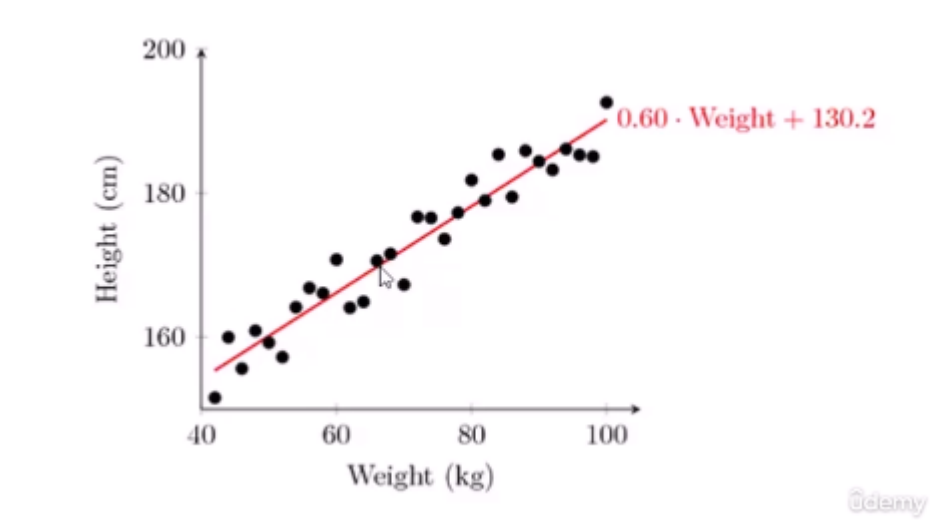


11 - Prática: Predição e a Base de Aprendizado de Máquina (II)

▼ Modelos preditivos

▼ Regressão linear

Alocamos os pontos com dados em um plano. A partir disso conseguimos identificar um determinado desenho neste plano, seguindo os pontos abordados.



Se processo em questão construir uma linha seguindo esses pontos conseguimos prever dados que ainda não estão no nosso dataset, tanto no futuro, quanto passado.

Outro fato é que ideia de regressão não é nada mais que uma simplificação do processo, afinal, é como se reduzíssemos o ruído e focássemos numa tendencia generalista.

Ex: Não nos atemos ao peso acima 80,5 - altura 182, ele não é a reta, mas através dele, conseguimos criar uma reta com média ideal para esse peso

Se tivermos 100 valores com esta mesma numeração, a reta estaria muito próxima dele, se não tivermos nenhum ponto, a reta também

poderia estar próxima dele, afinal, o que convém é a predição futura, sem deter propriamente dos dados.

- O mínimo dos quadrados minimiza a soma dos quadrados erráticos.
- É o mesmo que maximizar a probabilidade dos dados observados, se você começar a pensar no problema em termos de probabilidades e funções de distribuição de probabilidade
- O máximo da estimação da probabilidade

Gradient Descent é uma alternativa para o método de mínimos quadrados

Basicamente inteiro para encontrar a linha que melhor segue os contornos definidos pelos dados.

Passa a fazer sentido gerando através de dados 3d

É simples de tentar em Python

Porém em termos de usabilidade o mínimo dos quadrados é uma ótima escolha

Entendendo erros com o r-squared

Como construímos a medida, através de ajustes na linha ou nos dados

R-square(aka coeficiente do determinante):

A fração do total da variação em Y é capturada pelo modelo

Computando r-squared

$$1.0 - \frac{\text{sum of squared errors}}{\text{sum of squared variation from mean}}$$

0 significa que nenhuma variância foi capturada e 1 significa que toda variância foi capturada

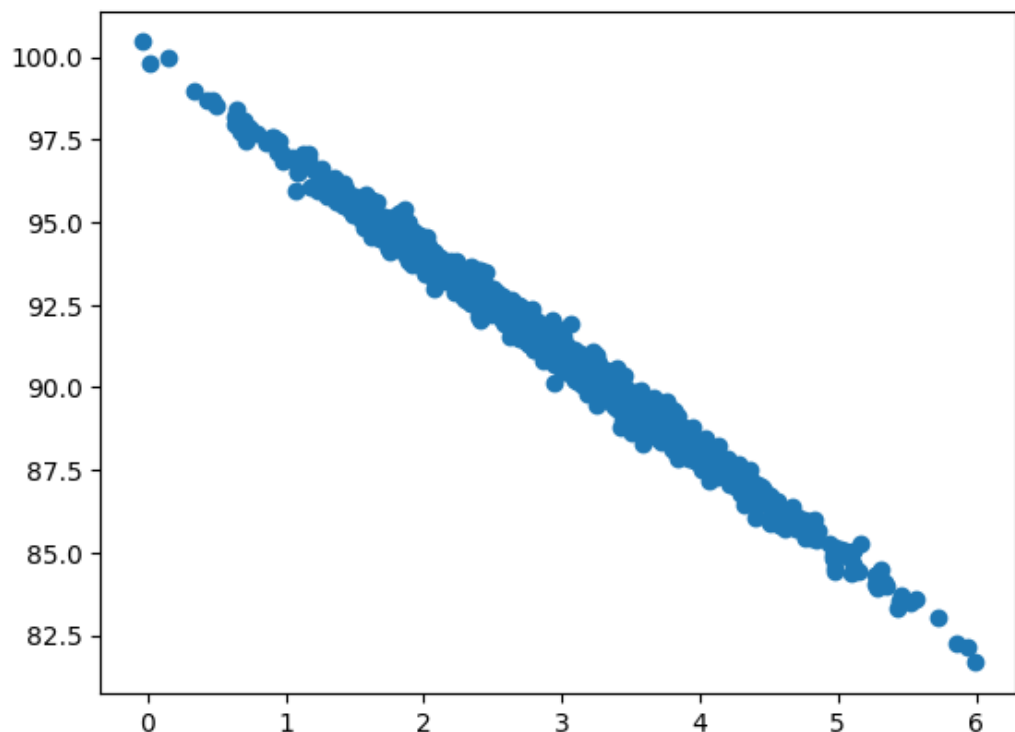
Em python

```
import numpy as np
from random import random
from matplotlib.pyplot import scatter

pageSpeeds = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = 100 - (pageSpeeds + np.random.normal(0, 0.1, 1000)) * 3

scatter(pageSpeeds, purchaseAmount)
```

Ao utilizarmos seguinte comando, estamos criando dados, estes que são utilizados para geração de uma regressão linear:



▼ Regressão Polinomial

Nem todas as relações são lineares

Formula Linear:

$$y = mx + b$$

Essa é a primeira ordem, o primeiro degrau polinomial, do qual o "poder" de x é 1

Segunda ordem/degrau

$$y=ax^2+bx+c$$

Terceira ordem/degrau

$$y=ax^3 + bx^2 + cx + d$$

Ordens/degraus maiores produzem curvas mais complexas

Porém, haver mais degraus não significa que é melhor.

Não utilize mais degraus que você precisa

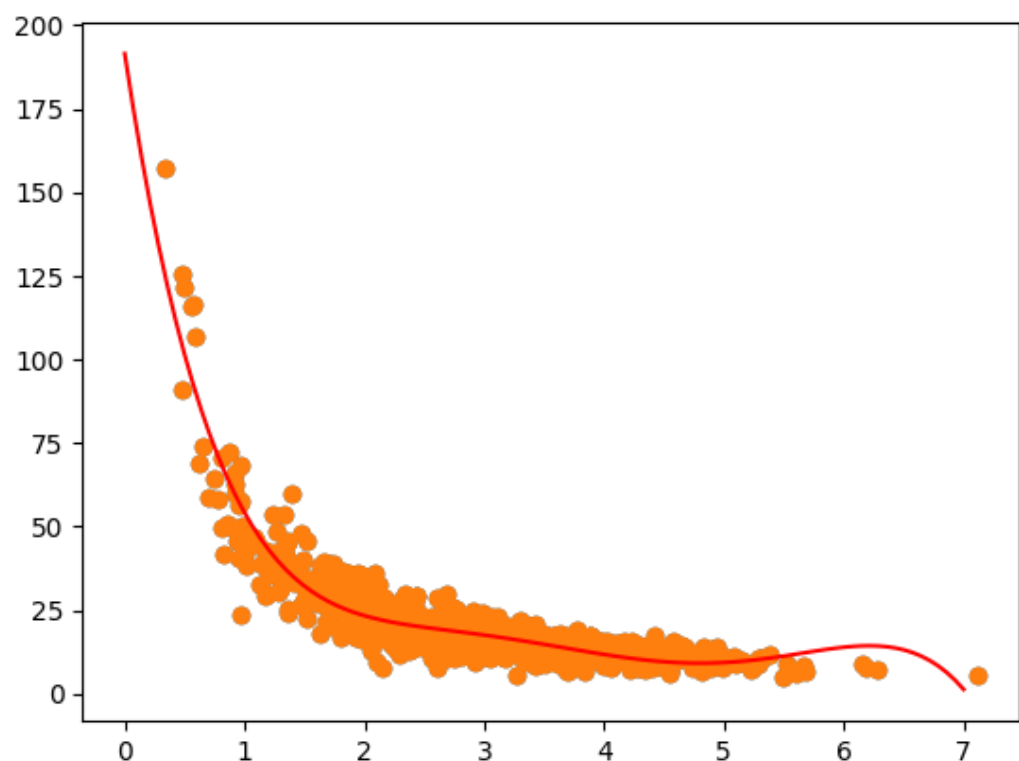
Visualize seus dados com o primeiro e identifique o quão complexo a curva pode realmente ficar.

Visualize o ajuste, sua curva esta indo para fora para acompanhar os outliers?

Um alto r-squared simplifica o significado do encaixe de suas curvas para que possa treinar seus dados. Porém, ela pode não ser um bom preditor

Python

`numpy.polyfit()`



```

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2)
pageSpeeds = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = np.random.normal(50.0, 10.0, 1000) /

plt.scatter(pageSpeeds, purchaseAmount)

x = np.array(pageSpeeds)
y = np.array(purchaseAmount)

p4 = np.poly1d(np.polyfit(x, y, 5))

xp = np.linspace(0, 7, 100)
plt.scatter(x, y)
plt.plot(xp, p4(xp), c='r')
plt.show()

from sklearn.metrics import r2_score

r2 = r2_score(y, p4(x))

print (r2)

```

▼ Multipla Regressão

Quando detemos somente de uma variável de dimensão como preço de alguma coisa, conseguimos ter um certo nível de predição, possibilitado pela análise dos valores já detidos.

Porém se adicionarmos mais dimensões de campos novos: Marca, modelo, Quilometragem, motor, etc.

Conseguimos ter possibilidades diferentes para analisar. Este processo chamamos de multipla regressão

Preço = a + b1 Quilometragem + b2 Idade + b3 Portas

Cada fator tem importância, porém, é claro que alguns não terão tanta relevância no todo, e estes podem ser retirados da equação.

Você ainda pode medir o tamanho usando o r-squared

Precisa assumir diferentes fatores onde eles mesmos tem co-dependencia entre si

```
import pandas as pd

df = pd.read_excel("http://cdn.sundog-soft.com/Udemy/Da

%matplotlib inline
import numpy as np
df1=df[['Mileage', 'Price']]
bins = np.arange(0,50000,10000)
groups = df1.groupby(pd.cut(df1['Mileage'],bins)).mean()
print(groups.head())
groups['Price'].plot.line()
```

Neste exemplo, construímos um gráfico em linha para análise através do valor por quilometragem

▼ **Multi-level Models**

O conceito é que diversos efeitos ocorrem em diferentes níveis.

Sua saúde depende hierarquicamente da saúde dos seus órgãos, de você como um todo, da sua família, da sua cidade, do mundo em que vive;

Sua riqueza depende de onde você trabalha, do que seus pais fizeram, do que seus avós fizeram.

O modelo de multi-nível aborda para o modelo do qual existe interdependências.

Modelagem de múltiplos níveis

Para realizar isto, deverá ser levado em conta todos os fatores que afetam o resultado se você estiver tentando prever cada nível.

Como exemplo, notas SAT, deverá ter em mente para predição os dados genéticos de cada criança, O ambiente individual de cada criança, a taxa de criminalidade da vizinhança onde ela vive, a qualidade dos professores da escola, onde a escola se encontra, as políticas educacionais de cada estado.

Todos estes fatores, afetam cada nível, como exemplo, a taxa de criminalidade afeta o ambiente em casa.

▼ Machine Learning With Python

▼ Aprendizado sem supervisão

Você indica um agrupamento de dados e espera que o algoritmo através de testes encontre correlações entre estes dados.

Diversas bolas azuis, Outras bolas vermelhas, outros cubos azuis e outros cubos vermelhos.

O algoritmo irá analisar esses dados e a resposta dependerá da métrica mais próxima. Ele separará os cubos das bolas? Ele separará as cores? Ele separará os objetos e dentro dos objetos as cores?

Ele estará encontrando classificações diversas para esse agrupamento de dados.

"Talvez você não saiba o que esta buscando, mas esta buscando algo, e através do aprendizado sem supervisão, é possível encontrar variáveis latentes que podem ser utilizada para relacionar informações".

Ex: Um agrupamento de usuários em um site de relacionamento com informações básicas do mesmo e comportamento. Talvez você encontrará um grupo de pessoas que não se conformam com o estereótipo.

Um agrupamento de filmes baseados em suas propriedades, talvez os conceitos do gênero estejam datados.

Analisar o texto da descrição de um produto para encontrar os termos que tem maior significado em determinada categoria.

▼ Aprendizado supervisionado

Você adiciona as perguntas e os dados que detém e busca utilizar o algoritmo para encontrar as corretas. Logo, você detém não somente das perguntas, mas as respostas devidas para elas.

Com o algoritmo treinado, é possível utilizar novas perguntas sem respostas, das quais o modelo irá utilizar do conhecimento adquirido para responde-las.

Train // Test

Basicamente se você detém de uma base de dados relativamente grande. É possível separar ela em 2 partes, sendo a maior utilizada para treino e a menor, utilizada para os testes com este treino.

Precisa ter certeza que a quantidade de dados é grande, afim de encontrar os padrões devidos(construir um bom modelo)

Train // Test não é perfeito

Caso o tamanho dos seus dados sejam baixos é passível de não encontrar o ponto devido

Ou existe uma chave randômica de seus dados treinados e o modelo de treino serem iguais

O sob reajuste pode ocorrer

K-fold Cross Validation

Basicamente pegar seus dados, segmentar eles em diversas partes já havendo parte para aprendizado e parte para treino.

A partir do aprendizado e treino os dados são medidos.

Python

```
%matplotlib inline
import numpy as np
from pylab import *
```

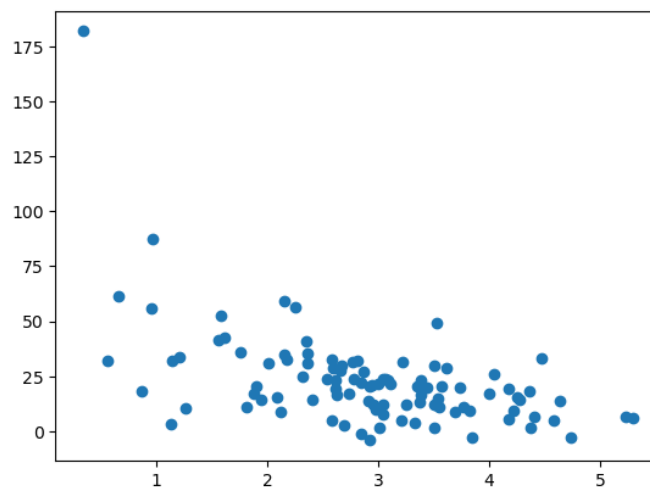


```
np.random.seed(2)

pageSpeeds = np.random.normal(3.0, 1.0, 100)
purchaseAmount = np.random.normal(50.0, 30.0, 100)

scatter(pageSpeeds, purchaseAmount)
```

Em primeiro momento realizamos processo de seleção e formação de uma base de dados, esta que nos retorna a partir da geração de um gráfico scatter o seguinte desenho:

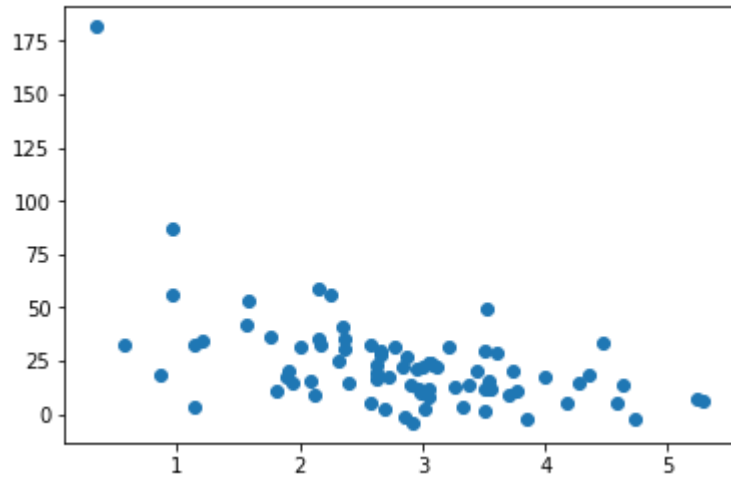


```
trainX = pageSpeeds[:80]
testX = pageSpeeds[80:]

trainY = purchaseAmount[:80]
testY = purchaseAmount[80:]
scatter(trainX, trainY)
```

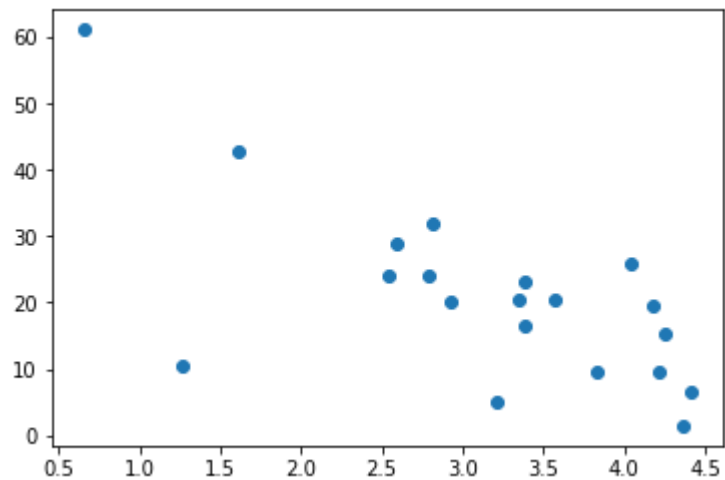
Feito isto, é definido as fatias dos dados.

Sendo train os dados de treino e test os dados de teste. É visível a definição do array a partir de x valor através do valor entre [:]. A partir disso se gerarmos um gráfico Scatter temos seguinte desenho:



Enquanto os valores de teste, representam:

```
scatter(testX, testY)
```



Agora, devemos definir o oitavo degrau polinomial com estes dados(o que quase certamente é um ajuste excessivo, considerando o que sabemos sobre como ele foi gerado)

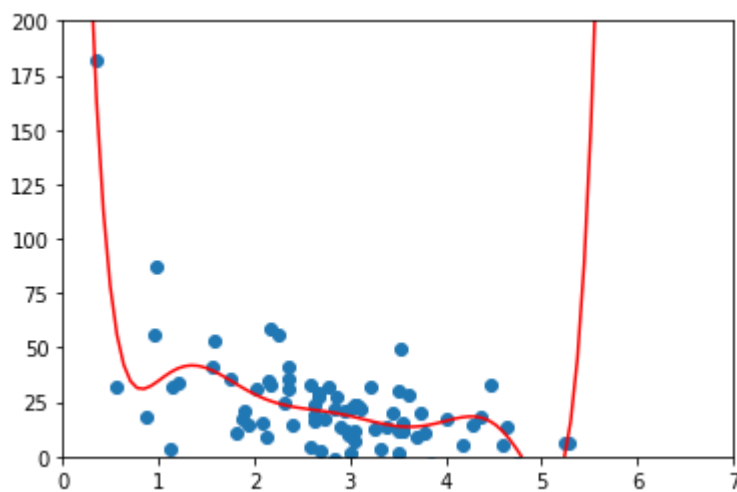
```
x = np.array(trainX)
y = np.array(trainY)

p4 = np.poly1d(np.polyfit(x, y, 8))
```

A partir disso, vamos gerar um scatter plot deste gráfico:

```
import matplotlib.pyplot as plt

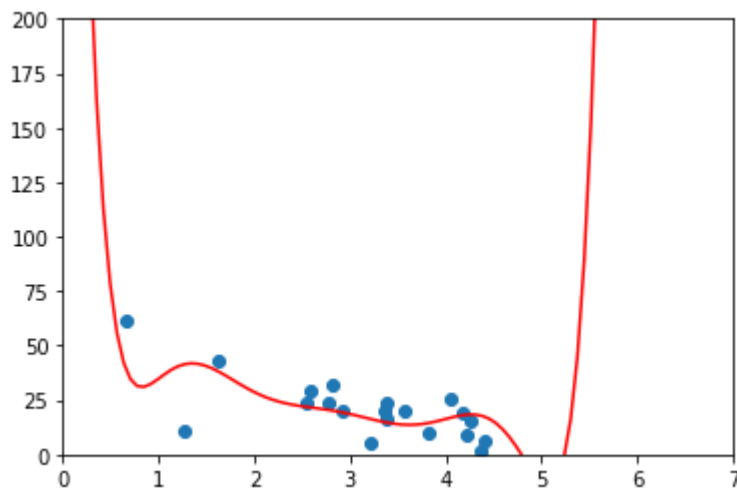
xp = np.linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0, 7])
axes.set_ylim([0, 200])
plt.scatter(x, y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```



Como é possível validar ele encontrou o “Meio dos dados” porém, teve um sobreajuste excessivo, caso direcionemos o teste através dos dados de teste:

```
testx = np.array(testX)
testy = np.array(testY)

axes = plt.axes()
axes.set_xlim([0, 7])
axes.set_ylim([0, 200])
plt.scatter(testx, testy)
plt.plot(xp, p4(xp), c='r')
plt.show()
```



Podemos averiguar que também seguem o padrão anterior, havendo um formato correspondente aos dados, porém com um massivo sobreajuste, basicamente o score do R-square é horrível, sendo assim modelo não está bom.

É possível, utilizarmos a função `r2` do **sklearn.metrics** para conseguirmos melhorar isto.

```
from sklearn.metrics import r2_score

r2 = r2_score(testy, p4(testx))

print(r2)
```

Retorno = 0.3001816861141787

Treinando assim novamente nossos dados:

```
from sklearn.metrics import r2_score

r2 = r2_score(np.array(trainY), p4(np.array(trainX)))

print(r2)
```

Resultado = 0.6427069514694241

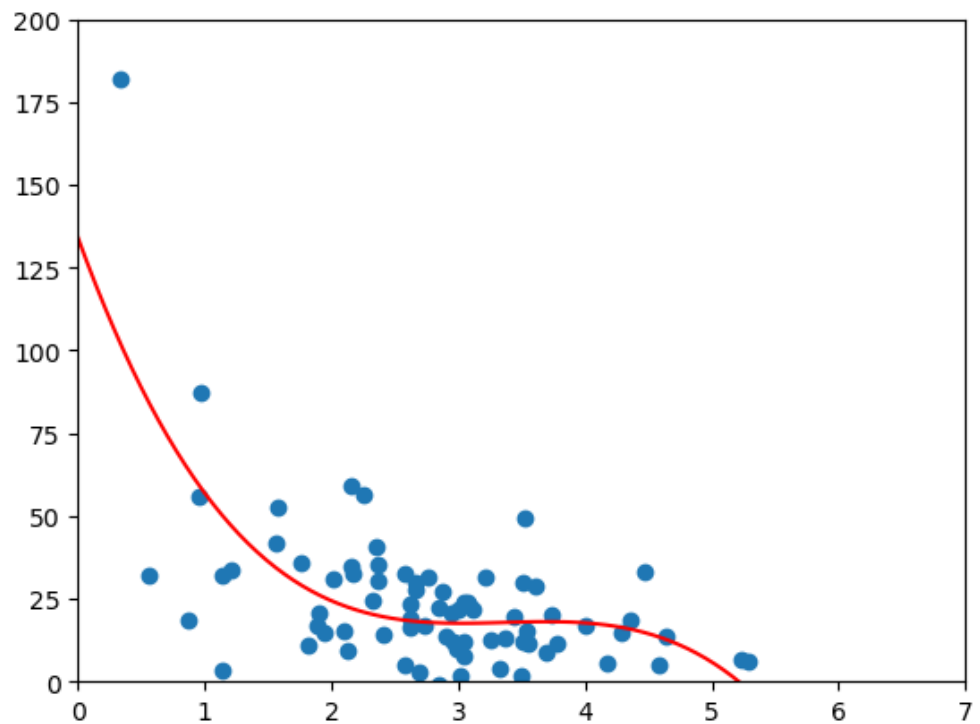
Atividade:

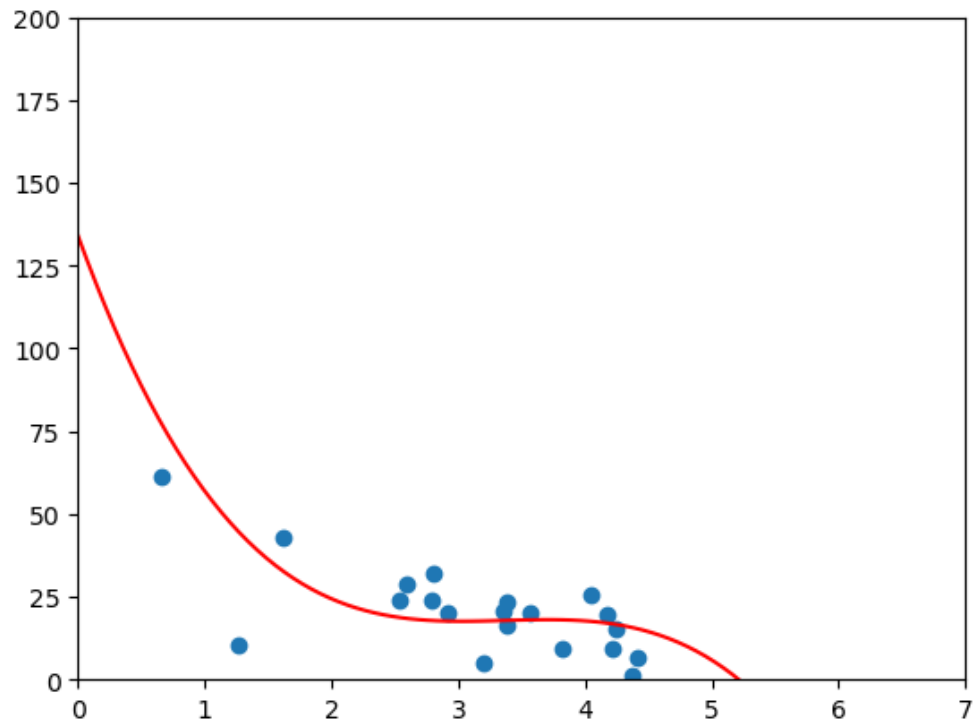
Através do ajuste da função polinomial no terceiro grau

```
x = np.array(trainX)
y = np.array(trainY)

p4 = np.poly1d(np.polyfit(x, y, 3))
```

Encontrei um desenho que não detém de sobre ajuste excessivo





Tornando assim, tanto os dados de treino, quanto de testes num desenho utilizavel.

▼ Métodos de aprendizagem de maquina

▼ Bayesian Methods

É um classificador para que possamos identificar se algo pode vir a ser um spam ou não.

Basicamente a probabilidade de uma palavra ser um spam pode ser alta, mas de um spam conter esta determinada palavra não. O que o algoritmo em si faz é analisar palavra por palavra em individual, enquanto nossa maquina passa a aprender com isto.

Basicamente um grande classificador de palavras, do qual busca aquelas que tem maior proximidade com a probabilidade de se tornarem spam, para predizer se um email é spam ou não. Este processo damos o nome de Naive Bayes

Bibliotecas para auxilio:

Scikit-learn

The countVectorizer

MultinomialNB

▼ Funcionamento através do python

A principio é necessário a criação de um dataframe, este que conterá 2 colunas essenciais.

A primeira será o conteúdo do email, este que detém das palavras em questão.

Já o segundo será classe, que conterá se este email é spam ou não.

Este será um aprendizado com supervisão, pelo fato de você dar input da saída que deseja para que ML aprenda com isto.

```
vectorizer = CountVectorizer()  
counts = vectorizer.fit_transform(data['message'].values)  
  
classifier = MultinomialNB()  
targets = data['class'].values  
classifier.fit(counts, targets)
```

Feita entrada destes dados, realizamos 3 processos. O primeiro é abertura de uma variavel com nome "Vectorizer" da qual chama a função CountVectorizer()

Após isto, ele utiliza uma outra variavel Counts que irá utilizar a contagem gerada pela variavel que chamou a função "Vectorizer" transformando os dados da coluna Message do dataframe.

Feito isto, Abrimos uma nova variavel que será nosso classificador, este que será o que definirá se alguma coisa será ou não spam, ou email normal.

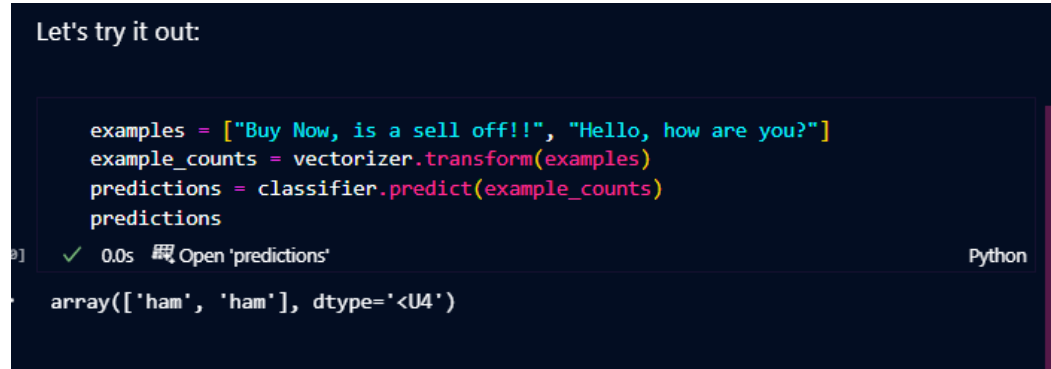
Para isto ele utiliza dos dados que encontramos através da variavel Class. Muito semelhante a definirmos para nós através de uma planilha, filtrando as classes entre si.

Após isto, ele utiliza esse classificador com as duas variáveis para que o ML aprenda e utilize isto para um teste futuro

```
examples = ["Free Viagra now!!!", "Hi Bob, how about  
example_counts = vectorizer.transform(examples)  
predictions = classifier.predict(example_counts)  
predictions
```

```
Retorno = array(['spam', 'ham'], dtype='<U4')
```

Sendo assim, primeira mensagem seria considerada um Spam, enquanto a segunda não.



```
Let's try it out:

examples = ["Buy Now, is a sell off!!", "Hello, how are you?"]
example_counts = vectorizer.transform(examples)
predictions = classifier.predict(example_counts)
predictions

array(['ham', 'ham'], dtype='<U4')
```

Porém, como é possível ver acima, muito possivelmente o que considero um Spam, não trás como padrão o que é definido.

Isso se dá pela base utilizada, validando estes dados dos arquivos, os mesmos não são tão, específicos com determinados temas.

▼ K-Means Clustering

Separar em diversos grupos, afim de que algum componente entre os mesmos se ligue.

Como exemplo temos uma escola, da qual alunos do 1 gostam de usar roupas marrons

2 ano gostam de utilizar vermelho

3 ano gostam de utilizar preto

Deste modo através da idade/periodo escolar, conseguimos trazer 3 grupos, dos quais estes grupos tem outra classificação referente a cor que tem preferencia.

Isto permite que ações possam ser tomadas, como a não exigência de uma cor especifica de camisa para um determinado grupo.

▼ Escolhendo o K

Vá acrescentando K valores até que você consiga uma grande redução entre cada ponto e o centro.

▼ Evite mínimos locais

Randomize a escolha do centro inicial do seu cluster, para ter resultados diferentes

Rode diversas vezes até ter certeza de que os resultados iniciais não são malucos

▼ Rotule seu cluster

K-means não tenta atribuir qualquer significado no que o cluster encontrar.

Fica sob sua responsabilidade determinar isto.

▼ Python

Para trabalhar com Python, primeiro realizamos o processo de criação de uma função que irá identificar alguns pontos randomicos e definir eles como o "centroid" bem como onde os pontos de dados serão alocados:

```
from numpy import random, array

# Create fake income/age clusters for N people in k clusters
def createClusteredData(N, k):
    random.seed(10)
    pointsPerCluster = float(N) / k
    X = []
    for i in range(k):
        incomeCentroid = random.uniform(20000.0, 200000.0)
        ageCentroid = random.uniform(20.0, 70.0)
        for j in range(int(pointsPerCluster)):
            X.append(
                [
                    random.normal(incomeCentroid, 10000.0),
                    random.normal(ageCentroid, 2.0),
                ]
            )
```

```
X = array(X)
return X
```

Basicamente é uma chamada de função que define os pontos baseados na ideia de separação de pontos por cluster. Logo na variável `pointPerCluster` temos no exemplo que utilizei $100/5 = 20$ pontos.

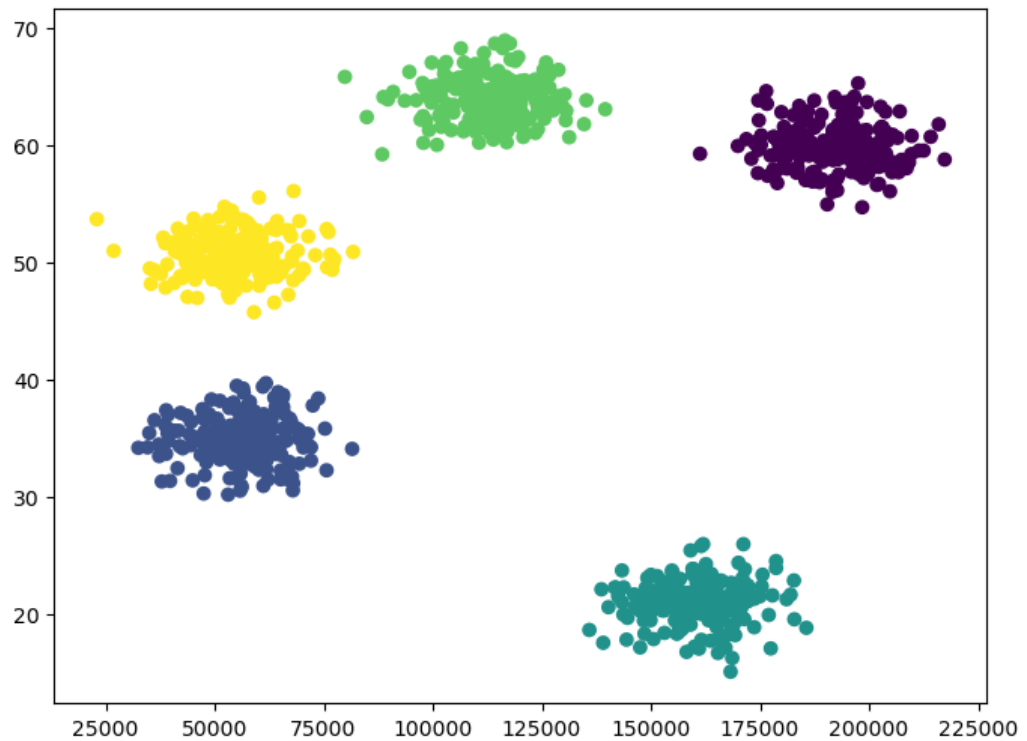
abaixo existe uma estrutura de repetição com range de 5 que no meu caso foi definido por 5 do qual passa a definir pontos nele, tanto de quando essas pessoas recebem quando de idade.

Após isto, ele roda uma função para alocação dos pontos por cluster para assim, definir no array `x` em que de fato cada cluster os valores irão entrar.

```
data = createClusteredData(1000, 5)

model = KMeans(n_clusters=5)
```

A aplicação de valores diferentes tanto na quantidade de dados definida pelo 1000 quanto pela quantidade de 5 clusters muda a geração da nossa formação do respectivo gráfico:



Quanto mais pontos de dados, mais os clusters acabam se agrupando. Quanto mais clusters, maior a possibilidade dos pontos acabarem se unindo.

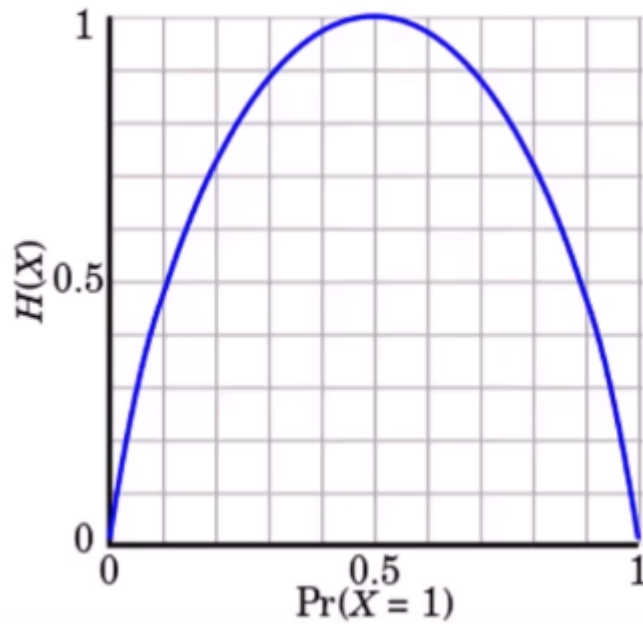
▼ Entropia

A maior parte dos dados estão definidos em desordem, como definir as diferenças e igualdades entre si.

Se classificar os dados em N diferentes classes (Atributos de animais e espécies)

A entropia é 0 se todas as classes desse dado forem iguais (Todos os animais são elefantes)

Com uma entropia alta todos os dados são diferentes



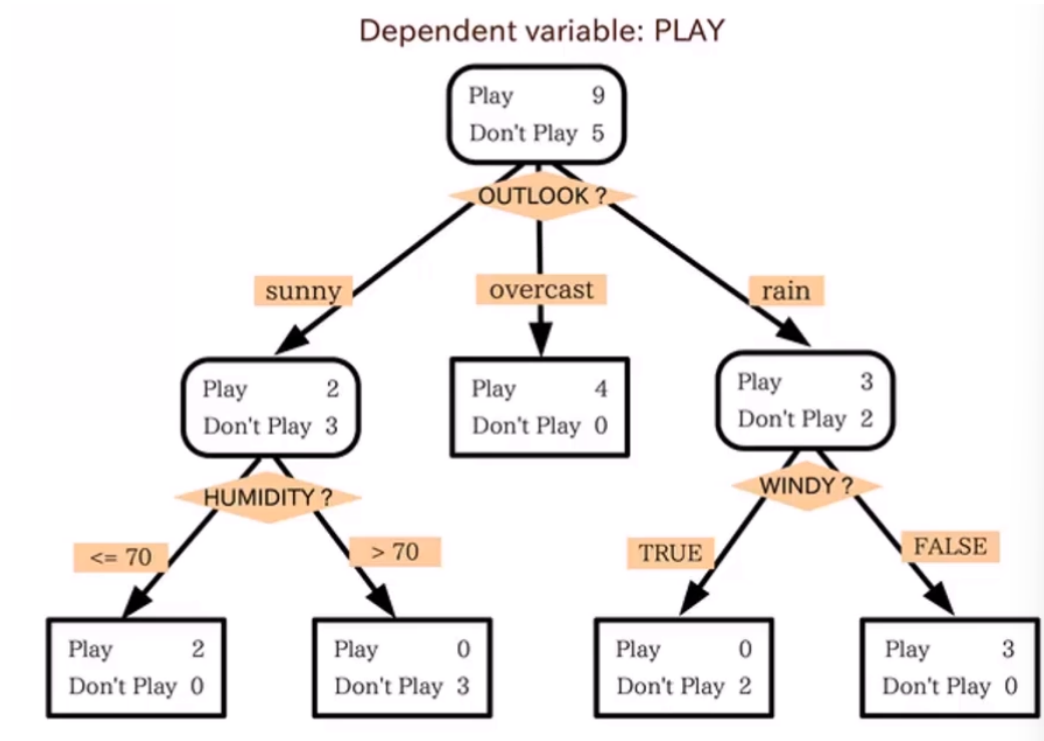
Em poucas palavras é o quão desordenado seu DataSet se encontra e o quanto um objeto esta diferente do todo.

Ex: Ponto vermelho em uma parede de blocos brancos.

▼ Decision Tree

Geração de fluxogramas em Python para tomada de decisões utilizando aprendizado de maquina.

Possibilitando assim identificação de predições e classificações com base na arvore de decisão.



Você detém de informações base, das quais, utiliza para classifica-las em etapas menores. Com estas etapas menores, consegue identificar se sua decisão esta de acordo ou não com os parametros, tendo maior possibilidade de dar certo.

No exemplo acima, é um histórico de jogo, do qual é levado em conta alguns pontos.

Se houver sol, temos uma escala negativa, do qual 2 pontos para jogar e 3 para não jogar, a partir disso, geramos mais 2 perguntas, com outros parâmetros destes dados, como exemplo da humidade do local, se ela for superior a 70, o é definido 3 pontos para não jogar, enquanto se humidade for menor, há 2 pontos para se jogar.

Caso esteja nublado, por ter 4 pontos diretos em jogar, não levamos em conta outro parâmetro para redução.

Já se estiver chovendo, temos 3 pontos para jogar e 2 para não jogar, caso esteja ventando, os 2 pontos ficam com não jogar e caso não esteja 3 pontos em jogar.

Você detém de dados de um RH para seleção de currículos, irá fazer uma nova chamada e precisa filtrar os novos currículos.

É possível utilizar estes dados que já detém, afim de que a partir destes dados históricos, consiga treinar a árvore de decisão com estes dados afim de prever se um candidato será contratado com base nisto.

Candidate ID	Years Experience	Employed?	Previous employers	Level of Education	Top-tier school	Interned	Hired
0	10	1	4	0	0	0	1
1	0	0	0	0	1	1	1
2	7	0	6	0	0	0	0
3	2	1	1	1	1	0	1
4	20	0	2	2	1	0	0

Em cada passo, você busca encontrar um atributo, que pode utilizar para particionar seus dados e minimizar a entropia destes dados.

Chamamos isto de ID3

A medida que ele desce na árvore ele busca identificar decisões que irão reduzir a entropia.

Pode ou não resultar em uma árvore ideal.

Um grande problema da árvore de decisão é que ela é suscetível a subajuste.

A alternativa para combater isto, é que podemos construir diversas árvores de decisão e votar na que tiver a melhor classificação final.

É possível regerar o modelo de cada árvore, o termo para isto é Bootstrap Aggregating or Bagging

Randomizar o subsets do atributo a cada passo é uma forma de se escolher entre as formas

No Python:

```
import numpy as np
import pandas as pd
from sklearn import tree
```

```
input_file = "C:/Users/leofe/Downloads/MLCourse/PastHir
df = pd.read_csv(input_file, header=0)
```

No comando acima, realizo inicialização dos dados, e criação de um dataframe através do pandas, o dataframe contém 12 linhas de dados:

	Years Experience	Employed?	Previous employers	Level of Education	Top-tier school	Interned	Hired
0	10	Y	4	BS	N	N	Y
1	0	N	0	BS	Y	Y	Y
2	7	N	6	BS	N	N	N
3	2	Y	1	MS	Y	N	Y
4	20	N	2	PhD	Y	N	N
5	0	N	0	PhD	Y	Y	Y
6	5	Y	2	MS	N	Y	Y
7	3	N	1	BS	N	Y	Y
8	15	Y	5	BS	N	N	Y
9	0	N	0	BS	N	N	N
10	1	N	1	PhD	Y	N	N
11	4	Y	1	BS	N	Y	Y
12	0	N	0	PhD	Y	N	Y

A partir dele, realizo seguinte processo via python:

```
d = {"Y": 1, "N": 0}
df["Hired"] = df["Hired"].map(d)
df["Employed?"] = df["Employed?"].map(d)
df["Top-tier school"] = df["Top-tier school"].map(d)
df["Interned"] = df["Interned"].map(d)
d = {"BS": 0, "MS": 1, "PhD": 2}
df["Level of Education"] = df["Level of Education"].map
df.head()
```

Este código acima, basicamente transforma os valores que estão como letras em valores numéricos, possibilitando que consigamos classificar estes dados. Sendo assim, os valores Y se tornam 1 e N 0, enquanto o valor BS, MS e PHD se tornam 0, 1, 2.

Após executar este processo, nossos dados ficaram do seguinte modo:

	Years Experience	Employed?	Previous employers	Level of Education	Top-tier school	Interned	Hired
0	10	1	4	0	0	0	1
1	0	0	0	0	1	1	1
2	7	0	6	0	0	0	0
3	2	1	1	1	1	0	1
4	20	0	2	2	1	0	0
5	0	0	0	2	1	1	1
6	5	1	2	1	0	1	1
7	3	0	1	0	0	1	1
8	15	1	5	0	0	0	1
9	0	0	0	0	0	0	0
10	1	0	1	2	1	0	0
11	4	1	1	0	0	1	1
12	0	0	0	2	1	0	1

Tendo estes dados, realizamos a separação das Features destes dados, através das colunas de definição do dataframe:

```
features = list(df.columns[:6])
features
```

trazendo seguinte retorno

```
['Years Experience',
 'Employed?',
 'Previous employers',
 'Level of Education',
 'Top-tier school',
 'Interned']
```

Após isto, conseguimos construir nossa árvore de decisão, tendo o seguinte comando:

```
y = df["Hired"]
X = df[features]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X,y)
```

Nele definimos X e y, sendo y o valor de contratado, e X as features.

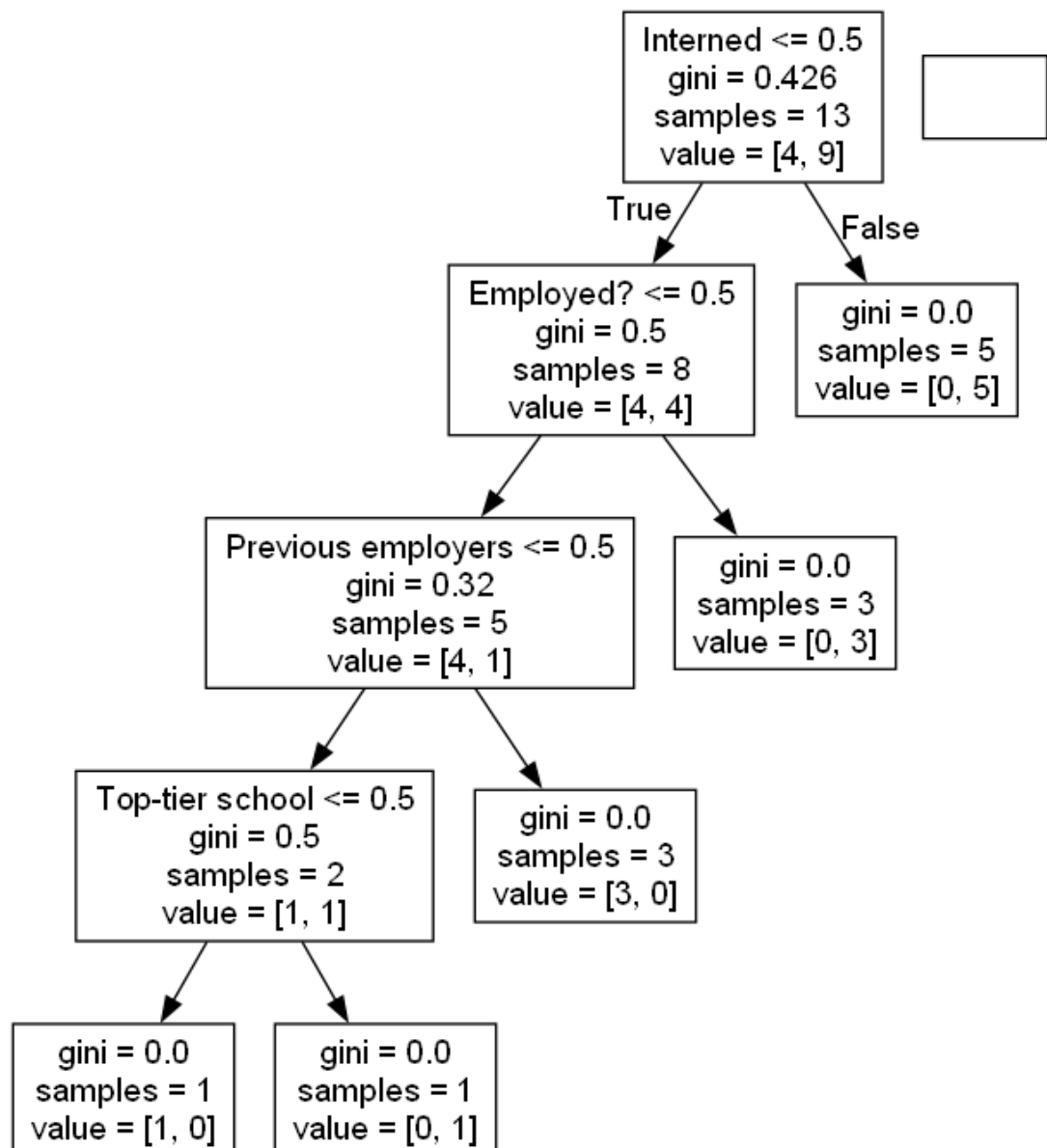

```

from IPython.display import Image
from io import StringIO
import pydotplus

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                    feature_names=features)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

```

E então realizamos o processo de definição da árvore com seguinte comando acima:



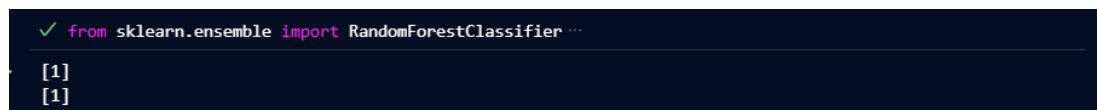
Utilizando esse molde, conseguimos identificar quais parâmetros estão sendo usados para aprendizado deste modelo.

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=10)
clf = clf.fit(X, y)

#Predict employment of an employed 10-year veteran
print (clf.predict([[10, 1, 4, 0, 0, 0]]))
#...and an unemployed 10-year veteran
print (clf.predict([[10, 0, 4, 0, 0, 0]]))
```

E então a partir do seguinte comando, realizamos a randomização do modelo, afim de que encontremos variações encima da construção dele, isto permite que consigamos ter efeitos diferentes de predições.



```
✓ from sklearn.ensemble import RandomForestClassifier ...
[1]
[1]
```

Sendo estes valores teste acima, os 2 como positivos, logo, contratados.

▼ Ensemble Learning

Utilização de diferentes modelos afim de encontrar resoluções diferentes para o mesmo problema, afim de votar no que melhor resulta num bom modelo

Bagging:

Muitos modelos criados para levar de modo randômica os filtros de features

Boosting

É um tipo de modelo evolucionário, basicamente a cada sequência do modelo, ele impulsiona os atributos que no modelo anterior ele classificou incorretamente.

Bucket of models

Roda diferentes modelos utilizando os dados de treino e utiliza o que melhor teve funcionamento com os dados de teste

Stacking

Roda diversos modelos nos dados e combina os resultados entre si.

Esse foi modelo utilizado pela netflix para sugestão de filmes

▼ Advanced Ensemble Learning

Bayes Optimal Classifier

É teoricamente o melhor, porém não consegue ser aplicado na maioria das vezes

Bayesian Parameter Averaging

Procura fazer a prática de BOC, porém ele continua incompreendido, suscetível a sobreajuste e frequentemente superado por uma aproximação mais simples.

Bayesian model combination

Tenta resolver todos os problemas

Porém, é praticamente a mesma coisa que utilizar a cross-validation para encontrar melhor combinação de modelos.

▼ XGBoost

eXtreme Gradient Boosted trees

Cada árvore impulsiona os atributos menos classificados da árvore anterior.

Rotineiramente vence competições do Kaggle

Fácil de utilizar

Rápido

Uma boa escolha para começar

Funções

Impulso regularizado(Previne o sobreajuste)

Pode lidar com a falta de valores automaticamente

Processo paralelo

Pode "Cross-Validate" em cada iteração

Permite a parada antecipada encontrando o número ideal de iterações

Treino incremental

Pode conectar-se com a otimização dos seus objetivos

Tree pruning

Geralmente os resultados são profundos(Gigantes) porém árvores otimizadas

Utilizando XGBoost

Instalação:

Pip install xgboost

Ele não é feito para para o scikit_learn

Utiliza DMatrix

Pode criar arrays do numpy

Todos parametros são passados via dicionário

Hyperparameters

Booster

gbtree ou gblinear

objetivo(ie, multi:softmax, multi:softprob)

Eta(learning rate = Ajuste altura de cada passo)

Max_depth(Profundidade da arvore)

Min_child_weight

Pode controlar o sobreajuste, porém se muito acaba gerando underfit

Python

O primeiro processo que vamos realizar é a importação dos dados que serão utilizados, estes que é o dataset de validação de Iris de flores.

```
from sklearn.datasets import load_iris

iris = load_iris()

numSamples, numFeatures = iris.data.shape
print(numSamples)
print(numFeatures)
print(list(iris.target_names))
```

Após carregar, dividimos nossos dados, afim de separarmos os dados de treino, dos dados que serão utilizados:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split
```

Isto faz com que nossos treinos sejam 80% dos dados, enquanto o teste 20%

```
import xgboost as xgb

train = xgb.DMatrix(X_train, label=y_train)
test = xgb.DMatrix(X_test, label=y_test)
```

Após isto, carregamos o xgb e convertemos nossos dados no modelo do xgb

```
param = {
    'max_depth': 4,
    'eta': 0.3,
    'objective': 'multi:softmax',
    'num_class': 3}
epochs = 10
```

Definimos os hypparameters, sendo máximo de profundidade da árvore 4, processo de aprendizagem como 0,3, o objetivo como multi:softmax, numero de classificações e por fim a quantidade de vezes que o algoritmo será rodado

```
model = xgb.train(param, train, epochs)
```

Rodamos então nosso treino de dados

```
predictions = model.predict(test)
```

Realizamos o teste

```
from sklearn.metrics import accuracy_score  
  
accuracy_score(y_test, predictions)
```

E a partir disso, utilizamos um método da biblioteca sklearn afim de validar a metrica de acuracia.

Ao rodar, pude encontrar medida de 1.0, sendo exata a classificação.

Sendo assim, algoritmo rápido, relativamente simples e que neste caso em questão, encontrou exata resposta para problema.

▼ Support Vector Machines(SVM) Overview

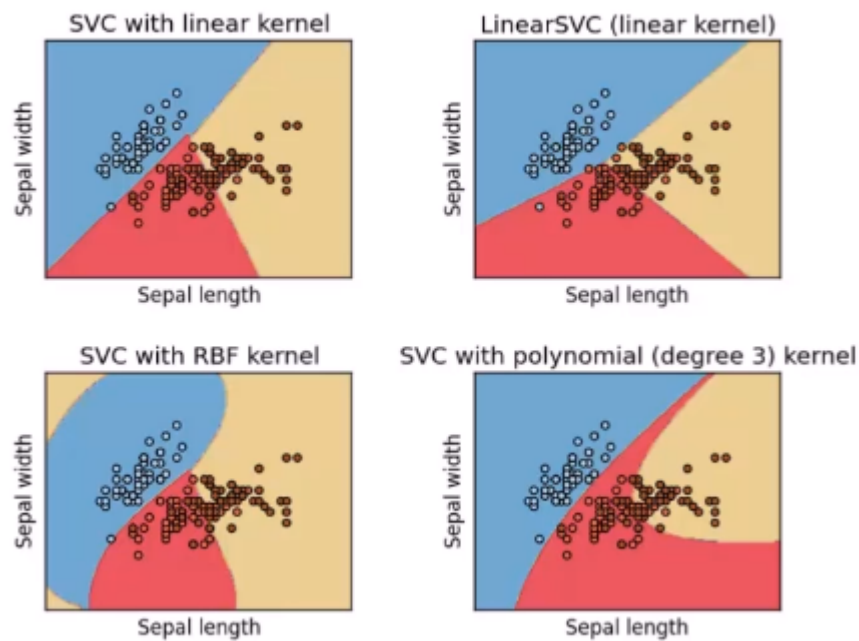
Trabalha bem com Classificação de dados com muitas dimensões(Muitas features/colunas)

Encontra através maior dimensão o "Support Vectores" através da divisão dos dados.

Utiliza de um processo chamado "Kernel Trick" para representar os dados Super dimensionais para encontrar os hiperplanos que podem não ser aparentes em dimensões mais baixas

Na prática é utilizado chamando SVC para classificar os dados usando SVM.

Você pode utilizar diferentes Kernels com SVC, cada qual terá um resultado diferente entre si



Basicamente utilizando diferentes modelos de kernel, alcançamos visualizações diferentes:

```
from sklearn import svm, datasets

C = 1.0
svc = svm.SVC(kernel='linear', C=C).fit(X, y)
```

Através do código acima, conseguimos transformar pontos vindos de X e y de um cluster em uma visualização plotada:

```
def plotPredictions(clf):
    # Create a dense grid of points to sample
    xx, yy = np.meshgrid(np.arange(-1, 1, .001),
                        np.arange(-1, 1, .001))

    # Convert to Numpy arrays
    npx = xx.ravel()
    npy = yy.ravel()

    # Convert to a list of 2D (income, age) points
    samplePoints = np.c_[npx, npy]
```

```

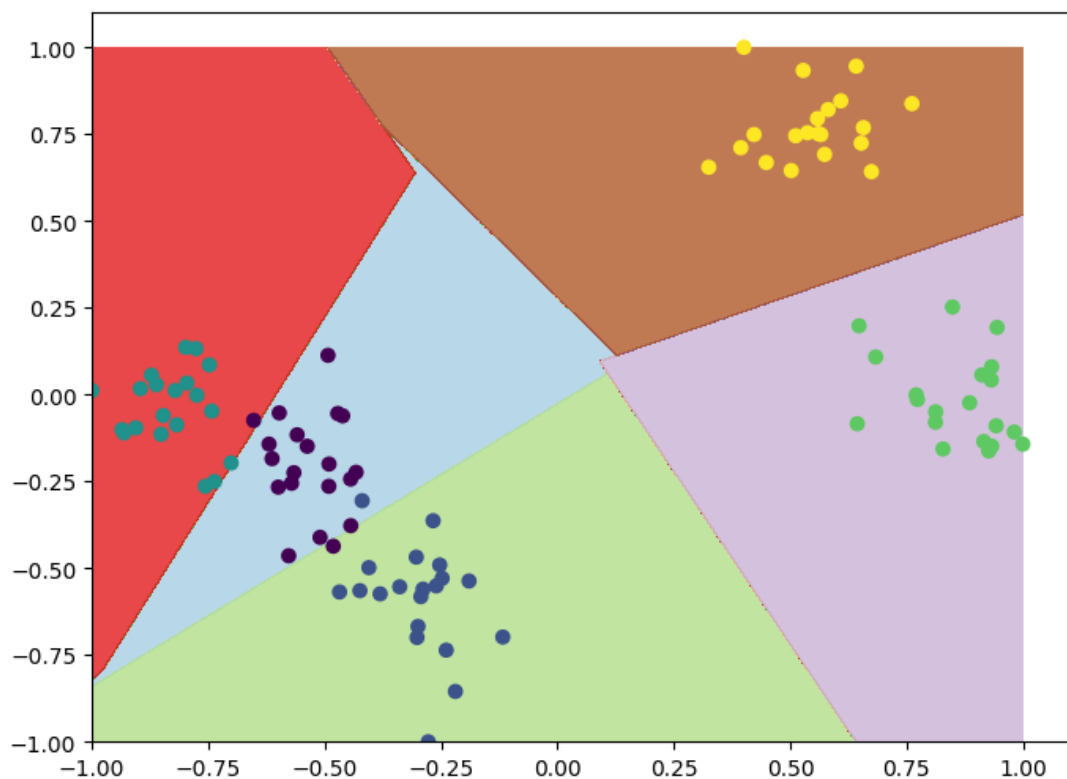
# Generate predicted labels (cluster numbers) for e
Z = clf.predict(samplePoints)

plt.figure(figsize=(8, 6))
Z = Z.reshape(xx.shape) #Reshape results to match x
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0
plt.scatter(X[:,0], X[:,1], c=y.astype(float)) # Dr
plt.show()

plotPredictions(svc)

```

Ao indicar função acima, que organiza o grid, alcançamos seguinte visualização:



E então. Ao realizarmos um novo teste, alcançamos o resultado mediante ao mesmo:

```

print(svc.predict(scaling.transform([[200000, 40]])))
print(svc.predict(scaling.transform([[50000, 65]])))

```



```
✓ print(svc.predict(scaling.transform([[200000, 40]]))) ...  
... [3]  
    [2]
```

Logo, conseguimos predizer se uma pessoa com 200000 recebimento ficaria no grupo 1-2-3-4-5

Ficando no grupo 3 referente a classificação.

Mesmo caso com pessoa que recebe 50000 da qual estará no grupo 2.