EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México
Instituto Tecnológico de Tijuana

ACADEMIC SUBDIRECTION
Systems and Computing Department

SEMESTER
August - December 2021

CAREER
Information and Communication Technologies Engineer

SUBJECT AND KEY:
Big Data BDD-1704TI9A

STUDENT'S NAME AND REGISTRATION:

Castillo Ramirez Guadalupe 17213043
Velázquez Farrera César Alejandro 17212937

NAME OF THE JOB:
Practice # 6 -Linear Support Vector Machine

UNIT TO BE EVALUATED
Unit II

TEACHER'S NAME:
M.C. Jose Christian Romero Hernandez

Introduction

The practice will be developed according to the Multilayer Perceptron classifier theme, which will be developed in scala, which was taken as an example from the apache spark document.

Linear Support Vector Machine

A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. LinearSVC in Spark ML supports binary classification with linear SVM. Internally, it optimizes the Hinge Loss using OWLQN optimizer.

Library: The LinearSVC library is imported, which is the one that will help us create the object that will help to classify the data, and that allows access to the methods contained in it.

```
import org.apache.spark.ml.classification.LinearSVC
```

Load training data. The data that will be used to train the model is loaded, this is done by specifying the type of format in which the data comes, and then, through load, the directory where the data file is located is placed. This is going to be saved in the training variable, since SVM needs a set of data to be able to determine the separation hyperplane, and in this way determine if a future data is part of a certain class or another.

```
val training = spark.read.format("libsvm").load("../sample_libsvm_data.txt")
```

setMaxIter: Set the maximum number of iterations and setRegParam: Set the regularization parameter. Here the LinearSVC object is created, which is where the classification will be performed, the setMaxIter method determines the maximum number of iterations to perform and setRegParam serves to establish a regulation in the entered data.

```
val lsvc = new LinearSVC().setMaxIter(10).setRegParam(0.1)
```

Fit the model and fits a model to the input data. Once the model has been created, now the data saved in the training variable are passed, in which the model must be adjusted to the data, this so that they can be classified based on these and obtain the coefficients that will be shown in the next step.

```
val lsvcModel = lsvc.fit(training)
```

Print the coefficients and intercept for linear svc. At the end, the coefficients that gave as results when going through the SVM model are shown, which is how they are classified in order to determine if it belongs to one or another class within SVM, as well as the intercept line that resulted when classifying the data.

```
println(s"Coefficients: ${lsvcModel.coefficients} Intercept: ${lsvcModel.intercept}")
```

Result:

```
lsvcModel: org.apache.spark.ml.classification.LinearSVCModel =
linearsvc_8ea15c77724c
Coefficients: [0.0,0.0,0.0,0.0,0.0,0.0,0…….]
```

Example of Titanic with LinearSVC
Libraries:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.classification.LinearSVC
import org.apache.spark.ml.feature.VectorAssembler
```

Read the titanic dataset

```
val spark = SparkSession.builder().getOrCreate()
val titanic_df = spark.read.format("csv").option("inferSchema",
"true").option("header", "true").load("test.csv")
titanic_df.show()
```

```
titanic_df: org.apache.spark.sql.DataFrame = [PassengerId: int, Survived: int ... 11
more fields]
```

Delete unnecessary columns

```
val data =
titanic_df.drop("PassengerId","Name","Ticket","Cabin","Embarked","Sex","SibSp","
Parch")
data.show()
```

Result:

```
data: org.apache.spark.sql.DataFrame = [Survived: int, Pclass: int ... 3 more fields]
```

We use VectorAssembler to merge the multi-column features into one vector column

```
val feature = new
VectorAssembler().setInputCols(Array("Pclass","Age","Fare","Sex_index")).setOutp
utCol("features")
val feature_vector= feature.transform(data)
feature_vector.select("Survived","Pclass","Age","Fare","Sex_index","features").sho
w()
```

Result:

```
feature_vector: org.apache.spark.sql.DataFrame = [Survived: int, Pclass: int ... 4
more fields]
```

The data is prepared for training and the test

```
val Array(trainingData, testData) = feature_vector.randomSplit(Array(0.7, 0.3))
```

Fits the model to the input data

```
val lnsvc = new LinearSVC().setLabelCol("Survived").setFeaturesCol("features")
val lnsvc_model = lnsvc.fit(trainingData)
```

Prediction is made with the test data

```
val lnsvc_prediction = lnsvc_model.transform(testData)
lnsvc_prediction.select("prediction", "Survived", "features").show()
```

Result:

```
lnsvc_prediction: org.apache.spark.sql.DataFrame = [Survived: int, Pclass: int ... 6
more fields]
```

Select (prediction, the value to compare and the accuracy) and compute test error.

```
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("Survived").setPredictionCol("predi
ction").setMetricName("accuracy")
val lnsvc_accuracy = evaluator.evaluate(lnsvc_prediction)
print("Accuracy of Support Vector Machine is = " + (lnsvc_accuracy))
print(" and Test Error of Support Vector Machine = " + (1.0 - lnsvc_accuracy))
```

Result:

```
Accuracy of Support Vector Machine is = 0.8063380281690141 and Test Error of
Support Vector Machine = 0.19366197183098588
```