SMS Spam Collection Data Set

December 7, 2018

1 Part 2: SMS Spam Collection Data Set

1.0.1 Soufiane MOUTEI - Ahmed BEN SAAD

```
In [1]: from pyspark.sql import SparkSession

        spark = SparkSession.builder.appName("ML").getOrCreate()
```

```
In [2]: spark
```

```
Out[2]: <pyspark.sql.session.SparkSession at 0x105f848d0>
```

Let's load the data using Spark

```
In [3]: from pyspark.sql.types import StructType, StringType, StructField

        schema = StructType([
            StructField("class", StringType(),  True),
            StructField("sms", StringType(),  True),
        ])

        df = spark.read.csv("SMSSpamCollection", sep="\t", schema=schema)
        df.show(n=5)
```

```
+-----+--------------------+
|class|                 sms|
+-----+--------------------+
|  ham|Go until jurong p...|
|  ham|Ok lar... Joking ...|
| spam|Free entry in 2 a...|
|  ham|U dun say so earl...|
|  ham|Nah I don't think...|
+-----+--------------------+
only showing top 5 rows
```

1.0.2 a) The first step is to change the text features into numeric using the suitable classes (StringIndexer, Tokenizer, StopWordsRemover, CountVectorizer, IDF, VectorAssembler).

Let's break up the sequence of strings of the SMS into words:

```
In [4]: from pyspark.ml.feature import Tokenizer

        tokenizer = Tokenizer(inputCol='sms', outputCol='words')
        tokenized_df = tokenizer.transform(df)
        tokenized_df.show(n=5, truncate=False)
```

```
+-----+------------------------------------------------------------------------------------------------
|class|sms
+-----+------------------------------------------------------------------------------------------------
|ham  |Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine
|ham  |Ok lar... Joking wif u oni...
|spam |Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to r
|ham  |U dun say so early hor... U c already then say...
|ham  |Nah I don't think he goes to usf, he lives around here though
+-----+------------------------------------------------------------------------------------------------
only showing top 5 rows
```

Let's exclude the stop words (that don't carry an important meaning) using **StopWordsRemover**

```
In [5]: from pyspark.ml.feature import StopWordsRemover
        remover = StopWordsRemover(inputCol='words', outputCol='tokens')
        tokens_filtered = remover.transform(tokenized_df)
        tokens_filtered.show(5)
```

```
+-----+--------------------+--------------------+--------------------+
|class|                 sms|               words|              tokens|
+-----+--------------------+--------------------+--------------------+
|  ham|Go until jurong p...|[go, until, juron...|[go, jurong, poin...|
|  ham|Ok lar... Joking ...|[ok, lar..., joki...|[ok, lar..., joki...|
| spam|Free entry in 2 a...|[free, entry, in,...|[free, entry, 2, ...|
|  ham|U dun say so earl...|[u, dun, say, so,...|[u, dun, say, ear...|
|  ham|Nah I don't think...|[nah, i, don't, t...|[nah, think, goes...|
+-----+--------------------+--------------------+--------------------+
only showing top 5 rows
```

Let's now convert the collection of text documents in "tokens" to vectors of token counts using **CountVectorizer**. We choose *minDF=4* as the minimum number of documents a term must appear in to be included in the vocabulary.

```
In [6]: from pyspark.ml.feature import CountVectorizer
        count_vec = CountVectorizer(inputCol='tokens', outputCol='count_features',  minDF=4)
        model = count_vec.fit(tokens_filtered)
        data = model.transform(tokens_filtered)
        data.show(5)
```

```
+-----+--------------------+--------------------+--------------------+--------------------+
|class|                 sms|               words|              tokens|      count_features|
+-----+--------------------+--------------------+--------------------+--------------------+
|  ham|Go until jurong p...|[go, until, juron...|[go, jurong, poin...|(2355,[7,11,31,62...|
|  ham|Ok lar... Joking ...|[ok, lar..., joki...|[ok, lar..., joki...|(2355,[0,24,295,4...|
| spam|Free entry in 2 a...|[free, entry, in,...|[free, entry, 2, ...|(2355,[2,13,19,30...|
|  ham|U dun say so earl...|[u, dun, say, so,...|[u, dun, say, ear...|(2355,[0,69,80,12...|
|  ham|Nah I don't think...|[nah, i, don't, t...|[nah, think, goes...|(2355,[36,134,311...|
+-----+--------------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

Now we're going to perform TF-IDF to reflect the importance of a term to a document.

```
In [7]: from pyspark.ml.feature import IDF

        idf = IDF(inputCol='count_features', outputCol='idf_features')
        idf_model = idf.fit(data)
        data = idf_model.transform(data)
        data.show(5)
```

```
+-----+--------------------+--------------------+--------------------+--------------------+-----
|class|                 sms|               words|              tokens|      count_features|
+-----+--------------------+--------------------+--------------------+--------------------+-----
|  ham|Go until jurong p...|[go, until, juron...|[go, jurong, poin...|(2355,[7,11,31,62...|(2355
|  ham|Ok lar... Joking ...|[ok, lar..., joki...|[ok, lar..., joki...|(2355,[0,24,295,4...|(2355
| spam|Free entry in 2 a...|[free, entry, in,...|[free, entry, 2, ...|(2355,[2,13,19,30...|(2355
|  ham|U dun say so earl...|[u, dun, say, so,...|[u, dun, say, ear...|(2355,[0,69,80,12...|(2355
|  ham|Nah I don't think...|[nah, i, don't, t...|[nah, think, goes...|(2355,[36,134,311...|(2355
+-----+--------------------+--------------------+--------------------+--------------------+-----
only showing top 5 rows
```

Now we'll transform the class column to 0/1 using **StringIndexer**

```
In [8]: from pyspark.ml.feature import StringIndexer

        string_indexer = StringIndexer(inputCol='class', outputCol='label')
        string_indexer_model = string_indexer.fit(data)
        data=string_indexer_model.transform(data)
        data.show(5)
```

```
+-----+--------------------+--------------------+--------------------+--------------------+-----
|class|                 sms|               words|              tokens|      count_features|
+-----+--------------------+--------------------+--------------------+--------------------+-----
|  ham|Go until jurong p...|[go, until, juron...|[go, jurong, poin...|(2355,[7,11,31,62...|(2355
|  ham|Ok lar... Joking ...|[ok, lar..., joki...|[ok, lar..., joki...|(2355,[0,24,295,4...|(2355
| spam|Free entry in 2 a...|[free, entry, in,...|[free, entry, 2, ...|(2355,[2,13,19,30...|(2355
|  ham|U dun say so earl...|[u, dun, say, so,...|[u, dun, say, ear...|(2355,[0,69,80,12...|(2355
|  ham|Nah I don't think...|[nah, i, don't, t...|[nah, think, goes...|(2355,[36,134,311...|(2355
+-----+--------------------+--------------------+--------------------+--------------------+-----
only showing top 5 rows
```

Now we'll define the columns to use in the algorithm:

```
In [9]: cols = ("count_features", "idf_features")
        print(cols)
```

```
('count_features', 'idf_features')
```

We'll split now the data into 70% for training and 30% for testing:

```
In [10]: trainData, testData = data.randomSplit([0.7, 0.3])
         trainData.cache()
         testData.cache();
```

Fially, we'll define a **VectorAssembler** to put the columns to use in the algorithm in one column:

```
In [11]: from pyspark.ml.feature import VectorAssembler

         vec_assembler = VectorAssembler(inputCols=cols, outputCol = 'features')
```

1.0.3 b) Then You shall train 4 classifiers and compare them. These are:

1. LogisticRegression,
2. DecisionTreeClassifier
3. RandomForestClassifier
4. NaiveBayes

```
In [12]: from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, Decis
         from pyspark.ml.evaluation import MulticlassClassificationEvaluator
         from pyspark.ml import Pipeline

         models = {
             "Logistic Regression": Pipeline(stages=[vec_assembler, LogisticRegression()]),
             "Random Forest": Pipeline(stages=[vec_assembler, RandomForestClassifier()]),
             "Decision Tree": Pipeline(stages=[vec_assembler, DecisionTreeClassifier()]),
```

```
        "Naive Bayes": Pipeline(stages=[vec_assembler, NaiveBayes()])
    }
    evaluator = MulticlassClassificationEvaluator(metricName='f1')

    for name, model in models.items():
        print("-- Fitting %s --" % name)
        rmodel = model.fit(trainData)
        preds = rmodel.transform(testData)
        print("F1 score: %.4f" % evaluator.evaluate(preds))
        print()
```

```
-- Fitting Logistic Regression --
F1 score: 0.9565

-- Fitting Random Forest --
F1 score: 0.8544

-- Fitting Decision Tree --
F1 score: 0.9217

-- Fitting Naive Bayes --
F1 score: 0.9726
```

Using the F1-score, we can see that **Naive Bayes** is the best algorithm among the four with an F1-score of 0.97.

1.0.4 c) For one of these classifiers, you shall tune at least one important hyper parameter using ParamGridBuilder and CrossValidator

In [13]:
```
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.tuning import ParamGridBuilder

naive = NaiveBayes()

pipeline = Pipeline(stages=[vec_assembler, naive])

paramGrid = (
    ParamGridBuilder()
    .addGrid(naive.smoothing, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
    .addGrid(naive.thresholds, [[1., 1.], [0.05, .95], [.5, .5], [0.95, .05]])
    .build()
)

crossval = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
```

```
        evaluator=evaluator,
        numFolds=5
    )

    # Run cross-validation, and choose the best set of parameters.
    cvModel = crossval.fit(trainData)

    # Predict for testData
    preds = cvModel.transform(testData)

    print("F1 score: %.4f" % evaluator.evaluate(preds))
```

```
F1 score: 0.9760
```

2 d) Conclusions: Compare and comment the obtained results (you may use a comparison table).

Let's get the best parameters of the Naive Bayes:

```
In [14]: best_params = list(cvModel.bestModel.stages[-1].extractParamMap().values())[-2:]
         print("Best smoothing is: %.2f" % best_params[0])
         print("Best threeshold is:", best_params[1])
```

```
Best smoothing is: 0.80
Best threeshold is: [0.05, 0.95]
```

Then, we're going to get the score of each parameter

```
In [15]: import pandas as pd

         smoothing = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
         threshold = [[1., 1.], [0.05, .95], [.5, .5], [0.95, .05]]

         df = pd.DataFrame()
         df["Smoothing"] = [smoothing[i//len(threshold)] for i in range(len(threshold)*len(smoot
         df["Threshold"] = threshold*len(smoothing)
         df["Score"] = cvModel.avgMetrics
```

```
In [16]: df
```

Out[16]:

	Smoothing	Threshold	Score
0	0.0	[1.0, 1.0]	0.936989
1	0.0	[0.05, 0.95]	0.941192
2	0.0	[0.5, 0.5]	0.936989
3	0.0	[0.95, 0.05]	0.916187
4	0.2	[1.0, 1.0]	0.967506

5	0.2	[0.05, 0.95]	0.972023
6	0.2	[0.5, 0.5]	0.967506
7	0.2	[0.95, 0.05]	0.945399
8	0.4	[1.0, 1.0]	0.967022
9	0.4	[0.05, 0.95]	0.971546
10	0.4	[0.5, 0.5]	0.967022
11	0.4	[0.95, 0.05]	0.945249
12	0.6	[1.0, 1.0]	0.967822
13	0.6	[0.05, 0.95]	0.971568
14	0.6	[0.5, 0.5]	0.967822
15	0.6	[0.95, 0.05]	0.944376
16	0.8	[1.0, 1.0]	0.967098
17	0.8	[0.05, 0.95]	0.972088
18	0.8	[0.5, 0.5]	0.967098
19	0.8	[0.95, 0.05]	0.943283
20	1.0	[1.0, 1.0]	0.966633
21	1.0	[0.05, 0.95]	0.972073
22	1.0	[0.5, 0.5]	0.966633
23	1.0	[0.95, 0.05]	0.941529

We can see that for some parameters, the Naive Bayes performs less than Logistic Regression without tuning. Thanks to **ParamGridBuilder** and **CrossValidator**, we were able to test the performance of 24 case of parameters and chooose the best one.