CS 660 - Data Science at Scale

Assignment 2: Spam Classification

Group: Eric Jiang, Joonsoo Park, Franco Pettigrosso, McWelling Todman

1 Implement a Spam Classifier using MapReduce (MRJob)

1. Filename: SPAMANALYZER_RESULTS.py
2. Datasource Filename: spamPOC.csv
3. see Index section for reference

2 Implement a Spam Classifier using PySpark (you may use library routines from MLlib)

3 Run, test and time on large data set

- **MRJob Framework:** Under the MRJob framework, our Naive Bayes model performed quite well from a classification standpoint. After a 70 : 30 train-test split with Laplace smoothing, the model accurately classified approximately 97% of records in the test correctly (Precision 0.8538, Recall 0.9384), while maintaining a false negative rate under 1% when the classification threshold was set to 0.99999 (calculated probability that an email is spam must exceed 0.99999 in order to be classified as spam). Performance-wise, the script took slightly over 1.5 seconds to run.

```
(base) Todman-MacBook-Pro:CS660 Mac$ time python3
SPAMANALYZER_RESULTS.py spamPOC.csv
...
Creating temp directory ...
Running step 1 of 3...
Running step 2 of 3...
Running step 3 of 3...
job output is in ...
Streaming final output from ...

"real spam 12.5"     null
"real ham 84.53947368421053"          null
"false spam 2.138157894736842"        null
"false ham 0.8223684210526315"        null
```

```
Removing temp directory ...

real        0m1.605s
user        0m1.419s
sys 0m0.167s
```

- **PySpark Framework:** Under the PySpark framework, our Naive Bayes model showed decent performance with 91% accuracy after a 80 : 20 train-test split with Laplace smoothing, resulting in a precision of 0.4439 and recall of 0.7327. Unfortunately the false negative rate was slightly higher than observed with the MRJob framework Performance-wise, the script took approximately 10 seconds to run.

4 Report experiment and accuracy results along with key observations

- **MRJob Framework:** Under the MRJob framework, the Naive Bayes Model exhibited excellent performance. Our overall accuracy rate of approximately 97% was encouraging, however most exciting was our false negative rate clocking in at under 1% (0.82%). There was one important trade off that needs to be addressed, which is the false negative rate can be further reduced by relaxing the classification threshold. When lowered from 0.99999 to 0.95 the false negative drops to 0.05%. This is important to keep in mind from a user standpoint, as it may be the case that a user is willing to take a few more false positives if it means they will be exposed to fewer false negatives.

		True Class		
		Spam	Ham	Total
Predicted Class	Spam	12.5%	2.14%	14.64%
	Ham	0.82%	84.54%	85.36%
	Total	13.32%	86.68%	$N = 1872$

- **PySpark Framework:** Under the PySpark framework, our Naive Bayes model exhibited similar, albeit somewhat less excellent performance, $Accuracy = 91\%$ (precision 0.4439, recall 0.7327). There are a few future improvements we can make: 1) Fine tuning the parameters of Naive Bayes to increase model complexity and maximize both precision and recall, 2) Trying out different threshold level (currently set at 0.5) to see if accuracy can be improved, 3) applying other algorithms to compare results. Because there are many optimizations we can still make with this model, the improvement potential is a lot higher than with the MRJob framework.

		True Class		
		Spam	Ham	Total
Predicted Class	Spam	5.62%	7.04%	12.66%
	Ham	2.05%	85.3%	87.35%
	Total	7.67%	92.34%	$N = 1121$

5 Conclusion

Our major take away from this exercise is that regardless of the framework used, this is a straightforward classification problem in which one thing matters to users: the ability of the filter to reliably identify spam, while keeping the false positive rate low enough to avoid major disruptions. In order to accomplish this, the developer must optimize around recall

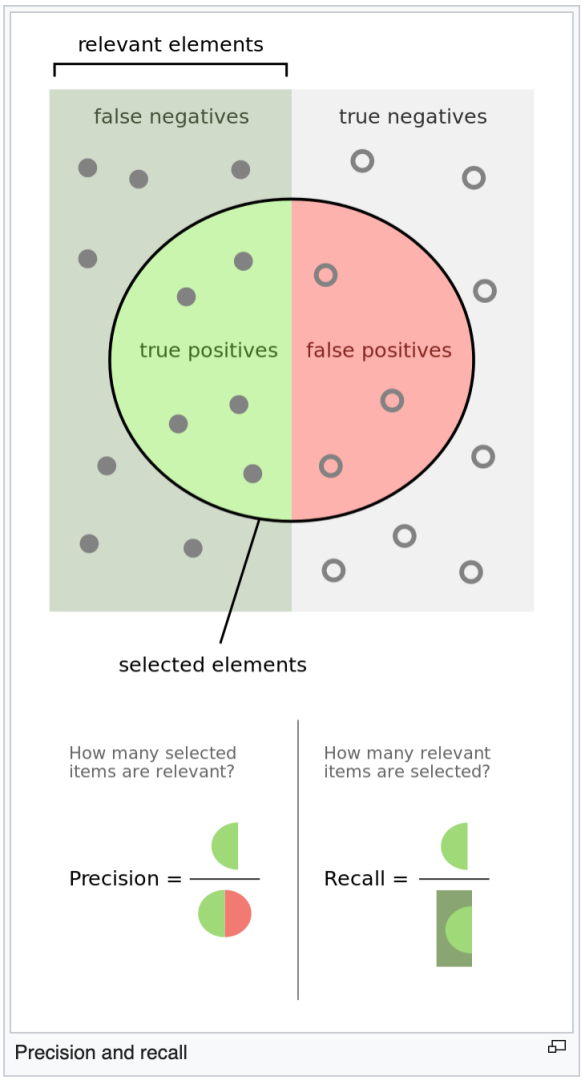

Figure 1: Precision and Recall - Source: Wikipedia

6 Index

Naive Bayes: MRJob Framework

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import math
```

```
WORD_RE = re.compile(r"[a-z|A-Z]+")
stop_words = ['ourselves', 'hers', 'between', 'yourself', 'but', 'again',
'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with',
'they', 'own', 'an', 'be', 'some', 'for','do', 'its', 'yours', 'such',
'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or',
'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below',
'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me',
'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our',
'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she',
'all', 'no', 'when', 'at', 'any', 'before', 'them', 'same', 'and',
'been', 'have', 'in', 'will', 'on', 'does','yourselves', 'then',
'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not',
'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where',
'too', 'only', 'myself', 'which','those', 'i', 'after', 'few', 'whom',
't', 'being', 'if', 'theirs', 'my','against', 'a', 'by','doing', 'it',
'how', 'further', 'was', 'here', 'than' ]


class spam_analizer(MRJob):
    def map_training_data(self, _, line):
        #file_contents = open(filepath,'r', encoding = "ISO-8859-1")
        type_of_tuple = line.split(',',1)
        holder = type_of_tuple[1].split(',', 1)
        type_of_tuple_training = type_of_tuple[0] == 'training'
        if type_of_tuple_training:
            for term in WORD_RE.findall(holder[1].strip()):
                term = term.lower()
                if term not in stop_words:
                    yield [term, holder[0], 'training'], 1
        else:
            yield [holder[1].lower(), holder[0],'confusion'], 1

    def combine_word_terms(self, term_type_matrix, count):
        yield None, [term_type_matrix[0], term_type_matrix[1]
                , sum(count), term_type_matrix[2]]

    def reducer_make_dictionary(self, _, term_type_count_matrix):
        P = [{},{},[]]
        for term, classification, count, matrix in term_type_count_matrix:
            is_training_matrix = matrix == "training"
            if is_training_matrix:
                if(classification == "ham"):
                    P[0][term] = count
                else:
                    P[1][term] = count
            else:
                P[2].append([classification, term])
```

```
        yield P, None

    def reducer_do_calculation(self, dictionaries, _):
        spam_threshold = 0.99999
        ham_count = 4825
        spam_count = 747
        ham_probability = 0
        spam_probability = 0
        ham_keys = dictionaries[0].keys()
        spam_keys = dictionaries[1].keys()
        p_spam = spam_count / (ham_count + spam_count)
        p_ham = ham_count / (ham_count + spam_count)
        tests_counts = len(dictionaries[2])
        false_positives = 0
        false_negatives = 0
        true_positives = 0
        true_negatives = 0

        for classification ,line in dictionaries[2]:
            items_raw = WORD_RE.findall(line.strip())

            for i in items_raw:
                i = i.lower()

            def not_stop_words(content):
                for i in content:
                    if i not in stop_words:
                        yield i

            joint_spam = 1
            joint_ham = 1

            for i in not_stop_words(items_raw):
                if i not in spam_keys:
                    dictionaries[1][i] = 1
                if i not in ham_keys:
                    dictionaries[0][i] = 1
                joint_spam *= (dictionaries[1][i] / spam_count)
                joint_ham *= (dictionaries[0][i] / ham_count)

            #Naive Bayes
            spam_probability = (joint_spam * p_spam) /
                ((joint_spam * p_spam) + (joint_ham * p_ham))
            ham_probability = 1 - spam_probability
            #the spam threshold regulates the sensitivity of our classifier

            #if spam_probability > spam_threshold:
            #    yield 'spam', [line, ham_probability, spam_probability]
```

```
            #else:
            #    yield 'ham', [line, ham_probability, spam_probability]

            if spam_probability > spam_threshold:
                if classification == "spam":
                    true_positives += 1
                else:
                    false_positives += 1
            else:
                if classification == "ham":
                    true_negatives += 1
                else:
                    false_negatives += 1

        yield f"real spam {(true_positives/tests_counts)*100}", None
        yield f"real ham {(true_negatives/tests_counts)*100}", None
        yield f"false spam {(false_positives/tests_counts)*100}", None
        yield f"false ham {(false_negatives/tests_counts)*100}", None




    def steps(self):
        return [
            #this is the training data
            MRStep(
                mapper = self.map_training_data,
                reducer = self.combine_word_terms
                ),
            #we need to transpose it
            MRStep(
                reducer = self.reducer_make_dictionary
            ),
            #we then need to run it against the formula
            MRStep(
                reducer = self.reducer_do_calculation
            )
            ]

if __name__ == '__main__':
    spam_analizer.run()
```

Naive Bayes: PySpark Framework

```
from pyspark.sql import  *
from pyspark.sql.functions import *
```

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import StringIndexer,
VectorAssembler,IndexToString,StopWordsRemover
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import col


# Read csv
df = spark.read.csv('/Users/ericjiang/Downloads/spam.csv', header = True )
df=df.select(df.columns[:2])

#### Transformation and cleaning
df2=df.withColumnRenamed('v1','target').withColumnRenamed('v2','email')

# Remove null values
df2=df2.where(~df2.email.isNull())

# Convert target column to numeric
df2=df2.withColumn('target',when(df2.target=='spam',1).otherwise(0))\
.withColumn('email',lower(col('email')))\
.withColumn('split_email',split(col('email'), ' '))


# Removing stop words

STOP_WORDS = ['ourselves', 'hers', 'between', 'yourself',
'but', 'again','there', 'about','once', 'during', 'out', 'very', 'having',
'with', 'they', 'own', 'an', 'be', 'some', 'for','do', 'its','yours',
'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's',
'am','or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves',
'until', 'below', 'are', 'we','these', 'your', 'his', 'through', 'don',
'nor', 'me', 'were', 'her', 'more', 'himself', 'this','down', 'should',
'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had',
'she', 'all','no', 'when', 'at', 'any', 'before', 'them', 'same', 'and',
'been', 'have', 'in', 'will', 'on', 'does','yourselves', 'then', 'that',
'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not','now',
'under', 'he', 'you', 'herself', 'has', 'just', 'where', 'too', 'only',
'myself', 'which','those', 'i', 'after', 'few', 'whom', 't', 'being',
'if', 'theirs', 'my', 'against', 'a', 'by','doing', 'it', 'how',
'further', 'was', 'here', 'than' ]


add_stopwords = STOP_WORDS
stopwordsRemover = StopWordsRemover(inputCol="split_email",
outputCol="split_email2").setStopWords(add_stopwords)
```

```
df2=stopwordsRemover.transform(df2)


# Convert target to stringindex
label_stringIdx = StringIndexer(inputCol="target",
outputCol="label")
df2_new=label_stringIdx.fit(df2).transform(df2)

# Fectorize words
hashingTF = HashingTF(inputCol="split_email2",
outputCol="rawFeatures", numFeatures=100)
featurizedData = hashingTF.transform(df2_new)


# Split train and test data
[training_data, test_data] = featurizedData.randomSplit([0.8, 0.2])
training_data.cache()
test_data.cache()

# Naive Bayes
nb = NaiveBayes(smoothing=1.0,featuresCol='rawFeatures',
labelCol='label',modelType='multinomial')
# train the model
model = nb.fit(training_data)
# select example rows to display.
predictions = model.transform(test_data)

# Evaluate with ROC
evaluator = BinaryClassificationEvaluator(labelCol="target")
print('Test Area Under ROC', 1-evaluator.evaluate(predictions))
#0.8552561538793538

# Print out confusion matrix
predictions.crosstab('target','prediction').show()
+-----------------+---+---+
|target_prediction|0.0|1.0|
+-----------------+---+---+
|                1| 79| 63|
|                0|956| 23|
+-----------------+---+---+


# Random forest
# Create an initial RandomForest model.
rf = RandomForestClassifier(labelCol="label",
featuresCol="rawFeatures",numTrees=30)

# Train model with Training Data
```

```
rfModel = rf.fit(training_data)

predictions2=rfModel.transform(test_data)


### Evaluating the model
evaluator = BinaryClassificationEvaluator(labelCol="label")
print('Test Area Under ROC', evaluator.evaluate(predictions2))
#0.9497690946496158


# Print out confusion matrix
predictions2.crosstab('target','prediction').show()
```

```
+-----------------+---+---+
|target_prediction|0.0|1.0|
+-----------------+---+---+
|                1|124| 18|
|                0|979|  0|
+-----------------+---+---+
```