalho é classificar os clientes de cartão de crédito de acordo com seu comportamento de gastos.
# A segmentação é um aspecto importante na compreensão do cliente e na execução de campanhas de marketing
# eficazes e rentáveis. Usaremos o SVM como nosso modelo.

# Os dados demográficos, os detalhes sobre emprego e o estilo de vida dos clientes desempenham um papel vital na
# maneira como eles gastam. Existem fatores ocultos, bem como semelhança com as compras. A máquina de vetores
# de suporte pode ser usada para problemas de regressão e classificação.

# Usaremos SVM com Kernel Linear Multiclasse como nosso modelo proposto para classificar a variável target.
# No entanto, também avaliaremos outros Kernels, como RBF e Polinomial, para uma variedade de hiperparâmetros.
# Também levamos em consideração o viés no dados.

# Fonte dos dados: https://sorry.vse.cz/~berka/ (dados anônimos)

# Pacotes
install.packages("gains")
install.packages("pROC")
install.packages("ROSE")
install.packages("mice")
library(dplyr)
library(caret)
library(gains)
library(pROC)
library(ROCR)
library(ROSE)
library(e1071)
library(mice)

# Carregando os dados
dataset_clientes <- read.csv("dados/cartoes_clientes.csv")
View(dataset_clientes)

#### Análise Exploratória dos Dados ####
str(dataset_clientes)
summary(dataset_clientes)
summary(dataset_clientes$card2spent)

# Removemos a variável com ID do cliente pois não é necessário
dataset_clientes <- dataset_clientes[-1]
View(dataset_clientes)

# Checando valores missing
sapply(dataset_clientes, function(x)sum(is.na(x)))

# Checando se a variável alvo está balanceada
table(dataset_clientes$Customer_cat)
prop.table(table(dataset_clientes$Customer_cat)) * 100

# Outra alternativa
as.data.frame(table(dataset_clientes$Customer_cat))

# Análise Visual

# BoxPlot e Histograma
boxplot(dataset_clientes$card2spent)
summary(dataset_clientes$card2spent)
hist(dataset_clientes$card2spent)

boxplot(dataset_clientes$hourstv)
summary(dataset_clientes$hourstv)
hist(dataset_clientes$hourstv)

# Scatter Plot
plot(dataset_clientes$card2spent, dataset_clientes$hourstv, xlab = "Gasto Cartão", ylab = "Horas TV")

# Classificação Multiclasse com SVM - Prevendo Gastos com Cartão de Crédito em 3 Categorias

```

```

# Obs: Caso tenha problemas com a acentuação, consulte este link:
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap11/R")

#### Definido o Problema de Negócio ####

# A identificação e a capacidade de classificar os clientes com base nos gastos sempre foram uma área de
# interesse para instituições bancárias e empresas de cartão de crédito. É um aspecto importante no
# gerenciamento de relacionamento com o cliente e ajuda a aumentar a receita com clientes existentes. Várias
# tentativas foram feitas a esse respeito. Os emissores de cartões de crédito tradicionalmente têm como alvo
# os consumidores usando informações sobre seus comportamentos e dados demográficos.

# Nosso trabalho é classificar os clientes de cartão de crédito de acordo com seu comportamento de gastos.
# A segmentação é um aspecto importante na compreensão do cliente e na execução de campanhas de marketing
# eficazes e rentáveis. Usaremos o SVM como nosso modelo.

# Os dados demográficos, os detalhes sobre emprego e o estilo de vida dos clientes desempenham um papel vital na
# maneira como eles gastam. Existem fatores ocultos, bem como semelhança com as compras. A máquina de vetores
# de suporte pode ser usada para problemas de regressão e classificação.

# Usaremos SVM com Kernel Linear Multiclasse como nosso modelo proposto para classificar a variável target.
# No entanto, também avaliaremos outros Kernels, como RBF e Polinomial, para uma variedade de hiperparâmetros.
# Também levamos em consideração o viés no dados.

# Fonte dos dados: https://sorry.vse.cz/~berka/ (dados anônimos)

# Pacotes
install.packages("gains")
install.packages("pROC")
install.packages("ROSE")
install.packages("mice")
library(dplyr)
library(caret)
library(gains)
library(pROC)
library(ROCR)
library(ROSE)
library(e1071)
library(mice)

# Carregando os dados
dataset_clientes <- read.csv("dados/cartoes_clientes.csv")
View(dataset_clientes)

#### Pré-Processamento dos Dados ####

# Removemos a variável com ID do cliente pois não é necessário
dataset_clientes <- dataset_clientes[-1]
View(dataset_clientes)

# Função para Fatorização de variáveis categóricas
to.factors <- function(df, variables){
  for (variable in variables){
    df[[variable]] <- as.factor(paste(df[[variable]]))
  }
  return(df)
}

# Lista de variáveis categóricas
categorical.vars <- c('townsize', 'jobcat', 'retire', 'hometype', 'addresscat',
                     'cartype', 'carvalue', 'carbought', 'card2', 'gender', 'card2type',
                     'card2benefit', 'card2benefit', 'bfast', 'internet', 'Customer_cat')

# Fatorizando as variáveis categóricas (alterando as variáveis categóricas para fatores)
str(dataset_clientes)
dataset_clientes <- to.factors(df = dataset_clientes, variables = categorical.vars)
str(dataset_clientes)
View(dataset_clientes)
str(dataset_clientes$gender)

# Aplicando Imputação em Valores Missing Usando Método PMM (Predictive Mean Matching)

```

```

# Checando valores missing
sapply(dataset_clientes, function(x)sum(is.na(x)))
sum(is.na(dataset_clientes))

# A correspondência média preditiva (PMM) é uma maneira atraente de fazer imputação múltipla para dados
# ausentes, especialmente para imputar variáveis quantitativas que não são normalmente distribuídas.

# Variável dummy
# Variável sexo = 0 ou 1
# sexo_M = 1
# sexo_F = 0

# Comparado com métodos padrão baseados em regressão linear e distribuição normal, o PMM produz valores
# imputados que são muito mais parecidos com valores reais. Se a variável original estiver inclinada, os
# valores imputados também serão inclinados. Se a variável original estiver delimitada por 0 e 100, os
# valores imputados também serão delimitados por 0 e 100. E se os valores reais forem discretos
# (como número de filhos), os valores imputados também serão discretos.

# Descobrimos os números das colunas das variáveis fatores, para excluí-las da imputação
fac_col <- as.integer(0)
facnames <- names(Filter(is.factor, dataset_clientes))
k = 1

for(i in facnames){
  while (k <= 16){
    grep(i, colnames(dataset_clientes))
    fac_col[k] <- grep(i, colnames(dataset_clientes))
    k = k + 1
    break
  }
}

# Colunas que são do tipo fator
fac_col

# Imputação

# Fatiamento do dataset
View(dataset_clientes)
View(dataset_clientes[, -c(fac_col)])

# Definindo a regra de imputação
?mice
regra_imputacao <- mice((dataset_clientes[, -c(fac_col)]),
  m = 1,
  maxit = 50,
  meth = 'pmm',
  seed = 500)

# Aplicando a regra de imputação
?mice::complete
total_data <- complete(regra_imputacao, 1)
View(total_data)

# Junta novamente as variáveis categóricas
dataset_clientes_final <- cbind(total_data, dataset_clientes[, c(fac_col)])
View(dataset_clientes_final)

# Dimensões
dim(dataset_clientes_final)

# Tipos de dados
str(dataset_clientes_final)
str(dataset_clientes_final$gender)

# Checando valores missing
sapply(dataset_clientes_final, function(x)sum(is.na(x)))
sum(is.na(dataset_clientes_final))
sum(is.na(dataset_clientes))

# Variável target como fator
dataset_clientes_final$Customer_cat <- as.factor(dataset_clientes_final$Customer_cat)
str(dataset_clientes_final$Customer_cat)

# Dividindo randomicamente o dataset em 80% para dados de treino e 20% para dados de teste

# Seed para reproduzir os mesmos resultados
set.seed(100)

```

```

# Índice de divisão dos dados
indice_divide_dados <- sample(x = nrow(dataset_clientes_final),
                             size = 0.8 * nrow(dataset_clientes_final),
                             replace = FALSE)
View(indice_divide_dados)

# Aplicando o índice
dados_treino <- dataset_clientes_final[indice_divide_dados,]
dados_teste <- dataset_clientes_final[-indice_divide_dados,]

View(dados_treino)
View(dados_teste)

# Checando o balanceamento de classe da variável target
prop.table(table(dados_treino$Customer_cat)) * 100

# Podemos ver que os dados apresentam um desequilíbrio alto com:
# 2% high_spend_cust, 30% low_spend_cust enquanto a maioria de 68% é medium_spend_cust
# Vamos balancear a classe usando Oversampling com SMOTE.

# Balanceamento de Classe com SMOTE
# Oversampling x Undersampling

# Seed
set.seed(301)

# Pacote
install.packages("DMwR")
library(DMwR)

# SMOTE - Synthetic Minority Oversampling Technique
?SMOTE
dados_treino_balanceados <- SMOTE(Customer_cat ~ ., dados_treino, perc.over = 3000, perc.under = 200)

# Checando o balanceamento de classe da variável target
prop.table(table(dados_treino_balanceados$Customer_cat)) * 100

# Salvando os datasets após o pré-processamento
class(dados_treino_balanceados)
class(dados_teste)

write.csv(dados_treino_balanceados, "dados/dados_treino_balanceados.csv")
write.csv(dados_teste, "dados/dados_teste.csv")

dim(dados_treino_balanceados)
dim(dados_teste)

View(dados_treino_balanceados)
View(dados_teste)

sum(is.na(dados_treino_balanceados))
sum(is.na(dados_teste))
sapply(dados_teste, function(x)sum(is.na(x)))

```

Classificação Multiclasse com SVM - Prevendo Gastos com Cartão de Crédito em 3 Categorias

Obs: Caso tenha problemas com a acentuação, consulte este link:
<https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding>

```

# Definindo o diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap11/R")

```

Definido o Problema de Negócio

A identificação e a capacidade de classificar os clientes com base nos gastos sempre foram uma área de interesse para instituições bancárias e empresas de cartão de crédito. É um aspecto importante no gerenciamento de relacionamento com o cliente e ajuda a aumentar a receita com clientes existentes. Várias tentativas foram feitas a esse respeito. Os emissores de cartões de crédito tradicionalmente têm como alvo os consumidores usando informações sobre seus comportamentos e dados demográficos.

Nosso trabalho é classificar os clientes de cartão de crédito de acordo com seu comportamento de gastos. A segmentação é um aspecto importante na compreensão do cliente e na execução de campanhas de marketing eficazes e rentáveis. Usaremos o SVM como nosso modelo.

Os dados demográficos, os detalhes sobre emprego e o estilo de vida dos clientes desempenham um papel vital na


```

# maneira como eles gastam. Existem fatores ocultos, bem como semelhança com as compras. A máquina de vetores
# de suporte pode ser usada para problemas de regressão e classificação.

# Usaremos SVM com Kernel Linear Multiclasse como nosso modelo proposto para classificar a variável target.
# No entanto, também avaliaremos outros Kernels, como RBF e Polinomial, para uma variedade de hiperparâmetros.
# Também levamos em consideração o viés no dados.

# Fonte dos dados: https://sorry.vse.cz/~berka/ (dados anônimos)

# Pacotes
install.packages("gains")
install.packages("pROC")
install.packages("ROSE")
install.packages("mice")
library(dplyr)
library(caret)
library(pROC)
library(e1071)
library(mice)
library(readr)

# Carregando os dados pré-processados
?read.csv
dados_treino1 <- read.csv("dados/dados_treino_balanceados.csv")
dados_teste1 <- read.csv("dados/dados_teste.csv")
dim(dados_treino1)
dim(dados_teste1)
View(dados_treino1)
View(dados_teste1)

# A função read_csv mostra o que aconteceu
?read_csv
dados_treino <- read_csv("dados/dados_treino_balanceados.csv")
dados_teste <- read_csv("dados/dados_teste.csv")
dim(dados_treino)
dim(dados_teste)
View(dados_treino)
View(dados_teste)

# Removemos a coluna X criada na indexação randômica
dados_treino <- dados_treino[-1]
dados_teste <- dados_teste[-1]

dim(dados_treino)
dim(dados_teste)

# Transformando a variável target em valor numérico
View(dados_treino)
str(dados_treino$Customer_cat)
View(dados_treino$Customer_cat)
dados_treino$Customer_cat <- as.numeric(as.factor(dados_treino$Customer_cat))
str(dados_treino$Customer_cat)
View(dados_treino$Customer_cat)
dados_treino$Customer_cat <- as.factor(dados_treino$Customer_cat)
str(dados_treino$Customer_cat)
View(dados_treino$Customer_cat)

dados_teste$Customer_cat <- as.numeric(as.factor(dados_teste$Customer_cat))
dados_teste$Customer_cat <- as.factor(dados_teste$Customer_cat)
str(dados_teste$Customer_cat)
View(dados_teste$Customer_cat)

##### Modelagem Preditiva #####

# Primeira versão do modelo SVM - Versão Padrão com Kernel Radial (RBF)
# O algoritmo escolhe o tipo de SVM de acordo com o tipo de dado da variável target
?svm
modelo_v1 <- svm(Customer_cat ~ ., data = dados_treino, na.action = na.omit, scale = TRUE)
summary(modelo_v1)
print(modelo_v1)

# Fazendo previsões com o modelo
previsoes_v1 <- predict(modelo_v1, newdata = dados_teste)

# Matriz de Confusão
?caret::confusionMatrix
caret::confusionMatrix(previsoes_v1, dados_teste$Customer_cat)

```

```

# Por que o erro aconteceu? Vamos checar! Isso chama-se troubleshooting.

# Comprimento do valor real e do valor previsto
length(dados_teste$Customer_cat)
length(previsoes_v1)

# Temos valores NA em teste?
sum(is.na(dados_teste))

# Removemos valores NA
dados_teste = na.omit(dados_teste)
length(dados_teste$Customer_cat)
sum(is.na(dados_teste))

# E agora sim a matriz de confusão
caret::confusionMatrix(previsoes_v1, dados_teste$Customer_cat)

# Métricas
install.packages("multiROC")
library(multiROC)

?multiclass.roc
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = previsoes_v1)
class(dados_teste$Customer_cat)
class(previsoes_v1)

# Faz a conversão
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v1)))

# Score AUC (Area Under The Curve)
curva_roc$auc

# Juntando valores reais e previstos na mesma tabela

# Previsões
valores_previstos <- data.frame(as.numeric(as.factor(previsoes_v1)))
colnames(valores_previstos) <- ("previsão")

# Valores reais
valores_reais <- data.frame(as.numeric(as.factor(dados_teste$Customer_cat)))
colnames(valores_reais) <- ("valor_real")

# Dataframe final
final_df <- cbind(valores_reais, valores_previstos)
View(final_df)

# Segunda versão do modelo SVM - Versão com Kernel Linear e GridSearch

# Vamos fazer uma pesquisa em grade (Grid Search) para o ajuste de hiperparâmetros e usar Kernel linear.
# Mas aqui não manteremos o custo superior a 2, para que valores discrepantes não afetem extensivamente
# a criação de limites de decisão e, portanto, levem ao ajuste excessivo (overfitting).
set.seed(182)
?tune
modelo_grid1 <- tune(svm,
  Customer_cat ~ .,
  data = dados_treino,
  kernel = 'linear',
  ranges = list(cost = c(0.05, 0.1, 0.5, 1, 2)))

summary(modelo_grid1)

# Parâmetros do melhor modelo
modelo_grid1$best.parameters

# Melhor modelo
modelo_grid1$best.model
modelo_v2 <- modelo_grid1$best.model
summary(modelo_v2)

# Previsões
previsoes_v2 <- predict(modelo_v2, dados_teste)

# Matriz de Confusão e Score AUC
confusionMatrix(previsoes_v2, dados_teste$Customer_cat)
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v2)))

```

```
curva_roc$auc
```

```
# Está oferecendo um desempenho muito melhor em termos de sensibilidade na previsão de classes minoritárias
```

```
# Terceira versão do modelo SVM - Versão com Kernel RBF e Otimização no Parâmetro Gamma
```

```
# Vamos fazer uma pesquisa em grade para o ajuste de parâmetros com kernel radial, e não manteremos  
# o custo superior a 2, para que os discrepantes não afetem extensivamente a criação de limites  
# de decisão e, portanto, levem ao ajuste excessivo.
```

```
# Da mesma forma, não manteremos um valor muito abaixo de 0,001 para gamma, pois isso levaria a  
# um excesso de ajuste.
```

```
set.seed(182)
```

```
modelo_grid2 <- tune(svm,  
  Customer_cat ~ .,  
  data = dados_treino,  
  kernel='radial',  
  ranges = list(cost = c(0.01,0.05,0.1,0.5,1,2),  
    gamma = c(0.0001,0.001,0.01,.05,0.1,0.01,1,2)))
```

```
summary(modelo_grid2)
```

```
# Parâmetros do melhor modelo
```

```
modelo_grid2$best.parameters
```

```
# Melhor modelo
```

```
modelo_v3 <- modelo_grid2$best.model
```

```
# Previsões
```

```
previsoes_v3 <- predict(modelo_v3, dados_teste)
```

```
# Matriz de Confusão e Score AUC
```

```
confusionMatrix(previsoes_v3, dados_teste$Customer_cat)
```

```
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v3)))
```

```
curva_roc$auc
```

```
# Quarta versão do modelo SVM - Versão com Kernel Polinomial
```

```
# Vamos fazer uma pesquisa em grade para ajustar parâmetros com kernel polinomial, e não manteremos  
# o custo superior a 2, para que os discrepantes não afetem extensivamente a criação de limites  
# de decisão e, portanto, levem a um excesso de ajuste.
```

```
# Da mesma forma, não manteremos o grau polinomial de ordem superior a 4, pois isso levaria a  
# um ajuste excessivo
```

```
set.seed(182)
```

```
modelo_grid3 <- tune(svm,  
  Customer_cat ~ .,  
  data = dados_treino,  
  kernel = 'polynomial',  
  ranges = list(cost = c(1,2), degree = c(2,3,4)))
```

```
summary(modelo_grid3)
```

```
# Parâmetros do melhor modelo
```

```
modelo_grid3$best.parameters
```

```
# Melhor modelo
```

```
modelo_v4 <- modelo_grid3$best.model
```

```
# Previsões
```

```
previsoes_v4 <- predict(modelo_v4, dados_teste)
```

```
# Matriz de Confusão e Score AUC
```

```
confusionMatrix(previsoes_v4, dados_teste$Customer_cat)
```

```
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v4)))
```

```
curva_roc$auc
```

```
# Podemos ver que o modelo ajustado com kernel polinomial tem uma sensibilidade fraca na  
# previsão de clientes com alto gasto para o conjunto de testes.
```

```
# Comparação dos Modelos
```

```
# Resultados do Modelo 1
```

```
resultados_v1 <- caret::confusionMatrix(previsoes_v1, dados_teste$Customer_cat)
```

```
resultados_v1$overall
```

```

resultados_v1$byClass

# Medidas Globais do Modelo 1
acuracia_v1 <- resultados_v1$overall['Accuracy']
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v1)))
score_auc_v1 <- curva_roc$auc

# Exemplo: Caso você queira outras medidas como precision e recall, lembre-se que elas são por classe
precision_v1_classe1 <- resultados_v1$byClass[1, 'Precision']
precision_v1_classe2 <- resultados_v1$byClass[2, 'Precision']
recall_v1_classe3 <- resultados_v1$byClass[3, 'Sensitivity']

# Vetor com os resultados de avaliação do Modelo v1
vetor_modelo1 <- c("Modelo1 Kernel RBF", round(acuracia_v1, 4), round(score_auc_v1, 4))

# Medidas Globais do Modelo 2
resultados_v2 <- caret::confusionMatrix(previsoes_v2, dados_teste$Customer_cat)
acuracia_v2 <- resultados_v2$overall['Accuracy']
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v2)))
score_auc_v2 <- curva_roc$auc

# Vetor com os resultados de avaliação do Modelo v2
vetor_modelo2 <- c("Modelo2 Kernel Linear", round(acuracia_v2, 4), round(score_auc_v2, 4))

# Medidas Globais do Modelo 3
resultados_v3 <- caret::confusionMatrix(previsoes_v3, dados_teste$Customer_cat)
acuracia_v3 <- resultados_v3$overall['Accuracy']
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v3)))
score_auc_v3 <- curva_roc$auc

# Vetor com os resultados de avaliação do Modelo v1
vetor_modelo3 <- c("Modelo3 Kernel RBF Tunning", round(acuracia_v3, 4), round(score_auc_v3, 4))

# Medidas Globais do Modelo 4
resultados_v4 <- caret::confusionMatrix(previsoes_v4, dados_teste$Customer_cat)
acuracia_v4 <- resultados_v4$overall['Accuracy']
curva_roc <- multiclass.roc(response = dados_teste$Customer_cat, predictor = as.numeric(as.factor(previsoes_v4)))
score_auc_v4 <- curva_roc$auc

# Vetor com os resultados de avaliação do Modelo v1
vetor_modelo4 <- c("Modelo4 Kernel Polinomial", round(acuracia_v4, 4), round(score_auc_v4, 4))

# Concatenando os resultados
# Dataframe para os resultados dos modelos
?base::rbind
compara_modelos <- rbind(vetor_modelo1, vetor_modelo2, vetor_modelo3, vetor_modelo4)
View(compara_modelos)
rownames(compara_modelos) <- c("1", "2", "3", "4")
colnames(compara_modelos) <- c("Modelo", "Acuracia", "AUC")
View(compara_modelos)
class(compara_modelos)
compara_modelos <- as.data.frame(compara_modelos)
class(compara_modelos)
View(compara_modelos)

# Plot
library(ggplot2)

# Acurácia
ggplot(compara_modelos, aes(x = Modelo, y = Acuracia, fill = Modelo)) +
  geom_bar(stat = "identity")

# AUC
ggplot(compara_modelos, aes(x = Modelo, y = AUC, fill = Modelo)) +
  geom_bar(stat = "identity")

# Assim, o método final proposto é baseado no Kernel Linear, a versão 2 do nosso modelo.

# Previsões com o Modelo Escolhido

# Salvando o modelo selecionado
?saveRDS

```

```

saveRDS(modelo_v2, "modelos/modelo_v2.rds")

# Carregando o modelo salvo
modelo_svm <- readRDS("modelos/modelo_v2.rds")
print(modelo_svm)

# Carrega o arquivo com dados de novos clientes.
# Para esses clientes não temos a variável target, pois isso é o que queremos prever.
novos_clientes <- read.csv("dados/novos_clientes.csv", header = TRUE)
View(novos_clientes)
dim(novos_clientes)

# Fazendo previsões
previsoes_novos_clientes <- predict(modelo_svm, novos_clientes)

# Apresentando o resultado final

# Previsões
previsoes_gastos_novos_clientes <- data.frame(as.numeric(as.factor(previsoes_novos_clientes)))
colnames(previsoes_gastos_novos_clientes) <- ("Previsão de Gasto")

# Idade dos clientes
idades_novos_clientes <- data.frame(novos_clientes$age)
colnames(idades_novos_clientes) <- ("Idades")

# Dataframe final
resultado_final <- cbind(idades_novos_clientes, previsoes_gastos_novos_clientes)
View(resultado_final)

# Ajusta o label da previsão
library(plyr)
?mapvalues
resultado_final$`Previsão de Gasto` <- mapvalues(resultado_final$`Previsão de Gasto`,
                                                from = c(1,2,3),
                                                to = c("Alto", "Médio", "Baixo"))

View(resultado_final)
write.csv(resultado_final, "dados/resultado_final.csv")

```

Mini-Projeto 2 – Prevendo a Intenção de Compra de Usuários de E-Commerce em Python

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 11 - Mini-Projeto</font>
### Usando SVM Para Prever a Intenção de Compra de Usuários de E-Commerce
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Definição do Problema:
Neste Mini-Projeto, nosso trabalho será avaliar quais atributos influenciam um usuário na compra de produtos online e construir um modelo
preditivo para realizar previsões de compras futuras.
Usaremos como fonte de dados o dataset:
Online Shoppers Purchasing Intention Dataset
https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset
O conjunto de dados consiste em vetores de recursos pertencentes a 12.330 sessões online. O conjunto de dados foi formado de modo que cada
sessão pertença a um usuário diferente em um período de 1 ano para evitar qualquer tendência para uma campanha específica, dia especial, usuário,
perfil ou período.
O conjunto de dados consiste em 10 atributos numéricos e 8 categóricos. O atributo 'Revenue' pode ser usado como o rótulo da classe.
## Importando os Pacotes
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# ->
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Imports
import time

```

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_auc_score
from sklearn import svm
import sklearn
import matplotlib
import warnings
warnings.filterwarnings('ignore')
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversion
## Carga e Dicionário de Dados
# ->
# Carregando os dados
df_original = pd.read_csv('dados/online_shoppers_intention.csv')
df_original.head()
Dicionário de Dados:
"Administrativo", "Duração administrativa", "Informativo", "Duração informativo", "Relacionado ao produto" e "Duração relacionada ao produto"
representam o número de diferentes tipos de páginas visitadas pelo visitante nessa sessão e o tempo total gasto em cada uma dessas categorias de
página. Os valores desses recursos são derivados das informações de URL das páginas visitadas pelo usuário e atualizadas em tempo real quando
um usuário executa uma ação, por exemplo, passando de uma página para outra.
Os recursos "Taxa de rejeição", "Taxa de saída" e "Valor da página" representam as métricas medidas pelo "Google Analytics" para cada página no
site de comércio eletrônico.
O valor do recurso "Taxa de rejeição" de uma página da web refere-se à porcentagem de visitantes que entram no site a partir dessa página e saem
("rejeição") sem acionar outras solicitações ao servidor durante essa sessão.
O valor do recurso "Taxa de saída" para uma página da web específica é calculado como a porcentagem que foi a última na sessão, para todas as
exibições de página a página.
O recurso "Valor da página" representa o valor médio para uma página da web que um usuário visitou antes de concluir uma transação de comércio
eletrônico.
O recurso "Dia especial" indica a proximidade do horário de visita do site a um dia especial específico (por exemplo, dia das mães, dia dos
namorados) em que as sessões têm mais probabilidade de serem finalizadas com a transação. O valor desse atributo é determinado considerando a
dinâmica do comércio eletrônico, como a duração entre a data do pedido e a data de entrega. Por exemplo, no dia dos namorados, esse valor assume
um valor diferente de zero entre 2 e 12 de fevereiro (dia dos namorados nos EUA e Europa), zero antes e depois dessa data, a menos que esteja
próximo de outro dia especial e seu valor máximo de 1 em 8 de fevereiro.
O conjunto de dados também inclui o tipo de sistema operacional, navegador, região, tipo de tráfego, tipo de visitante como visitante novo ou
recorrente, um valor booleano indicando se a data da visita é final de semana e mês do ano.
A variável alvo (Revenue) é booleana, com True se a sessão gerou receita e False se não gerou.
## Análise Exploratória
# ->
# Shape
df_original.shape
# ->
# Tipos de Dados
df_original.dtypes
# ->
# Verificando valores missing
print(df_original.isna().sum())
# ->
# Removendo as linhas com valores missing
df_original.dropna(inplace = True)
# ->
# Verificando valores missing
print(df_original.isna().sum())
# ->
# Shape
df_original.shape
# ->
# Verificando Valores Únicos
df_original.nunique()
Para fins de visualização, dividiremos os dados em variáveis contínuas e categóricas. Trataremos todas as variáveis com menos de 30 entradas
únicas como categóricas.
# ->
# Preparando os dados para o plot
# Cria uma cópia do dataset original
df = df_original.copy()
# Listas vazias para os resultados
continuous = []
categorical = []
# Loop pelas colunas
for c in df.columns[:-1]:
    if df.nunique()[c] >= 30:

```

```

        continuous.append(c)
    else:
        categorical.append(c)
# ->
continuous
# ->
# Variáveis contínuas
df[continuous].head()
# ->
# Variáveis categóricas
df[categorical].head()
Gráficos para variáveis numéricas.
# ->
# Plot das variáveis contínuas
# Tamanho da área de plotagem
fig = plt.figure(figsize = (12,8))
# Loop pelas variáveis contínuas
for i, col in enumerate(continuous):
    plt.subplot(3, 3, i + 1);
    df.boxplot(col);
    plt.tight_layout()
plt.savefig('imagens/boxplot1.png')
Variáveis contínuas parecem extremamente distorcidas. Vamos aplicar transformação de log para melhor visualização.
# ->
# Transformação de log nas variáveis contínuas
df[continuous] = np.log1p(1 + df[continuous])
# ->
# Plot das variáveis contínuas
# Tamanho da área de plotagem
fig = plt.figure(figsize = (12,8))
# Loop pelas variáveis contínuas
for i,col in enumerate(continuous):
    plt.subplot(3,3,i+1);
    df.boxplot(col);
    plt.tight_layout()
plt.savefig('imagens/boxplot2.png')
Matriz de Correlação Entre Variáveis Contínuas.
# ->
# Área de plotagem
plt.figure(figsize = (10,10))
# Matriz de Correlação
sns.heatmap(df[['Administrative_Duration',
                'Informational_Duration',
                'ProductRelated_Duration',
                'BounceRates',
                'ExitRates',
                'PageValues',
                'Revenue']].corr(), vmax = 1., square = True)
Visualização de gráficos de variáveis categóricas para analisar como a variável de destino é influenciada por elas.
# ->
# Countplot Venda ou Não
plt.subplot(1,2,2)
plt.title("Venda ou Não")
sns.countplot(df['Revenue'])
# ->
# Countplot Tipo de Visitante
plt.xlabel("Tipo de Visitante")
sns.countplot(df['VisitorType'])
# ->
# Stacked Bar Tipo de Visitante x Revenue
pd.crosstab(df['VisitorType'], df['Revenue']).plot(kind = 'bar',
                                                    stacked = True,
                                                    figsize = (15, 5),
                                                    color = ['red', 'green'])
# ->
# Pie Chart Tipos de Visitantes
labels = ['Visitante_Retornando', 'Novo_Visitante', 'Outro']
plt.title("Tipos de Visitantes")
plt.pie(df['VisitorType'].value_counts(), labels = labels, autopct = '%.2f%%')
plt.legend()
# ->
# Countplot Final de Semana ou Não
plt.subplot(1,2,1)
plt.title("Final de Semana ou Não")
sns.countplot(df['Weekend'])
# ->
# Stacked Bar Final de Semana x Revenue
pd.crosstab(df['Weekend'], df['Revenue']).plot(kind = 'bar',

```

```

        stacked = True,
        figsize = (15, 5),
        color = ['red', 'green'])

# ->
# Countplot Tipos de Sistemas Operacionais
plt.figure(figsize = (15,6))
plt.title("Tipos de Sistemas Operacionais")
plt.xlabel("Sistema Operacional Usado")
sns.countplot(df['OperatingSystems'])
# ->
# Stacked Bar Tipo de SO x Revenue
pd.crosstab(df['OperatingSystems'], df['Revenue']).plot(kind = 'bar',
        stacked = True,
        figsize = (15, 5),
        color = ['red', 'green'])

# ->
# Countplot Tipo de Tráfego
plt.title("Tipos de Tráfego")
plt.xlabel("Tipo de Tráfego")
sns.countplot(df['TrafficType'])
# ->
# Stacked Bar Tipos de Tráfego x Revenue
pd.crosstab(df['TrafficType'], df['Revenue']).plot(kind = 'bar',
        stacked = True,
        figsize = (15, 5),
        color = ['red', 'green'])

## Pré-Processamento dos Dados
# ->
df_original.head()
# ->
# Cria o encoder
lb = LabelEncoder()
# Aplica o encoder nas variáveis que estão com string
df_original['Month'] = lb.fit_transform(df_original['Month'])
df_original['VisitorType'] = lb.fit_transform(df_original['VisitorType'])
# Remove valores missing eventualmente gerados
df_original.dropna(inplace = True)
# ->
df_original.head()
# ->
# Shape
df_original.shape
# ->
# Verificando se a variável resposta está balanceada
target_count = df_original.Revenue.value_counts()
target_count
# ->
# Plot
sns.countplot(df_original.Revenue, palette = "OrRd")
plt.box(False)
plt.xlabel('Receita (Revenue) Por Sessão Não
(0) / Sim (1)', fontsize = 11)
plt.ylabel('Total Sessões', fontsize = 11)
plt.title('Contagem de Classes\n')
plt.show()
# ->
# Instala e importa o pacote imblearn
!pip install -q imblearn
import imblearn
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
# ->
# Shape
df_original.shape
# ->
# Variáveis explicativas
df_original.iloc[:, 0:17].head()
# ->
# Variável Target
df_original.iloc[:, 17].head()
Balanceamento de Classe - Oversampling
# ->
# Importa a função
from imblearn.over_sampling import SMOTE
# Seed para reproduzir o mesmo resultado
seed = 100

```



```

# Separa X e y
X = df_original.iloc[:, 0:17]
y = df_original.iloc[:, 17]
# Cria o balanceador SMOTE
smote_bal = SMOTE(random_state = seed)
# Aplica o balanceador
X_res, y_res = smote_bal.fit_resample(X, y)
# ->
# Plot
sns.countplot(y_res, palette = "OrRd")
plt.box(False)
plt.xlabel('Receita (Revenue) Por Sessão Não (0) / Sim (1)', fontsize = 11)
plt.ylabel('Total Sessões', fontsize = 11)
plt.title('Contagem de Classes\n')
plt.show()
# ->
# Shape dos dados originais
df_original.shape
# ->
# Shape dos dados reamostrados
X_res.shape
# ->
# Shape dos dados reamostrados
y_res.shape
# ->
# Ajustando X e y
X = X_res
y = y_res
# ->
# Divisão em Dados de Treino e Teste.
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size = 0.3, random_state = 42)
## Modelo SVM
### Modelo Base com Kernel Linear
# ->
# Cria o modelo
modelo_v1 = svm.SVC(kernel = 'linear')
# ->
# Treinamento
start = time.time()
modelo_v1.fit(X_treino, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo:', end - start)
# ->
# Previsões
previsoes_v1 = modelo_v1.predict(X_teste)
# ->
# Dicionário de métricas e metadados
SVM_dict_v1 = {'Modelo': 'SVM',
               'Versão': '1',
               'Kernel': 'Linear',
               'Precision': precision_score(previsoes_v1, y_teste),
               'Recall': recall_score(previsoes_v1, y_teste),
               'F1 Score': f1_score(previsoes_v1, y_teste),
               'Acurácia': accuracy_score(previsoes_v1, y_teste),
               'AUC': roc_auc_score(y_teste, previsoes_v1)}

# ->
# Print
print("Métricas em Teste:\n")
SVM_dict_v1
### Modelo com Kernel Linear e Dados Padronizados (Scaled)
# ->
# ***** Atenção *****
# O método nesta célula não deve ser usado, pois estaríamos aplicando o fit em teste e isso não é o ideal
# Aplicamos o fit somente nos dados de treino e aplicamos o transform nos dados de teste
# Padronização
# X_treino_scaled = StandardScaler().fit_transform(X_treino)
# X_teste_scaled = StandardScaler().fit_transform(X_teste)
# ->
# Agora sim, a forma ideal de aplicar a padronização em treino e teste
# Padronização
sc = StandardScaler()
X_treino_scaled = sc.fit_transform(X_treino)
X_teste_scaled = sc.transform(X_teste)
Obsevação:
Para impedir que as informações sobre a distribuição do conjunto de teste vazem em seu modelo, o ideal é aplicar a padronização em separado nos dados de treino e de teste, ajustando o redimensionador apenas aos dados de treinamento, padronizando então os conjuntos de treinamento e teste com esse redimensionador (exatamente como está na célula acima). Ao ajustar o redimensionador no conjunto de dados completo antes da divisão em treino e teste, informações sobre o conjunto de testes são usadas para transformar o conjunto de treinamento.

```

Conhecer a distribuição de todo o conjunto de dados pode influenciar como você detecta e processa outliers, bem como como você parametriza seu modelo. Embora os dados em si não sejam expostos, há informações sobre a distribuição dos dados. Como resultado, o desempenho do seu conjunto de testes não é uma estimativa real do desempenho em dados invisíveis.

Sempre aplique a padronização depois de fazer a divisão em treino e teste, exatamente como fizemos aqui. Usamos `fit_transform()` nos dados de treino e `transform()` nos dados de teste quando usamos o `StandardScaler()`.

```
# ->
X_treino_scaled
# ->
X_teste_scaled
# ->
# Cria o modelo
modelo_v2 = svm.SVC(kernel = 'linear')
# ->
# Treinamento
start = time.time()
modelo_v2.fit(X_treino_scaled, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo:', end - start)
# ->
# Previsões
previsoes_v2 = modelo_v2.predict(X_teste_scaled)
# ->
# Dicionário de métricas e metadados
SVM_dict_v2 = {'Modelo': 'SVM',
               'Versão': '2',
               'Kernel': 'Linear com Dados Padronizados',
               'Precision': precision_score(previsoes_v2, y_teste),
               'Recall': recall_score(previsoes_v2, y_teste),
               'F1 Score': f1_score(previsoes_v2, y_teste),
               'Acurácia': accuracy_score(previsoes_v2, y_teste),
               'AUC': roc_auc_score(y_teste, previsoes_v2)}

# ->
# Print
print("Métricas em Teste:\n")
SVM_dict_v2
#### Otimização de Hiperparâmetros com Grid Search e Kernel RBF
# ->
# Cria o modelo
modelo_v3 = svm.SVC(kernel = 'rbf')
# Valores para o grid
C_range = np.array([50., 100., 200.])
gamma_range = np.array([0.3*0.001, 0.001, 3*0.001])
# Grid de hiperparâmetros
svm_param_grid = dict(gamma = gamma_range, C = C_range)
# Grid Search
start = time.time()
modelo_v3_grid_search_rbf = GridSearchCV(modelo_v3, svm_param_grid, cv = 3)
# Treinamento
modelo_v3_grid_search_rbf.fit(X_treino_scaled, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo com Grid Search:', end - start)
# Acurácia em Treino
print(f"Acurácia em Treinamento: {modelo_v3_grid_search_rbf.best_score_ :.2%}")
print("")
print(f"Hiperparâmetros Ideais: {modelo_v3_grid_search_rbf.best_params_}")
# ->
# Previsões
previsoes_v3 = modelo_v3_grid_search_rbf.predict(X_teste_scaled)
# ->
# Dicionário de métricas e metadados
SVM_dict_v3 = {'Modelo': 'SVM',
               'Versão': '3',
               'Kernel': 'RBF com Dados Padronizados',
               'Precision': precision_score(previsoes_v3, y_teste),
               'Recall': recall_score(previsoes_v3, y_teste),
               'F1 Score': f1_score(previsoes_v3, y_teste),
               'Acurácia': accuracy_score(previsoes_v3, y_teste),
               'AUC': roc_auc_score(y_teste, previsoes_v3)}

# ->
# Print
print("Métricas em Teste:\n")
SVM_dict_v3
#### Otimização de Hiperparâmetros com Grid Search e Kernel Polinomial
# ->
# Cria o modelo
modelo_v4 = svm.SVC(kernel = 'poly')
# Valores para o grid
r_range = np.array([0.5, 1])
```

```

gamma_range = np.array([0.001, 0.01])
d_range = np.array([2,3, 4])
# Grid de hiperparâmetros
param_grid_poly = dict(gamma = gamma_range, degree = d_range, coef0 = r_range)
# Grid Search
start = time.time()
modelo_v4_grid_search_poly = GridSearchCV(modelo_v4, param_grid_poly, cv = 3)
# Treinamento
modelo_v4_grid_search_poly.fit(X_treino_scaled, y_treino)
end = time.time()
print('Tempo de Treinamento do Modelo com Grid Search:', end - start)
# Acurácia em Treino
print(f'Acurácia em Treinamento: {modelo_v4_grid_search_poly.best_score_:.2%}')
print("")
print(f'Hiperparâmetros Ideais: {modelo_v4_grid_search_poly.best_params_}')
# ->
# Previsões
previsoes_v4 = modelo_v4_grid_search_poly.predict(X_teste_scaled)
# ->
# Dicionário de métricas e metadados
SVM_dict_v4 = {'Modelo': 'SVM',
               'Versão': '4',
               'Kernel': 'Polinomial com Dados Padronizados',
               'Precision': precision_score(previsoes_v4, y_teste),
               'Recall': recall_score(previsoes_v4, y_teste),
               'F1 Score': f1_score(previsoes_v4, y_teste),
               'Acurácia': accuracy_score(previsoes_v4, y_teste),
               'AUC': roc_auc_score(y_teste, previsoes_v4)}
# ->
# Print
print("Métricas em Teste:\n")
SVM_dict_v4
# ->
# Concatena todos os dicionários em um dataframe do Pandas
resumo = pd.DataFrame({'SVM_dict_v1': pd.Series(SVM_dict_v1),
                       'SVM_dict_v2': pd.Series(SVM_dict_v2),
                       'SVM_dict_v3': pd.Series(SVM_dict_v3),
                       'SVM_dict_v4': pd.Series(SVM_dict_v4)})
# ->
# Print
resumo
#### Fazendo Previsões com o Modelo Treinado
# ->
# Novo registro
novo_x = np.array([4.0, 5.56, 1.0, 3.78, 2.995, 6.00, 0.69, 0.70, 0.69, 0, 6, 1, 1, 3, 3, 2, False]).reshape(1, -1)
# ->
# Padronizando o registro
novo_x_scaled = StandardScaler().fit_transform(novo_x)
# ->
# Previsão
previsao_novo_x = modelo_v3_grid_search_rbf.predict(novo_x_scaled)
# ->
previsao_novo_x
# Fim

```

Quizz

As Máquinas de Vetores de Suporte (Support Vector Machines) são um conjunto de técnicas de aprendizagem supervisionada para classificação e regressão (e também para detecção de valores outliers).

O que uma SVM faz é encontrar uma linha de separação, chamada de Hiperplano.

A distância entre o hiperplano e o primeiro ponto de cada classe é chamada de Margem.

A primeira tarefa realizada pelo algoritmo SVM, é classificar os dados em 2 grupos, considerando uma classificação binária. Para isso, o algoritmo gera uma linha que melhor divide as classes. Essa linha é chamada de hiperplano.

As SVM's permitem classificar dados não linearmente separáveis através do kernel trick, ou truque do kernel.

7.12. Processamento de Linguagem Natural

Processamento de Linguagem Natural

Processamento de Linguagem Natural (PLN) é uma subárea da inteligência artificial e da linguística que estuda os problemas da geração e compreensão automática de línguas humanas naturais.

Em uma analogia simples, PLN é como ensinar uma linguagem a uma criança.

Você utiliza PLN em aplicações como:

- Corretores Ortográficos (Microsoft Word)
- Engines de Reconhecimento de Voz (Siri, Google Voice)
- Classificadores de Spam
- Mecanismos de Busca (Google, Bing)
- Sistemas de Inteligência Artificial como Assistentes Pessoais
- Robôs Advogados • Robôs Médicos

Processamento de Linguagem Natural em Python

Boas Práticas/

Introdução PLN/

Introdução Py Torch/

BOW/

TF-IDF

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
## Boas Práticas
## Use Listas, Dicionários e Sets
Listas, conjuntos (sets) e dicionários são estruturas de dados otimizadas para processamento de texto e formam o núcleo de armazenamento e
processamento de dados da linguagem Python. Use-as!
### Listas
# ->
# Listas são objetos ordenados em Python
animais = ['cavalo', 'cachorro', 'gato', 'zebra']
# ->
# Tipo do objeto
type(animais)
# ->
# Lista de strings
animais
# ->
# Acesso ao elemento pelo índice
animais[2]
# ->
# Listas podem ter elementos de diferentes tipos
cursos_dsa = ['Matemática', 'Deep Learning', 1678, 9872, True]
# ->
# Lista de elementos com diferentes tipos
cursos_dsa
# ->
cursos_dsa[3]
### Dicionários
# ->
# Dicionários são estruturas de dados que armazenam chave e valor
dict1 = {'Nome': 'Bob', 'Idade': 39, 'Curso': 'Formação Cientista de Dados DSA'}
# ->
# Tipo do objeto
type(dict1)
# ->
# Dicionário
```

```

dict1
# ->
# Retorna todos os itens
dict1.items()
# ->
# Retorna apenas os valores
dict1.values()
#### Sets
# ->
# Sets são listas não ordenadas que podem ser modificadas
conjunto1 = {1, 2, 3}
print(conjunto1)
# ->
# Tipo do objeto
type(conjunto1)
# ->
# Podemos usar a função set() para criar um objeto desse tipo. Set armazena apenas valores únicos.
conjunto2 = set([1,2,3,2])
print(conjunto2)
# ->
# Objetos set armazenam dados de qualquer tipo
conjunto3 = {1.0, "Bob", (1, 2, 3)}
print(conjunto3)
# ->
# Podemos converter outros objetos para set
# Set vazio
print(set())
# String
print(set('Python'))
# Tupla
print(set(('a', 'e', 'i', 'o', 'u')))
# Lista
print(set(['a', 'e', 'i', 'o', 'u']))
# Range
print(set(range(5)))
# ->
# E podemos fazer união ou interseção entre sets
A = set(('a', 'e', 'i', 'o', 'u'))
B = set(['a', 2, 'z', 'o', (9,8)])
print('União:', A.union(B))
print('Interseção:', A.intersection(B))
### Unicode e Encoding

```

Fundamentalmente, os computadores lidam com números. Gravam letras e outros caracteres na memória designando um número para cada um deles. Antes de o Unicode ser inventado, havia centenas de sistemas diferentes de codificação . Nenhum destes sistemas de codificação, no entanto, poderia conter caracteres suficientes: por exemplo, a União Européia por si só requer vários sistemas de codificação diferentes para cobrir todas as línguas. Mesmo para uma única língua como o inglês não havia sistema de codificação adequado para todas as letras, pontuação e símbolos técnicos em uso corrente.

Estes sistemas de codificação são também conflitantes entre si. Em outras palavras, dois codificadores podem usar o mesmo número para dois caracteres diferentes ou usar números diferentes para o mesmo caracter. Qualquer computador em particular (especialmente os servidores) precisam suportar muitos codificadores diferentes; ainda assim toda as vezes que se passam dados entre codificadores ou plataformas diferentes, estes dados sempre correm o risco de serem corrompidos.

O Unicode fornece um único número para cada caracter, não importa a plataforma, não importa o programa, não importa a língua. O Padrão Unicode tem sido adotado por líderes do setor de informática tais como a Apple, HP, IBM, Microsoft, Oracle, SAP, Sun, Sybase, Unisys e muitos outros. O Unicode é necessário para padrões modernos tais como o XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc. e é a maneira oficial de implementar o ISO/IEC 10646. É suportado por muitos sistemas operacionais, todos os browsers modernos e muitos outros produtos. O surgimento do Padrão Unicode Standard e a disponibilidade de instrumentos para suportá-lo está entre as tendências recentes mais significativas das tecnológicas mundiais de software.

Textos Unicode podem ser codificados de várias formas diferentes — internamente, .NET e Java usam UTF-16; Python 3 escolhe entre ASCII, UTF-8, UTF-16 e UTF-32 dependendo dos caracteres que estão no texto que você está processando.

Ainda assim, UTF-8 é a codificação mais popular para arquivos texto (como arquivos-fonte Python).

****O Que é Codificação (Encoding) de Caracteres?***

De forma simples, é uma maneira de traduzir caracteres (como letras, pontuação, símbolos, espaços em branco e caracteres de controle) em números inteiros e, finalmente, em bits. Cada caractere pode ser codificado para uma sequência única de bits.

****E Unicode?***

O Unicode não é uma codificação (encoding) por si mesmo. Em vez disso, o Unicode é implementado por diferentes codificações de caracteres, o Unicode contém praticamente todos os caracteres que você pode imaginar, incluindo também outros não "imprimíveis".

Unicode é um padrão de codificação abstrato, não uma codificação. É aí que o UTF-8 e outros esquemas de codificação entram em cena. O padrão Unicode (um mapa de caracteres) define várias codificações diferentes de seu conjunto de caracteres único.

UTF-8 e seus primos menos utilizados, UTF-16 e UTF-32, são formatos de codificação para representar caracteres Unicode como dados binários de um ou mais bytes por caractere.

- O código-fonte do Python 3 é assumido como UTF-8 por padrão.

- Todo o texto (str) é Unicode por padrão. O texto Unicode codificado é representado como dados binários (bytes). O tipo str pode conter qualquer caractere Unicode literal, como " Δ / Δ ", todos os quais serão armazenados como Unicode.

```

# ->
a = 'maça'
print(a)
# ->
import sys

```

```

# ->
sys.getdefaultencoding()
# ->
import locale
# ->
locale.getpreferredencoding() # Em geral no Windows o padrão é 'cp1252'
# ->
#import os
#os.environ["PYTHONIOENCODING"] = ""
# ->
#!env | grep PYTHONIOENCODING
# ->
a = 'maça'
print (len(a))
# ->
print(a)
# ->
b = u'maça'
print (len(b))
# ->
print(b)
# ->
a == b
# ->
type(a)
# ->
type(b)
# ->
isinstance(a, str)
# ->
isinstance(b, str)
# ->
"Hello there!".encode("ascii")
# ->
"Hello there... 🍏!".encode("ascii")
# ->
a.decode('ascii')
# ->
b.decode('ascii')
# ->
a = a.encode('ISO-8859-1')
# ->
print(a)
Descobrir a codificação de texto é tarefa do desenvolvedor, não da linguagem Python! É o seu trabalho!
# ->
!pip install -q chardet
# ->
# Algumas bibliotecas podem ajudar, mas lembre-se: encontrar a codificação correta é uma ciência heurística!
import chardet
# ->
chardet.detect(a)
# ->
a.decode('ISO-8859-1')
# ->
print (a.decode(chardet.detect(a)['encoding']))
# ->
nome = 'José'
print(nome)
# ->
chardet.detect(nome)
# ->
nome_enc = nome.encode('ISO-8859-1')
# ->
print(nome_enc)
# ->
nome_dec = nome_enc.decode('ISO-8859-1')
# ->
print(nome_dec)
# ->
len(nome_dec.encode("utf-8"))
# ->
len(nome_dec.encode("utf-16"))
## List Comprehension
List Comprehensions são muito otimizadas. Use-as para filtrar dados!
**Calcule os quadrados dos números de 1 a 10.**
# ->
# Sem list comprehension
# Lista vazia para armazenar os resultados

```

```

quadrados_sem_lc = []
# Loop pelo range de números
for i in range(1,11):
    quadrados_sem_lc.append(i * i)
print(quadrados_sem_lc)
# ->
# Com list comprehension
quadrados_com_lc = [i * i for i in range(1,11)]
print(quadrados_com_lc)
**Imprima somente as vogais da frase abaixo usando LC.**
# ->
# Frase
frase = 'o meteoro veio em direção à terra'
# ->
# List Comprehension
vogais = [i for i in frase if i in 'aeiou']
print(vogais)
**Imprima o quadrado de cada número de 1 a 10 e imprima o número.**
# ->
# Nesse caso usamos Dict Comprehension
quadrados_dc = {i: i * i for i in range(1,11)}
print(quadrados_dc)
**Transforme a matriz abaixo em um vetor usando List Comprehension.**
# ->
# Matriz
matrix = [[0, 0, 0], [1, 1, 1], [2, 2, 2]]
print(matrix)
# ->
# Vetor
vector = [num for linha in matrix for num in linha]
print(vector)
# ->
# Cria uma matriz 3x4 com List Comprehensions aninhadas
matrix = [[item for item in range(4)] for item in range(3)]
print(matrix)
**Concatene a palavra código com cada item da lista abaixo.**
# ->
# Lista
cols = ['_A','_B','_C']
palavra = 'código'
# ->
# Concatenação
codigos = [palavra + x for x in cols]
print(codigos)
## Outras Estruturas de Dados e Pacotes
# ->
!pip install -q spacy
# ->
import spacy
# ->
# https://spacy.io/usage/models
!python -m spacy download pt_core_news_sm
# ->
nlp = spacy.load('pt_core_news_sm')
# ->
texto = ('Estudando Processamento de Linguagem Natural no Curso de Machine Learning da DSA')
# ->
tokens = nlp(texto)
# ->
print ([token.text for token in tokens])
# ->
frases = ('O futebol atrai pessoas de todas as idades'
          ' Eu visitei a Europa '
          ' Prefiro uma bela macarronada '
          ' Acho que vou gostar de PLN ')
# ->
about_doc = nlp(frases)
# ->
sentences = list(about_doc.sents)
# ->
len(sentences)
# ->
for sentence in sentences:
    print (sentence)
# ->
for token in about_doc:
    print (token, token.idx)
## Expressões Regulares

```



```

https://docs.python.org/3/howto/regex.html
https://www.w3schools.com/python/python_regex.asp
# ->
import re
Python oferece duas operações primitivas diferentes baseadas em expressões regulares: match verifica uma correspondência apenas no início da
sequência de caracteres, enquanto search verifica uma correspondência em qualquer parte da sequência de caracteres. Temos ainda diversas outras
funções de busca de padrões.
# ->
# Verificamos se a frase começa com expressão alfa-numérica
# "^": Esta expressão corresponde ao início de uma sequência
# "w+": Esta expressão corresponde ao caractere alfanumérico na string
texto1 = "AC56 é o código mais comum na lista de voos."
reg1 = re.findall("^w+", texto1)
print(reg1)
# ->
# Usamos \s para fazer a divisão de uma frase por espaços
print((re.split('\s', texto1)))
# ->
# Lista de voos internacionais
voos = ["AC56 Londres", "JK56 Paris", "AC921 Filadélfia"]
# Buscamos os códigos dos voos
for voo in voos:
    resultado = re.match("^(w+)",
voo)
    if resultado is not None:
        print(resultado.groups())
# ->
# Busca de padrões
# Lista de padrões
palavras = ['Inverno', 'Londres', 'Barcelona']
# Texto
texto = "AC56 é o código mais comum na lista de voos de Londres para Paris no Inverno."
# Busca
for palavra in palavras:
    print("\nPesquisando por \"%s\" no texto -> ' % (palavra), end = ' ')
    if re.search(palavra, texto):
        print('Encontrei o padrão no texto!')
else:
    print('Desculpe, não encontrei', palavra, 'no texto.')
# ->
# Busca pelo padrão de e-mail no texto
# Lista de e-mails
lista = 'bob@gmail.com, maria@hotmail.com, zicoyahoomail.com'
# E-mails encontrados
emails = re.findall('[\w\.-]+@[ \w\.-]+', lista)
# Retorna os e-mails encontrados
for email in emails:
    print("\nE-mail encontrado na lista: ", email)
### Fim

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->

```

Versão da Linguagem Python

```

from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())

```

Introdução ao Processamento de Linguagem Natural

Tudo o que expressamos (verbalmente ou por escrito) carrega enormes quantidades de informação.

O tópico que escolhemos, nosso tom, nossa seleção de palavras, tudo acrescenta algum tipo de informação que pode ser interpretada e com o valor extraído dela. Em teoria, podemos entender e até prever o comportamento humano usando essas informações.

Mas há um problema: uma pessoa pode gerar centenas ou milhares de palavras em uma declaração, cada sentença com sua complexidade correspondente. Se você deseja dimensionar e analisar várias centenas, milhares ou milhões de pessoas ou declarações em uma determinada região, a situação é incontrolável.

Dados gerados a partir de conversas, declarações ou até tweets são exemplos de dados não estruturados. Os dados não estruturados não se encaixam perfeitamente na estrutura tradicional de linhas e colunas de bancos de dados relacionais e representam a grande maioria dos dados disponíveis no mundo real. É confuso e difícil de manipular.

No entanto, graças aos avanços em disciplinas como aprendizado de máquina, uma grande revolução está acontecendo em relação a esse tópico.

Atualmente, não se trata mais de tentar interpretar um texto ou discurso com base em suas palavras-chave (a maneira mecânica antiquada), mas de entender o significado por trás dessas palavras (a maneira cognitiva). Dessa forma, é possível detectar figuras de linguagem como ironia, ou mesmo realizar análises de sentimentos.

O Que é Processamento de Linguagem Natural?

O Processamento de Linguagem Natural, geralmente abreviado como PLN, é um ramo da Inteligência Artificial que lida com a interação entre computadores e seres humanos usando a linguagem natural.

O objetivo final em PLN é ler, decifrar, entender e prever as linguagens humanas de uma maneira valiosa. A maioria das técnicas de PLN depende do aprendizado de máquina para derivar significado das linguagens humanas.

Casos de Uso

Em termos simples, PLN representa o tratamento automático da linguagem humana natural, como fala ou texto, e, embora o conceito em si seja fascinante, o valor real por trás dessa tecnologia vem dos casos de uso.

PLN pode ajudá-lo com muitas tarefas e os campos de aplicação parecem aumentar diariamente. Vamos mencionar alguns exemplos:

- PLN permite o reconhecimento e a previsão de doenças com base em registros eletrônicos de saúde e na fala do próprio paciente. Essa capacidade está sendo explorada em condições de saúde que vão de doenças cardiovasculares a depressão e até esquizofrenia. Por exemplo, o Amazon Comprehend Medical é um serviço que usa PLN para extrair condições de doenças, medicamentos e resultados de tratamento de anotações de pacientes, relatórios de ensaios clínicos e outros registros eletrônicos de saúde.
- As organizações podem determinar o que os clientes estão dizendo sobre um serviço ou produto, identificando e extraíndo informações em fontes como mídia social. Essa análise de sentimentos pode fornecer muitas informações sobre as escolhas dos clientes e seus motivadores de decisão.
- Um inventor da IBM desenvolveu um assistente cognitivo que funciona como um mecanismo de pesquisa personalizado, aprendendo tudo sobre você e depois lembrando um nome, uma música ou qualquer coisa que você não consegue lembrar no momento em que precisa.
- Empresas como Yahoo e Google filtram e classificam seus e-mails com PLN, analisando o texto nos e-mails que fluem através de seus servidores e interrompendo o spam antes mesmo de entrarem na sua caixa de entrada.
- Para ajudar a identificar notícias falsas, o Grupo NLP do MIT desenvolveu um novo sistema para determinar se uma fonte é precisa ou politicamente tendenciosa, detectando se uma fonte de notícias pode ser confiável ou não.
- O Alexa da Amazon e o Siri da Apple são exemplos de interfaces inteligentes orientadas por voz que usam PLN para responder às solicitações vocais e fazem de tudo, como encontrar uma loja em particular, informar a previsão do tempo, sugerir o melhor caminho para o escritório ou acender as luzes em casa.
- Ter uma ideia do que está acontecendo e do que as pessoas estão falando pode ser muito valioso para os operadores financeiros. PLN está sendo usada para rastrear notícias, relatórios, comentários sobre possíveis fusões entre empresas; tudo pode ser incorporado a um algoritmo de negociação para gerar lucros maciços. Lembre-se: compre o boato, venda as notícias.
- PLN também está sendo usado nas fases de busca e seleção do recrutamento de talentos, identificando as habilidades de possíveis contratados e também identificando perspectivas antes que elas se tornem ativas no mercado de trabalho.
- Desenvolvido pela tecnologia IBM Watson NLP, o LegalMation desenvolveu uma plataforma para automatizar tarefas rotineiras de litígio e ajudar as equipes jurídicas a economizar tempo, reduzir custos e mudar o foco estratégico.

E os exemplos não param. PLN é um campo ativo e cada vez mais presente em nossas vidas.

```
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
## Instalação do pacote NLTK
http://www.nltk.org/index.html
http://www.nltk.org/book/
# ->
# Instalação do módulo NLTK
!pip install -q nltk
# ->
# Imports
import re
import nltk
import spacy
import string
import numpy as np
import pandas as pd
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
# ->
# Instalando os arquivos de dados e dicionários do NLTK
nltk.download('all')
## Tokenization
Processo de dividir uma string em listas de pedaços ou "tokens". Um token é uma parte inteira. Por exemplos: uma palavra é um token em uma sentença. Uma sentença é um token em um parágrafo.
### Dividindo um Parágrafo em Frases
# ->
from nltk.tokenize import sent_tokenize
# ->
paragrafo = "Seja Bem-vindo a Data Science Academy. Bom saber que você está aprendendo PLN. Obrigado por estar conosco."
# ->
?sent_tokenize
# ->
# Dividindo o parágrafo em frases
sent_tokenize(paragrafo)
# ->
# Utilizando dicionário do pacote NLTK
tokenizer = nltk.data.load('tokenizers/punkt/portuguese.pickle')
# ->
tokenizer.tokenize(paragrafo)
# ->
# Dicionário em espanhol
spanish_tokenizer = nltk.data.load('tokenizers/punkt/spanish.pickle')
# ->
spanish_tokenizer.tokenize('Hola amigo. Estoy bien.')
### Dividindo uma Frase em Palavras
```

```

# ->
from nltk.tokenize import word_tokenize
# ->
word_tokenize('Data Science Academy')
# ->
from nltk.tokenize import TreebankWordTokenizer
# ->
tokenizer = TreebankWordTokenizer()
# ->
tokenizer.tokenize('Inteligência Artificial')
# ->
word_tokenize("can't") # cannot
# ->
from nltk.tokenize import WordPunctTokenizer
# ->
tokenizer = WordPunctTokenizer()
# ->
tokenizer.tokenize("Can't is a contraction.")
# ->
from nltk.tokenize import RegexpTokenizer
# ->
?RegexpTokenizer
# ->
tokenizer = RegexpTokenizer("[\w]+")
# ->
tokenizer.tokenize("Can't is a contraction.")
# ->
from nltk.tokenize import regexp_tokenize
# ->
?regexp_tokenize
# ->
regexp_tokenize("Can't is a contraction.", "[\w]+")
# ->
tokenizer = RegexpTokenizer("\s+", gaps = True)
# ->
tokenizer.tokenize("Can't is a contraction.")
# ->
# Uma operação única com List Comprehension
# Texto a ser tokenizado
texto = "Seja Bem-vindo a Data Science Academy. Bom saber que você está aprendendo PLN. Obrigado por estar conosco."
# List Comprehension
print([word_tokenize(frase) for frase in sent_tokenize(texto)])
## Stopwords
Stopwords são palavras comuns que normalmente não contribuem para o significado de uma frase, pelo menos com relação ao propósito da
informação e do processamento da linguagem natural. São palavras como "The" e "a" ((em inglês) ou "O/A" e "Um/Uma" ((em português). Muitos
mecanismos de busca filtram estas palavras (stopwords), como forma de economizar espaço em seus índices de pesquisa.
# ->
from nltk.corpus import stopwords
# ->
english_stops = set(stopwords.words('english'))
# ->
english_stops
# ->
palavras = ["Can't", 'is', 'a', 'contraction']
# ->
[palavra for palavra in palavras if palavra not in english_stops]
# ->
portuguese_stops = set(stopwords.words('portuguese'))
# ->
palavras = ["Aquilo", 'é', 'uma', 'ferrari']
# ->
[palavra for palavra in palavras if palavra not in portuguese_stops]
# ->
stopwords.fileids()
# ->
stopwords.words('portuguese')
#### Wordnet
WordNet é um banco de dados léxico (em Inglês). É uma espécie de dicionário criado especificamente para processamento de linguagem natural.
# ->
from nltk.corpus import wordnet
# ->
?wordnet
# ->
syn = wordnet.synsets('cookbook')[0]
# ->
syn.name()
# ->
syn.definition()

```

```
# ->
wordnet.synsets('cooking')[0].examples()
## Stemming e Lemmatization
Por razões gramaticais, os documentos usarão diferentes formas de uma palavra, como por exemplo:
organizar, organizado e organizando.
Além disso, existem famílias de palavras derivadamente relacionadas com significados semelhantes, como democracia, democrático e
democratização. Em muitas situações, pode ser útil procurar uma dessas palavras para retornar documentos que contenham outra palavra no
conjunto.
O objetivo da Derivação (Stemming) e da Lematização (Lemmatization) é reduzir
as formas flexionadas e, às vezes, derivadas de uma palavra para uma base comum.
#### Stemming
Stemming é a técnica de remover sufixos e prefixos de uma palavra, chamada stem. Por exemplo, o stem da palavra cooking é cook. Um bom
algoritmo sabe que "ing" é um sufixo e pode ser removido. Stemming é muito usado em mecanismos de buscas para indexação de palavras. Ao
invés de armazenar todas as formas de uma palavra, um mecanismo de busca armazena apenas o stem da palavra, reduzindo o tamanho do índice e
aumentando a performance do processo de busca.
# ->
from nltk.stem import PorterStemmer
# ->
?PorterStemmer
# ->
porter_stemmer = PorterStemmer()
# ->
porter_stemmer.stem('cooking')
# ->
porter_stemmer.stem('cookery')
# ->
from nltk.stem import LancasterStemmer
# ->
lanc_stemmer = LancasterStemmer()
# ->
lanc_stemmer.stem('cooking')
# ->
lanc_stemmer.stem('cookery')
# ->
from nltk.stem import RegexpStemmer
# ->
regexp_stemmer = RegexpStemmer('ing')
# ->
regexp_stemmer.stem('cooking')
# ->
lista_palavras = ["cat", "cats", "know", "knowing", "time", "timing", "football", "footballers"]
# ->
porter_stemmer = PorterStemmer()
# ->
for palavra in lista_palavras:
    print(palavra + ' -> ' + porter_stemmer.stem(palavra))
# ->
def SentenceStemmer(sentence):
    tokens = word_tokenize(sentence)
    stems = [porter_stemmer.stem(token) for token in tokens]
    return " ".join(stems)
# ->
SentenceStemmer("The cats and dogs are running")
#### Lemmatization
Lemmatization leva em consideração a análise morfológica das palavras. Para fazer isso, é necessário ter dicionários detalhados nos quais o
algoritmo possa procurar para vincular o formulário ao seu lema. Veja um exemplo:
- Forma flexionada: organizando
- Lema: organiza
- Forma flexionada: organizado
- Lema: organiza
Com Lemmatization as duas formas flexionadas organizando e organizado seria representadas somente pelo lema organiza.
# ->
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
# ->
wordnet_lemmatizer = WordNetLemmatizer()
# ->
print(wordnet_lemmatizer.lemmatize('mice'))
print(wordnet_lemmatizer.lemmatize('cacti')) # plural da palavra cactus - cactuses (inglês) ou cacti (latim)
print(wordnet_lemmatizer.lemmatize('horses'))
print(wordnet_lemmatizer.lemmatize('wolves'))
# ->
print(wordnet_lemmatizer.lemmatize('madeupwords'))
print(porter_stemmer.stem('madeupwords'))
# ->
from nltk import pos_tag
Part-of-Speech Tag (PoS-Tag)
http://www.nltk.org/book/ch05.html
```

```

# ->
def return_word_pos_tuples(sentence):
    return pos_tag(word_tokenize(sentence))
# ->
sentence = 'The cats and dogs are running'
# ->
return_word_pos_tuples(sentence)
# ->
def get_pos_wordnet(pos_tag):
    pos_dict = {"N": wordnet.NOUN,
                "V": wordnet.VERB,
                "J": wordnet.ADJ,
                "R": wordnet.ADV}
    return pos_dict.get(pos_tag[0].upper(), wordnet.NOUN)
# ->
get_pos_wordnet('VBG')
# ->
def lemmatize_with_pos(sentence):
    new_sentence = []
    tuples = return_word_pos_tuples(sentence)
    for tup in tuples:
        pos = get_pos_wordnet(tup[1])
        lemma = wordnet_lemmatizer.lemmatize(tup[0], pos = pos)
        new_sentence.append(lemma)
    return new_sentence
# ->
print(lemmatize_with_pos(sentence))

```

Corpus

Corpus é uma coleção de documentos de texto e Corpora é o plural de Corpus. Esse termo vem da palavra em Latim para corpo (nesse caso, o corpo de um texto). Um Corpus customizado é uma coleção de arquivos de texto organizados em um diretório. Se você for treinar seu próprio modelo como parte de um processo de classificação de texto (como análise de texto), você terá que criar seu próprio Corpus e treiná-lo.

```

# ->
import os
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
# ->
corpusdir = 'corpus/'
# ->
newcorpus = PlaintextCorpusReader(corpusdir, '.*')
# ->
newcorpus.fileids()
# ->
# Acessando cada arquivo no Corpus com loop for
for infile in sorted(newcorpus.fileids()):
    print(infile)
# ->
# Acessando cada arquivo no Corpus com list comprehension
[arquivo for arquivo in sorted(newcorpus.fileids())]
# ->
# Conteúdo do Corpus
print(newcorpus.raw().strip())
# ->
# Conteúdo do Corpus tokenizado por parágrafo (nova linha)
print(newcorpus.paras())
# ->
# Conteúdo do Corpus tokenizado por sentença (nova linha)
print(newcorpus.sents())
# ->
# Conteúdo do Corpus tokenizado por sentença por arquivo
print(newcorpus.sents(newcorpus.fileids()[0]))

```

O NLTK traz diversos Corpus e Corpora que podem ser usados para os mais variados fins. Vamos experimentar o Corpus do Projeto Gutenberg: <https://www.gutenberg.org/>

```

# ->
from nltk.corpus import gutenberg as gt
# ->
print(gt.fileids())
# ->
shakespeare_macbeth = gt.words("shakespeare-macbeth.txt")
print(shakespeare_macbeth)
# ->
raw = gt.raw("shakespeare-macbeth.txt")
print(raw)
# ->
sents = gt.sents("shakespeare-macbeth.txt")
print(sents)
# ->
for fileid in gt.fileids():
    num_words = len(gt.words(fileid))

```

```

num_sents = len(gt.sents(fileid))
print("Dados do Arquivo:", fileid)
print("Número de Palavras:", num_words)
print("Número de Frases:", num_sents, end = "\n\n\n")

```

Lab - Collocations e Processamento de Comentários de Avaliações de Hotéis

Collocations são duas ou mais palavras que tendem a aparecer frequentemente juntas, como "Estados Unidos", "Rio Grande do Sul" ou "Machine Learning". Essas palavras podem gerar diversas combinações e por isso o contexto também é importante no processamento de linguagem natural. Os dois tipos mais comuns de Collocations são bigramas e trigramas. Bigramas são duas palavras adjacentes, como "tomografia computadorizada", "aprendizado de máquina" ou "mídia social". Trigramas são três palavras adjacentes, como "fora do negócio" ou "Proctor and Gamble".

- Bigramas: (Nome, Nome), (Adjetivo, Nome)

- Trigramas: (Adjetivo/Nome, Qualquer_Item, Adjetivo/Nome)

Mas se escolhermos palavras adjacentes como bigrama ou trigramas, não obteremos frases significativas. Por exemplo, a frase 'Ele usa mídias sociais' contém bigramas: 'Ele usa', 'usa mídias', 'mídias sociais'. 'Ele usa' e 'usa mídias' não significa nada, enquanto 'mídias sociais' é um bigrama significativo.

Como fazemos boas seleções para Collocations? As co-ocorrências podem não ser suficientes, pois frases como 'assim como' podem co-ocorrer com frequência, mas não são significativas. Vamos explorar vários métodos para filtrar as Collocations mais significativas: contagem de frequências, informação mútua pontual (PMI) e teste de hipóteses (teste t e qui-quadrado).

Definição do Problema

Dado um conjunto de texto de avaliações (comentários) de hotéis, vamos buscar as Collocations mais relevantes que ajudam a explicar as avaliações!

```

# ->
# Se necessário, instale pacotes que não estejam instalados
!pip install -q spacy
# ->
# Imports
import pandas as pd
import nltk
import spacy
import re
import string
from nltk.corpus import stopwords
# ->
# Se necessário, faça o download das stopwords
nltk.download('stopwords')
# ->
# Carregando dados de avaliações de hotéis
# Fonte de dados: https://datafiniti.co/products/business-data/
avaliacoes_hotéis = pd.read_csv('https://raw.githubusercontent.com/dsacademybr/Datasets/master/dataset7.csv')
# ->
# Visualiza os dados
avaliacoes_hotéis.head(5)
# ->
# Tipo do objeto
type(avaliacoes_hotéis)
# ->
# Shape
avaliacoes_hotéis.shape
# ->
# Extraindo as avaliações
comentarios = avaliacoes_hotéis['reviews.text']
# ->
# Converte para o tipo string
comentarios = comentarios.astype('str')
# ->
# Função para remover caracteres non-ascii
def removeNoAscii(s):
    return "".join(i for i in s if ord(i) < 128)
# ->
# Remove caracteres non-ascii
comentarios = comentarios.map(lambda x: removeNoAscii(x))
# ->
# Obtém as stopwords em todos os idiomas
dicionario_stopwords = {lang: set(nltk.corpus.stopwords.words(lang)) for lang in nltk.corpus.stopwords.fileids()}
dicionario_stopwords
# ->
# Função para detectar o idioma predominante com base nas stopwords
def descobre_idioma(text):
    # Aplica tokenização considerando pontuação
    palavras = set(nltk.wordpunct_tokenize(text.lower()))
    # Conta o total de palavras tokenizadas considerando o dicionário de stopwords
    lang = max(((lang, len(palavras & stopwords)) for lang, stopwords in dicionario_stopwords.items()), key = lambda x: x[1])[0]
    # Verifica se o idioma é português
    if lang == 'portuguese':
        return True
    else:
        return False
# ->
# Filtra somente os comentários em português

```

```

comentarios_portugues = comentarios[comentarios.apply(descobre_idioma)]
# ->
# Shape
comentarios_portugues.shape
# ->
# Print
comentarios_portugues
# ->
# Função para detectar o idioma predominante com base nas stopwords
def descobre_idioma(text):
    words = set(nltk.wordpunct_tokenize(text.lower()))
    lang = max(((lang, len(words & stopwords)) for lang, stopwords in dicionario_stopwords.items()), key = lambda x: x[1])[0]
    if lang == 'english':
        return True
    else:
        return False
# ->
# Filtra somente os comentários em português
comentarios_ingles = comentarios[comentarios.apply(descobre_idioma)]
# ->
# Shape
comentarios_ingles.shape
# ->
# Print
comentarios_ingles.head()
# ->
# Removendo duplicidades
comentarios_ingles.drop_duplicates(inplace = True)
# ->
# Shape
comentarios_ingles.shape
# ->
# Baixando o dicionário inglês
# https://spacy.io/usage/models
!python -m spacy download en_core_web_sm
> Pode ser necessário reiniciar o Jupyter Notebook para executar a célula abaixo.
# ->
# Carrega o dicionário em nossa sessão SpaCy
nlp = spacy.load("en_core_web_sm")
# ->
# Função para limpar e lematizar os comentários
def limpa_comentarios(text):
    # Remove pontuação usando expressão regular
    regex = re.compile('[ ' + re.escape(string.punctuation) + '\r\t\n]')
    nopunct = regex.sub(" ", str(text))
    # Usa o SpaCy para lematização
    doc = nlp(nopunct, disable = ['parser', 'ner'])
    lemma = [token.lemma_
for token in doc]
    return lemma
# ->
# Aplica a função aos dados
comentarios_ingles_lemmatized = comentarios_ingles.map(limpa_comentarios)
# ->
# Coloca tudo em minúsculo
comentarios_ingles_lemmatized = comentarios_ingles_lemmatized.map(lambda x: [word.lower() for word in x])
# ->
# Visualiza os dados
comentarios_ingles_lemmatized.head()
# ->
# Vamos tokenizar os comentários
comentarios_tokens = [item for items in comentarios_ingles_lemmatized for item in items]
# ->
# Tokens
comentarios_tokens
#### Estratégia 1 - Buscando Bigramas e Trigramas Mais Relevantes nos Comentários Por Frequência
Nossa primeira estratégia é a mais simples de todas: contagem de frequência. Contamos quantas vezes cada Collocation aparece no texto e filtramos
pelos Collocations mais frequentes.
Vamos filtrar apenas os adjetivos e substantivos para reduzir o tempo de processamento e vamos contar a frequência dos Collocations, bigramas e
trigramas, nos comentários.
# ->
# Métricas de associação de bigramas (esse objeto possui diversos atributos, como freq, pmi, teste t, etc...)
bigramas = nltk.collocations.BigramAssocMeasures()
# ->
# Métricas de associação de trigramas
trigramas = nltk.collocations.TrigramAssocMeasures()
# ->
# O próximo passo é criar um buscador de bigramas nos tokens

```

```

buscaBigramas = nltk.collocations.BigramCollocationFinder.from_words(comentarios_tokens)
# ->
# Fazemos o mesmo com trigramas. Fique atento aos métodos que estão sendo usados
buscaTrigramas = nltk.collocations.TrigramCollocationFinder.from_words(comentarios_tokens)
# ->
# Vamos contar quantas vezes cada bigrama aparece nos tokens dos comentários
bigrama_freq = buscaBigramas.ngram_fd.items()
# ->
# Frequência dos bigramas
bigrama_freq
# ->
# Vamos converter o dicionário anterior em uma tabela de frequência no formato do Pandas para os bigramas
FreqTabBigramas = pd.DataFrame(list(bigrama_freq),
                                columns = ['Bigrama', 'Freq']).sort_values(by = 'Freq', ascending = False)
# ->
# Visualiza a tabela
FreqTabBigramas.head(5)
# ->
# Vamos contar quantas vezes cada trigrma aparece nos tokens dos comentários
trigrma_freq = buscaTrigramas.ngram_fd.items()
# ->
# Tabela de frequência no formato do Pandas para os trigramas
FreqTabTrigramas = pd.DataFrame(list(trigrma_freq),
                                columns = ['Trigrma', 'Freq']).sort_values(by = 'Freq', ascending = False)
# ->
# Visualiza a tabela
FreqTabTrigramas.head(5)
Temos muitas stopwords. Vamos removê-las.
# ->
# Vamos criar uma lista de stopwords
en_stopwords = set(stopwords.words('english'))
# ->
# Função para filtrar bigramas ADJ/NN e remover stopwords
def filtra_tipo_token_bigrma(ngram):
    # Verifica se é pronome
    if '-pron-' in ngram or 't' in ngram:
        return False
    # Loop nos ngramas para verificar se é stopword
    for word in ngram:
        if word in en_stopwords or word.isspace():
            return False
    # Tipos de tokens aceitáveis
    acceptable_types = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    # Subtipos
    second_type = ('NN', 'NNS', 'NNP', 'NNPS')
    # Tags
    tags = nltk.pos_tag(ngram)
    # Retorna o que queremos, ADJ/NN
    if tags[0][1] in acceptable_types and tags[1][1] in second_type:
        return True
    else:
        return False
# ->
# Agora filtramos os bigramas
bigramas_filtrados = FreqTabBigramas.Bigrma.map(lambda x: filtra_tipo_token_bigrma(x))
# ->
# Visualiza a tabela
bigramas_filtrados.head(5)
# ->
# Função para filtrar trigramas ADJ/NN e remover stopwords
def filtra_tipo_token_trigrma(ngram):
    # Verifica se é pronome
    if '-pron-' in ngram or 't' in ngram:
        return False
    # Loop nos ngramas para verificar se é stopword
    for word in ngram:
        if word in en_stopwords or word.isspace():
            return False
    # Tipos de tokens aceitáveis
    first_type = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    # Subtipos
    second_type = ('JJ', 'JJR', 'JJS', 'NN', 'NNS', 'NNP', 'NNPS')
    # Tags
    tags = nltk.pos_tag(ngram)
    # Retorna o que queremos, ADJ/NN
    if tags[0][1] in first_type and tags[2][1] in second_type:
        return True
    else:

```



```

return False
# ->
# Agora filtramos os trigramas
trigramas_filtrados = FreqTabTrigramas[FreqTabTrigramas.Trigrama.map(lambda x: filtra_tipo_token_trigrama(x))]
# ->
# Visualiza a tabela
trigramas_filtrados.head(5)
Já temos os bigramas e trigramas mais relevantes por frequência. Vamos usar os outros métodos e depois comparar todos eles.
### Estratégia 2 - Buscando Bigramas e Trigramas Mais Relevantes nos Comentários Por PMI
PMI significa Pointwise Mutual Information
PMI é um score que mede a probabilidade com que as palavras co-ocorrem mais do que se fossem independentes. No entanto, é muito sensível à
combinação rara de palavras. Por exemplo, se um bigrama aleatório 'abc xyz' aparecer e nem 'abc' nem 'xyz' aparecerem em nenhum outro lugar do
texto, 'abc xyz' será identificado como bigrama altamente significativo quando poderia ser apenas um erro ortográfico aleatório ou um frase muito
rara para generalizar como um bigrama. Portanto, esse método é frequentemente usado com um filtro de frequência.
# ->
# Vamos retornar somente bigramas com 20 ou mais ocorrências
buscaBigramas.apply_freq_filter(20)
# ->
# Criamos a tabela
PMITabBigramas = pd.DataFrame(list(buscaBigramas.score_ngrams(bigramas.pmi)),
                               columns = ['Trigrama', 'PMI']).sort_values(by = 'PMI', ascending = False)
# ->
# Visualiza a tabela
PMITabBigramas.head(5)
# ->
# Vamos retornar somente trigramas com 20 ou mais ocorrências
buscaTrigramas.apply_freq_filter(20)
# ->
# Criamos a tabela
PMITabTrigramas = pd.DataFrame(list(buscaTrigramas.score_ngrams(trigramas.pmi)),
                               columns = ['Trigrama', 'PMI']).sort_values(by = 'PMI', ascending = False)
# ->
# Visualiza a tabela
PMITabTrigramas.head(5)
### Estratégia 3 - Buscando Bigramas e Trigramas Mais Relevantes nos Comentários Por Teste t
O Teste t é um teste estatístico que assume uma distribuição normal dos dados. Na prática, é um teste de hipóteses, uma das bases da Inferência
Estatística.
- H0 é a hipótese nula, que palavras ocorrem em conjunto (bigramas ou trigramas) com determinada probabilidade.
- H1 é a hipótese alternativa, que palavras não ocorrem em conjunto (bigramas ou trigramas) com determinada probabilidade.
Ao aplicar o Teste t rejeitamos ou não a H0 através do cálculo de um score e assim representamos os Collocations mais relevantes no texto.
# ->
# Criamos a tabela para os bigramas
# Observe como o resultado do teste t é obtido: buscaBigramas.score_ngrams(bigramas.student_t)
TestetTabBigramas = pd.DataFrame(list(buscaBigramas.score_ngrams(bigramas.student_t)),
                                columns = ['Bigrama', 'Teste-t']).sort_values(by = 'Teste-t', ascending = False)
# ->
# Vamos aplicar o filtro pelo tipo de token conforme aplicamos no método 1
bigramas_t_filtrados = TestetTabBigramas[TestetTabBigramas.Bigrama.map(lambda x: filtra_tipo_token_bigrama(x))]
# ->
# Visualiza a tabela
bigramas_t_filtrados.head(5)
# ->
# Criamos a tabela para os trigramas
TestetTabTrigramas = pd.DataFrame(list(buscaTrigramas.score_ngrams(trigramas.student_t)),
                                columns = ['Trigrama', 'Teste-t']).sort_values(by = 'Teste-t', ascending = False)
# ->
# Vamos aplicar o filtro pelo tipo de token conforme aplicamos no método 1
trigramas_t_filtrados = TestetTabTrigramas[TestetTabTrigramas.Trigrama.map(lambda x: filtra_tipo_token_trigrama(x))]
# ->
# Visualiza a tabela
trigramas_t_filtrados.head(5)
### Estratégia 4 - Buscando Bigramas e Trigramas Mais Relevantes nos Comentários Por Teste do Qui-quadrado
O teste do qui-quadrado é uma alternativa ao teste t. O teste do qui-quadrado assume na hipótese nula que as palavras são independentes, assim
como no teste t.
# ->
# Prepara a tabela
# Observe como estamos coletando a estatística qui-quadrado: buscaBigramas.score_ngrams(bigramas.chi_sq)
QuiTabBigramas = pd.DataFrame(list(buscaBigramas.score_ngrams(bigramas.chi_sq)),
                              columns = ['Bigrama', 'Qui']).sort_values(by = 'Qui', ascending = False)
# ->
# Visualiza a tabela
QuiTabBigramas.head(5)
# ->
# Prepara a tabela
QuiTabTrigramas = pd.DataFrame(list(buscaTrigramas.score_ngrams(trigramas.chi_sq)),
                              columns = ['Trigrama', 'Qui']).sort_values(by = 'Qui', ascending = False)
# ->
# Visualiza a tabela

```

```

QuiTabTrigramas.head(5)
### Comparação e Resultado Final
# ->
# Vamos extrair os 10 Collocations bigramas mais relevantes de acordo com cada um dos 4 métodos usados
# Lembre-se que aplicamos filtros para remover as stopwords e devemos usar a tabela filtrada
metodo1_bigrama = bigramas_filtrados[:10].Bigrama.values
metodo2_bigrama = PMITabBigramas[:10].Bigrama.values
metodo3_bigrama = bigramas_t_filtrados[:10].Bigrama.values
metodo4_bigrama = QuiTabBigramas[:10].Bigrama.values
# ->
# Vamos extrair os 10 Collocations trigramas mais relevantes de acordo com cada um dos 4 métodos usados
# Lembre-se que aplicamos filtros para remover as stopwords e devemos usar a tabela filtrada
metodo1_trigrama = trigramas_filtrados[:10].Trigrama.values
metodo2_trigrama = PMITabTrigramas[:10].Trigrama.values
metodo3_trigrama = trigramas_t_filtrados[:10].Trigrama.values
metodo4_trigrama = QuiTabTrigramas[:10].Trigrama.values
# ->
# Vamos criar um super dataframe com todos os resultados para bigramas
comparaBigramas = pd.DataFrame([metodo1_bigrama, metodo2_bigrama, metodo3_bigrama, metodo4_bigrama]).T
# ->
# Nossa tabela precisa de nomes para as colunas
comparaBigramas.columns = ['Frequência',
                           'PMI',
                           'Teste-t',
                           'Teste Qui-quadrado']
# ->
# Visualiza a tabela - Padrão CSS
comparaBigramas.style.set_properties(**{'background-color':
    'green',
                                     'color': 'white',
                                     'border-color': 'white'})
# ->
# Vamos criar um super dataframe com todos os resultados para trigramas
comparaTrigramas = pd.DataFrame([metodo1_trigrama, metodo2_trigrama, metodo3_trigrama, metodo4_trigrama]).T
# ->
# Nossa tabela precisa de nomes para as colunas
comparaTrigramas.columns = ['Frequência',
                            'PMI',
                            'Teste-t',
                            'Teste Qui-quadrado']
# ->
# Visualiza a tabela
comparaTrigramas.style.set_properties(**{'background-color': 'blue',
    'color': 'white',
    'border-color': 'white'})

### Conclusão
Podemos ver que os métodos PMI e Qui-quadrado fornecem bons resultados. Seus resultados também são semelhantes.
Mas os métodos de Frequência e Teste-t apresentam os melhores resultados e são também muito semelhantes entre si.
Em aplicativos reais, podemos observar a lista e definir um limite em um valor a partir de quando a lista para de fazer sentido. Também podemos
fazer testes diferentes para ver qual lista parece fazer mais sentido para um determinado conjunto de dados.
Como alternativa, podemos combinar resultados de várias listas. Uma boa escolha é multiplicar o PMI e a Frequência para levar em consideração o
aumento da probabilidade e a frequência da ocorrência. Ou multiplicar a frequência pelo Teste-t criando assim um índice único de relevância das
Collocations.
Para este trabalho, vamos considerar as Collocations calculadas por Frequência como as mais relevantes. A escolha se deve ao fato de que as
suposições para o Teste-t não foram validadas e usar seu resultado seria inadequado. Salvamos a coluna de frequência do dataframe final em
formato csv ou txt e encaminhamos à área de Marketing da rede de hotéis.
Você pode continuar o trabalho e alterar os métodos acima, se desejar.
# Fim

```

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso <a href='https://www.datascienceacademy.com.br/course?courseid=deep-learning-frameworks'>Deep Learning Frameworks</a> e aplicado em PLN no curso <a href='https://www.datascienceacademy.com.br/course?courseid=processamento-de-linguagem-natural-e-reconhecimento-de-voz'>Processamento de Linguagem Natural</a>.
## Introdução ao Framework PyTorch
A computação em modelos de Deep Learning é feita com tensores, que são generalizações de uma matriz que pode ser indexada em mais de duas dimensões. O PyTorch é um framework que aplica operações matemáticas a tensores para então treinar modelos de Deep Learning.
https://pytorch.org/
**TorchScript**
O PyTorch TorchScript ajuda a criar modelos serializáveis e otimizáveis. Depois que treinamos esses modelos, eles também podem ser executados independentemente. Isso ajuda quando estamos no estágio de implantação do modelo de um projeto de Data Science.
Você pode treinar um modelo no PyTorch usando Python e depois exportá-lo via TorchScript para um ambiente de produção em que Python não esteja disponível.

```

****Treinamento Distribuído****

O PyTorch também oferece suporte a treinamento distribuído que permite que pesquisadores e profissionais paralelizem seus cálculos. O treinamento distribuído possibilita o uso de várias GPUs para processar lotes maiores de dados de entrada. Isso, por sua vez, reduz o tempo de computação.

****Suporte para Python****

O PyTorch tem uma interação muito boa com o Python. De fato, a codificação no PyTorch é bastante semelhante ao que fazemos em Python. Portanto, se você se sentir confortável com o Python, vai adorar trabalhar com o PyTorch.

O Que São Tensores?

![title](imagens/tensores.png)

->

Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:

pip install -U nome_pacote

Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:

!pip install torch==1.5.0

Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.

Instala o pacote watermark.

Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.

!pip install -q -U watermark

->

Instala o PyTorch

!pip install -q -U torch torchvision

->

Imports

import numpy

import torch

import torchvision

->

Versões dos pacotes usados neste jupyter notebook

%reload_ext watermark

%watermark -a "Data Science Academy" --iversions

Criando e Manipulando Tensores

A razão pela qual usamos o Numpy em Machine Learning é que é muito mais rápido do que as listas Python na execução de operações de matriz.

Por quê? Internamente, NumPy faz a maior parte do trabalho pesado em Linguagem C, que é muito mais veloz que Python.

Mas, no caso de treinar redes neurais profundas (Deep Learning), os arrays NumPy levariam meses para treinar algumas das redes de ponta. É aqui que os tensores entram em cena. O PyTorch nos fornece uma estrutura de dados chamada Tensor, que é muito semelhante à matriz ND do NumPy. Mas, diferentemente do último, os tensores podem aproveitar os recursos de uma GPU para acelerar significativamente as operações com matrizes.

->

Criando um tensor

x = torch.tensor([1., 2.])

->

Visualiza o tensor

print(x)

->

Shape

print(x.shape)

->

Criando um tensor

t = torch.tensor([[[1,1,1,1],
[2,2,2,2],
[3,3,3,3]], dtype = torch.float32)

Para determinar a forma desse tensor, examinamos primeiro as linhas (3) e depois as colunas (4). Portanto, esse tensor é 3 x 4 de classificação (rank) 2.

Rank é uma palavra comumente usada e significa apenas o número de dimensões presentes no tensor.

No PyTorch, temos duas maneiras de obter a forma (shape):

->

t.size()

->

t.shape

No PyTorch, o tamanho e a forma de um tensor significam a mesma coisa.

Normalmente, depois de conhecermos a forma de um tensor, podemos deduzir algumas coisas. Primeiro, podemos deduzir o rank do tensor. O rank de um tensor é igual ao comprimento da forma do tensor.

->

len(t.shape)

->

len(x.shape)

Também podemos deduzir o número de elementos contidos no tensor. O número de elementos dentro de um tensor (12 no nosso caso do tensor t e 2 no tensor x) é igual ao produto dos valores dos componentes da forma.

->

torch.tensor(t.shape).prod()

->

torch.tensor(x.shape).prod()

->

Retornando um elemento de um tensor

z = torch.tensor([1., 2.],[5., 3.],[0., 4.])

->

print(z)

->

```

# Shape
print(z.shape)
# ->
# Retornamos a primeira linha (índice 0) e segunda coluna (índice 1)
# O retorno é no formato de tensor
print(z[0][1])
# ->
# Retornamos a primeira linha (índice 0) e segunda coluna (índice 1)
# O retorno é no formato de escalar (apenas o valor)
print(z[0][1].item())
Quando criamos tensores com valores randômicos, passamos apenas o número de dimensões.
# ->
input1 = torch.randn([1, 4, 4, 2])
# ->
input2 = torch.randn(1, 4, 4, 2)
# ->
input1.shape
# ->
input2.shape
# ->
len(input1.shape)
# ->
len(input2.shape)
# ->
input1
# ->
input2
Considere tensores como o número de listas que uma dimensão contém. Por exemplo, um tensor (1, 4, 4, 2) terá:
1 lista contendo 4 elementos de 4 elementos de 2 elementos.
- A primeira dimensão pode conter 1 elemento.
- A segunda dimensão pode conter 4 elementos.
- A terceira dimensão pode conter 4 elementos.
- A quarta dimensão pode conter 2 elementos.

#### Array NumPy x Tensor PyTorch
# ->
# Cria um array NumPy
a = numpy.array(1)
# Cria um tensor PyTorch
b = torch.tensor(1)
# ->
# Tipo
type(a)
# ->
# Tipo
type(b)
# ->
# Print
print(a)
print(b)
#### Operações com Tensores
# ->
# Criamos 2 tensores
t1 = torch.tensor(12)
t2 = torch.tensor(4)
print(t1, t2)
# ->
# Soma
print(t1 + t2)
# ->
# Subtração
print(t1 - t2)
# ->
# Multiplicação
print(t1 * t2)
# ->
# Divisão
print(t1 // t2)
#### Operações com Matrizes
# ->
# Matriz (tensor rank 2) de números randômicos
t_rank2 = torch.randn(3,3)
t_rank2
# ->
# Tensor rank 3 de números randômicos
t_rank3 = torch.randn(3,3,3)
t_rank3
# ->

```

```

# Tensor rank 4 de números randômicos
t_rank4 = torch.randn(3,3,3,3)
t_rank4
# ->
# Multiplicação entre 2 tensores
A = torch.tensor([[[1, 2, 3], [4, 5, 6], [7, 8, 9]]])
B = torch.tensor([[[9, 8, 7], [6, 5, 4], [3, 2, 1]]])
# ->
A.shape
# ->
B.shape
# ->
len(A.shape)
# ->
len(B.shape)
# ->
A
# ->
B
# ->
resultado1 = A * B
# ->
# Resultado
print(resultado1)
# ->
resultado2 = torch.matmul(A, B)
# ->
# Resultado
print(resultado2)
# ->
resultado3 = torch.sum(A * B)
# ->
# Resultado
print(resultado3)
![title](imagens/mat-mul.gif)
Para multiplicação de matrizes, fazemos assim em PyTorch:
# ->
AB1 = A.mm(B)
# ou
AB2 = torch.mm(A, B)
# ou
AB3 = torch.matmul(A, B)
# Ou assim (Python 3.5+)
AB4 = A @ B
# ->
print(AB1)
print(AB2)
print(AB3)
print(AB4)
# ->
# Multiplicação de matrizes
A @ B
Essa notação realiza multiplicação element-wise:
# ->
# Operação element-wise
A * B
# ->
# Usando seed para iniciar 2 tensores com valores randômicos
torch.manual_seed(42)
a = torch.randn(3,3)
b = torch.randn(3,3)
# ->
# Adição de matrizes
print(torch.add(a, b))
# ->
# Subtração de matrizes
print(torch.sub(a, b))
# ->
# Multiplicação de matrizes
print(torch.mm(a, b))
# ->
# Divisão de matrizes
print(torch.div(a, b))
# ->
# Matriz Original
print(a, '\n')
# Matriz Transposta
torch.t(a)

```

Fim

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso <a href="https://www.datascienceacademy.com.br/course?courseid=deep-learning-frameworks">Deep Learning Frameworks</a> e aplicado em PLN no curso <a href="https://www.datascienceacademy.com.br/course?courseid=processamento-de-linguagem-natural-e-reconhecimento-de-voz">Processamento de Linguagem Natural</a>.
## Modelo de Classificação de Idiomas de Sentenças com Bag of Words e PyTorch
![title](imagens/bow.png)
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install torch==1.5.0
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Instala o PyTorch
!pip install -q -U torch torchvision
# ->
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn.functional as F
from torch import nn, optim
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
### Preparando os Dados
# ->
# Dados de treino
dados_treino = [("Tenho vinte paginas de leitura".lower().split(), "Portuguese"),
                ("I will visit the library".lower().split(), "English"),
                ("I am reading a book".lower().split(), "English"),
                ("This is my favourite chapter".lower().split(), "English"),
                ("Estou na biblioteca lendo meu livro preferido".lower().split(), "Portuguese"),
                ("Gosto de livros sobre viagens".lower().split(), "Portuguese")]
# ->
dados_treino
# ->
# Dados de teste
dados_teste = [("Estou lendo".lower().split(), "Portuguese"),
               ("This is not my favourite book".lower().split(), "English")]
# ->
dados_teste
# ->
# Prepara o dicionário do vocabulário
# Dicionário para o vocabulário
dict_vocab = {}
# Contador
i = 0
# Loop pelos dados de treino e teste
for palavras, idiomas in dados_treino + dados_teste:
    for palavra in palavras:
        if palavra not in dict_vocab:
            dict_vocab[palavra] = i
            i += 1
# ->
# Visualiza o vocabulário
print(dict_vocab)
# ->
# Tamanho do corpus
tamanho_corpus = len(dict_vocab)
# ->
tamanho_corpus
# ->
# Número de idiomas
idiomas = 2
```

```

# ->
# Índice para os idiomas
label_index = {"Portuguese": 0, "English": 1}
#### Construção do Modelo
# ->
# Classe para o modelo BOW de classificação
class ModeloBOW(nn.Module):
    # Método construtor
    def __init__(self, lista_idiomas, tamanho_do_corpus):
        super(ModeloBOW, self).__init__()
        self.linear = nn.Linear(tamanho_do_corpus, lista_idiomas)
    # Feed Forward
    def forward(self, bow_vec):
        return F.log_softmax(self.linear(bow_vec), dim = 1)
# ->
# Função para criar o vetor BOW necessário para o treinamento
def cria_bow_vetor(sentence, word_index):
    word_vec = torch.zeros(tamanho_corpus)
    for word in sentence:
        word_vec[dict_vocab[word]] += 1
    return word_vec.view(1, -1)
# ->
# Função para criar a variável target
def cria_target(label, label_index):
    return torch.LongTensor([label_index[label]])
# ->
# Cria o modelo
modelo = ModeloBOW(idiomas, tamanho_corpus)
# ->
# Função de perda (loss)
loss_function = nn.NLLLoss()
# ->
# Otimizador
optimizer = optim.SGD(modelo.parameters(), lr = 0.1)
#### Treinamento do Modelo
# ->
# Loop de treinamento
for epoch in range(100):
    for sentence, label in dados_treino:
        modelo.zero_grad()
        bow_vec = cria_bow_vetor(sentence, dict_vocab)
        target = cria_target(label, label_index)
        log_probs = modelo(bow_vec)
        loss = loss_function(log_probs, target)
        loss.backward()
        optimizer.step()
    if epoch % 10 == 0:
        print('Epoch: ', str(epoch+1),', Loss: ' + str(loss.item()))
#### Previsões e Avaliação do Modelo
# ->
# Função para previsões
def faz_previsao(data):
    with torch.no_grad():
        sentence = data[0]
        label = data[1]
        bow_vec = cria_bow_vetor(sentence, dict_vocab)
        log_probs = modelo(bow_vec)
        print(sentence)
        print("Probabilidade de ser o label: " + label, 'é igual a: ', np.exp(log_probs))
# ->
# Previsão com a primeira sentença de teste
faz_previsao(dados_teste[0])
# ->
dados_teste[0]
# ->
# Previsão com a segunda sentença de teste
faz_previsao(dados_teste[1])
# ->
dados_teste[1]
#### Previsões com Novas Frases
# ->
# Nova frase
novas_frases = [("Tenho livros sobre viagens".lower().split(), "Portuguese"),
                 ("Estou escrevendo".lower().split(), "Portuguese"),
                 ("Gosto de biblioteca".lower().split(), "Portuguese")]
# ->
novas_frases
# ->

```

```
faz_previsao(novas_frases[0])
# ->
faz_previsao(novas_frases[1])
# ->
faz_previsao(novas_frases[2])
# Fim
```

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso <a href="https://www.datascienceacademy.com.br/course?courseid=deep-learning-frameworks">Deep Learning Frameworks</a> e aplicado em PLN no curso <a href="https://www.datascienceacademy.com.br/course?courseid=processamento-de-linguagem-natural-e-reconhecimento-de-voz">Processamento de Linguagem Natural</a>.
## TF-IDF Para Identificação das Palavras Mais Relevantes em Um Livro
TF-IDF significa "Frequência do Termo - Frequência Inversa de Documentos".
Essa é uma técnica para quantificar uma palavra nos documentos; geralmente calculamos um peso para cada palavra, o que significa a importância da palavra no documento e no corpus. Este método é uma técnica amplamente usada em Recuperação de Informação e Mineração de Texto.
Se eu lhe der uma frase, por exemplo, "Este edifício é tão alto". É fácil para nós entender a sentença como conhecemos a semântica das palavras e da sentença. Mas como o computador entenderá essa frase? O computador pode entender qualquer dado apenas na forma de valor numérico.
Portanto, por esse motivo, vetorizamos todo o texto para que o computador possa entender melhor o texto.
Ao vetorizar os documentos, podemos executar várias tarefas, como encontrar documentos relevantes, classificação, agrupamento e assim por diante. É a mesma coisa que acontece quando você realiza uma pesquisa no Google. As páginas da web são chamadas de documentos e o texto com o qual você pesquisa é chamado de consulta. o Google mantém uma representação fixa para todos os documentos. Quando você pesquisa com uma consulta, o Google encontra a relevância da consulta com todos os documentos, classifica-os na ordem em que é relevante e mostra os principais documentos. Todo esse processo é feito usando a forma vetorizada da consulta e dos documentos. Embora os algoritmos do Google sejam altamente sofisticados e otimizados, essa é a estrutura usada.
Vamos extrair um livro inteiro, construir nossas funções para TF e IDF e então identificar as palavras mais relevantes em algumas frases do livro.
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install torch==1.5.0
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Imports
import nltk
import numpy as np
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
### Preparando os Dados
https://www.gutenberg.org/files/158/158-h/158-h.htm
# ->
# Carrega os dados
dados_livro_emma = nltk.corpus.gutenberg.sents('austen-emma.txt')
# ->
# Listas para receber as frases e as palavras do texto
dados_livro_emma_frases = []
dados_livro_emma_palavras = []
# ->
# Loop para a tokenização
# https://docs.python.org/3/library/stdtypes.html
for sentence in dados_livro_emma:
    dados_livro_emma_frases.append([word.lower() for word in sentence if word.isalpha()])
    for word in sentence:
        if word.isalpha():
            dados_livro_emma_palavras.append(word.lower())
# ->
# Vamos converter a lista de palavras em um conjunto (set)
dados_livro_emma_palavras = set(dados_livro_emma_palavras)
# ->
# Visualiza as frases
dados_livro_emma_frases
# ->
# Visualiza as palavras
dados_livro_emma_palavras
### Frequência do Termo
A Frequência do Termo mede a frequência de uma palavra em um documento.
```


Isso depende muito do tamanho do documento e da generalidade da palavra, por exemplo, uma palavra muito comum como "era" pode aparecer várias vezes em um documento, mas se pegarmos dois documentos, um com 100 palavras e outro com 10.000, há uma alta probabilidade de que uma palavra comum como "era" possa estar mais presente no documento de 10.000 palavras. Mas não podemos dizer que o documento mais longo é mais importante que o documento mais curto. Por esse exato motivo, realizamos uma normalização no valor da frequência, dividindo a frequência com o número total de palavras no documento.

Lembre-se de que precisamos finalmente vetorizar o documento. Quando estamos planejando vetorizá-lo, não podemos considerar apenas as palavras que estão presentes nesse documento específico. Se fizermos isso, o comprimento do vetor será diferente para os dois documentos e não será possível calcular a semelhança. Então, o que fazemos é que vetorizar os documentos no vocabulário, que é a lista de todas as palavras possíveis no corpus.

Quando estamos vetorizando os documentos, verificamos a contagem de cada palavra. Na pior das hipóteses, se o termo não existir no documento, esse valor de TF específico será 0 e, em outro caso extremo, se todas as palavras no documento forem iguais, será 1. O valor final do documento normalizado estará no intervalo de [0 a 1], sendo 0, 1 inclusive.

$tf(t, d) = \text{contagem de } t \text{ em } d / \text{número de palavras em } d$

Se já calculamos o valor do TF e se isso produz uma forma vetorizada do documento, por que não usar apenas o TF para encontrar a relevância entre os documentos? Por que precisamos da IDF?

Embora tenhamos calculado o valor do TF, ainda existem alguns problemas, por exemplo, palavras mais comuns como "é" terão valores muito altos, dando a essas palavras uma importância muito alta. Mas usar essas palavras para calcular a relevância produz maus resultados.

Esse tipo de palavra comum é chamado de palavras de parada (stop words) e, embora removamos as stop words posteriormente na etapa de pré-processamento, descobrir a importância da palavra em todos os documentos e normalizando esse valor, representa muito melhor os documentos.

->

Função para calcular a Termo Frequência

```
def TermFreq(documento, palavra):
    doc_length = len(documento)
    ocorrencias = len([w for w in documento if w == palavra])
    return ocorrencias / doc_length
```

->

dados_livro_emma_frases[5]

->

Aplica a função

TermFreq(dados_livro_emma_frases[5], 'mother')

Podemos então criar um dicionário (vocabulário).

Cada palavra única será identificada de forma única no objeto de dicionário. Isso é necessário para criar representações de textos. O corpus Bag of Words é criado e será necessário para a construção do modelo TF-IDF.

O dicionário é criado pela lista de palavras. As frases/documentos, etc., podem ser convertidos em uma lista de palavras e depois alimentados nos corpora como parâmetro.

->

Criamos um corpus Bag of words

```
def cria_dict():
    output = {}
    for word in dados_livro_emma_palavras:
        output[word] = 0
    for doc in dados_livro_emma_frases:
        if word in doc:
            output[word] += 1
    return output
```

->

Cria o dicionário

df_dict = cria_dict()

->

Filtra o dicionário

df_dict['mother']

Frequência Inversa

Para compreender o que é IDF, primeiro temos que compreender o que é DF.

A **Frequência do Documento** (DF) mede a importância do documento em todo o corpus e é muito semelhante ao TF. A única diferença é que TF é contador de frequência para um termo t no documento d , onde DF é a contagem de ocorrências do termo t no conjunto de documentos N .

Em outras palavras, DF é o número de documentos em que a palavra está presente. Consideramos uma ocorrência se o termo consistir no documento pelo menos uma vez, não precisamos saber o número de vezes que o termo está presente.

$df(t) = \text{ocorrência de } t \text{ nos documentos}$

Para manter isso também em um intervalo, normalizamos dividindo com o número total de documentos. Nosso principal objetivo é conhecer a informatividade de um termo, e DF é o inverso exato dele. É por isso que invertemos o DF.

Frequência Inversa de Documentos

IDF é o inverso da frequência do documento que mede a informatividade do termo t . Quando calcularmos o IDF, será muito baixo para as palavras que mais ocorrem, como stop words (porque stop words como "é" estão presentes em quase todos os documentos e N / df atribuirá um valor muito baixo a essa palavra). Isso finalmente resulta o que queremos, uma ponderação relativa.

$idf(t) = N / df$

Agora, existem alguns outros problemas com o IDF, no caso de um corpus grande, digamos 10.000, o valor do IDF explode.

Então, para diminuir o efeito, aplicamos o log ao IDF.

Durante o tempo de consulta, quando uma palavra que não está no vocabulário ocorre, o df será 0. Como não podemos dividir por 0, suavizamos o valor adicionando 1 ao denominador.

$idf(t) = \log(N / (df + 1))$

Finalmente, considerando um valor multiplicativo de TF e IDF, obtemos a pontuação TF-IDF, existem muitas variações diferentes de TF-IDF, mas por enquanto vamos nos concentrar nessa versão básica.

$tf-idf(t, d) = tf(t, d) * \log(N / (df + 1))$

->

Função para calcular a Frequência Inversa de Documentos

```
def InverseDocumentFrequency(word):
    N = len(dados_livro_emma_frases)
    try:
```

```

    df = df_dict[word] + 1
except:
    df = 1
return np.log(N/df)
# ->
# Aplica a função
InverseDocumentFrequency('mother')
#### TF/IDF
# ->
# Função TF-IDF
def TFIDF(doc, word):
    tf = TermFreq(doc, word)
    idf = InverseDocumentFrequency(word)
    return tf * idf
# ->
dados_livro_emma_frases[5]
# ->
# Print
print('mother: ' + str(TFIDF(dados_livro_emma_frases[5], 'mother')))
# ->
dados_livro_emma_frases[30]
# ->
# Print
print('mother: ' + str(TFIDF(dados_livro_emma_frases[30],
'mother'))))
# Fim

```

Estudo de Caso – Inteligência Artificial Para Previsão de Sentenças em Embargos de Declaração

Neste estudo de caso vamos trabalhar com mais um modelo de PLN usando o PyTorch como nosso framework. Vamos receber como entrada um Embargo de Declaração (texto), preparar os dados, construir e treinar um modelo de rede neural artificial profunda (Deep Learning) e fazer previsões de sentenças com o modelo treinado. O Jupyter Notebook completo você encontra ao final do capítulo. Mais a frente no curso teremos um capítulo sobre Deep Learning e o objetivo aqui é mostrar a você exemplos de aplicações de PLN. De qualquer modo, o Deep Learning Book trata de diversos conceitos que podem ajudar na compreensão do Estudo de Caso, se considerar necessário: <http://www.deeplearningbook.com.br>

O que são embargos de declaração?

Os embargos de declaração são um instrumento jurídico por meio do qual uma das partes pode pedir esclarecimentos ao juiz ou tribunal sobre a decisão judicial proferida. Também conhecidos como embargos declaratórios, por meio deles é possível resolver dúvidas causadas por contradições ou obscuridades. Do mesmo modo, pode-se suprir omissões ou, ainda, apontar erros materiais.

Entretanto, merece esclarecimento uma das principais dúvidas sobre os embargos de declaração. Afinal, eles são ou não uma forma de recurso? Segundo o Novo Código de Processo Civil, sim, uma vez que estão incluídos no rol de recursos no Novo CPC, em seu art. 994. Apesar disso, esta não é uma inovação. De fato, os embargos de declaração já estavam incluídos entre os recursos desde o Código anterior, em seu art. 496, inciso IV. Aqui você encontra mais detalhes:

<https://blog.sajadv.com.br/embargos-de-declaracao-novo-cpc/>

E como vamos criar nossa solução de IA? Usando um modelo CBoW. Vamos compreender alguns conceitos e definir o que é um modelo CBoW (é uma rede neural artificial). Acompanhe a leitura do texto abaixo.

Word Embedding

A Word Embedding é uma das representações mais populares do vocabulário de um documento. É capaz de capturar o contexto de uma palavra em um documento, semelhança semântica e sintática, relação com outras palavras, etc.

Em termos gerais, as Word Embeddings são representações vetoriais de uma palavra específica. Word Embedding é apenas uma maneira elegante de dizer “representação numérica de palavras”. Uma boa analogia seria como usamos a representação RGB para cores.

Por que precisamos de Word Embeddings?

Como humano, intuitivamente falando, não faz muito sentido querer representar palavras ou qualquer outro objeto no universo usando números porque os números são usados para quantificação e por que seria necessário quantificar palavras?

Quando na ciência, dizemos que a velocidade do meu carro é 50 km/h, percebemos o quão rápido / lento estamos dirigindo. Se dissermos que meu amigo está dirigindo a 70 km/h, podemos comparar qual de nós está indo mais rápido. Além disso, podemos calcular onde estaremos em um determinado momento, quando chegaremos ao nosso destino, pois sabemos a distância de nossa jornada, etc.

Da mesma forma, fora da ciência, usamos números para quantificar uma qualidade; quando citamos o preço de um objeto, tentamos quantificar seu valor, o tamanho de uma peça de roupa, tentamos quantificar as proporções corporais que se ajustam melhor.

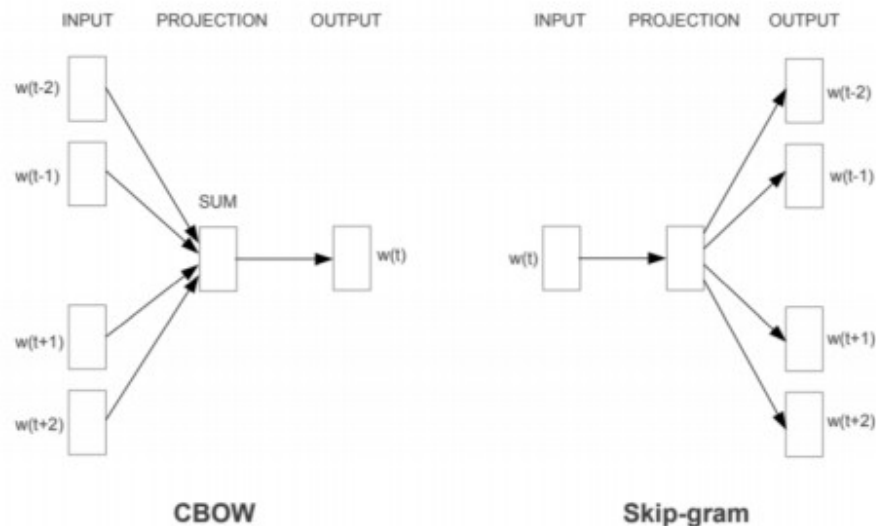
Todas essas representações fazem sentido porque, usando números, facilitamos muito a análise e a comparação com base nessas qualidades. Quanto vale um sapato ou uma bolsa? Bem, por mais diferentes que sejam esses dois objetos, uma maneira de responder é comparar os preços. Além do aspecto da quantificação, não há mais nada a ganhar com essa representação.

Agora que sabemos que a representação numérica de objetos ajuda na análise ao quantificar uma certa qualidade, a questão é que qualidade de palavras queremos quantificar?

A resposta para isso é que queremos quantificar a semântica. Queremos representar as palavras de tal maneira que capte seu significado da maneira que os humanos fazem. Não é o significado exato da palavra, mas contextual.

Continuous Bag of Words Model (CBOW) e Skip-gram

Ambas são arquiteturas para aprender as representações de palavras usando redes neurais, atualmente uma das principais técnicas de IA.



No modelo CBOW, as representações distribuídas de contexto (ou palavras circundantes) são combinadas para prever a palavra no meio. Enquanto no modelo Skip-gram, a representação distribuída da palavra de entrada é usada para prever o contexto.

Um pré-requisito para qualquer rede neural ou qualquer técnica de treinamento supervisionado é ter dados de treinamento rotulados. Como você treina uma rede neural para prever a Embedding de palavras quando não possui dados rotulados, como palavras e a Embedding de palavras correspondente? É onde entra o modelo Skip-gram.

Faremos isso criando uma tarefa "falsa" para a rede neural treinar. Não estaremos interessados nas entradas e saídas dessa rede, mas o objetivo é apenas aprender os pesos da camada oculta que são os "vetores de palavras" que estamos tentando aprender.

A tarefa “falsa” do modelo Skip-gram seria dada uma palavra, tentaremos prever as palavras vizinhas. Definiremos uma palavra vizinha pelo tamanho da janela - um hiperparâmetro. Essa imagem abaixo demonstra como poderíamos pré-processar os dados de texto para treinar o modelo:

Source Text	Training Samples generated from source text			
I will have orange juice and eggs for breakfast	(will, I)	(will, have)	(will, orange)	
I will have orange juice and eggs for breakfast	(have, I)	(have, will)	(have, orange)	(have, juice)
I will have orange juice and eggs for breakfast	(orange, will)	(orange, have)	(orange, juice)	(orange, and)
I will have orange juice and eggs for breakfast	(juice, have)	(juice, orange)	(juice, and)	(juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange)	(and, juice)	(and, eggs)	(and, for)
I will have orange juice and eggs for breakfast	(eggs, juice)	(eggs, and)	(eggs, for)	(eggs, breakfast)
I will have orange juice and eggs for breakfast	(for, and)	(for, eggs)	(for, breakfast)	

As dimensões do vetor de entrada serão $1 \times V$ - onde V é o número de palavras no vocabulário - ou seja, uma representação One-Hot Encoding da palavra. A única camada oculta terá a dimensão $V \times E$, onde E é o tamanho da word embedding e é um hiperparâmetro. A saída da camada oculta seria de dimensão $1 \times E$, que alimentaremos em uma camada softmax. As dimensões da camada de saída serão de $1 \times V$, onde cada valor no vetor será a pontuação de probabilidade da palavra-alvo nessa posição. O aprendizado é feito com o algoritmo Backpropagation.

Esse é um tema avançado e por isso é estudado em detalhes na Formação Inteligência Artificial e não na Formação Cientista de Dados.

Mas no Jupyter Notebook “06-DSA-Cap12-CBOW.ipynb” você encontra um exemplo completo, comentado linha a linha.

Execute o Jupyter Notebook, leia os comentários e tenha uma visão geral de como funciona esse modelo de PLN.

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso <a href="https://www.datascienceacademy.com.br/course?courseid=deep-learning-frameworks">Deep Learning Frameworks</a> e aplicado em PLN no curso <a href="https://www.datascienceacademy.com.br/course?courseid=processamento-de-linguagem-natural-e-reconhecimento-de-voz">Processamento de Linguagem Natural</a>.
### Estudo de Caso - Inteligência Artificial Para Previsão de Sentenças em Embargos de Declaração

**A definição deste estudo de caso está no manual em pdf no Capítulo 12 do Curso de <a href="https://www.datascienceacademy.com.br/course?courseid=machine-learning-engineer">Machine Learning</a>**.
Faça a leitura do manual antes de prosseguir com o Estudo de Caso.
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install torch==1.5.0
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Instala o PyTorch
!pip install -q torch
# ->
# Imports
import torch
import torch.nn as nn
import numpy as np
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
### Preparando os Dados
O texto abaixo é um exemplo de embargo de declaração. Embora o texto represente um embargo, dados críticos foram substituídos por informações genéricas, o que não compromete o objetivo do estudo de caso.
# ->
# Texto de embargo de declaração
embargo = """O embargante sofreu o ajuizamento de ação de danos morais e materiais, cujo objeto é o reaver os valores pagos pelo sinal dado em um contrato de compra e venda de imóvel no qual não foi dada continuidade. Em 24 de fevereiro de 2012, o Magistrado proferiu decisão de fls. 277 a 280, que condenou todas as demandadas solidariamente no seguinte teor: Diante de todo o exposto, com fundamento no art. 1234, I, do CPC/2015, julgo procedentes em parte os pedidos constantes na inicial, condenando solidariamente as demandadas, XPTO LTDA, BOB CAMARGO DE MORAES, a Pagarema título de indenização por danos morais, consoante fundamentação acima discorrida, o montante de R$ 1.500,00 (um mil e quinhentos reais), corrigidos monetariamente pelo INPC desde a data desta decisão, acrescidos de juros de 1% ao mês, a partir da citação; condeno ainda, à restituição do valor pago pelo demandante como sinal da entrada do imóvel, descontando apenas 20% (vinte por cento), referente às despesas, devendo incidir juros de 1% (um por cento) ao mês contados da citação e correção monetária pelo INPC a partir da sentença. Contudo, data venia, houve omissão e obscuridade na referida decisão, haja vista que a omissão se deu pela ausência dos julgamentos das preliminares (Necessidade de Perícia Técnica e a incompetência de Juizado Especial) proposta posteriormente em aditamento de contestação (Fls 251 a 254) para impugnar áudios
```

juntados pelo embargado, autorizado a ser realizada pela Douta Magistrada em audiência de Conciliação, instrução e julgamento de fls 235 e 236, por ausência de intimação anterior para realizar a já tratada impugnação aos áudios anexados. ""

```
# ->
# Limpeza do texto substituindo vírgulas e pontos por espaços e colocando as palavras em minúsculo
embargo = embargo.replace(',',' ').replace('.', ' ').lower().split()
# ->
# Criação do corpus com o texto acima
corpus = set(embargo)
# ->
# Visualizamos o corpus
corpus
# ->
# Comprimento do corpus
corpus_length = len(corpus)
# ->
# Dicionários para TF-IDF
dic_palavra = {}
dic_inverso_palavra = {}
# ->
# Loop pelo corpus para criar os dicionários
for i, palavra in enumerate(corpus):
    dic_palavra[palavra] = i
    dic_inverso_palavra[i] = palavra
# ->
# Lista para receber os dados
dados = []
# ->
# Loop pelo texto par extrair sentenças e palavras
for i in range(2, len(embargo) - 2):
    sentence = [embargo[i-2], embargo[i-1], embargo[i+1], embargo[i+2]]
    target = embargo[i]
    dados.append((sentence, target))
Você leu o código acima e compreendeu o que foi feito? Observe esta linha:
sentence = [embargo[i-2], embargo[i-1], embargo[i+1], embargo[i+2]]
Para uma palavra no índice i, obtemos duas palavras antes e duas palavras depois. A palavra no índice i será o nosso target e a sentença será composta das duas palavras e duas palavras depois da palavra target.
Após treinar o modelo, seremos capazes de prever cada palavra com base nas palavras a sua volta.
Aqui um exemplo:
# ->
# Visualiza os dados
print(dados[3])
As quatro palavras na lista serão os dados de entrada e a palavra fora da lista ('de' nesse caso), será a variável de saída.
### Construção do Modelo CBoW
# ->
# Vamos definir o comprimento de cada embedding
embedding_length = 20
# ->
# Classe para o modelo
class CBoW(torch.nn.Module):
    # Método construtor
    def __init__(self, corpus_length, embedding_dim):
        super(CBoW, self).__init__()
        # Camada de entrada do modelo para criação da embedding
        self.embeddings = nn.Embedding(corpus_length, embedding_dim)
        # Camadas lineares
        self.linear1 = nn.Linear(embedding_dim, 64)
        self.linear2 = nn.Linear(64, corpus_length)
        # Camadas de ativação
        self.activation_function1 = nn.ReLU()
        self.activation_function2 = nn.LogSoftmax(dim = -1)
    # Passo (forward)
    def forward(self, inputs):
        # Aqui definimos a ordem ds camadas da rede neural
        embeds = sum(self.embeddings(inputs)).view(1,-1)
        out = self.linear1(embeds)
        out = self.activation_function1(out)
        out = self.linear2(out)
        out = self.activation_function2(out)
        return out
    # Obtém a word_embedding
    def get_word_embedding(self, word):
        word = torch.LongTensor([dic_palavra[word]])
        return self.embeddings(word).view(1,-1)
# ->
# Cria o modelo CBoW
modelo = CBoW(corpus_length, embedding_length)
# ->
```

```

# Função de custo
loss_function = nn.NLLLoss()
# ->
# Otimizador do modelo (backpropagation)
optimizer = torch.optim.SGD(modelo.parameters(), lr = 0.01)
# ->
# Função para criar o vetor de sentenças, necessário para treinar o modelo
def make_sentence_vector(sentence, word_dict):
    idxs = [word_dict[w] for w in sentence]
    return torch.tensor(idxs, dtype = torch.long)
# ->
# Aqui está nosso dicionário de palavras
dic_palavra
# ->
# O dicionário de palavras será convertido em um vetor de sentenças. Aqui um exemplo:
print(make_sentence_vector(['pela','ausência','dos','julgamentos'], dic_palavra))
#### Treinamento do Modelo
# ->
# Loop por 150 passadas (epochs) de treinamento
for epoch in range(150):
    # Inicia o erro da época com 0
    epoch_loss = 0
    # Loop pelos dados de entrada (sentence) e saída (target)
    for sentence, target in dados:
        # Inicializa os gradientes com zero
        modelo.zero_grad()
        # Cria o vetor de sentença com os dados de entrada (que devem estar no dicionário de palavras)
        sentence_vector = make_sentence_vector(sentence, dic_palavra)
        # Usa o vetor para fazer previsões com o modelo e retorna as probabilidades
        log_probs = modelo(sentence_vector)
        # Calcula o erro do modelo
        loss = loss_function(log_probs, torch.tensor([dic_palavra[target]], dtype = torch.long))
        # Chama o método de backpropagation para calcular o gradiente da derivada
        loss.backward()
        # Otimiza os pesos do modelo e segue para a próxima passada
        # É aqui que o aprendizado acontece
        optimizer.step()
        # Atualiza o erro da época
        epoch_loss += loss.data
    # Imprime epoch e erro da epoch
    print('Epoch: ' + str(epoch) + ', Erro do Modelo: ' + str(epoch_loss.item()))

```

Observe como o erro foi reduzido a cada passada, nitidamente o aprendizado ocorrendo. Vamos agora usar o modelo para fazer previsões.

```

# ->
# Função para obter uma previsão
def get_resultado_previsto(input, dic_inverso_palavra):
    index = np.argmax(input)
    return dic_inverso_palavra[index]
# ->
# Função para prever sentenças (aplicamos aos novos dados o mesmo tratamento usado nos dados de treino)
def preve_sentenca(sentence):
    # Dividimos a sentença com split
    sentence_split = sentence.replace('.', '').lower().split()
    # Criamos o vetor de sentença
    sentence_vector = make_sentence_vector(sentence_split, dic_palavra)
    # Faz a previsão com o modelo
    prediction_array = modelo(sentence_vector).data.numpy()
    # Print dos resultados
    print('Palavras Anteriores: {} \n'.format(sentence_split[:2]))
    print('Palavra Prevista: {} \n'.format(get_resultado_previsto(prediction_array[0], dic_inverso_palavra)))
    print('Palavras Seguintes: {} \n'.format(sentence_split[2:]))

```

Previsões com o Modelo

Dentro da frase: **ausência de intimação anterior para realizar**, vejamos se o modelo consegue prever a palavra.
Vou omitir a palavra **intimação** e essa deve ser a palavra prevista pelo modelo. Vamos passar como dados de entrada as duas palavras anteriores e as duas palavras posteriores.

```

# ->
# Previsão com o modelo
preve_sentenca('ausência de anterior para')
# ->
# Emdedding da palavra
print(modelo.get_word_emdedding('intimação'))

```

Perfeito! O modelo fez a previsão da sentença no Embargo de Declaração! Mais um exemplo.

Dentro da frase: **devendo incidir juros de 1%**, vejamos se o modelo consegue prever a palavra.
Vou omitir a palavra **juros** e essa deve ser a palavra prevista pelo modelo. Vamos passar como dados de entrada as duas palavras anteriores e as duas palavras posteriores.

```

# ->
# Previsão com o modelo
preve_sentenca('devendo incidir de 1%')

```

Perfeito! O modelo fez a previsão da sentença no Embargo de Declaração! E o CBoW não é o modelo mais avançado em PLN.
Fim

Estudo de Caso – Buscador de Palavras em Texto por Similaridade

Trabalhar com palavras é sempre um desafio. Mesmo para um pequeno corpus, sua rede neural (ou qualquer tipo de modelo) precisa suportar milhares de entradas e saídas discretas.

Além das palavras numéricas brutas, a técnica padrão de representar palavras como vetores one-hot (por exemplo, "um" = [0 0 0 1 0 0 0 ...]) não captura nenhuma informação sobre relacionamentos entre palavras.

Os vetores de palavras (word embeddings) solucionam esse problema, representando palavras em um espaço vetorial multidimensional. Isso pode levar a dimensionalidade do problema de centenas de milhares para apenas centenas. Além disso, o espaço vetorial é capaz de capturar relações semânticas entre as palavras em termos de distância e aritmética vetorial.

Existem algumas técnicas para criar vetores de palavras. O algoritmo Word2vec prevê palavras em um contexto (por exemplo, qual é a palavra mais provável na frase "O gato _____ no telhado"), enquanto os vetores GloVe são baseados em contagens globais em todo o corpus.

Usaremos o GloVe agora neste estudo de caso e o Word2vec na próxima aula.

Modelo GloVe

O GloVe é uma técnica de vetor de palavras (embeddings). Os vetores de palavras colocam as palavras em um espaço vetorial, onde palavras semelhantes se agrupam e palavras diferentes se repelem. A vantagem do GloVe é que, diferentemente do Word2vec, o GloVe não depende apenas de estatísticas locais (informações de contexto local das palavras), mas incorpora estatísticas globais (co-ocorrência de palavras) para obter vetores de palavras. Mas há bastante sinergia entre o GloVe e o Word2vec.

E não se surpreenda ao saber que a ideia de usar estatísticas globais para derivar relacionamentos semânticos entre palavras remonta a um longo caminho. GloVe significa "Global Vectors" ou "Vetores Globais". E, como mencionado anteriormente, o GloVe captura estatísticas globais e estatísticas locais de um corpus, a fim de criar vetores de palavras. Mas precisamos de estatísticas globais e locais?

Acontece que cada tipo de estatística tem sua própria vantagem. Por exemplo, o Word2vec, que captura estatísticas locais, se sai muito bem em tarefas de analogia. No entanto, um método como o LSA (Latent Semantic Analysis), que usa apenas estatísticas globais, não funciona bem em tarefas de analogia.

Como Funciona o GloVe

Dado um corpus com V palavras, a matriz de co-ocorrência X será uma matriz $V \times V$, onde a i -linha e a j -ésima coluna de X , X_{ij} indica quantas vezes a palavra i co-ocorreu com a palavra j . Um exemplo de matriz de co-ocorrência pode ter a seguinte aparência.

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

Como obtemos uma métrica que mede a similaridade semântica entre as palavras? Para isso, você precisará de três palavras por vez. Deixe-me apresentar concretamente esta afirmação.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k ice)/P(k steam)$	8.9	8.5×10^{-2}	1.36	0.96

Onde:

$$P_{ik} / P_{jk} \text{ em que } P_{ik} = X_{ik} / X_i$$

Aqui P_{ik} denota a probabilidade de ver as palavras i e k juntas, o que é calculado dividindo o número de vezes que i e k apareceram juntas (X_{ik}) pelo número total de vezes que as palavras apareciam no corpus (X_i).

Você pode ver que, dadas duas palavras, ou seja, ice (gelo) e steam (vapor), se a terceira palavra k é muito semelhante a ice, mas irrelevante a steam (por exemplo, $k = fashion$), P_{ik} / P_{jk} será muito alto (> 1), e muito semelhante a steam, mas irrelevante para ice (por exemplo, $k = gas$), P_{ik} / P_{jk} será muito pequeno (< 1), e se estiver relacionado ou não relacionado a nenhuma das palavras, o P_{ik} / P_{jk} estará próximo de 1.

Portanto, se pudermos encontrar uma maneira de incorporar P_{ik} / P_{jk} na computação de vetores de palavras, alcançaremos o objetivo de usar estatísticas globais ao aprender vetores de palavras.

E é exatamente isso que faz o modelo GloVe. No link abaixo você encontra o paper original:

<https://nlp.stanford.edu/pubs/glove.pdf>

E ao final do capítulo você encontra o Jupyter Notebook “07-DSA-Cap12-GloVe.ipynb” com o modelo completo. Leia atentamente cada célula, execute o notebook e experimente o GloVe.

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
```

Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso Deep Learning Frameworks e aplicado em PLN no curso Processamento de Linguagem Natural.

Estudo de Caso - Buscador de Palavras em Texto Por Similaridade

![title](imagens/buscador.png)

A definição deste estudo de caso está no manual em pdf no Capítulo 12 do Curso de Machine Learning.

Faça a leitura do manual antes de prosseguir com o Estudo de Caso.

->

Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:

pip install -U nome_pacote

Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:

!pip install torch==1.5.0

Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.

Instala o pacote watermark.

Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.

!pip install -q -U watermark

->

Instala o PyTorch

!pip install -q torch

->

Imports

import torch

import numpy as np

import torch.nn as nn

import torch.optim as optim

import matplotlib

import matplotlib.pyplot as plt

from tqdm import tqdm

from torch.autograd import Variable

from nltk.tokenize import word_tokenize

%matplotlib inline

torch.manual_seed(1)

->

Versões dos pacotes usados neste jupyter notebook

%reload_ext watermark

%watermark -a "Data Science Academy" --iversions

Carregando e Processando os Dados

Para este estudo de caso, usaremos o famoso texto de Isaac Asimov: The Last Question.

<http://users.ece.cmu.edu/~gamvrosi/thelastq.html>

Traduzimos o texto e usaremos para treinar o modelo GloVe e depois buscar palavras por similaridade. Recomendados a leitura do arquivo asimov.txt (usado na célula abaixo) antes de executar o restante do Jupyter Notebook.

->

Abre o arquivo para leitura e carrega na variável arquivo_texto

arquivo_texto = open('dados/asimov.txt', 'r')

->

Converte as palavras para minúsculo

texto = arquivo_texto.read().lower()

->

Fecha o arquivo

arquivo_texto.close()

->

Tokenização do texto

texto_token = word_tokenize(texto)

->

Variável para o comprimento total dos tokens

comp_tokens = len(texto_token)

->

print("Número de Tokens: ", comp_tokens)

Criando o Vocabulário

->

Criando o vocabulário

vocab = set(texto_token)

vocab_size = len(vocab)

print("Tamanho do Vocabulário:", vocab_size)

->

Dicionário para mapear as palavras aos índices

palavra_indice = {palavra: i for i, palavra in enumerate(vocab)}

palavra_indice

->

Dicionário para mapear os índices às palavras

indice_palavra = {i: palavra for i, palavra in enumerate(vocab)}

indice_palavra

Salvo indicação em contrário, usamos um contexto de dez palavras à esquerda e dez palavras à direita.

->

Tamanho do contexto

CONTEXT_SIZE = 10

```

# ->
# Matriz de co-ocorrência preenchida com zeros
co_occ_mat = np.zeros((vocab_size, vocab_size))
co_occ_mat
Agora percorremos os dicionários de mapeamento criados anteriormente e preenchemos a matriz de co-ocorrência.
# ->
# Loop externo por todo comprimento do vocabulário
for i in range(comp_tokens):
    # Loop interno pelo tamanho do contexto
    for dist in range(1, CONTEXT_SIZE + 1):
        # Obtém o índice do token
        ix = palavra_indice[texto_token[i]]
        # Se a palavra estiver à esquerda, inserimos à esquerda na matriz de co-ocorrência
        if i - dist > 0:
            left_ix = palavra_indice[texto_token[i - dist]]
            co_occ_mat[ix, left_ix] += 1.0 / dist
        # Se a palavra estiver à direita, inserimos à direita na matriz de co-ocorrência
        if i + dist < len(texto_token):
            right_ix = palavra_indice[texto_token[i + dist]]
            co_occ_mat[ix, right_ix] += 1.0 / dist
# ->
# Matriz de co-ocorrência
co_occ_mat
# ->
# Transposta da matriz de co-ocorrências
# Retorna um array 2-D com uma linha para cada elemento não-zero
co_occs = np.transpose(np.nonzero(co_occ_mat))
# ->
# Print
print("Shape da Matriz de Co-Ocorrência:", co_occ_mat.shape)
# ->
# Print
print("Matriz de Co-Ocorrência Não-Zero:\n", co_occs)
#### Criando o Modelo
# ->
# Tamanho da embedding
EMBEDDING_SIZE = 50
# ->
# Hiperparâmetros
X_MAX = 100
ALPHA = 0.75
BATCH_SIZE = 32
LEARNING_RATE = 0.05
EPOCHS = 200
# ->
# Classe para o modelo
class Glove(nn.Module):
    # Método construtor
    def __init__(self, vocab_size, comat, embedding_size, x_max, alpha):
        super(Glove, self).__init__()
        # Matriz de embeddings com as palavras centrais
        self.embedding_V = nn.Embedding(vocab_size, embedding_size)
        # Matriz de embeddings com as palavras de contexto
        self.embedding_U = nn.Embedding(vocab_size, embedding_size)
        # Bias
        self.v_bias = nn.Embedding(vocab_size, 1)
        self.u_bias = nn.Embedding(vocab_size, 1)
        # Inicializa os parâmetros (pesos que a rede aprende durante o treinamento)
        for params in self.parameters():
            nn.init.uniform_(params, a = -0.5, b = 0.5)
        # Define os hiperparâmetros (que controlam o treinamento)
        self.x_max = x_max
        self.alpha = alpha
        self.comat = comat
    # Função de forward
    def forward(self, center_word_lookup, context_word_lookup):
        # Matrizes embedding de pesos para centro e contexto
        center_embed = self.embedding_V(center_word_lookup)
        target_embed = self.embedding_U(context_word_lookup)
        # Matrizes embedding de bias para centro e contexto
        center_bias = self.v_bias(center_word_lookup).squeeze(1)
        target_bias = self.u_bias(context_word_lookup).squeeze(1)
        # Elementos da matriz de co-ocorrência
        co_occurrences = torch.tensor([self.comat[center_word_lookup[i].item(), context_word_lookup[i].item()]
                                         for i in range(BATCH_SIZE)])
        # Carrega os pesos
        weights = torch.tensor([self.weight_fn(var) for var in co_occurrences])
        # Função de perda

```

```

        loss = torch.sum(torch.pow((torch.sum(center_embed * target_embed, dim = 1)
        + center_bias + target_bias) - torch.log(co_occurrences), 2) * weights)
    return loss
# Definição do peso
def weight_fn(self, x):
    if x < self.x_max:
        return (x / self.x_max) ** self.alpha
    return 1
# Soma de V e U como nossos vetores de palavras
def embeddings(self):
    return self.embedding_V.weight.data + self.embedding_U.weight.data
# ->
# Função para gerar um batch de palavras
def gera_batch(model, batch_size = BATCH_SIZE):
    # Extrai uma amostra
    sample = np.random.choice(np.arange(len(co_occs)), size = batch_size, replace = False)
    # Listas de vetores
    v_vecs_ix, u_vecs_ix = [], []
    # Loop pela amostra para gerar os vetores
    for chosen in sample:
        ind = tuple(co_occs[chosen])
        lookup_ix_v = ind[0]
        lookup_ix_u = ind[1]
        v_vecs_ix.append(lookup_ix_v)
        u_vecs_ix.append(lookup_ix_u)
    return torch.tensor(v_vecs_ix), torch.tensor(u_vecs_ix)
### Treinamento do Modelo
# ->
# Função para o treinamento
def treina_glove(comat):
    # Lista para os erros
    losses = []
    # Cria o modelo Glove
    model = Glove(vocab_size, comat, embedding_size = EMBEDDING_SIZE, x_max = X_MAX, alpha = ALPHA)
    # Otimizador
    optimizer = optim.Adagrad(model.parameters(), lr = LEARNING_RATE)
    # Loop pelo número de épocas
    for epoch in range(EPOCHS):
        # Erro total
        total_loss = 0
        # Número de batches
        num_batches = int(len(texto_token) / BATCH_SIZE)
        # Loop pelos batches
        for batch in tqdm(range(num_batches)):
            # Zera os gradientes do modelo
            model.zero_grad()
            # Obtém o batch de dados
            data = gera_batch(model, BATCH_SIZE)
            # Calcula o erro
            loss = model(*data)
            # Executa o backpropagation
            loss.backward()
            # Otimiza os pesos (aqui é onde ocorre o aprendizado)
            optimizer.step()
            # Erro total para a epoch
            total_loss += loss.item()
        # Erros do modelo
        losses.append(total_loss)
        # Print da epoch e erro médio do modelo
        print('Epoch : %d, Erro Médio : %.02f' % (epoch, np.mean(losses)))
    return model, losses
# ->
# Executa a função de treinamento e retorna o modelo e os erros
model, losses = treina_glove(co_occ_mat)
# ->
# Função para o plot do erro durante o treinamento
def plot_loss(losses, title):
    plt.plot(range(len(losses)), losses)
    plt.xlabel('Epoch')
    plt.ylabel('Erro')
    plt.title(title)
    plt.figure()
# ->
# Plot
plot_loss(losses, "Erro de Treinamento do Modelo GloVe")
### Testando o Modelo: Similaridade de Palavras, analogias de palavras
# ->
# Função que retorna a embedding de uma palavra

```

```

def get_palavra(palavra, modelo, word_to_ix):
    return model.embeddings()[word_to_ix[palavra]]
# ->
# Função para busca a palavra mais próxima
def busca_palavra_similaridade(vec, word_to_ix, n = 10):
    all_dists = [(w, torch.dist(vec, get_palavra(w, model, palavra_indice))) for w in palavra_indice]
    return sorted(all_dists, key = lambda t: t[1])[n]
# ->
# Gerando o vetor (embedding) de uma palavra
vector = get_palavra("espaço", model, palavra_indice)
print(vector)
# ->
# Busca as palavras
similares à palavra "espaço"
busca_palavra_similaridade(vector, palavra_indice)
Observe que a palavra "espaço" tem 0 de distância para si mesma. A próxima palavra mais parecida com "espaço" é "universo" e assim por diante.
Quanto menor a distância, mais parecida a palavra. Lembrando que a busca por similaridade é feita com as embeddings treinadas com o modelo
GloVe.
Mais um exemplo:
# ->
# Gerando o vetor (embedding) de uma palavra
vector = get_palavra("solar", model, palavra_indice)
print(vector)
# ->
# Busca as palavras similares à palavra "solar"
busca_palavra_similaridade(vector, palavra_indice)
A distância da palavra "solar" para si mesma é 0 e a palavra com maior similaridade é "energia" o que faz todo sentido se você leu o texto do
Asimov usado para treinar o modelo.
### Analogia

Observe na imagem acima que criamos uma "fórmula" com 3 palavras visando buscar a quarta palavra, o que é feito por analogia das embeddings
(vetores de palavras).
Criamos então uma função para buscar a palavra por analogia no formato:
palavra1 : palavra2 :: palavra3 : ?
# ->
# Função para busca de palavra por analogia
def busca_analogia(p1, p2, p3, n = 5, filtro = True):
    # Print
    print("\n[%s : %s :: %s : ?]' % (p1, p2, p3))
    # p2 - p1 + p3 = p4
    closest_words = busca_palavra_similaridade(get_palavra(p2, model, palavra_indice) -
                                                get_palavra(p1, model, palavra_indice) +
                                                get_palavra(p3, model, palavra_indice),
                                                palavra_indice)
    # Vamos excluir as 3 palavras passadas como parâmetro
    if filtro:
        closest_words = [t for t in closest_words if t[0] not in [p1, p2, p3]]
    for tuple in closest_words[:n]:
        print('%4f' % tuple[1], tuple[0])
# ->
# Busca por analogia
busca_analogia("família", "crianças", "humano")
E aí estão as palavras que melhor se encaixam na quarta palavra, de acordo com nosso modelo.
Quanto maior a distância, menor a similaridade! Treine o modelo com seus próprios textos e experimente a busca por similaridade.
# Fim

```

Estudo de Caso – Previsão de Palavras com Base no Contexto e Visualização com PCA

O Word2vec é uma rede neural de duas camadas que processa o texto “vetorizando” as palavras. Sua entrada é um corpus de texto e sua saída é um conjunto de vetores; vetores de recursos que representam palavras nesse corpus. Embora o Word2vec não seja uma rede neural profunda, ele transforma o texto em uma forma numérica que as redes neurais profundas podem entender.

O Word2vec é um método de calcular representações vetoriais de palavras e foi desenvolvido por uma equipe de pesquisadores do Google liderada por Tomas Mikolov. O Google hospeda uma versão de código aberto do Word2vec lançada sob uma licença Apache 2.0. Em 2014, Mikolov deixou o Google para o Facebook e, em maio de 2015, foi concedida ao Google uma patente para o método, que não revoga a licença do Apache sob a qual foi lançado. Aqui o paper original:

<https://arxiv.org/pdf/1301.3781.pdf>

As aplicações do Word2vec vão além da análise de sentenças. Também pode ser aplicado a genes, códigos, curtidas em redes sociais, listas de reprodução, gráficos de mídias sociais e outras séries verbais ou simbólicas nas quais os padrões podem ser discernidos.

Por quê? Como as palavras são simplesmente estados discretos, como os outros dados mencionados acima, estamos simplesmente procurando as probabilidades de transição entre esses estados, ou seja, a probabilidade de que elas co-ocorram. Assim, gene2vec, like2vec e follower2vec são todos possíveis.

O objetivo e a utilidade do Word2vec é agrupar os vetores de palavras semelhantes no espaço de vetores. Ou seja, ele detecta semelhanças matematicamente. O Word2vec cria vetores que são representações numéricas distribuídas de recursos de palavras, recursos como o contexto de palavras individuais. Faz isso sem intervenção humana.

Com dados e contextos suficientes, o Word2vec pode fazer suposições altamente precisas sobre o significado de uma palavra com base no contexto. Essas suposições podem ser usadas para estabelecer a associação de uma palavra com outras palavras (por exemplo, "homem" é "garoto" e "mulher" é "garota") ou agrupar documentos e classificá-los por tópico. Esses agrupamentos podem formar a base da pesquisa, análise de sentimentos e recomendações em diversos campos, como pesquisa científica, mineração de documentos legais e jurídicos, comércio eletrônico e gerenciamento de relacionamento com o cliente.

A saída da rede neural do Word2vec é um vocabulário no qual cada item tem um vetor anexado, que pode ser alimentado em uma rede de aprendizado profundo ou simplesmente consultado para detectar relações entre palavras.

Medindo a similaridade do cosseno, nenhuma similaridade seria expressa como um ângulo de 90 graus, enquanto a similaridade total de 1 é um ângulo de 0 graus, ou seja, a palavra Suécia é igual à palavra Suécia, enquanto a palavra Noruega tem uma distância de cosseno de 0,760124 da palavra Suécia, por exemplo.

Similaridade de Cosseno

Entre diferentes métricas de distância, a similaridade de cosseno é mais intuitiva e mais usada no Word2vec. É um produto normalizado de 2 vetores e essa relação define o ângulo entre eles. Dois vetores com a mesma orientação têm uma similaridade de cosseno de 1, dois vetores a 90 ° têm uma similaridade de 0 e dois vetores diametralmente opostos têm uma similaridade de -1, independentemente de sua magnitude.

Uma vez que as palavras são representadas por vetores, a tarefa de encontrar palavras semelhantes ou diferentes torna-se mais fácil. Quaisquer combinações de vetores resultam em um novo vetor e as distâncias do cosseno ou outras medidas de similaridade podem ser usadas. É assim que resolvemos a famosa equação que define o Word2vec:

$$\text{'rei - homem + mulher = rainha'}$$

Com um modelo Word2vec, conseguimos associar palavras com base no seu contexto, usando nossa boa e velha Matemática.

Isso é o que faremos neste Estudo de Caso, que você encontra no Jupyter Notebook “08-DSA-Cap12-Word2vec”. Leia os comentários com atenção, execute as células, modifique os exemplos e compreenda o funcionamento do Word2vec.

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso <a href="https://www.datascienceacademy.com.br/course?courseid=deep-learning-frameworks">Deep Learning Frameworks</a> e aplicado em PLN no curso <a href="https://www.datascienceacademy.com.br/course?courseid=processamento-de-linguagem-natural-e-reconhecimento-de-voz">Processamento de Linguagem Natural</a>.
## Estudo de Caso - Previsão de Palavras com Base no Contexto e Visualização com PCA

**A definição deste estudo de caso está no manual em pdf no Capítulo 12 do Curso de <a href="https://www.datascienceacademy.com.br/course?courseid=machine-learning-engineer">Machine Learning</a>**.
Faça a leitura do manual antes de prosseguir com o Estudo de Caso.
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install torch==1.5.0
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Instala o PyTorch
!pip install -q torch
# ->
# Pacote para gráficos com Scikit-learn
!pip install -q scikit-plot
# ->
# Imports
import torch
import scipy
import sklearn
import scikitplot
import numpy as np
import torch.nn.functional as F
from torch.optim import SGD
from torch.autograd import Variable, profiler
from sklearn.decomposition import PCA
from scipy.spatial.distance import cosine
from scikitplot.decomposition import plot_pca_2d_projection
%matplotlib inline
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
#### Preparação dos Dados
# ->
# Corpus
corpus = ['ele é um rei',
          'ela é uma rainha',
          'ele é um homem',
          'ela é uma mulher',
          'Madrid é a capital da Espanha',
          'Berlim é a capital da Alemanha',
          'Lisboa é a capital de Portugal']
# ->
# Construindo o vocabulário com tokenização
palavras = []
for sentence in corpus:
    for palavra in sentence.split():
        if palavra not in palavras:
            palavras.append(palavra)
# ->
# Visualiza os dados
palavras
# ->
# Criamos o mapeamento palavra - índice
word2idx = {w:idx for (idx, w) in enumerate(palavras)}
word2idx
```

```

# ->
# Criamos o mapeamento inverso índice - palavra
idx2word = {idx:w for (idx, w) in enumerate(palavras)}
idx2word
# ->
# Tamanho do vocabulário
tamanho_vocab = len(word2idx)
tamanho_vocab
### Construção do Modelo
# ->
# Função para gerar os embeddings
def get_word_embedding(word):
    word_vec_one_hot = np.zeros(tamanho_vocab)
    word_vec_one_hot[word2idx[word]] = 1
    return word_vec_one_hot
# ->
# Função para gerar os vetores, da palavra central e do contexto
def gera_vetores():
    for sentence in corpus:
        words = sentence.split()
        indices = [word2idx[w] for w in words]
        # Loop pelo range de índices
        # Aqui geramos o vetor da palavra central em i
        # E geramos o vetor de contexto
        for i in range(len(indices)):
            for w in range(-window_size, window_size + 1):
                context_idx = i + w
                if context_idx < 0 or context_idx >= len(indices) or i == context_idx:
                    continue
                # Gera os vetores
                center_vec_one_hot = np.zeros(tamanho_vocab)
                center_vec_one_hot[indices[i]] = 1
                context_idx = indices[context_idx]
                yield center_vec_one_hot, context_idx
# ->
# Hiperparâmetros
embedding_dims = 10
window_size = 2
Definição dos pesos da rede neural.
- W1 é uma matriz de pesos de dimensões embedding_dims x tamanho_vocab
- W2 é uma matriz de pesos de dimensões tamanho_vocab x embedding_dims
Os pesos (ou coeficientes ou parâmetros) é aquilo que a rede aprende durante o treinamento. Como no início não sabemos qual o valor ideal de pesos (isso é o que queremos descobrir) iniciamos com valores randômicos usando torch.randn().
Ao final do aprendizado, o modelo em si nada mais é do que os valores ideais de W1 e W2.
# ->
# Definição dos pesos da rede neural
W1 = Variable(torch.randn(embedding_dims, tamanho_vocab).float(), requires_grad = True)
W2 = Variable(torch.randn(tamanho_vocab, embedding_dims).float(), requires_grad = True)
# ->
# Treinamento
print("\nIniciando o Treinamento...\n")
for epoch in range(1001):
    # Inicializa o erro médio da rede
    avg_loss = 0
    # Inicializa o controle do número de amostras
    samples = 0
    # Loop pelos dados (vetores de entrada)
    for data, target in gera_vetores():
        # Coleta x (vetor da palavra central)
        x = Variable(torch.from_numpy(data)).float()
        # Coleta y (vetor do contexto)
        y_true = Variable(torch.from_numpy(np.array([target])).long())
        # Atualiza o número de amostras
        samples += len(y_true)
        # Resultado da multiplicação entre os pesos e as primeiras camadas da rede
        a1 = torch.matmul(W1, x)
        a2 = torch.matmul(W2, a1)
        # A função softmax entrega a probabilidade da previsão da rede
        log_softmax = F.log_softmax(a2, dim = 0)
        # Previsão da rede
        network_pred_dist = F.softmax(log_softmax, dim = 0)
        # Calcula o erro, comparando a previsão da rede com o valor real
        # (como fazemos em qualquer modelo de aprendizagem supervisionada)
        loss = F.nll_loss(log_softmax.view(1,-1), y_true)
        # Erro médio
        avg_loss += loss.item()
    # Inicia o backpropagation
    loss.backward()

```



```

# Atualiza o valor dos pesos para a próxima passada
W1.data -= 0.002 * W1.grad.data
W2.data -= 0.002 * W2.grad.data
# Zera o valor do gradiente depois de atualizar os pesos
W1.grad.data.zero_()
W2.grad.data.zero_()
# Imprime o erro da rede
if epoch % 10 == 0:
    print('Erro de Treinamento:', avg_loss / samples)
print("\nTreinamento Concluído.")

```

Teste do Modelo e Redução de Dimensionalidade com PCA

Para testar o modelo, tudo que precisamos é dos pesos, em nosso exemplo W1 e W2. Mas visualizar os dados é desafiador, pois a dimensionalidade é alta e quanto maior o número de palavras do vocabulário, mais complicado. Uma alternativa, é reduzir a dimensionalidade dos dados. Convertemos todos os atributos em 2 componentes principais usando PCA (Principal Component Analysis) e com 2 componentes podemos visualizar os dados. Cada componente principal nada mais é do que a junção matemática da informação em outras variáveis. O PCA é um algoritmo de Machine Learning por si mesmo, da categoria de aprendizagem não supervisionada. Vamos aplicar o PCA para visualizar os dados.

```

# ->
# Cria o objeto para redução de dimensionalidade
pca = PCA(n_components = 2)
# ->
# Treina o modelo PCA
pca.fit(W1.data.numpy().T)
# ->
# Calcula a projeção PCA para o Plot
proj = pca.transform(W1.data.numpy().T)
# ->
# Plot
ax = plot_pca_2d_projection(pca,
                           W1.data.numpy().T,
                           np.array(palavras),
                           feature_labels = palavras,
                           figsize = (16,10),
                           text_fontsize = 12)

```

Legenda

```

for i, txt in enumerate(palavras):
    ax.annotate(txt, (proj[i,0], proj[i,1]), size = 15)

```

Observe a legenda no gráfico acima! Palavras similares com base no contexto, estão com a "bolinha" com cores parecidas. No topo da lista temos países e cidades, depois pronomes e a palavra "capital", temos então homem, mulher, rainha e rei e por fim artigos e um verbo. Tudo isso foi aprendido pela rede com base no contexto, que nada mais é do que a distância de cosseno entre as embeddings, os vetores que representam as palavras.

A visualização acima mostra que palavras que estão na mesma direção possui alguma similaridade, por exemplo "Alemanha" e "Berlim". Passe uma linha reta imaginária que "corta" as palavras "Alemanha" e "Berlim". Consegue? Se a resposta for sim, as palavras são similares. Abaixo terá outro exemplo.

Vamos extrair as distâncias com base na pergunta:

Espanha está para Madrid, assim como Alemanha está para ?

Vamos perguntar ao modelo.

```

# ->
# Função para obter um vetor de palavras no peso W1 (esse é o contexto)
def get_word_vector_v(word):
    return W1[:, word2idx[word]].data.numpy()
# ->
# Função para obter um vetor de palavras no peso W2 (essa é a palavra central)
def get_word_vector_u(word):
    return W2[word2idx[word],:].data.numpy()
# ->
# Vamos obter os vetores das palavras
espanha = 1 * get_word_vector_v('Espanha') + 1 * get_word_vector_u('Espanha')
alemanha = 1 * get_word_vector_v('Alemanha') + 1 * get_word_vector_u('Alemanha')
madrid = 1 * get_word_vector_v('Madrid') + 1 * get_word_vector_u('Madrid')
# ->
# Resultado
resultado = madrid - espanha + alemanha

```

O que fizemos acima foi um cálculo vetorial e toda a Matemática por trás desse processo é estudada em detalhes no curso <https://www.data-science-academy.com.br/course?courseid=matematica-para-machine-learning> Matemática Para Machine Learning.

```

# ->
# Este é o resultado, ou seja, uma embedding que representa a palavra mais similar à palavra "Alemanha",
# com base na similaridade (contexto) entre "Polônia" e "Varsóvia".
resultado
# ->
# Vamos extrair as distâncias de todas as outras palavras para a nossa palavra "secreta" que está
# no vetor embedding
chamado "resultado"
# Usamos a função cosine() do SciPy para calcular as distâncias
distancias = [(v, cosine(resultado, 1 * get_word_vector_u(v) + 1 * get_word_vector_v(v))) for v in palavras]
# ->
# Visualiza as distâncias

```

```

distancias
Acima temos uma lista de tuplas com as distâncias de cada palavra para nosso "resultado". Vamos ordenar isso.
# ->
# Ordenando a lista de tuplas pelo segundo elemento da tupla
distancias.sort(key = lambda tup: tup[1])
# ->
# Agora sim
distancias
O vetor "resultado" foi uma previsão do nosso modelo e as palavras "Madrid" e "Berlim" são as mais similares. Observe que "Berlim" é a palavra
mais similar com base no contexto, uma vez que Madrid já foi usada em nossa fórmula.
Imagine que um vetor (uma flecha) sai da origem do sistema de coordenadas (Honestidade = 0 e Experiência = 0, chamaremos de ponto O) e
termina no ponto X. Este vetor é usado para localizar o ponto no nosso espaço de características. Não é diferente de simplesmente dizer que X
possui H = 0.4 e E = 0.2, é apenas outra maneira de ver isso.
![title](imagens/cos.png)
Se soubermos qual é o ângulo entre os vetores X e A, podemos usar uma calculadora simples e calcular a similaridade. Podemos ver pela imagem
acima que os vetores de X e A estão sobre a mesma linha reta (observe as cores dos círculos na legenda), logo o ângulo entre eles é zero graus e sua
similaridade é  $\cos(0) = 1$ . Já para a similaridade entre X e B não sabemos o ângulo e precisamos usar uma equação. O numerador (a parte de cima
da divisão) significa multiplicar os valores de Honestidade de X e B e somar com a multiplicação dos valores de Experiência de X e B.
Isso é o que faz o Word2vec. Brihante, não?
**Em que contexto aparece a palavra Lisboa?**
Aqui é como se estivéssemos usando o modelo para previsão.
# ->
# Extraí o contexto
context_to_predict = get_word_vector_v('Lisboa')
# Variável com o contexto a prever
hidden = Variable(torch.from_numpy(context_to_predict)).float()
# Executa o modelo e extraí as probabilidades
# (executar o modelo nada mais é do que multiplicar os novos dados de entrada pelos pesos aprendidos no treinamento)
a = torch.matmul(W2, hidden)
probs = F.softmax(a, dim = 0).data.numpy()
# Imprime o resultado
for context, prob in zip(palavras, probs):
    print(f'{context}: {prob}')
O contexto da palavra "Lisboa" é representado pelas palavras "é", "a", "Portugal".
Nosso modelo não conseguiu aprender o contexto "capital". Quem sabe você consegue otimizar o treinamento do modelo e aumentar sua precisão.
# Fim

```

Estudo de Caso – Tradutor de Idioma com Machine Learning e PLN

O Modelo Seq2seq usado neste Estudo de Caso é um modelo avançado e não espere aprender tudo sobre ele em um Estudo de Caso. Não é esse nosso objetivo aqui. O Modelo Seq2Seq é estudado em 3 capítulos no curso de Processamento de Linguagem Natural na DSA e nosso objetivo com este Estudo de Caso é mostrar a você uma das muitas aplicações de Inteligência Artificial na atualidade. Especialização em IA é um caminho natural para quem estuda Machine Learning.

O Seq2seq foi introduzido pela primeira vez para tradução automática, pelo Google. Antes disso, a tradução funcionava de maneira muito ingênua. Cada palavra que você costumava digitar era convertida para o idioma de destino, sem considerar a gramática e a estrutura da frase. O Seq2seq revolucionou o processo de tradução, utilizando o aprendizado profundo (Deep Learning). Ele não apenas leva em consideração a palavra / entrada atual durante a tradução, mas também sua vizinhança.

Atualmente, é usado para uma variedade de aplicações diferentes, como legendas de imagens, modelos de conversação, resumo de texto, tradução, etc.

Modelo Seq2seq

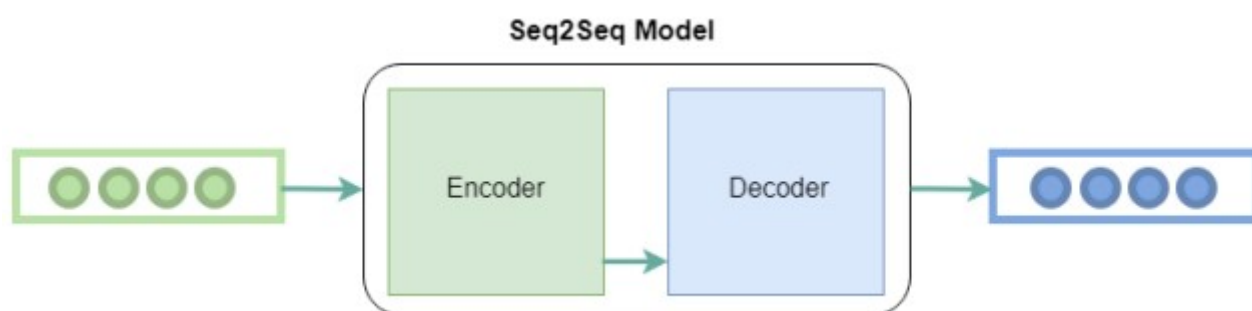
Como o nome sugere, seq2seq usa como entrada uma sequência de palavras (sentença ou sentenças) e gera uma sequência de saída de palavras. Faz isso usando a rede neural recorrente (RNN), sendo comum usarmos versões avançadas da RNN, ou seja, LSTM ou GRU (estudadas no curso Deep Learning II). Isso ocorre porque a RNN sofre com o problema da dissipação do gradiente. O modelo LSTM é usado na versão proposta pelo Google. Ele desenvolve o contexto da palavra, recebendo 2

entradas em cada ponto do tempo. Um atual e outro da saída anterior, daí o nome recorrente (a saída entra como entrada).

O Seq2seq possui principalmente dois componentes: codificador e decodificador, e, portanto, às vezes é chamado de Rede Codificador-Decodificador.

Codificador: Utiliza camadas de rede neural profunda e converte as palavras de entrada em vetores ocultos correspondentes. Cada vetor representa a palavra atual e o contexto da palavra.

Decodificador: É semelhante ao codificador. Toma como entrada o vetor oculto gerado pelo codificador, seus próprios estados ocultos e a palavra atual para produzir o próximo vetor oculto e finalmente prever a próxima palavra.



Além desses dois elementos, muitas otimizações levaram a outros componentes do seq2seq:

Attention: A entrada para o decodificador é um único vetor que deve armazenar todas as informações sobre o contexto. Isso se torna um problema com grandes sequências. Portanto, o mecanismo de atenção é aplicado, permitindo que o decodificador observe a sequência de entrada seletivamente.

Beam Search: A palavra com maior probabilidade é selecionada como saída pelo decodificador. Mas isso nem sempre produz os melhores resultados, devido ao problema básico dos algoritmos gananciosos. Portanto, a pesquisa por feixe é aplicada, o que sugere possíveis traduções em cada etapa. Isso é feito criando uma árvore dos melhores resultados.

Bucketing: Sequências de comprimento variável são possíveis em um modelo seq2seq, devido ao preenchimento de 0, que é feito na entrada e na saída. No entanto, se o comprimento máximo definido por nós for 100 e a sentença tiver apenas 3 palavras, isso causará enorme desperdício de espaço. Então, usamos o conceito de Bucketing. Criamos variáveis de tamanhos diferentes, como (4, 8) (8, 15) e assim por diante, onde 4 é o comprimento máximo de entrada definido por nós e 8 é o comprimento máximo de saída definido.

Seq2seq é um dos modelos mais avançados para PLN e vamos usá-lo para construir um tradutor de idioma.

Isso é o que faremos neste Estudo de Caso, que você encontra no Jupyter Notebook “09-DSA-Cap12-Seq2seq”. Leia os comentários com atenção, execute as células, modifique os exemplos e compreenda o funcionamento do Seq2seq.

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 12 - Processamento de Linguagem Natural</font>
# ->
# Versão da Linguagem Python
```

```

from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
Obs: Este é um material de bônus incluído neste curso. PyTorch é estudado em detalhes no curso <a href="https://www.datascienceacademy.com.br/course?courseid=deep-learning-frameworks">Deep Learning Frameworks</a> e aplicado em PLN no curso <a href="https://www.datascienceacademy.com.br/course?courseid=processamento-de-linguagem-natural-e-reconhecimento-de-voz">Processamento de Linguagem Natural</a>.
## Estudo de Caso - Tradutor de Idioma com Machine Learning e PLN

**A definição deste estudo de caso está no manual em pdf no Capítulo 12 do Curso de <a href="https://www.datascienceacademy.com.br/course?courseid=machine-learning-engineer">Machine Learning</a>**.
Faça a leitura do manual antes de prosseguir com o Estudo de Caso.
O dataset que usaremos oferece texto somente em inglês, alemão e francês. Como nosso objetivo é estudar a construção do modelo e não os idiomas, isso não faz diferença e trabalharemos com tradução inglês/alemão.
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# !pip install torch==1.5.0
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Instala o PyTorch
!pip install -q torch
# ->
# O pacote torchtext fornece diversos datasets e funções para PLN
# https://torchtext.readthedocs.io/en/latest/index.html
!pip install -q torchtext
# ->
# Instala o spacy
!pip install -q spacy
# ->
# Imports
import math
import time
import spacy
import torch
import random
import numpy as np
import torch.nn as nn
import torch.optim as optim
import torchtext
from torchtext.datasets import Multi30k
from torchtext.data import Field, BucketIterator
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
Nota: O treinamento do modelo deste estudo de caso é computacionalmente intensivo e por isso treinamos o modelo no Titan, o super servidor da DSA, com 3 GPUs e 128 GB de Memória RAM. O acesso a esse servidor é gratuito para alunos das Formações:
- <a href="https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial">Formação Inteligência Artificial</a>
- <a href="https://www.datascienceacademy.com.br/pages/formacao-ia-aplicada-a-medicina">Formação Inteligência Artificial Aplicada à Medicina</a>
- <a href="https://www.datascienceacademy.com.br/pages/formacao-engenheiro-blockchain">Formação Engenheiro Blockchain</a>
O treinamento pode ser feito em um computador apenas com CPU. O tempo de treinamento será um pouco maior, mas o estudo de caso poderá ser executado sem problemas.
# ->
# Aqui definimos o device que será usado para treinar o modelo
# Se pelo menos uma GPU estiver disponível, usaremos o device 'cuda' (nome da plataforma da Nvidia para GPU)
# Se não tiver GPU, usaremos a CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
Abaixo a descrição das GPUs do servidor da DSA. O comando abaixo funcionará somente se a plataforma CUDA da Nvidia estiver instalada no computador. Se quiser conhecer mais sobre a plataforma CUDA, acesse aqui:
https://developer.nvidia.com/cuda-toolkit
# ->
# GPUs no servidor da DSA
!nvidia-smi
### Carregando os Dicionários
Precisamos instalar os dicionários dos idiomas que serão usados para treinar o modelo. Aqui você encontra detalhes sobre os datasets:
https://www.statmt.org/wmt16/multimodal-task.html
# ->
# Download do dicionário em inglês
!python -m spacy download en
# ->
# Download do dicionário em alemão
!python -m spacy download de
Agora carregamos os dicionários na memória.

```

```

# ->
# Carregando os dicionários
spacy_german = spacy.load('de')
spacy_english = spacy.load('en')
Vamos criar duas funções para tokenização dos dicionários.
# ->
# Função para tokenização do dicionário em inglês
def tokenize_english(text):
    return [token.text for token in spacy_english.tokenizer(text)][::-1]
# ->
# Função para tokenização do dicionário em alemão
def tokenize_german(text):
    return [token.text for token in spacy_german.tokenizer(text)]
Precisamos agora criar a fonte e o destino, ou seja, o idioma fonte e o idioma destino para nosso tradutor.
Nosso modelo deverá fazer a tradução do inglês para o alemão. Inglês será a fonte (SOURCE) e Alemão será o destino (TARGET).
# ->
# Idioma de origem
SOURCE = Field(tokenize = tokenize_english, init_token = '<sos>', eos_token = '<eos>', lower = True)
# ->
# Idioma de destino
TARGET = Field(tokenize = tokenize_german, init_token = '<sos>', eos_token = '<eos>', lower = True)
# ->
# Usamos a split() do pacote Multi30k do torchtext para separar os dicionários em SOURCE e TARGET
# e então em treino, validação e teste
# Obs: Será feito o download dos dados no pacote Multi30k
dados_treino, dados_valid, dados_teste = Multi30k.splits(exts = ('en', 'de'), fields = (SOURCE, TARGET))
# ->
# Visualizando os dados de treino, SOURCE e TARGET
print(dados_treino.examples[0].src)
print(dados_treino.examples[0].trg)
# ->
print("Tamanho do Dataset de Treino: " + str(len(dados_treino.examples)))
print("Tamanho do Dataset de Validação: " + str(len(dados_valid.examples)))
print("Tamanho do Dataset de Teste: " + str(len(dados_teste.examples)))
# ->
# Vamos criar os vocabulários de SOURCE e TARGET
SOURCE.build_vocab(dados_treino, min_freq = 2)
TARGET.build_vocab(dados_treino, min_freq = 2)
# ->
# Print do tamanho dos vocabulários
print("Tamanho do Vocabulário em Inglês (SOURCE): " + str(len(SOURCE.vocab)))
print("Tamanho do Vocabulário em Alemão (TARGET): " + str(len(TARGET.vocab)))
#### Construindo o Modelo
Criaremos 3 classes:
- Encoder
- Decoder
- Seq2Seq
# ->
# Classe para o Encoder
class Encoder(nn.Module):
    # Método construtor
    def __init__(self, input_dims, emb_dims, hid_dims, n_layers, dropout):
        super().__init__()
        # Camadas do modelo
        self.hid_dims = hid_dims
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_dims, emb_dims)
        self.rnn = nn.LSTM(emb_dims, hid_dims, n_layers, dropout = dropout)
        self.dropout = nn.Dropout(dropout)
    # Método forward para o treinamento
    def forward(self, src):
        # Execução do modelo
        embedded = self.dropout(self.embedding(src))
        outputs, (h, cell) = self.rnn(embedded)
        return h, cell
# ->
# Classe para o Decoder
class Decoder(nn.Module):
    # Método construtor
    def __init__(self, output_dims, emb_dims, hid_dims, n_layers, dropout):
        super().__init__()
        # Camadas do modelo
        self.output_dims = output_dims
        self.hid_dims = hid_dims
        self.n_layers = n_layers
        self.embedding = nn.Embedding(output_dims, emb_dims)
        self.rnn = nn.LSTM(emb_dims, hid_dims, n_layers, dropout = dropout)
        self.fc_out = nn.Linear(hid_dims, output_dims)

```

```

        self.dropout = nn.Dropout(dropout)
# Método forward para o treinamento
def forward(self, input, h, cell):
    # Execução do modelo
    input = input.unsqueeze(0)
    embedded = self.dropout(self.embedding(input))
    output, (h, cell) = self.rnn(embedded, (h, cell))
    pred = self.fc_out(output.squeeze(0))
    return pred, h, cell
# ->
# Classe para o modelo Seq2Seq
class Seq2Seq(nn.Module):
    # Método construtor
    def __init__(self, encoder, decoder, device):
        super().__init__()
        # Componentes do modelo
        self.encoder = encoder
        self.decoder = decoder
        self.device = device
    # Método forward para o treinamento
    def forward(self, src, trg, teacher_forcing_rate = 0.5):
        # Execução do modelo
        batch_size = trg.shape[1]
        target_length = trg.shape[0]
        target_vocab_size = self.decoder.output_dims
        outputs = torch.zeros(target_length, batch_size, target_vocab_size).to(self.device)
        h, cell = self.encoder(src)
        input = trg[0,:]
        for t in range(1, target_length):
            output, h, cell = self.decoder(input, h, cell)
            outputs[t] = output
            top = output.argmax(1)
            input = trg[t] if (random.random() < teacher_forcing_rate) else top
        return outputs

```

Vamos definir alguns hiperparâmetros e os gerados de dados.

```

# ->
# Hiperparâmetros
batch_size = 32
input_dimensions = len(SOURCE.vocab)
output_dimensions = len(TARGET.vocab)
encoder_embedding_dimensions = 256
decoder_embedding_dimensions = 256
hidden_layer_dimensions = 512
num_layers = 2
encoder_dropout = 0.5
decoder_dropout = 0.5
epochs = 20
grad_clip = 1
lowest_validation_loss = float('inf')
# ->
# Geradores de dados
iterador_treino, iterador_valid, iterador_teste = BucketIterator.splits((dados_treino, dados_valid, dados_teste),
                                                                    batch_size = batch_size,
                                                                    device = device)

```

Aqui nós criamos o encoder, decoder e o modelo:

```

# ->
# Instância do Encoder
encod = Encoder(input_dimensions,
                encoder_embedding_dimensions,
                hidden_layer_dimensions,
                num_layers,
                encoder_dropout)
# ->
# Instância do Decoder
decod = Decoder(output_dimensions,
                decoder_embedding_dimensions,
                hidden_layer_dimensions,
                num_layers,
                decoder_dropout)
#
->
# Instância do Modelo
modelo = Seq2Seq(encod, decod, device).to(device)
# ->
# Modelo criado
modelo

```

Vamos definir a função de inicialização dos pesos, função de custo e otimizador.

```

# ->

```

```

# Precisamos de uma função para inicializar os pesos da rede neural
def inicializa_pesos(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param.data, -0.1, 0.1)
# ->
# Incluímos a função de inicialização dos pesos no modelo
modelo.apply(inicializa_pesos)
# ->
# Definimos a função de custo para calcular o erro do modelo
criterion = nn.CrossEntropyLoss(ignore_index = TARGET.vocab.stoi[TARGET.pad_token])
# ->
# Criamos o otimizador para atualizar os pesos do modelo a cada passada de treinamento
optimizer = optim.Adam(modelo.parameters())
Embora não seja obrigatório, criar funções para treino e avaliação do modelo ajuda a modularizar nosso processo de treinamento do modelo.
# ->
# Função para treinar o modelo
def treina_modelo(modelo, iterator, optimizer, criterion, clip):
    # Inicia o método de treinamento
    modelo.train()
    # Inicializa o erro da epoch
    epoch_loss = 0
    # Loop pelo iterador (gerador de dados)
    for i, batch in enumerate(iterator):
        # Coletamos dados fonte e destino
        src = batch.src
        trg = batch.trg
        # Zeramos os gradientes
        optimizer.zero_grad()
        # Fazemos as previsões com o modelo
        output = modelo(src, trg)
        # Ajustamos o shape das previsões
        output_dims = output.shape[-1]
        output = output[1:].view(-1, output_dims)
        trg = trg[1:].view(-1)
        # Calculamos o erro do modelo
        loss = criterion(output, trg)
        # Iniciamos o backpropagation
        loss.backward()
        # Calculamos os gradientes da derivada para atualização dos pesos
        torch.nn.utils.clip_grad_norm_(modelo.parameters(), clip)
        # Aplicamos a atualização dos pesos
        optimizer.step()
        # Armazenamos o erro da epoch
        epoch_loss += loss.item()
    return epoch_loss / len(iterator)
# ->
# Função para avaliar o modelo
def avalia_modelo(modelo, iterator, criterion):
    # Inicia função de avaliação
    modelo.eval()
    # Inicializa o erro da epoch
    epoch_loss = 0
    # Vamos fazer as previsões com o modelo
    with torch.no_grad():
        # Loop pelo iterador (gerador de dados)
        for i, batch in enumerate(iterator):
            # Extrai fonte e destino
            src = batch.src
            trg = batch.trg
            # Previsão com o modelo
            output = modelo(src, trg, 0)
            # Ajusta as dimensões das previsões
            output_dim = output.shape[-1]
            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)
            # Calcula o erro do modelo
            loss = criterion(output, trg)
            # Armazena o erro na epoch
            epoch_loss += loss.item()
    return epoch_loss / len(iterator)
### Treinando o Modelo
O treinamento do modelo é demorado. Seja paciente.
# ->
# Loop pelo número de epochs para treinar o modelo
for epoch in range(epochs):
    # Grava o tempo quando começamos
    start_time = time.time()
    # Treinamento

```

```

train_loss = treina_modelo(modelo, iterador_treino, optimizer, criterion, grad_clip)
# Validação
valid_loss = avalia_modelo(modelo, iterador_valid, criterion)
# Grava o tempo quando finalizamos
end_time = time.time()
# Verificamos o erro mínimo e então salvamos o modelo fazendo um checkpoint do modelo com melhor performance
if valid_loss < lowest_validation_loss:
    lowest_validation_loss = valid_loss
    torch.save(modelo.state_dict(), 'modelos/seq2seq.pt')
# Print
print(f'Epoch: {epoch+1:02} | Time: {np.round(end_time-start_time,0)}s')
print(f'\tErro em Treino: {train_loss:.4f}')
print(f'\t Erro em Validação: {valid_loss:.4f}')
### Avaliando o Modelo
Com o modelo treinado, avaliamos com dados de teste.
# ->
# Carregamos o modelo treinado
modelo.load_state_dict(torch.load('modelos/seq2seq.pt'))
# ->
# Avaliamos o modelo
test_loss = avalia_modelo(modelo, iterador_teste, criterion)
# ->
# Print
print(f'Erro em Teste: {test_loss:.4f}')
### Traduzindo Idioma
Modelo treinado e avaliado, vamos usá-lo para o fim para o qual ele foi criado.
# ->
# Função para tradução de idioma em 5 sentenças
def traduz_idioma(modelo, iterator, limit = 5):
    with torch.no_grad():
        # Loop pelo iterador
        for i, batch in enumerate(iterator):
            # Enquanto estivermos dentro do limite, vamos fazendo tradução
            if i < limit :
                # Extraímos SOURCE e TARGET
                # Fazemos isso para poder comparar a tradução correta com a previsão
                src = batch.src
                trg = batch.trg
                # Previsão do modelo
                output = modelo(src, trg, 0)
                # Todas as previsões
                preds = torch.tensor([torch.argmax(x).item() for x in output])
                # Prints
                print('Texto de Origem em Inglês: ' + str([SOURCE.vocab.itos[x] for x in src][1:-1][:-1]))
                print('Texto de Destino em Alemão (Valor Esperado): ' + str([TARGET.vocab.itos[x] for x in trg][1:-1]))
                print('Texto de Destino em Alemão (Valor Previsto): ' + str([TARGET.vocab.itos[x] for x in preds][1:-1]))
                print('\n')
# ->
# Vamos gerar texto randômico a partir dos dados disponíveis
_, _, iterador_translate = BucketIterator.splits((dados_treino, dados_valid, dados_teste),
                                                batch_size = 1,
                                                device = device)
# ->
# Tradução de idioma
saida = traduz_idioma(modelo, iterador_translate)
Parabéns! Já está seu tradutor de texto com Machine Learning e PLN.
# Fim

```

Mini-Projeto 3 em R – Text Analytics em Recursos Humanos

Neste Mini-Projeto de mineração de texto vamos determinar como os funcionários avaliam a Amazon e o Google.

Vamos determinar qual empresa tem melhor equilíbrio entre trabalho e vida pessoal e uma melhor remuneração de acordo com avaliações de funcionários coletadas do site de vagas Glassdoor.

Carregaremos 2 datasets e faremos diversas atividades de mineração de texto e análise de dados (Text Analytics) e ao final emitiremos nossa conclusão. Qual empresa apresenta melhor equilíbrio entre trabalho e vida pessoal?


```

# Machine Learning

# Mini-Projeto 3 - Text Analytics em Recursos Humanos

# Detalhes sobre o projeto no manual em pdf do Capítulo 12.

# Diretório de trabalho
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap12/R")
getwd()

# Pacotes
install.packages("qdap")
install.packages("RWeka")
install.packages("ggthemes")
library(readr)
library(qdap)
library(tm)
library(RWeka)
library(wordcloud)
library(plotrix)
library(ggthemes)
library(ggplot2)

# Carrega os dados
df_amazon <- read_csv("dados/amazon.csv")
df_google <- read_csv("dados/google.csv")

# Visualiza os dados
View(df_amazon)
View(df_google)

# Tipos de dados
str(df_amazon)
str(df_google)

# Dimensões
dim(df_amazon)
dim(df_google)

# Prós e contras da Amazon
amazon_pros <- df_amazon$pros
amazon_cons <- df_amazon$cons

# Prós e contras do Google
google_pros <- df_google$pros
google_cons <- df_google$cons

# Organização do texto

# Função para limpeza do texto
func_limpa_texto <- function(x){

  x <- na.omit(x)
  x <- replace_abbreviation(x)
  x <- replace_contraction(x)
  x <- replace_number(x)
  x <- replace_ordinal(x)
  x <- replace_symbol(x)
  x <- tolower(x)

  return(x)
}

# Aplicando limpeza aos dados
amazon_pros <- func_limpa_texto(amazon_pros)
amazon_cons <- func_limpa_texto(amazon_cons)
google_pros <- func_limpa_texto(google_pros)
google_cons <- func_limpa_texto(google_cons)

# O próximo passo é converter o vetor contendo os dados de texto em um corpus.
# Corpus é uma coleção de documentos, mas também é importante saber que no pacote tm, R o reconhece
# como um tipo de dado.

# Usaremos o corpus volátil, que é mantido na RAM do computador em vez de salvo no disco, apenas para
# ter mais eficiência de memória.

# Para criar um corpus volátil, R precisa interpretar cada elemento em nosso vetor de texto como um documento.
# O pacote tm fornece funções para fazer exatamente isso!

```

```

# Usaremos uma função Source chamada VectorSource() porque nossos dados de texto estão contidos em um vetor.
?VCorpus
amazon_p_corp <- VCorpus(VectorSource(amazon_pros))
amazon_c_corp <- VCorpus(VectorSource(amazon_cons))
google_p_corp <- VCorpus(VectorSource(google_pros))
google_c_corp <- VCorpus(VectorSource(google_cons))

# Agora aplicamos limpeza ao Corpus

# Função de limpeza do Corpus
func_limpa_corpus <- function(x){

  x <- tm_map(x,removePunctuation)
  x <- tm_map(x,stripWhitespace)
  x <- tm_map(x,removeWords, c(stopwords("en"), "Amazon", "Google", "Company"))

  return(x)
}

# Aplicando a função
amazon_pros_corp <- func_limpa_corpus(amazon_p_corp)
amazon_cons_corp <- func_limpa_corpus(amazon_c_corp)
google_pros_corp <- func_limpa_corpus(google_p_corp)
google_cons_corp <- func_limpa_corpus(google_c_corp)

# Feature Extraction

# Como amzn_pros_corp, amzn_cons_corp, goog_pros_corp e goog_cons_corp foram pré-processados,
# agora podemos extrair os recursos que desejamos examinar.

# Como estamos usando a abordagem do saco de palavras (bag of words), podemos criar um bigrama TermDocumentMatrix
# para o corpus de avaliações positivas e negativas da Amazon.

# A partir disso, podemos criar rapidamente uma nuvem de palavras para entender quais frases as pessoas
# associam positivamente e negativamente ao trabalhar na Amazon.

# Tokenização
tokenizer <- function(x)
?NGramTokenizer
NGramTokenizer(x, Weka_control(min = 2, max = 2))

# Feature extraction e análise de avaliações positivas
amazon_p_tdm <- TermDocumentMatrix(amazon_pros_corp)
amazon_p_tdm_m <- as.matrix(amazon_p_tdm)
amazon_p_freq <- rowSums(amazon_p_tdm_m)
amazon_p_f.sort <- sort(amazon_p_freq, decreasing = TRUE)

# Plot
barplot(amazon_p_freq[1:5])

# Prepara os dados para a wordcloud
amazon_p_tdm <- TermDocumentMatrix(amazon_pros_corp, control = list(tokenize=tokenizer))
amazon_p_tdm_m <- as.matrix(amazon_p_tdm)
amazon_p_freq <- rowSums(amazon_p_tdm_m)
amazon_p_f.sort <- sort(amazon_p_freq,decreasing = TRUE)

# Cria o dataframe de comentários positivos
df_amazon_p <- data.frame(term = names(amazon_p_f.sort), num = amazon_p_f.sort)
View(df_amazon_p)

# Nuvem de palavras
wordcloud(df_amazon_p$term,
          df_amazon_p$num,
          max.words = 100,
          color = "tomato4")

# Feature extraction e análise de avaliações negativas
amazon_c_tdm <- TermDocumentMatrix(amazon_cons_corp, control = list(tokenize = tokenizer))
amazon_c_tdm_m <- as.matrix(amazon_c_tdm)
amazon_c_freq <- rowSums(amazon_c_tdm_m)
amazon_c_f.sort <- sort(amazon_c_freq, decreasing = TRUE)

# Cria o dataframe de comentários negativos
df_amazon_c <- data.frame(term = names(amazon_c_f.sort), num = amazon_c_f.sort)
View(df_amazon_c)

# Nuvem de palavras
wordcloud(df_amazon_c$term,

```

```

df_amazon_c$num,
max.words = 100,
color = "palevioletred")

# Parece que há uma forte indicação de longas horas de trabalho e equilíbrio entre trabalho e
# vida pessoal nas avaliações. Como uma técnica de agrupamento simples, vamos realizar um
# agrupamento hierárquico e criar um dendrograma para ver como essas frases estão conectadas.
amazon_c_tdm <- TermDocumentMatrix(amazon_cons_corp, control = list(tokenize = tokenizer))
amazon_c_tdm <- removeSparseTerms(amazon_c_tdm, 0.993)

# Cria o dendrograma
amazon_c_hclust <- hclust(dist(amazon_c_tdm, method = "euclidean"), method = "complete")

# Plot
plot(amazon_c_hclust)

# Voltando aos comentários positivos, vamos examinar as principais frases que apareceram
# nas nuvens de palavras. Esperamos agora encontrar termos associados usando a função findAssocs() do pacote tm.
amazon_p_tdm <- TermDocumentMatrix(amazon_pros_corp, control = list(tokenize=tokenizer))
amazon_p_m <- as.matrix(amazon_p_tdm)
amazon_p_freq <- rowSums(amazon_p_m)
token_frequency <- sort(amazon_p_freq, decreasing = TRUE)
token_frequency[1:5]

# Encontramos as associações
findAssocs(amazon_p_tdm, "fast paced", 0.2)

# Vamos criar uma nuvem de palavras comparativa das avaliações positivas e negativas do Google para comparação
# com a Amazon. Isso dará uma compreensão rápida dos principais termos.
all_google_pros <- paste(df_google$pros, collapse = "")
all_google_cons <- paste(df_google$cons, collapse = "")
all_google <- c(all_google_pros, all_google_cons)
all_google_clean <- func_limpa_texto(all_google)
all_google_vs <- VectorSource(all_google_clean)
all_google_vc <- VCorpus(all_google_vs)
all_google_clean2 <- func_limpa_corpus(all_google_vc)
all_google_tdm <- TermDocumentMatrix(all_google_clean2)

# Colnames
colnames(all_google_tdm) <- c("Google Pros", "Google Cons")

# Converte para matriz
all_google_tdm_m <- as.matrix(all_google_tdm)

# Nuvem de comparação
?comparison.cloud
comparison.cloud(all_google_tdm_m, colors = c("orange", "blue"), max.words = 50)

# As críticas positivas da Amazon parecem mencionar bigramas como "bons benefícios", enquanto suas
# críticas negativas se concentram em bigramas, como questões de "equilíbrio trabalho-vida".

# Em contraste, as análises positivas do Google mencionam "regalias", "pessoas inteligentes", "boa comida"
# e "cultura divertida", entre outras coisas. As avaliações negativas do Google discutem "política", "crescer",
# "burocracia" e "média gerência".

# Agora faremos um gráfico de pirâmide alinhando comentários positivos para a Amazon e o Google para que você
# possa ver adequadamente as diferenças entre quaisquer bigramas compartilhados.
amazon_pro <- paste(df_amazon$pros, collapse = "")
google_pro <- paste(df_google$pros, collapse = "")
all_pro <- c(amazon_pro, google_pro)
all_pro_clean <- func_limpa_texto(all_pro)
all_pro_vs <- VectorSource(all_pro)
all_pro_vc <- VCorpus(all_pro_vs)
all_pro_corp <- func_limpa_corpus(all_pro_vc)

# Matriz termo-documento
tdm.bigram = TermDocumentMatrix(all_pro_corp, control = list(tokenize = tokenizer))

# Colnames
colnames(tdm.bigram) <- c("Amazon", "Google")

# Converte para matriz
tdm.bigram <- as.matrix(tdm.bigram)

# Palavras comuns
common_words <- subset(tdm.bigram, tdm.bigram[,1] > 0 & tdm.bigram[,2] > 0 )

# Diferença

```

```

difference <- abs(common_words[, 1] - common_words[,2])

# Vetor final
common_words <- cbind(common_words, difference)
common_words <- common_words[order(common_words[,3],decreasing = TRUE),]

# Dataframe
top25_df <- data.frame(x = common_words[1:25,1],
                      y = common_words[1:25,2],
                      labels = rownames(common_words[1:25,]))

# Plot
pyramid.plot(top25_df$x,
              top25_df$y,
              labels=top25_df$labels,
              gap=15,
              top.labels=c("Amazon Pros", "Vs", "Google Pros"),
              unit = NULL,
              main = "Palavras em Comum")

# Os funcionários da Amazon mencionaram o "equilíbrio entre vida pessoal e profissional" como um aspecto positivo.
# Em ambas as organizações, as pessoas mencionaram "cultura" e "pessoas inteligentes", portanto, há alguns
# aspectos positivos semelhantes entre as duas empresas.

# Agora vamos voltar a atenção para as avaliações negativas e criar os mesmos recursos visuais.
amazon_cons <- paste(df_amazon$cons, collapse = "")
google_cons <- paste(df_google$cons, collapse = "")
all_cons <- c(amazon_cons,google_cons)
all_cons_clean <- func_limpa_texto(all_cons)
all_cons_vs <- VectorSource(all_cons)
all_cons_vc <- VCorpus(all_cons_vs)
all_cons_corp <- func_limpa_corpus(all_cons_vc)

# Matriz termo-documento
tdm.cons_bigram = TermDocumentMatrix(all_cons_corp,control=list(tokenize =tokenizer))

# Preparação dos dados conforme feito anteriormente
colnames(tdm.cons_bigram) <- c("Amazon", "Google")
tdm.cons_bigram <- as.matrix(tdm.cons_bigram)
common_words <- subset(tdm.cons_bigram, tdm.cons_bigram[,1] > 0 & tdm.cons_bigram[,2] > 0 )
difference <- abs(common_words[, 1] - common_words[,2])
common_words <- cbind(common_words, difference)
common_words <- common_words[order(common_words[,3], decreasing = TRUE),]

# Dataframe
top25_df
<- data.frame(x = common_words[1:25,1],
              y = common_words[1:25,2],
              labels = rownames(common_words[1:25,]))

# Plot
pyramid.plot(top25_df$x,
              top25_df$y,
              labels=top25_df$labels,
              gap=10,
              top.labels = c("Amazon Cons","Vs","Google Cons"),
              unit = NULL,
              main = "Palavras em Comum")

# Usaremos nuvem de comunalidade para mostrar o que é comum entre Amazon e Google
# com o tokenizer Unigram, Bigram e Trigram para identificar mais insights.

# Unigram
tdm.unigram <- TermDocumentMatrix(all_pro_corp)
colnames(tdm.unigram) <- c("Amazon","Google")
tdm.unigram <- as.matrix(tdm.unigram)

?commonality.cloud
commonality.cloud(tdm.unigram, colors = c("tomato2", "yellow2"), max.words = 100)

# Bigram
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
tdm.bigram <- TermDocumentMatrix(all_pro_corp,control = list(tokenize=BigramTokenizer))
colnames(tdm.bigram) <- c("Amazon", "Google")
tdm.bigram <- as.matrix(tdm.bigram)

commonality.cloud(tdm.bigram, colors = c("tomato2", "yellow2"), max.words = 100)

```

```
# Trigram
TrigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 3, max = 3))
tdm.trigram <- TermDocumentMatrix(all_pro_corp, control = list(tokenize=TrigramTokenizer))
colnames(tdm.trigram) <- c("Amazon", "Google")
tdm.trigram <- as.matrix(tdm.trigram)

commonality.cloud(tdm.trigram, colors = c("tomato2", "yellow2"), max.words = 100)

# Palavras mais frequentes nos comentários dos funcionários
amazon_tdm <- TermDocumentMatrix(amazon_p_corp)
associations <- findAssocs(amazon_tdm, "fast", 0.2)
associations_df <- list_vect2df(associations)[,2:3]

ggplot(associations_df, aes(y = associations_df[,1])) +
  geom_point(aes(x = associations_df[,2]),
    data = associations_df, size = 3) +
  theme_gdocs()

google_tdm <- TermDocumentMatrix(google_c_corp)
associations <- findAssocs(google_tdm, "fast", 0.2)
associations_df <- list_vect2df(associations)[,2:3]

ggplot(associations_df, aes(y=associations_df[,1])) +
  geom_point(aes(x = associations_df[,2]),
    data = associations_df, size = 3) +
  theme_gdocs()

# Conclusão

# O Google tem um melhor equilíbrio entre vida profissional e pessoal de acordo com as avaliações dos funcionários.

# Fim
```

7.13. Redes Neurais Artificiais

Redes Neurais

O cérebro humano tem sido extensamente estudado, mas ainda não somos capazes de entender completamente o seu funcionamento.

Redes Neurais Artificiais podem ser consideradas um paradigma diferente de computação.

Redes Neurais Artificiais consistem em um modo de abordar a solução de problemas de Inteligência Artificial.

Assim como um cérebro usa uma rede de células interconectadas chamadas neurônios para criar um processador paralelo maciço, a rede neural usa uma rede de neurônios artificiais para resolver problemas de aprendizagem.

- Cérebro humano – 85 bilhões de neurônios
- Cérebro de um gato – 1 bilhão de neurônios
- Cérebro de um rato – 75 milhões de neurônios
- Cérebro de uma barata – 1 milhão de neurônios

Agora você entende porque a computação paralela em GPU's está acelerando o desenvolvimento de sistemas inteligentes, pois somos capazes de processar cada vez mais dados em redes neurais artificiais com cada vez mais neurônios.

Exemplos de onde se utiliza redes neurais artificiais:

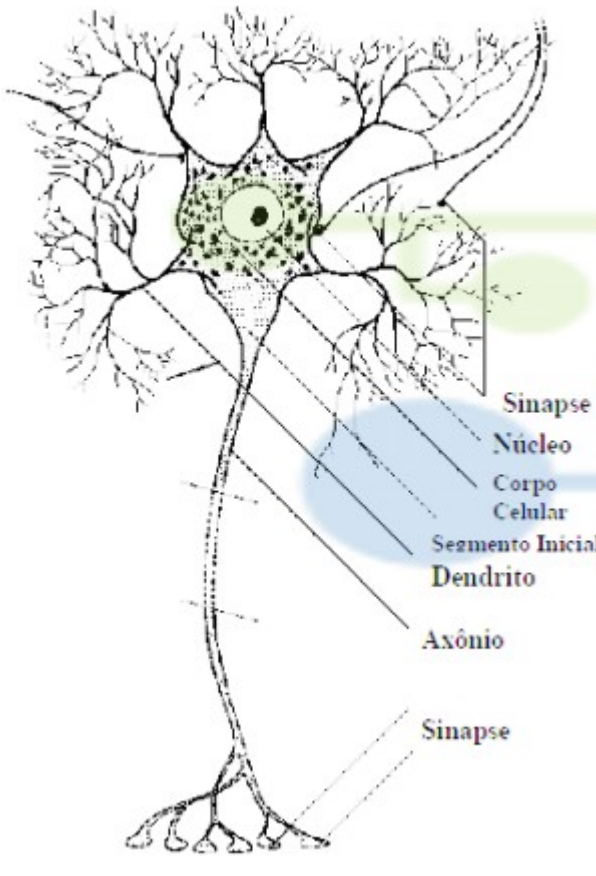
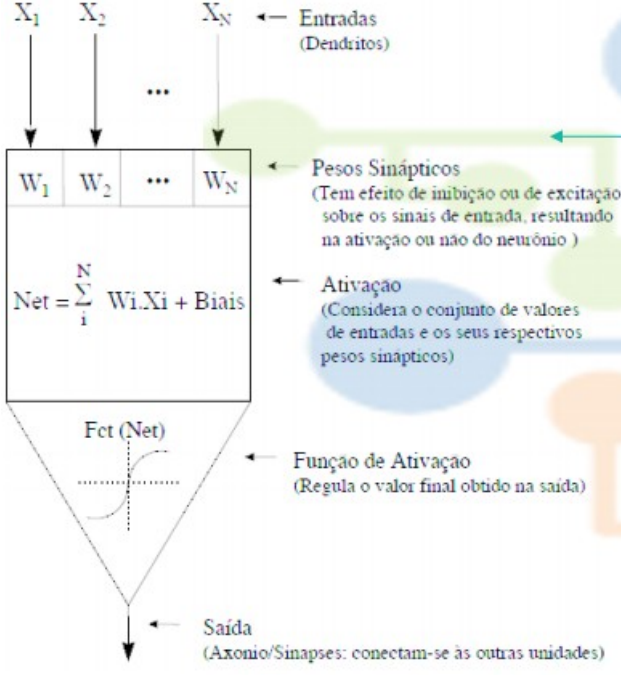
- Programas de reconhecimento de voz e escrita
- Automação de dispositivos inteligentes
- Modelos sofisticados de padrões climáticos

As Redes Neurais Artificiais são modelos versáteis que podem ser aplicadas a quase todas as tarefas de aprendizagem: classificação, previsão numérica e mesmo reconhecimento não supervisionado de padrões.

As redes neurais artificiais são melhor aplicadas a problemas onde os dados de entrada e os dados de saída são bem definidos ou, pelo menos, bastante simples, mas o processo que relaciona a entrada com a saída é extremamente complexo.

O conhecimento de uma Rede Neural Artificial (RNA) está codificado na estrutura da rede, onde se destacam as conexões (sinapses) entre as unidades (neurônios) que a compõe.

Neurônio Biológico	Neurônio Matemático
--------------------	---------------------

	 <p>Entradas (Dendritos)</p> <p>Pesos Sinápticos (Tem efeito de inibição ou de excitação sobre os sinais de entrada, resultando na ativação ou não do neurônio)</p> <p>Ativação (Considera o conjunto de valores de entradas e os seus respectivos pesos sinápticos)</p> <p>Função de Ativação (Regula o valor final obtido na saída)</p> <p>Saída (Axônio/Sinapses: conectam-se às outras unidades)</p>
<p>O ponto de contato entre a terminação axônica de um neurônio e o dendrito de outro é chamado sinapse.</p> <p>Se esses sinais forem superiores a aproximadamente 50mV (limiar do disparo), seguem pelo axônio. Caso contrário, são bloqueados e não prosseguem (são considerados irrelevantes).</p> <p>Se o sinal for superior a certo limite (threshold), vai em frente; caso contrário é bloqueado e não segue.</p> <p>Um neurônio recebe sinais através de inúmeros dendritos, os quais são ponderados e enviados para o axônio, podendo ou não seguir adiante (threshold)</p> <p>Cada condutor, está associado um peso pelo qual o sinal é multiplicado. A memória são os pesos.</p> <p>Cada região do cérebro possui uma arquitetura de rede diferente: varia o número de neurônios, de sinapses por neurônio, valor dos thresholds e dos pesos, etc...</p>	<p>Um neurônio dispara quando a soma dos impulsos que ele recebe ultrapassa o seu limiar de excitação chamado de threshold.</p> <p>Note que este modelo matemático simplificado de um neurônio é estático, ou seja, não considera a dinâmica do neurônio natural. No neurônio biológico, os sinais são enviados em pulsos e alguns componentes dos neurônios biológicos, a exemplo do axônio, funcionam como filtros de frequência.</p> <p>Dentre as funções de ativação utilizadas, podemos destacar:</p> <ul style="list-style-type: none"> Função Sigmóide Função Gaussiana Função Tangente Hiperbólica <p>O modelo neuronal matemático também pode incluir uma polarização ou bias de entrada. Esta variável é incluída ao somatório da função de ativação, com o intuito de aumentar o grau de liberdade desta função e, consequentemente, a capacidade de aproximação da rede.</p>

Os valores dos pesos são estabelecidos por meio de treinamento recebido pelo cérebro durante a vida útil. É a memorização.	
--	--

Um dos benefícios das redes diz respeito ao tratamento de um problema clássico da Inteligência Artificial, que é a representação de um universo não-estacionário (onde as estatísticas mudam com o tempo).

Uma desvantagem das redes neurais é o fato delas , normalmente, serem uma "caixa preta".

A solução de problemas através das RNAs é bastante atrativa, pois o paralelismo constitui-se na característica principal das RNAs, onde esta cria a possibilidade de um desempenho superior em relação a solução de problemas baseados nos modelos convencionais.

A generalização está associada à capacidade da rede em aprender através de um conjunto reduzido de exemplos, e posteriormente, dar respostas coerentes a dados não apresentados a rede.

Para compreender a lógica de funcionamento das redes neurais, alguns conceitos básicos referentes ao funcionamento do cérebro humano e seus componentes, os neurônios, são de fundamental importância.

A Arquitetura de Redes Neurais Artificiais

Vértices são os neurônios e as arestas as sinapses.

- Feed Forward Networks (Redes Diretas)
 - Os neurônios são arranjados em camadas, sendo que a camada inicial recebe os sinais de entrada enquanto a camada final obtém as saídas. As camadas intermediárias são chamadas de camadas ocultas.
 - Cada neurônio de uma camada é conectado com todos os neurônios da camada seguinte.
 - Não há conexões entre neurônios de uma mesma camada.
- Feed Backward Networks (Redes Recorrentes)
- Redes Competitivas

Processo de Aprendizagem

Processo de Aprendizagem de Redes Neurais:

- Aprendizagem Supervisionada
- Aprendizagem Não Supervisionada

Tipo de Regra de Aprendizagem	Descrição
Aprendizagem por Correção de Erro (Regra Delta)	Utilizado em treinamento supervisionado, esta técnica ajusta os pesos sinápticos por meio do erro, que é obtido através da diferença entre o valor de saída da rede e o valor esperado em um ciclo de treinamento. Com isso gradualmente vai diminuindo o erro geral da rede.

Aprendizagem Hebbiana	Baseado no postulado de aprendizagem de Hebb, que afirma: “se dois neurônios em ambos os lados de uma sinapse são ativados sincronamente e simultaneamente, então a força daquela sinapse é seletivamente aumentada”. Este processo de treinamento é feito localmente, ajustando o peso das conexões baseado nas atividades dos neurônios.
Aprendizagem de Boltzmann	Método de aprendizagem estocástico derivado de conceitos da Estatística. Neste modelo os neurônios são estocásticos, podendo residir em dois estados possíveis, ligado (+1) e desligado (-1), e ainda são divididos em dois grupos funcionais, presos e livres, sendo responsáveis pela interação com o ambiente e pela explicação das restrições subjacentes dos padrões de entrada do ambiente, respectivamente. Um ponto importante deste modelo de aprendizagem é que os neurônios possuem conexões bidirecionais.
Aprendizagem Competitiva	Neste modelo de aprendizagem os neurônios são forçados a competir entre si e somente um será ativo, em uma dada iteração, o vencedor, ou seja, o que tiver maior similaridade com o padrão de entrada. Todos os pesos dos neurônios próximos ao neurônio vencedor terão seus valores ajustados.

Aprendizagem Hebbiana

Quando um axônio da célula A está perto o suficiente para excitar uma célula B e participa do seu disparo repetidamente, então algum processo de crescimento ou modificação metabólica acontece em uma das células ou em ambas, de tal forma que a eficiência de A como uma das células que dispara B é aumentada.

Se dois neurônios em ambos os lados de uma sinapse são ativados sincronamente, então a força desta sinapse é aumentada. Se dois neurônios em ambos os lados de uma sinapse são ativados assincronamente, então a força desta sinapse é enfraquecida.

Perceptron

O Perceptron e suas limitações inspiraram os modelos mais avançados de redes neurais.

Um perceptron funciona analogamente a um neurônio.

Os perceptrons são capazes de aprender online, com o erro.

Com o Perceptron, o objetivo é aprender pesos sinápticos de tal forma que a unidade de saída produza a saída correta para cada exemplo. O algoritmo faz atualizações iterativamente até chegar aos pesos corretos.

Limitações do Perceptron

Um único Perceptron consegue resolver somente funções linearmente separáveis. Em funções não linearmente separáveis o perceptron não consegue gerar um hiperplano para separar os dados.

Adaline (Adaptive Linear Element)

O Adaline é similar ao Perceptron, com diferença apenas pelo seu algoritmo de treinamento. Enquanto o Perceptron ajusta os pesos somente quando um padrão é classificado incorretamente, o Adaline utiliza a Regra Delta para minimizar o erro médio (MSE) após cada padrão ser apresentado, ajustando os pesos proporcionalmente ao erro.

Regra Delta

Inicialmente, atribui-se aos pesos valores aleatórios e, com eles, apresenta-se um conjunto de sinais de entrada e calcula-se a resposta da rede. Então, comparam-se os valores calculados com os valores desejados (treinamento supervisionado).

Esse algoritmo, conhecido como regra delta, deu origem, anos mais tarde, ao primeiro algoritmo de treinamento de redes Perceptron de múltiplas camadas, o Backpropagation.

Perceptron → Hiperplanos factíveis (vários)

Adaline → Hiperplano ótimo (único)

Perceptrons de Múltiplas Camadas

Redes Multilayer Perceptron são redes diretas (feed forward) que possuem uma ou mais camadas de neurônios entre as camadas de entrada e saída, chamada de camada oculta. Esta camada adiciona um poder maior em relação às redes Perceptron de camada única, que classifica apenas padrões linearmente separáveis, sendo os neurônios ocultos responsáveis por capturar a não-linearidade dos dados.

Algoritmo de treinamento backpropagation:

```
1 begin
2   Atribuição de valores iniciais aos pesos sinápticos;
3   repeat
4     Apresentação à rede dos padrões de entrada e as saídas desejadas;
5     Cálculo dos valores de saída dos neurônios ocultos;
6     Cálculo dos valores de saída dos neurônios de saída (resposta real da rede);
7     Cálculo do erro (diferença entre resposta da rede e valor esperado);
8     Ajuste dos pesos sinápticos;
9   until Condição de parada não satisfeita;
10 end
```

Alguns parâmetros são determinados por tentativa e erro, ou seja, são atribuídos vários valores distintos aos parâmetros e analisando os resultados obtidos, a melhor configuração é escolhida!

Outra dificuldade é a determinação do número ideal de ciclos de treinamento da rede, que é determinado por tentativa e erro.

O overfitting é identificado quando o erro de teste, obtido pela validação cruzada, começa a aumentar depois de ter diminuído.

O Algoritmo Backpropagation

O Backpropagation é multicamada, pois tem no mínimo 3 camadas (entrada, intermediária, saída).

Os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retropropagado.

As redes que utilizam backpropagation trabalham com uma variação da regra delta, apropriada para redes multi-camadas: a regra delta generalizada.

O treinamento das redes MLP com backpropagation pode demandar muitos passos no conjunto de treinamento, resultando um tempo de treinamento consideravelmente longo.

Introdução às Redes Neurais Artificiais

O cérebro humano é uma máquina altamente poderosa e complexa capaz de processar uma grande quantidade de informações em tempo mínimo. As unidades principais do cérebro são os neurônios, e é por meio deles que as informações são transmitidas e processadas. As tarefas realizadas pelo cérebro, intrigam os pesquisadores, como por exemplo, a capacidade do cérebro de reconhecer um rosto familiar dentre uma multidão em apenas milésimos de segundo. As respostas sobre alguns enigmas do funcionamento do cérebro se perpetuam até os dias de hoje. O que é conhecido sobre o funcionamento do cérebro é que o mesmo desenvolve suas regras através da experiência adquirida em situações vividas anteriormente.

O desenvolvimento do cérebro humano ocorre principalmente nos dois primeiros anos de vida, mas se arrasta por toda a vida. Inspirando-se neste modelo, diversos pesquisadores tentaram simular o funcionamento do cérebro, principalmente o processo de aprendizagem por experiência, a fim de criar sistemas inteligentes capazes de realizar tarefas como classificação, reconhecimento de padrões, processamento de imagens, entre outras. Como resultados destas pesquisas surgiram o modelo do neurônio artificial e posteriormente um sistema com vários neurônios interconectados, a chamada Rede Neural Artificial.

É o que vamos estudar neste capítulo!

Redes Neurais Artificiais é um tema extenso e poderia ser um curso inteiro. De fato é. Na verdade 3. Aqui na DSA temos 3 cursos inteiros dedicados às Redes Neurais Artificiais:

- Deep Learning Frameworks
- Deep Learning I
- Deep Learning II

Nosso objetivo neste capítulo é trazer uma introdução completa ao tema, com exemplos completos e conhecimento suficiente para que você comece a construir seus próprios modelos de Redes Neurais Artificiais.

Recomendamos como bibliografia complementar o Deep Learning Book:

www.deeplearningbook.com.br

O Dispositivo Mais Incrível da História Humana



Este é o dispositivo mais incrível da história humana e que os cientistas estão tentando reproduzir em computadores!

As redes neurais, ou redes neurais artificiais para sermos mais precisos, representam uma tecnologia que tem raízes em muitas disciplinas: neurociência, matemática, estatística, física, ciência da computação e engenharia. As redes neurais encontram aplicações em campos tão diversos, como modelagem, análise de séries temporais, reconhecimento de padrões, processamento de sinais e muito mais. E tantas aplicações ocorrem em virtude de uma importante propriedade: a habilidade de aprender a partir de dados de entrada, com ou sem supervisão. Este capítulo é uma introdução ao fascinante mundo da Inteligência Artificial, através das redes neurais.

O cérebro é muito complexo, até mesmo o comportamento de um simples neurônio é extremamente complexo. No cérebro, o comportamento inteligente é uma propriedade que emerge de um grande número de unidades simples (ao contrário do que acontece com regras e algoritmos simbólicos). Neurônios ligam e desligam em alguns milissegundos, enquanto o hardware atual pode fazer o mesmo em nano segundos. Entretanto, o cérebro realiza tarefas cognitivas complexas (visão, reconhecimento de voz) em décimos de segundo. O cérebro utiliza um paralelismo massivo. Redes Neurais Artificiais podem ser consideradas um paradigma diferente de computação.

Redes neurais são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neuronal de organismos inteligentes e que adquirem conhecimento através da experiência. Baseado no funcionamento do cérebro humano, ou no procedimento como o cérebro aprende, as redes neurais artificiais são capazes de simular conexões sinápticas. Cada neurônio do cérebro humano tem a capacidade de criar até 10 mil sinapses com outros neurônios.

Pode-se dizer que redes neurais artificiais consistem em um modo de abordar a solução de problemas de inteligência artificial. Neste caso, em lugar de tentar programar um computador de modo a fazê-lo imitar um comportamento inteligente (saber jogar xadrez, compreender e manter um diálogo, traduzir línguas estrangeiras, resolver problemas de matemática, etc.) procura-se construir um computador que tenha circuitos, modelando os circuitos cerebrais e espera-se ver um comportamento inteligente emergindo, aprendendo novas tarefas, errando, fazendo generalizações e descobertas.

Da mesma forma, estes circuitos neurais artificiais poderão se auto organizar, quando apresentados a ambientes diversos, criando suas próprias representações internas e apresentar comportamentos imprevisíveis. E, melhor ainda, (ou pior talvez) ter um comportamento que nem sempre pode-se prever e compreender, tal como hoje não compreendemos mecanismos do nosso próprio cérebro.

Uma Rede Neural Artificial modela a relação entre um conjunto de sinais de entrada e um sinal de saída usando um modelo derivado de nossa compreensão de como um cérebro biológico responde a

estímulos de entradas sensoriais. Assim como um cérebro usa uma rede de células interconectadas chamadas neurônios para criar um processador paralelo maciço, a rede neural usa uma rede de neurônios artificiais para resolver problemas de aprendizagem.

O cérebro humano é o “dispositivo” mais incrível da história humana.

Recomendamos como bibliografia complementar o Deep Learning Book:

www.deeplearningbook.com.br

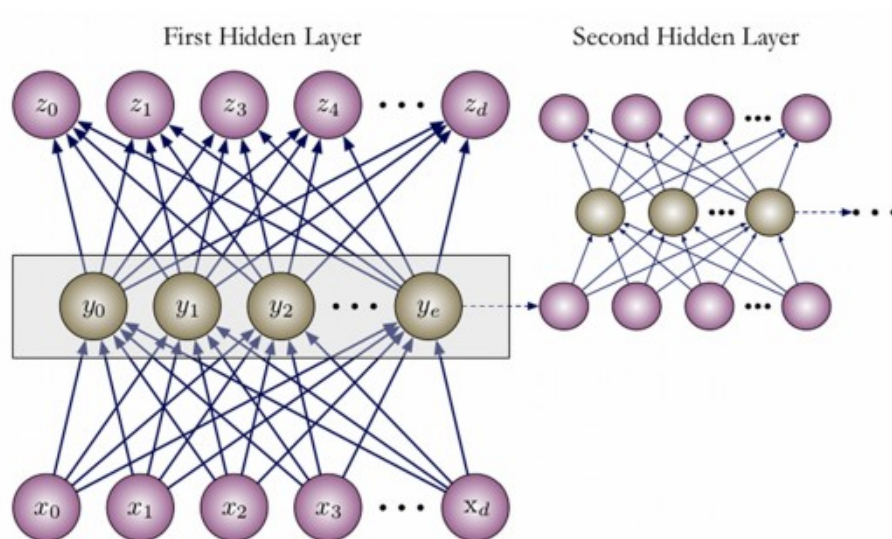
A Origem das Redes Neurais Artificiais

O trabalho em redes neurais artificiais, usualmente denominadas apenas “redes neurais”, tem sido motivado desde o começo pelo reconhecimento de que o cérebro humano processa informações de uma forma inteiramente diferente do computador convencional. O cérebro humano é em essência um computador altamente complexo, não-linear e paralelo. Ele tem a capacidade de organizar seus componentes estruturais, conhecidos como neurônios, de forma a realizar certos processamentos (como reconhecimento de padrões, percepção e controle motor), muito mais rapidamente do que o mais rápido computador digital hoje existente. A verdade é que não exploramos nem 5% da capacidade de nossos cérebros. Ainda assim, os cientistas tem tentado reproduzir o processo de aprendizagem do cérebro humano, através de redes neurais artificiais.

As redes neurais artificiais são, provavelmente, a mais antiga técnica de Inteligência Artificial em uso. Este instrumento foi desenvolvido na década de 40 por Warren mac culoch e Walter Pitts, o primeiro neurofisiologista e o segundo matemático. A ideia era fazer uma analogia entre neurônios biológicos e circuitos eletrônicos, capazes de simular conexões sinápticas pelo uso de resistores variáveis e amplificadores. Mas foi no final da década de 80, o ressurgimento da área de Redes Neurais Artificiais, também conhecida como connexionismo ou sistemas de processamento paralelo e distribuído. Esta forma de computação não é algorítmica e sim caracterizada por sistemas que, de alguma forma ou de alguma maneira, relembram a estrutura do cérebro humano.

O começo do estudo das rede neurais artificiais pode ser atribuído à criação do Psychon em 1943 por Warren McCulloch e Walter Pitts, sendo que alguns anos mais tarde, em 1949 o pesquisador D. O. Hebb publicava uma importante obra, o livro “The Organization of Behaviour” que influenciou vários modelos de RNAs de destaque na atualidade.

A representação de conhecimentos nas redes connexionistas, como diz o próprio nome, é fortemente ligada a noção de conexão entre neurônios (elementos processadores de informação) que interagem uns com os outros através destas ligações. O modelo connexionista possui sua origem nos estudos feitos sobre as estruturas de nosso cérebro – sofrendo uma grande simplificação do modelo original – onde encontramos no modelo artificial, que é simulado, elementos como os neurônios e as suas conexões, chamadas de sinapses.



Os primeiros estudos e propostas de modelos sobre as Redes Neurais Artificiais surgiram nos anos 40. Os primeiros modelos evoluíram bastante, onde alguns deles se destacaram e tornaram-se famosos, mas mesmo assim até hoje continuam sendo propostos novos modelos de redes neurais. O estudo dos primeiros modelos e de sua evolução, nos ajuda a entender melhor as redes neurais, e seu estágio atual de evolução.

A Evolução das Redes Neurais Artificiais

O começo do estudo das rede neurais artificiais pode ser atribuído à criação do Psychon em 1943 por Warren McCulloch e Walter Pitts, sendo que alguns anos mais tarde, em 1949 o pesquisador D. O. Hebb publicava uma importante obra, o livro “The Organization of Behaviour” que influenciou vários modelos de RNAs de destaque na atualidade.

Em 1959, Frank Rosenblatt criou o Perceptron que, como será visto mais a frente neste capítulo, tem até hoje uma grande influência sobre os estudos das redes neurais, mostrando que apesar desta área de estudos ter crescido muito na atualidade, suas bases foram estruturadas juntamente com a criação dos fundamentos da ciência da computação. Alguns outros modelos similares ao Perceptron foram também desenvolvidos nesta época, como é o caso do Adaline (Adaptive Linear Element), criado por Bernard Widrow em 1962. Os modelos do tipo Perceptron, incluindo o Adaline, são baseados no aprendizado supervisionado por correção de erros, uma classe muito importante de redes neurais artificiais, que possui uma larga aplicação na atualidade.

Em 1969 os modelos baseados no Perceptron receberam uma dura crítica feita por Minsky e Papert através de sua obra “Perceptrons: An Introduction to Computational Geometry”. Através deste livro, Minsky e Papert provaram matematicamente que os modelos de redes neurais baseados no Perceptron (redes de um só nível, o que na época era o tipo de rede de Perceptrons utilizado), não eram capazes de aprender uma simples função lógica do tipo “ou-exclusivo” (XOR = Exclusive Or). A função XOR possui um padrão de valores de entrada e de saída cuja associação não podia ser aprendida pelos modelos de redes baseados em Perceptron disponíveis naquela época. O impacto da publicação deste livro abalou profundamente as pesquisas realizadas nesta área de estudos.

O Madaline (Many Adaline), também criado por Widrow, podia de certa forma resolver o problema, mas o aprendizado não podia ser realizado de uma forma muito “natural” e automatizada, pois

requeria a intervenção humana na construção da rede. Devido as críticas feitas e a falta de uma solução para os problemas apresentados, as redes neurais ficaram “esquecidas” por um certo tempo período conhecido como o Inverno da IA.

Somente na década de 80, surgiram novos modelos que deram um novo impulso as redes neurais. Em 1982 surgiu um modelo importante de rede criado por J. Hopfield. O modelo que Hopfield criou era baseado em um tipo de rede diferente dos modelos baseados no Perceptron, sendo uma rede com conexões recorrentes e com um comportamento baseado na competição entre os neurônios, onde o aprendizado era não supervisionado. Outros modelos similares ao modelo de Hopfield surgiram pouco depois, onde podemos citar alguns como por exemplo: a máquina de Boltzmann e o BAM (Binary Associative Memory).

A década de 80 ficou também marcada profundamente pelo reaparecimento das redes baseadas em Perceptrons. Isto deveu-se ao desenvolvimento dos computadores, que eram mais velozes e permitiam realizar melhores simulações das redes neurais, bem como o desenvolvimento de modelos matemáticos que permitiram a solução do problema apontado por Minsky e Papert. Também podemos associar em parte este renascimento das redes neurais ao suposto desencanto com a I.A. clássica.

O modelo que permitiu o ressurgimento das redes baseadas em Perceptrons foi o das redes multi-nível, onde o novo algoritmo de aprendizado chamado Back-Propagation resolveu em grande parte os problemas de aprendizado existentes até então. Este modelo foi desenvolvido por diferentes pesquisadores quase ao mesmo tempo, como D. Parker e D. Rumelhart, mas foi Rumelhart e Hinton que tornaram este algoritmo famoso com a sua obra “Parallel Distributed Processing - PDP”. Este algoritmo, o Back-Propagation permitia realizar o aprendizado por correção de erros em uma rede com múltiplas camadas (níveis) e consequentemente resolveria o problema do XOR. Geoffrey Hinton é professor da Universidade de Toronto no Canadá e possui um dos mais famosos cursos online sobre Redes Neurais, cujo link você encontra ao final do capítulo.

Além dos modelos de Hopfield e do modelo de redes multi-nível com Back-Propagation (chamado de Multi-Layer Perceptron – MLP), outro modelo importante que surgiu nesta década foi o modelo de Teuvo Kohonen. O modelo de Kohonen é muito interessante pois permite o aprendizado competitivo com uma auto-organização da rede neural, criando os chamados “mapas de atributos auto-organizáveis”, tema estudado no curso Deep Learning II.

Por fim, o último modelo de destaque nesse período, foi o modelo ART (Adaptive Resonance Therapy) criado por Gail Carpenter e Stephen Grossberg. Esse modelo possui um aprendizado do tipo não supervisionado, criando protótipos (clusters) dos padrões aprendidos. O modelo ART teve diversas versões posteriores, entre elas versões do tipo semisupervisionado e com uso de conceitos da lógica nebulosa (Fuzzy-ART).

Nessa mesma época, era lançado o algoritmo SVM, Support Vector Machines, que estudamos aqui do curso de Machine Learning da DSA.

Os estudos sobre as redes neurais sofreram uma grande revolução a partir dos anos 80 e esta área de estudos tem se destacado, seja pelas promissoras características apresentadas pelos modelos de redes neurais propostos, seja pelas condições tecnológicas atuais de implementação que permitem desenvolver arrojadas implementações de arquiteturas neurais paralelas em hardwares dedicados, obtendo assim ótimas performances destes sistemas (bastante superiores aos sistemas

convencionais). A evolução natural das redes neurais, são as redes neurais profundas (ou Deep Learning), assunto do próximo capítulo aqui do nosso curso!

De fato estamos neste momento escrevendo parte da história das redes neurais artificiais, criando assim um novo capítulo na história da Inteligência Artificial. Não é fascinante poder participar dessa história?

Parâmetros x Hiperparâmetros

Alguns termos podem parecer confusos quando você começa a aprender Machine Learning e Inteligência Artificial. Há tantos termos a serem usados e muitos dos termos são intercambiáveis. Isto é especialmente verdadeiro se você veio de outro campo de estudo que pode usar alguns dos mesmos termos utilizados em aprendizagem de máquina, mas de forma diferente. E um bom exemplo disso são os termos “parâmetro” e “hiperparâmetro”.

Não ter uma definição clara para estes termos é uma dúvida comum para iniciantes. Vamos então compreender as diferenças entre esses 2 termos.

O que é um parâmetro do modelo?

Um parâmetro do modelo é uma variável de que é interna ao modelo e cujo valor pode ser estimado a partir de dados. O parâmetro também costuma ser chamado de peso ou coeficiente. Suas principais características são:

- Eles são requeridos pelo modelo para fazer previsões.
- Os valores dos parâmetros definem a habilidade do modelo em resolver seu problema.
- Eles são estimados ou aprendidos a partir de dados.
- Muitas vezes, eles não precisam ser configurados manualmente.
- Eles geralmente são salvos como parte do modelo aprendido.

Os parâmetros são fundamentais para algoritmos de aprendizagem de máquina. Eles são parte do modelo que é construído com os dados históricos de treinamento.

Na literatura de aprendizagem de máquina, podemos pensar no modelo como a hipótese e os parâmetros como a adaptação da hipótese a um conjunto específico de dados. Muitas vezes, os parâmetros do modelo são estimados usando um algoritmo de otimização, que é um tipo de pesquisa eficiente através de possíveis valores de parâmetros. Em Estatística e Programação, o termo parâmetro pode ter definições ligeiramente diferentes:

Estatística: nas estatísticas, você pode assumir uma distribuição para uma variável, como uma distribuição Gaussiana. Dois parâmetros da distribuição Gaussiana são a média e o desvio padrão. Isso é válido no aprendizado de máquina, onde esses parâmetros podem ser estimados a partir de dados e usados como parte de um modelo preditivo.

Programação: na programação, você pode passar um parâmetro para uma função. Neste caso, um parâmetro é um argumento de função que poderia assumir um valor de um intervalo de valores. Na aprendizagem de máquina, o modelo específico que você está usando é a função e requer parâmetros para fazer uma previsão sobre novos dados.

Alguns exemplos de parâmetros do modelo incluem:

- Os pesos em uma rede neural artificial.
- Os vetores de suporte em uma máquina vetorial de suporte.
- Os coeficientes em uma regressão linear ou regressão logística.

O que é um hiperparâmetro de um modelo?

Um hiperparâmetro de um modelo é uma configuração externa ao modelo e cujo valor não pode ser estimado a partir de dados. Suas principais características são:

- Eles são frequentemente usados em processos para ajudar a estimar os parâmetros do modelo.
- Eles são frequentemente especificados pelo Cientista de Dados.
- Eles geralmente podem ser configurados usando heurísticas.
- Eles são frequentemente ajustados para um determinado problema de modelagem preditiva.

Não temos como conhecer o melhor valor para um hiperparâmetro em um determinado problema. Podemos usar regras básicas, copiar valores usados em outros problemas ou procurar o melhor valor por tentativa e erro.

Quando um algoritmo de aprendizagem de máquina é ajustado para um problema específico, usando por exemplo grid search ou pesquisa aleatória, você está ajustando os hiperparâmetros do modelo para descobrir os parâmetros do modelo que resultam nas previsões mais precisas.

Os hiperparâmetros do modelo são frequentemente referidos como parâmetros do modelo o que acaba gerando confusão. Uma boa regra geral para superar essa confusão é a seguinte:

Se você precisa especificar um parâmetro do modelo manualmente, então provavelmente, este é um hiperparâmetro.

Alguns exemplos de hiperparâmetros modelo incluem:

- A taxa de aprendizado para treinar uma rede neural.
- Os hiperparâmetros C e sigma para máquinas de vetor de suporte (SVMs).
- k em k-vizinhos mais próximos.

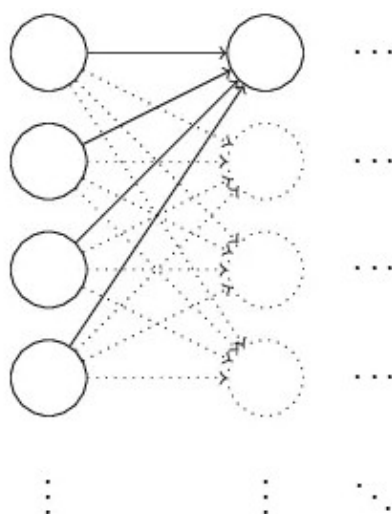
Por Que Inicializamos os Pesos de Um Modelo de Rede Neural?

Quando criamos nossas redes neurais, temos que fazer escolhas para os valores iniciais de pesos e vieses (bias). Até agora, nós os escolhemos de acordo com uma prescrição que discutimos nos capítulos anteriores. Só para lembrar, a prescrição era escolher tanto os pesos quanto os vieses usando variáveis aleatórias Gaussianas independentes, normalizadas para ter a média 0 e desvio padrão 1 (esse é um conceito fundamental em Estatística e caso queira adquirir conhecimento em Estatística, confira nossa mais nova Formação: Formação Análise Estatística Para Cientistas de Dados).

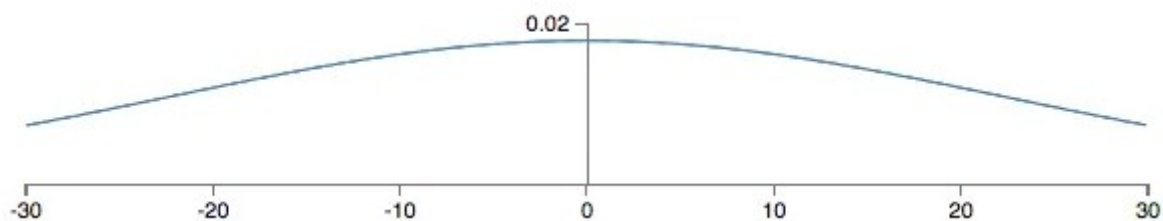
Embora esta abordagem tenha funcionado bem, foi bastante ad-hoc, e vale a pena revisitar para ver se podemos encontrar uma maneira melhor de definir nossos pesos e vieses iniciais, e talvez ajudar nossas redes neurais a aprender mais rápido. É o que iremos estudar neste capítulo.

Para começar, vamos compreender porque podemos fazer um pouco melhor do que inicializar pesos e vieses com valores Gaussianos normalizados. Para ver porque, suponha que estamos trabalhando com uma rede com um grande número – digamos 1.000 – de neurônios de entrada. E vamos supor que usamos valores Gaussianos normalizados para inicializar os pesos conectados à primeira

camada oculta. Por enquanto, vou me concentrar especificamente nos pesos que conectam os neurônios de entrada ao primeiro neurônio na camada oculta e ignorar o restante da rede:



Vamos supor, por simplicidade, que estamos tentando treinar usando uma entrada de treinamento x na qual metade dos neurônios de entrada estão ativados, isto é, configurados para 1, e metade dos neurônios de entrada estão desligados, ou seja, ajustados para 0. O argumento a seguir aplica-se de forma mais geral, mas você obterá a essência deste caso especial. Vamos considerar a soma ponderada $z = \sum j w_j x_j + b$ de entradas para nosso neurônio oculto. Ocorre que 500 termos nesta soma desaparecem, porque a entrada correspondente x_j é zero e, assim, z é uma soma sobre um total de 501 variáveis aleatórias Gaussianas normalizadas, representando os 500 termos de peso e o termo extra de viés (bias). Logo, z é ele próprio uma distribuição Gaussiana com média zero e desvio padrão ≈ 22.4 (raiz quadrada de 501). Ou seja, z tem uma distribuição Gaussiana muito ampla, sem um pico agudo, conforme a figura abaixo:



Em particular, podemos ver neste gráfico que é bem provável que $|z|$ será bastante grande, isto é, $z > 1$ ou $z < -1$. Se for esse o caso, a saída $\sigma(z)$ do neurônio oculto estará muito próxima de 1 ou 0. Isso significa que nosso neurônio oculto terá saturado. E quando isso acontece, como sabemos, fazer pequenas mudanças nos pesos fará apenas mudanças absolutamente minúsculas na ativação de nosso neurônio oculto. Essa mudança minúscula na ativação do neurônio oculto, por sua vez, dificilmente afetará o resto dos neurônios na rede, e veremos uma mudança minúscula correspondente na função de custo. Como resultado, esses pesos só aprenderão muito lentamente quando usarmos o algoritmo de descida do gradiente. É semelhante ao problema que discutimos anteriormente em outros capítulos, no qual os neurônios de saída que saturaram o valor errado fizeram com que o aprendizado diminuísse. Abordamos esse problema anterior com uma escolha

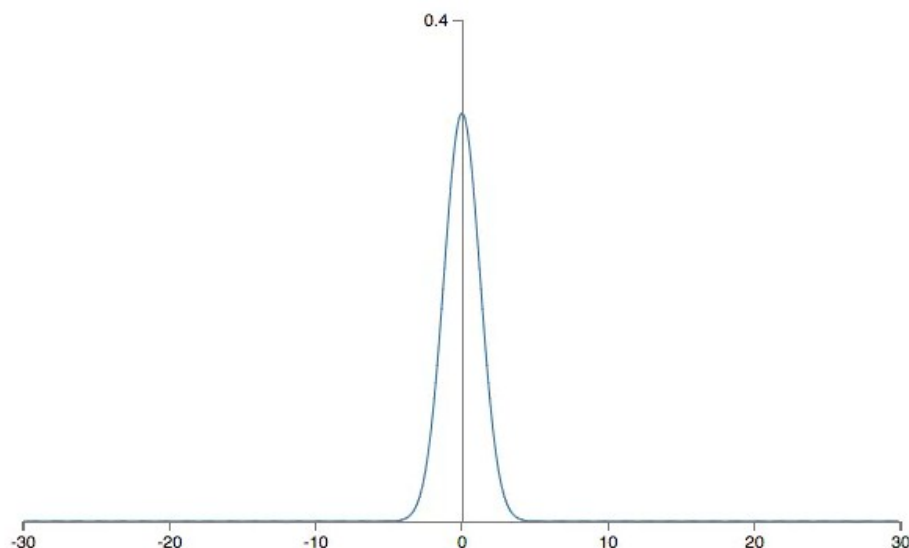
inteligente de função de custo. Infelizmente, enquanto isso ajudou com os neurônios de saída saturados, ele não faz nada pelo problema dos neurônios ocultos saturados.

Temos falado sobre a entrada de pesos para a primeira camada oculta. Naturalmente, argumentos semelhantes aplicam-se também a camadas ocultas posteriores: se os pesos em camadas ocultas posteriores forem inicializados usando Gaussianos normalizados, então as ativações estarão frequentemente muito próximas de 0 ou 1, e o aprendizado prosseguirá muito lentamente.

Existe alguma maneira de escolhermos melhores inicializações para os pesos e vieses, para que não tenhamos esse tipo de saturação e, assim, evitar uma desaceleração na aprendizagem? Suponha que tenhamos um neurônio com pesos de entrada n_{in} . Então, inicializaremos esses pesos como variáveis aleatórias gaussianas com média 0 e desvio padrão:

$$1/\sqrt{n_{in}}.$$

Isto é, vamos “esmagar os gaussianos”, tornando menos provável que nosso neurônio seja saturado. Continuaremos a escolher o viés como um Gaussiano com média 0 e desvio padrão 1, por motivos pelos quais voltaremos daqui a pouco. Com essas escolhas, a soma ponderada $z = \sum j w_j x_j + b$ será novamente uma variável aleatória Gaussiana com média 0, mas será muito mais aguda que antes. Suponha, como fizemos anteriormente, que 500 das entradas são zero e 500 são 1. Então é fácil mostrar (veja o gráfico abaixo) que z tem uma distribuição Gaussiana com média 0 e desvio padrão igual a $1,22\dots$ (raiz quadrada de $3/2$). Isso é muito mais agudo do que antes, tanto que até o gráfico abaixo subestima a situação, já que precisamos redimensionar o eixo vertical, quando comparado ao gráfico anterior:



É muito menos provável que tal neurônio sature e, correspondentemente, é muito menos provável que tenha problemas com a lentidão do aprendizado.

Eu afirmei acima que nós continuaremos a inicializar os vieses como antes, como variáveis aleatórias Gaussianas com uma média de 0 e um desvio padrão de 1. Isto não tem problema, pois é pouco provável que nossos neurônios vão saturar. Na verdade, não importa muito como inicializamos os vieses, desde que evitemos o problema com a saturação dos neurônios. Algumas

pessoas vão tão longe a ponto de inicializar todos os vieses com 0, e dependem da descida de gradiente para aprender vieses apropriados. Mas como é improvável que faça muita diferença, continuaremos com o mesmo procedimento de inicialização de antes.

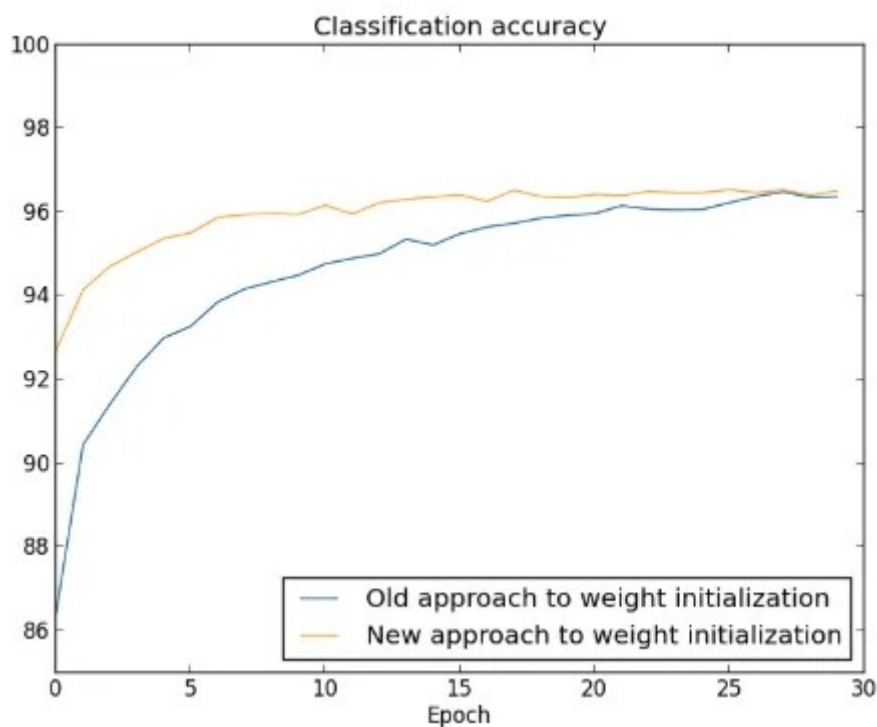
Vamos comparar os resultados para as nossas abordagens antiga e nova para inicialização de peso, usando a tarefa de classificação de dígitos MNIST. Como antes, usaremos 30 neurônios ocultos, um tamanho de mini-lote de 10, um parâmetro de regularização $\lambda = 5.0$ e a função de custo de entropia cruzada. Diminuiremos ligeiramente a taxa de aprendizado de $\eta = 0,5$ para 0,1, pois isso torna os resultados um pouco mais visíveis nos gráficos. Podemos treinar usando o antigo método de inicialização de peso (o código pode ser encontrado no repositório deste livro no Github):

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.1, lambda = 5.0,
... evaluation_data=validation_data,
... monitor_evaluation_accuracy=True)
```

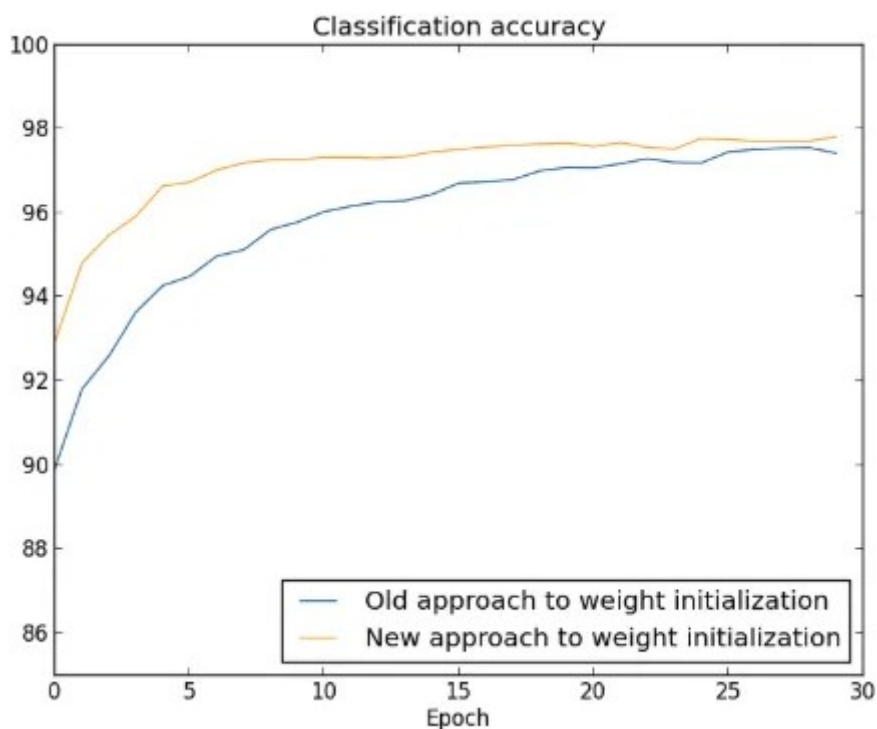
Também podemos treinar usando a nova abordagem para inicializar o peso. Na verdade, isso é ainda mais fácil, já que a maneira padrão de inicializar os pesos da rede2 é usar essa nova abordagem. Isso significa que podemos omitir a chamada `net.large_weight_initializer()` acima:

```
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.SGD(training_data, 30, 10, 0.1, lambda = 5.0,
... evaluation_data=validation_data,
... monitor_evaluation_accuracy=True)
```

Plotando os resultados, obtemos:



Em ambos os casos, acabamos com uma precisão de classificação um pouco acima de 96%. A precisão final da classificação é quase exatamente a mesma nos dois casos, mas a nova técnica de inicialização é muito, muito mais rápida. No final da primeira época de treinamento, a antiga abordagem de inicialização de peso tem uma precisão de classificação abaixo de 87%, enquanto a nova abordagem já chega a quase 93%. O que parece estar acontecendo é que nossa nova abordagem para a inicialização do peso nos leva a um processo muito melhor, o que nos permite obter bons resultados muito mais rapidamente. O mesmo fenômeno também é visto se traçarmos resultados com 100 neurônios ocultos:



Neste caso, as duas curvas não se encontram. No entanto, nossas experiências sugerem que, com apenas mais algumas épocas de treinamento (não mostradas), as precisões se tornam quase exatamente as mesmas. Portanto, com base nesses experimentos, parece que a inicialização do peso aprimorado apenas acelera o aprendizado, não altera o desempenho final de nossas redes. No entanto, veremos mais a frente alguns exemplos de redes neurais em que o comportamento de longo prazo é significativamente melhor com a inicialização de peso usando:

$$1/\sqrt{n_{in}}.$$

Assim, não é apenas a velocidade de aprendizado que é melhorada, mas também o desempenho final.

A abordagem acima para a inicialização do peso ajuda a melhorar a maneira como nossas redes neurais aprendem. Outras técnicas para inicialização de peso também foram propostas, muitas baseadas nessa ideia básica. Não vamos rever as outras abordagens aqui, já que a descrita anteriormente funciona bem o suficiente para nossos propósitos.

Backpropagation, Gradiente Descendente e Chain Rule

Por Que Precisamos de Backpropagation (Retropropagação)?

Ao projetar uma rede neural, primeiro, precisamos treinar um modelo e atribuir pesos específicos a cada uma das entradas. Esse peso decide o quão vital é esse recurso para nossa previsão. Quanto maior o peso, maior a importância. No entanto, inicialmente, não sabemos o peso específico exigido pelas entradas. Então, o que fazemos é atribuir um peso aleatório às nossas entradas e nosso modelo calcula o erro na previsão. Depois disso, atualizamos nossos valores de peso e executamos novamente o código (retropropagação). Após várias iterações, podemos obter valores de erro mais baixos e maior precisão.

Para que a retropropagação funcione usamos um algoritmo que verifica quanto o valor de cada peso afeta o erro do modelo, calculando as derivadas parciais. Esse algoritmo é chamado de Gradiente Descendente e é aplicado através da Chain Rule (Regra da Cadeia).

Em outras palavras, a retropropagação visa minimizar a função de custo ajustando os pesos e vieses (bias) da rede. O nível de ajuste é determinado pelos gradientes da função de custo em relação a esses parâmetros.

Uma pergunta pode surgir: por que calcular gradientes?

O gradiente de uma função $C(x_1, x_2, \dots, x_m)$ no ponto x é um vetor das derivadas parciais de C em x .

A derivada de uma função C mede a sensibilidade à alteração do valor da função (valor de saída) em relação a uma alteração no argumento x (valor de entrada). Em outras palavras, a derivada nos diz a direção que C está seguindo.

O gradiente mostra quanto o parâmetro x precisa mudar (na direção positiva ou negativa) para minimizar C . O cálculo desses gradientes acontece usando uma técnica chamada Regra da Cadeia.

Mini-Projeto 4 – A Matemática das Redes Neurais Artificiais

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 13</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
## A Matemática das Redes Neurais Artificiais
### Construindo a Rede Neural com Programação e Matemática
![title](imagens/math.png)
Teremos 2 Partes:
- Parte 1 - Vamos construir uma rede neural artificial somente com operações matemáticas
- Parte 2 - Vamos treinar a rede para Prever a Ocorrência de Câncer
## A Arquitetura de Redes Neurais Artificiais
Uma rede neural típica é constituída por um conjunto de neurônios interligados, influenciando uns aos outros formando um sistema maior, capaz de armazenar conhecimento adquirido por meio de exemplos apresentados e, assim, podendo realizar inferências sobre novos conjuntos de dados.
Vejam a arquitetura de redes neurais artificiais.
As redes neurais são comumente apresentadas como um grafo orientado, onde os vértices são os neurônios e as arestas as sinapses. A direção das arestas informa o tipo de alimentação, ou seja, como os neurônios são alimentados (recebem sinais de entrada). As redes neurais derivam seu poder devido a sua estrutura massiva e paralela e a habilidade de aprender por experiência. Essa experiência é transmitida por meio de exemplos obtidos do mundo real, definidos como um conjunto de características formados por dados de entrada e de saída. Se apresentamos esses dados de entrada e saída à rede, estamos diante de aprendizagem supervisionada e caso apresentemos apenas os dados de entrada, estamos diante de aprendizagem não supervisionada!
O conhecimento obtido pela rede através dos exemplos é armazenado na forma de pesos das conexões, os quais serão ajustados a fim de tomar decisões corretas a partir de novas entradas, ou seja, novas situações do mundo real não conhecidas pela rede. O processo de ajuste dos pesos sinápticos é realizado pelo algoritmo de aprendizagem, responsável em armazenar na rede o conhecimento do mundo real obtido através de exemplos. Existem vários algoritmos de aprendizagem, dentre eles o backpropagation que é o algoritmo mais utilizado.
![title](imagens/nnet.png)
### Importando os Pacotes
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# ->
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Por enquanto precisaremos somente do NumPy
import numpy as np
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversons
### Parte 1 - Implementando Uma Rede Neural Artificial Somente com Fórmulas Matemáticas (Sem Frameworks)
Faça a leitura do manual em pdf no próximo item de aprendizagem: Parâmetros x Hiperparâmetros.
### Parte 1A - Forward Propagation
https://arxiv.org/pdf/1905.07490.pdf
![title](imagens/nn.png)
Faça a leitura do manual em pdf no próximo item de aprendizagem: Por Que Inicializamos os Pesos de Um Modelo de Rede Neural?
### Desenvolvendo a Função Para Inicialização de Pesos
# ->
# Função para inicialização randômica dos parâmetros do modelo
def inicializa_parametros(dims_camada_entrada):
    # Dicionário para os parâmetros
    parameters = {}
    # Comprimento das dimensões das camadas
    comp = len(dims_camada_entrada)
    # Loop pelo comprimento
    for i in range(1, comp):
        # Inicialização da matriz de pesos
        parameters["W" + str(i)] = np.random.randn(dims_camada_entrada[i], dims_camada_entrada[i - 1]) * 0.01
        # Inicialização do bias
        parameters["b" + str(i)] = np.zeros((dims_camada_entrada[i], 1))
    return parameters
### Desenvolvendo a Função Sigmóide
A principal razão pela qual usamos a função sigmóide é porque ela permite converter números para valores entre 0 e 1. Portanto, é especialmente usada para modelos em que temos que prever a probabilidade como uma saída. Como a probabilidade de qualquer coisa existir apenas entre o intervalo de 0 e 1, sigmoide é a escolha certa. Algumas características da função sigmóide:
- A função é diferenciável. Isso significa que podemos encontrar a inclinação da curva sigmóide em dois pontos.
- A função sigmóide logística pode fazer com que uma rede neural fique presa no momento do treinamento.
- A função softmax é uma função de ativação logística mais generalizada, utilizada para a classificação em várias classes.
![title](imagens/sigmoid.png)
Se a função parecer muito abstrata ou estranha para você, não se preocupe muito com detalhes como o número de Euler e ou como alguém criou essa função. Para aqueles que não são conhecedores de matemática, a única coisa importante sobre a função sigmóide é primeiro, sua curva e,
```

segundo, sua derivada. Aqui estão mais alguns detalhes:

- **A função sigmóide produz resultados semelhantes aos da função de passo (Step Function) em que a saída está entre 0 e 1. A curva cruza 0,5 a $z = 0$, e podemos definir regras para a função de ativação, como: Se a saída do neurônio sigmóide for maior que ou igual a 0,5, gera 1; se a saída for menor que 0,5, gera 0.**

- A função sigmóide é suave e possui uma derivada simples de $\sigma(z) * (1 - \sigma(z))$, que é diferenciável em qualquer lugar da curva.

- Se z for muito negativo, a saída será aproximadamente 0; se z for muito positivo, a saída é aproximadamente 1; mas em torno de $z = 0$, onde z não é muito grande nem muito pequeno, temos um desvio relativamente maior à medida que z muda.

**Afinal, O Que é Derivada?*

![title](imagens/derivada.png)

No Cálculo, a derivada em um ponto de uma função $y = f(x)$ representa a taxa de variação instantânea de y em relação a x neste ponto.

Um exemplo típico é a função velocidade que representa a taxa de variação (derivada) da função espaço. Do mesmo modo, a função aceleração é a derivada da função velocidade. Geometricamente, a derivada no ponto $x = a$ de $y = f(x)$ representa a inclinação da reta tangente ao gráfico desta função no ponto $(a, f(a))$.

A função que a cada ponto x associa a derivada neste ponto de $f(x)$ é chamada de função derivada de $f(x)$.

![title](imagens/derivada.gif)

Em cada ponto, a derivada de $f(x)$ é a tangente do ângulo que a reta tangente à curva faz em relação ao eixo das abscissas. A reta é sempre tangente à curva azul; a tangente do ângulo que ela faz com o eixo das abscissas é a derivada. Note-se que a derivada é positiva quando verde, negativa quando vermelha, e zero quando preta.

A derivada de uma função $y = f(x)$ num ponto $x = x_0$, é igual ao valor da tangente trigonométrica do ângulo formado pela tangente geométrica à curva representativa de $y=f(x)$, no ponto $x = x_0$, ou seja, a derivada é o coeficiente angular da reta tangente ao gráfico da função no ponto x_0 .

A função derivada é representada por $f'(x)$.

->

Função sigmóide

def sigmoid(Z):

A = 1 / (1 + np.exp(-Z))

return A, Z

Desenvolvendo a Função ReLU

Para usar a descida de gradiente estocástico com retropropagação de erros para treinar redes neurais profundas, é necessária uma função de ativação que se assemelhe e atue como uma função linear, mas é, de fato, uma função não linear que permite que relacionamentos complexos nos dados sejam aprendidos.

A solução é usar a função de ativação linear retificada ou ReL para abreviar. Um nó ou unidade que implementa essa função de ativação é chamado de unidade de ativação linear retificada ou ReLU, para abreviar. Frequentemente, as redes que usam a função retificadora para as camadas ocultas são chamadas de redes retificadas.

A função ReLU é definida como $f(x) = \max(0, x)$. Normalmente, ela é aplicada elemento a elemento à saída de alguma outra função, como um produto de vetor e matriz.

A adoção da ReLU pode ser facilmente considerada um dos marcos na revolução do aprendizado profundo, por ex. as técnicas que agora permitem o desenvolvimento rotineiro de redes neurais muito profundas.

A derivada da função linear retificada também é fácil de calcular. **A derivada da função de ativação é necessária ao atualizar os pesos de um nó como parte da retropropagação de erro.**

A derivada da função é a inclinação. A inclinação para valores negativos é 0,0 e a inclinação para valores positivos é 1,0.

Tradicionalmente, o campo das redes neurais evitou qualquer função de ativação que não fosse completamente diferenciável, talvez adiando a adoção da função linear retificada e de outras funções lineares. Tecnicamente, não podemos calcular a derivada quando a entrada é 0,0; portanto, podemos assumir que é zero. Este não é um problema na prática.

Os gradientes das ativações tangentes e hiperbólicas são menores que a porção positiva da ReLU. Isso significa que a parte positiva é atualizada mais rapidamente à medida que o treinamento avança. No entanto, isso tem um custo. O gradiente 0 no lado esquerdo tem seu próprio problema, chamado "neurônios mortos", no qual uma atualização de gradiente define os valores recebidos para uma ReLU, de modo que a saída é sempre zero; unidades ReLU modificadas, como ELU (ou Leaky ReLU, ou PReLU, etc.) podem melhorar isso.

![title](imagens/relu.png)

->

Função de ativação ReLu (Rectified Linear Unit)

def relu(Z):

A = abs(Z * (Z > 0))

return A, Z

![title](imagens/net-relu.png)

Desenvolvendo a Ativação Linear

->

Operação de ativação

A é a matriz com os dados de entrada

W é a matriz de pesos

b é o bias

def linear_activation(A, W, b):

Z = np.dot(W, A) + b

cache = (A, W, b)

return Z, cache

Construindo o Processo de Forward Propagation

->

Movimento para

frente (forward)

def forward(A_prev, W, b, activation):

Se a função de ativação for Sigmoid, entramos neste bloco

if activation == "sigmoid":

Z, linear_cache = linear_activation(A_prev, W, b)

A, activation_cache = sigmoid(Z)

Se não, se for ReLu, entramos neste bloco

elif activation == "relu":

Z, linear_cache = linear_activation(A_prev, W, b)

A, activation_cache = relu(Z)

cache = (linear_cache, activation_cache)


```

    return A, cache
#### Combinando Ativação e Propagação
# ->
# Propagação para frente
def forward_propagation(X, parameters):
    # Lista de valores anteriores (cache)
    caches = []
    # Dados de entrada
    A = X
    # Comprimento dos parâmetros
    L = len(parameters) // 2
    # Loop
    for i in range(1, L):
        # Guarda o valor prévio de A
        A_prev = A
        # Executa o forward
        A, cache = forward(A_prev, parameters["W" + str(i)], parameters["b" + str(i)], activation = "relu")
        # Grava o cache
        caches.append(cache)
    # Saída na última camada
    A_last, cache = forward(A, parameters["W" + str(L)], parameters["b" + str(L)], activation = "sigmoid")
    # Grava o cache
    caches.append(cache)
    return(A_last, caches)
#### Desenvolvendo a Função de Custo
![title](imagens/custo.png)
# ->
# Função de custo (ou função de erro)
def calcula_custo(A_last, Y):
    # Ajusta o shape de Y para obter seu comprimento (total de elementos)
    m = Y.shape[1]
    # Calcula o custo comparando valor real e previsto
    custo = (-1 / m) * np.sum((Y * np.log(A_last)) + ((1 - Y) * np.log(1 - A_last)))
    # Ajusta o shape do custo
    custo = np.squeeze(custo)
    return(custo)
#### Parte 1B - Backward Propagation
![title](imagens/backpropagation.png)
#### Desenvolvendo o Backward Propagation - Função Sigmóide Backward
# ->
# Função sigmoid para o backpropagation
# Fazemos o cálculo da derivada pois não queremos o valor completo da função, mas sim sua variação
def sigmoid_backward(da, Z):
    # Calculamos a derivada de Z
    dg = (1 / (1 + np.exp(-Z))) * (1 - (1 / (1 + np.exp(-Z))))
    # Encontramos a mudança na derivada de z
    dz = da * dg
    return dz
# Compare com a função sigmoid do forward propagation
# A = 1 / (1 + np.exp(-Z))
#### Desenvolvendo o Backward Propagation - Função ReLu Backward
# ->
# Função relu para o backpropagation
# Fazemos o cálculo da derivada pois não queremos o valor completo da função, mas sim sua variação
def relu_backward(da, Z):
    dg = 1 * (Z >= 0)
    dz = da * dg
    return dz
# Compare com a função relu do forward propagation:
# A = abs(Z * (Z > 0))
#### Desenvolvendo o Backward Propagation - Ativação Linear Backward
# ->
# Ativação linear para o backpropagation
def linear_backward_function(dz, cache):
    # Recebe os valores do cache (memória)
    A_prev, W, b = cache
    # Shape de m
    m = A_prev.shape[1]
    # Calcula a derivada de W (resultado da operação com dz)
    dW = (1 / m) * np.dot(dz, A_prev.T)
    # Calcula a derivada de b (resultado da operação com dz)
    db = (1 / m) * np.sum(dz, axis = 1, keepdims = True)
    # Calcula a derivada da operação
    dA_prev = np.dot(W.T, dz)
    return dA_prev, dW, db
#### Desenvolvendo o Backward Propagation - Ativação Linear Backward
# ->
# Função que define o tipo de ativação (relu ou sigmoid)

```

```

def linear_activation_backward(dA, cache, activation):
    # Extrai o cache
    linear_cache, activation_cache = cache
    # Verifica se a ativação é relu
    if activation == "relu":
        dZ = relu_backward(dA, activation_cache)
        dA_prev, dW, db = linear_backward_function(dZ, linear_cache)
    # Verifica se a ativação é sigmoid
    if activation == "sigmoid":
        dZ = sigmoid_backward(dA, activation_cache)
        dA_prev, dW, db = linear_backward_function(dZ, linear_cache)
    return dA_prev, dW, db
#### Combinando Ativação e Retropropagação - Algoritmo Backpropagation
# ->
# Algoritmo Backpropagation (calcula os gradientes para atualização dos pesos)
# AL = Valor previsto no Forward
# Y = Valor real
def backward_propagation(AL, Y, caches):
    # Dicionário para os gradientes
    grads = {}
    # Comprimento dos dados (que estão no cache)
    L = len(caches)
    # Extrai o comprimento para o valor de m
    m = AL.shape[1]
    # Ajusta o shape de Y
    Y = Y.reshape(AL.shape)
    # Calcula a derivada da previsão final da rede (feita ao final do Forward Propagation)
    dAL = -(Y / AL) - ((1 - Y) / (1 - AL))
    # Captura o valor corrente do cache
    current_cache = caches[L - 1]
    # Gera a lista de gradiente para os dados, os pesos e o bias
    # Fazemos isso uma vez, pois estamos na parte final da rede, iniciando o caminho de volta
    grads["dA" + str(L - 1)], grads["dW" + str(L)], grads["db" + str(L)] = linear_activation_backward(dAL, current_cache, activation = "sigmoid")
    # Loop para calcular a derivada durante as ativações lineares com a relu
    for l in reversed(range(L - 1)):
        # Cache atual
        current_cache = caches[l]
        # Calcula as derivadas
        dA_prev, dW, db = linear_activation_backward(grads["dA" + str(l + 1)], current_cache, activation = "relu")
        # Alimenta os gradientes na lista, usando o índice respectivo
        grads["dA" + str(l)] = dA_prev
        grads["dW" + str(l + 1)] = dW
        grads["db" + str(l + 1)] = db
    return grads
#### Gradientes e Atualização dos Pesos
# ->
# Função de atualização de pesos
def atualiza_pesos(parameters, grads, learning_rate):
    # Comprimento da estrutura de dados com os parâmetros (pesos e bias)
    L = len(parameters)//2
    # Loop para atualização dos pesos
    for l in range(L):
        # Atualização dos pesos
        parameters["W" + str(l + 1)] = parameters["W" + str(l + 1)] - (learning_rate * grads["dW" + str(l + 1)])
        # Atualização do bias
        parameters["b" + str(l + 1)] = parameters["b" + str(l + 1)] - (learning_rate * grads["db" + str(l + 1)])
    return parameters
#### Implementando a Rede Completa
# ->
# Modelo completo da rede neural
def modeloNN(X, Y, dims_camada_entrada, learning_rate = 0.0075, num_iterations = 100):
    # Lista para receber o custo a cada época de treinamento
    custos = []
    # Inicializa os parâmetros
    parametros = inicializa_parametros(dims_camada_entrada)
    # Loop pelo número de iterações (épocas)
    for i in range(num_iterations):
        # Forward Propagation
        AL, caches = forward_propagation(X, parametros)
        # Calcula o custo
        custo = calcula_custo(AL, Y)
        # Backward Propagation
        # Nota: ao invés de AL e Y, poderíamos passar somente o valor do custo
        # Estamos passando o valor de AL e Y para fique claro didaticamente o que está sendo feito
        gradientes = backward_propagation(AL, Y, caches)
        # Atualiza os pesos
        parametros = atualiza_pesos(parametros, gradientes, learning_rate)
        # Print do valor intermediário do custo

```

```

    # A redução do custo indica o aprendizado do modelo
    if i % 10 == 0:
        print("Custo Após " + str(i) + " iterações é " + str(custo))
        custos.append(custo)
    return parametros, custos
# ->
# Função para fazer as previsões
# Não precisamos do Backpropagation pois ao fazer previsões como o modelo treinado,
# teremos os melhores valores de pesos (parametros)
def predict(X, parametros):
    AL, caches = forward_propagation(X, parametros)
    return AL
#### Parte 2 - Vamos treinar a rede para Prever a Ocorrência de Câncer
#### Mini-Projeto 4 - Usando a Rede Neural Para Prever a Ocorrência de Câncer
# ->
# Imports
import sklearn
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
#### Carregando os Dados
https://scikit-learn.org/stable/datasets/index.html#breast-cancer-dataset
# ->
# Carregamos o objeto completo
temp = load_breast_cancer()
# ->
# Tipo do objeto
type(temp)
# ->
# Visualiza o objeto
temp
# ->
# Carregamos o dataset
dados = pd.DataFrame(columns = load_breast_cancer()["feature_names"], data = load_breast_cancer()["data"])
# ->
# Shape
dados.shape
# ->
# Visualiza os dados
dados.head()
# ->
# Verifica se temos valores ausentes
dados.isnull().any()
# ->
# Separa a variável target
target = load_breast_cancer()["target"]
# ->
type(target)
# ->
# Visualiza a variável
target
# ->
# Total de registros por classe - Câncer Benigno
np.count_nonzero(target == 1)
# ->
# Total de registros por classe - Câncer Maligno
np.count_nonzero(target == 0)
# ->
# Vamos extrair os labels
# Dicionário para os labels
labels = {}
# Nomes das classes da variável target
target_names = load_breast_cancer()["target_names"]
# Mapeamento
for i in range(len(target_names)):
    labels.update({i:target_names[i]})
# ->
# Visualiza os labels
labels
# ->

```

```

# Agora preparamos as variáveis preditoras em X
X = np.array(dados)
# ->
# Visualiza os dados de entrada
X
# ->
# Dividimos os dados de entrada e saída em treino e teste
X_treino, X_teste, y_treino, y_teste = train_test_split(X, target, test_size = 0.15, shuffle = True)
# ->
# Shape dos dados de treino
print(X_treino.shape)
print(y_treino.shape)
# ->
# Shape dos dados de teste
print(X_teste.shape)
print(y_teste.shape)
# ->
# Ajusta o shape dos dados de entrada
X_treino = X_treino.T
X_teste = X_teste.T
# ->
print(X_treino.shape)
print(X_teste.shape)
# ->
# Precisamos ajustar também os dados de saída
y_treino = y_treino.reshape(1, len(y_treino))
y_teste = y_teste.reshape(1, len(y_teste))
# ->
print(y_treino.shape)
print(y_teste.shape)
# ->
# Variável com as dimensões de entrada para o número de neurônios
dims_camada_entrada = [X_treino.shape[0],
50, 20, 5, 1]
# ->
dims_camada_entrada
# ->
# Treinamento do modelo
print("\nIniciando o Treinamento.\n")
parametros, custo = modeloNN(X = X_treino,
Y = y_treino,
dims_camada_entrada = dims_camada_entrada,
num_iterations = 3000,
learning_rate = 0.0075)
print("\nTreinamento Concluído.\n")
# ->
# Plot do erro durante o treinamento
plt.plot(custo)
# ->
# Previsões com os dados de treino
y_pred_treino = predict(X_treino, parametros)
# ->
# Visualiza as previsões
y_pred_treino
# ->
# Ajustamos o shape em treino
y_pred_treino = y_pred_treino.reshape(-1)
y_treino = y_treino.reshape(-1)
# ->
y_pred_treino > 0.5
# ->
# Convertemos as previsões para o valor binário de classe
# (0 ou 1, usando como threshold o valor de 0.5 da probabilidade)
y_pred_treino = 1 * (y_pred_treino > 0.5)
# ->
y_pred_treino
# ->
# Calculamos a acurácia comparando valor real com valor previsto
acc_treino = sum(1 * (y_pred_treino == y_treino)) / len(y_pred_treino) * 100
# ->
print("Acurácia nos dados de treino: " + str(acc_treino))
# ->
print(classification_report(y_treino, y_pred_treino, target_names = ['Maligno', 'Benigno']))
# ->
# Previsões com o modelo usando dados de teste
y_pred_teste = predict(X_teste, parametros)
# ->
# Visualiza os dados

```

```

y_pred_teste
# ->
# Ajustamos os shapes
y_pred_teste = y_pred_teste.reshape(-1)
y_teste = y_teste.reshape(-1)
# ->
# Convertemos as previsões para o valor binário de classe
y_pred_teste = 1 * (y_pred_teste > 0.5)
# ->
# Visualizamos as previsões
y_pred_teste
# ->
# Calculamos a acurácia
acuracia = sum(1 * (y_pred_teste == y_teste)) / len(y_pred_teste) * 100
# ->
print("Acurácia nos dados de teste: " + str(acuracia))
# ->
print(classification_report(y_teste, y_pred_teste, target_names = ['Maligno', 'Benigno']))
# Fim

```

Mini-Projeto 5 – Rede Neural com TensorFlow Para Classificação de Imagens de Vestuário

Neste Mini-Projeto vamos construir uma rede neural artificial com TensorFlow para classificação de imagens, especificamente classificação de imagens de roupas e acessórios.

Execute cada célula do Jupyter Notebook deste Mini-Projeto 5 (disponível ao final do capítulo) e acompanhe os comentários. Fique à vontade para fazer mudanças e experimentar diferentes configurações.

Compare o que fazemos neste Mini-Projeto 5 com o que estudamos no Mini-Projeto 4 ao criar a rede usando apenas operações matemáticas. Aqui, usaremos o TensorFlow com o Keras.

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 13</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
## Mini-Projeto 5
### Rede Neural com TensorFlow Para Classificação de Imagens de Vestuário

Neste Mini-Projeto vamos construir uma rede neural artificial com TensorFlow para classificação de imagens, especificamente classificação de
imagens de roupas e acessórios.
Execute cada célula e acompanhe os comentários. Fique à vontade para fazer mudanças e experimentar diferentes configurações. Compare o que
fazemos neste Mini-Projeto 5 com o que estudamos no Mini-Projeto 4 ao criar a rede usando apenas operações matemáticas. Aqui, usaremos o
TensorFlow com o Keras.
### Instalando e Carregando os Pacotes
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# ->
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
> Atenção:
Vamos criar um gráfico do modelo e para isso precisaremos de pacotes Python cujas versões disponíveis com o Anaconda não funcionam
adequadamente. Sendo assim, buscaremos versões mais novas, o que requer o uso do conda.
Encerre este Jupyter Notebook, acesse o terminal ou prompt de comando e execute os comandos abaixo:
- conda install python-graphviz
- conda install pydot
- conda install pydotplus
Digite **yes** quando solicitado. Então retorne ao Jupyter Notebook.
# ->
# Imports
import os
import numpy as np

```

```

import pandas as pd
import tensorflow as tf
from tensorflow import keras
# Configuração de gráficos
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsz=14)
mpl.rc('xtick', labelsz=12)
mpl.rc('ytick', labelsz=12)
# Para tornar a saída deste notebook estável em todas as execuções
np.random.seed(42)
tf.random.set_seed(42)
# ->
# Aqui definimos os diretórios onde salvaremos as imagens
PROJECT_ROOT_DIR = "."
PROJECT_NUM = "Mini-Projeto5"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "dados", PROJECT_NUM)
os.makedirs(IMAGES_PATH, exist_ok = True)
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
Observe que estamos usando a versão do Keras dentro do TensorFlow.
## Carregando e Preparando os Dados
Vamos começar carregando o conjunto de dados Fashion MNIST com artigos de moda masculina e feminina. O Keras possui várias funções para
carregar conjuntos de dados populares no `keras.datasets`. O conjunto de dados já está dividido para você entre um conjunto de treinamento e um
conjunto de testes, mas pode ser útil dividir ainda mais o conjunto de treinamento para ter um conjunto de validação.
# ->
# Carrega o dataset
fashion_mnist = keras.datasets.fashion_mnist
# ->
# Extraímos os dados de treino e de teste
(X_treino_full, y_treino_full), (X_teste, y_teste) = fashion_mnist.load_data()
O conjunto de treinamento contém 60.000 imagens em escala de cinza, cada uma com 28x28 pixels:
# ->
# Shape
X_treino_full.shape
Cada intensidade de pixel é representada como um byte (0 a 255):
# ->
# Tipo de dados
X_treino_full.dtype
Vamos dividir o conjunto de treinamento completo em um conjunto de validação e um conjunto de treinamento (menor). Também dimensionamos
as intensidades de pixel para o intervalo de 0-1 (padronização) e as convertemos em float, dividindo por 255.
# ->
# Preparação dos dados
X_valid, X_treino = X_treino_full[:5000] / 255., X_treino_full[5000:] / 255.
y_valid, y_treino = y_treino_full[:5000], y_treino_full[5000:]
X_teste = X_teste / 255.
Você pode plotar uma imagem usando a função `imshow ()` do Matplotlib, com um mapa de cores **binary**.
# ->
# Plot de uma imagem
plt.imshow(X_treino[0], cmap = "binary")
plt.axis('off')
plt.show()
Os rótulos são os IDs de classe (representados como uint8), de 0 a 9:
# ->
# Labels (dados de saída) de treino
y_treino
Aqui estão os nomes de classe correspondentes:
# ->
# Nomes das classes
nomes_classe = ["T-shirt/top",
               "Trouser",
               "Pullover",
               "Dress",
               "Coat",
               "Sandal",
               "Shirt",
               "Sneaker",
               "Bag",
               "Ankle boot"]
Mais detalhes do dataset aqui: https://www.tensorflow.org/datasets/catalog/fashion\_mnist
Portanto, a primeira imagem no conjunto de treinamento é um casaco:
# ->
# Nome de classe
nomes_classe[y_treino[0]]
O conjunto de validação contém 5.000 imagens e o conjunto de testes contém 10.000 imagens:

```

```

# ->
# Shape
X_valid.shape
# ->
# Shape
X_teste.shape
Vamos dar uma olhada em uma amostra das imagens no conjunto de dados.
# ->
# Função para salvar as imagens
def salva_imagem(fig_id, tight_layout = True, fig_extension = "png", resolution = 300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Salvando a imagem...", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format = fig_extension, dpi = resolution)
# ->
# Plot de algumas imagens
# Vamos plotar 4 linhas e 10 coluns
n_rows = 4
n_cols = 10
# Área de plotagem
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
# Loop pelas linhas e colunas
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_treino[index], cmap = "binary", interpolation = "nearest")
        plt.axis('off')
        plt.title(nomes_classe[y_treino[index]], fontsize = 12)
# Ajusta os plots
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
# Salva a imagem em disco
salva_imagem('plot_fashion_mnist', tight_layout = False)
# Mostra a imagem
plt.show()
Perfeito! Os dados estão prontos. Vamos construir e treinar o modelo.
## Construção do Modelo
# ->
# Modelo de Rede Neural com 2 Camadas Densas
# Cria o objeto do tipo sequência
modelo = keras.models.Sequential()
# Camada para receber os dados de entrada
modelo.add(keras.layers.Flatten(input_shape = [28, 28]))
# Primeira camada oculta com ativação relu
modelo.add(keras.layers.Dense(300, activation = "relu"))
# Segunda camada oculta com ativação relu
modelo.add(keras.layers.Dense(100, activation = "relu"))
# Camada de saída com ativação softmax
# Teremos uma probabilidade prevista para cada classe
modelo.add(keras.layers.Dense(10, activation = "softmax"))
# ->
# Limpamos a sessão Keras e
keras.backend.clear_session()
# ->
# Camadas do modelo
modelo.layers
# ->
# Sumário do modelo
modelo.summary()
# ->
# Vamos criar um plot com o modelo completo e salvar a imagem em disco
keras.utils.plot_model(modelo, IMAGES_PATH + "/modelo_fashion_mnist.png", show_shapes = True)
# ->
# Vamos nomear a primeira camada oculta do modelo
hidden1 = modelo.layers[1]
hidden1.name
# ->
# Verificamos se a camada com novo nome existe
modelo.get_layer(hidden1.name) is hidden1
# ->
# Extraímos pesos e bias da primeira camada oculta
weights, biases = hidden1.get_weights()
# ->
# Pesos que serão usados no começo do treinamento e são gerados de forma aleatória pelo Keras/TensorFlow
weights
# ->
# Shape

```

```

weights.shape
# ->
# Bias que serão usados no começo do treinamento
biases
# ->
# Shape
biases.shape
# ->
# Agora compilamos o modelo com o otimizador, função de custo e a métrica
# https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD
# https://www.tensorflow.org/api_docs/python/tf/keras/losses/sparse_categorical_crossentropy
modelo.compile(optimizer = "sgd", loss = "sparse_categorical_crossentropy", metrics = ["accuracy"])
Podemos então treinar o modelo.
# ->
# Treinamento
history = modelo.fit(X_treino,
                    y_treino,
                    epochs = 50,
                    validation_data = (X_valid, y_valid))
# ->
# Hiperparâmetros do modelo
history.params
# ->
# Aqui estão as métricas disponíveis após o treinamento (erro e acurácia)
history.history.keys()
# ->
# Colocamos o histórico de treinamento em um dataframe, plotamos e salvamos a figura
pd.DataFrame(history.history).plot(figsize = (8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
salva_imagem("plot_keras_learning_curves")
plt.show()
## Avaliando o Modelo
Você já conhece o ritual. Depois de treinar, testamos o modelo com dados de teste.
# ->
# Avalia o modelo
modelo.evaluate(X_teste, y_teste)
Conseguimos mais de 88% de acurácia em teste. Vamos fazer a previsão de algumas imagens.
# ->
# Vamos extrair 5 imagens de teste
X_new = X_teste[:5]
# ->
# E então prever a probabilidade de cada classe para cada imagem
y_proba = modelo.predict(X_new)
# ->
# Previsões de probabilidade
y_proba
# ->
# As previsões de classes são mais fáceis de interpretar
y_proba.round(2)
# ->
# Vamos gravar as previsões das 5 imagens
y_pred = modelo.predict_classes(X_new)
y_pred
# ->
# E então extraímos os nomes das classes associados a cada previsão
np.array(nomes_classes)[y_pred]
Vamos plotar as previsões.
# ->
# Plot
plt.figure(figsize = (8, 5))
for index, image in enumerate(X_new):
    plt.subplot(1, 5, index + 1)
    plt.imshow(image, cmap = "binary", interpolation = "nearest")
    plt.axis('off')
    plt.title(nomes_classes[y_teste[index]], fontsize = 12)
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
salva_imagem('plot_previsoes_fashion_mnist_images', tight_layout = False)
plt.show()
Previsões feitas com sucesso!
# Fim

```


Previendo os Efeitos do Consumo de Álcool em Doenças do Fígado

Neste estudo de caso vamos usar um modelo de rede neural para prever os efeitos do consumo de álcool em doenças do fígado.

O fígado é um dos maiores e mais importantes órgãos do corpo humano. O peso do fígado é de cerca de 1,36 kg e é de cor marrom avermelhada.

O fígado desempenha mais de 500 funções, como produção de bile, produção de proteínas importantes para a coagulação do sangue, purificação do sangue, auxílio na digestão de gorduras, decomposição de glóbulos vermelhos e desintoxicação de produtos químicos nocivos.

A doença hepática causada por vírus hepatotrópicos impõe uma carga substancial aos recursos de saúde. O diagnóstico preciso dos pacientes é muito importante na ciência médica. A medicação errada pode levar ao desperdício de dinheiro e tempo para os pacientes, às vezes isso pode levar à perda irreparável (morte).

Para este trabalho, usaremos uma versão modificada do dataset disponível no UCI:

<https://archive.ics.uci.edu/ml/datasets/liver+disorders>

O arquivo que você encontra junto com o script ao final do capítulo, contém mais colunas do que a versão original. Os dados adicionados são fictícios (embora possam representar dados originais). As colunas representam o resultado de exames de sangue dos pacientes e nosso objetivo é prever a ocorrência de doença do fígado em 5 novos pacientes com base em seus exames de sangue. Faça uma pesquisa para compreender o resultado de um exame de sangue.

```
# Estudo de Caso - Prevendo os Efeitos do Consumo de Álcool em Doenças do Fígado
```

```
# Leia o manual em pdf no Capítulo 13 do curso com a especificação do estudo de caso.
```

```
# Obs: Caso tenha problemas com a acentuação, consulte este link:
```

```
# https://support.rstudio.com/hc/en-us/articles/200532197-Character-Encoding
```

```
# Definindo o diretório de trabalho
```

```
getwd()
```

```
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap13/R/EstudoCaso")
```

```
# Pacotes
```

```
library(dplyr)
```

```
library(caret)
```

```
library(neuralnet)
```

```
##### Carregando os dados #####
```

```
df_pacientes <- read.csv("dados/dataset.csv")
```

```
View(df_pacientes)
```

```
##### Análise Explorória #####
```

```
# Vamos verificar os tipos de dados
```

```
str(df_pacientes)
```

```
# Vamos checar se temos valores missing
```

```
sum(is.na(df_pacientes))
```

```
# Resumo estatístico
```

```
summary(df_pacientes)
```

```
# Proporção da classe
```

```
# Em nosso dataset, a coluna target "classe" representa:
```

```
# 1 significa a presença de doença do fígado
```

```
# 2 significa que nenhuma doença do fígado foi identificada
```

```
table(df_pacientes$classe)
```

```
prop.table(table(df_pacientes$classe))
```

```
# As classes estão desbalanceadas, com mais registros da classe positiva (tem doença).
```

```

# Cuidaremos disso daqui a pouco

##### Pré-Processamento e Engenharia de Atributos #####

# Vamos criar outra coluna chamada "doente" e preencher com os valores sim/nao
# Esse passo não é obrigatório, sendo apenas didático
# Usaremos essa nova coluna como coluna target
df_pacientes["doente"] <- ifelse(df_pacientes$classe == 2, "nao", "sim")
View(df_pacientes)
str(df_pacientes)

# Transformando a coluna target em fator
df_pacientes["doente"] <- factor(df_pacientes$doente)
str(df_pacientes)

# A coluna alkphos possui 4 valores nulos. Vamos fazer imputação substituindo pela mediana.
alkphos_mediana <- median(df_pacientes$alkphos, na.rm = T)
df_pacientes$alkphos[is.na(df_pacientes$alkphos)] <- alkphos_mediana
sum(is.na(df_pacientes))

# Vamos criar variáveis dummy para o sexo do paciente
df_pacientes["Mulher"] <- ifelse(df_pacientes$sexo == "Mulher", 1, 0)
df_pacientes["Homem"] <- ifelse(df_pacientes$sexo == "Homem", 1, 0)
View(df_pacientes)
str(df_pacientes)

# Divisão dos dados em treino e teste com proporção 70/30
df_treino <- df_pacientes[1:as.integer(0.70 * nrow(df_pacientes)),]
df_teste <- df_pacientes[(1+as.integer(0.70 * nrow(df_pacientes))),]

# Verificamos os dados
View(df_treino)
View(df_teste)

# Verificamos os tipos de dados
str(df_treino)
str(df_teste)

# Vimos na análise exploratória que as classes estão desbalanceadas.
# Vamos aplicar o upsampling e criar amostras para a classe negativa
# Fazemos isso apenas para dados de treino
?upSample
df_treino <- upSample(x = df_treino, df_treino$doente)
prop.table(table(df_treino$doente))
str(df_treino)

# Função para padronização das variáveis quantitativas
func_normaliza <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}

# Aplica a função
# Observe que estamos excluindo da padronização as colunas de índice 2 e de 11 a 15
# Por que? Porque são colunas categóricas, onde a padronização não é necessária
str(df_treino)
df_treino_norm <- as.data.frame(lapply(df_treino[, -c(2, 11:15)], FUN = func_normaliza))
df_teste_norm <- as.data.frame(lapply(df_teste[, -c(2, 11:15)], FUN = func_normaliza))

# Agora criamos o dataframe final retornando as colunas que não precisam de padronização
# e que usaremos para modelagem preditiva
df_treino_final <- data.frame(df_treino_norm, df_treino[, c(13, 14, 12)])
df_teste_final <- data.frame(df_teste_norm, df_teste[, c(13, 14, 12)])

# Visualiza os dados
View(df_treino_final)
View(df_teste_final)

# Verifica os tipos de dados e total de variáveis
str(df_treino_final)
str(df_teste_final)

##### Modelagem Preditiva #####

# Construindo o modelo
formula_nn <- paste("doente", paste(colnames(df_treino_final)[-12]), collapse = "+", sep = "~")
modelo_figado <- neuralnet(formula_nn, data = df_treino_final)

# Resumo do modelo

```

```
str(modelo_figado)
plot(modelo_figado)

# Previsões com o modelo treinado
set.seed(7)
previsoes_figado <- predict(modelo_figado, df_teste_final[1:11])
View(previsoes_figado)

# O resultado das previsões é em probabilidade. Vamos ajustar a saída.
previsoes_figado_final <- ifelse(previsoes_figado[,1] > previsoes_figado[,2], "nao", "sim")
View(previsoes_figado_final)

# Acurácia do modelo
mean(previsoes_figado_final == df_teste_final$doente)

# Previsão com novos dados
# Vejamos de os 5 novos pacientes podem desenvolver doença do fígado
# com base nos resultados dos exames de sangue

##### Carregando os dados #####
df_novos_pacientes <- read.csv("dados/novos_pacientes.csv")
View(df_novos_pacientes)
str(df_novos_pacientes)

# Padronizamos os novos dados
df_novos_pacientes_norm <- as.data.frame(lapply(df_novos_pacientes, FUN = func_normaliza))
View(df_novos_pacientes_norm)

# Previsões
previsoes_novos_pacientes <- predict(modelo_figado, df_novos_pacientes_norm)
previsoes_novos_pacientes_final <- ifelse(previsoes_novos_pacientes[,1] > previsoes_novos_pacientes[,2], "nao", "sim")
View(previsoes_novos_pacientes_final)

# Previsões feitas com sucesso!
```

7.14. Redes Neurais Artificiais Profundas (Deep Learning)

O Que é Deep Learning?

Deep Learning é uma subcategoria de Machine Learning.

Inteligência Artificial > Machine Learning > Redes Neurais > Deep Learning.

Deep learning é uma rede neural com mais camadas intermediárias (hidden layer).

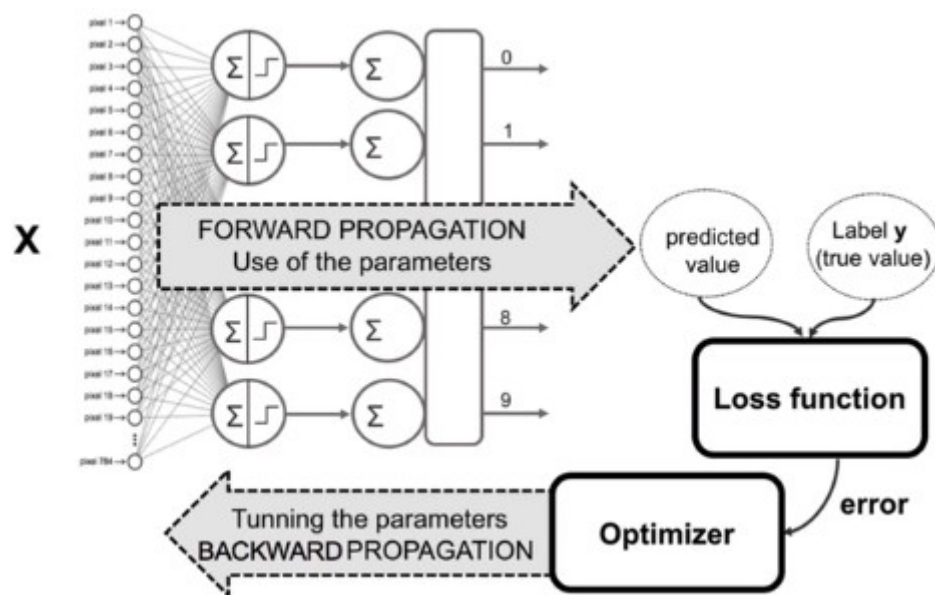
Processo de Aprendizagem de Modelos de Deep Learning

Em Deep Learning nosso trabalho é preparar as estruturas de dados. O aprendizado de recursos e a classificação/regressão são ambos feitos pelo modelo.

Basic features (atributos de entrada) → Embeddings (retorno numérico que representa as palavras de um texto, por exemplo) → Convolution (aprende os detalhes sobre os recursos) → Max pooling (redução de dimensionalidade) → “Supervised” learning (modelo de classificação).

Em Visão Computacional, primeiro a rede realiza o processo de aprendizagem dos recursos e depois faz a classificação.

A aprendizagem ocorre em duas etapas: Forward Propagation e Backward Propagation!



Otimização com Stochastic Gradient Descent

O treinamento de uma rede neural é convertido em um problema de otimização, cujo objetivo é minimizar o erro cometido pela rede, quando considerados todos os exemplos de treinamento.

O gradiente de uma função f mede o quanto f varia uma vez que seus argumentos são alterados. Se f for uma função multivariada de n variáveis, então ∇f é um vetor n -dimensional cujas componentes são as derivadas parciais de f .

Além de ser computacionalmente intensivo, com Gradient Descent você precisa calcular o gradiente de cada elemento do seu conjunto de treinamento, o que pode levar muito tempo em grandes conjuntos de dados.

A solução encontrada para esse problema, foi o Stochastic Gradient Descent (SGD) que é uma versão do Gradient Descent, em que trabalhamos com amostras aleatórias.

SGD é uma aproximação de Gradient Descent e quanto mais lotes processados pela rede neural (ou seja, mais amostra aleatórias), melhor a aproximação.

A implementação do SGD compreende:

1. Amostragem aleatória de um lote de dados do conjunto de dados total.
2. Executar a rede para frente e para trás para calcular o gradiente (com dados gerados no item 1).
3. Aplicar a atualização de descida de gradiente.
4. Repetir os passos 1 a 3 até que a convergência ou o ciclo seja interrompido por outro mecanismo, ou seja, o número de épocas (epochs).

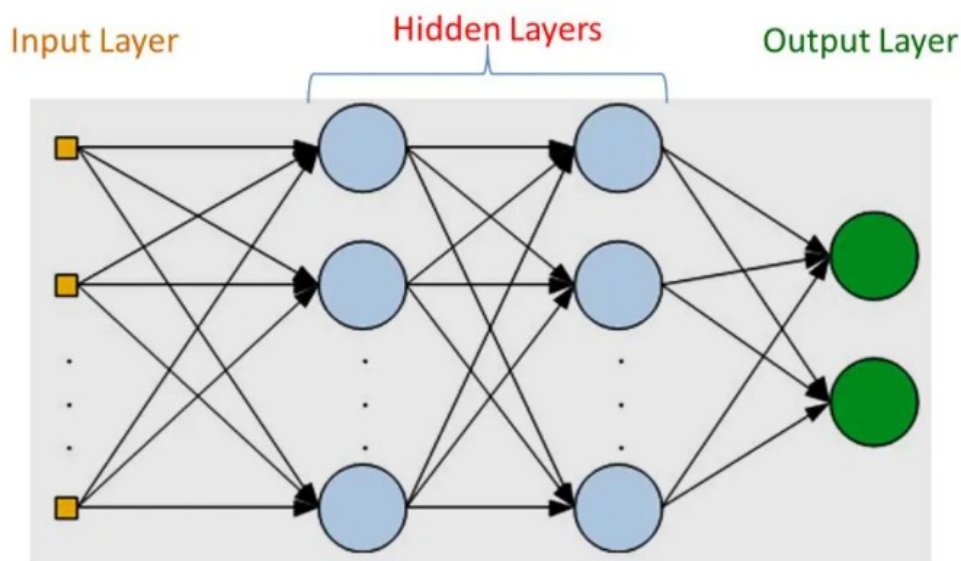
A técnica de Stochastic Gradient Descent está no cerne do Deep Learning. Isso ocorre porque o SGD se equilibra bem com os dados e o tamanho do modelo, e queremos trabalhar com Big Data e modelos com muitas camadas ocultas.

Principais Arquiteturas de Redes Neurais

São 10 as principais arquiteturas de Redes Neurais (dentre elas as principais arquiteturas de Deep Learning):

1- Redes Multilayer Perceptrons

O Perceptron, conforme estudamos nos capítulos anteriores, é um algoritmo simples destinado a realizar a classificação binária; isto é, prevê se a entrada pertence a uma determinada categoria de interesse ou não: fraude ou não_fraude, gato ou não_gato.



Um Perceptron é um classificador linear; ou seja, é um algoritmo que classifica a entrada separando duas categorias com uma linha reta. A entrada geralmente é um vetor de recursos x multiplicado por

pesos w e adicionado a um viés (ou bias) b . Aqui um exemplo do Perceptron: $y = w * x + b$. Um Perceptron produz uma única saída com base em várias entradas de valor real, formando uma combinação linear usando os pesos (e às vezes passando a saída através de uma função de ativação não linear).

Rosenblatt construiu um Perceptron de uma camada. Ou seja, seu algoritmo não inclui múltiplas camadas, o que permite que as redes neurais modelem uma hierarquia de recursos. Isso impede que o Perceptron consiga realizar classificação não linear, como a função XOR (um disparador do operador XOR quando a entrada exibe uma característica ou outra, mas não ambas, significa “OR exclusivo” “), como Minsky e Papert mostraram em seu livro.

Um Multilayer Perceptron (MLP) é uma rede neural artificial composta por mais de um Perceptron. Eles são compostos por uma camada de entrada para receber o sinal, uma camada de saída que toma uma decisão ou previsão sobre a entrada, e entre esses dois, um número arbitrário de camadas ocultas que são o verdadeiro mecanismo computacional do MLP. MLPs com uma camada oculta são capazes de aproximar qualquer função contínua.

O Multilayer Perceptron é uma espécie de “Hello World” da aprendizagem profunda: uma boa forma de começar quando você está aprendendo sobre Deep Learning.

Os MLPs são frequentemente aplicados a problemas de aprendizagem supervisionados: treinam em um conjunto de pares entrada-saída e aprendem a modelar a correlação (ou dependências) entre essas entradas e saídas. O treinamento envolve o ajuste dos parâmetros, ou os pesos e bias, do modelo para minimizar o erro. O backpropagation é usado para fazer os ajustes dos pesos e de bias em relação ao erro, e o próprio erro pode ser medido de várias maneiras, inclusive pelo erro quadrático médio (MSE – Mean Squared Error).

As redes feed forward, como MLPs, são como ping-pong. Elas são principalmente envolvidas em dois movimentos, uma constante de ida e volta. Na passagem para a frente, o fluxo de sinal se move da camada de entrada através das camadas ocultas para a camada de saída e a decisão da camada de saída é medida em relação às saídas esperadas.

Na passagem para trás, usando o backpropagation e a regra da cadeia (Chain Rule), derivadas parciais da função de erro dos vários pesos e bias são reproduzidos através do MLP. Esse ato de diferenciação nos dá um gradiente, ao longo do qual os parâmetros podem ser ajustados à medida que movem o MLP um passo mais perto do erro mínimo. Isso pode ser feito com qualquer algoritmo de otimização baseado em gradiente, como descida estocástica do gradiente. A rede continua jogando aquele jogo de ping-pong até que o erro não possa mais ser reduzido (chegou ao mínimo possível). Este estado é conhecido como convergência.

2- Redes Neurais Convolucionais

Em 1998, Yann LeCun e seus colaboradores desenvolveram um reconhecedor, realmente bom, para dígitos manuscritos chamado LeNet. Ele usou o backpropagation em uma rede feed forward com muitas camadas ocultas, muitos mapas de unidades replicadas em cada camada, agrupando as saídas de unidades próximas, formando uma rede ampla que pode lidar com vários caracteres ao mesmo tempo, mesmo se eles se sobrepõem e uma inteligente maneira de treinar um sistema completo, não apenas um reconhecedor. Mais tarde, esta arquitetura foi formalizada sob o nome de redes neurais convolucionais.

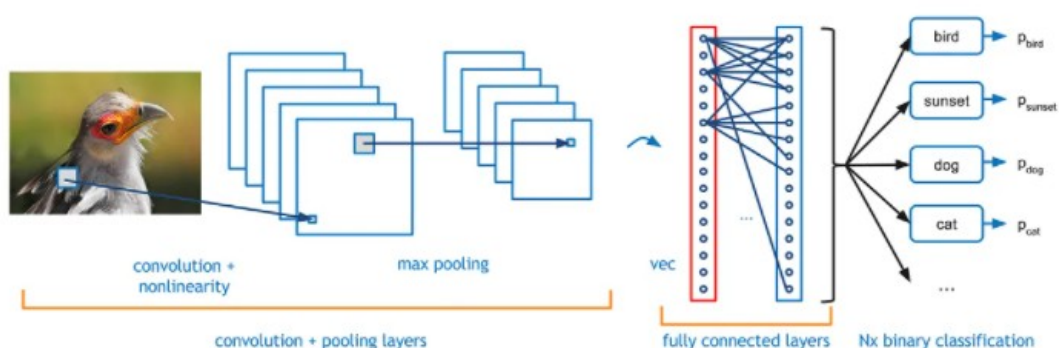
As Redes Neurais Convolucionais (ConvNets ou CNNs) são redes neurais artificiais profundas que podem ser usadas para classificar imagens, agrupá-las por similaridade (busca de fotos) e realizar reconhecimento de objetos dentro de cenas. São algoritmos que podem identificar rostos, indivíduos, sinais de rua, cenouras, ornitorrincos e muitos outros aspectos dos dados visuais.

As redes convolucionais realizam o reconhecimento óptico de caracteres (OCR) para digitalizar texto e tornar possível o processamento de linguagem natural em documentos analógicos e manuscritos, onde as imagens são símbolos a serem transcritos. CNNs também podem ser aplicadas a arquivos de áudio quando estes são representados visualmente como um espectrograma. Mais recentemente, as redes convolucionais foram aplicadas diretamente à análise de texto, bem como dados gráficos.

A eficácia das redes convolucionais no reconhecimento de imagem é uma das principais razões pelas quais o mundo testemunhou a eficácia do aprendizado profundo. Este tipo de rede está impulsionando grandes avanços em Visão Computacional, que tem aplicações óbvias em carros autônomos, robótica, drones, segurança, diagnósticos médicos e tratamentos para deficientes visuais.

As redes convolucionais ingerem e processam imagens como tensores e tensores são matrizes de números com várias dimensões. Eles podem ser difíceis de visualizar, então vamos abordá-los por analogia. Um escalar é apenas um número, como 7; um vetor é uma lista de números (por exemplo, [7,8,9]); e uma matriz é uma grade retangular de números que ocupam várias linhas e colunas como uma planilha. Geometricamente, se um escalar é um ponto de dimensão zero, então um vetor é uma linha unidimensional, uma matriz é um plano bidimensional, uma pilha de matrizes é um cubo tridimensional e quando cada elemento dessas matrizes tem uma pilha de mapas de recursos ligados a ele, você entra na quarta dimensão. Calma, não se desespere (ainda). Veremos isso mais a frente com calma, quando estudarmos exclusivamente esta arquitetura. Em nossos cursos na Data Science Academy incluímos aulas completas sobre Álgebra Linear, onde escalares, vetores, matrizes e tensores são estudados na teoria e prática, pois este conhecimento é fundamental na construção de redes neurais profundas.

A primeira coisa a saber sobre redes convolucionais é que elas não percebem imagens como os humanos. Portanto, você terá que pensar de uma maneira diferente sobre o que uma imagem significa quando é alimentada e processada por uma rede convolucional.



As redes convolucionais percebem imagens como volumes; isto é, objetos tridimensionais, em vez de estruturas planas a serem medidas apenas por largura e altura. Isso porque as imagens de cores digitais têm uma codificação vermelho-verde-azul (RGB – Red-Green-Blue), misturando essas três

cores para produzir o espectro de cores que os seres humanos percebem. Uma rede convolucional recebe imagens como três estratos separados de cores empilhados um em cima do outro.

Assim, uma rede convolucional recebe uma imagem como uma caixa retangular cuja largura e altura são medidas pelo número de pixels ao longo dessas dimensões e cuja profundidade é de três camadas profundas, uma para cada letra em RGB. Essas camadas de profundidade são referidas como canais.

À medida que as imagens se movem através de uma rede convolucional, descrevemos em termos de volumes de entrada e saída, expressando-as matematicamente como matrizes de múltiplas dimensões dessa forma: $30 \times 30 \times 3$. De camada em camada, suas dimensões mudam à medida que atravessam a rede neural convolucional até gerar uma série de probabilidades na camada de saída, sendo uma probabilidade para cada possível classe de saída. Aquela com maior probabilidade, será a classe definida para a imagem de entrada, um pássaro por exemplo.

Você precisará prestar muita atenção às medidas de cada dimensão do volume da imagem, porque elas são a base das operações de álgebra linear usadas para processar imagens.

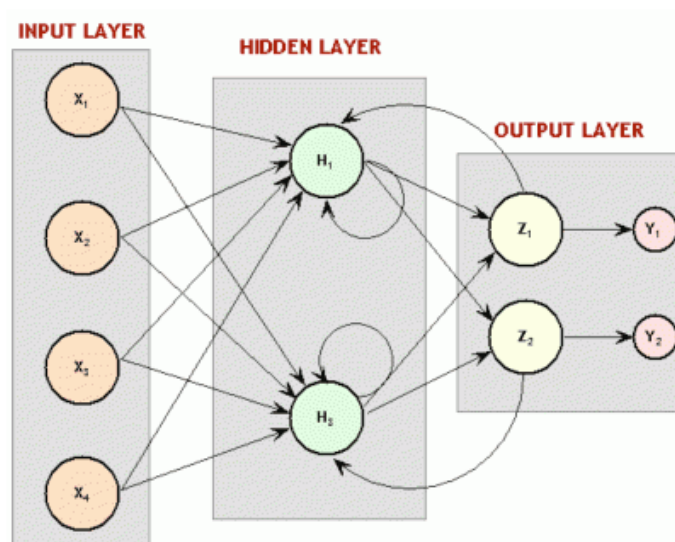
3- Redes Neurais Recorrentes

As redes recorrentes são um poderoso conjunto de algoritmos de redes neurais artificiais especialmente úteis para o processamento de dados sequenciais, como som, dados de séries temporais ou linguagem natural. Uma versão de redes recorrentes foi usada pelo DeepMind no projeto de videogames com agentes autônomos.

As redes recorrentes diferem das redes feed forward porque incluem um loop de feedback, pelo qual a saída do passo $n-1$ é alimentada de volta à rede para afetar o resultado do passo n , e assim por diante para cada etapa subsequente. Por exemplo, se uma rede é exposta a uma palavra letra por letra, e é solicitado a adivinhar cada letra a seguir, a primeira letra de uma palavra ajudará a determinar o que uma rede recorrente pensa que a segunda letra pode ser.

Isso difere de uma rede feed forward, que aprende a classificar cada número manuscrito por exemplo, independentemente, e de acordo com os pixels de que é exposto a partir de um único exemplo, sem se referir ao exemplo anterior para ajustar suas previsões. As redes feed forward aceitam uma entrada por vez e produzem uma saída. As redes recorrentes não enfrentam a mesma restrição um-para-um.

Embora algumas formas de dados, como imagens, não pareçam ser sequenciais, elas podem ser entendidas como sequências quando alimentadas em uma rede recorrente. Considere uma imagem de uma palavra manuscrita. Assim como as redes recorrentes processam a escrita manual, convertendo cada imagem em uma letra e usando o início de uma palavra para adivinhar como essa palavra terminará, então as redes podem tratar parte de qualquer imagem como letras em uma sequência. Uma rede neural que percorre uma imagem grande pode aprender a partir de cada região, o que as regiões vizinhas, são mais prováveis de ser.



As redes recorrentes e as redes feed forward “lembram” algo sobre o mundo, modelando os dados que estão expostos. Mas elas se lembram de maneiras muito diferentes. Após o treinamento, a rede feed forward produz um modelo estático dos dados e esse modelo pode então aceitar novos exemplos e classificá-los ou agrupá-los com precisão.

Em contraste, as redes recorrentes produzem modelos dinâmicos – ou seja, modelos que mudam ao longo do tempo – de formas que produzem classificações precisas dependentes do contexto dos exemplos que estão expostos.

Para ser preciso, um modelo recorrente inclui o estado oculto que determinou a classificação anterior em uma série. Em cada etapa subsequente, esse estado oculto é combinado com os dados de entrada do novo passo para produzir a) um novo estado oculto e, em seguida, b) uma nova classificação. Cada estado oculto é reciclado para produzir seu sucessor modificado.

As memórias humanas também são conscientes do contexto, reciclando a consciência de estados anteriores para interpretar corretamente novos dados. Por exemplo, vamos considerar dois indivíduos. Um está ciente de que ele está perto da casa de Bob. O outro está ciente de que entrou em um avião. Eles interpretarão os sons “Oi Bob!” de duas formas muito diferentes, precisamente porque retêm um estado oculto afetado por suas memórias de curto prazo e sensações precedentes.

Diferentes lembranças de curto prazo devem ser recontadas em momentos diferentes, a fim de atribuir o significado certo à entrada atual. Algumas dessas memórias terão sido forjadas recentemente e outras memórias terão forjado muitos passos antes de serem necessários. A rede recorrente que efetivamente associa memórias e entrada remota no tempo é chamada de Memória de Longo Prazo (LSTM), a qual veremos em seguida.

4- Long Short-Term Memory (LSTM)

Em meados dos anos 90, a proposta dos pesquisadores alemães Sepp Hochreiter e Juergen Schmidhuber apresentou uma variação da rede recorrente com as chamadas unidades de Long Short-Term Memory, como uma solução para o problema do vanishing gradient, problema comum em redes neurais recorrentes.

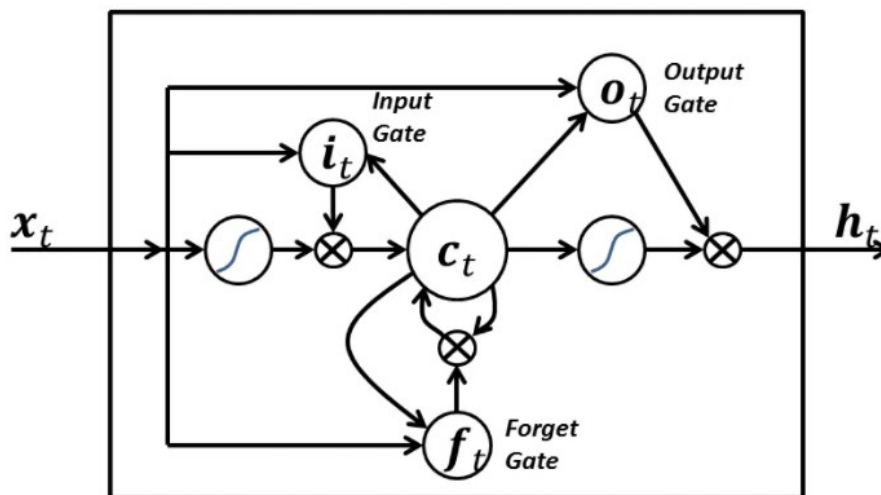
Os LSTMs ajudam a preservar o erro que pode ser copiado por tempo e camadas. Ao manter um erro mais constante, eles permitem que as redes recorrentes continuem aprendendo durante vários

passos de tempo (mais de 1000), abrindo assim um canal para vincular causas e efeitos remotamente. Este é um dos desafios centrais para a aprendizagem de máquina e a IA, uma vez que os algoritmos são frequentemente confrontados por ambientes onde os sinais de recompensa são escassos e atrasados, como a própria vida. (Os pensadores religiosos abordaram este mesmo problema com ideias de karma ou recompensas divinas, teorizando consequências invisíveis e distantes para nossas ações).

Os LSTMs contêm informações fora do fluxo normal da rede recorrente em uma célula fechada. As informações podem ser armazenadas, escritas ou lidas a partir de uma célula, como dados na memória de um computador. A célula toma decisões sobre o que armazenar, e quando permitir leituras, gravações e exclusões, através de portões abertos e fechados. Ao contrário do armazenamento digital em computadores, no entanto, esses portões são analógicos, implementados com a multiplicação de elementos por sigmóides, que estão todos na faixa de 0-1. Analógico tem a vantagem sobre o digital de ser diferenciável e, portanto, adequado para backpropagation.

Esses portões atuam sobre os sinais que recebem e, de forma semelhante aos nós da rede neural, eles bloqueiam ou transmitem informações com base em sua força e importância, que eles filtram com seus próprios conjuntos de pesos. Esses pesos, como os pesos que modulam a entrada e estados ocultos, são ajustados através do processo de aprendizagem das redes recorrentes. Ou seja, as células aprendem quando permitir que os dados entrem, saiam ou sejam excluídos através do processo iterativo de fazer suposições, calculando o erro durante o backpropagation e ajustando pesos através da descida do gradiente.

O diagrama abaixo ilustra como os dados fluem através de uma célula de memória e são controlados por seus portões.

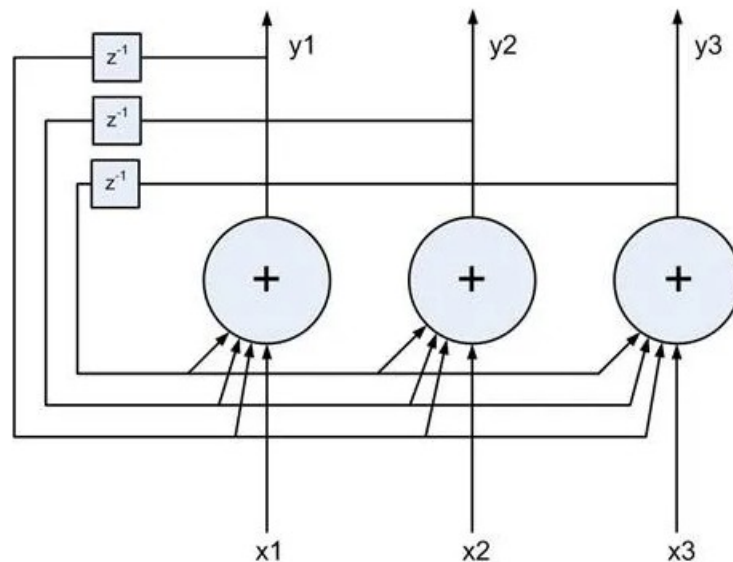


Os LSTM's possuem muitas aplicações práticas, incluindo processamento de linguagem natural, geração automática de texto e análise de séries temporais.

5- Redes de Hopfield

Redes recorrentes de unidades não lineares geralmente são muito difíceis de analisar. Elas podem se comportar de muitas maneiras diferentes: se estabelecer em um estado estável, oscilar ou seguir trajetórias caóticas que não podem ser previstas no futuro. Uma Rede Hopfield é composta por unidades de limite binário com conexões recorrentes entre elas. Em 1982, John Hopfield percebeu

que, se as conexões são simétricas, existe uma função de energia global. Cada “configuração” binária de toda a rede possui energia, enquanto a regra de decisão do limite binário faz com que a rede se conforme com um mínimo desta função de energia. Uma excelente maneira de usar esse tipo de computação é usar memórias como energia mínima para a rede neural. Usar mínimos de energia para representar memórias resulta em uma memória endereçável ao conteúdo. Um item pode ser acessado por apenas conhecer parte do seu conteúdo. É robusto contra danos no hardware.



Cada vez que memorizamos uma configuração, esperamos criar um novo mínimo de energia. Mas e se dois mínimos próximos estão em um local intermediário? Isso limita a capacidade de uma Rede Hopfield. Então, como aumentamos a capacidade de uma Rede Hopfield? Os físicos adoram a ideia de que a matemática que eles já conhecem pode explicar como o cérebro funciona. Muitos artigos foram publicados em revistas de física sobre Redes Hopfield e sua capacidade de armazenamento. Eventualmente, Elizabeth Gardner descobriu que havia uma regra de armazenamento muito melhor que usa a capacidade total dos pesos. Em vez de tentar armazenar vetores de uma só vez, ela percorreu o conjunto de treinamento muitas vezes e usou o procedimento de convergência Perceptron para treinar cada unidade para ter o estado correto, dado os estados de todas as outras unidades nesse vetor. Os estatísticos chamam essa técnica de “pseudo-probabilidade”.

Existe outro papel computacional para as Redes Hopfield. Em vez de usar a rede para armazenar memórias, usamos para construir interpretações de entrada sensorial. A entrada é representada pelas unidades visíveis, a interpretação é representada pelos estados das unidades ocultas e o erro da interpretação é representado pela energia.

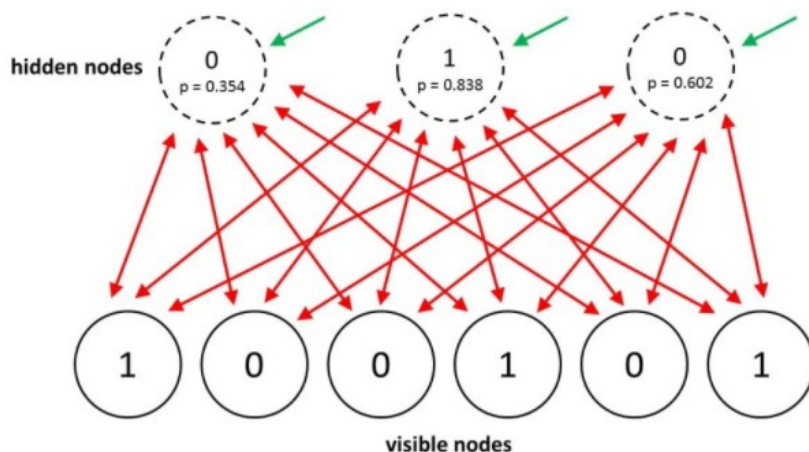
6- Máquinas de Boltzmann

Uma Máquina de Boltzmann é um tipo de rede neural recorrente estocástica. Pode ser visto como a contrapartida estocástica e generativa das Redes Hopfield. Foi uma das primeiras redes neurais capazes de aprender representações internas e é capaz de representar e resolver problemas combinatórios difíceis.

O objetivo do aprendizado do algoritmo da Máquina de Boltzmann é maximizar o produto das probabilidades que a Máquina de Boltzmann atribui aos vetores binários no conjunto de

treinamento. Isso equivale a maximizar a soma das probabilidades de log que a Máquina de Boltzmann atribui aos vetores de treinamento. Também é equivalente a maximizar a probabilidade de obtermos exatamente os N casos de treinamento se fizéssemos o seguinte: 1) Deixar a rede se estabelecer em sua distribuição estacionária no tempo N diferente, sem entrada externa e 2) Mudar o vetor visível uma vez em cada passada.

Um procedimento eficiente de aprendizado de mini-lote foi proposto para as Máquinas de Boltzmann por Salakhutdinov e Hinton em 2012.



Em uma Máquina de Boltzmann geral, as atualizações estocásticas de unidades precisam ser sequenciais. Existe uma arquitetura especial que permite alternar atualizações paralelas que são muito mais eficientes (sem conexões dentro de uma camada, sem conexões de camada ignorada). Este procedimento de mini-lote torna as atualizações da Máquina de Boltzmann mais paralelas. Isso é chamado de Deep Boltzmann Machines (DBM), uma Máquina de Boltzmann geral, mas com muitas conexões ausentes.

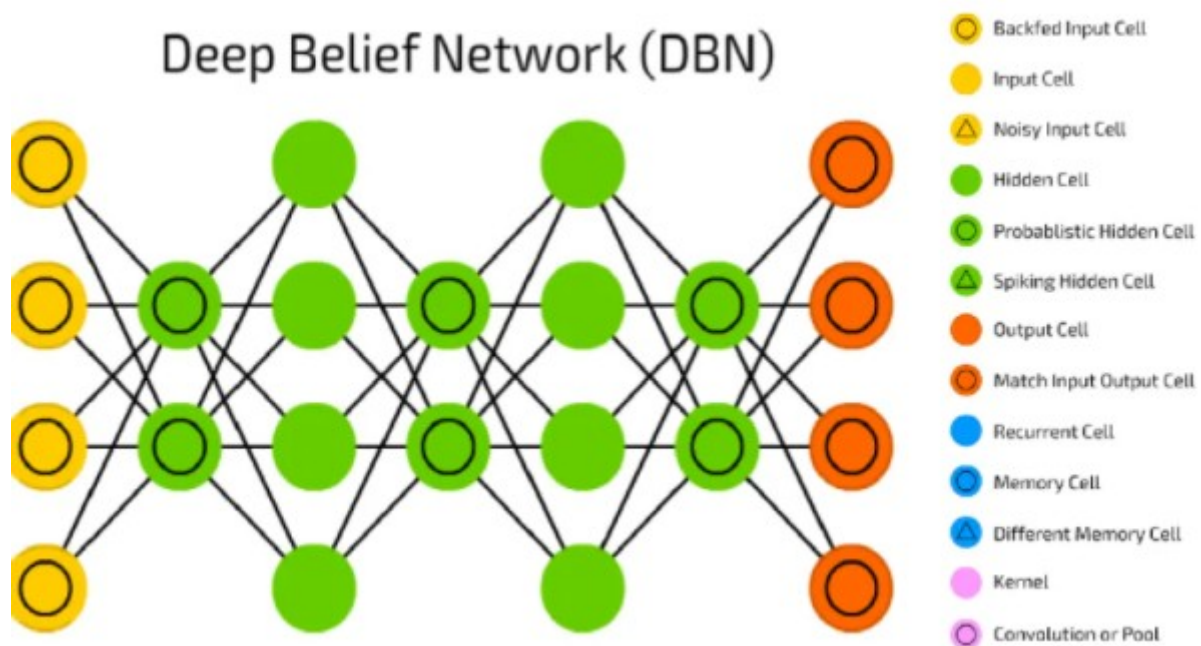
Em 2014, Salakhutdinov e Hinton apresentaram outra atualização para seu modelo, chamando-o de Máquinas Boltzmann Restritas. Elas restringem a conectividade para facilitar a inferência e a aprendizagem (apenas uma camada de unidades escondidas e sem conexões entre unidades ocultas). Em um RBM, é preciso apenas um passo para alcançar o equilíbrio.

7- Deep Belief Network

O backpropagation é considerado o método padrão em redes neurais artificiais para calcular a contribuição de erro de cada neurônio após processar um lote de dados (teremos um capítulo inteiro sobre isso). No entanto, existem alguns problemas importantes no backpropagation. Em primeiro lugar, requer dados de treinamento rotulados; enquanto quase todos os dados estão sem rótulos. Em segundo lugar, o tempo de aprendizagem não escala bem, o que significa que é muito lento em redes com múltiplas camadas ocultas. Em terceiro lugar, pode ficar preso em um “local optima”. Portanto, para redes profundas, o backpropagation está longe de ser ótimo.

Para superar as limitações do backpropagation, os pesquisadores consideraram o uso de abordagens de aprendizado sem supervisão. Isso ajuda a manter a eficiência e a simplicidade de usar um método de gradiente para ajustar os pesos, mas também usá-lo para modelar a estrutura da entrada sensorial. Em particular, eles ajustam os pesos para maximizar a probabilidade de um modelo gerador ter gerado a entrada sensorial. A questão é que tipo de modelo generativo devemos

aprender? Pode ser um modelo baseado em energia como uma Máquina de Boltzmann? Ou um modelo causal feito de neurônios? Ou um híbrido dos dois?



Uma Deep Belief Network pode ser definida como uma pilha de Máquinas de Boltzmann Restritas (RBM – Restricted Boltzmann Machines), em que cada camada RBM se comunica com as camadas anterior e posterior. Os nós de qualquer camada única não se comunicam lateralmente.

Esta pilha de RBMs pode terminar com uma camada Softmax para criar um classificador, ou simplesmente pode ajudar a agrupar dados não gravados em um cenário de aprendizado sem supervisão.

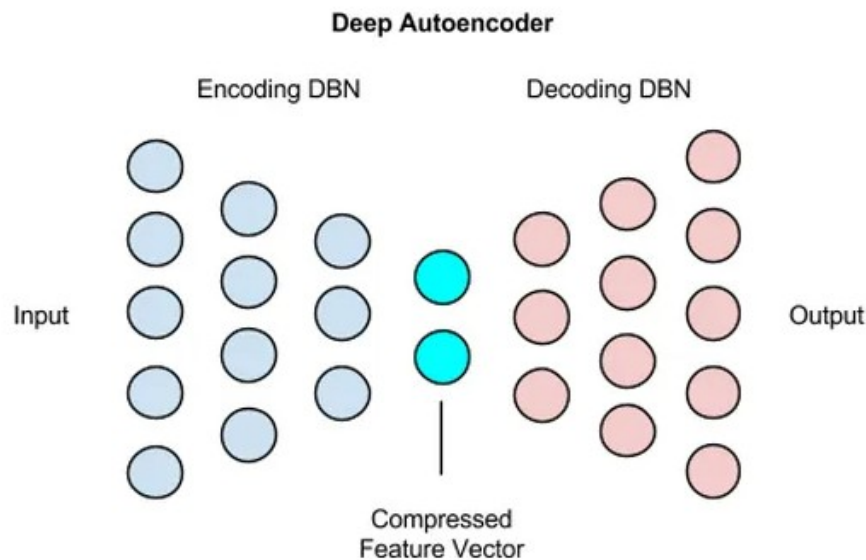
Com a exceção das camadas inicial e final, cada camada em uma Deep Belief Network tem uma função dupla: ela serve como a camada oculta para os nós que vem antes, e como a camada de entrada (ou “visível”) para a nós que vem depois. É uma rede construída de redes de camada única.

As Deep Belief Networks são usadas para reconhecer, agrupar e gerar imagens, sequências de vídeos e dados de captura de movimento. Outra aplicação das Deep Belief Networks é no Processamento de Linguagem Natural. Esse tipo de rede foi apresentado por Geoff Hinton e seus alunos em 2006.

8- Deep Auto-Encoders

Um Deep Auto-Encoder é composto por duas redes simétricas Deep Belief que tipicamente têm quatro ou cinco camadas rasas que representam a metade da codificação (encoder) da rede e o segundo conjunto de quatro ou cinco camadas que compõem a metade da decodificação (decoder).

As camadas são Máquinas de Boltzmann Restritas, os blocos de construção das Deep Belief Networks, com várias peculiaridades que discutiremos abaixo. Aqui está um esquema simplificado da estrutura de um Deep Auto-Encoder:



Os Deep Auto-Encoders são uma maneira muito agradável de reduzir a dimensionalidade não linear devido a alguns motivos: eles fornecem mapeamentos flexíveis em ambos os sentidos. O tempo de aprendizagem é linear (ou melhor) no número de casos de treinamento. E o modelo de codificação final é bastante compacto e rápido. No entanto, pode ser muito difícil otimizar Deep Auto-Encoders usando backpropagation. Com pequenos pesos iniciais, o gradiente do backpropagation morre. Mas temos maneiras de otimizá-las, usando o pré-treinamento camada-por-camada sem supervisão ou apenas inicializando os pesos com cuidado.

Os Deep Auto-Encoders são úteis na modelagem de tópicos ou modelagem estatística de tópicos abstratos que são distribuídos em uma coleção de documentos. Isso, por sua vez, é um passo importante em sistemas de perguntas e respostas como o IBM Watson.

Em resumo, cada documento em uma coleção é convertido em um Bag-of-Words (ou seja, um conjunto de contagens de palavras) e essas contagens de palavras são dimensionadas para decimais entre 0 e 1, o que pode ser pensado como a probabilidade de uma palavra ocorrer no documento.

As contagens de palavras em escala são então alimentadas em uma Deep Belief Network, uma pilha de Máquinas de Boltzmann Restritas, que elas mesmas são apenas um subconjunto de Autoencoders. Essas Deep Belief Networks, ou DBNs, comprimem cada documento para um conjunto de 10 números através de uma série de transformações sigmóides que o mapeiam no espaço de recursos.

O conjunto de números de cada documento, ou vetor, é então introduzido no mesmo espaço vetorial, e sua distância de qualquer outro vetor de documento medido. Em termos aproximados, os vetores de documentos próximos se enquadram no mesmo tópico. Por exemplo, um documento poderia ser a “pergunta” e outros poderiam ser as “respostas”, uma combinação que o software faria usando medidas de espaço vetorial.

Em resumo, existem agora muitas maneiras diferentes de fazer pré-treinamento camada-por-camada de recursos. Para conjuntos de dados que não possuem um grande número de casos rotulados, o pré-treinamento ajuda a aprendizagem discriminativa subsequente. Para conjuntos de dados muito grandes e rotulados, não é necessário inicializar os pesos utilizados na aprendizagem supervisionada

usando pré-treinamento não supervisionado, mesmo para redes profundas. O pré-treinamento foi o primeiro bom caminho para inicializar os pesos para redes profundas, mas agora existem outras formas. Mas se construímos redes muito maiores, precisaremos de pré-treinamento novamente!

9- Generative Adversarial Network

As Generative Adversarial Networks (GANs) são arquiteturas de redes neurais profundas compostas por duas redes, colocando uma contra a outra (daí o nome, “adversária”).

Os GANs foram introduzidos em um artigo de Ian Goodfellow e outros pesquisadores da Universidade de Montreal no Canadá, incluindo Yoshua Bengio, em 2014. Referindo-se aos GANs, o diretor de pesquisa de IA do Facebook, Yann LeCun, chamou de treinamento adversário “a ideia mais interessante nos últimos 10 anos em Machine Learning”.

O potencial de GANs é enorme, porque eles podem aprender a imitar qualquer distribuição de dados. Ou seja, os GANs podem ser ensinados a criar mundos estranhamente semelhantes aos nossos em qualquer domínio: imagens, música, fala, prosa. Eles são artistas robôs em um sentido, e sua produção é impressionante – até mesmo pungente.

Para entender os GANs, você deve saber como os algoritmos geradores funcionam, e para isso, contrastá-los com algoritmos discriminatórios é útil. Os algoritmos discriminatórios tentam classificar dados de entrada; isto é, dados os recursos de uma instância de dados, eles predizem um rótulo ou categoria a que esses dados pertencem.

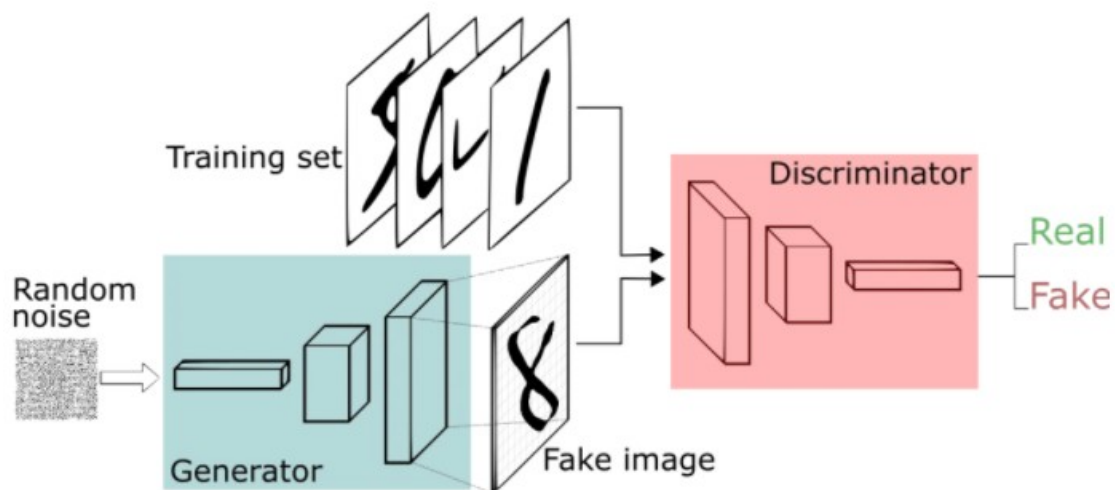
Por exemplo, tendo em conta todas as palavras em um e-mail, um algoritmo discriminatório pode prever se a mensagem é spam ou not_spam. O spam é um dos rótulos, e o saco de palavras (Bag of Words) coletadas do e-mail são os recursos que constituem os dados de entrada. Quando este problema é expresso matematicamente, o rótulo é chamado y e os recursos são chamados de x . A formulação $p(y | x)$ é usada para significar “a probabilidade de y dado x ”, que neste caso seria traduzido para “a probabilidade de um email ser spam com as palavras que contém”.

Portanto, algoritmos discriminatórios mapeiam recursos para rótulos. Eles estão preocupados apenas com essa correlação. Uma maneira de pensar sobre algoritmos generativos é que eles fazem o contrário. Em vez de prever um rótulo com determinados recursos, eles tentam prever os recursos com um determinado rótulo.

A questão que um algoritmo gerador tenta responder é: assumir que este e-mail é spam, qual a probabilidade dos recursos? Enquanto os modelos discriminativos se preocupam com a relação entre y e x , os modelos generativos se preocupam com “como você obtém x ”. Eles permitem que você capture $p(x | y)$, a probabilidade de x dado y , ou a probabilidade de características oferecidas em uma classe. (Dito isto, os algoritmos geradores também podem ser usados como classificadores, embora eles podem fazer mais do que categorizar dados de entrada.)

Outra maneira de pensar sobre isso é distinguir discriminativo de gerador assim:

- Modelos discriminativos aprendem o limite entre as classes
- Modelos generativos modelam a distribuição de classes individuais



Uma rede neural, chamada de gerador, gera novas instâncias de dados, enquanto a outra, o discriminador, as avalia por autenticidade; ou seja, o discriminador decide se cada instância de dados que revisa pertence ao conjunto de dados de treinamento real ou não.

Digamos que estamos tentando fazer algo mais banal do que imitar a Mona Lisa. Vamos gerar números escritos à mão como os encontrados no conjunto de dados MNIST, que é retirado do mundo real. O objetivo do discriminador, quando mostrado uma instância do verdadeiro conjunto de dados MNIST, é reconhecê-los como autênticos.

Enquanto isso, o gerador está criando novas imagens que passa para o discriminador. Isso acontece com a esperança de que eles, também, sejam considerados autênticos, embora sejam falsos. O objetivo do gerador é gerar dígitos escritos à mão por si mesmo. O objetivo do discriminador é identificar as imagens provenientes do gerador como falsas.

Aqui estão os passos que um GAN realiza:

- O gerador recebe números aleatórios e retorna uma imagem.
- Essa imagem gerada é alimentada no discriminador ao lado de um fluxo de imagens tiradas do conjunto de dados real.
- O discriminador assume imagens reais e falsas e retorna probabilidades, um número entre 0 e 1, com 1 representando uma previsão de autenticidade e 0 representando falsas.

Então você tem um loop de feedback duplo:

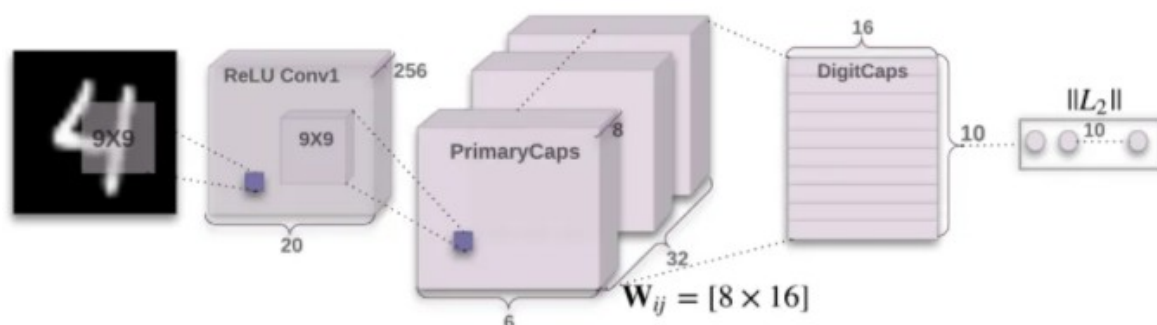
- O discriminador está em um loop de feedback com as imagens verdadeiras, que conhecemos.
- O gerador está em um loop de feedback com o discriminador.

10- Deep Neural Network Capsules

No final de 2017, Geoffrey Hinton e sua equipe publicaram dois artigos que introduziram um novo tipo de rede neural chamada Capsules. Além disso, a equipe publicou um algoritmo, denominado roteamento dinâmico entre cápsulas, que permite treinar essa rede.

Para todos na comunidade de Deep Learning, esta é uma grande notícia, e por várias razões. Em primeiro lugar, Hinton é um dos fundadores do Deep Learning e um inventor de inúmeros modelos

e algoritmos que hoje são amplamente utilizados. Em segundo lugar, esses artigos apresentam algo completamente novo, e isso é muito emocionante porque provavelmente estimulará a onda adicional de pesquisas e aplicativos muito inovadores.



As Capsules introduzem um novo bloco de construção que pode ser usado na aprendizagem profunda para modelar melhor as relações hierárquicas dentro da representação do conhecimento interno de uma rede neural. A intuição por trás deles é muito simples e elegante.

Hinton e sua equipe propuseram uma maneira de treinar essa rede composta de cápsulas e treinou-a com êxito em um conjunto de dados simples, alcançando desempenho de ponta. Isso é muito encorajador. No entanto, há desafios. As implementações atuais são muito mais lentas do que outros modelos modernos de aprendizado profundo. O tempo mostrará se as redes Capsules podem ser treinadas de forma rápida e eficiente. Além disso, precisamos ver se elas funcionam bem em conjuntos de dados mais difíceis e em diferentes domínios.

Em qualquer caso, a rede Capsule é um modelo muito interessante e já funcionando, que definitivamente se desenvolverá ao longo do tempo e contribuirá para uma maior expansão de aplicações de aprendizagem profunda.

Incluímos as Capsules entre as 10 principais arquiteturas de redes neurais, pois elas representam a inovação e o avanço na incrível e vibrante área de Deep Learning e sistemas de Inteligência Artificial. Profissionais que realmente desejem abraçar IA como carreira, devem estar atentos aos movimentos e inovações na área.

Esta não é uma lista definitiva de arquiteturas e existem outras, tais como Word2Vec, Doc2vec, Neural Embeddings e variações das arquiteturas aqui apresentadas, como Denoising Autoencoders, Variational Autoencoders, além de outras categorias como Deep Reinforcement Learning. Exatamente para auxiliar aqueles que buscam conhecimento de ponta 100% em português e 100% online, que nós criamos a Formação Inteligência Artificial, o único programa do Brasil completo, com todas as ferramentas que o aluno precisa para aprender a trabalhar com IA de forma eficiente. O aluno aprende programação paralela em GPU, Deep Learning e seus frameworks, estuda as principais arquiteturas com aplicações práticas e desenvolve aplicações de Visão Computacional e Processamento de Linguagem Natural.

Redes Neurais Profundas e o Córtex Visual

Uma das motivações para a pesquisa em redes neurais artificiais compostas de muitas camadas veio da neurociência, mais especificamente do estudo da parte do cérebro denominada córtex visual.

Sabe-se que, quando algum estímulo visual chega à retina, o sinal correspondente percorre uma sequência de regiões do cérebro, e que cada uma dessas regiões é responsável por identificar características específicas na imagem correspondente ao estímulo visual. Em particular, neurônios das regiões iniciais são responsáveis por detectar formas geométricas simples na imagem, tais como cantos e bordas, enquanto que neurônios nas regiões finais têm a atribuição de detectar formas gráficas mais complexas, compostas das formas gráficas simples detectadas por regiões anteriores. Dessa forma, cada região de neurônios (que é análoga ao conceito de camada em redes neurais artificiais) combina padrões detectados pela região imediatamente anterior para formar características mais complexas. Esse processo se repete através das regiões até que os neurônios da região final têm a atribuição de detectar características de mais alto nível de abstração, tais como faces ou objetos específicos.

Há outra motivação, dessa vez teórica, para o estudo de redes neurais profundas. Diversas pesquisas demonstram que é possível representar qualquer função lógica por meio de um circuito lógico de uma única camada, embora essa representação possa requerer uma quantidade de unidades na camada intermediária que cresce exponencialmente com o tamanho da entrada. Eles também demonstraram que, ao permitir que o circuito cresça em quantidade de camadas intermediárias, essas mesmas funções lógicas problemáticas poderiam ser representadas por uma quantidade de unidades ocultas que cresce polinomialmente com o tamanho da entrada.

Momentum e Learning Rate Decay

A forma mais comum para se treinar uma rede neural profunda é conhecida como Stochastic Gradient Descent. Esta abordagem não é utilizada apenas em redes neurais, e na verdade é bastante comum em casos em que existem grandes quantidades de dados nos quais métodos mais tradicionais podem demandar muito tempo para execução.

Tanto no GD quanto no SGD, você atualiza um conjunto de parâmetros de forma iterativa para minimizar uma função de erro. Enquanto com GD, você tem que percorrer TODAS as amostras em seu conjunto de treinamento para fazer uma única atualização para um parâmetro em uma iteração particular, com SGD, por outro lado, você usa SOMENTE uma amostra de seu conjunto de treinamento para fazer a atualização para um parâmetro em uma iteração específica.

Os pesos da rede neural podem ser atualizados conforme os dados são processados e os erros são calculados (abordagem conhecida como online learning) ou ao final do processo (abordagem conhecida como batch learning). Esta segunda abordagem é geralmente mais estável. Para otimizar o processamento da rede (muitas vezes devido a grande quantidade de dados) o tamanho do batch que possui os dados para atualização dos pesos é reduzido.

O principal parâmetro que controla a atualização dos pesos é conhecido como learning rate (taxa de aprendizado). Geralmente se utiliza-se taxas pequenas, como 0.1 ou 0.01 para este parâmetro. Dois parâmetros adicionais ainda podem ser usados na equação responsável pela atualização dos pesos:

- **Momentum:** Incorpora as propriedades da atualização de pesos anterior e faz com que os pesos continuem sendo atualizados na mesma direção, mesmo quando o erro diminui.
- **Learning Rate Decay:** Learning rate decay é usado para diminuir o valor da learning rate conforme os erros diminuem.

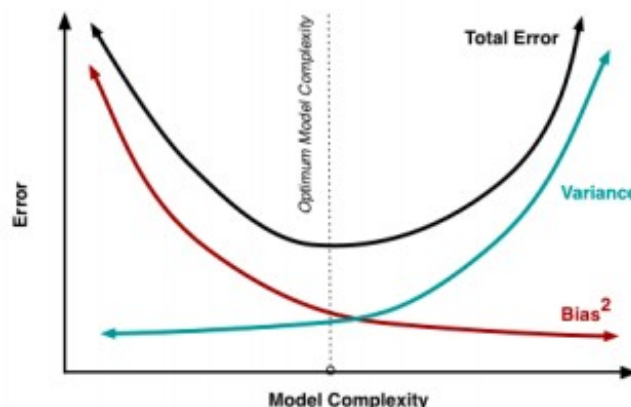
Regularização e Dropout

O objetivo do treinamento em redes neurais artificiais é obter uma rede que produza poucos erros no conjunto de treinamento, mas que também responda apropriadamente para novos padrões de entrada. A regularização é um método que busca melhorar a capacidade de generalização dos algoritmos de aprendizado, por meio de alguma restrição durante a fase de treinamento. A regularização ajuda a evitar o overfitting e melhora a generalização do modelo.

Em Deep Learning, é muito comum o modelo aprender demais sobre as peculiaridades nos dados, durante o treinamento e depois ter um baixo desempenho quando apresentado a novos dados. A regularização pode ser um remédio para este problema.

Regularização é qualquer modificação que nós fazemos no algoritmo de aprendizagem, com o objetivo de reduzir o erro de generalização, a fim de evitar o overfitting.

Existem muitas estratégias de regularização. Algumas colocam restrições extras em um modelo de aprendizado de máquina, como adicionar restrições nos valores dos parâmetros. Algumas acrescentam termos extras na função objetivo que podem ser considerados como correspondentes a uma restrição suave nos valores dos parâmetros. Se escolhidas cuidadosamente, essas restrições e penalidades extras podem levar a um melhor desempenho no conjunto de dados de teste. Às vezes, essas restrições e penalidades são projetadas para codificar tipos específicos de conhecimento prévio. Outras vezes, essas restrições e penalidades são projetadas para expressar uma preferência genérica por uma classe de modelo mais simples, a fim de promover a generalização.



No contexto de Deep Learning, a regularização de um estimador funciona através do aumento do viés para uma variância reduzida. Um regularizador efetivo é aquele que reduz significativamente a variância, sem aumentar excessivamente o viés (nós discutimos sobre viés e variância nos primeiros capítulos aqui do curso).

Na prática, um espaço de hipóteses excessivamente complexo não inclui necessariamente a função alvo ou o verdadeiro processo de geração de dados, ou mesmo uma aproximação de ambos. Nós quase nunca temos acesso ao verdadeiro processo de geração de dados, e nunca podemos saber com certeza se a família de modelos que está sendo estimada inclui o processo gerador ou não.

Os algoritmos de Deep Learning são tipicamente aplicados a domínios extremamente complicados, como imagens e sequências de áudio e texto, para os quais o verdadeiro processo de geração envolve essencialmente a simulação de todo o universo de dados. O que isto significa é que

controlar a complexidade do modelo não é uma questão simples de encontrar o modelo do tamanho correto, com o número correto de parâmetros. Ao invés disso, podemos encontrar - e de fato em cenários práticos de aprendizagem profunda, quase sempre achamos - que o melhor modelo (no sentido de minimizar o erro de generalização) é um grande modelo que foi regularizado.

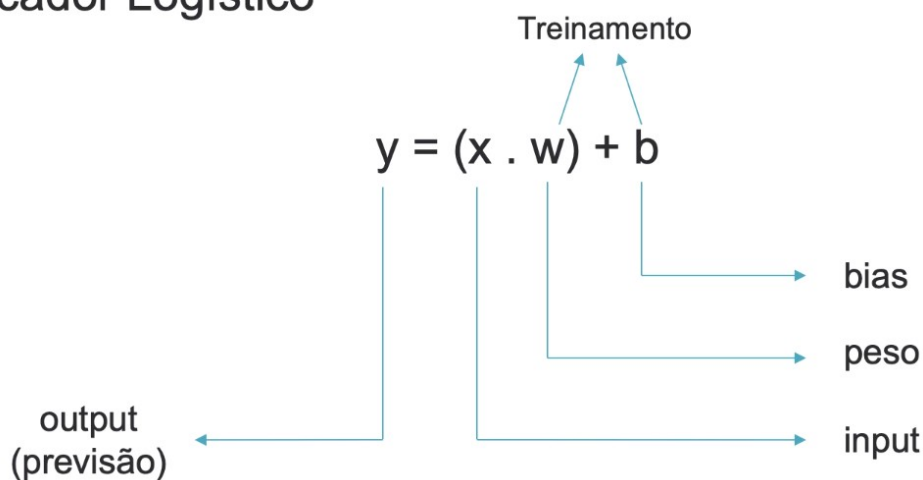
Portanto, seu objetivo em Deep Learning é encontrar um modelo que seja grande e profundo o suficiente para representar a complexidade nos dados e que possa ser aplicado a novos conjuntos de dados, com um bom desempenho. A regularização é uma das formas usadas para se alcançar esse objetivo.

O dropout é uma técnica de regularização criada por Geoffrey Hinton em 2012, portanto bem recente, que ajuda a mudar a saída dos neurônios de uma rede neural profunda, e que pode ser aplicado em qualquer camada das redes neurais profundas. O dropout desativa alguns dos neurônios da camada associada com alguma probabilidade p . Desativar um neurônio significa mudar o valor de saída para 0. No final, os neurônios que sofreram dropout tem os parâmetros reajustados, multiplicados por p (que é a probabilidade). O efeito de usar esse algoritmo é similar ao de fazer uma média de todos os possíveis modelos da rede neural que usam um subconjunto dos parâmetros disponíveis na camada afetada

Por que Usamos a Função Softmax?

Classificador Logístico:

Classificador Logístico



Ao final de um modelo de deep learning o que se tem é a previsão com os scores (as pontuações). Assim, a função softmax converte os scores em probabilidades correspondentes.

```
import numpy as np

scores = [3.0, 1.0, 0.2]

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis = 0)

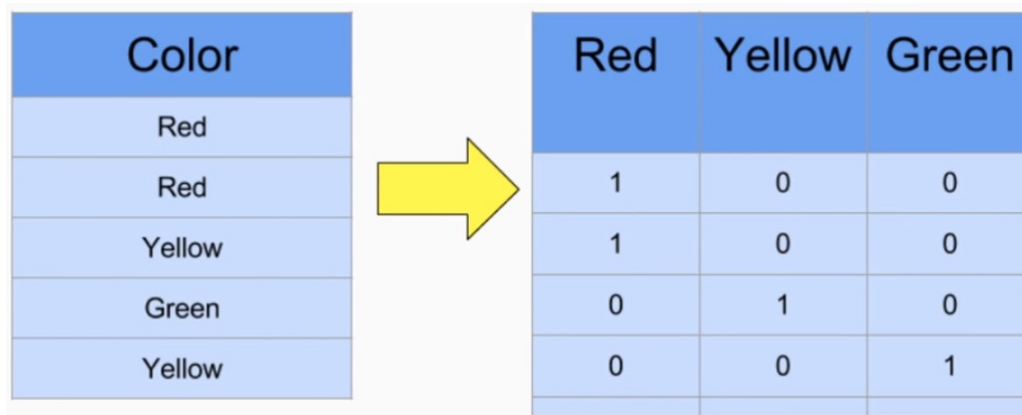
print(softmax(scores))

[ 0.8360188  0.11314284  0.05083836]
```

O que é Hot Encoding?

Hot Encoding é uma técnica de transformação de dados, que você pode aplicar durante a fase de pré-processamento, para transformar uma variável, ou então ao final da arquitetura de um modelo deep learning.

Não é um componente da arquitetura, mas é uma técnica de transformação de dados complementar ao conceito da função softmax.



Color		Red	Yellow	Green
Red		1	0	0
Red		1	0	0
Yellow		0	1	0
Green		0	0	1
Yellow				

Mini-Projeto 6 – Classificação de Imagens com Deep Learning e PyTorch e Torchvision em GPU

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 14</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
## Mini-Projeto 6
## Classificação de Imagens com Deep Learning e PyTorch
![title](imagens/mini-projeto6.png)
Você tem 3 opções:
- 1- Não tem GPU no seu computador? Executar este Jupyter Notebook normalmente, nesse caso com treinamento em CPU.
- 2- Tem GPU no seu computador? Executar este Jupyter Notebook normalmente, nesse caso com treinamento em GPU.
- 3- Não tem GPU no seu computador? Executar este Jupyter Notebook na nuvem, com o <a href='\"https://colab.research.google.com/\"'>Google Colab</a>.
## Definição do Problema
![title](imagens/CV.jpeg)
## Instalando e Carregando os Pacotes
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# ->
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Instala o PyTorch
!pip install -q torch
# ->
# Instala o Torchvision
!pip install -q torchvision
# ->
# Imports
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as transforms
```

```

from torchvision import datasets
from torch.utils.data.sampler import SubsetRandomSampler
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import tensorflow
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
### Verificando a GPU
http://pytorch.org/docs/stable/cuda.html
https://developer.nvidia.com/cuda-zone
Pode ser útil acelerar o tempo de treinamento usando uma GPU. CUDA é uma plataforma da Nvidia que permite usarmos as GPUs (Nvidia) para processamento paralelo). Os frameworks de Deep Learning dependem da plataforma CUDA para o processamento em GPU.
# ->
# Executar somente se a máquina tiver GPU e Plataforma CUDA instalada
# !nvidia-smi
# ->
# Verifica se a plataforma CUDA está disponível
train_on_gpu = torch.cuda.is_available()
# ->
# Mensagem para o usuário
if not train_on_gpu:
    print('Plataforma CUDA não está disponível. O treinamento será realizado com a CPU ...')
else:
    print('Plataforma CUDA está disponível! O treinamento será realizado com a GPU ...')
### Checando o Hardware Disponível no Servidor da DSA - CPU e GPUs
# ->
# Lista todos os dispositivos disponíveis
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
> Se por acaso não aparecer para você todas as GPUs, reinstale o TensorFlow:
# ->
# pip uninstall tensorflow
# pip install tensorflow
# ->
import tensorflow as tf
print("Número Disponível de GPUs: ", len(tf.config.experimental.list_physical_devices('GPU')))
# ->
# Lista o código de cada GPU
tf.config.list_physical_devices('GPU')
## Carregando o Dataset
http://pytorch.org/docs/stable/torchvision/datasets.html
O download pode demorar um minuto. Carregamos os dados de treinamento e teste, dividimos os dados de treinamento em um conjunto de treinamento e validação e, em seguida, criamos DataLoaders para cada um desses conjuntos de dados.
Dataset usado: https://www.cs.toronto.edu/~kriz/cifar.html
# ->
# Função que converte os dados em um tensor normalizado
transform = transforms.Compose([transforms.RandomHorizontalFlip(),
                                transforms.RandomRotation(10),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# ->
# Download dos dados de treino
dados_treino = datasets.CIFAR10('dados',
                                train = True,
                                download = True,
                                transform = transform)
# ->
# Download dos dados de teste
dados_teste = datasets.CIFAR10('dados',
                                train = False,
                                download = True,
                                transform = transform)
## Preparando os Data Loaders
# ->
# Dados de treino
dados_treino
# ->
# Dados de teste
dados_teste
# ->
# Número de amostras de treino
num_amstras_treino = len(dados_treino)

```

```

num_amstras_treino
# ->
# Criamos um índice e o tornamos randômico
indices = list(range(num_amstras_treino))
np.random.shuffle(indices)
# ->
# Percentual dos dados de treino que usaremos no dataset de validação
valid_size = 0.2
# ->
# Agora fazemos o split para os dados de treino e validação
split = int(np.floor(valid_size * num_amstras_treino))
idx_treino, idx_valid = indices[split:], indices[:split]
# ->
# Definimos as amostras de treino
amostras_treino = SubsetRandomSampler(idx_treino)
# ->
# Definimos as amostras de validação
amostras_valid = SubsetRandomSampler(idx_valid)
Agora preparamos os data loaders.
# ->
# Número de subprocessos para carregar os dados
num_workers = 0
# ->
# Número de amostras por batch
batch_size = 20
# ->
# Data Loader de dados de treino
loader_treino = torch.utils.data.DataLoader(dados_treino,
                                             batch_size = batch_size,
                                             sampler = amostras_treino,
                                             num_workers = num_workers)
# ->
# Data Loader de dados de validação
loader_valid = torch.utils.data.DataLoader(dados_treino,
                                           batch_size = batch_size,
                                           sampler = amostras_valid,
                                           num_workers = num_workers)
# ->
# Data Loader de dados de teste
loader_teste = torch.utils.data.DataLoader(dados_teste,
                                           batch_size = batch_size,
                                           num_workers = num_workers)
# ->
# Lista de classes das imagens
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
## Visualizando os Dados
# ->
# Função para desnormalização das imagens
def imshow(img):
    # Desfaz a normalização
    img = img / 2 + 0.5
    # Converte em tensor e imprime
    plt.imshow(np.transpose(img, (1, 2, 0)))
# ->
# Obtém um batch de dados de treino
dataiter = iter(loader_treino)
images, labels = dataiter.next()
# ->
# Converte as imagens em formato NumPy
images = images.numpy()
# ->
# Plot de um batch de imagens de treino
# Área de plotagem
fig = plt.figure(figsize = (25, 4))
# Loop e print
for idx in np.arange(20):
    # Cria os subplots
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    # Desfaz a normalização
    # images[idx]
    imshow(images[idx])
    # Coloca o título
    ax.set_title(classes[labels[idx]])
## Visualizando Uma Imagem em Mais Detalhes
Aqui, observamos os canais de cores normalizados de vermelho, verde e azul (RGB) como três imagens separadas com intensidade de tons de cinza.
# ->
# Extraímos os canais de cores
rgb_img = np.squeeze(images[3])

```

```

channels = ['Canal Vermelho (Red)', 'Canal Verde (Green)', 'Canal Azul (Blue)']
# ->
# Loop e print
# Área de plotagem
fig = plt.figure(figsize = (36, 36))
# Loop pelas imagens
for idx in np.arange(rgb_img.shape[0]):
    # Subplot
    ax = fig.add_subplot(1, 3, idx + 1)
    # Índice
    img = rgb_img[idx]
    # Mostra a imagem em escala de cinza
    ax.imshow(img, cmap = 'gray')
    # Título
    ax.set_title(channels[idx])
    # Largura e altura da imagem
    width, height = img.shape
    # Limite
    thresh = img.max()/2.5
    # Loop
    for x in range(width):
        for y in range(height):
            val = round(img[x][y],2) if img[x][y] !=0 else 0
            ax.annotate(str(val),
                        xy = (y,x),
                        horizontalalignment = 'center',
                        verticalalignment = 'center',
                        size = 8,
                        color = 'white' if img[x][y] < thresh else 'black')

## Definindo a Arquitetura da Rede
http://pytorch.org/docs/stable/nn.html
Vamos definir uma arquitetura CNN (Convolutional Neural Network).
* [Camadas convolucionais](https://pytorch.org/docs/stable/nn.html#conv2d), podem ser consideradas como uma pilha de imagens filtradas.
* [Camadas de Maxpool](https://pytorch.org/docs/stable/nn.html#maxpool2d), reduzem o tamanho x-y de uma entrada, mantendo apenas os pixels mais ativos da camada anterior.
* As camadas Linear + Dropout podem evitar sobreajuste e produzir uma saída de 10 dimensões.
# ->
# Arquitetura do Modelo
class ModeloCNN(nn.Module):
    # Método construtor
    def __init__(self):
        super(ModeloCNN, self).__init__()
        # Camada Convolucional de entrada
        self.conv1 = nn.Conv2d(3, 16, 3, padding = 1)
        # Camada Convolucional oculta
        self.conv2 = nn.Conv2d(16, 32, 3, padding = 1)
        # Camada Convolucional oculta
        self.conv3 = nn.Conv2d(32, 64, 3, padding = 1)
        # Camada de Max Pooling
        self.pool = nn.MaxPool2d(2, 2)
        # Camada Totalmente Conectada 1
        self.fc1 = nn.Linear(64 * 4 * 4, 500)
        # Camada Totalmente Conectada 2
        self.fc2 = nn.Linear(500, 10)
        # Camada de Dropout (Regularização)
        self.dropout = nn.Dropout(0.5)
    # Método Forward
    def forward(self, x):
        # Adiciona uma camada de ativação Relu para cada camada convolucional
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        # Faz o "achatamento" da matriz resultante da convolução e cria um vetor
        x = x.view(-1, 64 * 4 * 4)
        # Adiciona uma camada de dropout para regularização
        x = self.dropout(x)
        # Adiciona a 1ª camada oculta, com função de ativação relu
        x = F.relu(self.fc1(x))
        # Adiciona uma camada de dropout para regularização
        x = self.dropout(x)
        # Adiciona a 2ª camada oculta (classificação feita pelo modelo)
        x = self.fc2(x)
        return x
# ->
# Cria o modelo
modelo = ModeloCNN()
print(modelo)

```



```

# ->
# Movemos o modelo para a GPU se disponível
if train_on_gpu:
    modelo.cuda()
## Função de Perda (Loss Function)
http://pytorch.org/docs/stable/nn.html#loss-functions
# ->
# Loss function como categorical cross-entropy
criterion = nn.CrossEntropyLoss()
## Otimizador
http://pytorch.org/docs/stable/optim.html
# ->
# Hiperparâmetro
taxa_aprendizado = 0.01
# ->
# Otimizador com SGD
optimizer = optim.SGD(modelo.parameters(), lr = taxa_aprendizado)
## Treinamento
Lembre-se de observar como a perda em treinamento e validação diminui com o tempo; se a perda em validação aumentar, isso indica um possível
sobreajuste (overfitting).
# ->
# Número de épocas para treinar o modelo
num_epochs = 30
# ->
# hiperparâmetro para controlar a mudança do erro em validação
erro_valid_min = np.Inf
Treinamos o modelo (a execução desta célula pode demorar):
# ->
%%time
for epoch in range(1, num_epochs + 1):
    # Parâmetros para acompanhar o erro total em treinamento e validação
    erro_treino = 0.0
    erro_valid = 0.0
    # Inicia o treinamento do modelo
    modelo.train()
    # Loop pelos batches de dados de treino
    for batch_idx, (data, target) in enumerate(loader_treino):
        # Move os tensores para a GPU se disponível
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()
        # Limpa os gradientes de todas as variáveis otimizadas
        optimizer.zero_grad()
        # Forward: calcula as saídas previstas
        output = modelo(data)
        # Calcula o erro no batch
        loss = criterion(output, target)
        # Backward: calcula o gradiente da perda em relação aos parâmetros do modelo
        loss.backward()
        # Realiza uma única etapa de otimização (atualização dos parâmetros)
        optimizer.step()
        # Atualiza o erro total em treino
        erro_treino += loss.item() * data.size(0)
    # Inicia a validação do modelo
    modelo.eval()
    # Loop pelos batches de dados de validação
    for batch_idx, (data, target) in enumerate(loader_valid):
        # Move os tensores para a GPU se disponível
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()
        # Forward: calcula as saídas previstas
        output = modelo(data)
        # Calcula o erro no batch
        loss = criterion(output, target)
        # Atualiza o erro total de validação
        erro_valid += loss.item() * data.size(0)
    # Calcula o erro médio
    erro_treino = erro_treino / len(loader_treino.dataset)
    erro_valid = erro_valid / len(loader_valid.dataset)
    # Print
    print("\nEpoch: {} \tErro em Treinamento: {:.6f} \tErro em Validação: {:.6f}'.format(epoch,
                                                                                          erro_treino,
                                                                                          erro_valid))
    # Salva o modelo sempre que a perda em validação diminuir
    if erro_valid <= erro_valid_min:
        print('Erro em Validação foi Reduzido ({:.6f} --> {:.6f}). Salvando o modelo...'.format(erro_valid_min,
                                                                                          erro_valid))
        torch.save(modelo.state_dict(), 'modelos/modelo_final.pt')
    erro_valid_min = erro_valid

```

```

### Carrega o Modelo Final
# ->
# Carrega o modelo
modelo.load_state_dict(torch.load('modelos/modelo_final.pt'))
## Testando e Avaliando o Modelo Final
Testamos o modelo treinado em dados nunca vistos anteriormente! Um resultado "bom" será uma CNN que obtenha cerca de 70% (ou mais, tente o seu melhor!) de precisão nas imagens de teste.
# ->
# Erro em teste
erro_teste = 0.0
# ->
# Controle de acertos do modelo
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
# ->
# Inicia a avaliação do modelo
modelo.eval()
# Loop pelos batches de dados de teste
for batch_idx, (data, target) in enumerate(loader_teste):
    # Move os tensores para GPU se disponível
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    # Forward
    output = modelo(data)
    # Calcula o erro
    loss = criterion(output, target)
    # Atualiza o erro em teste
    erro_teste += loss.item() * data.size(0)
    # Converte probabilidades de saída em classe prevista
    _, pred = torch.max(output, 1)
    # Compara as previsões com o rótulo verdadeiro
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    # Calcula a precisão do teste para cada classe
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1
# Erro médio em teste
erro_teste = erro_teste / len(loader_teste.dataset)
print("\nErro em Teste: {:.6f}\n".format(erro_teste))
# Calcula a acurácia para cada classe
for i in range(10):
    if class_total[i] > 0:
        print('Acurácia em Teste da classe %5s: %2d%% (%2d/%2d)' % (classes[i],
                                                                    100 * class_correct[i] / class_total[i],
                                                                    np.sum(class_correct[i]),
                                                                    np.sum(class_total[i])))
    else:
        print('Acurácia em Teste de %5s:' % (classes[i]))
# Calcula a acurácia total
print("\nAcurácia em Teste (Total): %2d%% (%2d/%2d)' % (100. * np.sum(class_correct) / np.sum(class_total),
                                                         np.sum(class_correct),
                                                         np.sum(class_total)))

## Previsões com o Modelo Treinado
# ->
# Obtém um batch de dados de teste
dataiter = iter(loader_teste)
images, labels = dataiter.next()
images.numpy()
# ->
# Move as imagens para a GPU se disponível
if train_on_gpu:
    images = images.cuda()
# ->
# Faz as previsões com o modelo treinado
output = modelo(images)
# ->
# Converte probabilidades de saída em classe prevista
_, preds_tensor = torch.max(output, 1)
preds = np.squeeze(preds_tensor.numpy()) if not train_on_gpu else np.squeeze(preds_tensor.cpu().numpy())
# ->
# Plot das previsões
fig = plt.figure(figsize = (25, 4))
print("\nEntre parênteses a classe real. Vermelho indica erro do modelo.\n")
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    imshow(images[idx].cpu())

```

```
ax.set_title("{} {}".format(classes[preds[idx]], classes[labels[idx]]),
            color = ("green" if preds[idx] == labels[idx].item() else "red"))
Deep Learning é o estado da arte em sistemas de Inteligência Artificial nos dias de hoje!
# Fim
```

Estudo de Caso – Deep Learning em R Para Classificação de Imagens

O objetivo deste Estudo de Caso é reproduzir o Mini-Projeto 6 para classificação de imagens. Lá trabalhamos em Python. Agora trabalharemos em R.

O script está sendo fornecido a você, mas o dataset deverá ser baixado no endereço abaixo:

<https://www.cs.toronto.edu/~kriz/cifar.html>

Baixe o arquivo: CIFAR-10 binary version (suitable for C programs)

Descompacte o arquivo e coloque na mesma pasta do script.

```
# Estudo de Caso - Deep Learning em R Para Classificação de Imagens

# O objetivo deste Estudo de Caso é reproduzir o Mini-Projeto 6 para classificação de imagens.
# Lá trabalhamos em Python. Agora trabalharemos em R.

# Esse exercício vai ajudar você a consolidar o conhecimento sobre Deep Learning, ao mesmo tempo que
# aprimora suas habilidades em R.

# Leia todos os comentários e execute todas as linhas. Os pacotes precisam ser instalados apenas uma vez!

# Os detalhes do Estudo de Caso (incluindo a fonte de dados) estão no manual em pdf.

# Diretório de trabalho
getwd()
setwd("~/Dropbox/DSA/MachineLearning2.0/Cap14/R/EstudoCaso")

# Pacotes reticulate e devtools serão usados para instalar Keras e TensorFlow
install.packages("reticulate")
library(reticulate)
install.packages("devtools")
library(devtools)

# Vamos instalar o Keras para criação do modelo CNN
devtools::install_github("rstudio/keras")
force = TRUE
library(keras)

# O Keras depende do TensorFlow, que também deve ser instalado
install.packages("tensorflow")
library(tensorflow)
install_tensorflow()

# Carregando a lista de labels disponíveis no dataset
labels <- read.table("cifar-10-batches-bin/batches.meta.txt")

# Criando listas para imagens e labels
images.rgb <- list()
images.lab <- list()

# Vamos extrair 10000 imagens de cada arquivo binário
num.images = 10000

# Loop por todos arquivos binários para carregar as listas de imagens e labels
for (f in 1:5) {

  # Loop por cada arquivo
  arquivo <- file(paste("cifar-10-batches-bin/data_batch_", f, ".bin", sep = ""), "rb")
  for(i in 1:num.images) {

    # Lê o arquivo
    l <- readBin(arquivo, integer(), size = 1, n = 1, endian = "big")

    # Carrega os pixels para os 3 canais de cores: red, green e blue
    r <- as.integer(readBin(arquivo, raw(), size = 1, n = 1024, endian = "big"))
```

```

g <- as.integer(readBin(arquivo, raw(), size = 1, n = 1024, endian = "big"))
b <- as.integer(readBin(arquivo, raw(), size = 1, n = 1024, endian = "big"))

# Índice
index <- num.images * (f-1) + i

# Listas
images.rgb[[index]] = data.frame(r, g, b)
images.lab[[index]] = l+1
}

# Fechamos o arquivo e removemos os objetos temporários para liberar memória
close(arquivo)
remove(l,r,g,b,f,i, index, arquivo)
}

# Função para imprimir uma imagem
# Aqui organizamos os pixels para mostrar a imagem como um plot
drawImage <- function(index) {
  img <- images.rgb[[index]]
  img.r.mat <- matrix(img$r, ncol = 32, byrow = TRUE)
  img.g.mat <- matrix(img$g, ncol = 32, byrow = TRUE)
  img.b.mat <- matrix(img$b, ncol = 32, byrow = TRUE)
  img.col.mat <- rgb(img.r.mat, img.g.mat, img.b.mat, maxColorValue = 255)
  dim(img.col.mat) <- dim(img.r.mat)

  # Cria o grid
  library(grid)
  grid.raster(img.col.mat, interpolate=FALSE)

  # Remove objetos temporários
  remove(img, img.r.mat, img.g.mat, img.b.mat, img.col.mat)

  # Define labels
  labels[[1]][images.lab[[index]]]
}

# Executa a função
drawImage(sample(1:(num.images * 5), size = 1))

# Vamos ler novamente a lista de classes, mas agora usaremos para as imagens de teste
labels1 <- read.table("cifar-10-batches-bin/batches.meta.txt")

# Listas de imagens e labels
images.rgb1.test <- list()
images.lab.test <- list()

# Temos 10 mil imagens de teste
num.images1 = 10000

# Carrega o arquivo com as imagens de teste
arquivo_teste <- file(paste("cifar-10-batches-bin/test_batch", ".bin", sep = ""), "rb")

# Loop para carregar as listas de imagens e labels de teste (igual ao que fizemos em treino)
for(i in 1:num.images1) {
  l <- readBin(arquivo_teste, integer(), size = 1, n = 1, endian = "big")
  r <- as.integer(readBin(arquivo_teste, raw(), size = 1, n = 1024, endian = "big"))
  g <- as.integer(readBin(arquivo_teste, raw(), size = 1, n = 1024, endian = "big"))
  b <- as.integer(readBin(arquivo_teste, raw(), size = 1, n = 1024, endian = "big"))
  images.rgb1.test[[i]] <- data.frame(r,g,b)
  images.lab.test[[i]] <- l+1
}

# Vamos encerrar o objeto para liberar memória do computador
close(arquivo_teste)
remove(l,r,g,b,i, arquivo_teste)

# Dados de treino e teste
dados_treino <- images.rgb
dados_teste <- images.rgb1.test

# Criamos agora um array de 32x32x3 (altura x largura x canais de cores)
# O array terá 50.000 imagens que serão usadas para treinar o modelo CNN

# Shape da matriz de treino com 4 dimensões
matriz_treino <- array(dim = c(length(dados_treino), 32, 32, 3))
dim(matriz_treino)

```

```

# Loop para preparar a matriz de dados de treino
for (i in 1:length(dados_treino)){
  d_1 <- matrix(dados_treino[[i]][,1], nrow = 32, ncol = 32, byrow = TRUE)
  for (j in 1:32){
    matriz_treino[i,j,,1] <- as.numeric(unlist(d_1[j,]))
  }
  d_2 <- matrix(dados_treino[[i]][,2], nrow = 32, ncol = 32, byrow = TRUE)
  for (j in 1:32){
    matriz_treino[i,j,,2] <- as.numeric(unlist(d_2[j,]))
  }
  d_3 <- matrix(dados_treino[[i]][,3], nrow = 32, ncol = 32, byrow = TRUE)
  for (j in 1:32){
    matriz_treino[i,j,,3] <- as.numeric(unlist(d_3[j,]))
  }
  rm(d_1,d_2,d_3)
}

```

```

# Shape da matriz de teste com 4 dimensões
matriz_teste <- array(dim = c(length(dados_teste), 32, 32, 3))
dim(matriz_teste)

```

```

# Loop para carregar a matriz com dados de teste
for (i in 1:length(dados_teste)){
  d_1 <- matrix(dados_teste[[i]][,1], nrow = 32, ncol = 32, byrow = TRUE)
  for (j in 1:32){
    matriz_teste[i,j,,1] <- as.numeric(unlist(d_1[j,]))
  }
  d_2 <- matrix(dados_teste[[i]][,2], nrow = 32, ncol = 32, byrow = TRUE)
  for (j in 1:32){
    matriz_teste[i,j,,2] <- as.numeric(unlist(d_2[j,]))
  }
  d_3 <- matrix(dados_teste[[i]][,3], nrow = 32, ncol = 32, byrow = TRUE)
  for (j in 1:32){
    matriz_teste[i,j,,3] <- as.numeric(unlist(d_3[j,]))
  }
  rm(d_1,d_2,d_3)
}

```

```

# Agora separamos imagens e labels em treino e teste
set.seed(02122020)

```

```

# Imagens de treino e teste
x_treino <- matriz_treino[1:50000,,]
x_teste <- matriz_teste[1:10000,,]

```

```

# Labels de treino e teste
y_treino <- array(unlist(images.lab[1:50000]), dim = c(50000,1))
y_teste <- array(unlist(images.lab.test[1:10000]), dim = c(10000,1))

```

```

# Os labels estão no intervalo de 0 a 9, então estamos subtrair 1
# Se não for feito, pode haver um erro como índice fora do limite
y_treino1 <- y_treino - 1
y_teste1 <- y_teste - 1

```

```

# One Hot Encoding para formatar os labels
y_treino2 <- to_categorical(y_treino1, num_classes = 10)
y_teste2 <- to_categorical(y_teste1, num_classes = 10)

```

```

# Criação do Modelo CNN
modelo_cnn <- keras_model_sequential()

```

```

# Arquitetura do Modelo CNN
modelo_cnn %>%
  layer_conv_2d(filter = 32,
    kernel_size = c(3,3),
    padding = "same",
    input_shape = c(32,32,3),
    activation = "relu") %>%
  layer_conv_2d(filter = 32,
    kernel_size = c(3,3),
    activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(0.25) %>%
  layer_conv_2d(filter = 32,
    kernel_size = c(3,3),
    padding = "same",
    activation = "relu") %>%
  layer_conv_2d(filter = 32, kernel_size = c(3,3),

```

```

        activation = "relu" ) %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_dropout(0.25) %>%
layer_flatten() %>%
layer_dense(units = 512,
            activation = "relu") %>%
layer_dropout(0.5) %>%
layer_dense(10) %>%
layer_activation("softmax")

# Otimizador para o modlo
opt <- optimizer_adam(lr = 0.0001, decay = 1e-6)

# Compila o modelo com a função de custo e o otimizador
modelo_cnn %>% compile(loss = "categorical_crossentropy",
                    optimizer = opt,
                    metrics = "accuracy")

# Sumário do modelo
summary(modelo_cnn)

# Treinamento do modelo (aumente ou diminua o número de epochs se desejar)
hist_cnn <- modelo_cnn %>% fit(x_treino,
                            y_treino2,
                            batch_size = 32,
                            epochs = 50,
                            validation_data = list(x_teste, y_teste2),
                            shuffle = TRUE)

# Histórico de treinamento
plot(hist_cnn)

# Avaliação do modelo
aval <- modelo_cnn %>% evaluate(x_teste, y_teste2)
aval

# Fim

```

7.15. Sistemas de Recomendação

O Que São Sistemas de Recomendação?

Para todos os negócios ligados a comércio eletrônico, é provável que os sistemas de recomendação sejam a ferramenta analítica mais importante já implementada. Embora não existam estimativas oficiais, muitas fontes estimam que, para as principais plataformas de comércio eletrônico, como a Amazon e a Netflix, os sistemas de recomendação podem ser responsáveis por até 10% a 35% da receita destas empresas. Esses são números consideráveis. Em um mundo com baixo crescimento, conseguir aumentar em 10% o seu faturamento, faz dos sistemas de recomendação algo que devemos olhar com mais atenção. Vamos compreender o que são esses sistemas.

Sempre que falamos em alguma tecnologia que pode trazer benefícios financeiros para a empresa, que sejam significativos, os investimentos nesta tecnologia se justificam. E aí vemos uma vantagem dos sistemas de recomendação (ou talvez mais uma). O custo de implementação desses sistemas é relativamente baixo e seus ganhos são consideráveis. Sistemas de recomendação são apenas uma das muitas especialidades em rápida evolução dentro de análise preditiva e trabalho garantido para os Cientistas de Dados.

O Que São Sistemas de Recomendação?

São as aplicações que personalizam a experiência de compra do cliente, recomendando as melhores opções seguintes à luz das suas recentes compras ou atividade de navegação.

Sistemas de Recomendação são, portanto, modelos preditivos que a partir de análise estatística, determinam a probabilidade de um cliente estar interessado por um outro item similar ao que está comprando em um dado momento.

Sistemas de recomendação são simples, pelo menos, em seu objetivo.

Mas é importante deixar claro, que enquanto sistemas de recomendação são usados em aplicativos de comércio eletrônico onde o cliente está comprando algo, eles são igualmente aplicáveis em situações em que você está tentando maximizar o tempo do usuário em um site com a finalidade de aumentar a exposição publicitária. Um bom exemplo é o portal do Yahoo que personaliza seu feed de notícias de modo que você gaste mais tempo no site. Este é um bom exemplo de aplicação de sistemas de recomendação. Você não sabe (ou pelo menos não sabia até agora), mas as notícias que vão sendo recomendadas a você no portal têm como objetivo aumentar sua exposição aos anunciantes do site, que pagam exatamente pelo número de visualizações ou de cliques em seus banners. Outros exemplos de sistemas de recomendação incluem associações e Market Basket Analysis que são tão úteis para atividades de Marketing.

Tipos de Sistemas de Recomendação

Sistemas de recomendação não são todos iguais. Se você for implementar um sistema de recomendação pela primeira vez é muito provável que você comece com uma solução pronta de um fornecedor ou que você desenvolva seu próprio sistema do zero usando R ou Python por exemplo e faremos isso daqui a pouco. Além das questões táticas de negócios sobre as considerações que devem ser feitas na implementação de sistemas de recomendação, é importante conhecer tecnicamente o que são esses sistemas. Isso significa olhar profundamente para os diferentes tipos de sistema de recomendação e compreender os pontos fortes e fracos de cada um.

Os tipos de sistemas de recomendação dependem de diferentes algoritmos.

- Alguns têm um foco maior nas semelhanças de clientes, alguns em similaridades de conteúdo e alguns uma mistura dos dois.
- Diferentes algoritmos variam em sua utilidade com base no quanto você sabe com antecedência sobre os clientes, seu conteúdo e se você pode obter feedback dos clientes após a compra (isso seria fundamental).
- Todos devem satisfazer a exigência geral de oferecer um resultado personalizado dentro de cerca de 50 milissegundos (esse é um desafio).

Os sistemas de recomendação se resumem a oferecer pontuações aos clientes de acordo com vários critérios. Essas tags de pontuação que você coloca nos clientes e registros de conteúdo raramente são em tempo real e são normalmente atualizadas durante a noite, mais ou menos frequentemente, dependendo de cada circunstância e objetivo de negócio. Portanto, as atualizações de pontuação off-line podem depender de cálculos recomendados, mas podem ser complementadas com pontuação e análise realizadas separadamente em uma escala de tempo mais longa. Ou seja, o que parece tempo real é na verdade quase tempo real.



Os avanços nos sistemas de recomendação não foram tão grandes quanto algumas outras áreas da ciência de dados. A mais complexa dessas opções, os Filtros Colaborativos, existem a mais de dez anos. De uma perspectiva empresarial a boa notícia é que você pode se sentir confortável construindo um portfólio de tecnologia existente e bem estabelecida. A maioria dos avanços na ciência dos dados em torno de sistemas de recomendação tem sido focada em melhorar gradualmente o seu desempenho nos domínios específicos em que foram implantados. E com a grande quantidade de dados disponíveis (Big Data), desempenho deve ser uma preocupação constante.

Vamos estudar em mais detalhes cada um dos 5 tipos de sistemas de recomendação mostrados na figura acima.

Sistema de Recomendação Baseado no Item Mais Popular

Este é o modelo mais simples e básico de sistema de recomendação.

A estratégia é simplesmente oferecer ao cliente o que é mais popular, seja um filme, um livro ou um artigo de vestuário. Sem fazer nada mais do que observar nos registros de vendas, é possível construir um sistema de recomendação simples como esse. Isso não é necessariamente ciência de dados. Não é particularmente personalizado. Mas pode ser útil se você sabe muito pouco sobre o seu visitante.

Exige alguns atributos de conteúdos que podem ser usados para criar subcategorias que podem corresponder à navegação atual do visitante em um site de comércio eletrônico. Por exemplo, se você está oferecendo uma grande variedade de produtos como um único pacote ou combo, desde ferramentas a roupas, ou filmes, livros ou notícias que apelam para interesses diferentes, então você precisará tentar combinar itens pelo menos na mesma categoria visualizada pelo seu visitante. Empresas como a rede de lojas Home Depot usa regularmente um espaço em seu website chamado de "best sellers" e a GAP usa regularmente "produtos mais recentes" para indicar os produtos mais populares, indicá-los a outros consumidores e aumentar a receita. Esse tipo de sistema de recomendação pode ser usado como uma estratégia complementar a outros sistemas mais robustos.

Associação e Modelos Market Basket

Sistemas de recomendação baseados em análise de Associação e Análise de Cesta de Mercado (Market Basket) olham quase exclusivamente em conteúdo. Este tipo de análise estatística baseia-se apenas no mais simples dos cálculos para encontrar itens que são frequentemente consumidos juntos. A análise matemática de associação e de market basket é a mesma. Quando os clientes normalmente adquirem os itens ou serviços um de cada vez (como serviços bancários) chamamos de Associação. Quando os clientes potencialmente compram várias coisas de uma só vez chamamos de Market Basket. Assim, a Análise da Associação é realizada ao nível do cliente, enquanto a Análise de Market Basket é conduzida ao nível da transação. Existem três etapas principais neste processo de análise:

1. Avaliar a força da relação entre cada um de seus produtos e todos os outros produtos que você oferece usando os algoritmos de associação.
2. Identificar aqueles pares que têm afinidade muito forte (tipicamente uma pontuação de afinidade de 2 ou superior). Por exemplo, um cliente com um cartão de crédito pode ter duas ou três vezes mais probabilidade de adquirir um empréstimo do que um cliente selecionado ao acaso.
3. Criar uma oferta personalizada para clientes que têm um produto (de um par de produtos fortemente associados), mas não o outro.

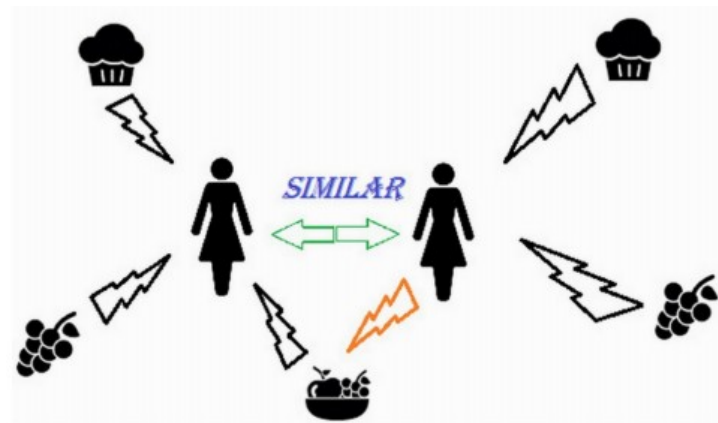
Vantagens:

- É extremamente simples e rápido.
- Funcionará com bases de clientes muito pequenas.
- O conhecimento do cliente e seu relacionamento com produtos e serviços não é necessário.
- A preparação de dados é mínima.
- Associação e Market Basket Analysis são normalmente o método mais rentável para criar ofertas personalizadas.

Filtros Colaborativos

A filtragem colaborativa foca o usuário e outros usuários que são matematicamente similares. Em teoria, não são necessários atributos específicos para o conteúdo que os filtros colaborativos podem inferir. Mais tarde veremos que adicionar atributos de conteúdo pode melhorar o desempenho, mas não é tecnicamente necessário. A premissa é que se dois usuários tiveram uma forte semelhança de gostos e desgostos no passado, é provável que eles vão continuar a ter uma forte semelhança no futuro. Sistemas de recomendação baseados em Filtros Colaborativos irão recomendar às pessoas que gostam de filmes de romance, aqueles filmes que têm forte conteúdo romântico sem a exigência de definir necessariamente o que seja "romance". São apenas números sendo analisados. Uma vez

estabelecida a semelhança, os itens consumidos por um usuário podem ser recomendados a outros usuários semelhantes. Essa é a ideia por trás dos filtros colaborativos.



Sistemas de recomendação de filtragem colaborativa são formas básicas de motores de recomendação. Neste tipo de mecanismo de recomendação, filtrar itens de um grande conjunto de alternativas é feito pela colaboração de preferências dos usuários. A suposição básica em um sistema de recomendação de filtragem colaborativa é que, se dois usuários compartilharam os mesmos interesses um do outro no passado, eles também terão gostos semelhantes no futuro. Se, por exemplo, o usuário A e o usuário B tiverem preferências de filme semelhantes, e o usuário A assistiu recentemente ao Titanic, que o usuário B ainda não viu, então a ideia é recomendar esse filme ao usuário B. As recomendações do filme no Netflix são um bom exemplo deste tipo de sistema de recomendação. Existem dois subtipos de sistemas de recomendação de filtragem colaborativa:



Na filtragem colaborativa baseada no usuário (User Based), as recomendações são geradas considerando as preferências na vizinhança do usuário. A filtragem colaborativa baseada em usuário é feita em duas etapas: Identificar usuários semelhantes com base em preferências de outros usuários semelhantes e recomendar novos itens a um usuário ativo com base na classificação dada por usuários semelhantes.

Nos sistemas baseados em itens (Item Based), as recomendações podem ser feitas com base na semelhança de produtos com outros produtos que o cliente comprou ou navegou.

A vantagem dos sistemas de filtragem colaborativa é que eles são simples de implementar e muito precisos. No entanto, eles têm seu próprio conjunto de limitações, como o problema do Cold Start

(ou início frio), o que significa que os sistemas de filtragem colaborativa falham em recomendar aos usuários iniciantes cujas informações não estão disponíveis no sistema.

Ao construir sistemas de recomendação de filtragem colaborativa, vamos aprender sobre os seguintes aspectos: Como calcular a similaridade entre os usuários? Como calcular a similaridade entre itens? Como são geradas as recomendações? Como lidar com novos itens e novos usuários cujos dados não são conhecidos? Veremos isso em linguagem R e Python.

Filtragem de Conteúdo

Filtragem baseada em conteúdo foi o estado da arte há 10 anos e ainda é bastante encontrado em uso e tem muitas aplicações na atualidade, mas com o crescimento exponencial no volume de dados, esta técnica perdeu espaço para os filtros colaborativos e você já vai entender o porquê.

Como o nome indica, a filtragem de conteúdo procura semelhanças entre os itens que o cliente consumiu ou navegou no passado para apresentar opções no futuro. Filtragem de conteúdo são classificadores específicos que aprendem a categorizar positivamente ou negativamente as alternativas baseadas nos gostos ou desgostos do usuário. E quando falamos de conteúdo estamos nos referindo aos atributos dos itens que estão sendo recomendados. Por exemplo, um serviço de músicas online como o Spotify, pode apresentar um sistema de recomendação baseado em conteúdo considerando os atributos das músicas (como autor, gênero, ano) e também o perfil do usuário.

Na filtragem colaborativa, consideramos apenas as preferências de itens do usuário e criamos os sistemas de recomendação. Embora essa abordagem seja precisa, faz mais sentido se considerarmos as propriedades do usuário e as propriedades do item enquanto construímos motores de recomendação. Ao contrário da filtragem colaborativa, usamos as propriedades do item e as preferências do usuário nas propriedades do item enquanto criamos mecanismos de recomendação baseados em conteúdo. Um sistema de recomendação de conteúdo normalmente contém uma etapa de geração de perfil de usuário, etapa de geração de perfil de item e criação de modelo para gerar recomendações para um usuário ativo. O sistema de recomendação baseado em conteúdo recomenda itens aos usuários, levando em consideração o conteúdo ou atributos de itens e perfis de usuário. Como exemplo, se você pesquisou vídeos de Lionel Messi no YouTube, o sistema de recomendação baseado em conteúdo aprenderá suas preferências e recomendará outros vídeos relacionados a Lionel Messi e outros vídeos relacionados ao futebol.

Em termos mais simples, o sistema recomenda itens semelhantes aos que o usuário gostou no passado. A similaridade de itens é calculada com base nos atributos associados aos outros itens comparados e é combinada com as preferências históricas do usuário. Ao construir um sistema de recomendação baseado em conteúdo, levamos em consideração as seguintes perguntas: Como escolhemos conteúdo ou recursos dos produtos? Como criamos perfis de usuário com preferências semelhantes às do conteúdo do produto? Como criar semelhança entre itens com base em suas características? Como criar e atualizar perfis de usuário continuamente?

Mas afinal, qual a diferença deste tipo de sistema de recomendação, para os filtros colaborativos?

Essa técnica não leva em consideração as preferências de vizinhança do usuário. Portanto, não exige uma grande preferência do grupo de usuários por itens para uma melhor precisão de recomendação. Considera apenas as preferências passadas do usuário e as propriedades / características dos itens.

O sistema cria um perfil baseado em conteúdo específico do usuário usando atributos discretos. O histórico de consumo ou navegação do usuário é usado para criar um vetor ponderado dos atributos do item. Os pesos são aprendidos ou atribuídos para variar a importância dos atributos para o usuário em particular. Esse peso é usado para comparar com o peso do vetor de diferentes itens que podem ser recomendados. As técnicas de cálculo podem variar de médias simples ponderadas a classificadores bayesianos, análise de agrupamentos, árvores de decisão ou abordagens mais complexas, incluindo redes neurais artificiais.

Um requisito é que você seja capaz de fornecer um número razoavelmente grande de descritores de conteúdo para usar na classificação. Estes podem ser booleanos (o filme é animado, sim ou não, por exemplo). Eles também podem ser contínuos, como a classificação recebida pelo filme a partir de uma fonte de classificação, a "classificação média por estrelas" de outros clientes que consumiram o item ou a porcentagem ou o número de minutos no filme considerado "ação" ou 'romance'. A capacidade de adquirir e manter atributos de conteúdo é um critério chave e uma limitação fundamental deste tipo de sistema de recomendação. Alguns atributos podem ser fáceis de adquirir, mas outros não (por exemplo, atributos constantemente atualizados de produtos eletrônicos ou atributos de filmes). Em ambientes como filmes, música e notícias, o inventário pode mudar tão rapidamente e ser tão grande que adquirir e manter atributos é muito difícil ou muito caro. O site de recomendação de filmes Rotten Tomatoes e o site de Radio na Internet, o Pandora, são exemplos de sistemas de recomendação baseados em conteúdo.

Modelos Híbridos

Nas aulas anteriores estudamos sobre os sistemas baseados em filtros colaborativos e baseados em conteúdo e acredito que você tenha se perguntado se não poderíamos unir as melhores características dos 2 modelos, a fim de criar algo mais robusto. Na verdade, modelos híbridos que utilizam os benefícios de ambos esses modelos já existem e normalmente serão a melhor opção em sistemas de recomendação.

Os motores de recomendação híbridos são construídos pela combinação de vários sistemas de recomendação para construir um sistema mais robusto. Ao combinar vários sistemas de recomendação, podemos substituir as desvantagens de um sistema com as vantagens de outro sistema e, assim, construir um sistema final mais confiável. Por exemplo, ao combinar métodos de filtragem colaborativa (em que o modelo falha quando novos itens não possuem classificação), com sistemas baseados em conteúdo (onde informações de atributos sobre os itens estão disponíveis), novos produtos podem ser recomendados com mais precisão e eficiência.

Por exemplo, se você é um leitor frequente de notícias no Google News, o mecanismo de recomendação, recomenda artigos de notícias para você, combinando artigos de notícias populares, lidos por pessoas semelhantes a você e usando suas preferências pessoais, calculadas usando as informações do clique anterior. Com este tipo de sistema de recomendação, as recomendações de filtragem colaborativa são combinadas com recomendações baseadas em conteúdo para gerar as recomendações finais.

Antes de construir um modelo híbrido, devemos considerar as seguintes perguntas: Que técnicas de recomendação devem ser combinadas para alcançar a solução de negócios? Como combinar várias técnicas e seus resultados para melhores previsões? A vantagem dos motores de recomendação híbridos é que esta abordagem irá aumentar a eficiência das recomendações em comparação com as

técnicas de recomendação individuais. Esta abordagem também sugere uma boa mistura de recomendações para os usuários, tanto no nível personalizado quanto no nível de usuários similares.

Não existem práticas universais específicas para sistemas de recomendação híbridos, e sua construção exigirá a sua visão (você Cientista de Dados) sobre as circunstâncias especiais do problema de negócio a ser resolvido.

Evolução dos Sistemas de Recomendação

Ao longo dos anos, os sistemas de recomendação evoluíram, passaram de métodos mais básicos baseados em similaridades para sistemas de recomendações personalizadas, de recomendações baseadas em contexto, para recomendações em tempo real, de recomendações baseadas em heurísticas básicas, como cálculo de similaridade para abordagens baseadas em Machine Learning.

Nos estágios iniciais desses sistemas de recomendação, apenas as classificações dos usuários sobre os produtos foram utilizadas para gerar recomendações. Nessa época, os pesquisadores usavam apenas as informações de classificação disponíveis. Eles simplesmente aplicaram abordagens heurísticas como o cálculo de similaridade usando distâncias euclidianas, o coeficiente de Pearson, similaridade de cosseno e etc.... Essas abordagens foram bem recebidas e, surpreendentemente, elas funcionam muito bem até hoje.

Esta primeira geração de motores de recomendação é chamada de filtragem colaborativa ou sistemas de recomendação de métodos de vizinhança. Embora eles executem muito bem, estes sistemas vêm com seu próprio conjunto de limitações, tais como problemas de Cold Start (Início frio), quando não são capazes de recomendar produtos a novos usuários sem informações de classificação. Além disso, esses sistemas não conseguem lidar com cenários onde os dados são muito escassos, de modo que as classificações de usuários dos produtos são muito menores.

Para superar essas limitações, novas abordagens foram desenvolvidas. Por exemplo, a fim de lidar com problemas de conjuntos de dados muito esparsos, abordagens matemáticas, como Fatoração de Matriz e Singular Value Decompositions, têm sido utilizados. São técnicas matemáticas que ajudam a resolver problemas nos dados, aumentando a eficácia de sistemas de recomendação.

No início, os cálculos de similaridade foram usados em sistemas de recomendação baseados em conteúdo, mas com avanços em tecnologia e infraestrutura, métodos mais avançados, como modelos de aprendizado de máquina, substituíram os métodos heurísticos. Estes novos modelos baseados em Machine Learning melhoraram a precisão das recomendações. Embora os sistemas de recomendação baseados em conteúdo tenham resolvido muitas das deficiências da filtragem colaborativa, eles têm suas próprias deficiências inerentes, como não ser capaz de recomendar novos itens fora do escopo de preferência do usuário, o que a filtragem colaborativa poderia fazer.

Para resolver esse problema, os pesquisadores começaram a combinar diferentes modelos de recomendação para chegar a modelos híbridos, que são muito mais poderosos do que qualquer um dos modelos individuais. E esses modelos híbridos atualmente englobam não apenas os modelos tradicionais de sistemas de recomendação, como técnicas matemáticas e de Machine Learning, levando a sistemas totalmente personalizados a cada cliente.

Com implementações bem-sucedidas de mecanismos de recomendação personalizados, as pessoas começaram a estender a personalização a outras dimensões chamadas contextos, como a adição de localização, hora, grupo e assim por diante e alteraram o conjunto de recomendações com cada

contexto. Com avanços em tecnologia como grandes ecossistemas de dados, ferramentas analíticas que executam em memória e em tempo real como o Apache Spark, a capacidade de manipulação de bancos de dados muito grandes tornou-se possível. Atualmente estamos nos movendo para sistemas ainda mais personalizados com o uso de inteligência artificial. No aspecto de tecnologia, as recomendações estão passando de abordagens de aprendizado de máquina para abordagens mais avançadas de aprendizagem profunda como redes neurais.

Agora, vamos implementar um sistema de recomendação completo!

Mini-Projeto 7 – Sistema de Recomendação de Filmes da Netflix

```
# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 15</font>
# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
## Mini-Projeto 7
## Sistema de Recomendação de Filmes da Netflix

## Definição do Problema
<p>
O objetivo da Netflix é conectar as pessoas aos filmes que elas adoram. Para ajudar os clientes a encontrar esses filmes, eles desenvolveram um sistema de recomendação de filmes de classe mundial: CinematchSM. Seu trabalho é prever se alguém vai gostar de um filme com base no quanto gostou ou não de outros filmes. A Netflix usa essas previsões para fazer recomendações pessoais de filmes com base nos gostos exclusivos de cada cliente. E embora o <b>Cinematch</b> esteja indo muito bem, sempre pode ser melhorado.
</p>
<p> Existem várias abordagens alternativas interessantes de como o Cinematch funciona que o netflix ainda não experimentou. Alguns são descritos na literatura, outros não. Estamos curiosos para saber se algum deles pode vencer o Cinematch fazendo previsões melhores. Porque, francamente, se houver uma abordagem muito melhor, isso pode fazer uma grande diferença para nossos clientes e nossos negócios. </p>
Objetivos:
1. Prever a avaliação que um usuário daria a um filme que ainda não avaliou.
2. Minimizar a diferença entre a avaliação prevista e real (RMSE e MAPE).
Restrições:
1. Alguma forma de interpretabilidade.
## Fonte de Dados
<p>
A Netflix forneceu muitos dados de classificação anônimos e uma barra de precisão de predição que é 10% melhor do que o que o Cinematch pode fazer no mesmo conjunto de dados de treinamento. A precisão é uma medida de quão próximo as classificações previstas dos filmes correspondem às classificações reais subsequentes.
</p>
<ul>
<li> <a href="https://www.netflixprize.com/rules.html">Netflix Prize</a></li>
<li> <a href="https://www.kaggle.com/netflix-inc/netflix-prize-data">Dataset</a></li>
</ul>
## Instalando e Carregando os Pacotes
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# ->
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Imports
import os
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
import scipy
import sklearn
from scipy import sparse
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
from datetime import datetime
# Formatação dos gráficos
```

```

matplotlib.use('nbagg')
plt.rcParams.update({'figure.max_open_warning': 0})
sns.set_style('whitegrid')
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
### Carregando os Dados
Para carregar os dados vamos executar as seguintes operações:
- 1- Fazer a leitura das linhas de todos os arquivos disponíveis.
- 2- Combinar todas as linhas de todos os arquivos em um único arquivo.
- 3- Carregar o arquivo gerado em um dataframe do pandas.
# ->
# Marca o início da execução de leitura dos arquivos.
start = datetime.now()
# ->
# Criaremos um arquivo final chamado dados.csv
# Se o arquivo não existir, criamos o arquivo em modo de escrita (w)
if not os.path.isfile('dados/dados.csv'):
    # Cria e abre o arquivo para gravação
    dataset = open('dados/dados.csv', mode = 'w')
    # Lista para as linhas dos arquivos
    linhas = list()
    # Nomes e caminhos dos arquivos
    arquivos = ['dados/combined_data_1.txt',
                'dados/combined_data_2.txt',
                'dados/combined_data_3.txt',
                'dados/combined_data_4.txt']
    # Loop por cada arquivo na lista de arquivos
    for arquivo in arquivos:
        # Print
        print("Lendo o arquivo {}".format(arquivo))
        # Com o arquivo aberto, extraímos as linhas
        with open(arquivo) as f:
            # Loop por cada linha do arquivo
            for linha in f:
                # Deletamos o conteúdo da lista
                del linhas[:]
                # Divide as linhas do arquivo pelo caracter de final de linha
                linha = linha.strip()
                # Se encontramos "dois pontos" ao final da linha, fazemos replace removendo o caracter,
                # pois queremos apenas o id do filme
                if linha.endswith(':'):
                    movie_id = linha.replace(':', '')
                    # Se não, criamos uma lista comprehension para fazer a separação das colunas por vírgula
                else:
                    # Separa as colunas
                    linhas = [x for x in linha.split(',')]
                    # Usa o id do filme na posição de índice zero
                    linhas.insert(0, movie_id)
                    # Grava o resultado no novo arquivo
                    dataset.write(','.join(linhas))
                    dataset.write('\n')
            print("Concluído.\n")
        dataset.close()
# ->
# Imprime o tempop total
print("Tempo Total Para Carregar os Arquivos:", datetime.now() - start)
# ->
print("Criando o dataframe pandas a partir do arquivo dados.csv...")
df_netflix = pd.read_csv('dados/dados.csv', sep = ',', names = ['movie', 'user', 'rating', 'date'])
df_netflix.date = pd.to_datetime(df_netflix.date)
print('Concluído.')
# ->
# Ordenando o dataframe por data
print('Ordenando o dataframe por data..')
df_netflix.sort_values(by = 'date', inplace = True)
print('Concluído.')
# ->
# Shape
df_netflix.shape
# ->
# Visualizando os dados
df_netflix.head()

```

Trabalhar com 100 Milhões de Registros não é fácil e isso pode consumir muitos recursos computacionais. Algumas dicas:

- 1- Feche todos os arquivos e softwares no seu computador. Deixe apenas o que for realmente necessário.
- 2- Considere o uso de um ambiente em nuvem ou mesmo cluster de computadores, se possível.
- 3- Reduza o tamanho de cada arquivo. Aqui algumas sugestões de softwares "File Splitter":

```

http://www.fastfilejoiner.com/
https://www.gdgsoft.com/gsplit/download
http://www.kcsoftwares.com/?kfk
## Análise Exploratória dos Dados
# ->
# Resumo dos dados
print("Resumo dos Dados")
print("-"*50)
print("Número Total de Filmes:", len(np.unique(df_netflix.movie)))
print("Número Total de Usuários:", len(np.unique(df_netflix.user)))
print("Número Total de Avaliações:", df_netflix.shape[0])
# ->
# Vamos salvar esses dois valores para usar mais tarde
total_users = len(np.unique(df_netflix.user))
total_movies = len(np.unique(df_netflix.movie))
# ->
# Verificando a média das avaliações
df_netflix.describe()['rating']
# ->
# Verificando se temos valores ausentes
sum(df_netflix.isnull().any())
# ->
# Verificando se temos valores duplicados (para esse caso não consideramos a data)
sum(df_netflix.duplicated(['movie', 'user', 'rating']))
Vamos dividir os dados em treino e teste antes de continuar como a análise exploratória, pois algumas análises só fazem sentido para os dados de
treino. Usaremos a proporção 80/20 para treino/teste.
# ->
# Criaremos um dataset em disco com os dados de treino
# Dessa forma não precisamos executar todo o processo de carga novamente cada vez que executar este notebook
if not os.path.isfile('dados/dados_treino.csv'):
    df_netflix.iloc[:int(df_netflix.shape[0] * 0.80)].to_csv("dados/dados_treino.csv", index = False)
# ->
# Criaremos um dataset em disco com os dados de teste
# Dessa forma não precisamos executar todo o processo de carga novamente cada vez que executar este notebook
if not os.path.isfile('dados/dados_teste.csv'):
    df_netflix.iloc[int(df_netflix.shape[0] * 0.80):].to_csv("dados/dados_teste.csv", index = False)
# ->
# Deletamos o dataframe original para liberar memória
del df_netflix
Quando executar este Jupyter Notebook novamente, pode iniciar a partir desta célula abaixo (após carregar os pacotes).
Caso tenha erro no Jupyter Noteboook, faça um refresh na aba com http://localhost:8888/tree.
# ->
# Agora carregamos os arquivos em dataframes do pandas
df_netflix_treino = pd.read_csv("dados/dados_treino.csv", parse_dates = ['date'])
df_netflix_teste = pd.read_csv("dados/dados_teste.csv")
# ->
# Resumo dos dados de treino
print("Resumo dos Dados de Treino")
print("-"*50)
print("Número Total de Filmes:", len(np.unique(df_netflix_treino.movie)))
print("Número Total de Usuários:", len(np.unique(df_netflix_treino.user)))
print("Número Total de Avaliações:", df_netflix_treino.shape[0])
# ->
# Resumo dos dados de teste
print("Resumo dos Dados de Teste")
print("-"*50)
print("Número Total de Filmes:", len(np.unique(df_netflix_teste.movie)))
print("Número Total de Usuários:", len(np.unique(df_netflix_teste.user)))
print("Número Total de Avaliações:", df_netflix_teste.shape[0])
A função abaixo vai ajustar as medidas em milhares, milhões e bilhões para facilitar a leitura dos gráficos.
# ->
# Função para ajuste das unidades de medida
def ajusta_unidades(num, units = 'M'):
    units = units.lower()
    num = float(num)
    if units == 'k':
        return str(num/10**3) + " K"
    elif units == 'm':
        return str(num/10**6) + " M"
    elif units == 'b':
        return str(num/10**9) + " B"
# ->
# Supress warnings
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
__Vamos verificar a distribuição das avaliações.__

```



```

# ->
# Plot
fig, ax = plt.subplots()
plt.title('Distribuição das Avaliações nos Dados de Treino', fontsize = 15)
sns.countplot(df_netflix_treino.rating)
ax.set_yticklabels([ajusta_unidades(item,
'M') for item in ax.get_yticks()])
ax.set_ylabel('Número de Avaliações (em Milhões)')
plt.show()
__Será que o dia da semana tem influencia na avaliação do usuário? Vamos incluir uma coluna com o dia da semana e descobrir.__
# ->
# Parâmetro para evitar warning devido ao alto volume de dados
pd.options.mode.chained_assignment = None
# ->
# Extraí o dia da semana e grava em uma nova coluna
df_netflix_treino['dia_semana'] = df_netflix_treino['date'].dt.strftime("%A")
df_netflix_treino.head()
# ->
# Plot
fig, ax = plt.subplots()
sns.countplot(x = 'dia_semana', data = df_netflix_treino, ax = ax)
plt.title('Número de Avaliações Por Dia da Semana')
plt.ylabel('Total de Avaliações')
plt.xlabel('')
ax.set_yticklabels([ajusta_unidades(item, 'M') for item in ax.get_yticks()])
plt.show()
__Vamos calcular a média de avaliações por dia da semana.__
# ->
# Média de avaliações por dia da semana
media_dia_semana = df_netflix_treino.groupby(by = ['dia_semana'])['rating'].mean()
print("Média de Avaliações")
print("-"*30)
print(media_dia_semana)
print("\n")
O dia da semana não parecer ter influência na avaliação dos usuários.
__Vamos analisar as avaliações dos usuários ao longo do tempo.__
# ->
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
ax = df_netflix_treino.resample('m', on = 'date')['rating'].count().plot()
ax.set_title('Número de Avaliações Por Mês nos Dados de Treino')
plt.xlabel('Mês')
plt.ylabel('Número de Avaliações Por Mês')
ax.set_yticklabels([ajusta_unidades(item, 'M') for item in ax.get_yticks()])
plt.show()
Claramente há um aumento nas avaliações dos usuários ao longo do tempo, devido ao maior número de usuários ou porque os usuários aprenderam
a usar o recurso.
__Vamos verificar os usuários que mais fizeram avaliações de filmes.__
# ->
# Número de avaliações por usuário
num_aval_por_user = df_netflix_treino.groupby(by = 'user')['rating'].count().sort_values(ascending = False)
num_aval_por_user.head()
# ->
# Resumo estatístico
num_aval_por_user.describe()
__Vamos criar um plot da função de densidade de probabilidade e da função de distribuição acumulada.__
A função de densidade de probabilidade (pdf) e função de distribuição acumulada (cdf) são duas das funções estatísticas mais importantes em
confiabilidade e estão intimamente relacionadas. Quando essas funções são conhecidas, quase qualquer outra medida de confiabilidade de interesse
pode ser derivada ou obtida. Mais sobre isso aqui:
http://reliawiki.org/index.php/Basic\_Statistical\_Background
# ->
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
ax1 = plt.subplot(121)
sns.kdeplot(num_aval_por_user, shade = True, ax = ax1)
plt.xlabel('Número de Avaliações Por Usuário')
plt.title("PDF - Função de Densidade de Probabilidade")
ax2 = plt.subplot(122)
sns.kdeplot(num_aval_por_user, shade = True, cumulative = True, ax = ax2)
plt.xlabel('Número de Avaliações Por Usuário')
plt.title('CDF - Função de Densidade Acumulada')
plt.show()
Observe que a grande maioria dos usuários tem menos de 1000 avaliações.
__Quantas avaliações estão nos últimos 5% de todas as avaliações??__
# ->
# Vamos extrair os percentis
percentis = num_aval_por_user.quantile(np.arange(0,1.01,0.01), interpolation = 'higher')
# ->

```

```

# Visualizando de 5 em 5
percentis[::5]
# ->
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
plt.title("Percentis")
percentis.plot()
# Quartis com diferença de 0.05
plt.scatter(x = percentis.index[::5],
            y = percentis.values[::5],
            c = 'orange',
            label = "Percentis Com Intervalos de 0.05")
# Quartis com diferença de 0.25
plt.scatter(x = percentis.index[::25],
            y = percentis.values[::25],
            c = 'm',
            label = "Percentis Com Intervalos de 0.25")
# Labels e legenda
plt.ylabel('Número de Avaliações Por Usuário')
plt.xlabel('Valor no Percentis')
plt.legend(loc = 'best')
# Vamos marcar os percentis 25, 50, 75 e 100
for x,y in zip(percentis.index[::25], percentis[::25]):
    plt.annotate(s = "({} , {})".format(x,y), xy = (x,y), xytext = (x-0.05, y+500), fontweight = 'bold')
plt.show()
- Existem alguns filmes (que são muito populares) que são avaliados por um grande número de usuários.
- Mas a maioria dos filmes (como 90%) tem algumas centenas de avaliações.
### Criação de Matriz Esparsa


### Criação da Matriz Esparsa de Treino
# ->
# Criamos a matriz esparsa no formato Numpy caso não exista
# Se existir, apenas carregamos a partir do disco
if os.path.isfile('dados/matriz_esparsa_treino.npz'):
    matriz_esparsa_treino = sparse.load_npz('dados/matriz_esparsa_treino.npz')
    print("Matriz Carregada.")
else:
    matriz_esparsa_treino = sparse.csr_matrix((df_netflix_treino.rating.values, (df_netflix_treino.user.values,
                                                                              df_netflix_treino.movie.values)),)
    print('Matriz Criada. O shape é: (user, movie): ', matriz_esparsa_treino.shape)
    sparse.save_npz("dados/matriz_esparsa_treino.npz", matriz_esparsa_treino)
    print('Matriz Salva em Disco.')
# ->
# Calculamos a esparsidade da matriz
linhas, colunas = matriz_esparsa_treino.shape
elementos_nao_zero = matriz_esparsa_treino.count_nonzero()
print("Esparsidade da Matriz de Treino: {} %".format( (1 - (elementos_nao_zero / (linhas * colunas))) * 100) )
### Criação da Matriz Esparsa de Teste
# ->
# Criamos a matriz esparsa no formato Numpy caso não exista
# Se existir, apenas carregamos a partir do disco
if os.path.isfile('dados/matriz_esparsa_teste.npz'):
    matriz_esparsa_teste = sparse.load_npz('dados/matriz_esparsa_teste.npz')
    print("Matriz Carregada.")
else:
    matriz_esparsa_teste = sparse.csr_matrix((df_netflix_teste.rating.values, (df_netflix_teste.user.values,
                                                                              df_netflix_teste.movie.values)))
    print('Matriz Criada. O shape é: (user, movie): ', matriz_esparsa_teste.shape)
    sparse.save_npz("dados/matriz_esparsa_teste.npz", matriz_esparsa_teste)
    print('Matriz Salva em Disco.')
# ->
# Calculamos a esparsidade da matriz
linhas, colunas = matriz_esparsa_teste.shape
elementos_nao_zero = matriz_esparsa_teste.count_nonzero()
print("Esparsidade da Matriz de Teste: {} %".format( (1 - (elementos_nao_zero / (linhas * colunas))) * 100) )
__Vamos calcular a média global de todas as avaliações de filmes, avaliação média por usuário e avaliação média por filme.__
# ->
# Abaixo calculamos a média global de todas as avaliações de usuários.
medias_treino = dict()
medias_treino_global = matriz_esparsa_treino.sum() / matriz_esparsa_treino.count_nonzero()
medias_treino['global'] = medias_treino_global
medias_treino
__Vamos construir uma função para o cálculo da média de avaliações.__
# ->
# Função de cálculo da média
def calcula_media_avaliacoes(sparse_matrix, of_users):
    # Média de avaliações de usuários/eixos

```



```

        top = 100,
        verbose = False,
        verb_for_n_rows = 20,
        draw_time_taken = True):
# Variáveis de controle
no_of_users, _ = sparse_matrix.shape
row_ind, col_ind = sparse_matrix.nonzero()
row_ind = sorted(set(row_ind))
time_taken = list()
rows, cols, data = list(), list(), list()
if verbose: print("Calculando top", top, "similaridades para cada usuário...")
start = datetime.now()
temp = 0
# Loop pela matriz
for row in row_ind[:top] if compute_for_few else row_ind:
    temp = temp + 1
    prev = datetime.now()
    # Calculando a similaridade de cosseno
    sim = cosine_similarity(sparse_matrix.getrow(row), sparse_matrix).ravel()
    top_sim_ind = sim.argsort()[-top:]
    top_sim_val = sim[top_sim_ind]
    rows.extend([row]*top)
    cols.extend(top_sim_ind)
    data.extend(top_sim_val)
    time_taken.append(datetime.now().timestamp() - prev.timestamp())
    if verbose:
        if temp%verb_for_n_rows == 0:
            print("Cálculo concluído para {} usuários [ tempo total : {} ]".format(temp, datetime.now()-start))
if verbose: print("Criação de matriz esparsa a partir das semelhanças computadas...")
if draw_time_taken:
    plt.plot(time_taken, label = 'Tempo de cálculo de cada usuário')
    plt.plot(np.cumsum(time_taken), label = 'Tempo Total')
    plt.legend(loc = 'best')
    plt.xlabel('Usuário')
    plt.ylabel('Tempo (segundos)')
    plt.show()
return sparse.csr_matrix((data, (rows, cols)), shape = (no_of_users, no_of_users)), time_taken
# ->
# Calculamos a similaridade
# Marca o início
start = datetime.now()
# Calcula a similaridade
matriz_esparsa_user, _ = calcula_similaridade_usuario(matriz_esparsa_treino,
                                                    compute_for_few = True,
                                                    top = 100,
                                                    verbose = True)
print("Tempo Total de Processamento:", datetime.now() - start)
Temos **405.041 usuários** em nosso conjunto de treinamento e computação de semelhanças entre eles (**vetor dimensional de 17K**) é demorado.
Tentaremos reduzir as dimensões usando SVD, de modo a acelerar o processo.
## Redução de Dimensionalidade com TruncatedSVD
# ->
# Redução de dimensionalidade
# Marca o início
start = datetime.now()
# Cria o objeto TruncatedSVD reduzindo a dimensionalidade para 500 dimensões
netflix_svd = TruncatedSVD(n_components = 500, algorithm = 'randomized', random_state = 15)
# Aplica o TruncatedSVD
trunc_svd = netflix_svd.fit_transform(matriz_esparsa_treino)
print("Tempo Total de Processamento:", datetime.now() - start)
__Vamos calcular a variância explicada pelos componentes.__
# ->
# Calcula a variância explicada
expl_var = np.cumsum(netflix_svd.explained_variance_ratio_)
# ->
# Plot
fig, (ax1) = plt.subplots(nrows = 1, ncols = 1, figsize = plt.figaspect(.45))
ax1.set_ylabel("Variância Explicada", fontsize = 15)
ax1.set_xlabel("Fatores Latentes", fontsize = 15)
ax1.plot(expl_var)
# Vamos marcar algumas combinações de (fatores latentes, variância explicada) para tornar o gráfico mais claro
ind = [1, 2, 4, 8, 20, 60, 100, 200, 300, 400, 500]
ax1.scatter(x = [i-1 for i in ind], y = expl_var[[i-1 for i in ind]], c = '#ee4422')
for i in ind:
    ax1.annotate(s = "({}, {})".format(i, np.round(expl_var[i-1], 2)), xy = (i-1, expl_var[i-1]),
                xytext = (i+20, expl_var[i-1] - 0.01), fontweight = 'bold')
plt.show()
Com 500 componentes explicamos aproximadamente 65% da variância dos dados. Isso é suficiente para nosso exemplo.

```

```
# ->
# Vamos projetar nossa matriz no espaço de 500 dimensões
start = datetime.now()
trunc_matrix = matriz_esparsa_treino.dot(netflix_svd.components_.T)
print("Tempo de Processamento:", datetime.now() - start)
# ->
# Shape
trunc_matrix.shape
# ->
# Tipo
type(trunc_matrix)
# ->
# Vamos criar e salvar em disco a matriz com a a dimensionalidade reduzida para 500 dimensões
if not os.path.isfile('dados/matriz_esparsa_user_truncada.npz'):
    matriz_esparsa_user_truncada = sparse.csr_matrix(trunc_matrix)
    sparse.save_npz('dados/matriz_esparsa_user_truncada', matriz_esparsa_user_truncada)
else:
    matriz_esparsa_user_truncada = sparse.load_npz('dados/matriz_esparsa_user_truncada.npz')
# ->
# Conferindo o shape
matriz_esparsa_user_truncada.shape
__Agora calculamos novamente a similaridade de usuários usando a matriz truncada.__
# ->
# Calcula similaridade de usuários
# Marca o início
start = datetime.now()
# Calcula a similaridade
trunc_sim_matrix, _ = calcula_similaridade_usuario(matriz_esparsa_user_truncada,
                                                    compute_for_few = True,
                                                    top = 50,
                                                    verbose = True)
print("Tempo de Processamento:", datetime.now() - start)
## Calculando Matriz de Similaridade de Filmes
# ->
# Cálculo da similaridade de filmes
# Marca o início
start = datetime.now()
# Cria se não existir
if not os.path.isfile('dados/matriz_esparsa_filme.npz'):
    matriz_esparsa_filme = cosine_similarity(X = matriz_esparsa_treino.T, dense_output = False)
    print("Matriz Criada.")
    sparse.save_npz("dados/matriz_esparsa_filme.npz", matriz_esparsa_filme)
    print("Matriz Salva em Disco.")
else:
    matriz_esparsa_filme = sparse.load_npz("dados/matriz_esparsa_filme.npz")
    print("Matriz Carregada.")
print("Tempo de Processamento:", datetime.now() - start)
# ->
# Shape
matriz_esparsa_filme.shape
# ->
# Extra os ids dos filmes
movie_ids = np.unique(matriz_esparsa_filme.nonzero()[1])
# ->
# Calcula a similaridade de filmes de acordo com o padrão de avaliação dos usuários
# Marca o início
start = datetime.now()
# Dicionário para armazenar as similaridades
filmes_similares = dict()
# Loop pelos ids dos filmes
for movie in movie_ids:
    # Obtemos os top filmes semelhantes e armazenamos no dicionário
    filmes_sim = matriz_esparsa_filme[movie].toarray().ravel().argsort()[::-1][1:]
    filmes_similares[movie] = filmes_sim[:100]
print("Tempo de Processamento:", datetime.now() - start)
# ->
# Filmes similares ao filme de id 43
filmes_similares[43]
__Agora vamos encontrar os filmes mais semelhantes usando a matriz de similaridade.__
# ->
# Vamos carregar os títulos dos filmes do arquivo csv fornecido pela Netflix
titulos_filmes = pd.read_csv("dados/movie_titles.csv",
                             sep = ',',
                             header = None,
                             names = ['ID_Filme', 'Ano_Lancamento', 'Titulo'],
                             verbose = True,
                             index_col = 'ID_Filme',
                             encoding = "ISO-8859-1")
```

```

# ->
# Visualiza os dados
titulos_filmes.head()
__Vejamos quais são os filmes similares ao filme de ID 43.__
# ->
# ID do filme
id_filme = 43
# ->
# Print
print("Filme:", titulos_filmes.loc[id_filme].values[1])
print("Total de Avaliações de Usuários = {}".format(matriz_esparsa_treino[:,id_filme].getnnz()))
print("Encontramos {} filmes que são similares a este e vamos imprimir os mais similares.".format(matriz_esparsa_filme[:,id_filme].getnnz()))
# ->
# Encontrando todas as similaridades
similarities = matriz_esparsa_filme[id_filme].toarray().ravel()
similar_indices = similarities.argsort()[::-1][1:]
similarities[similar_indices]
sim_indices = similarities.argsort()[::-1][1:]
# ->
# Plot
fig = plt.figure(figsize = plt.figaspect(.45))
plt.plot(similarities[sim_indices], label = "Todas as Avaliações")
plt.plot(similarities[sim_indices[:100]], label = "Top 100 Filmes Similares")
plt.title("Filmes Similares ao Filme {}".format(id_filme), fontsize = 25)
plt.xlabel("Filmes", fontsize = 15)
plt.ylabel("Similaridade de Cosseno", fontsize = 15)
plt.legend()
plt.show()
# ->
# Aqui os top 10 filmes mais similares ao filme 43
titulos_filmes.loc[sim_indices[:10]]

```

Já poderíamos concluir o projeto aqui, pois já temos um sistema de recomendação. Mas iremos além e vamos construir um modelo de Machine Learning para fazer as previsões. Trabalharemos nisso na Parte 2 deste Mini-Projeto.

Fim

Mini-Projeto 7 – Construindo um Modelo de Machine Learning Para as Previsões

```

# <font color='blue'>Data Science Academy - Machine Learning</font>
# <font color='blue'>Capítulo 15</font>

```

Este Jupyter Notebook é um bônus do Mini-Projeto 7. Aqui construímos um modelo de Machine Learning para nosso sistema de recomendação com o objetivo de prever a avaliação que o usuário dará a um filme. O modelo de Machine Learning será criado com o algoritmo XGBoost.

Execute este Jupyter Notebook depois de executar o Jupyter **01-DSA-Cap15-Mini-Projeto7.ipynb** disponível no Capítulo 15 do curso de Machine Learning da DSA.

Leia todos os comentários, inclua a função print() quando quiser compreender a saída de uma operação e estude todo o código usado neste trabalho. Bons estudos.

```

# ->
# Versão da Linguagem Python
from platform import python_version
print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
## Mini-Projeto 7
## Sistema de Recomendação de Filmes da Netflix

## Instalando e Carregando os Pacotes
# ->
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install -U nome_pacote
# Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada
# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.
# ->
# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter notebook.
!pip install -q -U watermark
# ->
# Imports
import os
import random
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import scipy
from scipy import sparse

```

```

import sklearn
from sklearn.metrics.pairwise import cosine_similarity
import xgboost as xgb
from datetime import datetime
# ->
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy" --iversions
## Preparação dos Dados
Vamos trabalhar com amostras dos dados, caso contrário o Jupyter Notebook vai levar muitas horas para ser executado.
A função abaixo será usada para extrair amostras de dados das matrizes esparsas criadas na parte 1 do Mini-Projeto.
# ->
# Função para obter amostra da matriz esparsa
def gera_amostra_matriz_esparsa(sparse_matrix, num_users, num_movies, path, verbose = True):
    # Tupla: (row, col) e (rating) da matriz esparsa
    row_ind, col_ind, ratings = sparse.find(sparse_matrix)
    users = np.unique(row_ind)
    movies = np.unique(col_ind)
    # Random seed para reproduzir o processo aleatório
    np.random.seed(15)
    # Amostras de usuários e filmes
    sample_users = np.random.choice(users, num_users, replace = True)
    sample_movies = np.random.choice(movies, num_movies, replace = True)
    # Gera a máscara booleana
    mask = np.logical_and(np.isin(row_ind, sample_users), np.isin(col_ind, sample_movies))
    # Matriz esparsa com as amostras da matriz original
    amostra_matriz_esparsa = sparse.csr_matrix((ratings[mask], (row_ind[mask], col_ind[mask])),
                                              shape = (max(sample_users) + 1, max(sample_movies) + 1))

    # Salva em disco
    print('Salvando em disco...')
    sparse.save_npz(path, amostra_matriz_esparsa)
    if verbose:
        print('Tarefa concluída.\n')
    return amostra_matriz_esparsa
#### Gerando Amostra de Dados de Treino
# ->
%%time
# Caminho onde está a matriz esparsa de treino gerada na Parte 1 do Mini-Projeto
caminho_matriz_treino_original = "dados/matriz_esparsa_treino.npz"
# Carregando a matriz esparsa
matriz_esparsa_treino_loaded = sparse.load_npz(caminho_matriz_treino_original)
print("Matriz Original Carregada.")
# Onde salvar a amostra
path = 'dados/amostra_matriz_esparsa_treino.npz'
# Obtemos avaliações de 1000 usuários a 100 filmes na matriz esparsa de treino
amostra_matriz_esparsa_treino = gera_amostra_matriz_esparsa(matriz_esparsa_treino_loaded,
                                                            num_users = 1000,
                                                            num_movies = 100,
                                                            path = path)
#### Gerando Amostra de Dados de Teste
# ->
%%time
# Caminho onde está a matriz esparsa de treino gerada na Parte 1 do Mini-Projeto
caminho_matriz_teste_original = "dados/matriz_esparsa_teste.npz"
# Carregando a matriz de amostra, caso já exista
matriz_esparsa_teste_loaded = sparse.load_npz(caminho_matriz_teste_original)
print("Matriz Original Carregada.")
# Onde salvar a amostra
path = 'dados/amostra_matriz_esparsa_teste.npz'
# Obtemos avaliações de 200 usuários a 20 filmes na matriz esparsa de treino
amostra_matriz_esparsa_teste = gera_amostra_matriz_esparsa(matriz_esparsa_teste_loaded,
                                                            num_users = 200,
                                                            num_movies = 20,
                                                            path = path)
# ->
# Resumo
print('Número de avaliações na matriz com amostras de treino: {}'.format(amostra_matriz_esparsa_treino.count_nonzero()))
print('Número de avaliações na matriz com amostras de teste: {}'.format(amostra_matriz_esparsa_teste.count_nonzero()))
Amostras criadas. Altere o número de usuários e número de filmes caso queira trabalhar com amostras maiores.
### Métricas Extraídas dos Dados
Vamos verificar algumas métricas a partir dos dados. Nosso modelo vai prever a avaliação do usuário ao filme.
# ->
# Cria o dicionário
amostra_medias_treino = dict()
A função abaixo será usada para calcular a média de avaliações.
# ->
def calcula_media_ratings(sparse_matrix, of_users):
    # Média de avaliações

```

```

# 1 representa o eixo de usuários
# 0 representa o eixo de filmes
ax = 1 if of_users else 0
# Soma das avaliações
sum_of_ratings = sparse_matrix.sum(axis=ax).A1
# Matriz booleana de avaliações (se um usuário avaliou ou não um filme)
isRated = sparse_matrix != 0
# Número de avaliações de cada usuário ou filme
no_of_ratings = isRated.sum(axis = ax).A1
# Ids da matriz esparsa, u de usuário e m de movie
u,m = sparse_matrix.shape
# Dicionário de usuários e suas avaliações
average_ratings = {}
for i in range(u if of_users else m):
    if no_of_ratings[i] != 0:
        average_ratings[i] = sum_of_ratings[i] / no_of_ratings[i]
return average_ratings
Média global das avaliações dos filmes:
# ->
# Média global
media_global = amostra_matriz_esparsa_treino.sum() / amostra_matriz_esparsa_treino.count_nonzero()
amostra_medias_treino['global'] = media_global
amostra_medias_treino
Média de avaliação por usuário:
# ->
# Calcula a média de avaliação dos usuários
amostra_medias_treino['user'] = calcula_media_ratings(amostra_matriz_esparsa_treino, of_users = True)
# ->
# Vamos extrair um dos usuários do dicionário de filmes (o objetivo aqui é apenas automatizar o processo)
um_usuario = [a for a, b in amostra_medias_treino['user'].items()][0]
um_usuario
# ->
# Print
print('Média de Avaliação do Usuário ' + str(um_usuario) + ':', amostra_medias_treino['user'][um_usuario])
Média de avaliação por filme:
# ->
# Calcula a média de avaliação dos filmes
amostra_medias_treino['movie'] = calcula_media_ratings(amostra_matriz_esparsa_treino, of_users = False)
# ->
# Vamos extrair um dos filmes do dicionário de filmes (o objetivo aqui é apenas automatizar o processo)
um_filme = [a for a, b in amostra_medias_treino['movie'].items()][0]
um_filme
# ->
# Print
print('Média de Avaliação do Filme ' + str(um_filme) + ':', amostra_medias_treino['movie'][um_filme])
## Formatando os Dados
Iremos construir um modelo de regressão, uma vez que desejamos prever as avaliações (valores numéricos). Vamos preparar os dados de treino e teste nas células abaixo.
Essas são as variáveis com as quais vamos construir o modelo:
Variáveis Predictoras (entrada):
- **GAvg** : Média global das avaliações
- **Avaliação de usuários semelhantes** :
    - sur1, sur2, sur3, sur4, sur5 (5 principais usuários similares a cada usuário que avaliou um filme)
- **Filmes semelhantes avaliados por um usuário** :
    - smr1, smr2, smr3, smr4, smr5 (5 principais filmes similares a cada filme avaliado)
- **UAvg** : Média de avaliações dos usuários
- **MAvg** : Média de avaliação do filme
Variável Alvo (saída):
- **rating** : Avaliação do filme dada por um usuário
### Preparando os Dados de Treino Para o Modelo de Regressão
# ->
# Extraindo os dados da matriz de amostras
amostra_usuarios_treino, amostra_filmes_treino, amostra_avaliacoes_treino = sparse.find(amostra_matriz_esparsa_treino)
A célula abaixo leva bastante tempo para ser executada.
# ->
%%time
# Verificamos se o arquivo já existe
if os.path.isfile('dados/dados_treino_reg.csv'):
    print("O arquivo já existe e não precisamos criar novamente..." )
else:
    print('Preparando {} tuplas para o dataset.\n'.format(len(amostra_medias_treino)))
    with open('dados/dados_treino_reg.csv', mode = 'w') as reg_data_file:
        count = 0
        for (user, movie, rating) in zip(amostra_usuarios_treino, amostra_filmes_treino, amostra_avaliacoes_treino):
            ##### Avaliação de um "filme" por usuários similares ao usuário corrente #####
            # Calcula usuário similar ao usuário corrente
            user_sim = cosine_similarity(amostra_matriz_esparsa_treino[user],
                                         amostra_matriz_esparsa_treino).ravel()
            # Obtém top users

```



```

top_sim_users = user_sim.argsort()[::-1][1:]
# Obtém avaliações de usuários similares
top_ratings = amostra_matriz_esparsa_treino[top_sim_users, movie].toarray().ravel()
# Top usuários similares até 5
top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
top_sim_users_ratings.extend([amostra_medias_treino['movie'][movie]]*(5 - len(top_sim_users_ratings)))

##### Avaliações por usuário para filmes similares ao filme corrente #####
# Calcula filmes similares ao filme corrente
movie_sim = cosine_similarity(amostra_matriz_esparsa_treino[:,movie].T,
                              amostra_matriz_esparsa_treino.T).ravel()

# Top filmes
top_sim_movies = movie_sim.argsort()[::-1][1:]
# Obtém avaliações do filme mais similar para o usuário corrente
top_ratings = amostra_matriz_esparsa_treino[user, top_sim_movies].toarray().ravel()
# Top usuários similares até 5
top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
top_sim_movies_ratings.extend([amostra_medias_treino['user'][user]] * (5-len(top_sim_movies_ratings)))
##### Prepara a linha que será armazenada no arquivo #####
row = list()
row.append(user)
row.append(movie)
# Adicionamos outros atributos
row.append(amostra_medias_treino['global'])
row.extend(top_sim_users_ratings)
row.extend(top_sim_movies_ratings)
row.append(amostra_medias_treino['user'][user])
row.append(amostra_medias_treino['movie'][movie])
row.append(rating)
count = count + 1
# if count == 10:
#     break
reg_data_file.write(','.join(map(str, row)))
reg_data_file.write('\n')
if (count)%10000 == 0:
    print("Concluído para {} linhas----- {}".format(count, datetime.now() - start))
Carregamos o arquivo e colocamos em um dataframe.
# ->
df_dados_treino_reg = pd.read_csv('dados/dados_treino_reg.csv',
                                   names = ['user',
                                             'movie',
                                             'GAvg',
                                             'sur1',
                                             'sur2',
                                             'sur3',
                                             'sur4',
                                             'sur5',
                                             'smr1',
                                             'smr2',
                                             'smr3',
                                             'smr4',
                                             'smr5',
                                             'UAvg',
                                             'MAvg',
                                             'rating'],
                                   header = None)

# ->
# Dados
df_dados_treino_reg.head()
### Preparando os Dados de Teste Para o Modelo de Regressão
O processo aqui é igual ao que fizemos com dados de treino.
# ->
# Extraindo os dados da matriz de amostras
amostra_usuarios_teste, amostra_filmes_teste, amostra_avaliacoes_teste = sparse.find(amostra_matriz_esparsa_teste)
# ->
%%time
if os.path.isfile('dados/dados_teste_reg.csv'):
    print("O arquivo já existe e não precisamos criar novamente...")
else:
    print('Preparando {} tuplas para o dataset.\n'.format(len(amostra_avaliacoes_teste)))
    with open('dados/dados_teste_reg.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating) in zip(amostra_usuarios_teste, amostra_filmes_teste, amostra_avaliacoes_teste):
            st = datetime.now()
            # Similaridade de usuários
            try:
                user_sim = cosine_similarity(amostra_matriz_esparsa_treino[user],
                                             amostra_matriz_esparsa_treino).ravel()

```

```

top_sim_users = user_sim.argsort()[::-1][1:]
top_ratings = amostra_matriz_esparsa_treino[top_sim_users, movie].toarray().ravel()
top_sim_users_ratings = list(top_ratings[top_ratings != 0][:5])
top_sim_users_ratings.extend([amostra_medias_treino['movie'][movie]]*(5 - len(top_sim_users_ratings)))
except (IndexError, KeyError):
    top_sim_users_ratings.extend([amostra_medias_treino['global']]*(5 - len(top_sim_users_ratings)))
except:
    print(user, movie)
    raise
# Similaridade de filmes
try:
    movie_sim = cosine_similarity(amostra_matriz_esparsa_treino[:,movie].T,
                                  amostra_matriz_esparsa_treino.T).ravel()
    top_sim_movies = movie_sim.argsort()[::-1][1:]
    top_ratings = amostra_matriz_esparsa_treino[user, top_sim_movies].toarray().ravel()
    top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
    top_sim_movies_ratings.extend([amostra_medias_treino['user'][user]]*(5-len(top_sim_movies_ratings)))
except (IndexError, KeyError):
    top_sim_movies_ratings.extend([amostra_medias_treino['global']]*(5-len(top_sim_movies_ratings)))
except :
    raise
# Prepara os dados para gravar no arquivo
row = list()
row.append(user)
row.append(movie)
row.append(amostra_medias_treino['global'])
row.extend(top_sim_users_ratings)
row.extend(top_sim_movies_ratings)
try:
    row.append(amostra_medias_treino['user'][user])
except KeyError:
    row.append(amostra_medias_treino['global'])
except:
    raise
try:
    row.append(amostra_medias_treino['movie'][movie])
except KeyError:
    row.append(amostra_medias_treino['global'])
except:
    raise
row.append(rating)
count = count + 1
# if count == 5:
#     break
reg_data_file.write(', '.join(map(str, row)))
reg_data_file.write("\n")
if (count)%1000 == 0:
    print("Concluído em {} linhas----- {}".format(count, datetime.now() - start))
Carregamos o arquivo e colocamos em um dataframe.
# ->
# Gera o dataset de teste
df_dados_teste_reg = pd.read_csv('dados/dados_teste_reg.csv', names = ['user',
                                'movie',
                                'GAvg',
                                'sur1',
                                'sur2',
                                'sur3',
                                'sur4',
                                'sur5',
                                'smr1',
                                'smr2',
                                'smr3',
                                'smr4',
                                'smr5',
                                'UAvg',
                                'MAvg',
                                'rating'],
                                header = None)
# ->
df_dados_teste_reg.head()
## Construindo o Modelo de Machine Learning
A última etapa do trabalho é construir, treinar e avaliar o modelo.
# ->
# Dicionários para avaliação do modelo
models_evaluation_train = dict()
models_evaluation_test = dict()
Abaixo algumas funções para executar o modelo.
# ->

```

```

# Função para o cálculo do erro do modelo
def calcula_metricas(y_true, y_pred):
    rmse = np.sqrt(np.mean([ (y_true[i] - y_pred[i])**2 for i in range(len(y_pred)) ]))
    mape = np.mean(np.abs( (y_true - y_pred)/y_true )) * 100
    return rmse, mape
# ->
# Função para treino e teste do modelo
def executa_modelo_xgboost(modelo, x_train, y_train, x_test, y_test, verbose = True):
    # Dicionários
    train_results = dict()
    test_results = dict()
    # Treinamento do modelo
    print('Treinando o modelo..')
    start = datetime.now()
    modelo.fit(x_train, y_train, eval_metric = 'rmse')
    print('Concluído. Tempo total: {}'.format(datetime.now() - start))
    # Calculando o erro do modelo nos dados de treino
    print('Calculando as Métricas com Dados de Treino.')
    start = datetime.now()
    y_train_pred = modelo.predict(x_train)
    rmse_train, mape_train = calcula_metricas(y_train.values, y_train_pred)
    # Grava os resultados
    train_results = {'rmse': rmse_train, 'mape': mape_train, 'previsoes': y_train_pred}
    if verbose:
        print("\nErro do Modelo em Dados de Treino")
        print('-'*30)
        print('RMSE : ', rmse_train)
        print('MAPE : ', mape_train)
    # Avaliando o modelo com dados de teste
    print("\nAvaliando o modelo com dados de teste.")
    y_test_pred = modelo.predict(x_test)
    rmse_test, mape_test = calcula_metricas(y_test.values, y_test_pred)
    # Grava os resultados
    test_results = {'rmse': rmse_test, 'mape': mape_test, 'previsoes': y_test_pred}
    if verbose:
        print("\nErro do Modelo em Dados de Teste")
        print('-'*30)
        print('RMSE : ', rmse_test)
        print('MAPE : ', mape_test)
    return train_results, test_results
# ->
# Seed
my_seed = 15
random.seed(my_seed)
np.random.seed(my_seed)
## Treinamento do Modelo
# ->
# Prepara os dados de treino
x_treino = df_dados_treino_reg.drop(['user', 'movie', 'rating'], axis = 1)
y_treino = df_dados_treino_reg['rating']
# ->
# Prepara os dados de teste
x_teste = df_dados_teste_reg.drop(['user', 'movie', 'rating'], axis = 1)
y_teste = df_dados_teste_reg['rating']
# ->
# Cria
# o modelo de regressão com 100 estimadores
modelo_xgb = xgb.XGBRegressor(silent = False, random_state = 15, n_estimators = 100)
# ->
# Treinamento do modelo
train_results, test_results = executa_modelo_xgboost(modelo_xgb, x_treino, y_treino, x_teste, y_teste)
# ->
# Armazena os resultados da avaliação do modelo
models_evaluation_train['modelo_xgb'] = train_results
models_evaluation_test['modelo_xgb'] = test_results
# ->
# Variáveis mais importantes para o modelo
xgb.plot_importance(modelo_xgb)
plt.show()
Além de construir o modelo também identificamos as variáveis mais relevantes. Observe que não há surpresa. As avaliações de usuários são determinantes para recomendar os filmes avaliados para outros usuários.
## Salvando o Resultado
# ->
# Salva os resultados em disco
pd.DataFrame(models_evaluation_test).to_csv('dados/resultado.csv')
models = pd.read_csv('dados/resultado.csv', index_col = 0)
models.loc['rmse'].sort_values()
# Fim

```

