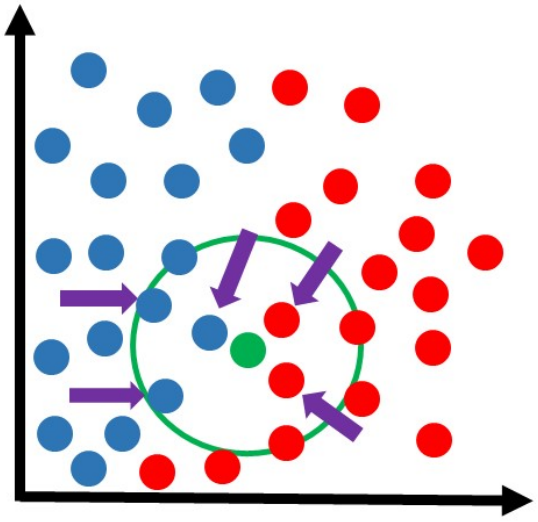
10 - Prática: Lidando com Dados do Mundo Real (II)

▼ KNN(K Nearest Neighbor)

Utilizado para classificar um novo ponto de dado, baseado na distancia dos dados já conhecidos.

Encontre o KNN, baseado na distancia da métrica



O que é buscado encontrar através primeira aula, é o quanto 2 filmes diferentes podem ter estilos semelhantes. Isto se dá pelo uso do KNN.

Uma das fórmulas utilizadas, foi utilizando um arquivo u.item para analise, este que continha dados formatados do seguinte modo:

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title
```

Como é possivel ver, este arquivo, detém de algumas separações.

1. ID(Index)
2. Nome
3. Data

4. URL
5. Sequências de 0s e 1s representando o que este filme tem(ação, comédia, investigação, terror,...).

```
import pandas as pd
import numpy as np
r_cols = ['user_id', 'movie_id', 'rating']
ratings = pd.read_csv('/content/u.data', sep='\t', names=r
```

Durante analise foi pego outra base chamada de u.data e utilizado de um agrupamento por ID de filme, unindo ao tamanho e média destas informações contidas no u.data:

```
movieProperties = ratings.groupby('movie_id').agg({'rating
movieProperties.head()
```

Para que então, pudesse ser normalizado os valores, processo de normalização:

> "O objetivo da normalização é mudar os **valores** das colunas numéricas no conjunto de dados para usar uma escala comum, sem distorcer as diferenças nos intervalos de **valores** nem perder informações. A normalização também é necessária para alguns algoritmos para modelar os dados corretamente."

O cálculo da normalização se dá do seguinte modo:

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Basicamente este cálculo se dá para encontrar um valor entre 0 e 1, tornando mais simples encontrar relações.

O cálculo realizado em aula foi:

```
movieNumRatings = pd.DataFrame(movieProperties['rating']['
movieNormalizedNumRatings = movieNumRatings.apply(lambda x
movieNormalizedNumRatings.head()
```

Encontrado este valor através da u.data, criamos uma biblioteca utilizando o u.item, não retendo dos dados de alguns campos:

```
movieDict = {}
with open(r'/content/u.item', encoding = "ISO-8859-1") as
  temp = ''
  for line in f:
    fields = line.rstrip('\n').split('|')
    movieID = int(fields[0])
    name = fields[1]
    genres = fields[5:25]
    genres = map(int, genres)
    movieDict[movieID] = (name, np.array(list(genres)), mo
```

Isso possibilitou criar o dicionário acima.

Havendo ele, construimos uma função, da qual compara 2 filmes e retorna o quão próximo são ou não, sendo mais próximo de 0 mais e mais próximo de 1 menos

```
from scipy import spatial

def computeDistance(a, b):
  genresA = a[1]
  genresB = b[1]
  genreDistance = spatial.distance.cosine(genresA, genresB
  popularityA = a[2]
  popularityB = b[2]
  popularityDistance = abs(popularityA - popularityB)
  return genreDistance + popularityDistance

computeDistance(movieDict[2], movieDict[4])
```

Para este caso do teste acima, o retorno foi:

```
0.8004574042309892
```

Sendo estes 2 filmes bem diferentes um do outro.

A partir disso, criamos outra função, esta que realiza o processo de geração dos pontos X e Y para um modelo de gráfico, através das informações que temos para geração do KNN.

```
import operator

def getNeighbors (movieId, K):
    distances = []
    for movie in movieDict:
      if (movie != movieID):
          dist = computeDistance(movieDict[movieID], movie
          distances.append((movie,dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(K):
        neighbors.append(distances[x][0])
    return neighbors
K=10
avgRating = 0
neighbors = getNeighbors(1, K)
for neighbor in neighbors:
    avgRating += movieDict[neighbor][3]
    print (movieDict[neighbor][0] + " " + str(movieDict[ne

avgRating /= float(K)
```

▼ Redução de dimensionalidade

Curse of Dimensionality (Pesadelo da dimensionalidade):

Tratar de um grande quantidade de valores, dos quais cada um tem diversas dimensões, que divergem um dos outros pode ser um problema.

> Como exemplo, podemos considerar um vetor do qual cada filme representa um valor, cada valor sendo uma própria dimensão.

Isto acaba criando um grande problema de visualização.

A redução de dimensionalidade tenta destilar essa alta quantidade de dados com um pequeno numero de dimensões, buscando preservar o

máximo a variância dos dados possível

▼ PCA(Principal Component Analysis)

Matemática sofisticada em altos niveis

Encontrar vetores próprios nos dados de dimensão mais alta

Eles definem hyperplanos que dividem os dados, buscando preservar a maior variação entre eles.

Os dados são projetados nesses hiperplanos, de modo que indicam as dimensões inferiores que deseja representar.

Uma implementação popular é a chamada Singular Value Decomposition(SVD)

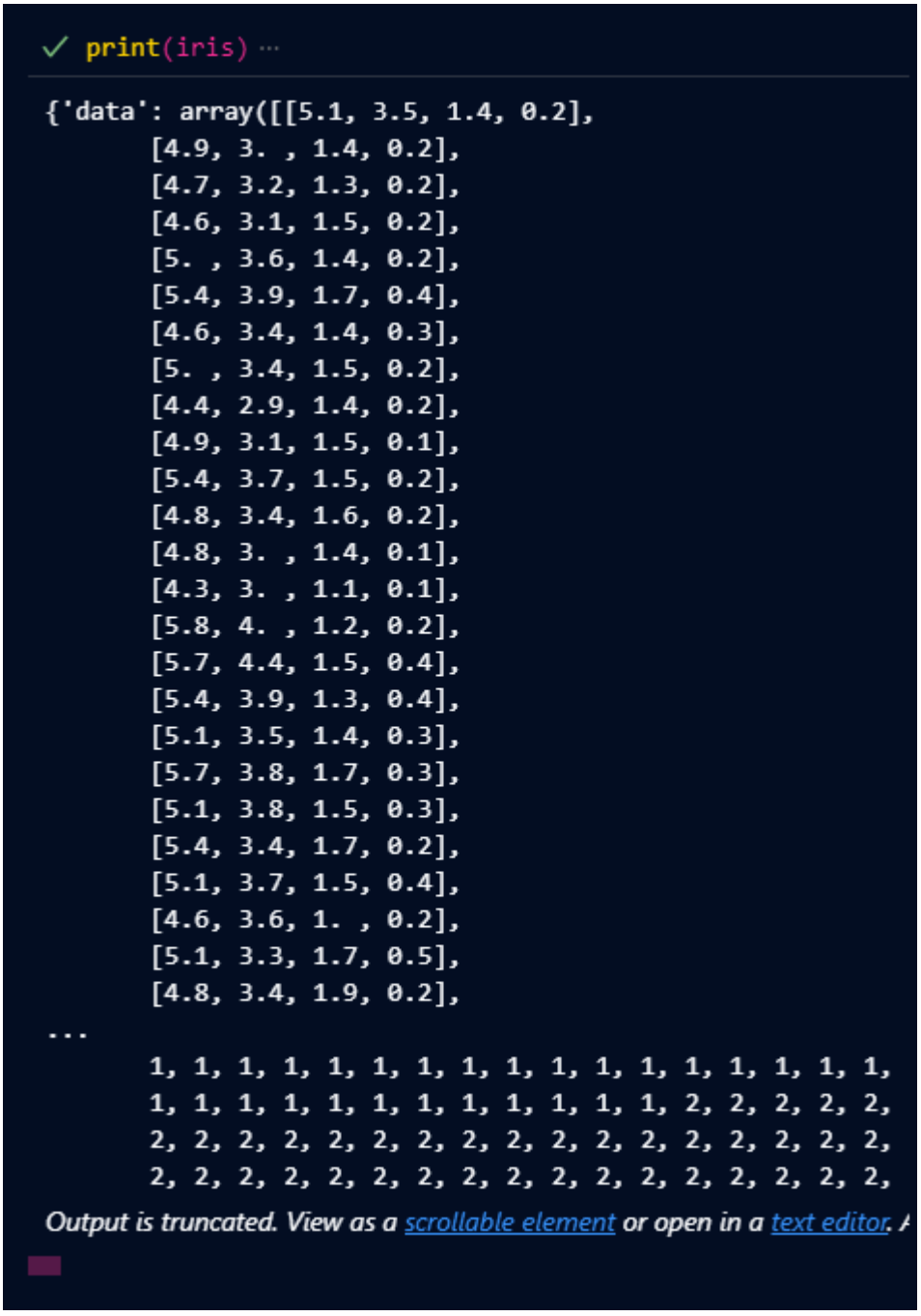
É muito util para coisas como compressão de imagem e reconhecimento facial

▼ Uso prático

Ao validar a implementação dos códigos, foram utilizadas bibliotecas próprias, creio eu que para testes, estas que realizei entrada através do pacote scikit-datasets e scikit-learn.

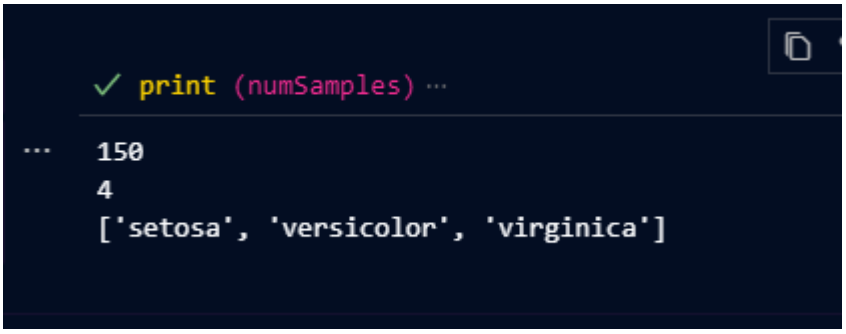
```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import pylab as pl
from itertools import cycle


iris = load_iris()

numSamples, numFeatures = iris.data.shape

print (numSamples)
print (numFeatures)
print (list(iris.target_names))
```

Primeiro passo, foi realizar inicialização das bibliotecas, tendo na sequência a inicialização de uma variavel, que recebe o valor de load_iris(), esta que detém dos dados deste dataset:

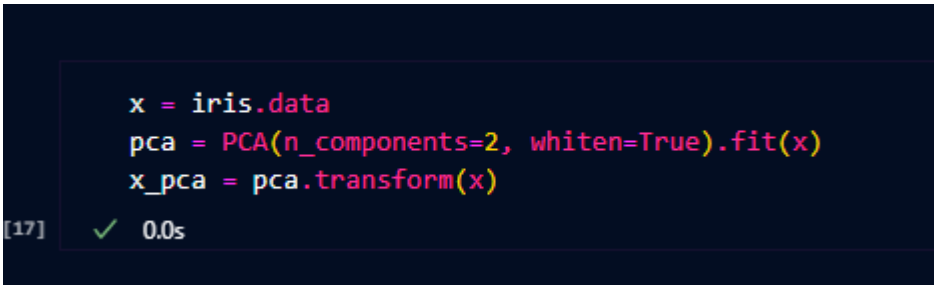


A partir disso, posso encaminhar para outras 2 variaveis estes valores a forma da variavel iris pela utilização do .data.shape, sendo eles numSamples e numFeatures, logo, quantidade de amostras e quantidade de recursos.

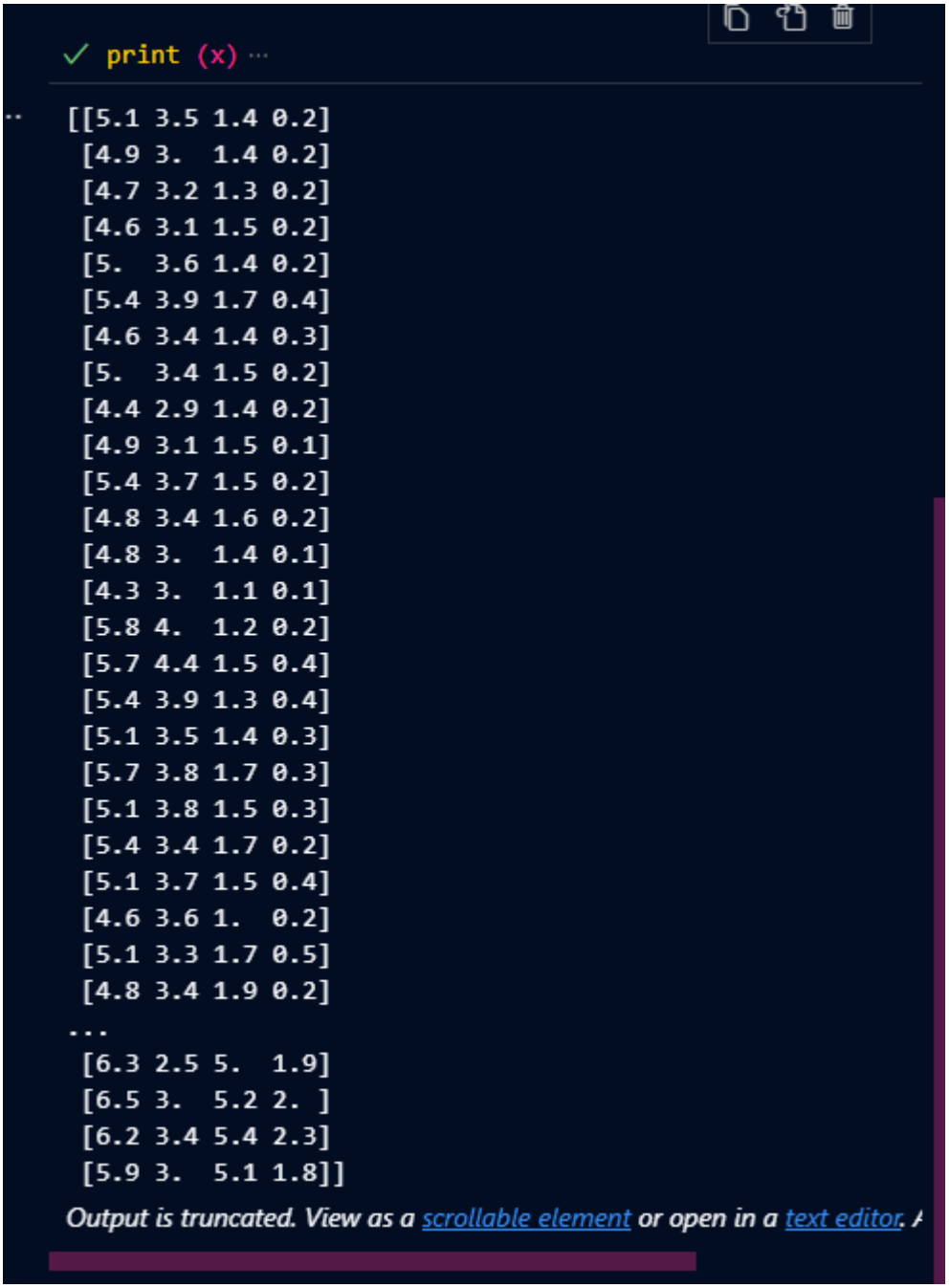
Feito isto, detenho agora, tanto da quantidade de linhas que formam esse dataset que é 150 e a quantidade de elementos em cada array que são 4:



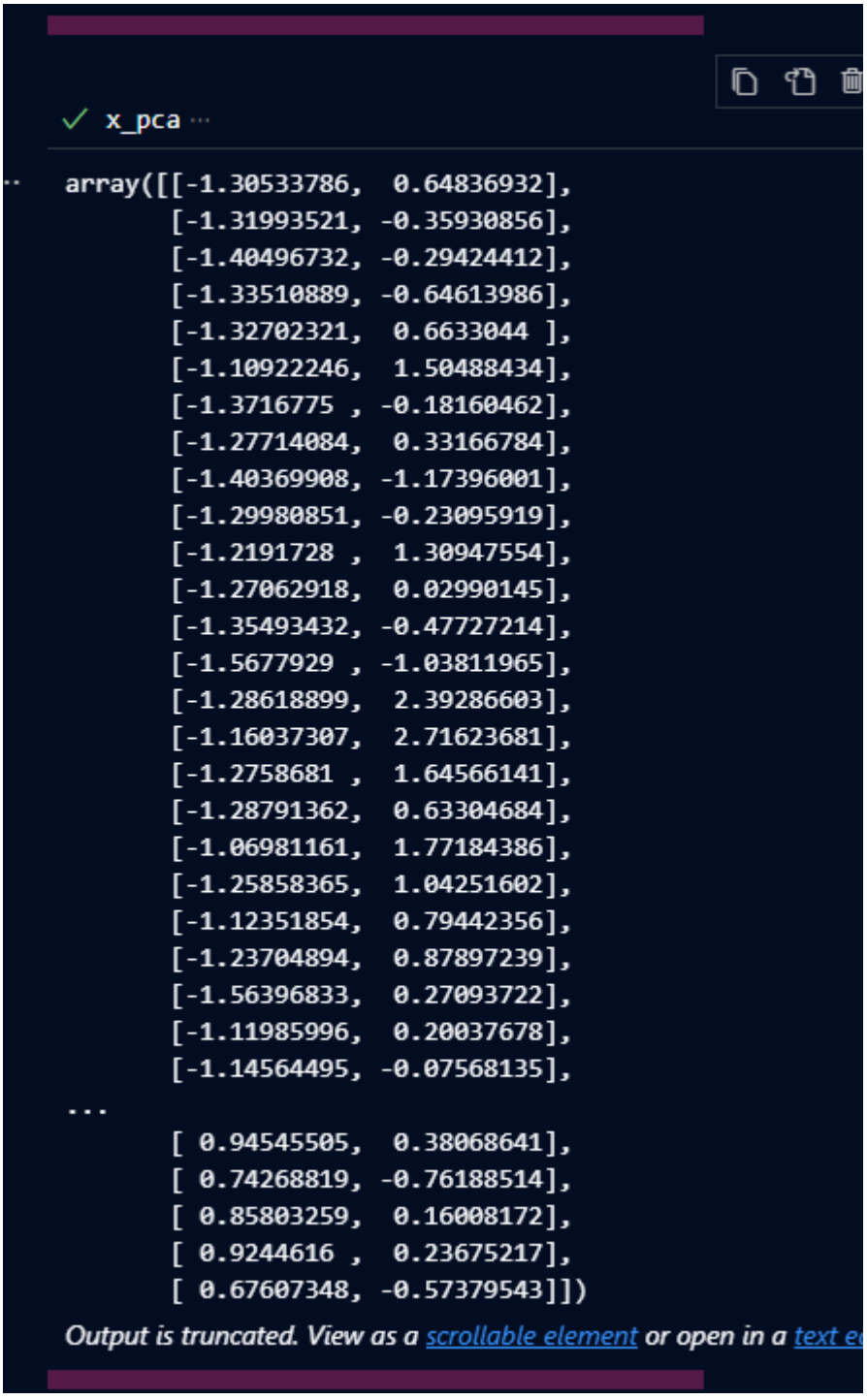
A partir disso, consigo realizar o processo de quebra dimensional, basicamente se detenho de um array de 4, posso dizer que ele tem 4 dimensões, porém, não se torna nada prático trabalhar com 4 dimensões, sendo necessário diminuir para duas:



utilizo então de uma outra variavel que é o x, esta que toma o valor dos dados da iris:



Sendo transformados na sequência para ao invés de 4 componentes, somente 2:



Sendo n_components=2 a definição para que o PCA calcule somente os 2 componentes principais, dos quais o resultado final será uma representação dos dados em um espaço de duas dimensões, do qual cada dimensão é uma combinação linear das características originais destes dados.

Whiten=True é o parâmetro que indique que os dados devem ser pré processados para terem uma variância unitária em todas as

suas dimensões após a projeção.

Estes dados, não são tão simples, o que devemos se ater é que ele transforma 4 ou mais dimensões de dados, na quantidade que definirmos.

Por fim:

```
colors = cycle ('rgb')
target_ids = range(len(iris.target_names))
# pl.figure()
for i, c, label in zip(target_ids, colors, iris.targ
        pl.scatter(x_pca[iris.target == i, 0],
                   x_pca[iris.target == i, 1],
                   c=c, label=label)
pl.legend()
pl.show()
```

Realizamos o processo final que é gerar um scatterplot com esses dados.

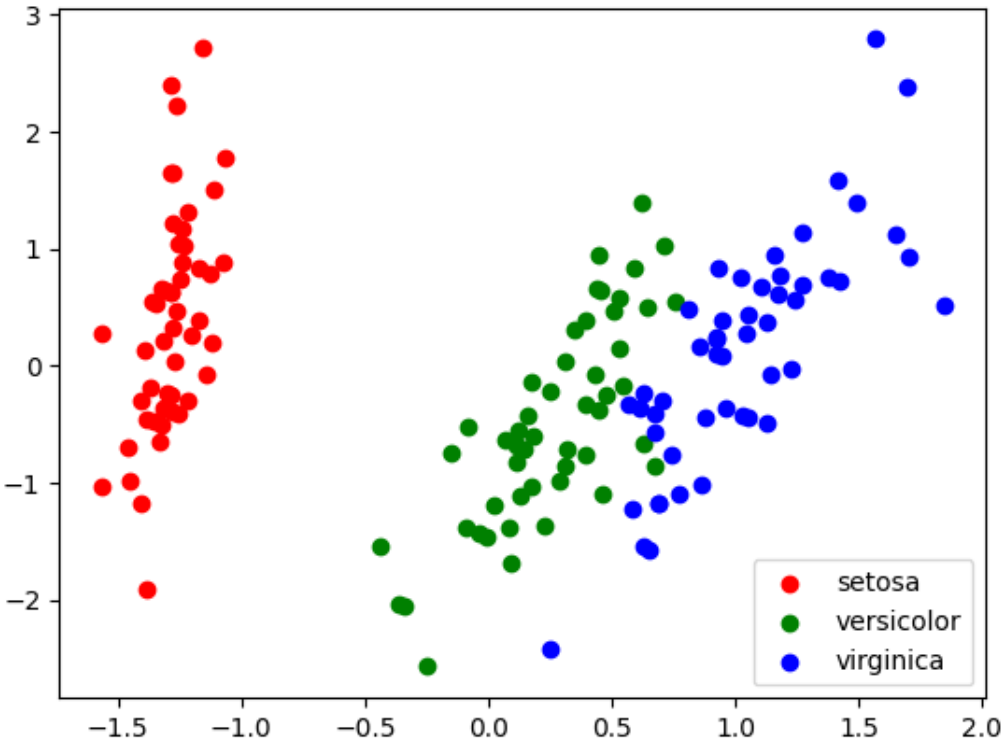
Primeiro definimos as cores que participarão deste scatterplot, na sequência, geramos uma variavel que tem tamanho da variavel iris.target_names.

A partir disso, geramos uma estrutura de repetição com variaveis, i, c e label, no range definido pelo agrupamento de target_ids, cores e a iris.target_names. Logo, ficaria algo como isto:

[(Range(0,3),'Blue','setosa'),(...)]

Dentro deste scatterplot foi definido o seguinte elemento, iris.target == i, 0 o zero em questão é referente a 1 das 2 dimensões, enquanto o 2 a segunda dimensão

A partir disso, bastou indicar o eixo X e Y através da variavel c e label



▼ Data Warehouse(ETL e ELT)

Locais grande e centralizados, dos quais detém de vários servidores de dados de diversos locais.

Estes são usados normalmente para armazenamento da “Big Data”.

Utilizam de SQL o de ferramentas como o Tableau.

Detém de departamentos dedicados a manutenção do data warehouse em questão.

Normalização de dados é complicado - Como fazer todos estes dados se relacionarem um com os outros? Quais os pontos de vista as pessoas necessitam para isto?

A manutenção da alimentação destes dados demanda muito trabalho.

Escalabilidade é complicado.

▼ ETL

Extract, Transform, Load

Dados brutos dos sistemas de operação são extraídos

Esses dados são transformados dentro do esquema necessário do DataWarehouse

Finalmente os dados são carregados para dentro do data Warehouse, já com estrutura necessário.

Porém quando tratamos de bigdata, este processo é um pouco demorado.

▼ ELT

Extract, Load, Transform

Hoje, as gigantescas instancias da Oracle não são os unicos grandes soluções de data warehouse.

Programas como o Hive, permitem que que você hosteie massivos bancos de dados em um cluster do Hadoop

Ou se você pode armazenar em uma grande distribuição NoSQL, utilizando spark ou MapReduce

A escalabilidade do Hadoop, permite você mude o processo de carregamento de dados, permitindo carrega-los e depois transforma-los com o Hadoop.

▼ Reinforcement Learning

▼ Q-Learning

Setamos os estados como s

Setamos as possíveis ações nestes estados com a

O valor de cada estado/ação no Q

Iniciamos com Q = 0

Exploramos o espaço

Nas decisões ruins reduzimos o valor de Q

Nas boas decisões recompensamos acrescentando o Q

▼ Problema da exploração

Como fazer o algoritmo eficientemente explorar todos os possíveis estados?

- Uma simples abordagem: Sempre escolher a ação para dar a um determinado estado, um alto Q. Havendo um empate, escolher randomicamente.

    - Porém isto se torna ineficiente e pode fazer perder muitos caminhos para os estados
- Melhor abordagem: Introduzir a epsilon term
    - Se um numero randômico é menor que a epsilon, ele não segue o highest Q, mas escolhe randomicamente
    - Deste modo, a exploração nunca verdadeiramente para
    - Escolher epsilon pode ser verdadeiramente Rápido
- Markov Decision Process
    - Pela wikipedia: Markov Decision Process(MDPs) prove um framework matemático para a modelagem de Decision Making, dos quais os resultados são randômicos e parcialmente sob o controle do Decision Maker.
    - MDP também é descrito utilizando notação matemática
    - Os estágios são descritos com s e s’
    - A função de transição é descrita por Pa(s,s’)
    - E o seu “Q” é descrito pela função de recompensa Ra(s,s’)

  O MDP é um discreto estocástico processo de controle
- Dynamic Programming
    - Dynamic programing é um método para resolver complexos problema reduzindo eles para subproblemas menores, resolvendo cada um desses subproblemas e armazenando as soluções ideias utilizando um “memória” baseada em estrutura de dados.
    - No momento seguinte caso o mesmo subproblema ocorrer, ele recomputa a solução, e verifica as soluções gravadas na memória, salvando assim tempo rodando o código, e diminuindo o espaço alocado.

▼ Prática

Na prática, trabalhamos com algumas novas bibliotecas, dentre elas a “GYM” que realiza processo de projeção de um “jogo” do qual consta regras e afins.
Jogo escolhido foi o “Taxi-v3” e os valores randomicos, utilizamos a seed(1234).

Inicialmente fizemos seguinte inicialização:

```
random.seed(1234)
metadata = {"render.modes": ["human", "ansi"]}
streets = gym.make("Taxi-v3")
streets.reset()
```

Esta que define os valores randômicos, faz inicialização de dicionário “rander.modes” que é utilizado para a definição no jogo,

Variável streets que recebe a estrutura de treino do “Taxi-v3”

e por fim streets.reset() que zera os parâmetros dela para inicialização.

Feito isto realizamos a inicialização dos estados iniciais:

```
streets.s = initial_state
streets.P[initial_state]


q_table = np.zeros([streets.observation_space.r
```

Estes que permitem que definamos onde irão começar a busca o nosso taxi.

Após isto, definimos uma tabela que seria a matriz utilizada para visualizar o jogo rodando.

```
learning_rate = 0.1
discount_factor = 0.6
exploration = 0.1
```

```
epochs = 10000
streets.render()
```

Feito isto, definimos os parâmetros de aprendizado, os parâmetros de desconto caso erre e exploração.

Após a quantidade de tentativas e por fim ele redeniza a matriz gerada anteriormente.

```
for taxi_run in range(epochs):
    state = streets.reset()
    done = False

    while not done:
        random_value = random.uniform(0, 1)
        if random_value < exploration:
            action = streets.action_space.sampl
        else:
            action = np.argmax(q_table[state])

        next_state, reward, done, info = street
        clear_output(wait=True)
        prev_q = q_table[state, action]
        next_max_q = np.max(q_table[next_state]
        new_q = (1 - learning_rate) * prev_q +
            reward + discount_factor * next_max
        )
        q_table[state, action] = new_q
        print("Trip Number" + str(tripnum))
        print(streets.render(mode="ansi"))
        sleep(0.2)
        state = next_state
```

Feito isto, criamos uma estrutura de repetição da qual realizará as tentativas.
Define state como o reset, done como false e passa a rodar até concluir as 10.000 tentativas.

Entra num while, este while da entrada em um valor de 0 a 1, se esse valor for maior que o de exploração ele define um

sample de ação, caso não ele da entrada na ação com o maior valor do array action.

Ele da entrada em ordem do próximo estado, da recompensa, do done e de uma info, isto através do passo que ele realizou na matriz, basicamente se eu piso em algum local que recompensa é baixa, ele define que o reward é negativo, já se piso num bloco normal, positivo, e caso pegue um passageiro em um dos locais, indica um valor bem acima.

Após isso ele indica um método que aguarda para limpar o output, afim de dar efeito de que o jogo esta rodando de fundo.

Analisa o passo anterior, analisa o próximo, e cria através um cálculo a próxima ação a ser realizada.
Após ele printa o numero da tentativa atual.

Redeniza o passo atual, aguarda 0.2 segundos e inicializa o próximo estagio.

Quando finaliza ele continua tentando até encontrar uma rota sem erros.

▼ Confusion Matrix

Em um vilarejo, havia um menino que tinha a única função de identificar se haviam lobos próximos das galinhas, ou se não haviam.

Se havia um lobo, ele gritasse “Lobo” as galinhas estavam salvas.

Se gritasse “Lobo” e não houvesse lobo, ele seria repreendido.

Se Houvesse Lobo e ele não gritasse as ovelhas seriam comidas.

Mas se não houvesse lobos e ele não gritasse não haveria problema algum.

Se em 10 momentos:

	VDD	FALSO
Lobo veio	TP - 3	FP -1

Lobo não veio	FN - 2	TN - 4

Podemos gerar uma média e dizer que o acerto foi 7/10, sendo assim 70% das respostas corretas.

Porém quando tratamos de um IA tratando disto, consideramos uma outra forma de cálcular. Sendo ela:

$$recall = \frac{TP}{TP + FN}$$

Logo:

3 / 3+2

3 / 5

0,6

Este processo chamamos de **RECALL**(De todos os possiveis positivos, quantos o modelo identificou corretamente?)

Também podemos tratar da ideia de **precisão**, essa que se baseia na seguinte pergunta:

-Qual a proporção de identificações positivas foi realmente correta?

$$precision = \frac{TP}{TP + FP}$$

Logo:

3 / 3 + 1

3 / 4

0,75

Temos também o **f-score**, que é o balanço entre a precisão e o recall, sua fórmula é:

$$2 * \frac{precision * recall}{precision + recall}$$

Sendo assim:

2*(0,75*0,6)/2*(0,75+0,6)

1,5*1,2/1,5+1,2

1,8 / 2,7

0,6666...

Logo, o mais devido nesses casos, de filtragem utilizando IA(Caso programa venha diferenciar objetos, é buscar uma maior classificação entre os objetos.

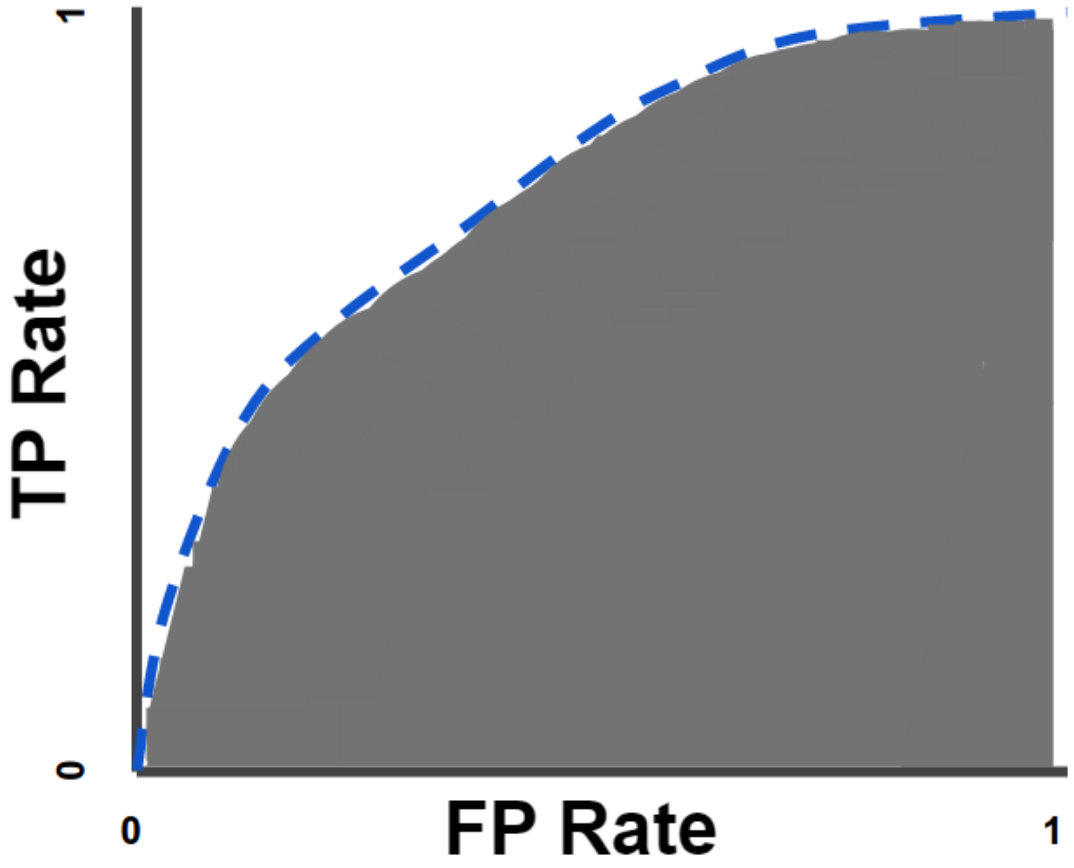
▼ Curva ROC

• Se escolher um ponto positivo e um ponto negativo, qual a probabilidade do meu modelo cassifica-los na ordem correta?
• Intuição: oferece uma medida agregada de desempenho agregado em todos os limiares de classificação possíveis

▼ AUC

Uma curva ROC mostra TPR x FPR em diferentes limiares de classificação. Reduzir o limite de classificação classifica mais itens como positivos, o que aumenta os falsos positivos e verdadeiros positivos. A figura a seguir mostra uma curva típica de ROC.

Logo:



$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

▼ Bias/ Variance Tradeoff

Média de previsões == Média de observações

Bias: O quão longe a média esta dos valores previstos da resposta real?

Variância é quão dispersos estão esses dados da resposta real

Logo, se tivermos um alvo, pontos estão dispersos sem nenhuma proximidade e distantes do centro, ele não detém de uma boa variancia, pois dados estão totalmente distantes, e nem de Bias, pois os dados não se aproximam do centro que é nosso ponto de analise.

Tenho diversos fundos de papel, cada qual com seu valor e seu retorno mensal.

Se a % de retorno que preciso é de 1% mas detenho somente de valores de 0,5 e 1,5% logo, posso dizer que esses 2 fundos tem um alto Bias, porém, se validar que 90% desses fundos paga 0,5 ao mês e somente 10% 1,5, posso considerar que existe uma boa variância, pela conjunção da maior parte desses dados.

Logo, Alto Bias: pontos distantes do centro.

Alta variância: pontos distantes entre si.

Em um gráfico, podemos considerar um baixo Bias quando uma linha cruza os pontos, enquanto um alto Bias, quando ela passa entre 2 pontos. Para isto, damos o nome de OverFitting e UnderFitting

- Error:

  Buscamos se preocupar com o Erro e não totalmente com o BIAS ou Variância. O error encontramos através de:

  $$ERROR = Bias^2 + Variancia$$

  Este é um erro que vocÊ deseja minimizar, não BIAS ou Variância

  É um complexo modelo se você tiver uma alta variância e um baixo BIAS

  É um modelo simples se você tiver uma baixa variância e alto BIAS

  Porém, ambos gerarão o mesmo erro, a complexidade ideal é o meio

Acrescentando o K no KNN diminuímos a variância e acrescentamos o BIA(calculando a média de mais vizinhos).

Uma arvore de decisão é propensa a sobreajuste(Alta variância)

Porém uma floresta randômica, diminui a variância

▼ K-Fold Cross-Validation

Mais uma maneira de estar protegendo contra overfitting é o k-fold cross validation.

Dividir seus dados em K randômicos segmentos atribuídos

Reservar um destes segmentos para testar seus dados

Treinar com o restante combinado k-1 segmentos e garantir sua performasse realizando os testes.

Repetir esses teste com cada segmento

Pegar a média de seus K pontuação ao quadrado

Previne contra overfitting com um único treino/divisão de teste

Scikit-learn torna esse processo muito simples. Na prática você precisa experimentar difentes variações no seu modelo e na sua medida de acuracia média utilizando K-FOLD Cross validation, até você encontrar um bom spot

Primeiro inicializamos as bibliotecas

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import datasets
from sklearn import svm
```

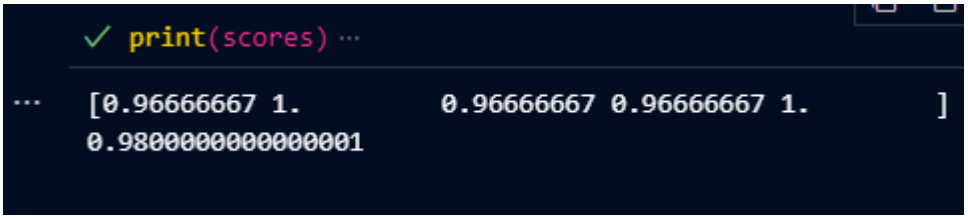
Após construímos o nosso dataset:

```
iris = datasets.load_iris()

# Definiu para variavel train e test 40% dos dados, log
x_train, x_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.4, random_state
)
```

Este que utiliza somente 40% dos dados e divide entre teste x,y e treino x,y.

```
# Construiu um modelo SVC para predição de classificaçã
clf = svm.SVC(kernel="linear", C=1).fit(x_train, y_trai

# Agora mediu a performace dos dados testados
clf.score(x_test, y_test)

# Dar para a função cross_val_score, o data set complet
scores = cross_val_score(clf, iris.data, iris.target, c
```

Primeiro construimos o primeiro treino usando o modelo SVC com kernel linear, a partir dele retornamos seguintes valores:

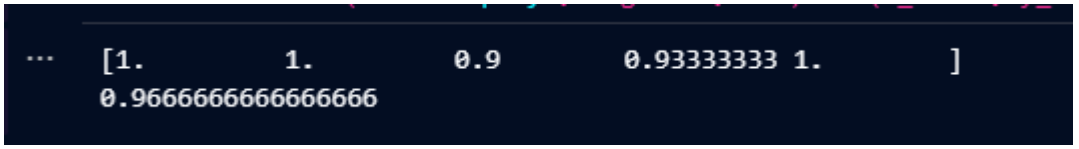


A partir deles, conseguimos identificar a acurácia de cada dobra e a média de acurácia entre as 5 dobras.

Podemos utilizar também um outro kernel para testes, o também chamado de polinomial. Processo é o mesmo que acima:

```
clf = svm.SVC(kernel="poly", degree=4, C=1).fit(x_train
scores = cross_val_score(clf, iris.data, iris.target, c
```

Através dele, temos valores diferentes:



Alcançando assim um valor mais próximo de 0,966

▼ Limpeza de dados

Uma maneira de encontrar dados maliciosos, ou encontrar o furo destes dados, é analisar os opostos neles, encontrar os extremos entre o minimo e maximo.

Como exemplo temos a ideia de um webscraping, o mesmo realiza um acesso ao site a cada tick, a partir disso identificamos que o mesmo não é humano, devemos colocar ele nos nossos dados de analise em um produto ou não?

> Dados irrelevantes, retirar dados desnecessários e restringir os presentes a somente dados que serão uteis

Dados inconsistentes: De quantas formas podemos falar do mesmo endereço, São paulo, SP, São P., S.Paulo, S.P.

Olhe seus dados, examine eles, questione sob seus dados, não somente quando eles fogem das suas expectativas.

▼ Normalizando dados

Anos - 1 - 100(Exemplo)

Recebimentos 0-xbilhões(Exemplo)

Normalizar dados é definir uma metrica para os valores ou dados que irão compor um dataframe, base de dados...

O scikit-learn PCA tem uma implementação 'Whiten' que ela normaliza os dados para nós.

Scikit-lean também tem um módulo de pré-processamento que detem de funções de normalização e escala

Seus dados tem Yes ou No, O que você precisa é somente convertelos para 1 e 0

▼ Detectando as discrepâncias

Se temos um ambiente de analise de filmes e damos a oportunidade das pessoas votarem a nota que desejarem, haverão filmes com valor 1000, outros com 10, outros com -50 e por ai vai.

Outro caso é da analise de filmes, do qual podemos identificar o uso de um bot pela quantidade de tentativas de acessos, sendo necessário retira-los para uma plena analise.

Mas se temos quantidade total de pessoas de um pais, não vamos retirar os milionários por serem 1% da população.

Encontre pontos com muitos desvios no treinamento de dados.

Para encontrar esses múltiplos, basta utilizar senso comum.

Por muitos momentos, não encontramos a variação entre o maior valor e o valor padrão, simplesmente pela quantidade que este valor maior se repete ser muito inferior aos demais. Para isto, é recomendado analisar não somente a média desses valores(dos quais conseguimos identificar variação) mas proporcionar uma visualização da qual esses dados são apresentados

▼ Feature engineering & curse of dimensionality

Curse of dimensionality

- Leads to sparse data(Algo como, caminho para os dados esparsos)
  - Você detém de diversas informações que não se complementam entre si. Idade, altura, peso, local que estudaram, valor de recebimento,...
  - Isto pode ser simplesmente resolvido, havendo uma analise do que de fato é seu problema e no que de fato precisa ser tratado.
  - Se você vai tratar de finanças, o que o peso da pessoa irá representar?
  - Mas se for tratar da relação de pessoas magras/gordas com valor que recebe, por qual motivo utilizar local que estudaram?

PCA e K-Means podem ser boas ferramentas para reduzir este processo.

▼ Missing Data

▼ Mean Replacement

Basicamente transformar o valor NaN em um valor da média dos demais valores

Funciona somente via linhas, perdendo correlação com demais features

Não pode ser utilizado em features categóricas.

Não é muito exato

▼ Dropping

Se consta NaN, você retira da sua visualização.

Se você não tem muito tempo para analisar, e essas linhas não farão você perder o Bias do seus dados, este é um processo razoável para estar realizando.

A partir dela, você nunca encontrará a resposta correta, somente uma aproximação do que poderia ter sido.

Todas demais métodos são melhores(Perder dados não é um bom caminho, mas pode ser uma alternativa)

- ▼ Machine Learning

  KNN: Encontrando K “Nearest” (Mais similar) linhas e seus valores médios

  Assume dados numéricos, não categóricos

  É o caminho caso tenha em mãos dados categóricos(Hamming distance), mas dados categóricos, é melhor utilizar Deep Learning

  - ▼ Deep Learning

    Construir um modelo de machine learning para dar input nos dados em seu modelo de aprendizado

    Trabalha bem com dados categóricos, porém é complicado

  - ▼ Regressão

    Encontrar relações lineares ou não linear acerca dos dados NaN e de outras features.

    A técnica mais avançada: MICE(Multiple Imputation by Chained Equations)

- ▼ Pegar mais dados

  É o melhor caminho e fará você encontrar a resposta real para o problema em questão.

  Porém, pode se tornar complicado coletar esses dados.

- ▼ Handling Unbalanced Data (Oversampling, undersampling and SMOTE)
  - ▼ Grande discrepância entre valores positivos e negativos

    Detecção de fraude

    Não se deixe levar pela terminologia, positivo não significa bom.

    Se seu modelo de machine learning foi feito para encontrar fraude, a fraude será o valor positivo

    Maior problema é a rede neural

    - ▼ Oversampling

Duplicar valores vindos de uma classe menor

Pode ser randomico

Basicamente pegar valores aleatórios e duplicar eles afim de que o treino seja mais eficaz

▼ Undersampling

Criar mais casos positivos, removendo os negativos.

Possibilita analise dos dados com menos hardware envolvido

Retirar dados, normalmente não é uma boa resposta.

E se estivermos tratando de dados na escala de “big data”, poderemos ter problema na analise.

▼ SMOTE(Synthetic Minority Over-sampling Techinique)

Gera artificialmente novos “samples” da minoria das classes utilizando KNN

Rode o Knn para cada sample da minoria das classes

Crie novos samples através resultado do KNN(Media dos vizinhos)

Ambos geram novos dados em sua maioria undersamples.

Geralmente são melhores do que apenas o oversampling

▼ Ajusting Thresholds

Se trabalhamos com valore de 1000 até 10.000, e um de nossos dados aponta que em 3 meses empresa x teve valores ok entre 4.000 porém no quarto mês apresentou 10 reais, podemos através de um liminar já descobrir que possivelmente ocorreu algum problema ou fraude.

Isto permite que diminuimos a quantidade de falso positivos, pois, mesmo sabendo que valor se encaixa dentro de um valor positivo, ele ainda é muito inferior aos demais.

Porém isso coloca em cheque ideia de que ou pode ter ocorrido algum problema com captura desses dados(Missing Data), que empresa em questão esta falindo ou que teve uma queda massiva nos seus valores, fazendo assim que diminuam as quantidades de falso positivos sim, mas que aumente a de falso negativos. Afinal, precisamos compreender o que de fato ocorreu.

▼ Bining, transforming, encoding, scaling, shuffling

▼ Binning

Separar dados em intervalos através de alguma característica deles.

▼ Transforming

Feito cálculos a partir dos dados afim de encontrar insights que possam ser utilizados.

Realizado a partir de cálculo de logaritimo afim de transformar eles em outros dados.

Aplicado nas recomendações do youtube

Um valor numérico pode ser representado por x^2 ou Raiz(x)

Também comporta aprendizado de super e funções sub lineares

▼ Encoding

Transformar dados em outras representações a partir de um modelo

One-Hot encoding

Criar Buckets para cada categoria

Os buckets de cada categoria tomam valor de 1 e todos os outros 0

Bem comum em deep learning onde as categoria de dados são representados com um output de “Neuronio”

▼ Scaling/Normalization

Muitos modelos preferem utilizar dados de uma distribuição linear proximos a 0 (Most neural nets)

Maior parte dos modelos necessita de dados específicos para encontrar a escala de valores comparáveis

Otherwise features com magnitude, onde seu peso é maior do que deveria ter

Modelo de idade: Valores que estão sendo analisados são supostamente superiores ao que deveria ter.

Scikit-Learn contem um módulo para auxiliar(MinMaxScaler)

Lembrar de aumentar o resultado dos seus dados novamente(Utilizar de um outro dataframe para analise)

▼ Shuffling

Muitos algoritmos se beneficiam pelo embaralhamento dos dados que serão utilizados para treino

Caso contrário ele aprenderia com sinais residuais, utilizando os dados em ordem para isto, iniciando de um valor 0 - 100, como identificar um problema quanto idade, ou insights sobre idade, com n+1?