

Processamento de dados em Big Data

Parte 4 - Machine learning

Luiz Henrique Zambom Santana, D.Sc.

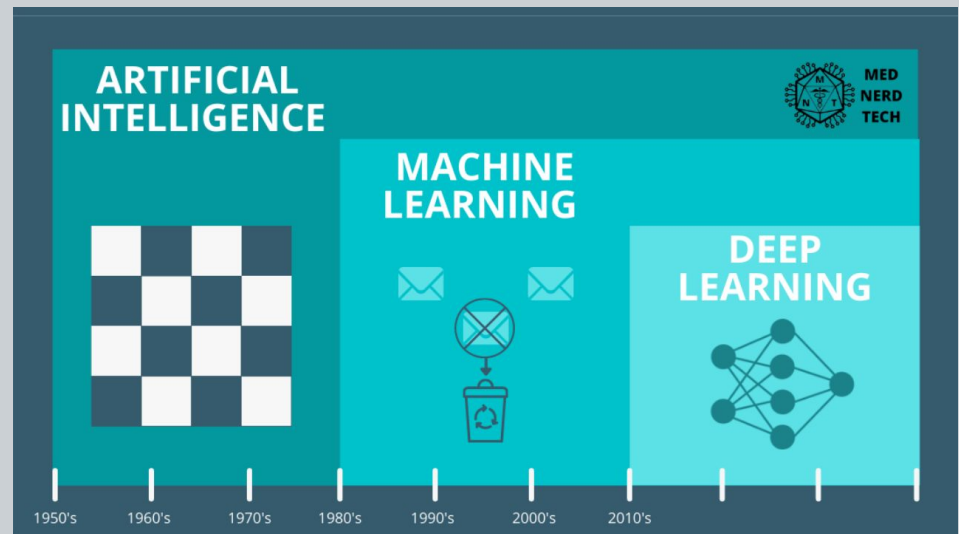
INE | CTC



UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Agenda

- Algoritmos tradicionais
 - Spark ML
 - Tensor Flow
- LLM
 - Introdução
 - LangChain



Spark ML

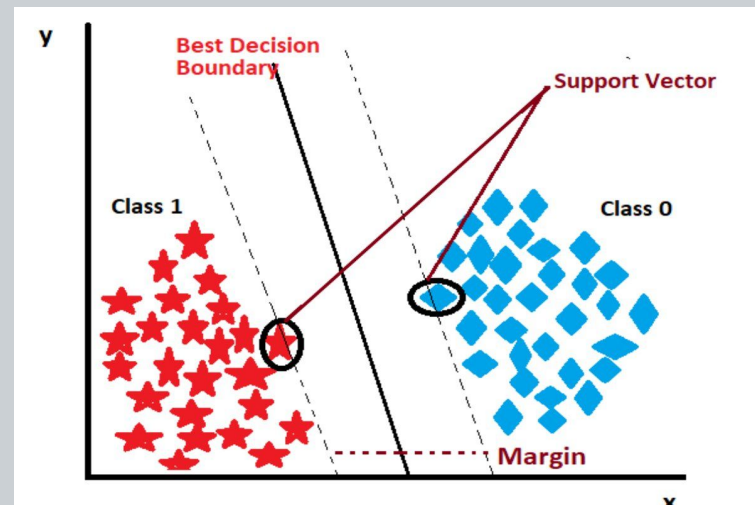


Algoritmos tradicionais

- Estatística básica
- Supervisionado e não supervisionado
 - Classificação
 - Naive Bayes
 - SVM
 - Recomendação
 - Filtragem colaborativa
 - Clustering
 - K-means

Naive Bayes e SVM

- O Spark ML oferece os algoritmos de classificação Naive Bayes e SVM
- Ambos são supervisionados
- Primeiramente são apresentados conjuntos de dados rotulados (labeled)
- Posteriormente, o modelo irá prever como os dados apresentados se classificam



A photograph of a whiteboard with the Naive Bayes formula written in blue marker:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Naive Bayes e SVM

```
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)























# Build the model
model = SVMWithSGD.train(parsedData, iterations=100)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda lp: lp[0] != lp[1]).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))

# Save and load model
model.save(sc, "target/tmp/pythonSVMWithSGDModel")
sameModel = SVMModel.load(sc, "target/tmp/pythonSVMWithSGDModel")
```

Filtragem colaborativa

- A ideia é usar a semelhança em escolhas passadas, para realizar sugestões
- Não supervisionado
- Pode ser baseado no item ou no usuário
- Tem o problema do *cold start*

					
	Book 1	Book 2	Book 3	Book 4	Book 5
 User A					
 User B					
 User C					
 User D					

Filtragem colaborativa

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

lines = spark.read.text("data/mllib/als/sample_movielens_ratings.txt").rdd
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=float(p[2]), timestamp=int(p[3])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating",
          coldStartStrategy="drop")
model = als.fit(training)

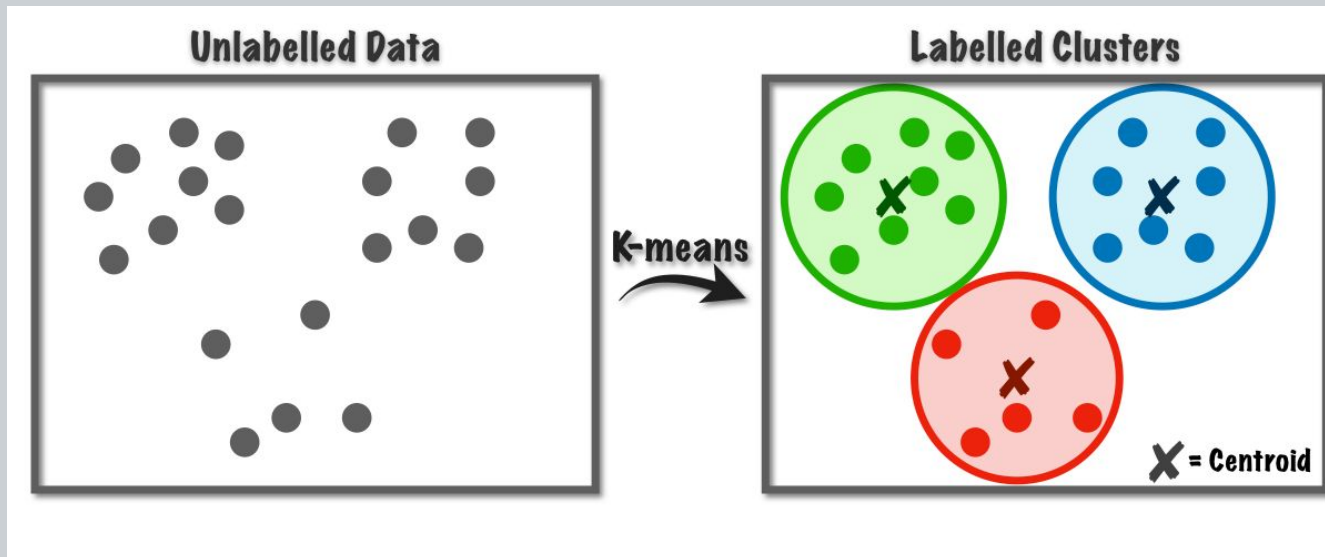
# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                               predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

# Generate top 10 movie recommendations for each user
userRecs = model.recommendForAllUsers(10)
# Generate top 10 user recommendations for each movie
movieRecs = model.recommendForAllItems(10)

# Generate top 10 movie recommendations for a specified set of users
users = ratings.select(als.getUserCol()).distinct().limit(3)
userSubsetRecs = model.recommendForUserSubset(users, 10)
# Generate top 10 user recommendations for a specified set of movies
movies = ratings.select(als.getItemCol()).distinct().limit(3)
movieSubSetRecs = model.recommendForItemSubset(movies, 10)
```


K-Means

- Um algoritmo que serve para agrupar items por semelhança
- Não supervisionado



K-Means

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

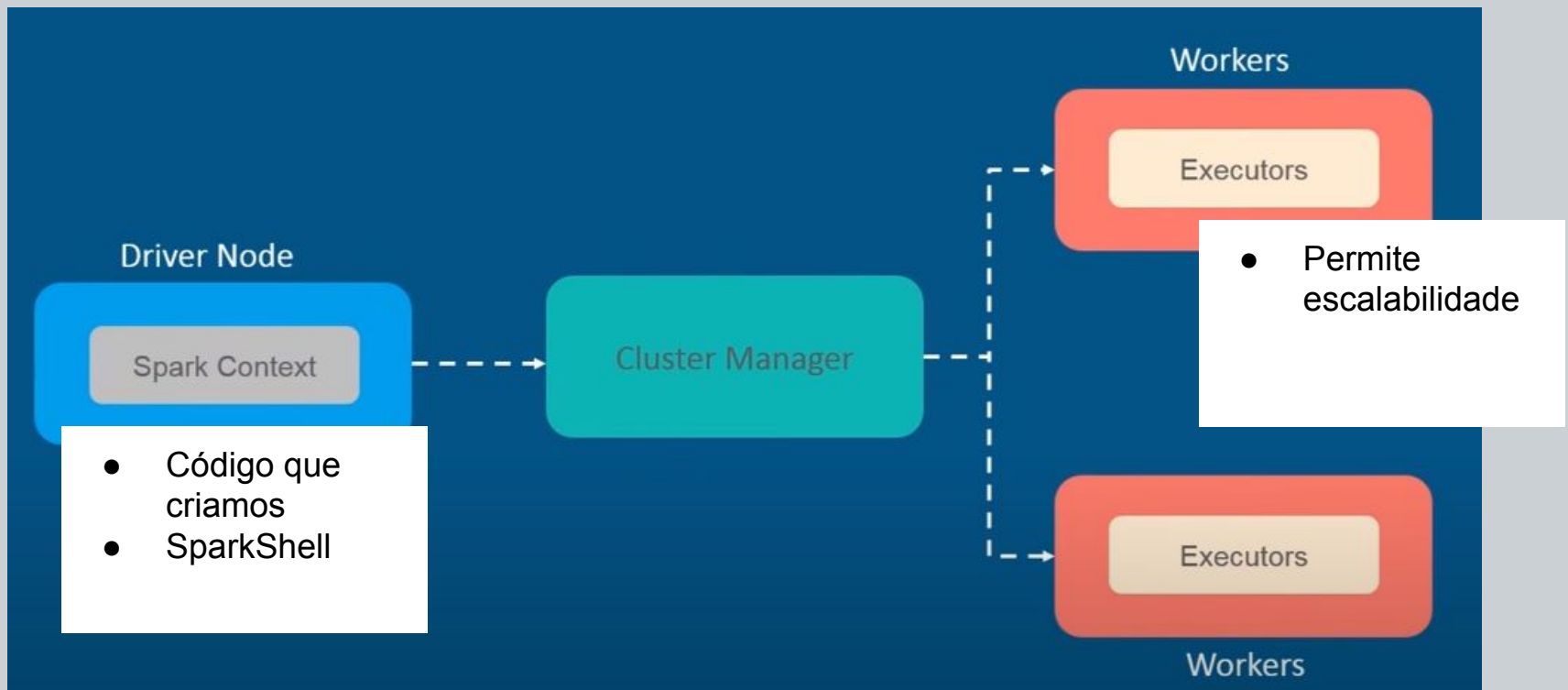
# Make predictions
predictions = model.transform(dataset)

# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Qual é a inovação?



<https://www.youtube.com/watch?v=jffQhcweGwY>

TensorFlow

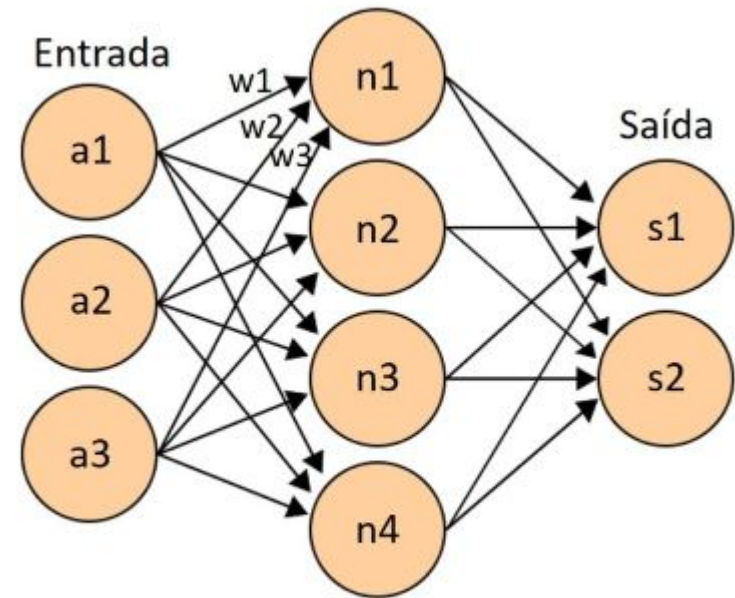
- Criado pela equipe do Google Brain
- Permite a criação de redes neurais em larga escala
- Keras
- Casos de uso:
 - NLP
 - Reconhecimento de imagens
 - Tratamento de vídeos e sons
 - Análise textual: sentimento e detecção de fraude

<https://www.tensorflow.org/tutorials/keras/classification?hl=pt-br>

TensorFlow

- Permite a criação de redes neurais

```
# Pesos da camada 1
w1 = tf.Variable(tf.random_normal([n_input, n_hidden_1]))
# Bias da camada 1
b1 = tf.Variable(tf.random_normal([n_hidden_1]))
# Aplicando a função sigmoide na camada 1
camada_1 = tf.nn.sigmoid(tf.add(tf.matmul(x,w1),b1))
```



$$n1 = \text{sigmoid}(a1*w1 + a2*w2 + a3*w3 + b)$$

<https://www.tensorflow.org/tutorials/keras/classification?hl=pt-br>

TensorFlow

- Exemplo para classificação de texto

Chegou a hora de criar sua rede neural:

```
[ ] 1 embedding_dim = 16

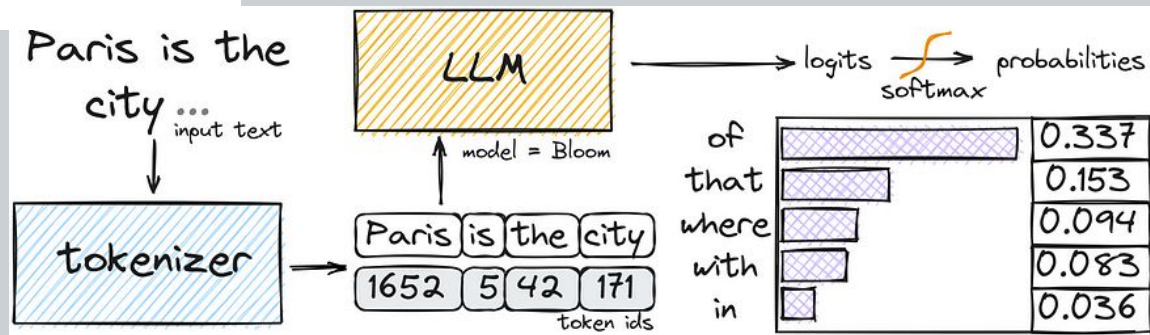
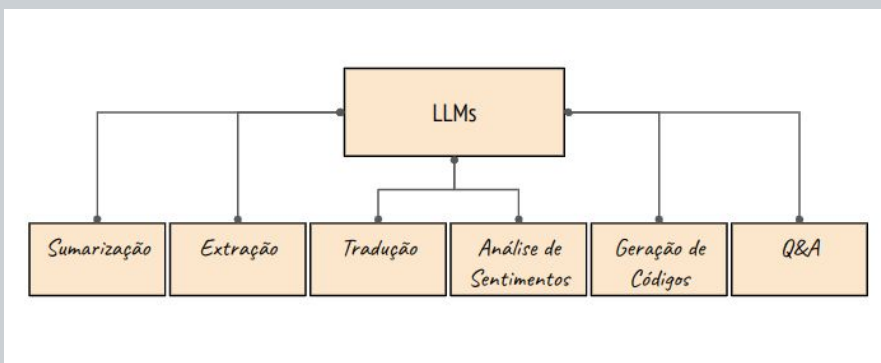
1 model = tf.keras.Sequential([
2     layers.Embedding(max_features + 1, embedding_dim),
3     layers.Dropout(0.2),
4     layers.GlobalAveragePooling1D(),
5     layers.Dropout(0.2),
6     layers.Dense(1)])
7
8 model.summary()
```

As camadas são empilhadas sequencialmente para construir o classificador:

1. A primeira é uma camada `Embedding`, que recebe avaliações codificadas em inteiros e avalia um vetor de embedding para cada palavra-índice. Esses vetores são aprendidos à medida que o modelo é treinado. Os vetores acrescentam uma dimensão à matriz de saída. As dimensões resultantes são: `(batch, sequence, embedding)` (lote, sequência, embedding). Para saber mais sobre embeddings, confira o tutorial [Embeddings de palavras](#).
2. A segunda camada é `GlobalAveragePooling1D`, que retorna um vetor de saída de tamanho fixo para cada exemplo, calculando a média da dimensão de sequência. Dessa forma, o modelo consegue lidar com entradas de tamanho variável da forma mais simples possível.
3. A última camada é densamente conectada com um único nó de saída.

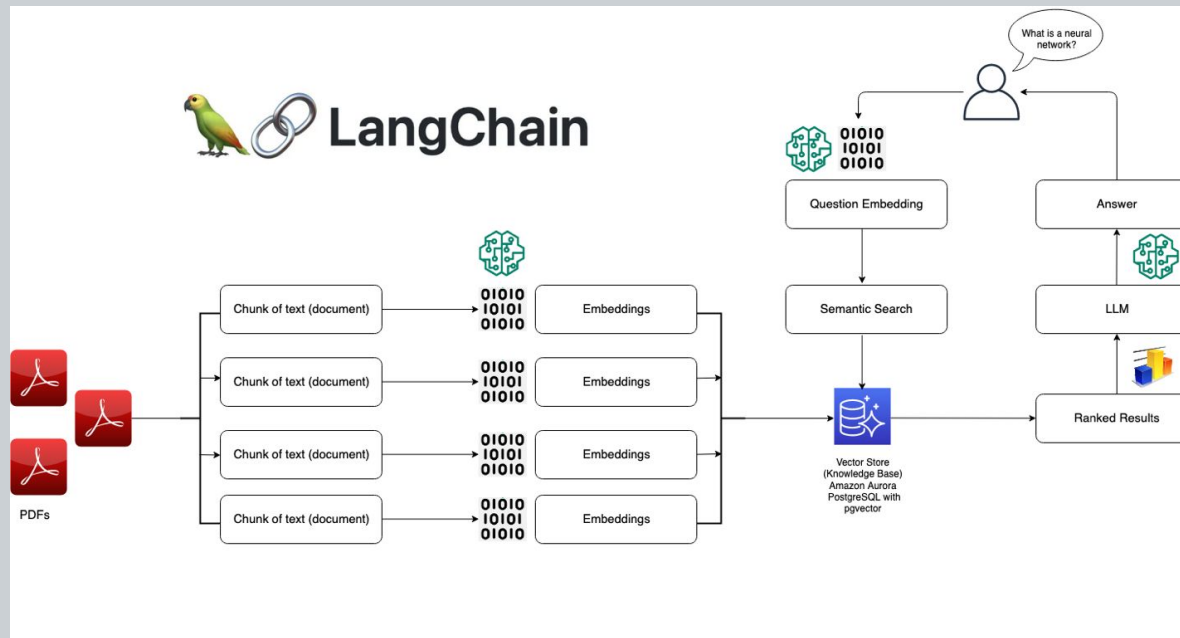
Large Language Models (LLM)

- O que são LLM?
 - Modelos treinados com um conjunto enorme de dados
 - Podem prever sequência de palavras no texto



Large Language Models (LLMs)

- LangChain (LangChain4J)



<https://aws.amazon.com/pt/what-is/langchain/>

Luiz Henrique Zambom Santana

lhzsantana@gmail.com

<https://www.linkedin.com/in/luizsantana/>



UNIVERSIDADE FEDERAL
DE SANTA CATARINA