Notas do autor

Erasmus 2024 - Suécia @JoseCutileiro

Estes são os meus apontamentos da cadeira de AML (applied machien learning, é uma versão integral de notas retiradas de aulas e videos, espero que gostem)

Aula 1: INTRODUÇÃO

L1: Introdução

ML é cada vez mais popular nos dias de hoje. Porque agora? Muitas histórias de sucesso desde tradução, geração de imagens e texto...

O que vamos aprender neste curso?

1. Modelos de machine learning
2. Contexto da vida real

aulas

1. Discussão (ver antes, perguntar, coding interativo...)
2. assistência (para ajuda no projeto)

avaliações

1. ML workflows e decision trees
2. random forests
3. text classification

4. neural network software
5. image classification
6. Exame: parte 1 (cenas básicas obrigatorio para passar) e parte 2 (cenas mais dificeis)

Fundamentos básicos de ML

Modelos preditivos

Objetivo: Fazer uma previsão

Este paciente é diabético? Qual é o animal desta imagem? Qual é o valor de mercado desta casa?

Oberservar dados --> fazer previsões

Porque ML?

Queremos escrever funções que não sabemos bem como são, quando sabes a função, perde um pouco o sentido.

ML não faz tudo

1. Definir tarefas, terminologia e métricas de avaliação
2. Labels no treino e no teste (às vezes)
3. Saber as features (às vezes)
4. Análise do erro

Learning types

1. Supervised: Training set consists of IN-OUT pairs, objetivo é aprender a produzir os outputs sozinho
2. Unsupervised: Dados não categorizados, objetivo é descobir uma estrutura nos dados

3. Reinforcement learning: O problema é formalizado num jogo, objetivo construir um agente que resolva o jogo

Aproaches diferentes

1. Separadores lineares - Fazer uma equação linear e separar
2. Separadores por arvores

Workflow

1. Dados -> Treinar -> Modelo
2. Input -> Modelo -> label (out)

AVALIAR O MODELO

1. Métrica de avaliação
2. Test se or cross-validation

Aula 2: DECISION TREE

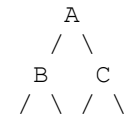
link: https://www.youtube.com/watch?v=fUoOhjMF9zg

VIDEO

Decision tree

Formalmente uma decision tree é um programa que vai basear as suas previsões dependendo de vários IF e ELSE (binário)

Exemplo



Se A vai para B se B escolhe algo ...

Nós internos: Decisões

Leaf node: Output value

Que funções matemáticas podemos retirar?

As regiões são todas com angulos retos

1. Fácil de entender e visualizar
2. Pode ter limitações

Como é que funciona o algoritmo?

Vemos qual é a feature mais útil, e vamos colocando no topo, repetimos isto quantas vezes quisersmos (ou até ao número de features no máximo).

Exemplos de algoritmos: ID3, C4.5, CART ...

PSEUDO-CODE

```
def TrainDecisionTreeClassifier(X,Y, depth=0):
    if (all ouputs in Y are the same):
        return leaf with that output

    if (depth > maxdepth):
        return leaf with majoraty class

    F = allFeatures(X)
```

```
    # Decide the best feature
    for fi in F:
        Xi,Yi <- subset onde F = fi
        Ti <- TRAIN(Xi,Yi)

    return tree that splits on F
```

USEFUL FEATURE?

É útil se o subgrupo é homogénio

Exemplos:

1. majoraty class sum
2. information gain
3. Gini impurity (é o default do scikit-learn)

Homogéneo?

Majority class

somar as frequências

Information gain

Baseado na entropia dos subsets

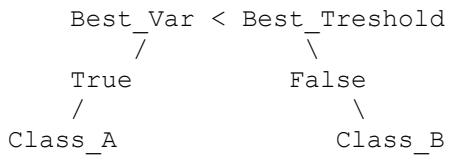
Relembra: Entropia a 0 se todos forem o mesmo valor, entropria máxima se todas as classes aparecem o mesmo número de vezes

Objetivo: Fazer com que a entropia seja o mais próximo de 0 possível, isso significa que os subsets são bastante homogeneos.

Treshold

É importante definir o treshhold como deve ser também, depois de escolheres a variavel mais util, escolhes o valor que mais uma vez, faz com que exista menos entropia no subset.

Exemplo:



Regression trees

Relembra: Regression é para preveres outputs numéricos, funciona exatamente da mesma forma mas em vez de preveres a class prevês o número

Problemas

Relembra:

1. Underfit: Demasiado simple
2. Overfit: Demasiado complexo

Nas decision trees, isto é visto pela DEPTH. Se tivermos muito pouca estamos a olhar para poucas features o que faz com que o modelo seja muito simples, no outro lado se tiveres demasiado simplesmente estás a decorar o dataset. O que faz com que tenha piores resultados no TEST SET. (os resultados no TRAIN SET são cada vez melhores)

Notas finais:

Datasets

https://archive.ics.uci.edu/datasets

Aula 3: ENSEMBLES

video: https://www.youtube.com/watch?v=TNkBkWkz6Jg

Introduction to Ensembles

A ideia chave é usar multiplos modelos em vez de um, isto permite aproveitar os aspetos positivos de cada um dos modelos. Exemplo simples:

Um dos modelos tem accuracy 60% e o outro 70%, assumimos que os erros dos modelos são independentes (o que não é realista na prática mas é bonito). Claro que juntanto muitos modelos, se os erros forem sempre independentes, a accuracy depois daria bem.

Como implementar os ensembles?

VOTING

Voto por maioria

eg.

1. Modelo X ---> vota 1
2. Modelo Y ---> vota 0
3. Modelo Z ---> vota 1

Resultado: 1

Voto por média

Cada modelos diz que o output é X com probabilidade P, depois a ideia é fazer a média e o que tiver melhor resultado sai no final.

Stacking

Usa o output de cada um dos modelos, e usa estes outputs como features para um novo modelo.

Programar:

Não é muito complicado de implementar uma coisa do genero de raiz, mas podemos usar com o scikit learn, já tem uma coisa que funciona bem.

E para a regressão?

Usamos as mesma técnicas, e.g média ou stacking. O por maioria não faria tanto sentido.

Training ensebles

A ideia é que os modelos sejam diversos. Como tornar os modelos diversos? Uma maneira simples é cada um dos modelos focasse numa parte do training set. Ou fazer pequenas alterações no training set para cada um dos modelos, também funciona bem.

BAGGING (bootstrap aggregating)

Training set original é partido (com reposição, ou seja cada split do set pode ter registos que aparecem noutro split)

Amostras do training set são alimentadas a cada um dos modelos, e no fim combinar os resultados

SUBSET OF FEATURES

feature bagging (sem reposição, as features que foram retiradas num dos datasets não seram retiradas no outro)

A mesma ideia, mas para colunas do dataset.

Aula 4: Random forests

Video: https://www.youtube.com/watch?v=ICp2VzmdM4c

Random forests

No fundo as random forests são ensembles de decision trees. Tal como as decision trees funcionam para classificação e regressão.

Decision tree (revisão)

1. Ver a feature F mais util
2. fazer a arvore com F
3. separar em subgrupos baseados em F
4. Repetir isto o nr de vezes que quiseres

(podes ver mais em A2)

Só aqui uma notazinha importante, nas decision trees normalmente quando chegamos à folha escolhemos o output. Mas o output da leaf pode ser uma probabilidade, isto é importante para os ensembles (ajuda a combinar é compativel tanto com os de maioria tanto com os de probabilidade)

(nota: MLE maximum likelihood estimator)

Decision trees não são perfeitas...

1. Tendem MUITOOO para overfit, são más a generalizar dado que decoram literalmente o dataset.

Os ensembles são ótimos a lidar com a redução do overfit

Random forests: Decision tree + ensemble

Random forests

São muito boas no geral, não são o melhor modelo para uma tarefa individualmente mas conseguem obter bons resultados genéricamente. Claro que isto pode ter casos particulares que acabam por não funcionar

Implementação

1. Bagging (separação do training set)

2. Usa subset de features

No final

1. Classificação: Fazer médias das médias
2. Regressão: Média dos resultados (ou média ponderada)

Hyperparameters nas random forests

1. Quantas arvores?
2. Quantas features é que cada nó considera?
3. Hyperparametros standard das decision trees (depth, feature selection criterion ...)

(relembra os critérios: information gain ou gini)

Pros and cons

PROS

1. são muito simples
2. melhores resultados que uma single tree
3. fácil de misturar features

CONS

1. Menos interpretaveis que a single tree
2. Mais pesados que a single tree
3. Precisam de boas features (não funciona bem com imagens ou texto ou som ...)

Aula 5: Generalization in Machine Learning

Video: https://www.youtube.com/watch?v=odkoeAKFYOU

What is happening when we learn?

1. O algoritmo vê exemplos do training set e tenta perceber padrões que estão por detrás dos dados. Tenta generalizar

2. Depois faz previsões de acordo com o que aprendeu

Como escolher bons modelos

Espaço e hipostese: Todos os possiveis outputs de um algoritmo

Exemplo para decision trees: Todas as decision trees possiveis dado um conjunto de features

Funcoes de erro

missclassification: 0 se errou 1 se acertou -> soma e vê a accuracy

squared error: ver quanto se distancia da média (ao quadrado). Para que serve o ao quadrado perguntam vocês: Dar mais valor ao que quase acertam e lixar os que falham por muito.

Objetivo: Minimizar o erro (regressão)

Generalization gap

A performance nos dados de treino vai ser sempre superior aos resultados nos testes, os dados podem ter um "noise" e quando o nosso modelo aprende o modelo acaba por captar este noise também. Depois trama-se nos resultados. A este fenomeno chamamos de Generalization gap.

Que fatores afetam este generalization gap?

1. Se o modelo for mais complexo, é mais propicio a que isto aconteça: Perigo da dimensionalidade
2. Modelos complexos conseguem captar o "noise" mais facilmente pois são mais poderosos

(muito relacionado com o overfit...)

Há sempre o tradeoff de underfit e overfit.

Complexity

O que é complexidade no geral? Depende do modelo claro. Mas quase todos os algoritmos têm HYPERPARAMETERS (depth, tamanho das hiddens layers, learning rate, pesos ...). Controlando estes parametros conseguimos controlar a complexidade do modelo.

goodness of fit: O modelo tem que ser capaz de coletar a informação do dataset.

regularization: O modelo deve ser simples

Fatores que afetam o fator de generalização

1. Tamanho do training set (se for mais pequeno é mais fácil de entrar em regime de overfit)
2. Muita complexidade

Outra coisa, na prática os dados de treino são muito diferentes e de fontes variadas. Exemplo: texto. É importante ter noção de uma coisa, os dados coletados devem ter alguma relação com os dados que queremos prever no fim de contas. Não vais coletar dados ao big brother para usar numa cirurgia.

Aula 6: Preprocessing and Encoding Features

Video: https://www.youtube.com/watch?v=60Oe5BQlC8E

Caso trivial

Todos os dados já estão com o formato numérico portanto é trivial converter para matriz

Caso não tão trivial

Nem todas as features estão no formato numérico, o que depois dá problema para os algoritmos de ML.

A ideia agora é perceber como é que podemso codificar estas features como valores numéricos (encoding features). Assim os algoritmos funcionarão melhor.

Notinha

Há muitas maneiras de ver dados em tabelas

1. Tabelas
2. Array de dicionarios
3. Array de Lists

One hot encoding (dummification)

Cada valor possivel de uma variavel passa a uma variavel, fica a 0 se não for o caso e 1 se for o caso.

Será isto um desperdicio?

Será complicar demasiado?

Por que não usar simplesmente numeros?

Quando utilizamos numeros estamos a criar relações que não existiam previamente no dataset. Em algumas variaveis isto fará sentido (exemplo: notas de A a F). Quando não conseguimos arranjar uma ordem -> Aplicar dummification.

Bag-of-words (tem em conta frequências)

Exemplo

"Example text" --> 0 1 1 0

"Another text" --> 1 0 1 0

"Another text in another example" --> 2 1 1 1

(another, example, text, in)

Magnitudes das variaveis

Para os algoritmos não interessa se está em metros, centimetros, kilometros ... Temos que fazer com que a magnitude não seja um problema --> SCALING

As magnitudes afetam modelos lineares, NN e os KNNs mas não afetam algortimos baseados nas decision trees.

Scaling and normalization

1. min/max scaling
2. standard scaling
3. length normalization (usado para documentos)

Outras transformações

1. Log
2. SquareRoot
3. Usar a intuição e a tentativa e erro

TF-IDF (term frequency and inverse document frequency)

tf-idf(t,D) = tf(t,D) * log (1 + N / 1 + df(t))

(despesar as palavras que usam em "demasiados" documentos, exemplo: "the" "a" ...)

Missing values

É muito comum isto acontecer, falha humana, erro nos sensores ...

1. Ignorar instancias que não têm valor, problema: Podemos ter que ignorar muitos registos e acabara por fazer asneirada no fim de contas

2. Outra solução é feature imputation (substituir pela média, moda, com um valor qualquer ou até tentar prever o valor observando as outras features)

Aula 7: Feature selection

Video: https://www.youtube.com/watch?v=jldtA2VA6Ro

Qual é a ideia?

Algumas das features que temos são realmente uteis para encontrar os resultados que queremos. Mas podemos cair no erro de acreditar que todas as features são uteis. Então a ideia é tentar limpar algumas das features. Isto diminui o risco de overfit e pode fazer com que o modelo acabe por generalizar melhor.

Feature selection é mesmo isto, tentar encontrar um subset das features originais.

Metodos

1. Metodos de filtragem: Originalmente começamso com o set de todas as features, depois escolhemos um subset, fazemos o nosso algoritmo e vemos a performance, depois manualmente decidimos o que acharmos melhor.
2. Wrapper methods: Inicialmente começamos com todas as features também, depois a ideia é fazer o mesmo que os metodos de filtragem mas a escolha do subset está incluida é feita com a ajuda do modelo (loop em que adaptamos o subset ao modelo). No final avaliamos a performance
3. Embedded methods: O mesmo que os wrapper methods mas a avaliação da performance também está incluida no loop e ajuda a escolher o melhor subset

Associar a feature e o output

Nota: Só aqui uma lembrança --> false predictor

Posto isto de lado, a ideia é ver que variaveis estão correlacionadas com o output, a ideia é ter uma função de classificação (podem haver várias dependendo se é para classificação ou regressão)

Funções de classificação:

1. Mutual information

Temos duas váriavies discretas (!= continuas), X e Y, l(X,Y) = somatorio de todos os X e Y de p(x,y) * log (p(x,y) / p(x)*p(y))

2. F-score (baseado no teste anova) (estatico)

3. Qui-Quadrado (baseado no teste do qui-quadrado) (estatico)

4. Self-check (o que acontece se X e Y forem independentes? => Naive bayes assumption)

Feature selection no scikit-learn

1. Seletores -> SelectKbest, SelectPercentile
2. Feature scoring function -> f_classif, mutual_info_classif, chi2, mutual_info_refression, f_regression

Problemas comuns:

Cada feature é considerada em isolação, podem haver features que funcionam apenas se tiverem combinadas. Exemplo do problema do XOR, se tivermos duas variaveis X1 e X2 e Y (output). Saber que X1 é 0 ou 1 em nada nos vai ajudar a prever o output. Apenas com a ajuda de X2 podemos prever o output na perfeição.

Solução

1. Brute force: Fazer TODOS os subsets possiveis dado um conjunto de variaveis. Isto não funciona na prática... (devido à explosão exponencial que vem do fator combinatório da questão, $2^{N)}$. Só aqui uma notinha se não perceberem o porque de 2^{N} (ou está ativo ou não está ativo). Poranto dado que isto não funciona, nós queremos de alguma forma aproximar este algoritmo.
2. Wrapper method (Sequential forward selection): Começar vazio e ir enchendo com as variaveis que mais ajudam até já não ajudar mais ou não ajudar quase nada. Tem as suas vantagens: Não adiciona features redundantes, é muito simples. Mas também têm as suas desvantagens: Pode ser impossivel de apanhar as features a funcionar em conjunto (não consegue resolver o XOR por exemplo), mesmo este aproach greedy pode ser pesado se houverem muitas features (apesar de exponencialmente menos pesado que o outro, mas estamos a falar de modelos de ML que por si só têm uma complexidade elevada), é sub-optimal, ou seja este algoritmo não irá encontrar o subset otimo para resolver o problema.

Embedded feature selection

O próprio algoritmo de ML trata deste problema, por exemplo as DECISION TREES (fazem feature selection a cada depth). Alguns modelos lineares também fazem isto utilizando a regularização. Alguns exemplos de regularização são, a L1 (LASSO), a L0 a L2 ... (isto vai ser falado)

Aula 8: Tuning Hyperparameters

Video: https://www.youtube.com/watch?v=XXC9Yw0RRqI

O que são hyperparametros?

São inputs que damos aos nossos algoritmos de ML, estes inputs controlam de alguma forma o comportamento base dos nossos algoritmos bem como a sua complexidade e adequação aos dados, muitas vezes são estes parametros que fazem com que o algoritmo funcione ou não

Exemplos:

1. Decision trees: Depth
2. Random forests: Ensemble size
3. ...

Nota para o projeto

Hyperparametros interessantes das random forests: depth, ensemble size, criterion (majority sum, info gain, gini)

À medida que os algoritmos ficam mais complexos normalmente existem mais hyperparametros.

Exemplo do professor: LinearSVC (support vector classifier), este modelo tem 10 hyperparametros que podemos modificar (é muito)

Como obter bons resultados?

1. Não fazer nada

2. Tentar utilizar conhecimento de dominio
3. Tentativa e erro (pode ser impossivel...)

GRID SEARCH => Todas as combinações são utilizadas a melhor é selecionada. (no scikit-learn: GridSearchCV)

RANDOM SEARCH => Funciona bem quando não tens assim tantos hyperparameters. Este aproach é muito bom se não tiveres assim tantos recursos para gastar, fazes uma data de randoms e escolhes o melhor naquele timeframe. É preciso perceber a distribuição dos hyperparametros para escolher os valores aleatorios de maneira correta. (no scikit-learn: RandomizedSearchCV)

Estas soluções são brute force, não encontramos nada que pareça mais inteligente? (bayesian optimization, stochastic optimization) isto é chamado black-box optimization dado que não sabemos muito bem como é que o podemos fazer.

Nota: AutoML (fazer este processo todo mas sozinho)

Aula 9: Imbalanced Classification Tasks

Video: https://www.youtube.com/watch?v=lO8EbeiMVsQ

Problemas de raridades

Ou chamados os problemas da aguda no palheiro, são problemas em que queremos identificar fenomenos menos comuns que o fenomeno "normal". Por exemplo terramotos, detetar fraúde, SPAM ... Quando lidamos com este tipo de problemas os datasets são geralmente muito desbalanceados. (Exemplo: classificador que avalia todos os seus pacientes como saudaveis --> accuracy é de 99%, mas não tem nenhum valor). A nossa maneira de avaliar o modelo tem que mudar de alguma forma, como é que isto funciona?

Utilizar outras métricas: RECALL, PRECISION, sensitivity/specificity. Isto consegue lidar com falsos positivos e falsos negativos.

Em ML

Normalmente estes modelos tendem a otimizar a sua performance, o que pode afetar negativamente a classe de minoria. (nota: os algoritmos não querem simplesmente melhorar a accuracy, eles melhoram em função da loss function que tende a ser melhor que simplesmente a accuracy, mas ainda assim pode ter problemas)

Soluções

1. Tentar melhorar o algortimo -> Exemplo: um parametro do LinearSVC existe uma classe que se chama class_weight
2. Tentar melhorar os dados -> Undersampling, Oversampling

Undersampling

Pegar na classe de maioria e descalar

Oversampling

Pegar na classe de minoria e duplicar de algum modo (copiar valores anteriores)

SMOTE

Fabricar novas instâncias da classe de minoria (data augmentation) --> Generate synthetic instances

Aula 10: Linear Algebra Refresher

Video: https://www.youtube.com/watch?v=KgfReKdPoyE

Algebra linear

algebra linear é o ramo de matemática que trata de relações, podem ser linhas ou planos.

Vetores

Operações básicas

1. Escalar: a * Vn

2. Adição e subtração: Vn + Wn
3. Produto escalar: Vn * Wn
4. Norma do vetor: | Vn |

Implementações em python

```
def scale(a, v):
    return [a*vk for vk in v]

def vsum(v, w):
    return [vk + wk for (vk,wk) in zip(v, w)]

def dot(v, w):
    return sum(vk * wk for (vk, wk) in zip(v, w))

def vlength(v):
    return math.sqrt(dot(v, v))
```

Nota: não usar estas porque são menos eficientes que as outras, em vez disso usar o Numpy ou o Scipy. Utilizam coisas diretamente ligadas com o CPU e a GPY

Matrizes

Vetores de vetores, também dá para escalar, adição e subtração. Mas tens que ter cuidado com dimensões. Mutiplicar matrizes (não é comutativo).

Nota: Sparse vectors, manter track apenas das entradas que não são zero.

Exemplo [0 1 0]

Passa para [(1,2)]

Claro que há sempre o balanço que temos de fazer, mas se o vetor tiver muitos zeors (for esparso), vai ser melhor

Como fazer código mais rápido (ainda mais)

1. Usar BLAS (é a biblioteca usada pelo numpy)

2. Cython
3. Usar GPU (Bibliotecas de alto nivel -> PyTorch, tensorFlow)
4. Usar GPU (Bilbliotecas de baixo nivel -> Kuda)

Normalmente é melhor começar por prototipar com o numpy e esse tipo de implementações, e após isto refazer com uma implementação menos genérica.

Problemas de ambiente

Estes modelos são muito pesados e acabam por ter impacto direto no ambiente.

Aula 11: Linear Classifiers

Video: https://www.youtube.com/watch?v=eTxdyMAAiaw

Binary classification tasks

Dizemos que a classificação é binária se tivermos duas classes possiveis no output.

Classificadores lineares:

1. Linear classifier

Temos uma função de scores

Score = w * x

(vetor de pesos e vetor de features) Também pode aparecer doutra maneira, mas a forma se cima não é mais que um caso geral.

Score = w * x + b

(exemplo):

Queremos classificar documentos como tendo uma opinião boa ou má

Temos um texto (INPUT):

This book is fantastic

Temos multiplas palavras:

[garbage, this, like, fantastic]

Temos pesos associados às palavrass:

[-3, 0.3, 1.2, 3]

Temos um bag of words (input intermédio):

CountVectorizer

[0,1,0,1]

E agora tentamos prever o input que nos foi dado.

Visualização no plano

Pontos e uma linha

Como treinar classificadores lineares?

Basicamente o nosso objetivo é chegar ao W correto, a questão é como. Pode ser com o percetrão com o naive bayes, com o support vecotr machines...

Percetrão

1. Começar com o nosso W com tudo a zeros.

2. repetir: classificar o resultado com o nosso W atual, atualizar se o resultado foi mau (ver pseudo codigo)

```
w = zeros()

repeat N times:
    for (xi, yi) in the training set:
        score = w * xi

        if (errado):
            if (y é positivo):
                w += xi
            else:
                w -= xi
return w
```

Limitação do percetrão: Os classificadores lineares só conseguem resultados bons se os dados forem linearmente separáveis. XOR problem

Como criar um classificador linear?

```
class LinearClassifier(obj):

    def predict(self, w):
        score = x.dot(self.w)

        if (score >= 0.0):
            return positive_class

        else:
            return negative_class

    def predict_all(self,X):
        scores = X.dot(self.w)
        out = numpy.select([scores >= 0.0, scores < 0.0],   [postive_class,negative_class])

        return out


class Perceptron(LinearClassifier):
```

```
def fit(self, X, Y):
    n_features  = X.shape[1]
    self.w = numpy.zeros(n_features)

    for i in range(NUM_ITERS):
        for x, y in zip(X,Y):
            score = self.w.dor(x)

            if (score <= 0 and y == positive_class):
                self.w += x

            if (score > 0 and y == negative_calss):
                self.w -= x
```

Este é apenas um dos LinearClassifiers, existem mais e já implementados no scikit learn (Percetron, logistic regression, linearSVC)

No scikit learn o W é o model.coef_

Aula 12: Regression models

Video: https://www.youtube.com/watch?v=deVqGW3YNrk

Regressão linear

O modelo prevê outputs com a seguinte formula

y = w * x

Em muitos dados este modelo é insuficiente, mas na prática eles funcionam muito bem

Como treinar estes modelos?

1. Least-squares regression: Basicamente a ideia é encontrar o W que minimiza o squared error

Existe uma formula fechada que resolve este problema

w = (Xt X)^-1 Xt Y contudo esta não é utilizada dado que é muito lenta (inversa das matrizes, e também é instavel)

Funciona melhor soluções iterativas

Widrow-Hoff

Um dos possiveis algoritmos, vê o pseudo codigo

```
w = zeros()

repetir N vezes:
    para cada entrada do training set x, y
        guess = w * x

        error = g - y

        w = w - lr * error * xi

return w


Nota: lr > learning rate
```

Linear regressors no scikit-learn

1. LinearRegression
2. Ridge
3. Lasso
4. LinearSVR

Aula 13: Collecting Data for Machine Learning and Annotating data

Video: https://www.youtube.com/watch?v=M4qrgqfKQDo

Parte 1: Collecting Data for Machine Learning

Dados

Usados para TREINAR e para AVALIAR. Pode server para mais que isto como jornalismo, investigação...

Modos de explorar os dados

1. Supervises: Pares de IN e OUT, usados para prever ou classificar
2. Unsupervised: Quermos dividir o dataset em grupos (exemplo: clusters)
3. Semisupervised: Usado quando fazer label de data é muito caro

De onde vêm os dados?

1. Online dataset (kaggle, UCI, MNIST...)
2. Mas os dados não aparecem do céu... Provavelmente o dataset que precisas não estará em dominio publico

Como coletar dados e como anotar dados?

Tipos de dados:

Numerical (1,2, 5%, ...) Categorical (male, female) Textual Graphical (imagem) Audio ...

Seleção de dados

1. TOP DOWN: O que queremos, qual é o proposito
2. BOTTOM UP: O que é que posso obter destes dados e como

Como ter os dados?

1. Databases já existentes (privados ou publicos)
2. Log file (em sensores, computadores ...)

3. Scraping from websites
4. Using APIs (exemplo de imagens: flickr)

Nota: Estar na net não é publico, treinar modelos com conteudo de autores pode dar bostique. Podem usar distribuição por URLs mas isto poderá ter outro problema pois o URL pode ficar inacessivel.

População e representação

Queremos modelar uma população, (pessoas, imagens, sons ...). A ideia base é que queremos que a nossa amostra seja representativa e verdadeira. O que pode ser um problema e uma dificuldade. Exemplo do professor das sondagens: Recolheram mdados de maneira errada, ou focaram num grupo de pessoas não representativo ou a propria forma de coletar dados está errada e desbalanceada (isto normalmente quando lidamos com pessoas).

Areas frageis:

1. Historia, não podemos criar mais dados
2. Material caro, é caro criar mais dados
3. Novidade, Novo cenário não sabemos bem o que será o normal

Fragilidades nos dados:

1. Tempo dos dados (há quanto tempo já foi)
2. Seleção dos dados (e.g: dados só foram recolhidos se foram ao médico, pode ser mau)
3. Demografia (e.g: dataset apenas com pessoas que usam oculos)

Annotating data (parte 2)

Primeiro temos de produzir os dados, muitos dados têm que ser anotados manualmente. Normalmento queremos que os modelos se comportem de alguma forma como humanos, para isto precisamos de ter dados anotados por humanos (não é obrigatório mas costuma ser).

Exemplo: image tagging, object annotation.

Muitas vezes temos UI que facilitam o modo como anotamos os dados. Problema, pode levar a BIAS. O anotator é humano, pode se enganar, é preguiçoso, fica cansado...

Antes de começarmos a encher o nosso dataset é importante criar guidelines para todos os que irão anotar os dados --> um manual. Não pode haver duvidas. As anotações TÊM que ser consistentes.

Ainda falta uma coisa, encontrar as pessoas que podem anotar os nossos dados. Os dados precisam de alguma especilização de dominio? Ou nem por isso. Arranjar estudantes... Sempre mantendo a ética.

Semi automatic annotation

Mais uma vez pode ser perigoso, e levar a bias adicional. O próprio computador sugeste as anotações e o anotador vê se está ou não correto.

Crowdsourcing

Crowdsourcing: Usar muito anotatores

MTURK

Atenção: As pessoas não são especilistas, o que implica que a nossa UI deva ser simples e a tarefa de anotação igualmente fácil.

Funciona na prática?

Para algumas tarefas é mais que suficiente, exemplo: Está um elefante na imagem ou não. Mas para algumas tarefas mais complexas obterão maus resultados (depende muito)

Ética

É importante reparar que as pessoas que trabalham nestes estabelicimentos acabam por ter muito poucas compensasões e isso poderá ser um problema ético

Outras maneiras engraçadas de coletar dados

1. CAPTCHA
2. GAMES WITH PURPOSE
3. AMNESTY DECODERS (voluntário)

Annotation as a business (também é uma ideia)

Controlo de qualidade

1. Double annotation (inter annotator agreement) (N annotation)
2. Mix annotation with checks
3. Inspection

e.g

1. Inter anotation score

P(a) = (number of agreements) / (number of items) (não é muito util)

2. Chance agreement probability

k = P(a) - P(e) / 1 - P(e)

P(a) --> estarem efitivamente em acordo P(e) --> probabilidade de estarem em acordo se fizerem tudo ao calhas

< 0.4: No good

0.4 - 0.6: average

0.6 - 0.8: good

0.8 - 1: really good

k é no good? Anotaçao imprecisa, problemas na UI,...

Varições

Cohen k se tivermos 2 annotators

Podes usar estes scores no python: StatsModels

Aula 14: Calculus refresher

Video: https://www.youtube.com/watch?v=Wz6wlkqkd4w

Objective function

Em muitos ML algorithms treinamos um modelos atraves da otimização de funcoes objetivos. Escolhemos parametros que nos e atraves da escolha destes parametros vemos os resultados. Para perceber isto precisamos de perceber matemática (exemplo: gradientes e cenas ...)

Os nossos dados ===> O nosso modelo y = w * x

Como escolher o melhor w?

Naive:

Escolhemos vários valores do w e vemos qual é um que seria bom e ficamos por aqui. Claro que isto é insuficiente e não obterá resultados muito bons

MSE

Mean square error: (1/n) * sum(y - w * x)^2

Para cada valor diferente de w iremos obter um resultado da MSE que serve para avaliar a qualidade do nosso w.

Minimizar o MSE

Otimization task

unconstrained optimization: Encontrar o w que nos dá o minimo (ou máximo) de uma dada função f

MSE: para uma unica variavel, mas como é de esperar em ML temos que generalizar isto para um número arbitrario de dimensões. Aqui é que entra o gradiente. O gradiente é zero então temos um optimum (valor otimo)

Nice function: Se tiver um máximo ou um minimo então a sua derivada é zero.

Como fazer os gradientes?

1. Saber calculo
2. Na pratica nao fazemos isto à mão -> Pytorch e tensorFlow ou Maple, Mathematica.

Aula 15: Optimizing objective functions

Video: https://www.youtube.com/watch?v=J5QF92e7OBY

Objective functions

Minimizar (ou maximizar) uma função objetivo, normalmente com uma loss function e um regularizador

1. loss function: para percebermos os resultados efetivamente que o nosso modelo têm
2. regularizador: para medirmos a complexidade do nosso modelo e perceber se vale a pena manter ou elevar a complexidade dados um certo resultado

Exemplo: minimizar os squared errors

f = (1/N) * sum(w * x - y)^2

Como minimizar ---> GRADIENT

Intuitivamente o gradiente "aponta" parao sitio que queremos (isto se formos exatamente no sentido oposto claro). A ideia é calcular o gradiente em cada ponto e ir descendo

Gradient descent

```
w = init_w()
lr = init_learning_rate()

while (not close to optimum):
    compute gradient
    if gradient is small than threshol we are done
```

```
    else subtract lr * gradient
    repetir
```

Problemas: O learning rate tem que ser bem definido caso contrario podemos entrar em loop. Também podemos ter um learning rate que se adapta automaticamente, normalmente esta solução é poderosa. Solução simples: começar com um lr alto e ir reduzing para não ter problemas. Existem outras soluções que tentam resolver este problema olhando diretamente para o valor do gradiente calculado. Outro problema comum é a existência de minimos/maximos locais. Em que o gradiente acaba por ficar estagnado ... Uma solução é usar o beam search, é inicializar os w em multiplos pontos ao calhas.

Funções convexas: Se a função for convexa e encontrarmos um minimo então este minimo é certamente local. Exemplo: Squared error é convexo. Mas isto não é o caso em todos os algoritmos de ML.

Variante do gradiente descent: STOCASTIC GRADIENT DESCENT. SGD é miuto eficiente nos dias de hoje graças às GPUs que permitem o processamento de código em paralelo muito rapidamente. SGD usa por definição um e apenas um elemento do training set de cada vez, mas podemos criar variantes desta versão (os chamados minibatch) que utilizam multiplas entragas do training set em paralelo.

SGD pseudo

```
iniciar o w e o lr

while
    escolher um x
    calcular o gradiente apenas com este x
    está ok -> acaba
    não está ok -> atualiza o w (w = w - lr * grad)
    repetir
```

Quando terminar o SGD?

solução simples: Escolher um número de iterações solução mais complexa: early stopping

mais sobre early stopping: "In machine learning, early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration."@wikipedia

Aula 16: Linear regression and regularization

Video: https://www.youtube.com/watch?v=WbXWq0evGLY

Minimizar squared errors

least squares aproach for linear regression, usamos o squared error outra vez

aplicar o SGD para o least square loss

f(w) = (w * x - yi)^2

grad(f(w)) = 2 * (w*x - y) * x

Mais uma vez atualizamos o peso com o calculo deste gradiente. (rever o widrow-hoff).

Underfitting | Ideal | Overfit

goodness of fit: o modelo aprendido deve descrever bem os exemplos dos dados de treino

regularização: O modelo deve ser o mais simples possivel

(RIDGE REGRESSION) Como manter o modelo simples?

1. Magnitude dos pesos: Penalizar pesos grandes (L2 regularizer) provavelmente é o regularizer mais conhecido em ML

Ideia: Loss_function + alpha * regularizer

alpha -> serve para controlar o balanço entre overfit e overfit. Se o alpha for mais baixo a penalização é mais pequena e o modelo tem mais risco de overfit, se for muito alto o modelo será muito simples e terá mais porbabilidade de underfit.

Ridge regrssion -> é o modelo com a least_squares_loss + alpha * regularizer.

SGD aplicado na ridge regression: O novo termo é chamado muitas vezes de weight decay.

Existem outras opções para além desta regularização: Lasso regression (usa a L1 norm). Podemos ainda fazer tanto a L1 e a L2 regularização e isso dá a ElasticNet (precisa de mais um parametro adicional). A L1 regularization tende a apresentar soluções esparsas. Ao contrario do L2.

Automaticamente a L1 faz uma feature selection AUTOMATICA (muito bom isto). O que faz com que possamos retirar/excluir estas features. O que faz com que o nosso algoritmo seja muito mais eficiente. Neste caso se o nosso alpha for muito alto, mais features vão tender a desaparecer se for baixo menos features vão desaparecer.

Temos sempre que encontrar o balanço entre utilizade, otimização, underfit e overfit... Aqueles problemas comuns desta área.

Existe uma explicação para o porque do lasso dar coeficientes a zero. https://stats.stackexchange.com/questions/151954/sparsity-in-lasso-and-advantage-over-ridge-statistical-learning

Aula 17: Logistic regression

Video: https://www.youtube.com/watch?v=OMdIU7iNduI

Logistic regression (intro)

Apesar do nome, vamos nos focar em classificação, começando com classificação binária.

Relembra:

score = w * x

Large positive score => x pertence à classe positiva

Large negative score => x pertence à classe negativa

Perto de zero => Não consegue dizer nada com muita confiança

A confiança não é diretamente interpretavel, podemos ter algum modelo que trate disto como probabilidades? Simmm => logistic regression model.

Logistic regression (detailed)

É um método para treinar classificadores lineares que nos devolve outputs probabilisticos. Como é que podemos obter estas probabilidades. Aqui é que entra um grande amigo de ML, o softmax. Só aqui uma nota engraçada, a função SIGMOID é um caso particular da funçáo softmax para classes

binárias.

Sigmoid:

input: x

output: 1 / (1 + exp(-x))

Este é um dos classificadores mais conhecidos em ML, também pode ser chamado de "Maximum entropy classifier" <=> logistic regression.

Intrepretar os resultados

O resultado pode ser chamado de "log odds". Definição de odds -> quanto mais provavel é o resultado ser positivo do que negativo?

odds = p / (1-p)

Só aqui uma nota da função sigmoid, repara que:

1 - sigmoid(x) == sigmoid(-x)

Esta igualdade facilita muito no processo de treino de um modelo

P(label | input) = sigmoid(label * score)

(se passarmos a label para 1 e -1, fica trivial obter o resultado invertido)

Relembrar: Principio da máxima verosimilhança

Nos modelos probabilisticos, podemos treinar modelos selecionando os parametros que atribuem uma alta probabilidade aos dados. No nosso caso estes parametros são o nosso vetor de pesos. Objetivo: ajustar os w para que o output label tenha uma probabilidade alta.

Likelihood function:

L(w) = P(y1 | x1) * P(y2 | x2)... P(ym | xm)

Queremos maximizar esta função, que é o mesmo que minimizar a LOG LOSS (que também é chamado de binary cross entropy loss). (há uma mini demonstração que não vou colocar aqui)

Loss(w,x,y) = log(1 + exp(-y * (w * x)))

Regularização

1. goodness of fit: O que aprendemos deve classificar corretamente os nossos exemplos do training set
2. regularization: O classificador deve-se manter o mais simples possivel. Para isso aplicamos regularização. Para a Logistic regression aplicamos os mesmos regularizadores que a linear regression (L1, L2 ...).

Objetive function (logistic regression)

Há duas...

(1 / N) * loss + (lambda / 2) * regularizer: Controlar o lamda para saber se damos mais ou menos foco ao regularizador.

(C / N) * loss + (1 / 2) * regularizer: Controlar o C para saber se damos mais ou menos foco ao nosso conjunto de treino.

Esta função objetivo é convexa! O gradient descent funciona bem (encontramos o global minimum caso encontremos um minimo)

Notas duvidosas:

Maximizar a nossa likelihood <=> minimizar a log loss

Apesar de se chamar logistic regression, este é um classificador.

O grande aspeto positivo deste modelo é que os resultados têm uma leitura probabilistica, o que pode ser muito positivo em alguns aspetos.

Aula 18: Support vector classifiers

video: https://www.youtube.com/watch?v=BHKKmO_D0U4

Vista geometrica de classificadores lineares

Existe uma linha (ou um plano). Que separa os nossos dados. Existe uma medida, à qual o professor chama de gama. Este gama representa a margem, ou seja o quão bem a nossa linha (ou plano) separa as nossas classes. No fundo serve para avaliar a qualidade dos nossos pesos (w).

Existe um resultado que nos diz que se as margens forem boas o classificador tende a conseguir generalizar melhor (o que acaba por ser bastante intuitivo). Isto é provado matematicamente pelo teorema de minimização de risco estrutural que tem um aspeto feioso

Support vector classifiers (machines)

SVM: Classificadores lineares que selecionam os w que maximizam a margem. A solução depende unicamente pelas instancias que estão nas fronteiras entre as classes. Estas instancias são chamadas de "support vectors". O que isto significa é que as instancias mais interiores (longe da fronteira). Têm zero impacto na decisão do nosso separador. Na prática, isto precisa de mais algumas coisas para fazer com que isto funcione.

1. Às vezes os nossos dados não são perfeitamente separaveis
2. Podemos ter outliars que lixam as nossas fronteiras

O nome da solução que resolve estes problemas é: soft-margin SVM, esta solução permite que existam alguns exemplos que não sejam considerados para o nosso separador final.

SVM <-> objective function

loss = max(0, 1 - y * (w * x))

Esta é chamada a hinge loss. Se virem os plots (que não colocarei aqui por preguicite, este plot é relativamente parecido à nossa log loss de anteriormente). Mas a hinge loss não têm a primeira derivada continua, e a log loss tem. Isto dificulta a aplicação do gradient descent para a hinge loss.

The nonlinear SVM

Podemos generalizar o SVM para utilizar uma kernel function. Quando usamos esta kernel function permitimos que o nosso modelo aprenda modelos que não são lineares. O que nos permite aprender dados com estruturas mais complexas.

Que kernels podemos utilizar?

1. Linear: K(xi,x) = xi * x
2. Quadratic (ou qualquer polinomio): K(xi,x) = (xi * x)^2
3. Radial basis function (RBF): K(xi,x) = exp(-lamda * ||xi-x||^2)

No scikit-learn:

sklearn.svm.LinearSVC: Mais rápido mas menos flexivel => Têm as mesmas limitações que os classificadores lineares comuns (não sabe trabalhar com dados que não são linearmente separaveis) sklearn.svm.SVC: Mais lento mas mais flexivel (permite utilizar os nosso porprios kernals (ou outros built in))

Aula 19: Multiclass linear classifiers

Video: https://www.youtube.com/watch?v=hMq-d969GHM

Until now:

Até agora aprendemos modelos capazes de lidar com duas classes, queremos agora elevar um bocadinho o nosso nivel e aprender modelos capazes de lidar com mais classes.

Ideias:

1. Partir o problema em modelos mais simples, e treinar um modelo binário para cada parte do problema.
2. Modificar diretamente o algoritmo para que seja capaz de lidar com várias classes.

Ideia 1: Reduzir o problema

1. One-versus-rest("long jump"): A ideia é treinar um classificador binário para cada classe, (uma sendo a positiva e as outras todas negativas, e repetir isto para todas as classes). Depois pegar no resultado mais forte

2. one-verus-one("football league"): Treinamos uma classificador para cada par (existem N * (n-1) / 2) classificadores. Quando formos classificar, escolhemos a classe que tiver mais vitorias.

Example

Temos tres frutas: Maça, Laranja, Manga

One versus rest:

1. apple X orange + mango
2. orange X apple + mango
3. mango X appls + orange

One versus one.

1. apple X mango
2. appls X orange
3. orange X mango (nota: apesar de nao se notar neste exemplo, o one versus one tem um crescimento exponencial, o que faz com que geralmente seja um aproach pouco eficaz)

No scikit learn

1. OneVsRestClassifier
2. OneVsoneClassifier

Existem dois aproaches para lidar com multiclass no scikit learn, o primeiro é utilizar os classificadores proprios da biblioteca que já resolvem o problema das multiclasses também. Outro é no caso de quereres lidar com o teu próprio classificador linear, para isso tens que fazer um ad-on com estas funções definidas lá em cima.

Ideia 2: Melhorar a solução

Só aqui uma relembrada de um binary logistic regression. A ideia é converter o nosso score numa probabilidade. A loss function é a loss loss (binary cross entropy loss). Lembram-se qu utilizávamos a nossa amiga sigmoid.

Agora: Passamos para a softmax (que é compativel com multiclass)

Softmax

Pseudo codigo

```
def softmax(scores):
    expscores = np.exp(scores)
    return expscores / sum(expscores)


# Isto pode dar bostique devido a overflows
# A ideia passa então para subtrair o máximo, evitanto overflows

def softmax(scores):
    exp_x = np.exp(x - np.max(x))
    return exp_x / sum(exp_x)
```

Como treinar?

Na binary logistic regression: log loss (binary)

Para multiclass: cross-entropy loss (que é muito semelhante mas usa o softmax em vez da sigmoid)

O resultado é exatamente o mesmo

grande chance de estar correta => loss pequena grande chance de não ser esta class => muita loss

Multiclass logistic regression no scikit learn:

LogisticRegression(multi_class='multinomial')

Aula 20: Boosting

Video: https://www.youtube.com/watch?v=Nj66jH4jrNE

Revisões de ensembles

Vimos que os ensembles podem-nos dar mais confiança e robustez no que toca à qualidade dos nossos modelos (em caso particular vimos as random forestss que são ensembles para decision trees). Também vimos bagging.

Hoje: Construir ensembles iterativamente

Qual é a ideia então?

Assumios que temos um classificador ou um regressor e que ele faz alguns erros no nosso conjunto de treino. A ideia é tentar aprender um modelo que limpe estes erros. Isto é o que chamamos boosting.

Modelos -> reziduals -> Modelos -> Reziduals -> ...

Depois pegamos nos nossos modelos e juntamo-los

Algoritmos:

1. AdaBoost: Especifico para problemas de classificação
2. Gradient Boosting: Generalização do AdaBoost (funciona para classificação e também para regressão)

Ada boost

Step 1: Modelo sobre os dados

Step 2: Ver que pontos errámos -> aumentar o peso destes pontos

Step 3: Repetir quantas vezes achares necessário

Gradient boosting

Ideia: Gradualmente adicionar sub-modelos para minimizar a nossa loss function

Pseudo code

```
F0 = dummyModel()

for m in range(M):
    for x,y in zip(training,set):
        ri = pseudoResidual(yi, Fm-1(xi))

    trainsubmodel hm on the pseudo residuals

    Fm += Fm-1.Append(hm)

return Fm
```

Geralmente usam trees lá dentro

O que é o psudo-residual: É o gradiente negativo da nossa loss function

O que significa adicionar um submodelo ao nosso modelo?

Adicionar o nosso h à nossa loss function, normalmente multiplicamos com um learning rate.

No scikit learn

parametros importantes do gradient boosting:

1. ensemble size (nr_estimators)
2. learning rate

No random forest o nr_estimators não implica overfitting, mas no gradient boosting este parametro afeta o nosso overfitting. Isto porque estamos a aperfeiçoar o nosso modelo aos dados diretamente.

Implementações do gradient boosting

1. scikitlearn (python)
2. XGBoost (várias linguagens)
3. H20 (Várias linguagens)

Gradient boosting

Este algoritmo começou a ficar famoso nos ultimos anos, conseguem ter resultados muito bons nas competições de AI (exemplo: Kaggle)

Aula 21: Overview of Evaluation Methodology in ML

Video: https://www.youtube.com/watch?v=7bMbolxKeic

Porque avaliar?

É normal que os nossos modelos falhem, é importante percebermos a sua verdadeira performance, e é importante destinguir os nossos modelos no contexto em que são aplicados. Não podemos permitir que uma carro automático falhe de vez em quando, mas podemos permitir que um detetor de spam falhe 2 ou 3 vezes a cada 100 emails. Qual é a perforamance que diz que um modelo é util?.

Temos o nosso training set e o nosso test set (ainda temos um validation set). O validation set serve para medirmos a performance do modelo DURANTE o processo de treino. Este procedimento é chamado de cross validation.

Como avaliar?

1. Avaliação intrinseca: medir a nossa performance em isolação utilizando algumas medidas para computar a nossa métrica automáticamente.

2. Avaliação extrinseca: Já mudei o meu previsor, como é que isto está a afetar os resultados (estou a receber mais dinheiro, mais cliques, está a funcionar?)

Diferentes dominios

É importante notar que as métricas de avaliação dependem de cada dominio, portanto também é importante repararmos o que está a ser utilizado no mercado, se faz ou não sentido. Trabalhar com pessoas do meio. Temos que PERCEBER os nossos dados. Isto pode acabar por ser dificil dependendo das áreas.

Algumas aplicações até utilizam HUMANOS como classificadores, ter pesssoas a avaliar o nosso modelo (exemplo: tradução). Claro que usar humanos é sempre dificil, caro e não são livres de erro.

Aula 22: Evaluation of Classifiers and Regressors

Video: https://www.youtube.com/watch?v=d-Bgg-2CRTw

Setup tipico

Avaliação intrinseca: Quão bons os outputs são

(pode ser feito olhando simplesmente para a ground_truth)

Métricas comuns para modelos de classificação

1. accuracy = correct / total
2. error = incorrect / total
3. Confusion matrix (aqui entra o conceito de TP, FP, TN, FN), isto é crucial para alguns problemas (exemplo: problemas com classes não balanceadas) e levanos a algumas medidas menos comuns -> Precision, Recall
4. Precision = TP / (TP + FP)
5. Recall = TP / (TP + FN)
6. F1 score = (2 * P * R) / (P + R)
7. Também é comum utilizar outras medidas -> Por exemplo na medicina existe a sensibilidade e a especificidade
8. Sensibilidade = TP / (TP + FN) (==recall)
9. Especificidade = TN / (TN + FP)
10. Ainda existem os rate -> True postivie rate, e false positive rate
11. TPR = TP / (TP + FN)
12. FPR = FP / (FP + TN)

Métricas comuns para modelos de regressão

Aqui não podemos ver se falharmos ou não, dada a continuade do espaço de solução seria basicamente impossivel acertar nos valores, para isso aparecem outras medidas: MSE e MAE

1. MSE: Mean squared error -> Se tivermos uma instância que falha muito esta métrica dará muita importância (devido ao quadrado)
2. MAE: Mean absolute error -> Se tivermos outliars não ligamos tanto

Temos ainda um problema das escalas. Para resolver este problema aparece o coefficient de determinação R2, esta métrica não depende da escala (se tiver valor 1 -> score perfeito, se tiver valor 0-> regressor burro). Vê na net a formula desta medida dado que é chata de colocar aqui.
https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html

Aula 23: Evaluating Scorers and Rankers

Video: https://www.youtube.com/watch?v=4Mltd4tE0Q0

Importância do dominio

Por exemplo, queremos encontrar tumores maligonos, para isso temos modelos que fazem essa separação. O importante para nós é encontrar os tumores maligonos. Ou seja queremos ter um recall grande. De modo a evitar ter pessoas com cancro mas que não sabem que têm cancro. Para isso podemos definir um treshold que nos permite dar mais importancia ou ao recall ou à precisão, neste caso queremos um modelo mais exigente que seja capaz de encontrar o máximo número de TP possivel e que quer minimizar os FN. Sem se importar muito que existam FP. Apesar de ser muito mau dizer a alguém que tem cancro mesmo sem ter é o preço a pagar.

Curvas

Para isso é importante perceber a precision/recall curve. Um sistema perfeito têm sempre a precisão e o recall no máximo. Para além desta curva também temos a ROC (receiver operating characteristic) curve, que se comporta de maniera ligeiramente diferente

Mais informação: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

Aula 24: Introduction to Neural Networks

Video: https://www.youtube.com/watch?v=zsFzV91ZGrY

Deep learning models

É a area que está mais na moda. A ideia das NN é fazer uma abstração dos inputs de modo a aprender os padrões que se escondem por detrás dos dados. E faz isto tudo automaticamente.

O grande aspeto positivo é conseguirmos simplificar o processo de feature eng. , antigamente tinhamos de fazer coisas complicadas nos dados para podermos tirar algum tipo de informação. Com o aparecimento destes modelos mais complexos conseguimos lidar quase com os dados 'crus'.

Pros:

1. Consegue captar relações complexas
2. São muito bons com dados 'noisy'
3. Permitem resolver problemas muito complicados e.g: tradução

Cons:

1. Computacionalmente pesados
2. Careful tweaking
3. Menos estáveis
4. Os modelos gerados são complexos -> exige mais dados
5. Resultados muitas vezes de black box

Estes modelos dominam areas com dados complexos (e.g: som ou imagens) mas quando lidamos com dados bem definidos normalmente não são utilizados.

Reaparecimento das NN

1. Mais dados
2. GPUs

3. Mais computadores
4. Melhores estratégias (modelos em si)

Hoje

NN básicas para classificação e regressão

Modelos lineares (relembrar)

Limitações: Os modelos lineares náo conseguem lidar com relações que não são linearmente separaveis. O grande problema do XOR. Contudo este problema pode ser resolvido se fizermos uma transformação no conjunto de dados. A isto chamamos uma transformação não linear. Dada que esta mudança é não linear o modelo final também não será linear.

Esta é a ideia das NN. Automaticamente mudarem o espaço do dataset, de modo a que o modelo consiga aprender relações não lineares no conjunto de dados original.

FFNN: Feed forward neural network

Este é um modelo de multiplas camadas e também é conhecido por multilayer perceptron (MLP)

Estes modelos consistem em camadas stackadas, e existem várias hidden units. A ultima camada é chamada de output layer. (Podem ser usadas para tarefa de classificação ou regressão).

Cada hidden unit tem uma parametrização diferente.

Implemetação

A base é feita com 'dot products' contudo estes modelos são mais que um dot product. Os calculos podem ser mais otimizados com a notação matricial. Na prática daria para fazer tudo individualmente, mas este processo é mais lento.

Funções de ativação

1. Sigmoid
2. tanh
3. ReLU

A escolha da função de ativação é por si só um hiperparametro e tem que ser 'tunado'.

Só aqui uma notita: A função sigmoid é especifica para outputs binários, o que às vezes não é o nosso caso. A função genérica com este formato é a softmax. Relembra outra coisa: Quando implementas a softmax tens o problema do overflow, para resolver isso fazes o tal shift que já falamso à algumas aulas atrás.

Se o output for 'regression', qual é a função de ativação que devemos usar? Normalmente não utilizamos nenhuma :>

NN no scikit learn

1. sklearn.neural_network.MLPClassifier
2. sklearn.neurral_network.MLPRegressor

Contudo estes modelos não nos dão tanta liberade como isso tudo, para isso teremos de utilizar outras bibliotecas (torch, kura, tensorflow ...)

Poder das NN

Universal aproximation theorem: FFNN podem aproximar qualquer função matematica (isto é verdade até para apenas uma hidden layer). Contudo isto pode não ter valro prático, se só tiveres uma hidden layer terá um grande crescimento na vertical...

Area de pesquisa dos dias de hoje: Porque é que é mais facil aprender em profundidade lateral (maior numero de hidden layers, ao em vez de profundidade na vertical??). As primeiras hidden layers lidão com a informação no nivel mais baixo. As hidden layers mais à frente lidam com dados mais abstraidos.

Aula 25: Training Neural Networks

Video: https://www.youtube.com/watch?v=OfE5U8lVIb0

O que significa treinar NN?

É em tudo muito semlhante aos modelos lineares, temos uma função que queremos otimizar (objective function). Na altura vimos um algoritmo que nos permitia otimziar os paremtros dos modelos lineares -> SGD

SGD: stocastic gradient descent

Apesar de ser um algoritmo utilizado em modelos lineares, é compative com modelos não lineares como as NN. Repetimos este processo um número de vezes suficientes e a magia acontece, no caso das NN, os pesos de cada unidade serão ajustados o melhor possivel

Relembra algumas losses:

1. Log loss (binary classification)
2. Cross entropy (multi classification)
3. Squared error (regression)

Output da NN

Temos os nossos pesos W

Temos o nosso input X

Calcular o z = dot(W,x)

Calcular o h = sigmoid(z) (sigmoid pode ser qualquer função de ativação)

Repetir isto para cada uma das layers, utilizando o output da layer anterior (h) como input (x) para a layer seguinte.

Este processo de repetir as contas faz com que cada layer no fundo seja uma computação aninhada noutra (composição de funções). Isto faz com que seja possivel utilzar a regra da cadeia para calcular os gradientes.

Regra da cadeia:

grad(f(g(x))) = grad(f(g)) * grad(g(x))

Isto faz com que seja possivel termos um grafo de computação. Para calcularmos as coisas mais facilmente.

Nota: Os gradientes são calculados em respeito à loss. (Ver o minuto 10:57 para teres informação visual).

Isto é CRUCIAL para fazer a backpropagation. Cada gradiente reutiliza passos do gradiente anterior para fazer as suas calculações. A cena boa é que as bibliotacas já resolvem estes problemas por nós.

Aula 26: Neural Network Practicalities

Video: https://www.youtube.com/watch?v=uDkSjwlCnis

NN: Tempos

Estes modelos são mais pesados que os modelos lineares, podem demorar horas, dias, semanas ... a correr dependendo do tamanho do modelo e do dataset. Isto porque requer muita matemática. Por isso é CRUCIAL utilizar implementações eficientes que lidam com a matematica do melhor modo que o ser humano sabe. Para isso é melhor trabalhar com bibliotecas, que já tratam desta implementação por nós

1. Google: TensorFlow
2. Facebook: PyTorch
3. Microsoft: CNTK

Estas bibliotecas já lidam com bprop, regularization ...

Ainda temos o Keras, que é mais high level. Hoje em dia o keras é uma parte do tensorflow.

Cuda API: Falar com GPU

Coding example with Keras:

```
keras_model = Sequential()
```

```
n_hidden = 3

keras_model.add(Dense(inputDim, outputDim))
keras_model.add(ACTIVATION_FUNC)

keras_model.add(Dense(outputDim, 1))
keras_model.add(ACTIVATION_FUNC)

# Definir a loss e o optimzer
keras_model.compile(binary_cross_entropy, sgd)
```

Relembrar: Optimizer => algoritmo que trata de atualizar os nossos pesos.

Feature preprocessing

As NN são muito sensiveis ao pre processamento de features, a escala das features também afetam muito, é boa prática aplicar scaling nas features se forem para alimentar estes modelos. (exemplo: min-max scaling)

Batch

Processar uma feature de cada vez não é bom. Se fizermos uma de cada vez não estamos a aproveitar o paralelismo desponivel no nosso computador. Para aproveitar isto temos que definir o batch_size. Isto é uma variante do SGD clássico, chamado o SGD com minibatch.

Otimizar NN

É comum inicializarmos os pesos das nossas NN com valores aleatórios. Isto dá nos mais um trunfo -> multiple initizalizations. Isto permite obter vários modelos, depois é uma questão de escolher o melhor.

Também é importante escolher um learning rate adequado, caso contrario poderemos não conseguir obter valores uteis. (nada de novo honestamente).

Mais uma vez => lidar com overfit

1. Regularizers => como antes (l1, l2... adicionados na loss function)

2. early stopping => começar a perceber quando o nosso validation set não apresent melhorias, e terminar.
3. dropout => desconectar unidades durante algumas etapas do treino, normalmente a cada batch.
4. data augmantation: Adicionar um noise ao conjunto de dados orignal. Segue o mesmo objetivo do dropout

Early stopping no Keras:

```
from keras import callbacks

cb = callback.EarlyStopping(monitor='val-loss',min_delta=0,patience=10, verbose = 0, mode='auto')

...

model.fit(...,callcack=[cb])
```

Dropout

Importância do droupout: Não ter cada uma das unidades a resolver um problema. Quando desativamos as unidades do modelo estamos a passar a responsabilidade para todo o modelo e não apenas uma parte dele.

Aula 27: Introduction to Convolutional Neural

Video: https://www.youtube.com/watch?v=m1cJyowhG_s

Motivação

Hoje em dia são as soluções padrões para lidar com problemas de classificação de imagens

Como representar uma imagem?

Uma imagem é um conjunto de pixeis, cada pixel representa uma cor. Existem muitas paletes de cores disponiveis, as mais comuns sao o RGB ou o black and white.

Existem muitos formatos

1. Direct formats: BMP ou TIFF: Guardam tudo, o que pode ser muito eficiente.
2. Compressed lossless formats: PNG: Guardam tudo de forma eficiente
3. Compressed lossy format: JPEG: Guargam tudo com um pouco de perda de informação

Existem ainda alguns formatos especificos para cada dominio (exemplo: imagens médicas)

Como contruir um Modelo que seja capaz dee lidar com imagens?

Modelos lineares: Não conseguem captar as interações entre as features ... O que aqui é completamente importante

Decision trees (and random forests): Podem apanhar estas relações, mas de modo muito ineficaz, teriam de ter um tamanho muitérrimo grande dada a grande explosão combinatória deste problema.

Soluções de antigamente:

Fazer feature extraction em esteróides para imagens. Tentamos encontrar padrões que aparecem nas imagens e depois ficamos com estes padrões para nós. Exemplo: SIFT. Por exemplo para texto temos o BAG OF WORDS.

Contudo isto não é escalável e dá muito trabalho.

Então e as NN? Elas conseguem aprender as abstrações autaticamente, isto não funciona?

Sim funciona, mas o número de parametros também não é escalavel. Exemplo: Para lidar com uma imagem de 200x200 com 40 hidden units fica com aproximadamente 2 biliões de parametros ... E ainda há um problema gigante. Não consegue lidar com SHIFTS. E lidar com shifts em imagens parece algo que é importante de resolver.

CNNs

Ideia: Apanhar padrões, também terá uma 'stacked architecture' e também será capaz de lidar com funções não lineares.

Na próxma aula:

Building blocks: filters, kernels, pooling Truques: Data augmentation, transfer learning, interpretation

Aula 28: Building Blocks of Convolutional Neural Networks

Video: https://www.youtube.com/watch?v=Kr1WP1eFPmE

CNN

Solução para image-based problems

Componenetes:

1. Convolutional filter (Kernel)
2. Pooling
3. Fully connected or dense layers (like a FFNN)
4. Residual connections / Normalizations

Filtros (convolutions/kernel)

Estas estruturas servem para detetar padrões, a ideia é varrer a imagem com o nosso filtro e fazer alguns calculos com o nosso filtro. Podemos ter multiplos filtros na mesma CNN. Os parametros dos filtros são aprendidos também. Isto porque não sabemos que padrões é que existem até ao treino.

Stride: De quanto em quanto é que aplicamos o nosso filtro. Exemplo stride = (1,1).

Flatten the output: Para aplicar a softmax no fim é importante fazermos o flat das nossas estruturas. Então o que fazemos é criar uma "Dense layer" no final que permite compatibilidade com o softmax.

Dimensão da convulução

1. 1D : usado para speech signal

2. 2D : usado para imagens
3. 3D : usado para videos

Claro que há mais aplicações sem ser estas, estes são apenas exemplos.

Poder das CNNs

O grande poder das Cnn aparece quando combinamos várias iterações de aplicar estes filtros

Pooling (sub sampling)

O exemplo que nos dão é o max pooling, o objetivo do pooling é reduzir o espaço de features. Permite que o modelo consiga fazer abstrações mais facilmente e ao mesmo tempo otimiza.

Nos dias de hoje:

Hoje em dia as Cnns ainda são muito utilizadas (apesar de estarem a perder terreno para os transformobots...) contudo para funcionarem muito bem as cnns precisam de ser muito "deep". Geralmente mais de 100 hidden layers.

Contudo ser deep faz com que apareça um novo problema: Vanishing gradients and exploding gradients. Para resolver isto criamos uma coisa que é a coneção residual.

Residual connections: A ideia é simplesmente é simplemente criar uma ligação adicional entre que "salta" uma camada.

Outra solução é a batch normalization. O objetivo é simplesmente standardizar cada dimensão em cada batch.

Aula 29: CNN Tricks

Video: https://www.youtube.com/watch?v=EGqqbSE94L0

Aula de hoje:

1. Modelos pre-treinados e transfer learning
2. data augmentation
3. interpreting Cnns

Pre-trained models

Transfer learning: Queremos aproveitar o que já fizemos num dado modelo anteriormente e transferir o conhecimento para um nobo modelo. Normalmente isto é feito com a reutuilização dos pesos encontrados por um dado modelo. Não só torna o processo de construir novos modelos mais rápido, como permite construir modelos em dominios que têm poucos dados.

Transfer learning in vision: ImageNet é um dataset com todo o tipo de imagens, é muito genérico, carros, casas, comida, ... Com este dataset podemos criar um sistema muito genérico e pouco especializado. Mas depois podemos aproveitar este modelo e alimentá-lo com um dataset mais especifico.

Como o modelo está pre-treinado nem nos precisamos de preocupar em treina-lo de novo.

Duas formas de usar modelos pre treinados:

1. Utilizar um modelo pre-treinado com pesos congelados. (não deixamos atualizar os novos pesos)
2. Utilizar um modelo pre-treinado mas permitimos que os pesos sejam atualizados.

Overfitting nas Cnns

Cnns são modelos muito complexos e profundos, por isso têm uma grande facilidade de entrar em regime de overfitting.

Para evitar overfitting basta -> regularization, early stopping and dropout

Mas existe outra maneira -> data augmentation (aplicar random noising, shearing, rotating, darkening, flipping...). Isto também é uma forma de parametrização. (o quanto de augmentation é que podemos aplicar)

Intrepertar as Cnns (desenhar os filtros)

- Feature visualization

- Perceber as decisões do modelo. Exemplo: Porque é que a rede acha que

Aula 30: Introduction to Unsupervised Learning

Video: https://www.youtube.com/watch?v=trvsx-KHUsM

Unsupervised learning

Agora vamos mudar um pouco o nsoso paradigma de aprendizagem automática. Anteriormente os nossos dados têm uma label (Y) que nos diz qual é o valor real/concreta, do que estamos a tentar prever.

A isto chamamos supervised learning. Existem ainda mais dois modos de treinar modelos de ML.

1. Unsupervised => Explorar os dados, sumarizar, visualizar ...
2. Semi supervised => Nem todos os nossos dados estão rotulados. Mas mesmo assim queremos fazer previsões.

Tecnicas de aprendizagem não supervisionada

1. clustering
2. distribuições estatisticas
3. aprender representações dos dados

Clustering:

Temos dados, depois tentamos perceber que grupos é que existem dentro dos nossos dados.

Modeling distributions:

Temos um dataset que queremos visualizar a distribuição, encontrar regiões mais ou menos provaveis, detetar outliars.

Aprender novas represetnações.

Exemplo: reduzir dimensões

Porque é que é importante?

1. visualizar e perceber
2. reduzir o tamanho
3. fazer com que outros ML alghorithms sejam mais rápidos ou mais fáceis

Aula 31: Overview of Clustering Methods

Video: https://www.youtube.com/watch?v=j6DaFp8vJE4

Clustering

O objetivo é encontrar grupos no conjunto de dados.

Aqui é que aparece o clustering

Para formar os clusters => similarity measure or distance functions

A métrica mais usada é a distância euclideana. (no nosso curso assumimos sempre que as features já estão representadas de forma numérica)

Clustering

1. flat => apenas grupos
2. hierachical => grupos e subgrupos

Flat clustering

1. Representativos dos clusters

2. k-means

3. k-medoids
4. mean
5. shift

Depois de encontrar os representativos dos clusters já podemos fazer uma avaliação

2. Perspetiva probabilistica
3. Encontrar regiões densas => DBSCAN

Hierachical clustering

Algoritmo: agglomerative => ir agrupando pares de registos aos poucos, os dois mais juntinhos ficam, depois serão tratados como um só. Repetir isto até que todos estejam num cluster.

Aula 32: Evaluation of Clustering Methods

Video: https://www.youtube.com/watch?v=dUaYcHNs--w

Como avaliar?

clustering é normalmente uma tarefa dificil. Comparando com tarefas de classificação, conseguimos perceber se estamos a ir numa boa direção ou não. Mas nestes algoritmos é mais dificil. Pode não existir uma resposta correta.

Como é que podemos então perceber se estamos a ir bem ou mal?

1. Internal evaluation methods => Os nossos clusters são coesos e estão bem separados?
2. External evaluation => Quão bem o clusters fez o que eu queria que fizesse?

Silhoutte score

si = (bi - ai) / max(ai,bi)

Onde:

1. ai => É a distância média do ponto a todos os pontos que foram colocados no mesmo cluster.
2. bi => Distância ao ponto mais próxima mas que seja de outro cluster.

Purity score

O quão "puro" é o nosso cluster?

Quanto é que a majority class domina o nosso cluster?

Este score tem uma batota: Se colocares todos os pontos num cluster diferente ficas com purity score de 1 para todos os pontos

https://towardsdatascience.com/evaluation-metrics-for-clustering-models-5dde821dd6cd

Inverse purity score

Para fazer batota basta simplesmente meter todos os individuos no mesmo cluster.

F-Score

Balancear os objetivos do "purity" e do "inverse purity". (não é a mesma coisa que o Fscore dos classificadores)

Aula 33: K-means Clustering

Video: https://www.youtube.com/watch?v=Ge6Djzgh2ac

K-means

1. Deve ser o algoritmo de cluster mais famoso
2. Ideia informal: Encontrar os K clusters tal que estão mais perto do centroid (vetor médio)

Loss function

residual sum of squares

L(S) = ...

O nosso objetivo é encontrar a partição S que minimiza esta loss function

Mas temos um problema, isso é NP hard. Ou seja não é fazivel encontrar esta solução => Surge uma aproximação

Lloyd algorithm => esta é a aproximação, e é o que é utilizado no K-means

Lloyd algorithm

```
1. Inicilizar os centroids ao calhas (centroids-> mew)
2. Cada ponto vai para o cluster S que está mais perto do centroid mew_i
3. Recomputar os centroids para cada S, como é que isto se faz? Basta calcular a média dos pontos desse centroid.
4. Repetir 2 e 3 até não convergir

return S = {S1, ..., Sk}
```

Converge?

Este algoritmo termina sempre?

SIMMMM

Explicação:

A cada iteração, ou vamos reduzir a loss function L ou não vamos mudar nada, se não mudar nada é porque convergiu.

Problema?

Apesar de convergir, ele pode não convergir sempre para a mesma solução. Depende da primeira inicialização aleatória. Isto invoca outro problema... Não encontra a solução otima.

Como minimizar este problema => BEAM SEARCH || K-MEANS++

BEAM SEARCH

É trivial, basta colocar vários e escolher o melhor

K-MEANS++

Os centroids iniciais são escolhidos de acordo com pontos do dataset, a ideia é escolher pontos afastados entre si

Implementações do k-means

1. scikit-learn => KMeans, MiniBatchKMeans
2. Spark
3. Dask

Nota: MiniBatchMeans => trabalha apenas com um subset do dataset , é mais rápido.

Parametros a tunar

1. K
2. Iniciliação dos centroids (estrategia standard => KMeans++)
3. Numero de vezes a correr
4. Como parar => numero de iterações ou convergir?

Uma dificuldade

Quantos clusters é que existem???

Apesar de parecer simples esta pergunta não é nada fácil de responder

Existe uma cursa => RSS (residual sum of squares). Esta curva é a relação entre numero de clusters e a loss. Para perceberes o problema, por exemplo se tiveres o mesmo numero de pontos de clusters tens uma loss de 0 mas isso não tem valor informativo.

Então => Como saber o numero de clusters=

1. Através do dominio
2. Heuristicas
3. Silhoutte score
4. Penalizar a loss function com o número de clusters => AIC ou BIC (tipo regualização)

Existe ainda outro método => elbow method, ver a curva RSS e ver onde é que está o nosso cotovelo.

O que podemos retirar do K-means?

1. Ver quais são as amostras representativas => Perto do cluster
2. Outras coisas ...

Coisas extra:

1. Atenção ao scaling
2. Atenção à medida de distância

Tens ainda o GMM (gaussian mixture model, TODO <= ver isto)

Aula 34: DBSCAN Clustering

Video: https://www.youtube.com/watch?v=yHGYg0Z3a7o

Comparações com o K-means

1. Aqui os clusters não têm representativos (centroids)

2. Aqui os clusters podem ter qualquer formm, não apenas circulares

Conceitos no DBSCAN

1. Métrica de distância (à escolha)
2. Ver se um ponto é 'directly reachable'. (a distância não pode ultrapaddar um treshhold)
3. Core point: P é core point se existirem min_samples - 1 pontos que são directlyy reachable
4. Um ponto que não é alcançado por ninguém é chamado de NOISE

Algoritmo (intuitivamente)

1. Encontrar core points
2. Conectar os core points
3. Pontos que não são core points ficam nesse cluster se forem reachable por esse cluster
4. Os outros são noise

Scikit learn

sklearn,cluster.DBSCAN

parametros:

1. eps: Distância
2. Min_samples
3. Metric (função distância)

Todos os hiperparametros afetmam muitos os resultados do algoritmo e por isso os seus resultados acabam por não ser muito estáveis.

Pros & Cons

pros:

1. não tens que escolher o numer ode clusters

2. nao está limitado a formas de clusters especificas
3. é compativel com qualquer métrica de distância
4. elimina os outliars automaticamente

cons:

1. Varia muito com os hiperparametros
2. Não funciona bem se clusters diferentes tiverem densidades muito diferentes

Aula 35: Introduction to Dimensionality Reduction

Video: https://www.youtube.com/watch?v=synM02mm9Wg

Nota: isto ainda é considerado unsupervised learning

Para que serve?

1. Melhorar interpretação
2. Descobriar alguns padrões => Representation learning
3. Aprender algumas coisas sobre o dataset

Dimensionality reduction

Queremos reduzir a dimensionalidade do dataset, temos um dadaset com muitas colunas aqui o objetivo é reduzir o número de colunas. Claro que o nosso objetivo é fazer isto sem perda de informação.

Porque é que queremos fazer isto?

1. Mais fácil interpretar um dataset com menos dimensões
2. Reduzir o overfitting => Largar a maldição da dimensionalidade
3. Otimização dos próprios algoritmos

PCA: Principal component analysis

Objetivo => ter um novo espaço de dimensões

Intuitivamente => Tentar encontrar a direção na qual o nosso dataset está a ir, isto dá-nos um dos vetores => PRIMEIRO PRINCIPAL COMPONENT || Depois removemos este PC e repetimos o primeiro processo. Agora temos o SEGUNDO PRINCIPAL COMPONENT ...

Podemos repetir isto o número de vezes que queremos

Temos um novo sistema de dimensões, mas para que é que isto serve?

Podemos reduzir as dimensões para o número que quisermos. Os principal components estão ordenados por ordem de relevância, e são escolhidos com a variancia da dimensão

Behind PCA: Singular value decomposition

==========

1. PCA pode ser implementado com uma simples fatorização de matrizes com uma técnica chamada SVD (singular value decomposition)

M = U EPSILON V*

U tem dimensão mxm

EPISLON tem dimensão mxn

V* tem dimensão nxn

Contudo isto é muito problemático e não escala para matrizes muito grandes como é o nosso caso

==========

2. Implementações nas bibliotecas => low-rank factorization

é um bocadinho mais dificil, para usar no scikit learn => truncatedSVD

==========

3. Outra forma => fatorização das matrizes com SGD (stocastic gradient descent)

Loss function: alternating least squares

==========

Aula 36: t-SNE

Video: https://www.youtube.com/watch?v=RJVL80Gg3lA

Técnicas de vizualização de dados com t-SNE

Temos X1 , X2 , ... , XN objetos de alta dimensão. O que queremos agora é visulizar de alguma forma as relações entre estes objetos.

Existem muitas maneiras de fazer isto, normalmente passamos por dar colapse em dimensões mais pequenas.

Ideia simples: Fazer um mapping das dimensões mais altas par dimensões mais baixas => PCA

PCA: Maximizar variancia, preservar distância que são muito grandes, ou seja se forem muito diferentes o PCA é bom a dizer que são efetivamente, mas não consegue captar muito bem as semelhanças.

As pessoas começaram a perceber que o PCA é muito limitado no aspeto de vizualização e tentaram aparecer com algumas soluções novas => Isomap, Locally Linear Embedding

Lobally linear Embedding => Tem uma coisas má, muitos pontos acabam colapsados na origem.

O t-SNE é muito baseado no locally linear embedding

t-SNE: t-distributed stochastic negihtbor embedding

1. Calcular Pij's

Isto é um bocado confuso vou duplicar e passar

Aula 37: Matrix Factorization for Recommender Systems

Video: https://www.youtube.com/watch?v=bmWPzffqo9k

Recommender system

1. Estão por todo o lado
2. Têm muito valor no mercado hoje em dia

Formalizar o problema

Pode ser visto de muitas formas:

1. binary classification: Um item X é relevante ou não para um user Y
2. regressão: Quanto é que o user Y gosta do item X
3. ranking: sorting the item for each user

Tipos de recommender systems

1. content-based: As recomendações são feitas consoante as featues de items já selecionados => Exige colocar features nos items

2. collaborative filtering: Recomendar items selecionados com users com histórico semelhante => Exige ter um histórico

Vamos-nos focar no collaborative filtering

Ideia: Construir representações dos utilizadores e dos items exclusivamente recorrendo à interação

1. explicit feedback: O user diz explicitamente quanto gostou ou não do item => star rating

2. implicit feedback: Vemos as ações que o user teve com o item para avaliar o quanto gostou ou não, se ouviu muitas vezes aquelas música é porque provavelmente gosta da musica.

cold start problem

O que fazer com NOVOS users e novos items, não há história portanto não sabemos nada.

Mudança

Os gostos do próprio utilizador podem mudar, e o próprio mundo também altera (exemplo: mundo da moda)

Matrix factorization in collaborative filtering

Objetivo: find low-rank matrices (user and item representation) so that we can find the missing cells

Prever uma missing cell: Rating an unseen item

Avaliação dos sistemas de recomendação

Caso simples: Utilizar explicit feedback como teste

Avaliação externa: Quanto melhor funcionou o meu sistema

Aula 38: Word embeddings

Video: https://www.youtube.com/watch?v=vEOhiyoBUqY

Relembrar: Tarefas de classificação de documentos

Exemplo:

"Estou ansioso que a Leonor venha!" - Positive

"Não quero nada ter este exame." - Negative

Deep learning in NLP

Cada vez mais o deep learning está a ficar popular em tarefas de NLP. É bom a fazer tarefas complexas, é compativel com transfer learning

Transfer learning

Aproveitar modelos aprendidos anteriormente para alimentar ou pre-treinar novos modelos

Representar documentos em machine learning

A mais fácil e comum => bag-of-words, permite representar, esta representação apesar de util não é muito otima, isto porque cada palavra adiciona uma coluna (uma nova dimensão). E para além disto não capta semelhanças entre palavras exemplo: batata vs cenoura == batata vs correr.

Para resolver estes problemas aparecem os word embeddings.

Training word embeddings

end-to-end training: Aprendemos embedingss especializados

pre-training: Aprendemos embeddings genéricos para depois usar em diferentes tarefas

Embeddings no KERAS

```
model = Sequential()
model.add(Embedding(1000,64))

# O que é que isto significa?
```

```
# Temos um vocabulário com 1000 palavras e
# queremos embeddings de 64 dimensões
```

Ideia no pre-training word embeddings: As palavras que têm um comportamento semelhante, contextos semelhantes <=> palavras semelhanças. Exemplo: Café e chá.

Objetivo: Tentar criar embeddings que captem estas semelhanças

co-occurrence matrix: Matriz que vê as palavras que aparecem perto umas das outras. Agora a ideia é fatorizar esta matriz

GloVe: matrix-based word embedding training method

1. Minimizar uma loss function com a matriz de coocurrence
2. Palavras que aparecem menos vezes têm pesos menores.

Depois de ter as embeddings temos o que precisamos para perceber as semelhanças entre as palavras. Podemos ainda usar o t-SNE e o PCA para visualizar estas semelhanças!

Aula 39: Introduction to ML for Sequences

Video: https://www.youtube.com/watch?v=zba5BHuzN8c

Introdução

Como lidar com uma sequência de dados tanto no input como no output

Como representar dados sequênciais com modelos de machine learning?

Classificação ou regressão

Tarefas

1. Sequencias de caracteres
2. Sequencias de som

3. Sequencia de ADN
4. ...

Extrapolação

1. Usar dados anteriores para prever sequencias do futuro

Transduction

1. Sequencia dada como input e retornar uma nova sequência, exemplo => tradução

Basic ML for sequences

Relembrar: Sequencias de texto => bag of words

Bag of words também pode ser usado para outro tipo de sequências.

Convoluções => varrer a sequência para tentar analisar os padrões

Exemplo: Prever load da eletricidade

Como construir um modelo que prevê a carga de eletricidade utilizada daqui para a frente?

Utilizamos dados do passado para prever os próximos valores => Autoregressive model

Autoregressive model

Um modelo que opera sequencialmente, usa o histórico anterior para prever as próximas sequências.

Como treinar autoregressive models? => teacher forcing

gold standard sequence

exposure bias => o modelo foi treinado com valores super corretos, quando somos testados podemos entrar em regime desconhecido, porque nos afastámos dos valores corretos (reais).

Aula 40: Recurrent Neural Networks

Video: https://www.youtube.com/watch?v=4JA0jXRsWKY

Ideias chave

RNN não são uma coisa exclusivamente, é uma familia de arquiteturas para processar sequências com machine learning.

Operação básica:

-> recebemos uma sequência de inputs, temos um cvv (continuos value vector) e devolvemos uma classificação, regressão ou uma nova sequência.

Exemplo:

INPUT: This is not a very good

Percorrer o input token a token (no caso de ser um texto) e ir alimentanto informação do passado ao token seguinte.

Alternativa: averagind the states, ir a todos os tokens e fazer uma média entre os estados de cada token para devolver o output

Aplicações:

1. Name entity recognition
2. Sequence generation

Como funcionam as RNNs?

RNN (simple RNN // Elman RNN)

É muito semelhante a uma FFNN, não é muito usada devido ao problema do vanishing gradient. É muito complicado fazer com que o modelo se lemmbre de coisas que aconteceram no inicio da sequência. A solução para este problema é feita já noutras RNNs -> gating

gating

Permite controlar a sequencia de informação de forma mais cuidada

LSTM (long short term memory)

É uma RRN que usa gates, de resto é muito semelhante a uma RNN normal

Função de ativação: Sigmoid

De onde vem o long-term: forget gate - cancelar o estado anterior ou deixar passar?

GRU: Gated recurrent units

É mais simples e mais eficiente para so computadores, é um concurrente da LSTM

Aula 41: Introduction to Transformers

Video: https://www.youtube.com/watch?v=3P_YqkIAOVQ

Transformers: Uma familia de arquiteturas

Palavras chave:

1. Mecanismo de atenção
2. Encoders => Sumarizar o input
3. Decoders => Gerar o output

Podemos ter só o decoder => Language model

Podemor ter só o encoder => Representar o modelo

Depois podemos encaixar este blocos em modelos diferentes

Existem ainda mais truques (camadas intermédias): Layer normalization, residual connection => importantes para se quisermos ter um modelo muito profundo.

Nota:

Infelizemente o professor não fala muito disto portanto vou deixar isto para depois