LINEAR SUPPORT VECTOR MACHINE

Import necessary libraries

Initialize the spark as the folder that installed spark in local disk, then create a SparkSession to run Spark in Jupiter notebook.

```
import findspark
findspark.init("D:\spark\spark-3.1.1-bin-hadoop3.2")
import pyspark
from pyspark.sql.functions import col, when
from pyspark.sql.types import IntegerType
from pyspark.sql import SparkSession
from pyspark.ml.classification import LinearSVC
from pyspark.sql.functions import collect_list
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.sql.functions import udf
from pyspark.ml.feature import VectorAssembler


spark = SparkSession.builder.getOrCreate()
```

Load data

❖ Introduction about dataset:

- ✓ Description: [2.2] "The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset."
- ✓ The dataset has 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa. And 1460 records in this file.

❖ Reference.

- ✓ Source: Kaggle Website.
- ✓ Author: **Dean De Cock.**
- ✓ Title: "House Prices - Advanced Regression TechniquesPredict sales prices and practice feature engineering, RFs, and gradient boosting".
- ✓ Link: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview
- ✓ Date accessed: 25.4.2021.

Firstly, we load the data from file "train.txt" to the spark data frame as *rawdata* variable using *spark.read.load* function pass the name of file "train.txt", its format "csv" and header equals True, delimiter is "," each value in row seperated.

```
rawdata = spark.read.load("train.csv", format="csv", header=True,
delimiter=",")
```

Then, we set the display mode in spark, display the dataframe into the table and print the data read from file on the screen.

```
spark.conf.set('spark.sql.repl.eagerEval.enabled', True)

rawdata
```

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1
1	60	RL	65	8450	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
2	20	RL	80	9600	Pave	NA	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr
3	60	RL	68	11250	Pave	NA	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
4	70	RL	60	9550	Pave	NA	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm
5	60	RL	84	14260	Pave	NA	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm
6	50	RL	85	14115	Pave	NA	IR1	Lvl	AllPub	Inside	Gtl	Mitchel	Norm
7	20	RL	75	10084	Pave	NA	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm

Figure 2.6: A part of input dataframe read from mentioned file.

❖ Describe the purpose of using linear SVM algorithm in this dataset.

- We only consider 3 features for analyzing the house price including:
  - OverallQual: Rates the overall material and finish of the house.
    - 10 Very Excellent
    - 9 Excellent
    - 8 Very Good
    - 7 Good
    - 6 Above Average
    - 5 Average
    - 4 Below Average
    - 3 Fair
    - 2 Poor
    - 1 Very Poor
  - OverallCond: Rates the overall condition of the house, which has the same measuring as OverallQual
  - TotalBsmtSF: The Type 2 finished square feet.
- In this task, we analyze the mentioned feature to predict or classify a house into 2 groups:
  - Group 1: House price is less than 150K: SalePrice < 1500000
  - Group 2: House price is higher or equal 150K: SalePrice >= 1500000
- To do this one, we need to preprocess the SalePrice into binary value, 0 if SalePrice less than 150k, 1 in the remaining case.

Linear Support Vector Machine algorithm

Step 1: Extract the data with 3 columns: OverallQual, Sale Price, TotalBsmtSF, OverallCond. Then convert its datatype from string to integer.

```
df = rawdata.select(rawdata.SalePrice, rawdata.OverallQual,
rawdata.OverallCond,rawdata.TotalBsmtSF)
df = df.withColumn("SalePrice",col("SalePrice").cast(IntegerType())) \
    .withColumn("OverallQual",col("OverallQual").cast(IntegerType()))\
    .withColumn("TotalBsmtSF",col("TotalBsmtSF").cast(IntegerType()))\
    .withColumn("OverallCond",col("OverallCond").cast(IntegerType()))
df
```

SalePrice	OverallQual	OverallCond	TotalBsmtSF
208500	7	5	856
181500	6	8	1262
223500	7	5	920
140000	7	5	756
250000	8	5	1145
143000	5	5	796
307000	8	5	1686
200000	7	6	1107
129900	7	5	952
118000	5	6	991
129500	5	5	1040
345000	9	5	1175
144000	5	6	912

Figure 2.7: A part of result after selecting the neccesary columns from dataframe.

Then we print the data's Schema to validate the converting step.

```
df.printSchema()
```

```
root
 |-- SalePrice: integer (nullable = true)
 |-- OverallQual: integer (nullable = true)
 |-- OverallCond: integer (nullable = true)
 |-- TotalBsmtSF: integer (nullable = true)
```

Figure 2.8: This result proved that we successful to covert the datatype of those columns into interger datatype.

Next to, we convert the values of SalePrice column to binary values as decribed above.

```
df = df.withColumn("SalePrice", when(df.SalePrice < 150000, 0).otherwise(1))
```

Then we collect the OverallQual and OverallCond and TotalBsmtSF into an array with 3 elements as the new column Features. Using assembler to convert the data into the vector.

```
assembler = VectorAssembler(inputCols=["OverallQual", "OverallCond",
"TotalBsmtSF"], outputCol="Features")

df = assembler.transform(df)

df = df.select(df.SalePrice, df.Features)

df
```

SalePrice	Features
1	[7.0,5.0,856.0]
1	[6.0,8.0,1262.0]
1	[7.0,5.0,920.0]
0	[7.0,5.0,756.0]
1	[8.0,5.0,1145.0]
0	[5.0,5.0,796.0]
1	[8.0,5.0,1686.0]

Figure 2.8: A part of the dataframe after converting SalePrice values to binary values and collecting the Overall Condition and Overall Quality into Features column.

To make the attribute Feature for the model, we must convert the column Features to vector dense datatype.

```
to_vector = udf(lambda a: Vectors.dense(a), VectorUDT())
data = df.select("SalePrice", to_vector("Features").alias("Feature"))
# Verify the converting step
data.printSchema()
```

```
root
 |-- SalePrice: integer (nullable = false)
 |-- Feature: vector (nullable = true)
```

Figure 2.9: A part of the dataframe after transforming steps

Finally, we ppit the data to 70% data for training into *train* variable and 30% data for testing into *test* variable.

```
(train, test) = data.randomSplit([0.7, 0.3])
```

Step 2: Using Linear support vector machine to build the model

Firstly, we create the trainer of Linear Support Vector Machine by function LinearSVC then pass the arguments the number of training epochs, the label column is SalePrice, the Features column is Feature. Then saved into linearSvm class.

```
linearSvm = LinearSVC(maxIter=20, regParam=0.1, labelCol="SalePrice",
featuresCol="Feature")
```

Then, we fit the training data into the model by using the fit function of linearSvm.

```
linearModel = linearSvm.fit(train)
```

Step 3: Evaluating the model.

We predict on the test sets. The output of the model saved into the prediction column and the truth label is saved into the SalePrice column.

```
prediction = linearModel.transform(test)
prediction
```

SalePrice	Feature	rawPrediction	prediction
0	[1.0,3.0,0.0]	[1.42080874447673...	0.0
0	[2.0,3.0,264.0]	[0.80673450487308...	0.0
0	[2.0,3.0,480.0]	[0.68578385827188...	0.0
0	[3.0,3.0,864.0]	[0.00451481500091...	0.0
0	[3.0,3.0,864.0]	[0.00451481500091...	0.0
0	[3.0,4.0,0.0]	[0.97854653126409...	0.0
0	[3.0,4.0,894.0]	[0.47794524394246...	0.0
0	[3.0,5.0,978.0]	[0.92113801123370...	0.0
0	[3.0,6.0,0.0]	[1.95900479098085...	0.0
0	[3.0,6.0,544.0]	[1.65438834768894...	0.0
0	[4.0,2.0,1095.0]	[-1.0813099834525...	1.0
0	[4.0,4.0,715.0]	[0.11193182121072...	0.0

Figure 2.10: The predicting result of the model in comparision with the truth label.

Next to, we Calculate the True Positive, True Negative, False Positive, False Negative by group by and filter to count those values from the confusion matrix prediction. And print it into screen.

```
TN = prediction.filter('prediction = 0 AND SalePrice = prediction').count()
TP = prediction.filter('prediction = 1 AND SalePrice = prediction').count()
FN = prediction.filter('prediction = 0 AND SalePrice <> prediction').count()
```

```
FP = prediction.filter('prediction = 1 AND SalePrice <> prediction').count()
print(TP, TN)
print(FP, FN)
```

```
The output of that step is:
      TP: 196
      TN: 119
      FP: 53
      FN: 50
```

Or we can use the group by function to group by the SalePrice and prediction, then count it into the count column -> Finally, show confusion matrix.

```
prediction.groupBy('SalePrice', 'prediction').count().show()
```

```
+---------+----------+-----+
|SalePrice|prediction|count|
+---------+----------+-----+
|        1|       0.0|   50|
|        0|       0.0|  119|
|        1|       1.0|  196|
|        0|       1.0|   53|
+---------+----------+-----+
```

Figure 2.11: The result of group by function to visualize the confusion matrix.

Finally, we caclulate the Accuracy, Precision, Recall, F1 Score by the confusion matrix to evaluating our model.

- Accuracy: It measures the exact probability that the dataset correctly predicted by the model.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$

- Precision: mesure the exactness, which percentage of samples are correctly predicted as positive in reality

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall: mesure the completeness, which percentage of positive samples are predicted.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}}$$

- F1 Score: is the harmonic mean of precision and recall.

$$\text{F1 Score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

```
Evaluation model by calculate metrics by the confusion matrix
Accuracy: 0.754
Precision: 0.787
Recall: 0.797
F1 Score: 0.792
```

Figure 2.12: The result of evaluating Linear SVM model.

REFERENCE

[1] Kaggle Website - House Prices - Advanced Regression Techniques – Author: Dean De Cock – Date accessed: 25/4/2021.

Link: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview