Machine Learning With Apache Spark

By:
Nikhilesh Vashisht

Table Of Contents

Machine Learning with Apache Spark:

Machine learning is making the computer learn by studying data and statistics. Machine Learning helps solve problems by analyzing the data and learning to predict the outcome. Instead of addressing the challenges posed by dispersed data, we can concentrate on the data issues and models thanks to Apache Spark machine learning such as infrastructure, configurations, and so on.

1: PREDICT DIABETES

Predict whether a person is at risk of becoming diabetic based on the individual's data. Before analyzing it is very important to understand what we are going to do.

Type 2 Diabetes is caused when there is not enough insulin generated by the pancreas.

Data Dictionary:

Field Name	Definition	Type
Pregnancies	Number of times pregnant	Feature
Glucose	Plasma glucose concentration, 2 hours in an oral glucose tolerance test	Feature
BloodPressure	Diastolic blood pressure (mm Hg)	Feature
SkinThickness	Triceps skin fold thickness (mm)	Feature
Insulin	2-Hour serum insulin (mu U/ml)	Feature
BMI	Body mass index (weight in kg/(height in m)^2)	Feature
DiabetesPedigreeFunction	Diabetes pedigree function, a score based on a genetic factor of a person (diabetes has a close relation to family history)	Feature
Age	Age (years)	Feature
Outcome	Outcome whether the patient has diabetes or not	Label

Selecting Features:

The major features in our dataset which cause diabetes are:

```
Correlation between Pregnancies and Outcome: 0.22189815303398636
Correlation between Insulin and Outcome: 0.13054795488404794
Correlation between Glucose and Outcome: 0.4665813983068737
Correlation between BloodPressure and Outcome: 0.065068359550332 74
Correlation between SkinThickness and Outcome: 0.07475223191831945
Correlation between BMI and Outcome: 0.2926946626444454
Correlation between DiabetesPedigreeFunction and Outcome: 0.17384406565296
Correlation between Age and Outcome: 0.23835598302719757
```

Since there's a weak correlation between SkinThickness, BloodPressure, and Outcome, these features can be neglected. Even though there is a weak correlation between insulin and outcome, we have to consider it as it is one of the major factors of type 2 diabetes.

Model Selection:

I have chosen Random Forest. Random Forest algorithm avoids overfitting by using multiple trees.

Hyper Parameters

Parameters of an algorithm that may be tweaked to improve performance. The optimum hyperparameters are typically hard to predict in advance, and tweaking a model is when machine learning transitions from science to trial-and-error engineering. Because hyperparameter tuning is based on experimental findings rather than theory, the easiest way to identify the ideal settings is to test many different combinations and evaluate the performance of each model.

maxDepth:

Max depth can be defines as the longest path between the root node and the leaf node or max number of levels in each decision tree.

maxBins:

Max bins is a decision tree parameter. Number of bins set while continuous features are made separate or discreet. The default value of maxBin in spark is 32. and the value needs to be set >=2

```
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 32)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 33)
accuracy of the model is :77.53623188405797
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 34)
accuracy of the model is :74.27536231884058
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 35)
accuracy of the model is :74.63768115942028
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 32)
accuracy of the model is :77.17391304347827
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 33)
accuracy of the model is :76.08695652173914
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 34)
accuracy of the model is :76.81159420289855
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 35)
accuracy of the model is :76.44927536231883
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 32)
accuracy of the model is :75.72463768115942
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 33)
accuracy of the model is :76.08695652173914
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 34)
accuracy of the model is :75.0
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 35)
accuracy of the model is :77.17391304347827
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 32)
accuracy of the model is :75.0
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 33)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 34)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 35)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 32)
accuracy of the model is :76.44927536231883
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 33)
accuracy of the model is :74.63768115942028
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 34)
accuracy of the model is :73.55072463768117
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 35)
accuracy of the model is :74.63768115942028
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 32)
accuracy of the model is :73.18840579710145
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 33)
accuracy of the model is :75.0
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 34)
accuracy of the model is :73.18840579710145
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 35)
accuracy of the model is :73.55072463768117
```

As we can see in the results above which parameters are yielding better results. When the maxdepth is 3,4 and 6. And when the maxBin is 32,33,34 & 35.

Accuracy:

On top of this, I wanted to check the accuracies of each submodel in the cross-validator. So, I had passed sedcollectSubModels to true, and ran it through a nested for loop to check the accuracies.

```
scala> for(models <- subModels_nikhilesh)
     |  for(s <- models) {val each_model = s.transform(testData)
     |  val accuracy_nikhilesh_subModel = evaluator_nikhilesh.evaluate(each_model)
     |  println("accuracy of the model is :"+(accuracy_nikhilesh_subModel*100))}
accuracy of the model is :72.46376811594203
accuracy of the model is :72.46376811594203
accuracy of the model is :71.73913043478261
accuracy of the model is :75.0
accuracy of the model is :72.82608695652173
accuracy of the model is :75.36231884057972
accuracy of the model is :74.63768115942028
accuracy of the model is :75.36231884057972
accuracy of the model is :76.81159420289855
accuracy of the model is :76.81159420289855
accuracy of the model is :73.91304347826086
accuracy of the model is :76.44927536231883
accuracy of the model is :73.91304347826086
accuracy of the model is :76.44927536231883
accuracy of the model is :74.27536231884058
accuracy of the model is :75.36231884057972
accuracy of the model is :75.36231884057972
accuracy of the model is :76.08695652173914
accuracy of the model is :76.81159420289855
accuracy of the model is :77.17391304347827
accuracy of the model is :76.08695652173914
accuracy of the model is :75.0
accuracy of the model is :77.89855072463769
accuracy of the model is :75.36231884057972
```

The cross validator choses the model which yields the best accuracy or result, and in this case the highest accuracy is chosen as the best model.
The highest accuracy in which this model can predict whether the person has diabetes or not is: **77.174%**

2: PREDICT CANCER

Cancer is a condition in which some cells in the body develop uncontrollably and spread to other parts of the body. In the millions of cells that make up the human body, cancer can develop practically anywhere. Human cells often divide to create new cells as the body requires them. New cells replace old ones when they die as a result of ageing or damage. Malignant (cancerous) or benign (non-cancerous) tumours can both occur in the body. It is typical for non-cancerous tumours to grow slowly and not spread. Cancerous tumours have the ability to spread throughout the body, grow quickly, infiltrate neighbouring normal tissues, and do great damage. Here we are predicting whether a person's tumour is cancerous in order to decide whether surgery is necessary or not

Data Dictionary:

Field Name	Value Range	Type
ID	Sample Code Number	ID
Clump Thickness	1-10	Feature
Uniformity of Cell Size	1-10	Feature
Uniformity of Cell Shape	1-10	Feature
Single Epithelial Cell Size	1-10	Feature
Bare Nuclei	1-10	Feature
Bland Chromatin	1-10	Feature
Normal Nucleol	1-10	Feature
Mitoses	1-10	Feature
Class	2 for benign, 4 for malignant	Label

Selecting features:

Since id is a just a sample code number, we can ignore it and consider all other features. All the other features have no null values. Are within the range fo 1-10 and the class variable which is our predictor variable has binomial values either 2 or 4.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

println("Correlation between UofCSize and Class: "+df_cacner_nikhilesh.stat.corr("UofCSize", "Class", "pear
son").toString
 +"\nCorrelation between UofCShape and Class: "+df_cacner_nikhilesh.stat.corr("UofCShape", "Class", "pearso
n").toString+
 "\nCorrelation between Marginal Adhesion and Class: "+df_cacner_nikhilesh.stat.corr("Marginal Adhesion", "
Class", "pearson").toString+
 "\nCorrelation between SECSize and Class: "+df_cacner_nikhilesh.stat.corr("SECSize", "Class", "pearson").t
oString+
 "\nCorrelation between Bare Nuclei and Class: "+df_cacner_nikhilesh.stat.corr("Bare Nuclei", "Class", "pea
rson").toString+
 "\nCorrelation between Mitoses and Class: "+df_cacner_nikhilesh.stat.corr("Mitoses", "Class", "pearson").t
oString+
 "\nCorrelation between Normal Nucleoli and Class: "+df_cacner_nikhilesh.stat.corr("Normal Nucleoli", "Clas
s", "pearson").toString)

// Exiting paste mode, now interpreting.

Correlation between UofCSize and Class: 0.8208014428258776
Correlation between UofCShape and Class: 0.821890947688868
Correlation between Marginal Adhesion and Class: 0.7062941354660839
Correlation between SECSize and Class: 0.6909581590873204
Correlation between Bare Nuclei and Class: 0.8226958729964622
Correlation between Mitoses and Class: 0.4234479212952124
Correlation between Normal Nucleoli and Class: 0.7186771878756356

scala>
```

None of the correlations is weak. Hence I have considered all the features except id

Model Selection

I have chosen Logistic Regression. Logistic Regression is usually used in binary classification problems. It is a supervised machine-learning approach that can be used to model the likelihood of a given class or occurrence. According to Kaggle, Logistic Regression is the most used classification algorithm. I have tried Naive-Bayes and Random Forest but Logistic Regression gave the best result.

Hyper Parameters

Parameters of an algorithm that may be tweaked to improve performance. The optimum hyperparameters are typically hard to predict in advance, and tweaking a model is when machine learning transitions from science to trial-and-error engineering. Because hyperparameter tuning is based on experimental findings rather than theory, the easiest way to identify the ideal settings is to test many different combinations and evaluate the performance of each model.

Tolerance:

The dual gap is tested for optimality and the optimization code continues until it is smaller than tolerance only if the updates are smaller than tolerance.

MaxIter:

The most iterations necessary for the solvers to coincide.

RegParam:

RegParam represents the compromise between the two objectives of minimising training error and minimising the model to prevent overfitting.

```
Vector(R, e, g, P, a, r, a, m, :,   , 0.1 MaxIter: 10)
best accuracy of the model is :99.42942942942943
Vector(R, e, g, P, a, r, a, m, :,   , 0.1 MaxIter: 20)
best accuracy of the model is :99.96996996996998
Vector(R, e, g, P, a, r, a, m, :,   , 0.1 MaxIter: 30)
best accuracy of the model is :99.97997997997997
Vector(R, e, g, P, a, r, a, m, :,   , 0.1 MaxIter: 40)
best accuracy of the model is :99.97997997997997
Vector(R, e, g, P, a, r, a, m, :,   , 0.2 MaxIter: 10)
best accuracy of the model is :99.64964964964965
Vector(R, e, g, P, a, r, a, m, :,   , 0.2 MaxIter: 20)
best accuracy of the model is :99.91991991991992
Vector(R, e, g, P, a, r, a, m, :,   , 0.2 MaxIter: 30)
best accuracy of the model is :99.94994994994995
Vector(R, e, g, P, a, r, a, m, :,   , 0.2 MaxIter: 40)
best accuracy of the model is :99.94994994994995
Vector(R, e, g, P, a, r, a, m, :,   , 0.3 MaxIter: 10)
best accuracy of the model is :99.70970970970971
Vector(R, e, g, P, a, r, a, m, :,   , 0.3 MaxIter: 20)
best accuracy of the model is :99.92992992992994
Vector(R, e, g, P, a, r, a, m, :,   , 0.3 MaxIter: 30)
best accuracy of the model is :99.92992992992994
Vector(R, e, g, P, a, r, a, m, :,   , 0.3 MaxIter: 40)
best accuracy of the model is :99.92992992992994
```

As we can see with all the combinations of RegParam and MaxIter, the model was still overfitting in some way or the other contributing to more than 99% accuracy when we were using BinaryClassificationEvaluator. To avoid overfitting, I changed the evaluator function to MultiClassificationEvaluator and the metric as “accuracy”. This resulted in avoiding overfitting.

```
res7: evaluator_nikhilesh.type = MulticlassClassificationEvaluator: uid=mcEval_7b87a8a40f71, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setPredictionCol("prediction")
res8: res7.type = MulticlassClassificationEvaluator: uid=mcEval_7b87a8a40f71, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setMetricName("accuracy")
res9: res8.type = MulticlassClassificationEvaluator: uid=mcEval_7b87a8a40f71, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala> :paste
// Entering paste mode (ctrl-D to finish)

for (RegParam1 <- RegParam_Array)
  for(MaxIter1 <- MaxIter_Array)
  {
val paramGrid_nikhilesh = new ParamGridBuilder().addGrid(logistic_nikhilesh.regParam, Array(RegParam1))
  .addGrid(logistic_nikhilesh.elasticNetParam, ElasticNetParam1)
  .addGrid(logistic_nikhilesh.tol, tolerance)
  .addGrid(logistic_nikhilesh.maxIter, Array(MaxIter1)).build()
val cross_validator_nikhilesh = new CrossValidator()
.setEstimatorParamMaps(paramGrid_nikhilesh)
  .setEstimator(pipeline_nikhilesh)
  .setEvaluator(evaluator_nikhilesh)
  .setNumFolds(5)
val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("RegParam: ":+RegParam1+" MaxIter: "+MaxIter1.toString)
println("best accuracy of the model is :"+(accuracy_nikhilesh*100))
}

// Exiting paste mode, now interpreting.

Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 10)
best accuracy of the model is :94.25837320574163
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 20)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 30)
best accuracy of the model is :99.04306220095694
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 40)
best accuracy of the model is :99.04306220095694
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 10)
best accuracy of the model is :98.08612440191388
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 20)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 30)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 40)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 10)
best accuracy of the model is :97.1291866028708
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 20)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 30)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 40)
best accuracy of the model is :98.56459330143541
```

After multiple trial and error methods, in order to avoid overfitting and minimizing the loss. I have taken RegParam as 0.1 and MaxIter as 10.

Accuracy:

BinaryClassficationEvaluator was overfitting the model. So I have chosen MultiClassEvaluator as my evaluator function. The cross validator chooses the model which yields the best accuracy or result, and in this case, the highest accuracy is chosen as the best model.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val paramGrid_nikhilesh = new ParamGridBuilder().addGrid(logistic_nikhilesh.regParam, Array(0.1))
  .addGrid(logistic_nikhilesh.elasticNetParam, ElasticNetParam1)
  .addGrid(logistic_nikhilesh.tol, tolerance)
  .addGrid(logistic_nikhilesh.maxIter, Array(10)).build()
val cross_validator_nikhilesh = new CrossValidator()
.setEstimatorParamMaps(paramGrid_nikhilesh)
  .setEstimator(pipeline_nikhilesh)
  .setEvaluator(evaluator_nikhilesh)
  .setNumFolds(5)
val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("best accuracy of the model is :"+(accuracy_nikhilesh*100))

// Exiting paste mode, now interpreting.

best accuracy of the model is :94.25837320574163
paramGrid_nikhilesh: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        logreg_43214f6bc7e4-elasticNetParam: 0.001,
        logreg_43214f6bc7e4-maxIter: 10,
        logreg_43214f6bc7e4-regParam: 0.1,
        logreg_43214f6bc7e4-tol: 1.0E-6
})
cross_validator_nikhilesh: org.apache.spark.ml.tuning.CrossValidator = cv_560ef2eddbe5
cvModel_nikhilesh: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_560ef2eddbe5, bestModel=pipeline_65d26d7eeba9, numFolds=5
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [id: int, Clump Thickness: int ... 14 more fields]
accuracy_nikhilesh: Double = 0.9425837320574163
```

The highest accuracy with which this model can predict whether the person has diabetes or not is: **94.258%**

3: KAGGLE COMPETITION -1 QUORA QUESTION PAIRS

Link: https://www.kaggle.com/c/quora-question-pairs
“Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.”
The goal of this competition is to predict which of the provided pairs of questions contain two questions with the same meaning.

Data Dictionary:

Field Name	Deinition	Type
ID	The id of a training set question pair	ID
qid1	Unique id of question1	ID
qid2	Unique id of question2	ID
question1	First Question	Feature
question2	Second Question	Feature
is_duplicate	Whether the two questions are duplicate or not	Label

Feature Engineering:

- Since id,qid1 and qid2 are unique identification numbers, these doesnt hold any value over the outcome.
- In the question1 and question2, there were lot of that there was a lot of special characters like double quotes, so in order to remove them from the features, I have used the regular expression **“[^a-zA-Z0-9]/g"** to remove the special characters.
- The “check” function in the code is used to remove the accent from the words. To do so I have used **stringAccents** in the **StringUtils** class.
- The first step of the NLP process is gathering the data and breaking it into understandable parts. Tokenizer splits paragraphs and sentences into smaller units that can be more easily assigned meaning.
- The StopWordsRemover function removes all stop words from input sequences of strings (such as the result of a tokenizer). The stopWords option specifies the list of

stopwords.Calling the StopWordsRemover will provide access to some languages' default stop words. The possible possibilities for DefaultStopWords(language) are "danish," "dutch," "english," "finnish," "french," "german," "hungarian," "italian," "norwegian," "portuguese," "russian," "spanish," "swedish," and "turkish." a case for boolean parameters If the matches should be case sensitive, sensitive lets you know that.

- Further, had to check how many of the words are the same in both the sentences. For this, the function “**getRatio_bw_question**” is coded.

Model Selection

To predict whether the 2 questions convey the same meaning, I have chosen Random Forest.

Hyper Parameters

Parameters of an algorithm that may be tweaked to improve performance. The optimum hyperparameters are typically hard to predict in advance, and tweaking a model is when machine learning transitions from science to trial-and-error engineering. Because hyperparameter tuning is based on experimental findings rather than theory, the easiest way to identify the ideal settings is to test many different combinations and evaluate the performance of each model.

maxDepth:

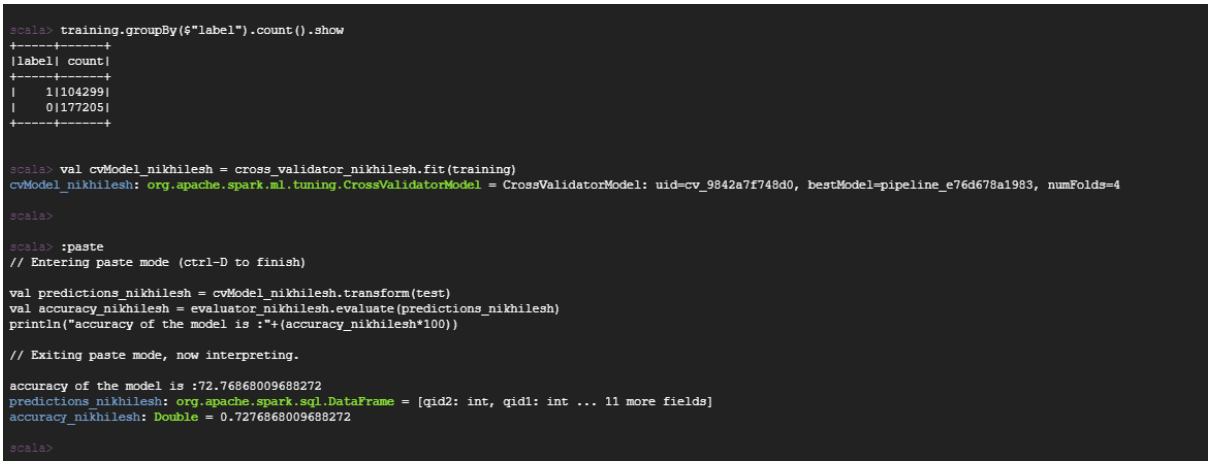
Max depth can be defined as the longest path between the root node and the leaf node or max number of levels in each decision tree.

maxBins:

Max bins is a decision tree parameter. A number of bins are set while continuous features are made separate or discreet. The default value of maxBin in spark is 32. and the value needs to be set >=2
After multiple trials and error methods, in order to avoid overfitting and minimize the loss. I have taken maxdepth is 3,4 and 6. And when the maxBin is 32,33,34 & 35.

Accuracy:

BinaryClassficationEvaluator was overfitting the model. So I have chosen MultiClassEvaluator as my evaluator function. The cross-validator chooses the model which yields the best accuracy or result, and in this case, the highest accuracy is chosen as the best model.



```
scala> training.groupBy($"label").count().show
+-----+------+
|label| count|
+-----+------+
|    1|104299|
|    0|177205|
+-----+------+


scala> val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
cvModel_nikhilesh: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_9842a7f748d0, bestModel=pipeline_e76d678a1983, numFolds=4

scala>

scala> :paste
// Entering paste mode (ctrl-D to finish)

val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("accuracy of the model is :"+(accuracy_nikhilesh*100))

// Exiting paste mode, now interpreting.

accuracy of the model is :72.76868009688272
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [qid2: int, qid1: int ... 11 more fields]
accuracy_nikhilesh: Double = 0.7276868009688272

scala>
```

The highest accuracy with which this model can predict whether the pair of questions are duplicates or not is: **72.76%**

4: KAGGLE COMPETITION - 2: TWEET SENTIMENT EXTRACTION

Source: https://www.kaggle.com/competitions/tweet-sentiment-extraction
“The competition is about predicting the word or phrase from the tweet that exemplifies the provided sentiment into 3 categories Positive, Negative & Neutral.”
The goal of this competition is to predict the sentiment of the tweet into any of the three categories Positive, Neutral, or Negative.

Data Dictionary:

Field Name	Definition	Type
textID	Unique ID for each piece of text.	ID
text	The text of the tweet	Feature
selected_text	Text that supports the tweet's sentiment	Feature
sentiment	Whether the two questions are duplicates or not	Label

Feature Engineering:

- Since textID is a unique identification number, these doesn't hold any value over the outcome.
- To remove special characters like Asterix or colon. I have used the regular expression **“[^a-zA-Z0-9]/g"** in the function **“check_special”** to remove the special characters.
- The “**check**” function in the code is used to remove the accent from the words. To do so I have used **stringAccents** in the **StringUtils** class.
- The first step of the NLP process is gathering the data and breaking it into understandable parts. RegexTokenizer splits paragraphs and sentences into smaller units that can be more easily assigned meaning. RegexTokenizer also considers only the alphabets by using the method .setPattern in which the regular expression needs to be passed. The regular expression passed here is “**[a-zA-Z']+**”.
- The StopWordsRemover function removes all stop words from input sequences of strings (such as the result of a tokenizer). The stopWords option specifies the list of stopwords.Calling the StopWordsRemover will provide access to some languages' default stop words. The possible possibilities for DefaultStopWords(language) are "danish," "dutch," "english," "finnish," "french," "german," "hungarian," "italian," "norwegian," "portuguese," "russian," "spanish," "swedish," and "turkish." a case for

boolean parameters If the matches should be case sensitive, sensitive lets you know that.

- N-grams are continuous word, symbol, or token sequences in a document. They can be described technically as the adjacent groups of items in a document. When working with text data in NLP, they come into play. Here the N in NGram is 2 to split into 2 or to make them as bigrams.
- Count Vectoriser converts a collection of strings to a matrix of token counts. Count Vectoriser has been used on top of N-Gram
- IDF is a method for determining the meaning of word-based sentences and eliminates the shortcomings of the Bag of Words technique, which is effective for classifying texts or assisting a machine to interpret words into numbers. Input for the IDF is the outcome of Count Vectoriser.
- String indexer converts the given set of strings to index. Like in this case there are 3 different labels Positive, Negative and Neutral. So, the string indexer has converted
  Neutral: 0
  Positive: 1
  Negative: 2

```
number of rows:
27481
+---------+-----+
|sentiment|count|
+---------+-----+
| positive| 8582|
|     null|    2|
|  neutral|11118|
| negative| 7779|
+---------+-----+

+----------+----+-------------+---------+
|textID    |text|selected_text|sentiment|
+----------+----+-------------+---------+
|fdb77c3752|null|null         |neutral  |
+----------+----+-------------+---------+

1
3
+----------+--------------------+--------------------+---------+
|    textID|        text_special|selected_text_special|sentiment|
+----------+--------------------+--------------------+---------+
|cb774db0d1| Id have responde...| Id have responded...|  neutral|
|549e992a42| Sooo SAD I will ...|            Sooo SAD| negative|
|088c60f138|my boss is bullyi...|         bullying me| negative|
|9642c003ef| what interview l...|       leave me alone| negative|
|358bd9e861| Sons of  why cou...|            Sons of | negative|
+----------+--------------------+--------------------+---------+
only showing top 5 rows

root
 |-- textID: string (nullable = true)
 |-- text_special: string (nullable = true)
 |-- selected_text_special: string (nullable = true)
 |-- sentiment: string (nullable = true)

+----------+--------------------+--------------------+---------+
|    textID|    tweet_text_final| selected_text_final|sentiment|
+----------+--------------------+--------------------+---------+
|cb774db0d1| Id have responde...|Id have responded...|  neutral|
|549e992a42| Sooo SAD I will ...|            Sooo SAD| negative|
|088c60f138|my boss is bullyi...|         bullying me| negative|
|9642c003ef| what interview l...|       leave me alone| negative|
|358bd9e861| Sons of  why cou...|            Sons of | negative|
+----------+--------------------+--------------------+---------+
only showing top 5 rows

+-----+---------+-----+
|label|sentiment|count|
+-----+---------+-----+
|  0.0|  neutral|11117|
|  1.0| positive| 8582|
|  2.0| negative| 7779|
+-----+---------+-----+
```

Model Selection

Since LinearSVC offers only binary classification, I had to consider other classification techniques. After testing Logistic Regression, Decision Trees, and Random Forest. I have used Random Forest because it was giving the best result out of all of them.

Hyper Parameters

Parameters of an algorithm that may be tweaked to improve performance. The optimum hyperparameters are typically hard to predict in advance, and tweaking a model is when machine learning transitions from science to trial-and-error engineering. Because hyperparameter tuning is based on experimental findings rather than theory, the easiest way to identify the ideal settings is to test many different combinations and evaluate the performance of each model.

maxDepth:

Max depth can be defined as the longest path between the root node and the leaf node or max number of levels in each decision tree.

maxBins:

Max bins are a decision tree parameter. A number of bins are set while continuous features are made separate or discreet. The default value of maxBin in spark is 32. and the value needs to be set >=2

After multiple trial and error methods, in order to avoid overfitting and minimize the loss. I have taken maxdepth is 3,4 and 6. And when the maxBin is 32,33,34 & 35.

Accuracy:

So I have chosen MultiClassEvaluator as my evaluator function. The cross-validator chooses the model which yields the best accuracy or result, and in this case, the highest accuracy is chosen as the best model.

```
scala> val predictions_nikhilesh = model_nikhilesh.transform(testData)
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [textID: string, tweet_text_final: string ... 12 more fields]

scala> val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
22/11/23 07:13:50 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 1791.0 KiB
accuracy_nikhilesh: Double = 0.7203810369307343

scala>
```

The highest accuracy with which this model can predict the sentiment of the tweet is: **72.039%**

References:

- https://blog.exploratory.io/exploratory-weekly-update-12-3-d4b1d0f620b9
- https://www.kaggle.com/c/quora-question-pairs
- https://www.kaggle.com/competitions/tweet-sentiment-extraction
- https://stackoverflow.com/questions/38874546/spark-crossvalidatormodel-access-other-models-than-the-bestmodel/38874828#38874828
- https://stats.stackexchange.com/questions/530745/what-happens-if-a-random-forest-max-bins-is-set-much-higher-than-the-number-of-c#:~:text=It%20is%20a%20decision%20tree,also%20increases%20computation%20and%20communication
- https://spark.apache.org/docs/2.2.1/api/java/org/apache/spark/ml/classification/RandomForestClassifier.html
- https://spark.apache.org/docs/3.1.1/api/scala/org/apache/spark/ml/classification/RandomForestClassifier.html
- https://spark.apache.org/docs/latest/mllib-decision-tree.html
- https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/ml/feature/RegexTokenizer.html
- https://spark.apache.org/docs/3.0.2/api/scala/org/apache/spark/ml/tuning/CrossValidator.html
- https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/ml/evaluation/BinaryClassificationEvaluator.html

Appendix: Screenshot of the Code

Q1: PREDICT DIABETES

```
scala> val data_nikhilesh = spark.read.format("csv").option("inferSchema", "true"). option("header", "true").load("hdfs://10.128.0.19/BigData/diabetes.csv")
data_nikhilesh: org.apache.spark.sql.DataFrame = [Pregnancies: int, Glucose: int ... 9 more fields]

scala>

scala> data_nikhilesh.drop("_c9","_c10")
res3: org.apache.spark.sql.DataFrame = [Pregnancies: int, Glucose: int ... 7 more fields]

scala>

scala> data_nikhilesh.na.drop().show(false)
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+---+----+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin|BMI|DiabetesPedigreeFunction|Age|Outcome|_c9|_c10|
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+---+----+
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+---+----+


scala>

scala> var df_nikhilesh = data_nikhilesh.drop("_c9","_c10")
df_nikhilesh: org.apache.spark.sql.DataFrame = [Pregnancies: int, Glucose: int ... 7 more fields]

scala>

scala> df_nikhilesh.filter("Outcome = 0").show(false)
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin|BMI |DiabetesPedigreeFunction|Age|Outcome|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
|1          |85     |66           |29           |0      |26.6|0.351                   |31 |0      |
|1          |89     |66           |23           |94     |28.1|0.167                   |21 |0      |
|5          |116    |74           |0            |0      |25.6|0.201                   |30 |0      |
|10         |115    |0            |0            |0      |35.3|0.134                   |29 |0      |
|4          |110    |92           |0            |0      |37.6|0.191                   |30 |0      |
|10         |139    |80           |0            |0      |27.1|1.441                   |57 |0      |
|1          |103    |30           |38           |83     |43.3|0.183                   |33 |0      |
|3          |126    |88           |41           |235    |39.3|0.704                   |27 |0      |
|8          |99     |84           |0            |0      |35.4|0.388                   |50 |0      |
|1          |97     |66           |15           |140    |23.2|0.487                   |22 |0      |
|13         |145    |82           |19           |110    |22.2|0.245                   |57 |0      |
|5          |117    |92           |0            |0      |34.1|0.337                   |38 |0      |
|5          |109    |75           |26           |0      |36.0|0.546                   |60 |0      |
|3          |88     |58           |11           |54     |24.8|0.267                   |22 |0      |
|6          |92     |92           |0            |0      |19.9|0.188                   |28 |0      |
|10         |122    |78           |31           |0      |27.6|0.512                   |45 |0      |
|4          |103    |60           |33           |192    |24.0|0.966                   |33 |0      |
|11         |138    |76           |0            |0      |33.2|0.42                    |35 |0      |
|3          |180    |64           |25           |70     |34.0|0.271                   |26 |0      |
|7          |133    |84           |0            |0      |40.2|0.696                   |37 |0      |
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
only showing top 20 rows


scala> val outcome_nikhilesh = df_nikhilesh.select("Outcome")
outcome_nikhilesh: org.apache.spark.sql.DataFrame = [Outcome: int]

scala>
```

```
scala> println("Correlation between Pregnancies and Outcome: "+df_nikhilesh.stat.corr("Pregnancies", "Outcome").toString+"\nCorrelation between Insulin and Outcome: "+df_nikhilesh.stat.corr("Insulin", "Outcome").toString+"\nCorrelation between Glucose and Outcome: "+df_nikhilesh.stat.corr("Glucose", "Outcome").toString+"\nCorrelation between BloodPressure and Outcome: "+df_nikhilesh.stat.corr("BloodPressure", "Outcome").toString+"\nCorrelation between SkinThickness and Outcome: "+df_nikhilesh.stat.corr("SkinThickness", "Outcome").toString+"\nCorrelation between BMI and Outcome: "+df_nikhilesh.stat.corr("BMI", "Outcome").toString+"\nCorrelation between DiabetesPedigreeFunction and Outcome: "+df_nikhilesh.stat.corr("DiabetesPedigreeFunction", "Outcome").toString+"\nCorrelation between Age and Outcome: "+df_nikhilesh.stat.corr("Age", "Outcome").toString)
Correlation between Pregnancies and Outcome: 0.22189815303398636
Correlation between Insulin and Outcome: 0.13054795488404794
Correlation between Glucose and Outcome: 0.4665813983068737
Correlation between BloodPressure and Outcome: 0.06506835955033274
Correlation between SkinThickness and Outcome: 0.07475223191831945
Correlation between BMI and Outcome: 0.2926946626444454
Correlation between DiabetesPedigreeFunction and Outcome: 0.17384406565296
Correlation between Age and Outcome: 0.2383359830271975?

scala> val assembler_nikhilesh = new VectorAssembler().setInputCols(Array("Glucose","BMI","DiabetesPedigreeFunction","Age","Insulin")).setOutputCol("features")
assembler_nikhilesh: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_472f7b39a2cc, handleInvalid=error, numInputCols=5

scala> val Array(trainingData, testData) = df_nikhilesh.randomSplit(Array(0.65, 0.35), 512)
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Pregnancies: int, Glucose: int ... 7 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Pregnancies: int, Glucose: int ... 7 more fields]

scala> val randomForestModel_nikhilesh = new RandomForestClassifier()
randomForestModel_nikhilesh: org.apache.spark.ml.classification.RandomForestClassifier = rfc_84e378c19fca

scala>   .setFeaturesCol("features")
res10: org.apache.spark.ml.classification.RandomForestClassifier = rfc_84e378c19fca

scala>   .setLabelCol("Outcome")
res11: org.apache.spark.ml.classification.RandomForestClassifier = rfc_84e378c19fca

scala>   .setSeed(745)
res12: res11.type = rfc_84e378c19fca

scala> val pipeline_nikhilesh = new Pipeline()
pipeline_nikhilesh: org.apache.spark.ml.Pipeline = pipeline_75ee5d5df855

scala>   .setStages(Array(assembler_nikhilesh, randomForestModel_nikhilesh))
res13: pipeline_nikhilesh.type = pipeline_75ee5d5df855

scala>

scala> val evaluator_nikhilesh = new MulticlassClassificationEvaluator()
evaluator_nikhilesh: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=mcEval_17276a411a49, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setLabelCol("Outcome")
res14: evaluator_nikhilesh.type = MulticlassClassificationEvaluator: uid=mcEval_17276a411a49, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setPredictionCol("prediction")
res15: res14.type = MulticlassClassificationEvaluator: uid=mcEval_17276a411a49, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setMetricName("accuracy")
res16: res15.type = MulticlassClassificationEvaluator: uid=mcEval_17276a411a49, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala> val Array(trainingData, testData) = df_nikhilesh.randomSplit(Array(0.65, 0.35), 512)
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Pregnancies: int, Glucose: int ... 7 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Pregnancies: int, Glucose: int ... 7 more fields]

scala>
```

```
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 32)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 33)
accuracy of the model is :77.53623188405797
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 34)
accuracy of the model is :74.27536231884058
Vector(m, a, x, D, e, p, t, h,  , 2 maxBin: 35)
accuracy of the model is :74.63768115942028
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 32)
accuracy of the model is :77.17391304347827
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 33)
accuracy of the model is :76.08695652173914
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 34)
accuracy of the model is :76.81159420289855
Vector(m, a, x, D, e, p, t, h,  , 3 maxBin: 35)
accuracy of the model is :76.44927536231883
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 32)
accuracy of the model is :75.72463768115942
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 33)
accuracy of the model is :76.08695652173914
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 34)
accuracy of the model is :75.0
Vector(m, a, x, D, e, p, t, h,  , 4 maxBin: 35)
accuracy of the model is :77.17391304347827
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 32)
accuracy of the model is :75.0
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 33)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 34)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 5 maxBin: 35)
accuracy of the model is :75.36231884057972
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 32)
accuracy of the model is :76.44927536231883
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 33)
accuracy of the model is :74.63768115942028
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 34)
accuracy of the model is :73.55072463768117
Vector(m, a, x, D, e, p, t, h,  , 6 maxBin: 35)
accuracy of the model is :74.63768115942028
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 32)
accuracy of the model is :73.18840579710145
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 33)
accuracy of the model is :75.0
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 34)
accuracy of the model is :73.18840579710145
Vector(m, a, x, D, e, p, t, h,  , 7 maxBin: 35)
accuracy of the model is :73.55072463768117
```

```
scala> val cvModel_nikhilesh = cross_validator_nikhilesh.fit(trainingData)
cvModel_nikhilesh: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_ea1356f3d6a9, bestModel=pipeline_d0a
7ab9cf0b4, numFolds=3

scala> val predictions_nikhilesh = cvModel_nikhilesh.transform(testData)
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [Pregnancies: int, Glucose: double ... 9 more fields]

scala> val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
accuracy_nikhilesh: Double = 0.7717391304347826

scala> println("accuracy of the model is :"+(accuracy_nikhilesh*100))
accuracy of the model is :77.17391304347827

scala>
```

Q2: PREDICT CANCER

```
val data_nikhilesh = spark.read.format("csv").option("inferSchema", "true").option("header", "true").load("hdfs://10.128.0.19/BigData/cancer.csv")

// Exiting paste mode, now interpreting.

import org.apache.spark.sql.functions._
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.DoubleType
import org.apache.spark.ml.linalg.{Matrix, Vectors}
import org.apache.spark.ml.stat.Correlation
import org.apache.spark.sql.Row
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassi...

scala> var df_cacner_nikhilesh = data_nikhilesh
df_cacner_nikhilesh: org.apache.spark.sql.DataFrame = [id: int, Clump Thickness: int ... 9 more fields]

scala> df_cacner_nikhilesh.printSchema
root
 |-- id: integer (nullable = true)
 |-- Clump Thickness: integer (nullable = true)
 |-- UofCSize: integer (nullable = true)
 |-- UofCShape: integer (nullable = true)
 |-- Marginal Adhesion: integer (nullable = true)
 |-- SECSize: integer (nullable = true)
 |-- Bare Nuclei: integer (nullable = true)
 |-- Bland Chromatin: integer (nullable = true)
 |-- Normal Nucleoli: integer (nullable = true)
 |-- Mitoses: integer (nullable = true)
 |-- Class: integer (nullable = true)

scala>
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

println("Correlation between UofCSize and Class: "+df_cacner_nikhilesh.stat.corr("UofCSize", "Class", "pear
son").toString
 +"\nCorrelation between UofCShape and Class: "+df_cacner_nikhilesh.stat.corr("UofCShape", "Class", "pearso
n").toString+
 "\nCorrelation between Marginal Adhesion and Class: "+df_cacner_nikhilesh.stat.corr("Marginal Adhesion", "
Class", "pearson").toString+
 "\nCorrelation between SECSize and Class: "+df_cacner_nikhilesh.stat.corr("SECSize", "Class", "pearson").t
oString+
 "\nCorrelation between Bare Nuclei and Class: "+df_cacner_nikhilesh.stat.corr("Bare Nuclei", "Class", "pea
rson").toString+
 "\nCorrelation between Mitoses and Class: "+df_cacner_nikhilesh.stat.corr("Mitoses", "Class", "pearson").t
oString+
 "\nCorrelation between Normal Nucleoli and Class: "+df_cacner_nikhilesh.stat.corr("Normal Nucleoli", "Clas
s", "pearson").toString)

// Exiting paste mode, now interpreting.

Correlation between UofCSize and Class: 0.8208014428258776
Correlation between UofCShape and Class: 0.821890947688868
Correlation between Marginal Adhesion and Class: 0.7062941354660839
Correlation between SECSize and Class: 0.6909581590873204
Correlation between Bare Nuclei and Class: 0.8226958729964622
Correlation between Mitoses and Class: 0.4234479212952124
Correlation between Normal Nucleoli and Class: 0.7186771878756356

scala>
```

```
scala> df_cacner_nikhilesh.stat.crosstab("UofCSize","Class").show()
+--------------+---+---+
|UofCSize_Class|  2|  4|
+--------------+---+---+
|             5|  0| 30|
|            10|  0| 67|
|             1|369|  4|
|             6|  0| 25|
|             9|  1|  5|
|             2| 37|  8|
|             7|  1| 18|
|             3| 27| 25|
|             8|  1| 27|
|             4|  8| 30|
+--------------+---+---+


scala> df_cacner_nikhilesh.stat.crosstab("UofCShape","Class").show()
+---------------+---+---+
|UofCShape_Class|  2|  4|
+---------------+---+---+
|              5|  2| 30|
|             10|  0| 58|
|              1|344|  2|
|              6|  2| 27|
|              9|  0|  7|
|              2| 51|  7|
|              7|  2| 28|
|              3| 30| 23|
|              8|  1| 26|
|              4| 12| 31|
+---------------+---+---+


scala> df_cacner_nikhilesh.stat.crosstab("Marginal Adhesion","Class").show()
+----------------------+---+---+
|Marginal Adhesion_Class|  2|  4|
+----------------------+---+---+
|                     5|  4| 19|
|                    10|  1| 54|
|                     1|363| 30|
|                     6|  3| 18|
|                     9|  0|  4|
|                     2| 37| 21|
|                     7|  0| 13|
```

```
scala> df_cacner_nikhilesh.stat.crosstab("SECSize","Class").show()
+-------------+---+---+
|SECSize_Class|  2|  4|
+-------------+---+---+
|            5|  5| 34|
|           10|  1| 30|
|            1| 43|  1|
|            6|  1| 39|
|            9|  0|  2|
|            2|355| 21|
|            7|  2|  9|
|            3| 28| 43|
|            8|  2| 19|
|            4|  7| 41|
+-------------+---+---+


scala> df_cacner_nikhilesh.stat.crosstab("Bare Nuclei","Class").show()
+-----------------+---+---+
|Bare Nuclei_Class|  2|  4|
+-----------------+---+---+
|                5| 10| 20|
|               10|  3|129|
|                1|387| 15|
|                6|  0|  4|
|                9|  0|  9|
|                2| 21|  9|
|                7|  1|  7|
|                3| 14| 14|
|                8|  2| 19|
|                4|  6| 13|
+-----------------+---+---+


scala> df_cacner_nikhilesh.stat.crosstab("Bland Chromatin","Class").show()
+--------------------+---+---+
|Bland Chromatin_Class|  2|  4|
+--------------------+---+---+
|                   5|  4| 30|
|                  10|  0| 20|
|                   1|148|  2|
|                   6|  1|  8|
|                   9|  0| 11|
|                   2|153|  7|
|                   7|  6| 65|
|                   3|125| 36|
|                   8|  0| 28|
|                   4|  7| 32|
+--------------------+---+---+
```

```
scala> df_cacner_nikhilesh.stat.crosstab("Bland Chromatin","Class").show()
+--------------------+---+---+
|Bland Chromatin_Class|  2|  4|
+--------------------+---+---+
|                   5|  4| 30|
|                  10|  0| 20|
|                   1|148|  2|
|                   6|  1|  8|
|                   9|  0| 11|
|                   2|153|  7|
|                   7|  6| 65|
|                   3|125| 36|
|                   8|  0| 28|
|                   4|  7| 32|
+--------------------+---+---+


scala> df_cacner_nikhilesh.stat.crosstab("Mitoses","Class").show()
+-------------+---+---+
|Mitoses_Class|  2|  4|
+-------------+---+---+
|            5|  1|  5|
|           10|  0| 14|
|            1|431|132|
|            6|  0|  3|
|            2|  8| 27|
|            7|  1|  8|
|            3|  2| 31|
|            8|  1|  7|
|            4|  0| 12|
+-------------+---+---+


scala> df_cacner_nikhilesh.stat.crosstab("Normal Nucleoli","Class").show()
+--------------------+---+---+
|Normal Nucleoli_Class|  2|  4|
+--------------------+---+---+
|                   5|  2| 17|
|                  10|  0| 60|
|                   1|391| 41|
|                   6|  4| 18|
|                   9|  0| 15|
|                   2| 30|  6|
|                   7|  2| 14|
|                   3| 11| 31|
|                   8|  3| 20|
|                   4|  1| 17|
+--------------------+---+---+
```

```
scala> df_cacner_nikhilesh.groupBy(col("UofCSize")).count().show
+---------+-----+
|UofCSize|count|
+---------+-----+
|        1|  373|
|        6|   25|
|        3|   52|
|        5|   30|
|        9|    6|
|        4|   38|
|        8|   28|
|        7|   19|
|       10|   67|
|        2|   45|
+---------+-----+

scala> df_cacner_nikhilesh.groupBy(col("UofCShape")).count().show
+----------+-----+
|UofCShape|count|
+----------+-----+
|         1|  346|
|         6|   29|
|         3|   53|
|         5|   32|
|         9|    7|
|         4|   43|
|         8|   27|
|         7|   30|
|        10|   58|
|         2|   58|
+----------+-----+

scala> df_cacner_nikhilesh.groupBy(col("Marginal Adhesion")).count().show
+------------------+-----+
|Marginal Adhesion|count|
+------------------+-----+
|                 1|  393|
|                 6|   21|
|                 3|   58|
|                 5|   23|
|                 9|    4|
|                 4|   33|
|                 8|   25|
|                 7|   13|
|                10|   55|
|                 2|   58|
+------------------+-----+
```

```
scala> df_cacner_nikhilesh.groupBy(col("SECSize")).count().show
+--------+-----+
|SECSize|count|
+--------+-----+
|       1|   44|
|       6|   40|
|       3|   71|
|       5|   39|
|       9|    2|
|       4|   48|
|       8|   21|
|       7|   11|
|      10|   31|
|       2|  376|
+--------+-----+

scala> df_cacner_nikhilesh.groupBy(col("Bare Nuclei")).count().show
+------------+-----+
|Bare Nuclei|count|
+------------+-----+
|           1|  402|
|           6|    4|
|           3|   28|
|           5|   30|
|           9|    9|
|           4|   19|
|           8|   21|
|           7|    8|
|          10|  132|
|           2|   30|
+------------+-----+

scala> df_cacner_nikhilesh.groupBy(col("Bland Chromatin")).count().show
+----------------+-----+
|Bland Chromatin|count|
+----------------+-----+
|               1|  150|
|               6|    9|
|               3|  161|
|               5|   34|
|               9|   11|
|               4|   39|
|               8|   28|
|               7|   71|
|              10|   20|
|               2|  160|
+----------------+-----+
```

```
+--------+-----+
|Mitoses|count|
+--------+-----+
|       1|  563|
|       6|    3|
|       3|   33|
|       5|    6|
|       4|   12|
|       8|    8|
|       7|    9|
|      10|   14|
|       2|   35|
+--------+-----+


scala> df_cacner_nikhilesh.groupBy(col("Normal Nucleoli")).count().show
+----------------+-----+
|Normal Nucleoli|count|
+----------------+-----+
|               1|  432|
|               6|   22|
|               3|   42|
|               5|   19|
|               9|   15|
|               4|   18|
|               8|   23|
|               7|   16|
|              10|   60|
|               2|   36|
+----------------+-----+


scala> df_cacner_nikhilesh.stat.crosstab("UofCSize",  "Class").show()
+---------------+---+---+
|UofCSize_Class|  2|  4|
+---------------+---+---+
|              5|  0| 30|
|             10|  0| 67|
|              1|369|  4|
|              6|  0| 25|
|              9|  1|  5|
|              2| 37|  8|
|              7|  1| 18|
|              3| 27| 25|
|              8|  1| 27|
|              4|  8| 30|
+---------------+---+---+
```

```
scala> val assembler_cancer_nikhilesh = new VectorAssembler().setInputCols(Array("Clump Thickness","UofCSize","UofCShape","Marginal Adhesion","SECSize","Bare Nuclei","Bland Chromatin","Normal Nucleoli","Mitoses","Class")).setOutputCol("features")
assembler_cancer_nikhilesh: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_5b0ab38a2fb2, handleInvalid=error, numInputCols=10

scala> val indexer_nikhilesh = new StringIndexer().setInputCol("Class").setOutputCol("label")
indexer_nikhilesh: org.apache.spark.ml.feature.StringIndexer = strIdx_9a30ea86cc0b

scala> val Array(training, test) = df_cacner_nikhilesh.randomSplit(Array(0.7, 0.3))
training: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: int, Clump Thickness: int ... 9 more fields]
test: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: int, Clump Thickness: int ... 9 more fields]

scala> val logistic_nikhilesh = new LogisticRegression().setMaxIter(10).setLabelCol("label").setFeaturesCol("features")
logistic_nikhilesh: org.apache.spark.ml.classification.LogisticRegression = logreg_9333a8d59d25

scala> val evaluator_nikhilesh = new BinaryClassificationEvaluator().setLabelCol("label").setRawPredictionCol("rawPrediction").setMetricName("areaUnderROC")
evaluator_nikhilesh: org.apache.spark.ml.evaluation.BinaryClassificationEvaluator = BinaryClassificationEvaluator: uid=binEval_c9046ce745d3, metricName=areaUnderROC, numBins=1000
```

```
val MaxIter_Array = Array(10,20,30,40)
val pipeline_nikhilesh = new Pipeline()
  .setStages(Array(assembler_cancer_nikhilesh,indexer_nikhilesh, logistic_nikhilesh))
for (RegParam1 <- RegParam_Array)
  for(MaxIter1 <- MaxIter_Array)
  {
val paramGrid_nikhilesh = new ParamGridBuilder().addGrid(logistic_nikhilesh.regParam, Array(RegParam1))
  .addGrid(logistic_nikhilesh.elasticNetParam, ElasticNetParam1)
  .addGrid(logistic_nikhilesh.tol, tolerance)
  .addGrid(logistic_nikhilesh.maxIter, Array(MaxIter1)).build()



val cross_validator_nikhilesh = new CrossValidator()
.setEstimatorParamMaps(paramGrid_nikhilesh)
  .setEstimator(pipeline_nikhilesh)
  .setEvaluator(evaluator_nikhilesh)
  .setNumFolds(5)
val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("RegParam: ":+RegParam1.toString+" MaxIter: "+MaxIter1.toString)
println("best accuracy of the model is :"+(accuracy_nikhilesh*100))
}

// Exiting paste mode, now interpreting.
```

```
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 10)
best accuracy of the model is :99.42942942942943
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 20)
best accuracy of the model is :99.96996996996998
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 30)
best accuracy of the model is :99.97997997997997
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 40)
best accuracy of the model is :99.97997997997997
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 10)
best accuracy of the model is :99.64964964964965
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 20)
best accuracy of the model is :99.91991991991992
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 30)
best accuracy of the model is :99.94994994994995
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 40)
best accuracy of the model is :99.94994994994995
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 10)
best accuracy of the model is :99.70970970970971
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 20)
best accuracy of the model is :99.92992992992994
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 30)
best accuracy of the model is :99.92992992992994
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 40)
best accuracy of the model is :99.92992992992994
```

```
res7: evaluator_nikhilesh.type = MulticlassClassificationEvaluator: uid=mcEval_7b87a8a40f71, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setPredictionCol("prediction")
res8: res7.type = MulticlassClassificationEvaluator: uid=mcEval_7b87a8a40f71, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>   .setMetricName("accuracy")
res9: res8.type = MulticlassClassificationEvaluator: uid=mcEval_7b87a8a40f71, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala> :paste
// Entering paste mode (ctrl-D to finish)

for (RegParam1 <- RegParam_Array)
  for(MaxIter1 <- MaxIter_Array)
  {
val paramGrid_nikhilesh = new ParamGridBuilder().addGrid(logistic_nikhilesh.regParam, Array(RegParam1))
  .addGrid(logistic_nikhilesh.elasticNetParam, ElasticNetParam1)
  .addGrid(logistic_nikhilesh.tol, tolerance)
  .addGrid(logistic_nikhilesh.maxIter, Array(MaxIter1)).build()
val cross_validator_nikhilesh = new CrossValidator()
.setEstimatorParamMaps(paramGrid_nikhilesh)
  .setEstimator(pipeline_nikhilesh)
  .setEvaluator(evaluator_nikhilesh)
  .setNumFolds(5)
val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("RegParam: ":+RegParam1+" MaxIter: "+MaxIter1.toString)
println("best accuracy of the model is :"+(accuracy_nikhilesh*100))
}

// Exiting paste mode, now interpreting.

Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 10)
best accuracy of the model is :94.25837320574163
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 20)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 30)
best accuracy of the model is :99.04306220095694
Vector(R, e, g, P, a, r, a, m, :,  , 0.1 MaxIter: 40)
best accuracy of the model is :99.04306220095694
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 10)
best accuracy of the model is :98.08612440191388
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 20)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 30)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.2 MaxIter: 40)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 10)
best accuracy of the model is :97.1291866028708
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 20)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 30)
best accuracy of the model is :98.56459330143541
Vector(R, e, g, P, a, r, a, m, :,  , 0.3 MaxIter: 40)
best accuracy of the model is :98.56459330143541
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val paramGrid_nikhilesh = new ParamGridBuilder().addGrid(logistic_nikhilesh.regParam, Array(0.1))
  .addGrid(logistic_nikhilesh.elasticNetParam, ElasticNetParam1)
  .addGrid(logistic_nikhilesh.tol, tolerance)
  .addGrid(logistic_nikhilesh.maxIter, Array(10)).build()
val cross_validator_nikhilesh = new CrossValidator()
.setEstimatorParamMaps(paramGrid_nikhilesh)
  .setEstimator(pipeline_nikhilesh)
  .setEvaluator(evaluator_nikhilesh)
  .setNumFolds(5)
val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("best accuracy of the model is :"+(accuracy_nikhilesh*100))

// Exiting paste mode, now interpreting.

best accuracy of the model is :94.25837320574163
paramGrid_nikhilesh: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        logreg_43214f6bc7e4-elasticNetParam: 0.001,
        logreg_43214f6bc7e4-maxIter: 10,
        logreg_43214f6bc7e4-regParam: 0.1,
        logreg_43214f6bc7e4-tol: 1.0E-6
})
cross_validator_nikhilesh: org.apache.spark.ml.tuning.CrossValidator = cv_560ef2eddbe5
cvModel_nikhilesh: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_560ef2eddbe5, bestModel=pipeline_65d26d7eeba9, numFolds=5
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [id: int, Clump Thickness: int ... 14 more fields]
accuracy_nikhilesh: Double = 0.9425837320574163
```

```
scala> predictions__naive_bayes_nikhilesh.show
+-------+--------------+--------+---------+-----------------+---------+----------+---------------+---------------+--------+-----+--------------------+-----+--------------------+--------------------+----------+
|     id|Clump Thickness|UofCSize|UofCShape|Marginal Adhesion|SECSize|Bare Nuclei|Bland Chromatin|Normal Nucleoli|Mitoses|Class|            features|label|       rawPrediction|         probability|prediction|
+-------+--------------+--------+---------+-----------------+---------+----------+---------------+---------------+--------+-----+--------------------+-----+--------------------+--------------------+----------+
|1016277|             6|       8|        8|                1|        3|         4|              3|              7|       1|    2|[6.0,8.0,8.0,1.0,...|  0.0|[-101.77243612572...|[0.00774613077586...|       1.0|
|1018099|             1|       1|        1|                1|        2|        10|              3|              1|       1|    2|[1.0,1.0,1.0,1.0,...|  0.0|[-55.113930032820...|[0.05711414481328...|       1.0|
|1031608|             2|       1|        1|                1|        1|         1|              2|              1|       1|    2|[2.0,1.0,1.0,1.0,...|  0.0|[-29.921723116904...|[0.83197575611721...|       0.0|
|1033078|             2|       1|        1|                1|        2|         1|              1|              1|       5|    2|[2.0,1.0,1.0,1.0,...|  0.0|[-41.061138657275...|[0.95743799685357...|       0.0|
|1035283|             1|       1|        1|                1|        1|         1|              3|              1|       1|    2|[1.0,1.0,1.0,1.0,...|  0.0|[-30.311700707573...|[0.80114442292229...|       0.0|
|1043068|             3|       1|        1|                1|        2|         1|              2|              1|       1|    2|[3.0,1.0,1.0,1.0,...|  0.0|[-33.727617616948...|[0.90313306944079...|       0.0|
|1044572|             8|       7|        5|               10|        7|         9|              5|              5|       4|    4|[8.0,7.0,5.0,10.0...|  1.0|[-149.92784770937...|[0.07450938677366...|       1.0|
|1050670|            10|       7|        7|                6|        4|        10|              4|              1|       2|    4|[10.0,7.0,7.0,6.0...|  1.0|[-126.59285195265...|[0.07703433576770...|       1.0|
|1056171|             2|       1|        1|                1|        2|         1|              2|              1|       1|    2|[2.0,1.0,1.0,1.0,...|  0.0|[-32.003951064144...|[0.86631675597749...|       0.0|
|1058849|             5|       1|        1|                1|        2|         1|              1|              1|       1|    2|[5.0,1.0,1.0,1.0,...|  0.0|[-35.061304779083...|[0.94281177212700...|       0.0|
|1059552|             1|       1|        1|                1|        2|         1|              3|              1|       1|    2|[1.0,1.0,1.0,1.0,...|  0.0|[-32.399330454813...|[0.84057932350835...|       0.0|
|1063373|             3|       2|        1|                1|        1|         1|              2|              1|       1|    2|[3.0,2.0,1.0,1.0,...|  0.0|[-34.205210106347...|[0.82925747117627...|       0.0|
|1067444|             2|       1|        1|                1|        2|         1|              2|              1|       1|    2|[2.0,1.0,1.0,1.0,...|  0.0|[-32.003951064144...|[0.86631675597749...|       0.0|
|1070935|             1|       1|        3|                1|        2|         1|              1|              1|       1|    2|[1.0,1.0,3.0,1.0,...|  0.0|[-33.147189571749...|[0.67074694202025...|       0.0|
|1072179|            10|       7|        7|                3|        8|         5|              7|              4|       3|    4|[10.0,7.0,7.0,3.0...|  1.0|[-131.93094173262...|[0.78078771038043...|       0.0|
|1073836|             5|       1|        1|                1|        2|         1|              2|              1|       1|    2|[5.0,1.0,1.0,1.0,...|  0.0|[-37.174950722556...|[0.95073587864513...|       0.0|
|1076352|             3|       6|        4|               10|        3|         3|              3|              4|       1|    4|[3.0,6.0,4.0,10.0...|  1.0|[-97.552760493596...|[0.06996446551566...|       1.0|
|1080233|             7|       6|        6|                3|        2|        10|              7|              1|       1|    4|[7.0,6.0,6.0,3.0,...|  1.0|[-108.37630817321...|[0.05661001096164...|       1.0|
|1081791|             6|       2|        1|                1|        1|         1|              7|              1|       1|    2|[6.0,2.0,1.0,1.0,...|  0.0|[-49.944439482123...|[0.96950565314905...|       0.0|
|1083817|             3|       1|        1|                1|        2|         1|              2|              1|       1|    2|[3.0,1.0,1.0,1.0,...|  0.0|[-33.727617616948...|[0.90313306944079...|       0.0|
+-------+--------------+--------+---------+-----------------+---------+----------+---------------+---------------+--------+-----+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows


scala> ;paste
<console>:2: error: illegal start of simple expression
       ;paste
       ^

scala> :paste
// Entering paste mode (ctrl-D to finish)

val evaluator_nikhilesh = new MulticlassClassificationEvaluator()
  .setLabelCol("label")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")

// Exiting paste mode, now interpreting.

evaluator_nikhilesh: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=mcEval_967c358920fe, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala> val accuracy_naive_nikhilesh = evaluator.evaluate(predictions__naive_bayes_nikhilesh)
<console>:66: error: not found: value evaluator
       val accuracy_naive_nikhilesh = evaluator.evaluate(predictions__naive_bayes_nikhilesh)
                                      ^

scala> val accuracy_naive_nikhilesh = evaluator_nikhilesh.evaluate(predictions__naive_bayes_nikhilesh)
accuracy_naive_nikhilesh: Double = 0.9261992619926199

scala>
```

KAGGLE COMPETITION -1 QUORA QUESTION PAIRS

```
number of rows:
404301
+----+----+----+---------+---------+------------+
|id  |qid1|qid2|question1|question2|is_duplicate|
+----+----+----+---------+---------+------------+
|null|0   |null|null     |null     |null        |
|null|null|null|null     |null     |null        |
|null|null|null|null     |null     |null        |
|null|null|0   |null     |null     |null        |
|null|null|null|null     |null     |null        |
|null|null|null|null     |null     |null        |
|null|0   |null|null     |null     |null        |
|null|0   |null|null     |null     |null        |
|null|0   |null|null     |null     |null        |
|null|0   |null|null     |null     |null        |
+----+----+----+---------+---------+------------+

+------+------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+---------+------------+
|id    |qid1  |qid2  |question1                                                                                                                                                                         |question2|is_duplicate|
+------+------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+---------+------------+
|2332  |4637  |4638  |Which is the best RO water purifier in Ahmedabad?                                                                                                                                 |null     |null        |
|null  |0     |null  |null                                                                                                                                                                              |null     |null        |
|null  |null  |null  |null                                                                                                                                                                              |null     |null        |
|null  |null  |null  |null                                                                                                                                                                              |null     |null        |
|105780|174363|174364|How can I develop android app?                                                                                                                                                    |null     |0           |
|null  |null  |0     |null                                                                                                                                                                              |null     |null        |
|185148|270195|282703|What are the differences between "                                                                                                                                                |null     |null        |
|null  |null  |null  |null                                                                                                                                                                              |null     |null        |
|null  |null  |null  |null                                                                                                                                                                              |null     |null        |
|201841|303951|174364|How can I create an Android app?                                                                                                                                                  |null     |0           |
|224791|270195|194360|What are the differences between "                                                                                                                                                |null     |null        |
|227767|336684|336685|Is the book Who Is Muhammad? [Book] By Jafar Subhani (Author) the best ever book about prophet muhammad?                                                                          |null     |null        |
|null  |0     |null  |null                                                                                                                                                                              |null     |null        |
|264607|381419|381420|Given a set of busy time intervals of two people as in a calendar, what is the algorithmic approach to find the free time intervals of both the people so as to arrange a new meeting?|null     |null        |
|null  |null  |null  |0                                                                                                                                                                                 |null     |null        |
|null  |0     |null  |null                                                                                                                                                                              |null     |null        |
|283933|404107|404108|What does the Quran say about the Bible?                                                                                                                                          |null     |null        |
|null  |0     |null  |null                                                                                                                                                                              |null     |null        |
|null  |0     |null  |null                                                                                                                                                                              |null     |null        |
+------+------+------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+---------+------------+

number of rows after removing duplicates:
402482
```

```
+------------+------+
|is_duplicate| count|
+------------+------+
|           1|149132|
|           5|     6|
|         -36|     1|
|           0|253343|
+------------+------+

+-----+------+
|label| count|
+-----+------+
|    1|149132|
|    5|     6|
|  -36|     1|
|    0|253343|
+-----+------+

quora_nikhilesh: org.apache.spark.sql.DataFrame = [id: string, qid1: string ... 4 more fields]
q_nikhilesh: org.apache.spark.sql.DataFrame = [id: int, qid1: int ... 4 more fields]
q_nikhilesh: org.apache.spark.sql.DataFrame = [id: int, qid1: int ... 4 more fields]
check: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$5588/496688719@62f4a010,StringType,List(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,true)
check_special: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$5589/1360736660@fbef88d,StringType,List(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,tr...
```

```
scala> training.groupBy($"label").count().show
+-----+------+
|label| count|
+-----+------+
|    1|104299|
|    0|177205|
+-----+------+


scala> val cvModel_nikhilesh = cross_validator_nikhilesh.fit(training)
cvModel_nikhilesh: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_9842a7f748d0, bestModel=pipeline_e76d678a1983, numFolds=4

scala>

scala> :paste
// Entering paste mode (ctrl-D to finish)

val predictions_nikhilesh = cvModel_nikhilesh.transform(test)
val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
println("accuracy of the model is :"+(accuracy_nikhilesh*100))

// Exiting paste mode, now interpreting.

accuracy of the model is :72.76868009688272
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [qid2: int, qid1: int ... 11 more fields]
accuracy_nikhilesh: Double = 0.7276868009688272

scala>
```

KAGGLE COMPETITION - 2: TWEET SENTIMENT EXTRACTION

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val first_dataframe_twitter_nikhilesh = spark.read.format("csv").option("inferSchema", "true").option("header", "true").load("hdfs://10.128.0.19/twitter_sentiment_analysis/train.csv")
println("number of rows: \n"+first_dataframe_twitter_nikhilesh.count().toString)
first_dataframe_twitter_nikhilesh.groupBy(col("sentiment")).count().show

first_dataframe_twitter_nikhilesh.filter(first_dataframe_twitter_nikhilesh("text").isNull).show(false)

println(first_dataframe_twitter_nikhilesh.filter(first_dataframe_twitter_nikhilesh("text").isNull || col("text")===("").count())
 println(first_dataframe_twitter_nikhilesh.filter(first_dataframe_twitter_nikhilesh("selected_text").isNull || col("selected_text")===("").count())
 val df_2_nikhilesh = first_dataframe_twitter_nikhilesh.na.drop()
 val check = udf((colValue: String) => {
  StringUtils.stripAccents(colValue)
})
val check_special = udf((colValue: String) => {
  colValue.replaceAll("[^\\p{Alpha}\\p{Digit} ]+","");
})
var df_3_nikhilesh = df_2_nikhilesh.select(col("textID"),check_special(col("text")).as("text_special"),check_special(col("selected_text")).as("selected_text_special"),col("sentiment"))
df_3_nikhilesh.show(5)
df_3_nikhilesh.printSchema
df_3_nikhilesh = df_3_nikhilesh.select(col("textID"),check(col("text_special")).as("tweet_text_final"),check(col("selected_text_special")).as("selected_text_final"),col("sentiment"))
df_3_nikhilesh.show(5)
val df_4_nikhilesh = df_3_nikhilesh.select(col("textID"), col("tweet_text_final"),col("sentiment"), col("selected_text_final"),row_number().over(Window.partitionBy("sentiment").orderBy(col("sentiment"))).alias("sentiment_2"))
val indexer_nikhilesh = new StringIndexer().setInputCol("sentiment").setOutputCol("label")
val df_5_nikhilesh = indexer_nikhilesh.fit(df_4_nikhilesh).transform(df_4_nikhilesh)
df_5_nikhilesh.groupBy(col("label"),col("sentiment")).count().show

// Exiting paste mode, now interpreting.
```

```
number of rows:
27481
+---------+-----+
|sentiment|count|
+---------+-----+
| positive| 8582|
|     null|    2|
|  neutral|11118|
| negative| 7779|
+---------+-----+

+----------+----+-------------+---------+
|textID    |text|selected_text|sentiment|
+----------+----+-------------+---------+
|fdb77c3752|null|null         |neutral  |
+----------+----+-------------+---------+

1
3
+----------+--------------------+--------------------+---------+
|    textID|        text_special|selected_text_special|sentiment|
+----------+--------------------+--------------------+---------+
|cb774db0d1| Id have responde...| Id have responded...|  neutral|
|549e992a42| Sooo SAD I will ...|            Sooo SAD| negative|
|088c60f138|my boss is bullyi...|          bullying me| negative|
|9642c003ef| what interview l...|       leave me alone| negative|
|358bd9e861| Sons of  why cou...|            Sons of  | negative|
+----------+--------------------+--------------------+---------+
only showing top 5 rows

root
 |-- textID: string (nullable = true)
 |-- text_special: string (nullable = true)
 |-- selected_text_special: string (nullable = true)
 |-- sentiment: string (nullable = true)

+----------+--------------------+--------------------+---------+
|    textID|    tweet_text_final| selected_text_final|sentiment|
+----------+--------------------+--------------------+---------+
|cb774db0d1| Id have responde...|Id have responded...|  neutral|
|549e992a42| Sooo SAD I will ...|            Sooo SAD| negative|
|088c60f138|my boss is bullyi...|          bullying me| negative|
|9642c003ef| what interview l...|       leave me alone| negative|
|358bd9e861| Sons of  why cou...|            Sons of  | negative|
+----------+--------------------+--------------------+---------+
only showing top 5 rows

+-----+---------+-----+
|label|sentiment|count|
+-----+---------+-----+
|  0.0|  neutral|11117|
|  1.0| positive| 8582|
|  2.0| negative| 7779|
+-----+---------+-----+
```

```
val tokenizer_nikhilesh = new RegexTokenizer()
 .setPattern("[a-zA-Z']+")
 .setGaps(false)
 .setInputCol("tweet_text_final")
 .setOutputCol("words")
val remover_nikhilesh = new StopWordsRemover()
 .setInputCol("words")
 .setOutputCol("filtered")
val ngram_nikhilesh = new NGram()
 .setN(2)
 .setInputCol("filtered")
 .setOutputCol("ngram-2")
val cv2_nikhilesh: CountVectorizer = new CountVectorizer()
 .setInputCol("ngram-2")
 .setOutputCol("ngram-2-features")
val cv2idf_nikhilesh = new IDF()
 .setInputCol("ngram-2-features")
 .setOutputCol("cv2-idf-features")

// Exiting paste mode, now interpreting.

tokenizer_nikhilesh: org.apache.spark.ml.feature.RegexTokenizer = RegexTokenizer: uid=regexTok_cc785d46e514, minTokenLength=1, gaps=false, pattern=[a-zA-Z']+, toLowercase=true
remover_nikhilesh: org.apache.spark.ml.feature.StopWordsRemover = StopWordsRemover: uid=stopWords_438627df0911, numStopWords=181, locale=en, caseSensitive=false
ngram_nikhilesh: org.apache.spark.ml.feature.NGram = NGram: uid=ngram_e73cdd0fda21, n=2
cv2_nikhilesh: org.apache.spark.ml.feature.CountVectorizer = cntVec_bde3e8b98fde
cv2idf_nikhilesh: org.apache.spark.ml.feature.IDF = idf_07d48420bfd1

scala>
```

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val randomForestModel_nikhilesh = new RandomForestClassifier()
 .setFeaturesCol("cv2-idf-features")
 .setLabelCol("label")
val pipeline_nikhilesh = new Pipeline()
  .setStages(Array(tokenizer_nikhilesh, remover_nikhilesh, ngram_nikhilesh, cv2_nikhilesh, cv2idf_nikhilesh,randomForestModel_nikhilesh))
val paramGrid_nikhilesh = new ParamGridBuilder()
  .addGrid(randomForestModel_nikhilesh.maxDepth, Array(3,4,5))
  .addGrid(randomForestModel_nikhilesh.maxBins, Array(32,33,34,35))
  .addGrid(randomForestModel_nikhilesh.impurity, Array("entropy","gini")).build()
val evaluator_nikhilesh = new MulticlassClassificationEvaluator()
 .setLabelCol("label")
 .setPredictionCol("prediction")
 .setMetricName("accuracy")
val cross_validator_nikhilesh = new CrossValidator()
 .setEstimator(pipeline_nikhilesh)
 .setEvaluator(evaluator_nikhilesh)
 .setEstimatorParamMaps(paramGrid_nikhilesh)
 .setNumFolds(3)
 val Array(trainingData, testData) = df_5_nikhilesh.randomSplit(Array(0.8, 0.2), 754)

// Exiting paste mode, now interpreting.

randomForestModel_nikhilesh: org.apache.spark.ml.classification.RandomForestClassifier = rfc_c81d1ae75155
pipeline_nikhilesh: org.apache.spark.ml.Pipeline = pipeline_d966218e488f
paramGrid_nikhilesh: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        rfc_c81d1ae75155-impurity: entropy,
        rfc_c81d1ae75155-maxBins: 32,
        rfc_c81d1ae75155-maxDepth: 3
}, {
        rfc_c81d1ae75155-impurity: entropy,
        rfc_c81d1ae75155-maxBins: 33,
        rfc_c81d1ae75155-maxDepth: 3
}, {
        rfc_c81d1ae75155-impurity: entropy,
        rfc_c81d1ae75155-maxBins: 34,
        rfc_c81d1ae75155-maxDepth: 3
}, {
        rfc_c81d1ae75155-impurity: entropy,
        rfc_c81d1ae75155-maxBins: 35,
        rfc_c81d1ae75155-maxDepth: 3
}, {
        rfc_c81d1ae75155-impurity: gini,
        rfc_c81d1ae75155-max...

scala>
```

```
scala> val model_nikhilesh = cross_validator_nikhilesh.fit(trainingData)
22/11/23 05:03:43 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 1178.4 KiB
22/11/23 05:03:46 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 2.3 MiB
22/11/23 05:03:46 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 2.3 MiB
22/11/23 05:03:47 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 3.1 MiB
22/11/23 05:04:24 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 3.9 MiB
22/11/23 05:05:12 WARN org.apache.spark.storage.BlockManagerMasterEndpoint: No more replicas available for rdd_143_0 !
22/11/23 05:05:12 WARN org.apache.spark.storage.BlockManagerMasterEndpoint: No more replicas available for rdd_143_1 !
22/11/23 05:05:12 WARN org.apache.spark.storage.BlockManagerMasterEndpoint: No more replicas available for rdd_97_0 !
22/11/23 05:05:12 WARN org.apache.spark.storage.BlockManagerMasterEndpoint: No more replicas available for rdd_97_2 !
22/11/23 05:05:12 WARN org.apache.spark.storage.BlockManagerMasterEndpoint: No more replicas available for rdd_97_1 !
22/11/23 05:05:12 WARN org.apache.spark.deploy.yarn.YarnAllocator: Container from a bad node: container_1668921571476_0017_01_000002 on host: dcc-m.us-central1-c.c.steady-adapter-365922.internal. Exit status: 137. Diagnostics: [2022-11-23 23:05:12.068]Container killed on request. Exit code is 137
[2022-11-23 05:05:12.069]Container exited with a non-zero exit code 137.
[2022-11-23 05:05:12.069]Killed by external signal
.
```

```
scala> val predictions_nikhilesh = model_nikhilesh.transform(testData)
predictions_nikhilesh: org.apache.spark.sql.DataFrame = [textID: string, tweet_text_final: string ... 12 more fields]

scala> val accuracy_nikhilesh = evaluator_nikhilesh.evaluate(predictions_nikhilesh)
22/11/23 07:13:50 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 1791.0 KiB
accuracy_nikhilesh: Double = 0.7203810369307343

scala>
```