parte 1. K-Nearest Neighbor (KNN)

KNN é um método de classificação com base em dados vizinhos, ou seja, é extremamente eficiente para usar em Datasets próximos a fase de treinamento. Esse método classifica dados baseando-se na distância até pontos “conhecidos”.

Primeiramente separamos o código em colunas que vamos usar para analisar:

```
import pandas as pd

r_cols = ['user_id', 'movie_id', 'rating']
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=r_cols, usecols=range(3))
ratings.head()
```

	user_id	movie_id	rating
0	0	50	5
1	0	172	5
2	0	133	1
3	196	242	3
4	186	302	3

após isso fazemos o agrupamento das informações que gostaríamos de analisar, sendo assim, usamos o método “.agg” e passamos os parâmetros “size”,”mean” na criação de uma lista baseada na variável “ratings” criada anteriormente. Assim, a nova lista vai mostrar a quantidade de pessoas que avaliaram o filme e a média de avaliações.

```
import numpy as np

movieProperties = ratings.groupby('movie_id').agg({'rating': [np.size, np.mean]})
movieProperties.head()
```

	rating	
	size	mean
movie_id		
1	452	3.878319
2	131	3.206107
3	90	3.033333
4	209	3.550239
5	86	3.302326

Entretanto, só esse número de avaliações sem parâmetro é inútil quando levado em consideração as distâncias entre os filmes, dessa forma criamos um novo dataframe com a normalização do cálculo feito anteriormente. A classificação é feita da seguinte forma: quanto mais próximo de 1, maior sua popularidade, quanto mais próximo de zero, menor sua popularidade.

```
movieNumRatings = pd.DataFrame(movieProperties['rating']['size'])
movieNormalizedNumRatings = movieNumRatings.apply(lambda x: (x - np.min(x)) / (np.max(x) - np.min(x)))
movieNormalizedNumRatings.head()
```

movie_id	size
1	0.773585
2	0.222985
3	0.152659
4	0.356775
5	0.145798

A partir desse ponto precisamos extrair as informações recolhidas e separar elas de acordo com gênero, avaliação, avaliação média, ID, nome e etc… Isso será feito da seguinte forma, existem 19 campos disponíveis, cada um pertencendo a um gênero específico e variando num valor de 0 a 1, seguindo a mesma lógica anterior de que quanto mais próximo de 1 mais associado aquele gênero em específico.

Os filmes serão colocados em um dicionário utilizando a linguagem de programação Python como de costume. Criaremos o dicionário movieDict e criaremos as separações em pipe para melhor organizar as informações que iremos analisar.

```
movieDict = {}
with open(r'ml-100k/u.item', encoding="ISO-8859-1") as f:
    temp = ''
    for line in f:
        #line.decode("ISO-8859-1")
        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)), movieNormalizedNumRatings.loc[movieID].get('size'), movieProperties.loc[movieID].rating.get('
```

For example, here's the record we end up with for movie ID 1, "Toy Story":

```
print(movieDict[1])
```

```
('Toy Story (1995)', array([0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 0.7735849056603774, 3.878318584070796)
```

A partir desse ponto que iremos de fato definir a distância entre dois filmes dentro da analise para saber a similaridade entre seus gêneros e suas popularidades. Novamente usaremos como parâmetros 0 e 1.

```
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])
```

```
0.8004574042309892
```

Podemos perceber que obtivemos como resultado um número muito mais próximo de 1 do que de 0, isso significa que obtivemos um resultado positivo, os dois filmes estão próximos.

Após isso vamos comparar a distancia desse filme que usamos no exemplo (Toy Story) e todos os outros filmes na base de dados que estamos analisando. A partir disso mostramos os K elementos vizinhos com a menor distancia do nosso elemento analisado. (página seguinte)

```
import operator

def getNeighbors(movieID, K):
    distances = []
    for movie in movieDict:
        if (movie != movieID):
            dist = ComputeDistance(movieDict[movieID], movieDict[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[neighbor][3]))

avgRating /= K
```

```
Liar Liar (1997) 3.156701030927835
Aladdin (1992) 3.8127853881278537
Willy Wonka and the Chocolate Factory (1971) 3.6319018404907975
Monty Python and the Holy Grail (1974) 4.0664556962025316
Full Monty, The (1997) 3.926984126984127
George of the Jungle (1997) 2.685185185185185
Beavis and Butt-head Do America (1996) 2.7884615384615383
Birdcage, The (1996) 3.4436860068259385
Home Alone (1990) 3.0875912408759123
Aladdin and the King of Thieves (1996) 2.8461538461538463
```

While we were at it, we computed the average rating of the 10 nearest neighbors to Toy Story:

```
avgRating
```

```
3.3445905900235564
```

O número alcançado pelo nosso código é de 3.34 aproximadamente, com isso comparamos com a classificação verdadeira de Toy story.

Parte 2. Dimensionality Reduction & PCA (Principal Component Analysis)

Quando temos um grande número de dimensões, encontramos algumas dificuldades, por exemplo, durante a recomendação de alguns filmes os vetores de classificação de cada um pode representar uma dimensão, sendo assim, cada filme tem sua própria dimensão. Reduzir a dimensionalidade é essencial para tentar distinguir os dados de dimensões maiores para dados de dimensões menores, além disso, deve-se manter a variedade disponível de dados, preservando a qualidade do dataset.

Já uma outra forma de diminuir a quantidade de dimensões é o PCA (Principal Component Analysis), que envolve matemática em alto nível. Esse método define os hiperplanos que separam os dados enquanto mantém a variedade de informações de forma proporcional. Uma implementação popular do método é chamado de SVD (Singular Value Decomposition). Método bastante eficiente para compreensão de imagem e reconhecimento facial.

Um exemplo prático desta técnica é a transformação de uma imagem 4D em uma visualização 2D preservando a variedade de dados presentes.

```
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))
```

```
150
4
['setosa', 'versicolor', 'virginica']
```

Nosso código retorna o tamanho do dataset com a linha “print(numSamples)” assim como a quantidade de dimensões da imagem e lista as 3 espécies de íris presentes na imagem.

```
X = iris.data
pca = PCA(n_components=2, whiten=True).fit(X)
X_pca = pca.transform(X)
```

O trecho de código tem como função armazenar os dados do dataset na variável X e em seguida diminuir a quantidade de dimensões, os componentes PCA são autovetores.

```
print(pca.explained_variance_ratio_)
print(sum(pca.explained_variance_ratio_))
```

A segunda linha mostra a quantidade total em todas as N dimensões escolhidas para o dataset X. Nesse caso obtivemos como resultado um valor próximo de 0.977 o que representa 97,7% de porcentagem preservada, representando assim um valor muito relevante na análise.

Parte 3. Data Warehousing

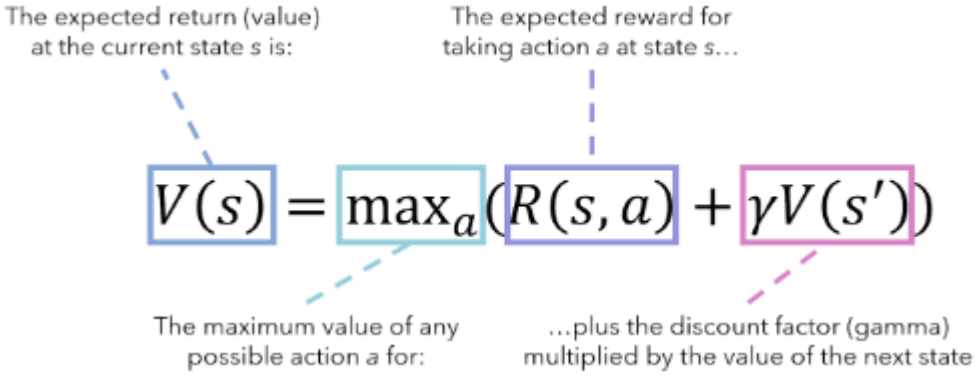
O vídeo introduz dando conceitos sobre o que é uma Data Warehousing, e a define como uma grande database centralizada com muitas informações e fontes e seu maior uso é por parte de grandes indústrias e empresas para análise de negócios e corporações. Geralmente sua consulta é feita por SQL ou ferramentas relacionadas. A manutenção de uma data warehousing é sempre destinada a equipes internas e que conhecem as databases, sua normalização e escalabilidade são de alta complexidade.

Existem dois tipos de dados dentro de uma data warehousing, são eles:

ETL - Extract, Load, Transform: É o mais comum de ser usado. Este método usa dados brutos extraídos dos SOs. Transforma a coleção de dados em schemas dificultando muitas vezes o processamento de BigDatas
ELT - Extract, Transform, Load: A tecnologia Hive permite que bancos de dados Oracle não sejam mais a única opção, permitindo a hospedagem de dados em um cluster de Hadoop, bem como em Warehouses NoSQL. Algumas consultas podem ser feitas usando Spark, MapReduce e outras tecnologias construídas em cima do Hadoop. A escalabilidade do Hadoop possibilita a otimização do processo de carregamento e transformação dos dados dentro do próprio repositório, melhorando assim o processamento e a escalabilidade dos dados da Warehouse.

Parte 4. Reinforcement Learning

O narrador do vídeo usa como exemplo o famoso jogo, pac man para mostrar o funcionamento de aprendizagem de um um "agente" explorando um espaço e aprendendo o que se deve fazer a partir de respostas de valores positivos ou negativos e a forma como isso impacta nos futuros passos e ações do agente, nesse caso, do pac-man.
Uma das formas de tornar eficiente o aprendizado do Pac-man é a implementação do Q-Learning. O Q-Learning é um algoritmo de aprendizado por reforço que opera em um ambiente composto por um conjunto de estados (condições do ambiente), um conjunto de ações possíveis e um valor Q atribuído a cada par estado-ação. O objetivo é explorar o espaço de estados e ações para aprender a melhor política de ação. Durante a exploração, são aplicadas punições quando ações levam a resultados negativos e recompensas quando levam a resultados positivos. Para explorar eficientemente, pode-se adotar estratégias como sempre escolher a ação com o maior valor Q ou introduzir um termo epsilon, que determina a probabilidade de escolher uma ação aleatória em vez da ação com o maior valor Q. Essa exploração mais ampla aumenta as chances de encontrar melhores políticas de ação.
Após os conceitos de Q-learning o narrador aborda os processos de decisão de Markov (MDPs) que é basicamente um framework matemático para tomar decisões em resultados que são, em sua maioria, aleatórios,mas também são controlados por um indivíduo.



Parte 5. Bias/Variance

Essa parte do curso diz muito sobre as dificuldades de lidar com dados do mundo real (Como é de se esperar pelo nome do módulo). Nessa seção irei falar sobre Viés e Variância durante o primeiro processo de treinamento de uma rede neural de forma supervisionada, o que se torna simples de entender quando vemos que os dois se referem ao quão porco estão seus resultados.

Bias: Distância entre os valores previstos e os valores reais em determinada direção.
Variância: A variância mede a dispersão entre as previsões feitas e os valores reais.

O que o narrador diz é que se suas previsões estão espalhadas então a variação é alta (Isso é ruim) e se elas estão muito concentradas, então a variação é pequena. Se voce pegar a média de todas as suas previsões e elas estiverem próximas, ou se seues erros estão enviesados de forma consistente em uma direção ou outra

Nessa ilustração vemos uma alta variância e baixa inclinação, isso é percebido pois os pontos não estão muito longes do desejado (O meio) entretanto, estão dispersos.

Baixa variância, Alta inclinação

Alta variância, baixa inclinação (Pior cenário possível)

Baixa variância, baixa inclinação (Melhor cenário possível)

Parte 6. Tradeoff - Overfitting x Underfitting

Underfitting - Baixa variância, sua representação é uma linha reta no gráfico, seu viés é alto pois os pontos estão longe da linha.
Overfitting - Alta variância e uma representação não linear no gráfico, seu viés é baixo pois os pontos muitas vezes estão suficientemente próximos da linha.

Taxa de Erro - Intuitivamente vemos que a variância e o viés contribuem para o aumento do erro por isso nosso objetivo é diminuir o máximo possível dos dois.

$$error = bias^2 + variance$$

A seção deste ponto até o fim apresenta várias formas de evitar o Overfitting como a técnica de K-Fold Cross Validation que consiste na divisão de dados em segmentos K e para cada segmento reservamos um para dados de teste e outros como dados de treinamento, além de treinar nos K-1 segmentos que faltarem, e medir a performance em relação ao dataset de teste.

parte 7. Data Cleaning and Normalization

- Em primeiro lugar, é preciso entender que trabalhar com dados brutos e sem filtros podem trazer diversos problemas durante nosso processo de desenvolvimento. Por isso é necessário que tenha uma parte do processo para realizar a limpeza dos dados para que possamos trabalhar de forma facilitada.

- **Outliers**: dados não relacionados; exagerados
- **Dados Faltantes**:de natureza diferente dos demais
- **Dados Maliciosos**: pessoas tentando enganar o sistema
- **Dados Errôneos**: erros comuns de software (acontece)
- **Dados Irrelevantes**: São dados que simplesmente são irrelevantes para o problema alvo
- **Dados Inconsistentes**: ex.: endereço, pode ser escrito de formas diferentes.
- **Formatação**: números de telefone com () em volta do DDD, CPFs com traço entre outros…

- Em segundo lugar, a normalização dos dados, necessários para que os dados dentro do modelo possam ser comparados e por isso devem estar organizados e semelhantes. A não normalização dos dados pode levar ao enviesamento e aumentando a taxa de erro.

parte 8. Normalizing numerical data

Como dito anteriormente é primordial que os dados numéricos presentes sejam comparáveis. Não faz o menor sentido haver dados de data no formato dd/mm/ano e mm/dd/ano no mesmo dataset, os dados não serão comparados igualmente e isso aumentará a taxa de erro do modelo

parte 9. Detecting Outliers

Uma boa alternativa é simplesmente removê-los dos dados de treinamento, caso isso não seja possível, deve-se classificar esses dados exagerados usando desvio padrão. Um bom exemplo disso é remover o salário de bilionários quando se deseja fazer a média de salário de cidadãos comuns.

parte 10. Feature Engineering

Basicamente são as aplicações dos conhecimentos sobre os dados e os modelos em desenvolvimento com o objetivo de criar features mais otimizados de treinamento, pensando também no excesso de dimensões do modelo. Porém, deve-se usar algumas técnicas de redução de dimensões, como vistos anteriormente, com isso, o número de features vai ser menor com a mesma variedade de informações.

parte 11. Imputation Techniques for Missing Data

- Substituir com a média
    - Uma técnica rápida e fácil, porém é impreciso e só funciona no nível de coluna
- Dropping
    - Utilizável em **POUQUÍSSIMOS** casos, apenas na falta de um número quase desprezível de informações
- K-Nearst Neighbors
    - Faz uma busca das K linhas mais parecidas e realiza a média de seus valores
- Deep Learning
    - Utilização de um modelo de ML para substituição dos dados
- Regressão
    - Encontra relações lineares entre 2 features
- Busca de dados
    - O próprio nome já diz

Todas essas técnicas possuem vantagens e desvantagens que devem ser analisadas conforme a necessidade do dataset

parte 12. Handling Unbalanced Data: Oversampling, Undersampling, and SMOTE

É interessante, antes de saber como lidar, saber o que são dados desbalanceados. De forma resumida, são dados com um grande espaço entre o que são casos positivos e o que são casos negativos. O vídeo deixa claro também que um resultado positivo nem sempre é algo bom, só quer dizer que o que está sendo testado aconteceu.

- Oversampling
    - Duplicar casos de minoria
- Undersampling
    - Remover as amostras negativas ao invés de criar mais positivas
- SMOTE (Synthetic Minority Over-sampling Technique)
    - Gerar mais amostras positivas usando KNN gerar resultados e executar undersampling na classe dominante
- Ajustando Limites

Na última parte é falado sobre técnicas adicionais de feature engineering como Binning (Que consiste em colocar observações juntas baseado em ranges de valores); Trasforming (Que é a aplicação de uma função a uma feature para tornar mais apto ao treinamento do modelo); Encoding (Que é a transformação de dados em outra representação requerida pelo modelo); Normalização e Embaralhemento.