UNIVERSIDADE ESTADUAL DO CEARÁ
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
MINERAÇÃO MASSIVA DE DADOS

CLASSIFICAÇÃO DE TIPOS DE ARROZ

KAIO RODRIGUES DA CUNHA

FORTALEZA
2023

Introdução

O objetivo desse relatório é descrever o processo de desenvolvimento de um modelo de classificação de tipos de grãos de arroz utilizando técnicas de mineração de dados e aprendizado de máquina.

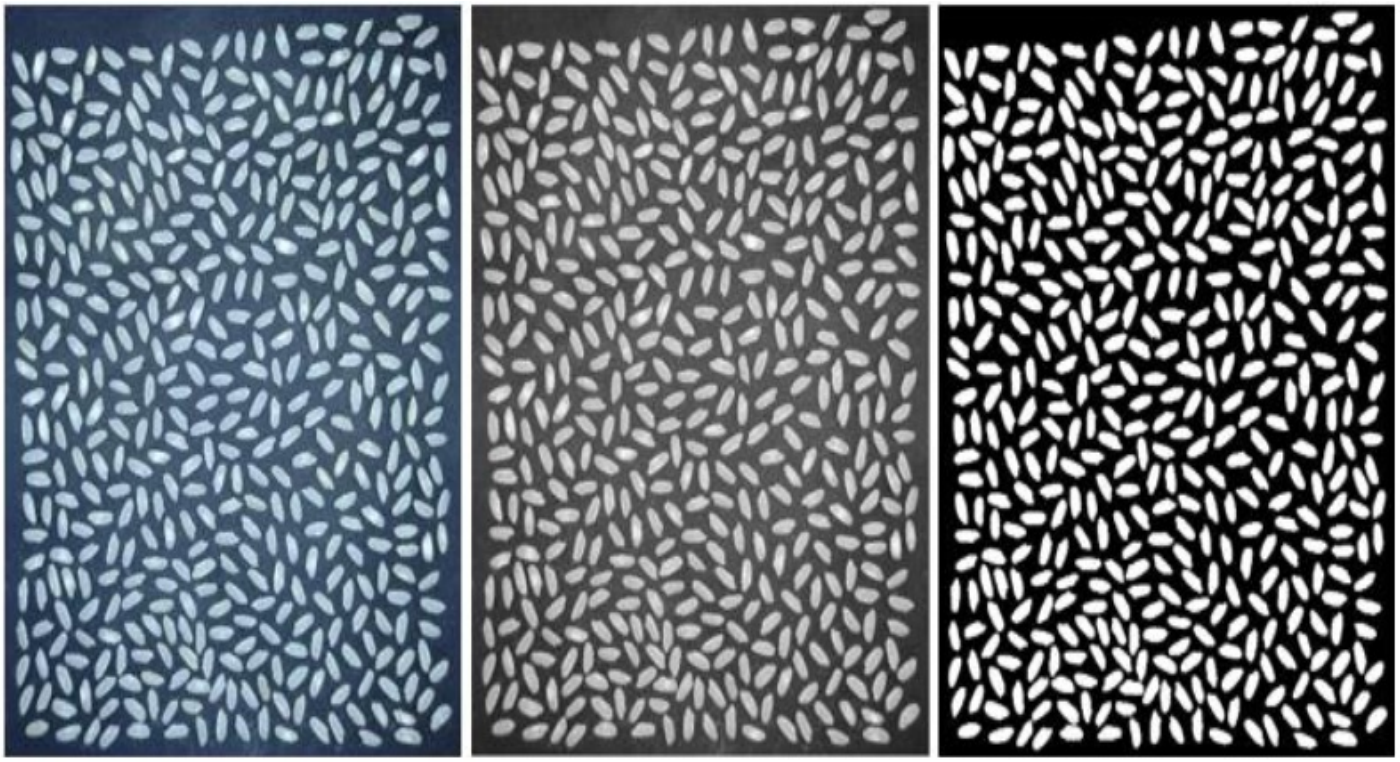
O relatório abrange a contextualização do problema, a execução do experimento e suas etapas - incluindo uma breve descrição das técnicas empregadas e dos resultados alcançados - e uma conclusão acerca do experimento.

Neste trabalho as técnicas de classificação empregadas foram KNN, Random Forest, Multilayer Perceptron, Naive Bayes e Support Vector Machine.

Dataset

- Rice (Cammeo e Osmancik)

Descrição: Um total de 3810 imagens de grãos de arroz foram tiradas para as duas espécies, processadas e inferências de características foram feitas. Foram obtidas 7 características morfológicas para cada grão de arroz.



Atributo a ser predito:
Tipo de grão de arroz

Número de instâncias:
3810

Número de Atributos:
7 features extraídas das imagens e a classe

Informações dos Atributos:

1.) Area: Retorna o número de pixels dentro dos limites do grão de arroz.

2.) Perimeter: Calcula a circunferência calculando a distância entre os pixels ao redor dos limites do grão de arroz.

3.) Major Axis Length: A linha mais longa que pode ser desenhada no grão de arroz, ou seja, a distância do eixo principal, dá.

4.) Minor Axis Length: A linha mais curta que pode ser desenhada no grão de arroz, ou seja, a distância do eixo menor, dá.

5.) Eccentricity: Mede o quão arredondada é a elipse, que tem os mesmos momentos do grão de arroz.

6.) Convex Area: Retorna a contagem de pixels da menor casca convexa da região formada pelo grão de arroz.

7.) Extent: Retorna a razão entre a região formada pelo grão de arroz e os pixels da caixa delimitadora.

8.) Class: Arroz Cammeo e Osmancik

Desenvolvimento

Bibliotecas Utilizadas

As seguintes bibliotecas foram importadas no código:
pyspark: Biblioteca principal para processamento distribuído e análise de dados no Spark.
pyspark.sql: Módulo que fornece suporte ao processamento de dados estruturados usando DataFrames.
pyspark.ml: Módulo para construção de pipelines de machine learning.
pyspark.ml.classification: Submódulo contendo algoritmos de classificação.
pyspark.ml.feature: Submódulo com classes para extração e transformação de características.
pyspark.ml.evaluation: Submódulo que contém métricas de avaliação para modelos de classificação.
pyspark.mllib.util: Módulo que oferece utilidades para algoritmos de machine learning do Spark.
sklearn: Biblioteca popular de machine learning em Python.
numpy: Biblioteca para operações numéricas eficientes em Python.
pandas: Biblioteca para manipulação e análise de dados.
matplotlib: Biblioteca para visualização de dados em Python.
seaborn: Biblioteca baseada em matplotlib para visualização estatística de dados.

Pré-processamento de Dados

O código inclui algumas etapas de pré-processamento de dados, utilizando tanto o Spark quanto o scikit-learn (sklearn).

O Spark é usado para carregar os dados, criar uma sessão Spark (SparkSession), definir o esquema dos dados e realizar transformações como StringIndexer e VectorAssembler.

O scikit-learn é utilizado para executar o KNN(K-Nearest Neighbors)

Modelagem de Machine Learning

O código apresenta várias técnicas de classificação utilizando os algoritmos disponíveis no Spark e no scikit-learn:

Naive Bayes: Utilizando a classe NaiveBayes do Spark.
Regressão Logística: Utilizando a classe LogisticRegression do Spark.
Support Vector Machine (SVM): Utilizando a classe LinearSVC e OneVsRest do Spark.
Random Forest: Utilizando a classe RandomForestClassifier do Spark.
Árvore de Decisão: Utilizando as classes DecisionTreeClassifier do Spark.
Multilayer Perceptron: Utilizando a classe MultilayerPerceptronClassifier do Spark.
K-Nearest Neighbors (KNN): Utilizando a classe KNeighborsClassifier do scikit-learn.

Avaliação do Modelo

A avaliação do modelo é realizada utilizando a métrica de acurácia (accuracy_score) implementada no spark e no scikit-learn.

Implementação

```
[1]: # Load libraries
import pyspark
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.util import MLUtils

from pyspark.ml.feature import StringIndexer, IndexToString
from pyspark.ml.feature import VectorAssembler, VectorIndexer
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import LinearSVC, OneVsRest
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.ml.linalg import Vectors
from pyspark.mllib.util import MLUtils

## SKLearn Lib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score

import time
start_time = time.time()
%matplotlib inline
```

```
[3]: # Path to dataset file
data_path='./data/'

# Sample of train and test dataset
train_sample = 0.7
test_sample = 0.3
```

Definição do caminho do arquivo do conjunto de dados como './data/' e especificação das proporções de amostra para o conjunto de treinamento e teste.

- O caminho do arquivo do conjunto de dados é definido como './data/'.
- A proporção de amostra para o conjunto de treinamento é definida como 0.7, o que significa que 70% dos dados serão usados para treinar o modelo.
- A proporção de amostra para o conjunto de teste é definida como 0.3, o que significa que 30% dos dados serão usados para testar o modelo.

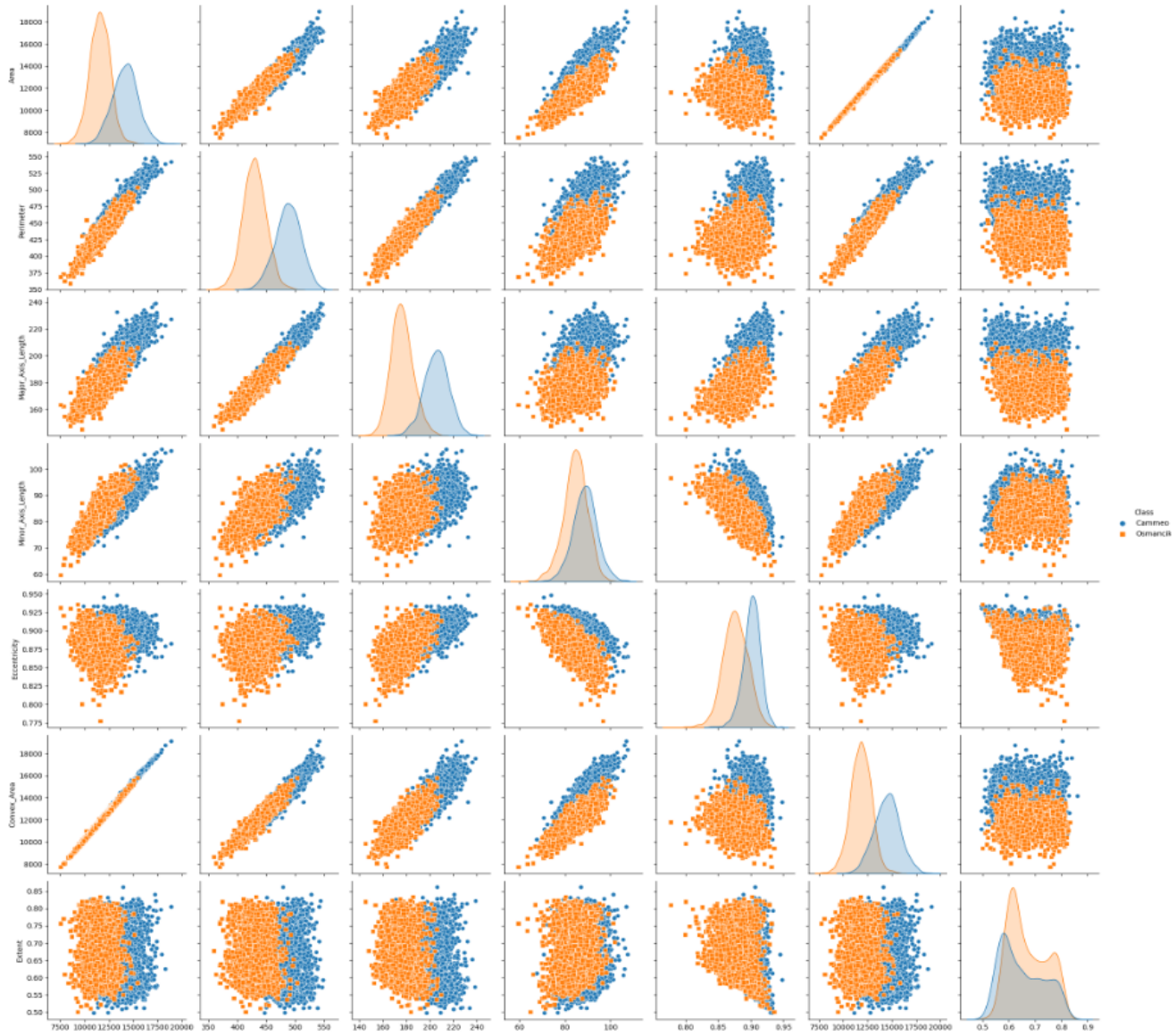
Essas variáveis fornecem informações sobre o caminho do arquivo do conjunto de dados e as proporções de amostra para treinamento e teste.

Importação de um conjunto de dados em formato CSV chamado 'Rice_Cammeo_Osmancik.csv' e exibição das primeiras cinco linhas do dataset.

```
[6]: print("Number of itens per class")
     dataset.groupby('Class').size()

     Number of itens per class
[6]: Class
     Cammeo      1630
     Osmancik    2180
     dtype: int64

[7]: # Read features and calass
     feature_columns = ['Area', 'Perimeter', 'Major_Axis_Length', 'Minor_Axis_Length','Eccentricity', 'Convex_Area', 'Extent']
     X = dataset[feature_columns].values
     y = dataset['Class'].values

     # SKLearn need all column as numbers. Tranform "class" column in number
     le = LabelEncoder()
     y = le.fit_transform(y)

     # Split ramdomly the dataset into train and test group
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = test_sample, random_state = 0)
```

1. Definição da lista de colunas de características como feature_columns. Essas colunas incluem 'Area', 'Perimeter', 'Major_Axis_Length', 'Minor_Axis_Length', 'Eccentricity', 'Convex_Area' e 'Extent'.
2. Extração das características (X) do conjunto de dados, selecionando apenas as colunas listadas em feature_columns. Os valores são armazenados na variável X.
3. Extração da variável de classe (y) do conjunto de dados, selecionando a coluna 'Class'. Os valores são armazenados na variável y.
4. Como o scikit-learn (sklearn) requer que todas as colunas sejam numéricas, é utilizado o LabelEncoder para transformar a coluna de classe em números. A variável “y” é convertida para seus equivalentes numéricos.
5. Divide aleatoriamente o conjunto de dados em grupos de treinamento e teste. A proporção do conjunto de teste é definida pela variável test_sample, que foi atribuída anteriormente. As variáveis X_train, X_test, y_train e y_test armazenam os conjuntos de treinamento e teste, respectivamente.

```
[7]: plt.figure()
     sns.pairplot(dataset, hue = "Class", height=3, markers=["o", "s"])
     plt.show()
```


Area
Perimeter
Major_Axis_Length
Minor_Axis_Length
Eccentricity
Convex_Area
Extent
Class
Cammeo
Osmancik

```
[9]: start_time_knn =  time.time()
     # Instantiate learning model (k = 3)
     classifier = KNeighborsClassifier(n_neighbors=3)

     # Fitting the model
     classifier.fit(X_train, y_train)

     # Predicting the Test set results
     y_pred = classifier.predict(X_test)
```

```
[10]: accuracy_knn = accuracy_score(y_test, y_pred)*100
      time_knn = time.time() - start_time_knn
      print('KNN accuracy = ' + str(round(accuracy_knn, 1)) + ' %.')
      print("K-Nearest Neighbors (KNN): accuracy = %3.1f %%" % accuracy_knn)
      print("K-Nearest Neighbors (KNN): time = %3.3f s" % time_knn)

      KNN accuracy = 87.9 %.
      K-Nearest Neighbors (KNN): accuracy = 87.9 %
      K-Nearest Neighbors (KNN): time = 0.205 s
```

Modelagem de classificação utilizando o algoritmo K-Nearest Neighbors (KNN) e exibição da acurácia e o tempo de execução do modelo.

Etapas realizadas pelo código:

1. A variável start_time_knn é inicializada para medir o tempo de execução do algoritmo KNN.
2. É instanciado o modelo de aprendizado KNN com o parâmetro n_neighbors definido como 3, ou seja, considerando os 3 vizinhos mais próximos.
3. O modelo é ajustado aos dados de treinamento usando o método fit, passando as variáveis de treinamento X_train e y_train.
4. É feita a previsão dos resultados do conjunto de teste utilizando o método predict do modelo KNN, passando as variáveis de teste X_test.
5. A acurácia do modelo KNN é calculada comparando as previsões (y_pred) com os valores reais do conjunto de teste (y_test), utilizando a função accuracy_score do sklearn. O resultado é multiplicado por 100 para obter a porcentagem de acurácia.
6. O tempo de execução do algoritmo KNN é calculado subtraindo o start_time_knn do tempo atual.
7. As métricas de acurácia e tempo de execução do modelo KNN são impressas na tela.

```
[11]: # Load rice CSV dataset to Spark Dataframe
      orig_data = spark.read.format("csv").options(sep=';',header='true',inferschema='true').\
                  load("Rice_Cammeo_Osmancik.csv")

      print("Original Dataframe read from CSV file")
      #orig_data.dtypes
      orig_data.show(5)

      Original Dataframe read from CSV file
      +-----+-----------+-----------------+-----------------+------------+-----------+-----------+------+
      | Area|  Perimeter|Major_Axis_Length|Minor_Axis_Length|Eccentricity|Convex_Area|     Extent| Class|
      +-----+-----------+-----------------+-----------------+------------+-----------+-----------+------+
      |15231|525.5789795|      229.7498779|      85.09378815| 0.928882003|      15617|0.572895527|Cammeo|
      |14656|494.3110046|      206.0200653|      91.73097229| 0.895404994|      15072|0.615436316|Cammeo|
      |14634|501.1220093|       214.106781|      87.76828766| 0.912118077|      14954|0.693258822|Cammeo|
      |13176|458.3429871|      193.3373871|      87.44839478| 0.891860902|      13368|0.640669048|Cammeo|
      |14688|507.1669922|      211.7433777|      89.31245422| 0.906690896|      15262|0.646023929|Cammeo|
      +-----+-----------+-----------------+-----------------+------------+-----------+-----------+------+
      only showing top 5 rows
```

Carregamento o conjunto de dados CSV 'Rice_Cammeo_Osmancik.csv' em um DataFrame do Spark e exibem as cinco primeiras linhas do DataFrame.
Etapas realizadas pelo código:

1. O conjunto de dados CSV é lido usando o método read do SparkSession e o formato "csv".
2. As opções são configuradas para especificar que o separador do CSV é ';' (ponto e vírgula), a primeira linha contém os nomes das colunas e o esquema dos dados é inferido automaticamente.
3. O método load é usado para carregar o arquivo CSV com as opções configuradas.
4. A mensagem "Original Dataframe read from CSV file" é impressa na tela.
5. As cinco primeiras linhas do DataFrame são exibidas usando o método show(5).

```
[13]: # ML libraries doesn't accept string column => everything should be numeric!
      # create a numeric column "label" based on string column "class"

      indexer = StringIndexer(inputCol="Class", outputCol="label").fit(orig_data)
      label_data = indexer.transform(orig_data)

      # Save the inverse map from numeric "label" to string "class" to be used further in response
      labelReverse = IndexToString().setInputCol("label")

      # Show labeled dataframe with numeric lable
      print("Dataframe with numeric lable")
      label_data.show(5)

      Dataframe with numeric lable
      +-----+-----------+----------------+----------------+------------+-----------+-----------+------+-----+
      | Area|  Perimeter|Major_Axis_Length|Minor_Axis_Length|Eccentricity|Convex_Area|     Extent| Class|label|
      +-----+-----------+----------------+----------------+------------+-----------+-----------+------+-----+
      |15231|525.5789795|      229.7498779|      85.09378815| 0.928882003|      15617|0.572895527|Cammeo|  1.0|
      |14656|494.3110046|      206.0200653|      91.73097229| 0.895404994|      15072|0.615436316|Cammeo|  1.0|
      |14634|501.1220093|       214.106781|      87.76828766| 0.912118077|      14954|0.693258822|Cammeo|  1.0|
      |13176|458.3429871|      193.3373871|      87.44839478| 0.891860902|      13368|0.640669048|Cammeo|  1.0|
      |14688|507.1669922|      211.7433777|      89.31245422| 0.906690896|      15262|0.646023929|Cammeo|  1.0|
      +-----+-----------+----------------+----------------+------------+-----------+-----------+------+-----+
      only showing top 5 rows
```

Transformação da coluna categórica "Class" em uma coluna numérica "label" usando a classe StringIndexer do Spark ML. Além disso, é salvo um mapeamento reverso do índice numérico para a string original da classe, para uso futuro.
Etapas realizadas pelo código:

1. A classe StringIndexer é instanciada com o parâmetro inputCol definido como "Class" (a coluna categórica a ser transformada) e outputCol definido como "label" (o nome da coluna numérica resultante).
2. O método fit é aplicado à instância do StringIndexer, passando o DataFrame orig_data, para ajustar o modelo de indexação com base na coluna "Class".
3. O DataFrame orig_data é transformado usando o modelo StringIndexer ajustado, e o resultado é armazenado no DataFrame label_data.
4. É criada uma instância do IndexToString para salvar o mapeamento inverso do índice numérico para a string original da classe.
5. A mensagem "Dataframe with numeric label" é impressa na tela.

6. As cinco primeiras linhas do DataFrame label_data são exibidas usando o método show(5).

```
[14]: # Drop string column "class", no string column
label_data = label_data.drop("Class")

# Most Machine Learning lib inpute 2 columns: label (output) and feature (input)
# The label column is the result to train ML algorithm
# The feature column should join all parameters as a Vector

# Set the column names that is not part of features list
ignore = ['label']
# list will be all columns parts of features
list = [x for x in label_data.columns if x not in ignore]

# VectorAssembler mount the vector of features
assembler = VectorAssembler(
            inputCols=list,
            outputCol='features')

# Create final dataframe composed by label and a column of features vector
data = (assembler.transform(label_data).select("label","features"))

print("Final Dataframe suitable to classifier input format")
#data.printSchema()
data.show(5)
```

```
Final Dataframe suitable to classifier input format
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  1.0|[15231.0,525.5789...|
|  1.0|[14656.0,494.3110...|
|  1.0|[14634.0,501.1220...|
|  1.0|[13176.0,458.3429...|
|  1.0|[14688.0,507.1669...|
+-----+--------------------+
only showing top 5 rows
```

Aqui são feitas etapas adicionais de pré-processamento para preparar o DataFrame label_data para o formato de entrada esperado pelos algoritmos de machine learning.

Etapas realizadas pelo código:

1. A coluna "Class" é removida do DataFrame label_data usando o método drop, uma vez que a maioria das bibliotecas de machine learning não aceita colunas de tipo string.
2. As colunas que não são parte das características (features) são definidas na lista ignore. Essas colunas serão ignoradas na criação do vetor de características.
3. A lista list é criada para conter todas as colunas que são parte das características, ou seja, todas as colunas que não estão na lista ignore.

4. É criado um VectorAssembler que irá combinar todas as colunas listadas em list em um único vetor de características, com o nome "features". O vetor de características será a entrada para o algoritmo de machine learning.
5. É criado um novo DataFrame chamado data que consiste nas colunas "label" (resultado) e "features" (vetor de características), obtidas a partir da aplicação do VectorAssembler ao DataFrame label_data.
6. A mensagem "Final Dataframe suitable to classifier input format" é impressa na tela.
7. As cinco primeiras linhas do DataFrame data são exibidas usando o método show(5).

O DataFrame data agora está pronto para ser utilizado como entrada nos algoritmos de classificação de machine learning. Ele possui uma coluna "label" com os valores numéricos correspondentes à classe e uma coluna "features" que contém um vetor com os valores das características para cada registro.

```
[15]: # Split ramdomly the dataset into train and test group
      # [0.7,0.3] => 70% for train and 30% for test
      # [1.0,0.2] => 100% for train and 20% for test, not good, acuracy always 100%
      # [0.1,0.02] => 10% for train and 2% for test, if big datasets
      # 1234 is the random seed

      (train, test) = data.randomSplit([train_sample, test_sample], 1234)
```

A célula fornecida realiza a divisão aleatória do DataFrame data em grupos de treinamento e teste, com base nas proporções especificadas.

Aqui está uma descrição da etapa realizada pelo código:

1. O método randomSplit é aplicado ao DataFrame data com os parâmetros [train_sample, test_sample], que especificam as proporções desejadas para os grupos de treinamento e teste, respectivamente.
2. O valor 1234 é fornecido como semente aleatória (random seed) para garantir que a divisão seja reproduzível.
3. O resultado da divisão aleatória é atribuído às variáveis train e test, que representam o DataFrame de treinamento e o DataFrame de teste, respectivamente.

A proporção definida por [train_sample, test_sample] determina a porcentagem dos dados que será atribuída ao grupo de treinamento e ao grupo de teste. Por exemplo, [0.7, 0.3] significa que 70% dos dados serão usados para treinamento e 30% dos dados serão usados para teste.

[16]:
```
start_time_dt =  time.time()

# impurity could be: entropy, gini'

trainer = DecisionTreeClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability',\
                                 rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0,\
                                 maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', seed=None)

#trainer = LogisticRegression(maxIter=10, tol=1E-6, fitIntercept=True)

# train the model and get the result
model = trainer.fit(train)
result_dt = model.transform(test)
```

[17]:
```
# compute accuracy on the test set against model
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",\
            metricName="accuracy")

accuracy_dt = evaluator.evaluate(result_dt) * 100
time_dt = time.time() - start_time_dt

print("Decision Tree: accuracy = %3.1f %%" % accuracy_dt)
print("Decision Tree: time = %3.3f s" % time_dt)
```

```
Decision Tree: accuracy = 92.1 %
Decision Tree: time = 2.781 s
```

[18]:
```
print("Decision Tree Final Result")
result_dt.show(5)
```

```
Decision Tree Final Result
+-----+--------------------+-------------+--------------------+----------+
|label|            features|rawPrediction|         probability|prediction|
+-----+--------------------+-------------+--------------------+----------+
|  0.0|[7551.0,369.06399...| [1093.0,9.0]|[0.99183303085299...|       0.0|
|  0.0|[7833.0,373.15701...| [1093.0,9.0]|[0.99183303085299...|       0.0|
|  0.0|[8499.0,370.44601...| [1093.0,9.0]|[0.99183303085299...|       0.0|
|  0.0|[8992.0,377.33200...| [1093.0,9.0]|[0.99183303085299...|       0.0|
|  0.0|[9050.0,379.89898...| [1093.0,9.0]|[0.99183303085299...|       0.0|
+-----+--------------------+-------------+--------------------+----------+
only showing top 5 rows
```

Treino de um modelo de um Decision Tree usando o algoritmo DecisionTreeClassifier do Spark ML, avaliando a acurácia do modelo e exibindo os resultados.

Etapas realizadas pelo código:

1. O DecisionTreeClassifier é instanciado com vários parâmetros, como featuresCol, labelCol, maxDepth, maxBins, etc. Esses parâmetros definem as configurações do modelo de árvore de decisão.
2. O método fit é aplicado ao modelo, passando o DataFrame de treinamento (train), para treinar o modelo com base nos dados de treinamento.
3. O método transform é aplicado ao modelo treinado, passando o DataFrame de teste (test), para gerar as previsões para o conjunto de teste.
4. É criado um avaliador MulticlassClassificationEvaluator para avaliar a acurácia das previsões em relação às etiquetas reais.
5. O avaliador é usado para calcular a acurácia do modelo (accuracy_dt).
6. O tempo de execução do treinamento e previsão do modelo é calculado subtraindo o tempo inicial (start_time_dt) do tempo atual.
7. A mensagem "Decision Tree: accuracy = ..." é impressa na tela, mostrando a acurácia do modelo de árvore de decisão.
8. A mensagem "Decision Tree: time = ..." é impressa na tela, mostrando o tempo de execução do modelo de árvore de decisão.
9. As cinco primeiras linhas do resultado do modelo (result_dt) são exibidas usando o método show(5).

Os resultados incluem a coluna "prediction" que contém as previsões do modelo para o conjunto de teste.

```
[19]: start_time_rf =  time.time()

      trainer = RandomForestClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability',\
                                       rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0,\
                                       numTrees=50, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0,\
                                       maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini')

      # impurity could be: entropy, gini'
      # numTrees= set the number of random trees to create

      # train the model and get the result
      model = trainer.fit(train)
      result_rf = model.transform(test)

[20]: # compute accuracy on the test set against model
      evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",\
                  metricName="accuracy")

      accuracy_rf = evaluator.evaluate(result_rf) * 100
      time_rf = time.time() - start_time_rf

      print("Random Forest: accuracy = %3.1f %%" % accuracy_rf)
      print("Random Forest: time = %3.3f s" % time_rf)

      Random Forest: accuracy = 92.3 %
      Random Forest: time = 2.376 s

[21]: print("Decision Tree Final Result")
      result_rf.show(5)

      Decision Tree Final Result
      +-----+--------------------+--------------------+--------------------+----------+
      |label|            features|       rawPrediction|         probability|prediction|
      +-----+--------------------+--------------------+--------------------+----------+
      |  0.0|[7551.0,369.06399...|[48.6765510463053...|[0.97353102092610...|       0.0|
      |  0.0|[7833.0,373.15701...|[48.6765510463053...|[0.97353102092610...|       0.0|
      |  0.0|[8499.0,370.44601...|[48.7250740553372...|[0.97450148110674...|       0.0|
      |  0.0|[8992.0,377.33200...|[48.7852303743163...|[0.97570460748632...|       0.0|
      |  0.0|[9050.0,379.89898...|[48.9866255998265...|[0.97973251199653...|       0.0|
      +-----+--------------------+--------------------+--------------------+----------+
      only showing top 5 rows
```

Treino de um modelo de Random Forest usando o algoritmo RandomForestClassifier do Spark ML, avaliando a acurácia do modelo e exibindo os resultados.

Etapas realizadas pelo código:

1. O RandomForestClassifier é instanciado com vários parâmetros, como featuresCol, labelCol, maxDepth, numTrees, etc. Esses parâmetros definem as configurações do modelo de floresta aleatória.
2. O método fit é aplicado ao modelo, passando o DataFrame de treinamento (train), para treinar o modelo com base nos dados de treinamento.
3. O método transform é aplicado ao modelo treinado, passando o DataFrame de teste (test), para gerar as previsões para o conjunto de teste.
4. É criado um avaliador MulticlassClassificationEvaluator para avaliar a acurácia das previsões em relação às etiquetas reais.
5. O avaliador é usado para calcular a acurácia do modelo (accuracy_rf).
6. O tempo de execução do treinamento e previsão do modelo é calculado subtraindo o tempo inicial (start_time_rf) do tempo atual.
7. A mensagem "Random Forest: accuracy = ..." é impressa na tela, mostrando a acurácia do modelo de floresta aleatória.

8. A mensagem "Random Forest: time = ..." é impressa na tela, mostrando o tempo de execução do modelo de floresta aleatória.
9. As cinco primeiras linhas do resultado do modelo (result_rf) são exibidas usando o método show(5).

```
[22]: start_time_pr =  time.time()

      # specify layers for the neural network
      # parameter 1: input layer, should be the number of features
      # parameter 2 and 3: the number os perceptron in two hidden layers
      # parameter 4: output layer should be the number os categories (labels)
      # More hidden layers will make the neural network more complex but do not ensure more accuracy.
      layers = [7, 128, 128, 128, 2]

      # Create the trainer and set its parameters:
      # featuresCol: name of feature_column
      # labelCol: name of label column
      # maxIter: number max interaction
      # layers: number of input, output and hidden layer (see above)
      # tol: convergence tolerance towards the outputs and the correct results. The default value is 1e-06.
      #      Smaller values yield more accurate results while large values might lead to overfitting.
      # seed: the random seed value using to random numbers generator.
      # blockSize: number of inputs to be included during each iteration of the algorithm.
      #       Default value is 128. Smaller blockSize improves accuracy at the expense of
      #       prolonged learning time and vice versa.
      # stepSize: learning rate of the algorithm, usually between 0.0 to 1.0. It is how quickly or slowly the model
      #       learns. The default is 0.03. A smaller value can lead to improved accuracy while larger values can
      #       lead to over-fitting.
      # solver: specifies which optimization algorithm should be used to find the local minimum.
      #       'gd' is gradient descent and 'l-bfgs' is Limited-memory BFGS (default).

      trainer = MultilayerPerceptronClassifier(featuresCol='features', labelCol='label',\
                maxIter=1000, tol=1e-07, layers=layers, seed=1234, blockSize=64, stepSize=0.02, solver='l-bfgs')

      # train the model and get the result
      model = trainer.fit(train)
      result_pr = model.transform(test)

      print("Perceptron Final Result")
      result_pr.show(5)
```

```
23/06/30 00:30:26 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
23/06/30 00:30:26 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.ForeignLinkerBLAS

23/06/30 00:30:26 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
23/06/30 00:30:26 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
Perceptron Final Result
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|  0.0|[7551.0,369.06399...|[0.29374495081634...|[0.56250000329076...|       0.0|
|  0.0|[7833.0,373.15701...|[0.29374495081634...|[0.56250000329076...|       0.0|
|  0.0|[8499.0,370.44601...|[0.29374495081634...|[0.56250000329076...|       0.0|
|  0.0|[8992.0,377.33200...|[0.29374495081634...|[0.56250000329076...|       0.0|
|  0.0|[9050.0,379.89898...|[0.29374495081634...|[0.56250000329076...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
```

```
[23]: # compute accuracy on the test set against model
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",\
            metricName="accuracy")

accuracy_pr = evaluator.evaluate(result_pr) * 100
time_pr = time.time() - start_time_pr

print("Multilayer Perceptron: accuracy = %3.1f %%" % accuracy_pr)
print("Multilayer Perceptron: time = %3.3f s" % time_pr)
```

```
Multilayer Perceptron: accuracy = 57.6 %
Multilayer Perceptron: time = 3.556 s
```

```
[24]: print("Perceptron final result with name of class")
labelReverse.transform(result_pr).show()
```

```
Perceptron final result with name of class
+-----+--------------------+--------------------+--------------------+----------+---------------------------------+
|label|            features|       rawPrediction|         probability|prediction|IndexToString_eb6637bff2fa__output|
+-----+--------------------+--------------------+--------------------+----------+---------------------------------+
|  0.0|[7551.0,369.06399...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[7833.0,373.15701...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[8499.0,370.44601...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[8992.0,377.33200...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9050.0,379.89898...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9070.0,375.09399...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9080.0,396.54000...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9117.0,393.83801...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9127.0,381.87899...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9165.0,380.66000...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9212.0,384.59600...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9236.0,394.34100...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9359.0,395.98800...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9369.0,396.76000...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9414.0,407.82501...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9431.0,383.95901...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9441.0,384.33999...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9447.0,407.09201...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9510.0,401.49200...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
|  0.0|[9516.0,386.82101...|[0.29374495081634...|[0.56250000329076...|       0.0|                         Osmancik|
+-----+--------------------+--------------------+--------------------+----------+---------------------------------+
only showing top 20 rows
```

Treinamento de um modelo de Multilayer Perceptron usando o algoritmo MultilayerPerceptronClassifier do Spark ML, avaliando a acurácia do modelo e exibindo os resultados.

Etapas realizadas pelo código:

1. É especificada uma lista layers que define a arquitetura da rede neural. A lista contém o número de neurônios em cada camada, incluindo a camada de entrada, as camadas ocultas e a camada de saída. No exemplo fornecido, a rede possui uma camada de entrada com 7 neurônios (correspondendo ao número de recursos), três camadas ocultas com 128 neurônios cada e uma camada de saída com 2 neurônios (correspondendo às categorias/classes).
2. O MultilayerPerceptronClassifier é instanciado com vários parâmetros, como featuresCol, labelCol, maxIter, tol, layers, seed, blockSize, stepSize e solver. Esses parâmetros definem as configurações do modelo de perceptron multicamadas.
3. O método fit é aplicado ao modelo, passando o DataFrame de treinamento (train), para treinar o modelo com base nos dados de treinamento.
4. O método transform é aplicado ao modelo treinado, passando o DataFrame de teste (test), para gerar as previsões para o conjunto de teste.
5. A mensagem "Perceptron Final Result" é impressa na tela, mostrando as cinco primeiras linhas do resultado do modelo (result_pr).

6. Um avaliador MulticlassClassificationEvaluator é criado para avaliar a acurácia das previsões em relação às etiquetas reais.
7. O avaliador é usado para calcular a acurácia do modelo (accuracy_pr).
8. O tempo de execução do treinamento e previsão do modelo é calculado subtraindo o tempo inicial (start_time_pr) do tempo atual.
9. A mensagem "Multilayer Perceptron: accuracy = ..." é impressa na tela, mostrando a acurácia do modelo de perceptron multicamadas.
10. A mensagem "Multilayer Perceptron: time = ..." é impressa na tela, mostrando o tempo de execução do modelo de perceptron multicamadas.
11. O método transform é aplicado ao resultado do modelo (result_pr) junto com o labelReverse para exibir as previsões com os nomes das classes.

```
# create the trainer and set its parameters
trainer = NaiveBayes(smoothing=1.0, modelType="multinomial")

#trainer = LogisticRegression(maxIter=10, tol=1E-6, fitIntercept=True)
#trainer = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)

# train the model and get the result
model = trainer.fit(train)
result_nb = model.transform(test)
```

[26]:
```
# compute accuracy on the test set against model
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",\
            metricName="accuracy")

accuracy_nb = evaluator.evaluate(result_nb) * 100
time_nb = time.time() - start_time_nb

print("Naive Bayes: accuracy = %3.1f %%" % accuracy_nb)
print("Naive Bayes: time = %3.3f s" % time_nb)
```

```
Naive Bayes: accuracy = 86.4 %
Naive Bayes: time = 0.742 s
```

[27]:
```
print("Naive Bayes Final Result")
result_nb.show()
```

```
Naive Bayes Final Result
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|  0.0|[7551.0,369.06399...|[-13675.340453705...|[0.99999089545080...|       0.0|
|  0.0|[7833.0,373.15701...|[-14124.785419430...|[0.99998658773609...|       0.0|
|  0.0|[8499.0,370.44601...|[-15060.469801154...|[0.99986962853963...|       0.0|
|  0.0|[8992.0,377.33200...|[-15797.596786382...|[0.99960509032412...|       0.0|
|  0.0|[9050.0,379.89898...|[-16048.047445041...|[0.99944252443073...|       0.0|
|  0.0|[9070.0,375.09399...|[-15938.569311382...|[0.99934303261366...|       0.0|
|  0.0|[9080.0,396.54000...|[-16160.710306313...|[0.99973909886354...|       0.0|
|  0.0|[9117.0,393.83801...|[-16145.932221638...|[0.99972281283282...|       0.0|
|  0.0|[9127.0,381.87899...|[-16036.396781564...|[0.99950690192799...|       0.0|
|  0.0|[9165.0,380.66000...|[-16130.206851103...|[0.99937783058370...|       0.0|
|  0.0|[9212.0,384.59600...|[-16242.683949417...|[0.99940796602718...|       0.0|
|  0.0|[9236.0,394.34100...|[-16320.736177592...|[0.99966716891679...|       0.0|
|  0.0|[9359.0,395.98800...|[-16500.216234872...|[0.99948089845218...|       0.0|
|  0.0|[9369.0,396.76000...|[-16535.140174160...|[0.99957270883879...|       0.0|
|  0.0|[9414.0,407.82501...|[-16642.451053650...|[0.99973831447673...|       0.0|
|  0.0|[9431.0,383.95901...|[-16491.149625173...|[0.99898629340137...|       0.0|
|  0.0|[9441.0,384.33999...|[-16515.536250611...|[0.99879237334021...|       0.0|
|  0.0|[9447.0,407.09201...|[-16732.769026692...|[0.99965168795222...|       0.0|
|  0.0|[9510.0,401.49200...|[-16796.607230431...|[0.99940214899284...|       0.0|
|  0.0|[9516.0,386.82101...|[-16703.256298768...|[0.99878427737382...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
only showing top 20 rows
```

```
[28]: print("Naive Bayes final result with name of class")
      labelReverse.transform(result_nb).show()

Naive Bayes final result with name of class
+-----+--------------------+--------------------+--------------------+----------+---------------------------------+
|label|            features|       rawPrediction|         probability|prediction|IndexToString_eb6637bff2fa__output|
+-----+--------------------+--------------------+--------------------+----------+---------------------------------+
|  0.0|[7551.0,369.06399...|[-13675.340453705...|[0.99999089545080...|       0.0|                         Osmancik|
|  0.0|[7833.0,373.15701...|[-14124.785419430...|[0.99998658773609...|       0.0|                         Osmancik|
|  0.0|[8499.0,370.44601...|[-15060.469801154...|[0.99986962853963...|       0.0|                         Osmancik|
|  0.0|[8992.0,377.33200...|[-15797.596786382...|[0.99960509032412...|       0.0|                         Osmancik|
|  0.0|[9050.0,379.89898...|[-16048.047445041...|[0.99944252443073...|       0.0|                         Osmancik|
|  0.0|[9070.0,375.09399...|[-15938.569311382...|[0.99934303261366...|       0.0|                         Osmancik|
|  0.0|[9080.0,396.54000...|[-16160.710306313...|[0.99973909886354...|       0.0|                         Osmancik|
|  0.0|[9117.0,393.83801...|[-16145.932221638...|[0.99972281283282...|       0.0|                         Osmancik|
|  0.0|[9127.0,381.87899...|[-16036.396781564...|[0.99950690192799...|       0.0|                         Osmancik|
|  0.0|[9165.0,380.66000...|[-16130.206851103...|[0.99937783058370...|       0.0|                         Osmancik|
|  0.0|[9212.0,384.59600...|[-16242.683949417...|[0.99940796602718...|       0.0|                         Osmancik|
|  0.0|[9236.0,394.34100...|[-16320.736177592...|[0.99966716891679...|       0.0|                         Osmancik|
|  0.0|[9359.0,395.98800...|[-16500.216234872...|[0.99948089845218...|       0.0|                         Osmancik|
|  0.0|[9369.0,396.76000...|[-16535.140174160...|[0.99957270883879...|       0.0|                         Osmancik|
|  0.0|[9414.0,407.82501...|[-16642.451053650...|[0.99973831447673...|       0.0|                         Osmancik|
|  0.0|[9431.0,383.95901...|[-16491.149625173...|[0.99898629340137...|       0.0|                         Osmancik|
|  0.0|[9441.0,384.33999...|[-16515.536250611...|[0.99879237334021...|       0.0|                         Osmancik|
|  0.0|[9447.0,407.09201...|[-16732.769026692...|[0.99965168795222...|       0.0|                         Osmancik|
|  0.0|[9510.0,401.49200...|[-16796.607230431...|[0.99940214899284...|       0.0|                         Osmancik|
|  0.0|[9516.0,386.82101...|[-16703.256298768...|[0.99878427737382...|       0.0|                         Osmancik|
+-----+--------------------+--------------------+--------------------+----------+---------------------------------+
only showing top 20 rows
```

Treinamento de um modelo de classificação Naive Bayes usando o algoritmo NaiveBayes do Spark ML, avaliando a acurácia do modelo e exibindo os resultados. Etapas realizadas pelo código:

1. O NaiveBayes é instanciado com dois parâmetros, smoothing e modelType. O parâmetro smoothing define o valor de suavização a ser aplicado durante a estimativa de probabilidade do modelo, enquanto o parâmetro modelType especifica o tipo de modelo a ser treinado (no exemplo fornecido, é utilizado o modelo multinomial).
2. O método fit é aplicado ao modelo, passando o DataFrame de treinamento (train), para treinar o modelo com base nos dados de treinamento.
3. O método transform é aplicado ao modelo treinado, passando o DataFrame de teste (test), para gerar as previsões para o conjunto de teste.
4. A mensagem "Naive Bayes: accuracy = ..." é impressa na tela, mostrando a acurácia do modelo Naive Bayes.
5. A mensagem "Naive Bayes: time = ..." é impressa na tela, mostrando o tempo de execução do modelo Naive Bayes.
6. A mensagem "Naive Bayes Final Result" é impressa na tela, mostrando o resultado final do modelo Naive Bayes.
7. O método transform é aplicado ao resultado do modelo (result_nb) junto com o labelReverse para exibir as previsões com os nomes das classes.

[29]:
```
start_time_svm =  time.time()

# create the trainer and set its parameters
trainer = LinearSVC(featuresCol='features', labelCol='label',\
                    maxIter=100, regParam=0.1)

# LinearSVC classify ONLY in two classes
# To classify in more than 2 classes, the OneVsrest should be used
# Cloud use any kind of classifies

# instantiate the One Vs Rest Classifier.
ovr_trainer = OneVsRest(classifier=trainer)

# train the multiclass model.
model = ovr_trainer.fit(train)

# score the model on test data.
result_svm = model.transform(test)
```

[30]:
```
# compute accuracy on the test set against model
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",\
            metricName="accuracy")

accuracy_svm = evaluator.evaluate(result_svm) * 100
time_svm = time.time() - start_time_svm

print("Suport Vector Machines (SVM): accuracy = %3.1f %%" % accuracy_svm)
print("Suport Vector Machines (SVM): time = %3.3f s" % time_svm)
```

```
[Stage 240:>                                                        (0 + 1) / 1]
Suport Vector Machines (SVM): accuracy = 92.2 %
Suport Vector Machines (SVM): time = 12.391 s
```

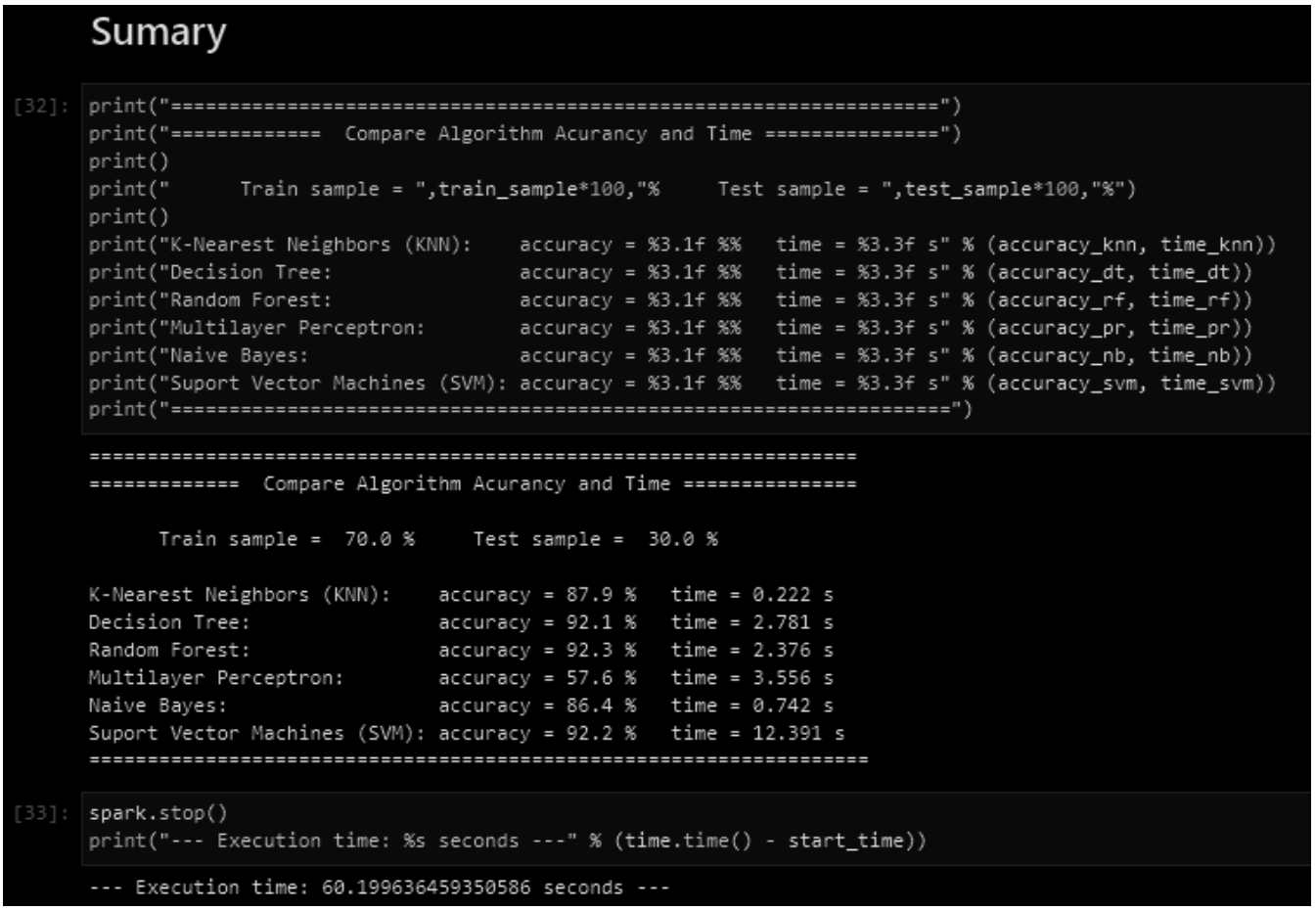
[31]:
```
print("Suport Vector Machines (SVM) Final Result")
result_svm.show()
```

```
Suport Vector Machines (SVM) Final Result
+-----+--------------------+--------------------+----------+
|label|            features|       rawPrediction|prediction|
+-----+--------------------+--------------------+----------+
|  0.0|[7551.0,369.06399...|[3.09802138949546...|       0.0|
|  0.0|[7833.0,373.15701...|[3.21167252160133...|       0.0|
|  0.0|[8499.0,370.44601...|[3.81531325002097...|       0.0|
|  0.0|[8992.0,377.33200...|[3.53597966404114...|       0.0|
|  0.0|[9050.0,379.89898...|[4.15131459829259...|       0.0|
|  0.0|[9070.0,375.09399...|[3.73887495294204...|       0.0|
|  0.0|[9080.0,396.54000...|[2.59775533485818...|       0.0|
|  0.0|[9117.0,393.83801...|[2.49773763721024...|       0.0|
|  0.0|[9127.0,381.87899...|[3.25173515357349...|       0.0|
|  0.0|[9165.0,380.66000...|[3.72431793972800...|       0.0|
|  0.0|[9212.0,384.59600...|[3.77003833226318...|       0.0|
|  0.0|[9236.0,394.34100...|[3.12984870460846...|       0.0|
|  0.0|[9359.0,395.98800...|[2.42668424650299...|       0.0|
|  0.0|[9369.0,396.76000...|[2.82213272794050...|       0.0|
|  0.0|[9414.0,407.82501...|[1.53736707184009...|       0.0|
|  0.0|[9431.0,383.95901...|[3.45758057957959...|       0.0|
|  0.0|[9441.0,384.33999...|[2.97550579325321...|       0.0|
|  0.0|[9447.0,407.09201...|[1.89592425733796...|       0.0|
|  0.0|[9510.0,401.49200...|[2.31890634782455...|       0.0|
|  0.0|[9516.0,386.82101...|[3.90429120672079...|       0.0|
+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

Treinamento de um modelo de classificação SVM (Support Vector Machines) usando o algoritmo LinearSVC do Spark ML, aplicação da abordagem One-vs-Rest para classificação multiclasse e avaliação da acurácia do modelo.
Etapas realizadas pelo código:

1. O LinearSVC é instanciado com dois parâmetros, featuresCol e labelCol, que especificam as colunas de recursos e rótulos, respectivamente.
2. O OneVsRest é instanciado, passando o LinearSVC como classificador interno.
3. O método fit é aplicado ao OneVsRest, passando o DataFrame de treinamento (train), para treinar o modelo SVM com base nos dados de treinamento usando a abordagem One-vs-Rest.
4. O método transform é aplicado ao modelo treinado, passando o DataFrame de teste (test), para gerar as previsões para o conjunto de teste.
5. A mensagem "Support Vector Machines (SVM): accuracy = ..." é impressa na tela, mostrando a acurácia do modelo SVM.
6. A mensagem "Support Vector Machines (SVM): time = ..." é impressa na tela, mostrando o tempo de execução do modelo SVM.
7. A mensagem "Support Vector Machines (SVM) Final Result" é impressa na tela, mostrando o resultado final do modelo SVM.

Conclusão



Sumary

```
[32]: print("================================================================")
      print("=============  Compare Algorithm Acurancy and Time ===============")
      print()
      print("      Train sample = ",train_sample*100,"%     Test sample = ",test_sample*100,"%")
      print()
      print("K-Nearest Neighbors (KNN):    accuracy = %3.1f %%   time = %3.3f s" % (accuracy_knn, time_knn))
      print("Decision Tree:                accuracy = %3.1f %%   time = %3.3f s" % (accuracy_dt, time_dt))
      print("Random Forest:                accuracy = %3.1f %%   time = %3.3f s" % (accuracy_rf, time_rf))
      print("Multilayer Perceptron:        accuracy = %3.1f %%   time = %3.3f s" % (accuracy_pr, time_pr))
      print("Naive Bayes:                  accuracy = %3.1f %%   time = %3.3f s" % (accuracy_nb, time_nb))
      print("Suport Vector Machines (SVM): accuracy = %3.1f %%   time = %3.3f s" % (accuracy_svm, time_svm))
      print("================================================================")

      ================================================================
      =============  Compare Algorithm Acurancy and Time ===============

            Train sample =  70.0 %     Test sample =  30.0 %

      K-Nearest Neighbors (KNN):    accuracy = 87.9 %   time = 0.222 s
      Decision Tree:                accuracy = 92.1 %   time = 2.781 s
      Random Forest:                accuracy = 92.3 %   time = 2.376 s
      Multilayer Perceptron:        accuracy = 57.6 %   time = 3.556 s
      Naive Bayes:                  accuracy = 86.4 %   time = 0.742 s
      Suport Vector Machines (SVM): accuracy = 92.2 %   time = 12.391 s
      ================================================================

[33]: spark.stop()
      print("--- Execution time: %s seconds ---" % (time.time() - start_time))

      --- Execution time: 60.199636459350586 seconds ---
```

Após observação dos resultados finais apresentados no sumário, pode-se concluir que, para os parâmetros utilizados, os melhores resultados foram apresentados utilizando Random Forest.

Vale ressaltar que não foram realizadas otimizações nos parâmetros, portanto é possível que se realizadas otimizações em todas as técnicas, o resultado final da melhor técnica pode ser diferente do apresentado.

A realização deste trabalho foi crucial para consolidar os conhecimentos adquiridos na disciplina. A prática desempenhou um papel fundamental na descoberta e aprofundamento de novos conceitos, assim como na sua aplicação.