Treinamento de modelos de aprendizado de máquina

Utilização dos modelos clássicos de regressão e classificação, suas peculiaridades e seu treinamento a partir do uso de técnicas de redução de dimensionalidade e configuração de hiperparâmetros.

Prof. Fernando Cardoso Durier da Silva

1. Itens iniciais

Propósito

Compreender a teoria por trás dos modelos máquina de vetores de suporte (SVM), árvores de decisão e florestas aleatórias, modelos fundamentais para qualquer experimento de Aprendizado de Máquina, bem como as técnicas de redução de dimensionalidade e configuração de hiperparâmetros para a melhoria dos resultados de experimentos futuros com quaisquer outros modelos.

Preparação

Antes de iniciar o conteúdo deste tema, é necessário que, em sua máquina, para desenvolvimento local, estejam instalados o Python na versão 3.7 ou superior e, com o comando pip install, as bibliotecas sklearn, numpy, pandas, matplotlib, seaborn, plotly, jupyter. Para estudos e simulações, é recomendado o uso do ambiente proporcionado pelo Jupyter Notebook. Para a organização dos projetos de Aprendizado de Máquina, é recomendado o uso da biblioteca pipenv, que versionará o projeto e listará as suas dependências. Caso o aluno não queira ou não possa utilizar uma máquina física própria, é sempre possível a utilização do Google Collab Notebooks (tomando os devidos cuidados na importação dos conjuntos de dados).

Objetivos

- Descrever o treinamento do modelo de máquinas de vetores de suporte.
- Descrever o treinamento do modelo de árvores de decisão.
- Descrever o treinamento do modelo de florestas aleatórias.
- Descrever o treinamento do modelo de redução de dimensionalidade.

Introdução

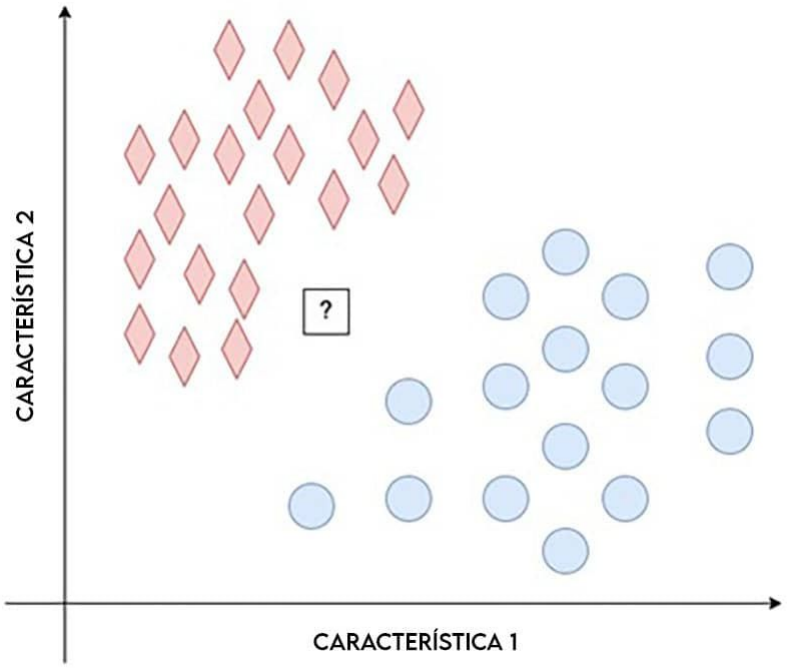
Neste tema, vamos descrever o treinamento de modelos clássicos de aprendizado de máquina, entre eles o SVM (Support Vector Machine), a Árvore de Decisão, a Floresta Aleatória, e a Análise Principal de Componentes.

Vamos reconhecer o funcionamento desses algoritmos usando bibliotecas na linguagem Python, aprendendo a configurá-los para classificar dados, assim como reduzir a dimensionalidade a fim de otimizar nosso processo de treinamento.

Descrevendo o SVM

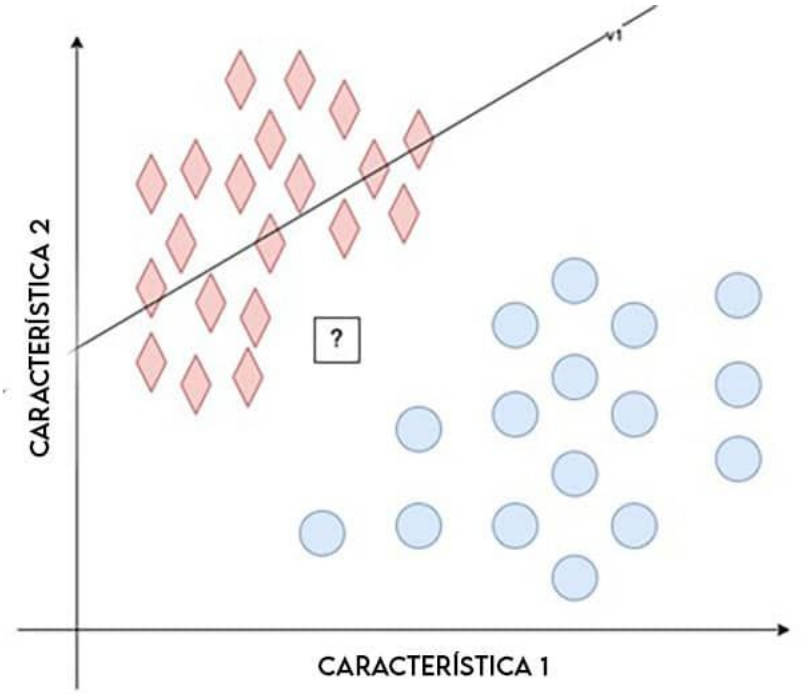
SVM (Support Vector Machine) é uma técnica apresentada por Vapnik *et al.* (1995) como uma abordagem geométrica para o problema de classificação. Cada elemento do conjunto de observações é um ponto Pi num espaço S, e o aprendizado consiste em separar os elementos positivos dos negativos (classificação um contra o resto), ou separar os elementos em grupos de classes (multiclassificação um a um), por meio da otimização dos limites de decisão (vetores de suporte).

O algoritmo SVM funciona da seguinte forma: imaginemos o espaço amostral, a seguir, em que podemos observar elementos de duas classes distintas.



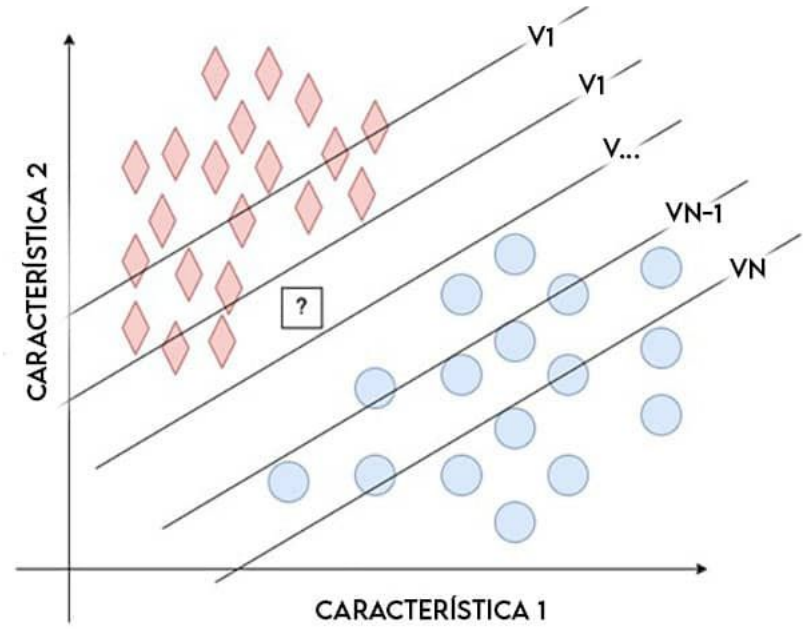
Passo 1 da etapa do processo de classificação.

Para tentarmos separá-las, vamos traçar um vetor aleatório no espaço, como veremos a seguir, em que ainda não conseguimos o resultado esperado.



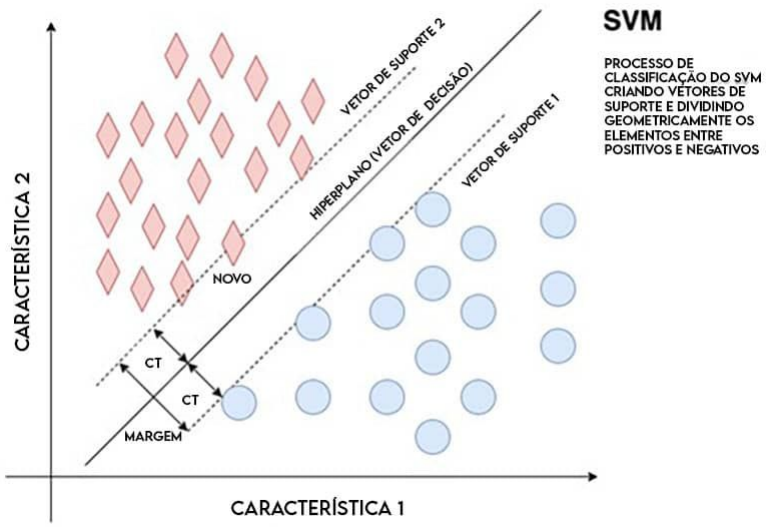
Etapa 2 do processo de classificação. Instanciação de um vetor de suporte.

Agora, vamos instanciar mais vetores; o algoritmo do SVM tenta maximizar as margens entre as distâncias mais próximas.



Etapa 3 do processo de classificação: várias instâncias de vetores de suporte a serem avaliadas.

Já aqui, podemos observar o estabelecimento dos nossos vetores de suporte (agora em pontilhado) e o vetor de decisão (agora em linha contínua). Ao final, a nova instância é classificada como um losango.



Etapa final do processo de classificação: são escolhidos os vetores ótimos, e a nova instância é classificada.

Olhando de forma mais matemática, conforme Santos (2002), imagine um conjunto de exemplos E, onde cada observação seja um par ordenado:

$$\{X_i, Y_i\} \mid i = [1, 2, \ldots, O]$$

Onde $X_i \in R^M$ é uma representação vetorial de uma observação, e $Y_i \in \{0, 1\}$ seria sua classe associada. Imagine que exista uma distribuição de probabilidade P(x,y) da qual os dados de exemplos foram extraídos, mas que não conhecemos. Durante o processo de treinamento, o classificador deverá aprender a fazer o mapeamento do conjunto de observações às suas classes (X→Y), de modo que o mesmo seja capaz de classificar uma nova instância inédita. Assim, o classificador SVM deverá minimizar a expectativa de erro E(α) em uma classificação, tal que E(α)é dado por:

1.1

$$E(\alpha) = \int \tfrac{1}{2}|Y - f(X, \alpha)| dP(X, Y)$$

Note que P(X,Y) é desconhecido, logo não seria possível calcular essa equação. Por outro lado, a função de Erro Empírico, $E_{\text{emp}}\ (a)$, é definida como a média da taxa de erro nos elementos do conjunto de exemplos e pode ser definida como:

1.2

$$E_{\text{emp}}\ (\alpha) = \tfrac{1}{2} O \sum_{i=1}^{O} |Y_i - f\ (X_i, \alpha)|\ (1.2)$$

É válido observar que o E_{emp} é fixo para um valor arbitrário de alpha (α) e um conjunto de exemplos $\{X_i, Y_i\}$.

Para entendermos melhor a matemática a seguir, apresentaremos o conceito de Dimensão Vapnik-Chervonenkis (Dimensão VC). Esta é uma medida de capacidade (complexidade, riqueza e flexibilidade) de um conjunto de funções que podem ser aprendidas por um algoritmo estatístico binário de classificação.

Dito isso, seja h a dimensão-VC do conjunto {f,(α)}, e seja E_{emp} (α) definido pela equação 1.2. Ao selecionarmos um k tal que 0 <= k <= 1 , o seguinte limite é válido com probabilidade de pelo menos 1 - k para *O* > h:

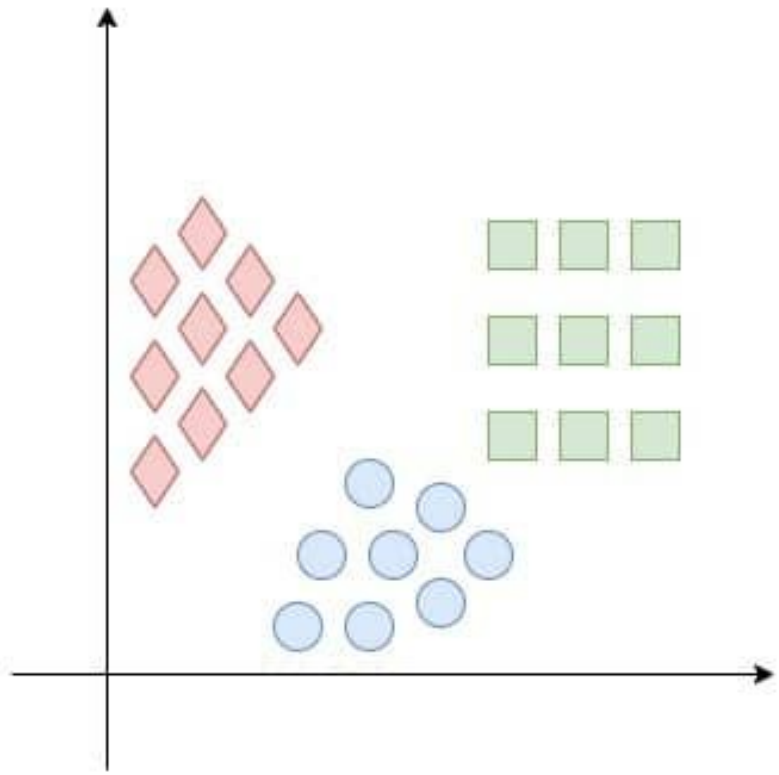
1.3

$$E(\alpha) <= E_{emp}(\alpha) + \sqrt{\frac{h \log\left[\frac{2O}{h}\right] - \log\left(\frac{k}{4}\right)}{O}}$$

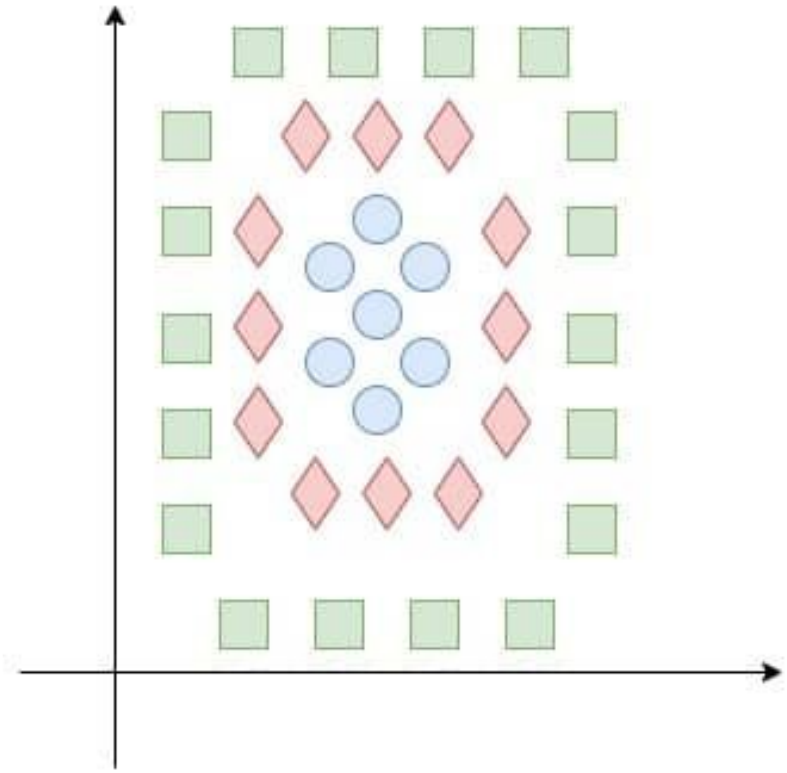
O termo somado a E(α) (componente em raiz quadrada) é conhecido como "termo de complexidade". Assim, minimizando o lado direito da equação 1.3, podemos encontrar a melhor máquina de vetores de suporte que atenda à tarefa proposta de classificação.

Tarefas de Aprendizado de Máquina

No âmbito de Aprendizado de Máquina, temos o que chamamos de tarefas de aprendizado. Algo semelhante ocorre com um ser humano, que, nos seus primeiros anos de vida, precisa aprender a encontrar significados para o que ele entende, seja por similaridade, por experiência/vivência ou por ser um axioma, isto é, a classificação. Ele pode, a partir de características das observações feitas, calcular e projetar ações/perspectivas, o que podemos entender como regressão, e, finalmente, juntar observações em um compartimento físico ou mental devido à proximidade física ou conceitual (similaridade). Chamamos isso de agrupamento (clusterização).



Dados linearmente separáveis.



Dados não linearmente separáveis.

Ao visualizarmos nossos dados, antes de partirmos para o treinamento de modelos, a clusterização é uma etapa muito importante, principalmente para classificadores que dependem da distribuição geométrica dos dados de treinamento, como o SVM. Assim, saberemos qual seria o tipo de classificação mais adequada, linear ou não linear, pois é possível separarmos nossos dados a partir de um hiperplano linear ou não, como acabamos de ver.

Na classificação linear, o hiperplano calculado pela máquina de vetores de suporte vai ter a função de erro a ser otimizada de forma muito parecida com a equação da reta que conhecemos com a seguinte forma, definindo, respectivamente, os limites inferior e superior:

$$+b + b >= 0$$

Já na classificação não linear, vamos observar ligeira mudança de comportamento. Como o espaço não é linearmente separável, entre as observações, as funções de decisão terão um comportamento similar aos algoritmos de vizinhos mais próximos, onde se deseja minimizar a distância entre elementos da mesma classe e maximizar a distância entre os conjuntos de diferentes classes.

> Chamamos de aprendizado de função de kernel a subtarefa de apreender essa função não linear, que é a mais comum e genérica, justamente por tornar este comportamento parecido com os vizinhos mais próximos, sendo a função baseada em radial (RBF).

Já para a tarefa de regressão, o objetivo do classificador é projetar uma função que seja o mais semelhante possível à equação da reta da classe numérica desejada.

Exemplo

Imaginemos que a classe Y desse novo problema siga a distribuição de uma parábola. O SVM vai otimizar o hiperplano que vimos anteriormente, de modo que a função de kernel seguirá esse mesmo comportamento.

Cotidiano do SVM

Usamos muito o SVM para tarefas de classificação, principalmente de conjuntos de dados cuja separação geométrica é evidente, como, por exemplo, imagens, mapas, grupos polarizados etc., Ou para conjuntos de dados cuja distribuição é complexa e tem *overlap* de grupos, como, por exemplo, categorias de textos.

Vantagem do SVM

É a eficácia de trabalhar em espaços dimensionais muito grandes, podendo ser usado mesmo quando a quantidade de dimensões é maior do que a quantidade de amostras. Ele utiliza um subconjunto de pontos de treinamento na função de decisão (vetores de suportes) e é versátil, pois permite que outras funções de kernel possam ser utilizadas para calcular os limites de decisão.

Desvantagem do SVM

É o custo computacional para fornecer probabilidades de predição, uma vez que isso não é natural do algoritmo por trás dele nem de sua concepção, além de tomar toma muito tempo de treinamento para conjuntos de dados muito grandes. Devido ao fato de ser um classificador geométrico, sofre bastante em virtude do excesso de características no *dataset*. Quando isso acontece, são necessárias regularizações e configuração de parâmetros para não ocorrer superajuste de modelo (*overfitting*).

Implementando um classificador SVM em Python

Para implementar um classificador SVM em Python, é necessária a instalação das bibliotecas pandas, numpy e matplotlib. Comecemos pela importação das bibliotecas desta forma:

```
python

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

Em seguida, importaremos o *dataset* que exploraremos em nossos estudos: o *dataset* Íris, clássico no âmbito do Aprendizado de Máquina.

> Íris é um *dataset* que classifica tipos de flores da espécie Íris a partir de determinadas características, como comprimento de pétala, largura da pétala, comprimento da sépala e largura da sépala (todas em cm).

Para isso, adicionaremos ao nosso código, observando o destaque da coluna de classe (target) das características:

```python
from sklearn.datasets import load_iris
data = load_iris()
iris = pd.DataFrame(data['data'], columns=data.feature_names)
target = data.target
```

Para a instanciação simples do classificador, podemos adicionar o seguinte bloco de código, observando que o classificador SVM é identificado no scikit-learn a partir do nome SVC (C de *Classifier*):

```python
#Importando o algoritmo de SVM
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
svc = SVC(gamma="auto")
```

Para treinar o modelo e conhecer sua performance, adicionaremos o bloco de código a seguir. A validação cruzada é um dos modos mais comuns de treinamento dos nossos modelos, pois divide o conjunto de dados em (k-1)/k partições de treinamento e 1/k de teste de maneira circular e interativa, com todas as 1/k possíveis partições podendo ser testadas contra o resto:

```python
#Testando o modelo 'svc' na nossa base 'iris'
cv_result = cross_val_score(svc, iris, target, cv=10, scoring="accuracy")
#Retorna a acurácia em porcentagem do nosso modelo
print("Acurácia com cross validation:", cv_result.mean()*100)
```

Agora que fizemos o experimento de construir o classificador com os parâmetros padrões da biblioteca, podemos fazer predições. Para tal, vamos treinar nosso modelo com o *dataset* inteiro e tentar predizer um valor inédito.

```python
svc.fit(iris, target)
#Prediz a que classe pertencerá a flor com sépala de comprimento 6.9cm e de largura 2.8cm
e com pétala de comprimento 6.1cm e de largura 2.3cm
svc.predict([[6.9,2.8,6.1,2.3]])
```

Feito isso, vamos visualizar nossos dados e os hiperplanos definidos pelo modelo. Nossos dados têm o seguinte comportamento:

```python
plt.scatter(iris['sepal length (cm)'], iris['petal width (cm)'], c=target)
plt.title('Iris')
plt.show()
```

Como podemos ver no Gráfico de Dispersão dos Dados Íris, nosso problema parece ser linearmente separável. Com esse outro bloco de código:

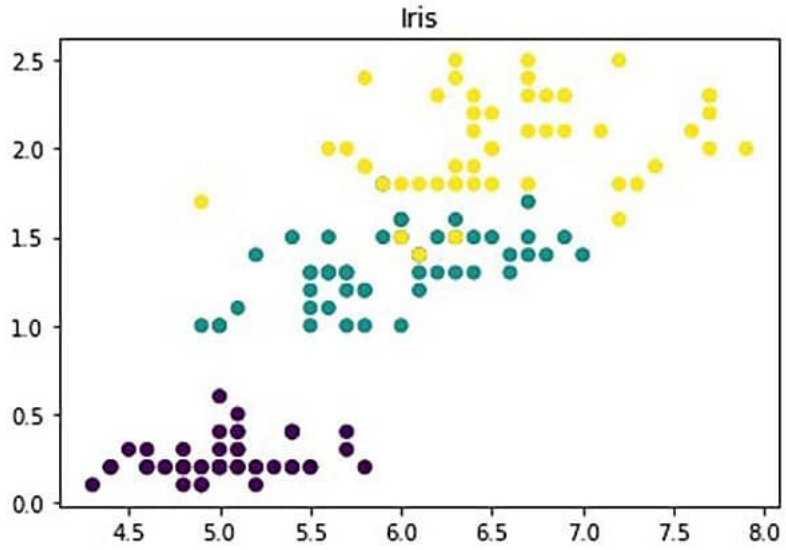


Gráfico de Dispersão dos Dados Íris.

Observe o bloco de código seguinte:

```
python

#Provavelmente criando duas features novas no iris o svm com 2 features terá mais
sucesso, mas por enquanto usei só
#sepal length e petal width (os mais relevantes das 4 features já existentes)
x0_min, x0_max = iris['sepal length (cm)'].min(), iris['sepal length (cm)'].max()
x1_min, x1_max = iris['petal width (cm)'].min(), iris['petal width (cm)'].max()
w = x0_max - x0_min
h = x1_max - x1_min
x0, x1 = np.meshgrid(np.linspace(x0_min-.1*w, x0_max+.1*w, 300), np.linspace(x1_min-.1*h,
x1_max+.1*h, 300))
svc.fit(iris[['sepal length (cm)', 'petal width (cm)']], target)
ypred = svc.predict(np.c_[x0.reshape(-1, 1), x1.reshape(-1, 1)])
ypred = ypred.reshape(x0.shape)
plt.contourf(x0, x1, ypred)
plt.scatter(iris['sepal length (cm)'], iris['petal width (cm)'], c=target, s=64,
edgecolors='k')
plt.title('Iris')
plt.show()
```

Podemos observar que, de fato, o SVM foi capaz de definir os hiperplanos, e, se olharmos bem no hiperplano superior, bem como no hiperplano, a seguir, veremos alguns outliers entre eles, ignorados estrategicamente pelo SMV para não superajustar sua classificação, conferindo certa generalização ao classificador.

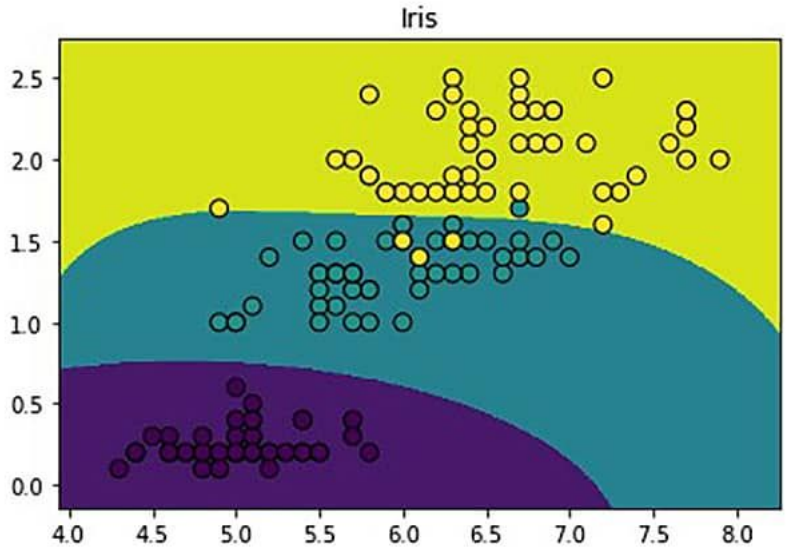


Gráfico de Hiperplanos gerados pelo modelo SVM.

Demonstração da implementação em Python do classificador SVM

No vídeo a seguir, será demonstrada a implementação do classificador SVM em Python.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

As máquinas de vetores de suporte são assim conhecidas devido:

A

Aos vetores que delineiam as margens de decisão.

B

Ao fato de o processo de classificação ser linear.

C

Ao fato de fazer regressões não lineares.

D

Ao fato de apreender uma função de kernel ótima.

E

Nenhuma das opções acima.

A alternativa A está correta.

O SVM tem esse nome devido aos vetores de suporte que definem as margens e balizam o hiperplano de decisão.

Questão 2

O SVM é utilizado para:

A

Agrupamento de Observações por Similaridade.

B

Classificação somente.

C

Regressão somente.

D

Mapeamento do espaço amostral.

E

Classificação e regressão.

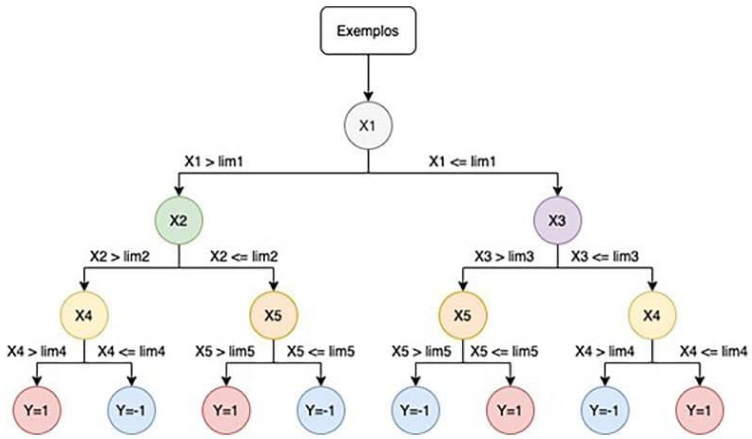
A alternativa E está correta.

A alternativa A não é tarefa de modelos de aprendizado supervisionado, como o SVM. As alternativas B e C estão incompletas. A alternativa D representa uma base fundamental do funcionamento do algoritmo, mas não é seu objetivo.

Descrevendo a árvore de decisão

A árvore de decisão é o modelo mais prático e de mais fácil entendimento dentre os classificadores clássicos, devido à estrutura de dados utilizada e ao algoritmo de construção (AMARAL, 2016).

É um classificador construído de cima (raiz) para baixo (folhas). Na raiz, são alocados todos os exemplos de treinamento; logo, eles são agrupados pelo atributo de melhor separação do conjunto inicial (escolhido de forma iterativa). Escolhido o atributo, são criados N nós filhos da raiz para os possíveis valores desse atributo, e os elementos que estavam na raiz são alocados para esses filhos de acordo com a condição do valor do atributo escolhido, como podemos ver, a seguir.



Representação de uma árvore de decisão: cada nó interno é um nó de decisão sobre uma característica, e as folhas são as classificações em si.

De forma recursiva para cada nó interno, fazemos o mesmo processo, ou seja, escolhemos o atributo mais adequado, agrupamos os dados, criamos filhos para cada valor possível e os dados são alocados nos nós filhos, até que o nó filho não possa mais ser partido em outros agrupamentos.

Tal algoritmo é muito versátil, pois atende a qualquer tipo de dado, seja categórico, seja numérico (diferentemente da maioria dos modelos que exigem pelo menos a transformação dos dados categóricos em numéricos). Devido à sua simplicidade e indução compreensível, é muitas vezes preferido em detrimento de classificadores mais complexos, e, por trabalhar com a estrutura de dados de árvore, é assistido por diversas técnicas de Análise de Algoritmos para Árvores, como busca binária, poda de ramos, balanceamento etc.

Treinando e visualizando a árvore de decisão

O processo de treinamento de uma árvore de decisão, assim como a maioria dos modelos de Aprendizado de Máquina, baliza-se na otimização de uma função de custo para que o modelo evolua e aprenda.

No caso das árvores de decisão, elas dependem de algumas funções de custo. Uma delas é a que determina qual é o melhor atributo a ser escolhido para dividir o conjunto de dados. Essa função se baseia no conceito de entropia, que caracteriza a impureza dos dados em seu conjunto, ou seja, a falta de homogeneidade dos dados de entrada em relação à sua classificação — **quanto maior a entropia, mais heterogênea é a amostra** (MITCHELL, 1997). Dado um conjunto S, de classes i com proporções p, a entropia é expressa pela fórmula:

2.1

$$\text{Entropia}(S) = \Sigma p_i \log_2 (p_i)$$

A outra função de custo é a de Ganho de Informação, que depende do conceito de entropia que apresentamos anteriormente, pois é dada em função do mesmo. O ganho de informação é dado pela seguinte fórmula:

2.2

$$\text{Ganho}(S, A) = \text{Entropia}(S) - \sum_{v \in \text{ volores } (A)} \frac{|S_v|}{|S|} \text{Entropia } (S_v)$$

Vamos interpretar esta fórmula 2.2, onde temos A como atributo analisado, v como possíveis valores do atributo A, S como o conjunto de todos os atributos e Sv como o subconjunto de S, em que A = v. Logo, a otimização para o aprendizado da árvore é maximizar o ganho de informação a cada etapa de construção da árvore.

Num exemplo mais prático (Exemplo de árvore de decisão e seu funcionamento, visitado em 10/01/2021), imaginemos o seguinte conjunto de dados na tabela:

Dia	Aspecto	Clima	Umidade	Vento	Jogar tênis
1	Sol	Quente	Elevada	Fraco	N
2	Sol	Quente	Elevada	Forte	N
3	Nuvens	Quente	Elevada	Fraco	S
4	Chuva	Ameno	Elevada	Fraco	S
5	Chuva	Fresco	Normal	Fraco	S
6	Chuva	Fresco	Normal	Forte	N
7	Nuvens	Fresco	Normal	Fraco	S
8	Sol	Ameno	Elevada	Fraco	N
9	Sol	Fresco	Normal	Fraco	S
10	Chuva	Ameno	Normal	Forte	S
11	Sol	Ameno	Normal	Forte	S
12	Nuvens	Ameno	Elevada	Forte	S
13	Nuvens	Quente	Normal	Fraco	S
14	Chuva	Ameno	Elevada	Forte	N

Tabela: Exemplo de conjunto de dados
Fernando Cardoso Durier da Silva.

Etapas

1. Análise de todos os atributos, começando pela Umidade:

S = [9 positivos, 5 negativos]
E = 0.940
v(Umidade) = Elevada → [3 positivos, 4 negativos] → E = 0.985
v(Umidade) = Normal → [6 positivos, 1 negativo] → E = 0.592
Ganho(S, Umidade) = 0.940 - (7/14)*0.985 - (7/14)*0.592 = 0.151

2. Calculando o ganho para os demais atributos:

Ganho(S, Umidade) =0.151
Ganho(S, Vento) =0.048
Ganho(S, Aspecto) =0.247
Ganho(S, Temp) =0.029

3. Escolhemos o atributo que maximiza a função de ganho, sendo este o Aspecto.

Ganho(S, Aspecto) =0.247
Com o aspecto, uma folha já é definida como Sim, a partir do ramo condicional Aspecto = Nuvens com 4 exemplos positivos e 0 negativos.

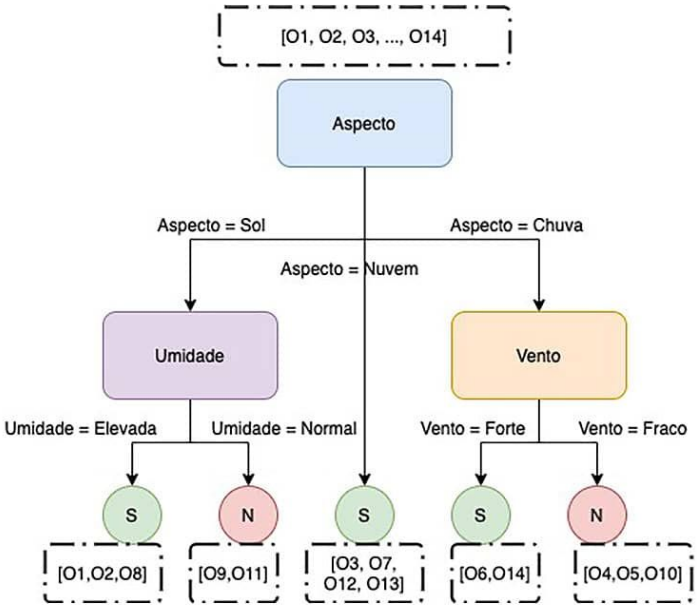
4. No próximo passo, o Aspecto sai do conjunto, e olhamos o restante a partir dos ramos condicionais de valores de aspecto ainda sem folhas determinadas, próximo Aspecto = Sol:

Ganho(S_{sol},Humidade)=0.970-(3/5)0.0 - 2/5(0.0) = 0.970
Ganho(S_{sol},Clima)=0.970-(2/5)0.0 - 2/5(1.0) - (1/5)0.0 = 0.570
Ganho(S_{sol},Vento)=0.970 - (2/5)1.0 - 3/5(0.918) = 0.019
Temos que Umidade é o maximizador desse ramo.

5. Com Umidade como o próximo nó intermediário, temos a separação condicional de seus ramos em Elevada e Normal, que, subsequentemente, geram folhas.

6. Vento tem o maior ganho de informação para o ramo condicional ainda em aberto, que, por meio de seus ramos condicionais, consegue gerar folhas e classificar todas as observações ainda remanescentes.

7. Ao final, temos a seguinte árvore de decisão:



Árvore resultante da classificação do problema de jogar tênis.

A árvore de decisão pode ser também empregada para a regressão.

A diferença está no tipo de variável que é a classe e nas funções que ela vai otimizar para ajustar o modelo à distribuição de probabilidade da classe. A classe passa a ser uma variável numérica, no domínio dos Reais, enquanto a função a ser otimizada passa a ser a do Mínimo Erro Quadrático, ou, em alguns casos, a função a ser otimizada pode ser a Desvio de Meia Poisson. Árvores de decisão usadas para problemas de regressão são chamadas também de árvores de regressão, onde cada nó terminal ou folha contém uma constante (geralmente, uma média) ou uma equação para o valor previsto de determinado conjunto de dados.

Comentário

Devido ao caráter genérico e altamente explicável da árvore de decisão, existe uma variedade chamada CART (Classification and Regression Trees), proposta por Breiman et al. (1998), que é um modelo híbrido capaz de lidar tanto com regressão quanto com classificação.

Identificando e configurando os hiperparâmetros da árvore de decisão

Algoritmos baseados em árvores herdam os benefícios das estruturas de dados que lhes dão nome, mas também herdam suas peculiaridades e limitações, como, por exemplo, necessidade de balanceamento para otimização de custo computacional.

Hiperparâmetros

São atributos dos modelos de classificação ou regressão que, ao serem regulados, podem causar impacto no processo de aprendizado e trazer resultado do algoritmo de Aprendizado de Máquina em questão. No caso

das árvores de decisão, os principais hiperparâmetros da árvore de decisão são sua altura (ou profundidade) e o número máximo de características a serem analisadas por nós internos e número mínimo de folhas.

A altura da árvore (ou profundidade máxima) é o parâmetro que dita a regra de expansão da árvore: se deixado nulo, a árvore expandirá até que as folhas estejam puras ou tenham a quantidade de amostras mínimas especificadas por folha; ou, se atribuído um valor inteiro, a árvore para sua ramificação até determinada altura e considera os nós dessa camada como folhas.

> Quanto maior a profundidade da árvore, maior é a chance de sobreajuste do conjunto de treinamento.

O **número máximo de características** a ser analisado por nó interno é uma restrição aplicada ao conjunto de características pelo qual o processo de escolha de melhor divisor pode optar, ou seja, de maneira forçada, dizemos para o algoritmo olhar para uma fração ou uma quantidade inteira de características e, dentre elas, escolher o melhor divisor de agrupamento. Essa restrição também é aplicada como forma de regularização do algoritmo, uma vez que conjuntos muito grandes de características podem causar sobreajuste do modelo. Contudo, lembrando que, caso o processo de decisão não consiga achar um candidato divisor válido, o algoritmo ignora a restrição e volta a olhar para todas as características até encontrar um divisor válido para aquela etapa.

O **número máximo de folhas** é um parâmetro que limita a quantidade de folhas que cada ramo pode ter. Este é mais um parâmetro de regularização da árvore, de modo que o modelo não fique com poucos nós de decisão (menos ramos) em prol de uma copa maior (mais folhas num só ramo), o que poderia causar desbalanceamento na árvore e deixá-la ineficiente para predições que caíssem de determinado "lado". Dessa forma, ao restringir a copa, escolhendo, por exemplo, duas folhas no máximo, temos uma árvore com mais decisões (mais robusta) e equilibrada.

Comentário

Ainda sobre as folhas, temos o tamanho mínimo da amostra que restringe ou flexibiliza a quantidade de amostras de valores da classe tolerável de coexistir num nó folha. Se considerado o padrão 1, a folha deve ser pura de determinado valor da classe; caso contrário, se escolhermos um número real, por exemplo, podemos ter proporções de observações da classe numa só folha, e, se tivermos valores inteiros, então, considera-se a quantidade mesmo. Tal regularização permite a flexibilização do modelo, podendo tolerar outliers mais facilmente.

O processo de configuração de hiperparâmetros não deve ser feito de forma arbitrária, apenas em caso de aprendizado, pois, ao construirmos um modelo de Aprendizado de Máquina, queremos otimizá-lo de forma sistemática, metodológica e replicável. Dito isso, o processo correto de escolha de configurações deve ser feito de forma iterativa, por meio de variação dos valores configurados e monitorando o resultado de métricas. Ao fim deste processo, escolhemos a combinação de configurações utilizadas que maximizam o resultado de métricas de avaliação.

Implementando a árvore de decisão em Python

Para implementarmos a árvore de decisão, vamos utilizar novamente a linguagem Python e precisaremos apenas das bibliotecas sklearn e matplotlib.

Vamos começar nosso código importando as bibliotecas e funções necessárias:

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

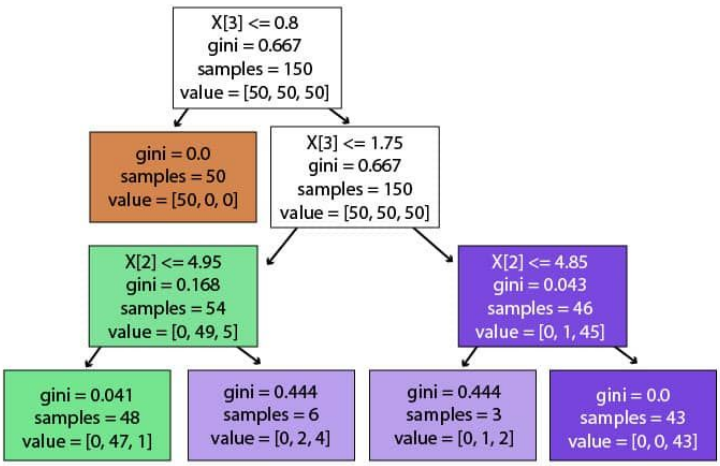
Agora, vamos ao processo de experimentação por meio do treinamento feito com validação cruzada; basta adicionarmos este bloco de código:

```python
clf = DecisionTreeClassifier(max_depth=3, random_state=0)
iris = load_iris()
cross_val_score(clf, iris.data, iris.target, cv=10)
```

Com os dados carregados e o experimento executado para checarmos a possível média de acurácia do modelo treinado com o conjunto, vamos ao treinamento propriamente dito e à visualização do resultado.

```python
clf.fit(iris.data, iris.target)
plot_tree(clf, filled=True)
plt.show()
```

Com poucas linhas de código, pudemos treinar um classificador de árvore de decisão e visualizar sua árvore resultante. Observe!



Árvore de decisão resultante da implementação em Python.

Demonstração da implementação em Python do classificador de árvore de decisão.

No vídeo a seguir, será demonstrada a implementação do classificador de árvore de decisão em Python.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Para um classificador árvore de decisão, a função de decisão para escolha de melhor divisor se baseia em qual das alternativas?

A

Entropia.

B

Ganho de Informação.

C

Mínimo Erro Quadrático.

D

Função de Base Radial.

E

Amostra Mínima de Folhas.

A alternativa B está correta.

A alternativa A faz parte do cálculo do Ganho de Informação, mas não é suficiente por si. A alternativa C seria a resposta válida para uma regressão. A alternativa D é a função a ser otimizada para Classificadores SVM Não Lineares. A opção E, por sua vez, é falsa, pois se trata de um hiperparâmetro da árvore de decisão, e não uma função de decisão.

Questão 2

Ao regularizarmos a altura máxima da árvore de decisão, temos o objetivo de:

A

Evitar o sobreajuste.

B

Balancear a árvore.

C

Diminuir a chance de acertos.

D

Evitar o sobreajuste e balancear a árvore.

E

Balancear a árvore e diminuir a chance de acertos.

A alternativa A está correta.

Evitar o sobreajuste do conjunto de treinamento é o principal objetivo desta regularização. A alternativa B não é totalmente verdadeira, pois a regularização mais efetiva para este caso seria a restrição do número máximo de folhas, e a restrição do tamanho da árvore sem a regularização do número de folhas pode não evitar que um lado tenha tamanho máximo, enquanto o outro esteja a ¼ ou ⅓ de profundidade do ramo maior. A alternativa C é falsa, uma vez que nenhuma técnica de regularização tem como objetivo prejudicar o modelo, mas, sim, ajudá-lo. As alternativas D e E são falsas devido a combinações das opções já citadas.

Compreendendo modelos incorporados

O aprendizado incorporado (*ensemble learning*) é a estratégia de aprendizado na qual se toma vantagem da variabilidade de conhecimentos que uma população tem em relação a somente um indivíduo.

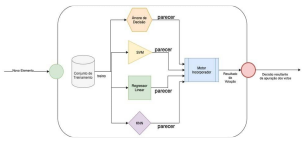
Exemplo

Dada uma feira pública, um fazendeiro propõe um concurso para adivinhar o peso de um de suas vacas: cada indivíduo presente chuta um número, mas, individualmente, ninguém acerta. Porém, ao se tirar a média dos palpites, chega-se muito perto ou acerta-se o peso do animal, ou seja, aposta-se na inteligência coletiva, e não na individual.

Alguns algoritmos no Aprendizado de Máquina bebem dessa mesma fonte; entre as técnicas mais usadas, temos: a votação e o empilhamento.

Votação

Consiste no agrupamento de três ou mais algoritmos de aprendizagem por tarefa de aprendizado (classificação, regressão ou clusterização) e estes treinam sobre o mesmo dataset. Então, quando uma nova observação é apresentada ao comitê de modelos treinados, cada integrante emite um parecer, e ocorre uma votação (normal ou ponderada) em relação aos pareceres, da qual sairá o resultado final.



Emplilhamento

O processo é parecido com o que o ocorre em redes neurais, em que há transformação do conjunto de treinamento à medida que ele vai sendo transformado durante o avanço do aprendizado. Ou seja, no caso do aprendizado empilhado, cada modelo do comitê recebe como input o resultado dos modelos anteriores e se chega a uma conclusão. Por exemplo, imagine um modelo não supervisionado que recebe o dataset inicial, descobre-se automaticamente agrupamentos escondidos nele, devolve-o acrescido dessas colunas e passa-o à frente para um classificador de árvore de decisão. Nesse caso, temos um modelo empilhado híbrido de aprendizado não supervisionado e aprendizado supervisionado para uma tarefa de classificação.

Existe também o método de incorporação bootstrap aggregating, que, além de envolver o uso de um comitê de modelos, ainda faz a randomização das amostras de distribuição dos dados entregues a cada um dos modelos membros do comitê. É justamente sobre ele que vamos falar mais à frente com as florestas aleatórias, Random Forests, demonstradas, a seguir.

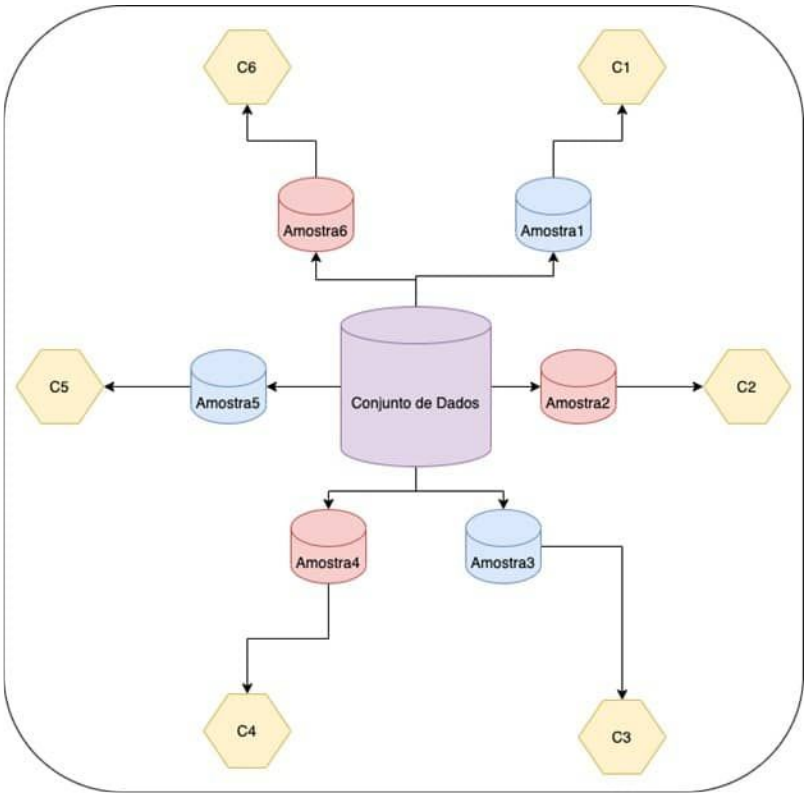


Ilustração de Bagging, repartindo aleatoriamente o conjunto de dados inicial entre os membros do comitê.

Treinamento da Floresta Aleatória

Inicialmente propostas por Breiman (2001), as florestas aleatórias, em um primeiro momento, foram criadas para regularizar um problema de árvores de decisão individuais que cresciam muito, ou seja, aumentavam indiscriminadamente seus nós internos e iam aprendendo padrões irregulares que não deviam, causando sobreajuste, diminuindo o viés e aumentando demais a variância. Muito semelhante ao fenômeno de estiolamento de plantas que são parcialmente ou totalmente privadas da luz solar e precisam se esticar até achar uma brecha de luz, deformando-se no processo.

O aprendizado da floresta aleatória se dá da seguinte forma para um conjunto de treinamento:

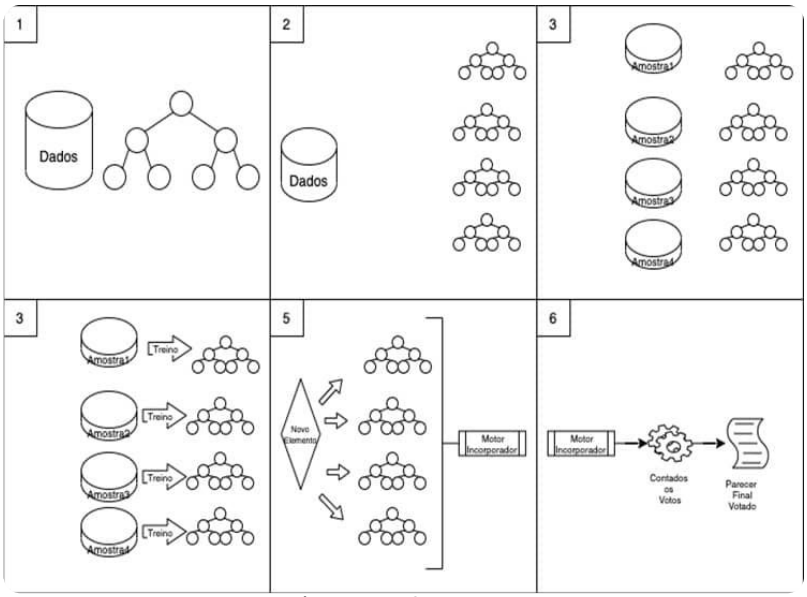
$$X_i = X_1, X_2, \ldots X_{n-1}, X_n$$

E um conjunto de classificações:

$$Y_i = Y_1, Y_2, \ldots, Y_{n-1}, Y_n$$

O algoritmo de aprendizado das florestas aleatórias começa a sortear amostras desses conjuntos com substituição, a fim de formar conjuntos aleatórios que são "fotos" de partes do conjunto de dados inicial. Feito isso, cada árvore de decisão na floresta recebe esse pacote de amostras e treina utilizando o mesmo algoritmo das árvores de decisão que vimos, anteriormente, no módulo 2.

Uma vez realizado o treinamento, quando um novo elemento for apresentado para a floresta, as árvores avaliam esse novo elemento, e cada uma emite um parecer sobre ele, ou emite o resultado de um cálculo em caso de regressão. Visto que todas as árvores terminaram sua análise e emissão de pareceres, o conselho/ comitê da floresta decide o resultado final a partir de uma votação, no caso de problemas de classificação, ou média, no caso de um problema de regressão, como vemos, a seguir.



Treinamento de Florestas Aleatórias: 1 - Definição do estimador base e dados; 2 - Replicar a classe básica; 3 - Fazer amostras dos dados originais; 4 - Treinamento da Floresta; 5 - Apresentação de Nova Instância e Votação; 6 - Emissão de Parecer Votado.

Ao mesmo tempo em que as florestas aleatórias são muito eficientes em prover uma decisão (classificação ou regressão) mais justa e confiável pela definição de seu algoritmo, temos a contrapartida: a falta de explicabilidade (*explainability*).

> **Explicabilidade:** é uma disciplina da Inteligência Artificial que tem como compromisso fazer a prova real do processo, algoritmo ou heurística de aprendizado envolvido em um modelo, explicando para o humano que está interagindo com o computador como uma decisão importante foi tomada, como, por exemplo, sobre concessão de crédito. Em árvores de decisão, a explicabilidade é evidente. Basta seguir o caminho da raiz até a folha de classificação, e assim entendemos como o modelo raciocinou sua escolha.

No caso das florestas randômicas, é muito mais complicado poder acompanhar esse caminho decisório, pois seria necessário iterar por cada membro da floresta, extrair o caminho decisório e, a fim de extrair todos os caminhos, teríamos de descobrir (caso a implementação não nos disponibilize) o critério do comitê: se foi votação simples, votação ponderada etc. Porém, tirando a explicabilidade, existem cuidados a serem tomados pelos usuários das florestas aleatórias, como veremos na próxima seção.

Comentário

Existe ainda um modelo variante das florestas aleatórias, que é o Extra Trees (árvores extras). Este modelo é similar às florestas aleatórias, porém, em vez de implementar a estratégia incorporada de bagging, repassa o conjunto completo para as árvores de suas florestas. O outro diferencial é que, na hora de decidir o ponto de corte nos nós internos da árvore, em vez de ocorrer a escolha do melhor atributo que divide os dados segundo o critério de separação pelo ganho de informação, as árvores extras fazem isso de forma aleatória.

Identificando e configurando os hiperparâmetros da Floresta Aleatória

Assim como a árvore de decisão, a floresta aleatória tem parâmetros parecidos, que regulam o que chamamos de estimador base, a classe que será "plantada" pela floresta. Então, aqui podemos considerar os mesmos hiperparâmetros vistos no módulo anterior sobre a árvore de decisão.

Além dos hiperparâmetros herdados da árvore de decisão, a floresta conta com o hiperparâmetro de número de estimadores, que define basicamente quão densa é a floresta. É muito importante tal parâmetro, pois se, naqueles herdados da árvore básica, nós queremos regularizar a altura da árvore, o número de folhas etc., ou seja, topear a árvore, garantindo que não irá superajustar o conjunto de treinamento, é por meio da quantidade de árvores na floresta que enaltecemos a grandeza desse modelo.

Comentário

Quanto maior o número de árvores, mais variado é o comitê, menor a chance de não termos predições viciadas, regularizando, assim, o aspecto de viés e deixando a floresta mais genérica (não terá grandes dificuldades para predizer elementos completamente inéditos). Claro que isso deve ser feito com o devido cuidado, pois uma floresta excessivamente grande pode se tornar computacionalmente muito mais custosa do que deveria e, devido ao excesso de variedade, pode não convergir em votação.

1 árvore

Sempre que fizermos configurações de hiperparâmetros, precisamos fazer isso de forma sistemática, ou seja, utilizaremos o método de *grid search*, no qual preparamos uma matriz onde cada coluna é um parâmetro e cada linha é preenchida com combinações de valores dessas colunas.

2 árvores

Após isso, iteramos sobre essas configurações linha a linha e, com validação cruzada, treinamos nossos modelos.

3 árvores

A cada resultado, normalmente, métricas de performance, como acurácia, precisão medida F, tempo de treino, são anotadas em colunas adicionadas nesta matriz à medida que os treinamentos acabam.

4 árvores

E, no fim, selecionamos a configuração que resultou no maior valor de métrica(s) selecionada(s).

De modo geral, em comparação com outros modelos, as florestas aleatórias tendem a alcançar melhor resultado em métricas, como acurácia, medida F, curva ROC, devido à estratégia de incorporação inerente de sua arquitetura. Contudo, é muito importante executarmos sempre a configuração de hiperparâmetros, pois, assim, não só tentaremos reduzir a chance de sobreajuste, mas também seremos transparentes em nosso processo experimental, uma vez que os modelos incorporados, naturalmente, têm sua explicabilidade prejudicada.

Implementando a Floresta Aleatória em Python

Para a implementação da Floresta Aleatória em Python, vamos precisar das seguintes bibliotecas: sklearn, numpy e pandas.

Para o nosso exemplo, vamos utilizar um *dataset* artificial gerado para problemas de classificação. Para efeito de simplicidade e visualização, vamos utilizar 5 estimadores para a floresta.

```
python

import pandas as pd
from matplotlib import pyplot as plt
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=4, n_informative=2, n_redundant=0,
random_state=0, shuffle=False)
clf = RandomForestClassifier(n_estimators = 5, max_depth=2, random_state=0)
clf.fit(X, y)
print(clf.predict([[0,0,0,0]]))
```

Para descobrir as importâncias de características da floresta aleatória, basta chamar o atributo da instância treinada do classificador com o seguinte bloco de código:

```
python

print(clf.feature_importances_)
```

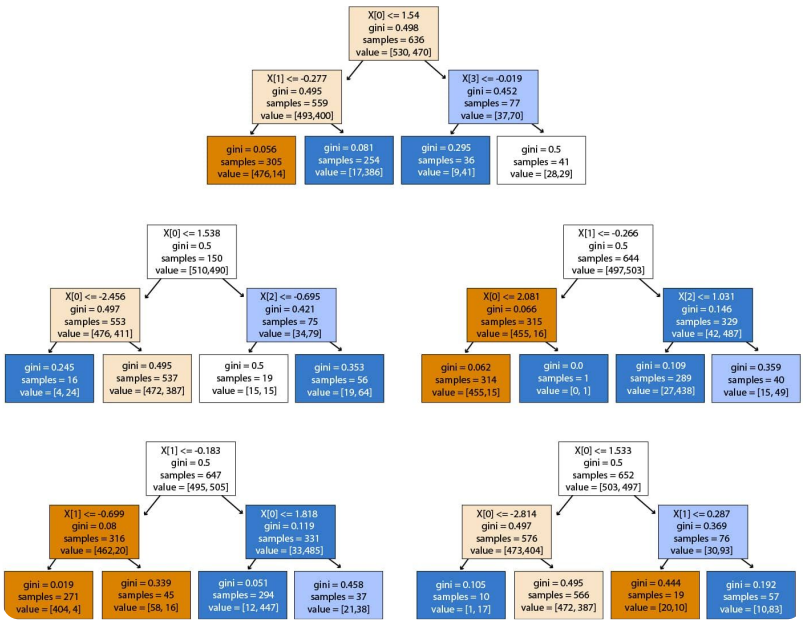
Para visualizar o conjunto de dados artificialmente gerado, utilize o seguinte bloco de código:

```
python

pd.concat([pd.DataFrame(X,columns=['atr1','atr2','atr3','atr4']),pd.DataFrame(y,
columns=['class'])] , axis=1)
```

Finalmente, para visualizar as árvores que compõem a floresta, utilize o seguinte bloco de código, observando que ele exibirá cada estimador separado por linhas tracejadas na célula do seu notebook:

```
python

for tf in clf.estimators_:
print("----------------------------")
plot_tree(tf, filled=True)
plt.show()
```

Para este exercício, as árvores resultantes devem ser similares às da imagem da Figura 16 (não estarão dispostas da mesma forma):



Visualização da Floresta Randômica do nosso exemplo.

Demonstração da implementação em Python do classificador de florestas aleatóriasx

No vídeo a seguir, será demonstrada a implementação do classificador de florestas aleatórias em Python.

> **Conteúdo interativo**
>
> Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Qual o tipo de incorporação que a Floresta Aleatória utiliza?

A

Bagging somente.

B

Votação somente.

C

Stacking somente.

D

Bagging e votação.

E

Votação e *stacking*.

A alternativa D está correta.

As alternativas A e B estão incorretas por não serem somente *bagging*. A Alternativa C está incorreta, pois o conjunto de dados é aleatoriamente amostrado para cada um dos membros da floresta, diferentemente do stacking, em que a saída de um modelo é a entrada do outro. A alternativa E está incorreta devido à combinação de *stacking*.

Questão 2

Qual o hiperparâmetro que confere à floresta aleatória capacidade de generalização?

A

Altura das árvores.

B

Número de estimadores.

C

Estimador base.

D

Número de folhas.

E

Mínima amostra por folhas.

A alternativa B está correta.

A alternativa A não é necessariamente verdadeira, pois, se uma floresta tiver poucos membros, mesmo com alturas regularizadas, ainda vai ser enviesada, por ter pouca variabilidade. A alternativa C é importante de fato, mas uma árvore só nesse tipo de algoritmo não é suficiente (“uma andorinha não faz verão”). A alternativa D está incorreta porque a simples restrição das folhas da árvore-base ainda pode gerar modelos deficientes, visto que o truque desse tipo de aprendizado é o fato de o conhecimento ser armazenado na floresta. A alternativa E seria uma boa opção, pois sabemos que ela pode flexibilizar a árvore-base, mas, ao mesmo tempo, se a quantidade de estimadores não for determinada, você pode ter um modelo tão flexível que seria melhor jogar uma moeda.

Conceito de redução de dimensionalidade

Dimensionalidade é o número de características (*features*) de um conjunto de dados.

As principais razões para diminuir a dimensionalidade são: diminuir o custo (tempo e processamento) de medição dos modelos e melhorar (ou, pelo menos, não prejudicar) a precisão do classificador. Se o modelo tiver de se preocupar só com as características principais, ele ocupará menos memória e poderá processar os dados muito mais rapidamente. Dessa maneira, é mais fácil o trabalho dele.

Assim como nós, em um processo de leitura dinâmica, focamos em aspectos-chave e aceleramos nas partes que não nos interessam, ou ainda, quando fazemos compras, nós nos atentamos a detalhes essenciais, e não cosméticos (e/ou dispensáveis à utilidade).

> É importante salientar que nem sempre conseguimos conjuntos muito grandes de dados para trabalhar (seja por falta de qualidade, acesso ou, simplesmente, novidade); então, para esse cenário, é preferível a escolha de um número menor de dimensões, evitando redundâncias, dispensando informação inútil etc.

É importante retomar o conceito de maldição da dimensionalidade, que está muito relacionado ao assunto, por se tratar de uma situação crítica para o desempenho dos modelos. Assim, a maldição da dimensionalidade (Dimensionality Curse) é o excesso de variáveis em um problema (em nosso caso, excesso de características em um problema de Aprendizado de Máquina).

Atenção

Segundo Watanabe (1985), é possível fazer dois padrões arbitrários se tornarem similares a partir da codificação de características similares. Para a redução de dimensionalidade, temos duas alternativas: a extração de características e a seleção de características.

Observe o seguinte comparativo:

Extração de características	Seleção de características
Este algoritmo combina as características originais em um conjunto reduzido de nova(s) característica(s). A extração de características, por mais que seja melhor para o modelo, acaba deformando os dados no sentido em que eles não fiquem tão explicáveis ou inteligíveis para nós, perdendo o seu significado físico/material.	Segundo um conjunto de regras e objetivos, escolhe as melhores características em detrimento das demais. Esta seleção, reduz o custo da medição de dados (mentalmente imagine uma expressão polinomial gigante se reduzindo a apenas 2 ou 3 termos). Porém, esse tipo de redução não deforma o dado, apenas destaca alguns aspectos e ofusca outros para facilitar o processamento, bem como para melhorar resultados.

O mais importante de tudo: o excesso de redução é perigoso para os modelos, pois ele pode ficar muito ingênuo e julgar resultados por aspectos rasos e/ou desconexos.

Exemplo

Vai chover hoje porque acordei com coceira no joelho esquerdo, quando o conjunto de características completo seria: o céu está nublado desde ontem, estamos no verão, e foi observada uma frente fria no começo da semana.

Análise de Componentes Principais (PCA)

É um método de redução de dimensionalidade do tipo extrator de características. A popularidade da PCA se deve ao fato de ser simples, rápida, linear e não paramétrica.

Comentário

Tal método é baseado na variância dos dados, tentando criar uma nova representação (combinação) destes, normalmente em uma dimensão menor, mantendo a variância entre eles. Imaginando um plano bidimensional cartesiano, cada eixo possuiria um percentual de variância individual com relação ao todo, e seriam completamente desconexos entre eles.

Matematicamente, a PCA é definida como uma transformação linear ortogonal que transporta os dados para um novo sistema de coordenadas, onde a variância é distribuída ordenadamente em grandeza pelas novas coordenadas, ou seja, a maior variância é passada para a primeira coordenada nova, a segunda maior variância para a segunda variável nova, e assim por diante. Normalmente, para fins de visualização, limitamos a quantidade de novas variáveis a apenas três, ainda que não seja metodologicamente o certo (que seria um processo similar ao de configuração de hiperparâmetros, ou seja, teste iterativo de parâmetros objetivando a maximização da métrica do modelo).

Imaginemos uma matriz de dados, $M^\top$, com média amostral nula. Cada linha n é uma observação, e cada coluna m seria uma característica. Agora, decomponham essa matriz em outras três, $W, \Sigma, e^\top$, de tal forma que

$$M^T = W\Sigma V^T$$

Onde W é a matriz de covariância $MM^\top$ com dimensões m por m . A matriz Σ é a matriz diagonal m por n retangular com números reais não negativos nesta diagonal. E , finalmente, a matriz $V^\top$ é a matriz de dimensões n por n dos autovetores de $M^\top M$. Então, a transformação que mantém a mesma dimensionalidade é dada por:

4.1

$$C^T = M^T W$$
$$C^T = V\Sigma^T W^T W$$
$$C^T = V\Sigma^T$$

Como W é uma matriz ortogonal graças à definição da Decomposição de Valores Singulares de uma Matriz Real da Álgebra Linear, e cada linha de $C^\top$ é apenas uma rotação da linha correspondente em $M^\top$, então, a primeira coluna de $C^\top$ é formada a partir dos resultados relativos à primeira componente, a segunda coluna $de^\top C^\top$ formada a partir dos resultados relativos à segunda componente, e assim por diante. Dessa forma, para a redução da dimensionalidade, basta projetar as observações ao espaço reduzido definido pelos primeiros X vetores da matriz W_X da seguinte forma:

4.2

$$C = {W_x}^T M = \Sigma_X V^T \mid \Sigma_X = I_{X,m}$$

Essencialmente, a PCA faz uma rotação dos pontos no espaço dimensional em torno da média, tentando alinhá-los aos componentes principais escolhidos a partir da minimização das distâncias quadradas a essa média. Desse jeito, essa operação de rotação deixa os primeiros componentes principais mais variados do que os últimos, podendo descartá-los por não terem tanto valor como informação. Assim, ela é a melhor transformação ortogonal para manter o menor subespaço mais variado; contudo, o custo computacional dessa operação é muito maior à medida que se tem muitas dimensões.

⚠

Atenção

Para que a PCA funcione de forma "justa", as componentes devem ter a mesma escala, senão, falando em alto nível, uma componente poderia absorver a outra devido à ordem de grandeza (por isso a normalização dos dados é tão importante num processo de Aprendizado de Máquina).

Observe um comparativo entre vantagens e desvantagens:

Vantagem

A PCA tem como vantagem a simplicidade do seu algoritmo devido ao fato de ser linear. É um algoritmo disponível em quase todas as bibliotecas de Aprendizado de Máquina, sendo altamente explicável.

Desvantagem

A PCA tem como desvantagens a sensibilidade à escala dos componentes analisados, o custo computacional, que é muito grande para conjuntos grandes, e a dificuldade de lidar com dados distribuídos de forma não linear no espaço geométrico de observações.

Redução de Dimensionalidade Não Linear através da Incorporação Linear Local (LLE)

Agora, no aspecto de problemas não lineares, ou redução de dimensionalidade não linear (NLDR), temos a Incorporação Linear Local (LLE). A LLE tem vantagens sobre algoritmos que lidam com matrizes esparsas,

bem como apresenta melhores resultados em problemas não lineares do que outras técnicas como a Kernel PCA.

Como o nome implica, a LLE tem o funcionamento parecido com os algoritmos de vizinhos mais próximos, no sentido de que cada ponto (observação) no espaço dimensional é descrito por meio do cálculo de pesos que representam tal ponto em função dos seus vizinhos localmente próximos, como uma combinação linear. Quando todas as observações forem mapeadas, o algoritmo, por meio da aplicação da técnica de otimização de autovetores, acha as representações com menor dimensão compatível com o mapeamento de seus vizinhos.

O erro de reconstrução das observações é dado pela seguinte função de custo E(W):

4.3

$$E(W) = \sum_i \left| X_i - \sum_j W_{ij} X_j \right|^2$$

Os pesos descritos pela matriz W_{ij} são os valores de contribuição que o ponto X_j tem, enquanto o ponto X_i é reconstruído. Essa função de custo, dada pela equação 4.3, é otimizada segundo duas restrições: cada ponto X_i só pode ser reconstruído a partir de seus vizinhos, reforçando que W_{ij} seria 0 se o ponto X_j não fosse vizinho de X_i ; e o somatório de pesos de cada linha da matriz W_{ij} seria igual a 1 , ou $\sum_j W_{ij} = 1$.

Estabelecido o mapeamento dos pesos em D, precisamos mapeá-los em uma dimensão d, menor. Para tal, teremos de minimizar a função de custo:

4.4

$$E(Y) = \sum_i \left| Y_i - \sum_j W_{ij} X_j \right|^2$$

Lembramos que, na maioria dos casos (projetos) em que um cientista de dados ou engenheiro de modelos trabalha, não é necessária toda essa matemática, mas é importante para compreendermos a complexidade por trás da decisão de utilizarmos um redutor de dimensionalidade e nos dar a ideia, por exemplo, de que, para esse caso, o algoritmo de LLE irá mapear todos os dados originais, calcular a distância entre eles num raio de vizinhos próximos, traduzir esse mapa em função desse mapeamento de vizinhos e reconstruir o mesmo mapeamento numa dimensão menor, mantendo essas proporções.

O algoritmo de LLE usa, normalmente, a distância euclidiana para calcular os pesos da matriz de vizinhanças W e tem um parâmetro livre K, que estipula justamente a quantidade de vizinhos a serem observados, lembrando que o K pode ser descoberto a partir do mesmo processo de configuração de hiperparâmetros que vimos anteriormente neste módulo. Quanto maior o número de dimensões e de dados, mais custoso será esse processo, sempre.

Comentário

O LLE é um algoritmo poderoso, adaptável e apresenta boa generalização para resolver problemas não lineares de redução de dimensionalidade. Porém, assim como a grande maioria dos algoritmos não lineares, causa certa dificuldade no quesito explicabilidade.

Implementando PCA em Python

Para a implementação da PCA em Python, vamos precisar das seguintes bibliotecas: sklearn, numpy e pandas.

Para o nosso exemplo, vamos utilizar um dataset IRIS de problemas de classificação.

Começamos carregando o conjunto de dados e importando as bibliotecas com os seguintes comandos:

```
python

import numpy as np
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(url, names=names)
```

Depois, faremos a separação do conjunto de dados em características (features) e rótulo (label). Para isso, utilizaremos o seguinte conjunto de comandos:

```
python

from sklearn.model_selection import train_test_split
X = dataset.drop('Class', 1)
y = dataset['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Lembrando que a PCA funciona melhor quando as características estão na mesma escala. Então, vamos aplicar o StandardScalar da biblioteca sklearn, que é nada mais do que colocar as variáveis em valores de proporção (entre 0 e 1) por coluna/característica. Usaremos o código a seguir:

```
python

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Feito isso, podemos experimentar o PCA. Além de vermos como ele é feito, podemos observar seu efeito prático favorecendo um classificador; no caso, uma Floresta Aleatória, utilizando o seguinte código:

```
python

from sklearn.decomposition import PCA

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

for i in range(X_train.shape[1]):
pca = PCA(n_components=i+1)
pca_X_train = pca.fit_transform(X_train)
pca_X_test = pca.transform(X_test)

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(pca_X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(pca_X_test)
cm = confusion_matrix(y_test, y_pred)
print ("----------------------------------------------------------------")
print("PCA: ", i+1)
print(cm)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Dataset Var: ', pca_X_train.var())
for c in range(pca_X_train.shape[1]):
print("Component(",c+1,"):", pca_X_train[c].var())
print("----------------------------------------------------------------")
```

Neste exemplo, estamos vendo o melhor valor para o número de componentes a serem transformados por meio do laço for de i até o número de colunas, dado por shape[1]. Assim, podemos gerar diferentes conjuntos transformados de PCA, variando o número de componentes e medindo as métricas da floresta aleatória treinada para cada conjunto transformado, bem como a variância dos seus n componentes.

Como estamos trabalhando com um conjunto de dados simples, o uso da PCA aqui é exploratório e não tem um resultado de melhora tão grande, mas, para um conjunto maior, tanto em linhas quanto, principalmente, em colunas, poderemos ver tal melhora mais aparente.

Demonstração da implementação em Python da análise de componente principal (PCA)

No vídeo, a seguir, será demonstrada a implementação da análise de componente principal (PCA) em Python.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

A redução de dimensionalidade é importante para o modelo não sofrer de ____. Qual é a opção que melhor completa a lacuna?

A

Subajuste.

B

Sobreajuste.

C

Dados faltantes.

D

Dados sujos.

E

Variação.

A alternativa B está correta.

A alternativa A não é causada normalmente por maldição de dimensionalidade, mas por esparsidade ou qualidade dos dados. A alternativa C é resolvida por meio de outros métodos da segunda fase do processo de descoberta de conhecimento em base de dados. A alternativa D é causada pela qualidade dos dados, da fonte ou do processo de extração, e não pode ser remediada pelo uso de técnicas de redução de dimensionalidade. A alternativa E não é um problema, mas até, muitas vezes, algo desejado.

Questão 2

A técnica que mapeia os pontos do espaço dimensional de amostras de um conjunto de dados e tenta minimizar uma função de custos baseada numa matriz de pesos dos vizinhos dos pontos observados é:

A

KNN.

B

KMeans.

C

PCA.

D

LLE.

E

SVM.

A alternativa D está correta.

A alternativa A é um classificador baseado no algoritmo dos K vizinhos mais próximos. A alternativa B é um algoritmo de clusterização da família dos algoritmos de aprendizado não supervisionado, que também se baseia no algoritmo de vizinhos mais próximos. A alternativa C é um dos algoritmos de redução de dimensionalidade, porém seu processo de funcionamento é mais simples, uma vez que é linear. A alternativa E é um modelo de aprendizado supervisionado com base em vetores de suporte.

Considerações finais

Como pudemos observar, os modelos de aprendizado supervisionado são muitos e variados, cada um com seus pontos fortes e fracos e com a capacidade de melhor se adaptar a um problema específico.

Vimos que as máquinas de vetores de suporte (SVM) são excelentes classificadores, mas podem ser custosas de treinar e aplicar se o conjunto de dados tiver dimensões muito grandes. Também notamos que, a partir do uso de Álgebra Linear, tais modelos têm alta capacidade de generalização, bastando apenas trocar a função de kernel a ser otimizada.

Como foi abordado, as árvores de decisão são modelos simples, fáceis de entender e, ainda assim, muito eficazes, principalmente para problemas de classificação. Basta apenas prestarmos atenção aos seus hiperparâmetros e teremos um classificador simples, eficiente e econômico.

Se uma árvore de decisão for boa, uma floresta delas pode ser ainda melhor. No módulo 3, exploramos as florestas aleatórias e o raciocínio por trás dos modelos incorporados e por que eles se sobressaem aos demais e ganham tanta popularidade.

Finalmente, no último módulo, ressaltamos o conceito de redução de dimensionalidade, bem como aprendemos a reduzir o conjunto de características de um conjunto de dados sem prejudicar o entendimento do modelo com relação ao referido conjunto. Aprendemos isso de dois modos, de forma linear, com a PCA, e de forma não linear, com a LLE.

Concluímos, assim, o treinamento de modelos de Aprendizado de Máquina. Com isso, você estará habilitado a aplicar esses modelos com as demonstrações e algumas configurações experimentais, como configuração de hiperparâmetros e visualização de resultados.

Podcast

Para encerrar, ouça sobre treinamento de modelos de aprendizado de máquina.

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore +

Ainda que tenhamos discutido bastante sobre o tema de treinamento de modelos de aprendizagem de máquina, é sempre bom estar em dia com o cotidiano da área, procurando artigos recém-publicados nas bibliotecas digitais da nossa área, como IEEE, ACM, Google Scholar, Scopus, Elsevier, Springer etc., bem como fontes confiáveis, como, por exemplo, documentações de bibliotecas estabelecidas nas comunidades de pesquisa e desenvolvimento, a exemplo de scikit-learn, keras e tensorflow.

Visto que nossa área é multidisciplinar, vale a pena pesquisar e se aprofundar em alguns tópicos extras descritos a seguir:

SVM

- Dimensão Vapnik-Chervonenkis.
- Conjuntos Partidos (Shattered Sets).
- Processos Empíricos.
- Documentação da Máquina de Vetores de Suporte implementada na biblioteca Scikit-Learn do Python.

Árvore de Decisão

- Tree Map — Visualização de Dados inspirada no conceito das árvores de decisão.
- Entropia — o conceito além da área de ciência da informação.
- Árvores de Probabilidade.

Florestas Aleatórias

- Ensemble Classifiers.
- Soft Voting.
- Weighted Voting.
- Collective Intelligence.
- Data Fusion.

PCA

- TSNE.

- Kernel PCA.

- Kernel Functions.

- Curse of Dimensionality.

Referências

AMARAL, F. **Aprenda mineração de dados**: teoria e prática. Vol. 1. Rio de Janeiro: Alta Books, 2016.

BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. (1998). **Classification and regression trees**. 1. ed. Ed. Routledge, 2017.

BREIMAN, L. **Random forests**. In: Machine learning, 45(1), p. 5-32, 2001.

DEMIR, N. **Ensemble methods**: elegant techniques to produce improved machine learning results. KDnuggets, 2016. Consultado em meio eletrônico em: 11 fev. 2021.

DIETTERICH, T. G. **Ensemble methods in machine learning**. *In*: International workshop on multiple classifier systems, 2000, p. 1-15, Springer, Berlin, Heidelberg.

MITCHELL, T. M. **Machine learning**. Nova York: McGraw Hill, 1997.

SANTOS, Eulanda Miranda dos. **Teoria e aplicação de support vector machines à aprendizagem e reconhecimento de objetos baseado na aparência.** 2002. 121 f. Dissertação (Mestrado em Informática) Programa de Pós-Graduação em Informática, Centro de Ciências e Tecnologia, Universidade Federal da Paraíba, Campina Grande, Paraíba, Brasil, 2002.

VAPNIK, V.; GUYON, I.; HASTIE, Trevor. **Support vector machines**. *In*: Mach. Learn, v. 20, n. 3, p. 273-297, 1995.

WATANABE, S. **Theorem of the ugly duckling**. *In*: Pattern Recognition: Human and Mechanical, 1985.