Introduction

The subject of my big data project is the kaggle Quora competition. Its object is the identification of duplicate and non duplicate questions. The data set is broken into two parts. A train set with a label indicating the questions position in the data set, two questions, two integers identifying each question, and a binary target variable indicating whether the two questions are addressing the same subject. The test set is considerably larger, with only a label to indicate the features position in the feature set, followed by two questions. Unlike the train set, the greater part of the test set consists of erroneous questions without any discernible meaning. At times, questions in the train set are not even questions. Also, when valid question are included, the difference between questions is often granular enough to differ by a single word or slight change in phrasing. My efforts to classify questions in the dataset is best divided into three stages: cleaning data, analyzing data, engineering features, and classifying the results. Data was cleaned by a process of lemmatization described in the O'rielly text book "Advanced Analytics for Spark", as well as by simple filtering of punctuation and conversion of upper case letters to lower case. Both analysis of individual question's structural, and semantic relation to the broader set of questions and the question's relation to a set of topics extracted from the wikipedia data dump, acting as a background corpus, guided the classification of duplicate/non-duplicate pairs.

Cleaning Data

Initially, both the train and test data sets were not structured in a desirable manner. Although written in a comma delimited format, quoting integer values for each label, target, and question, the questions themselves contained commas, obviating easy parsing of the data set with simple splits. Equally frustrating, while most questions were relegated to a single line, a handful were mistakenly split between lines, making reliable line by line parsing of the data very difficult. Despite these inconsistencies, I was able to discern a few helpful structural regularities to aid in assembling the data into a workable test and train set. The first can be found within questions. While every question, label, and target in the data set is surrounded by a single quote, quotes within questions are surrounded by double quotes. Removing occurrences of double quotes proved a fairly simple task, providing a means of distinguishing between the start and stop of questions by the occurrence of single quotes. By extension, given the regularity of single quotes within questions, breaks between lines became distinguishable by the the presence or absence of specific character patterns common to the end of all features.

<u>Example 1 (Raw/Clean Train Sample):</u>

<u>Raw Training Question:</u>
"633","1264","1265","Should the Hobbit be considered a canonical Tolkien work?","What is the proper, canonical spelling of Middle-earth in Tolkien's works?","0"

<u>Cleaned Training Question:</u>
633,Should the Hobbit be considered a canonical Tolkien work,What is the proper canonical spelling of Middle-earth in Tolkien's works,0

Question Data

Once reformatted, the train data was transformed into six sets in preparation for feature construction. The first set of data consists of the raw features, omitting the question id fields, leaving only a single question id, question pair, and target. The Second set of data consists of lemmatized question pairs along with the id, and target for the training set - omitting characters and words found in the stopwords.txt document obtained from the Stanford core nlp package. A concious decisio was made to include a selection of punctuation symbols, such as '+', '-', and '/'. Some essential words such as "c++" were stripped of meaning without them. In addition unlike the books implementation, single characters were permitted in the lemmatized vocabulary omitting only the usual stop-words.

Example 2 (Selection Of Lemmatized Train Questions):

5,astrology capricorn sun cap moon cap rise say,triple capricorn sun moon ascendant capricorn say,1
6,buy tiago,keep childern active far phone video game,0
7,good geologist,great geologist,1
8,use instead,use instead,0
9,motorola company hack charter motorolla dcx3400,hack motorola dcx3400 free internet,0
10,method find separation slit use fresnel biprism,thing technician tell durability reliability laptops component,0
11,read find youtube comment,see youtube comment,1
12,make physics easy learn,make physics easy learn,1

The third and last transformation of the train data was performed by simply filtering out punctuation and lowering the case of all characters within each question to provide some balancing between misclassifcation involving the inclusion of punctuation in the lemmatized dataset. The fourth, fith and sixth methods involved the application of the same processing of each of the first three sets, respectively, to a list of individual questions obtained from both the train and test Question sets. These data sets were used to buid models of the question pairs. The informativeness of each transformed feature set is entirely dependent on the method used to measure similarity between quesitons. Both overlapping and set specific methods of structural and semantic evaluation were applied to each training set to build features for classification.

The Initial steps taken to clean the data can be found in the “preparation” folder.

Wikipedia Data

The wikipedia data dump contains raw XML files with title and article content from over seventeen million wikipedia articles. Cleaning of the wikipedia data was accomplished using the starategy outlined in the O'rielly Advanced Analytics text. Additional measures were taken to constrain the selection of articles from the wikipedia corpus to topics relevant to the kaggle question data set. Collection of relevant titles was accomplished in four stages.:
1) Collecting unique question terms into a set
2) Making a single pass of the Wikipedia Question data, collecting lemmatized question titles comprised of words from the test/train question corpus set
3) sub setting article titles extracted from step two by searching for an exact sequential match in one or more questions of each title to every individual question within the train/test question set
4) making another pass of the Wikipedia data dump extracting lemmatized titles and articles corresponding to the titles subset found in step 3, then assembling and saving terms and titles in a document term matrix.

The code for step 3 can be found in the getArticles folder. All dependencies and source code are compiled into a jar file in the target folder. The product of this stage is saved in a parquet file entitled filteredTitles, storing the resulting data set. The code for step 4 may be found in the folder "filter". The product of this run is stored in another parquet file entitled "termsDF".

Data Analysis

The Lemmatized question pairs served as the primary question set of interest for the initial stages of the project. The Frequency of unique terms across the combined lemmatized train/test questions constituted the initial topic of investigation. This informed selection of the vocabulary size for models of the question pairs. Due to the sparsity of the question data, with lemmatized questions averaging around 5 words in length, and having a standard deviation of approx. 3 words, term frequency and inverse document frequency provided a useful but somewhat weak measure of a term's importance alone. The following provides a summary of term frequency analysis on the lemmatized question set.

Example 3:

Total Term Count: 149,567 (Out of ~ 5,000,000 questions)

% Terms Freq < n	n = 10	n = 5	n = 3	n = 2
	64%	32%	28%	23%

There is an observable lack of frequency across all questions in the majority of terms used in the data set. It is tempting to cut the bottom percentage of terms occurring least frequently, but many cases reveal instances where infrequent terms are key points of reference for deciding the duplication of meaning between question pairs. Instead of Throwing them out, I decided to take advantage of this by weighting matches between words with the the inverse document frequency in an adaptation of jacard similarity.

Another worth while analysis conducted prior to feature construction concerns the direct comparison of questions. Simply taking the lemmatized questions pairs, converting each to a sequence, then comparing

Feature Engineering

Question Features

Three primary means of generating features were chosen to develop a feature set from question data alone:

1) normalized mean of question vectors in word2vec space
2) weighted jacard distance using ratio of idf sums from the intersection and union of words in question pairs
3) simple jacard distance measuring ratio of counts from common term intersection and the union of terms between questions

These methods were applied to each transformation of the clean training set. The Code used to create features is located in the classifier folder. All classes used to create features were compiled into the jar file "classifier-2.0.0-jar-with-dependencies.jar" in the sub-folder target. The class "Jacard" contains functions to compute the idf from a file containing a list of sentences, here the AllQuestionLemmas.txt and AllQuestions.txt Files, as well as a function to compute a simple jacard measure of two sentences. The class Word2VecTrain contains functions to generate a word2vec model from a corpus, transform question feature vectors into word2vec space, and compute the normalized average cosine distance between transformed questions.

Each of these classes is brought together in the class "RunClassifier". Executing Run Classifier builds each of the for mentioned question features, then runs them through a classifier. I chose to apply the decision tree model discussed in chapter 4. Using only these features I achieve an accuracy of aprox. 70%.

Wikipedia Features

My initial plan was to use LSA to bind articles from the wikipedia data dump relevant to questions in the question data set and build a topic model linking the vocabulary of wikipedia articles to that of questions via a similarity measure between two question's relevance to a wikipedia document. Insofar as building and binding a topic model is concerned I was successful. Unfortunately the computational complexity of the solution I have worked out so far is less than desirable for building 400,000 feature variables for the train questions within a a reasonable allotment of time. Nonetheless, the results I obtained by applying my model to a selection of cases yielded some interesting results.

The Process of obtaining data for the model is divided between separate project builds described in the "Clean Data" section above. The next step beyond obtaining the data involves building a low rank representation of the resulting document-term matrix, then querying the resulting decomposition $U*S*V^T$. The AAS text and github code was modified to provide an acceleration (from 30 (s) to .1 (s)) to multiplication of the matrix US with a document vector u^T. It is implemented under the same name for the function used in the book "topDocsForTermQuery" in the main class of the project folder RunSVD. I used an indexed RDD of dense breeze vectors in place of the book code's rowmatrix, and multiplied the transposed document vector using elementwise multiplication and a left reduce. I found this approach to be generally useful in computing the dot product of breeze vector’s and matrices. Next I implemented the function getFeaturesFromLSA to take a sequence of document scores and ids from an adaptation of the books code’s printTopDocsForDocs, renamed generateFeatures in my code, and calculated two separate normalized averages of the top documents for terms within two questions. Follwoing are some of the results I noted:

<u>Test 1 - Actual Class: 1</u>

Result 1: normalized cosine (5 docs): 1.0
Result 2: normalized cosine (3 docs): 1.0

Test 1 Input:

```
val features = generateFeatures(Seq("read", "find", "youtube","comment"), Seq("see",
"youtube","comment"))
```

Test 1 Result 1 :

```
features: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((amos
yee,480.66225853554744), (youtube,470.54899281509734), (mtv,429.653040966128), (social
media,413.10680130094767),
(twitter,331.13768221954354)),ArrayBuffer((youtube,454.0919497466907), (amos
yee,443.22645558809364), (mtv,420.43792291995015), (social media,336.9931833333198),
(twitter,302.47746835747273)),1.0)
```

Test 1 Result 2 :

```
val features = generateFeatures(Seq("read", "find", "youtube","comment"), Seq("see",
"youtube","comment"))
features: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((amos
yee,480.66225853554744), (youtube,470.54899281509734),
(mtv,429.653040966128)),ArrayBuffer((youtube,454.0919497466907), (amos
yee,443.22645558809364), (mtv,420.43792291995015)),1.0000000000000007
```

<u>Test 2 - Actual Class: 0</u>

normalized cosine (5 docs): 0.27308003735142544
normalized cosine (3 docs): 0.33394055032768827

Test 2 Input :

```
val features2 = generateFeatures(Seq("step", "step", "guide","invest", "india", "share", "market"),
Seq("step", "step", "guide","invest", "share", "market"))
```

Test 2 Result 1 :

```
features2: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((bmtc
route,5703.856092232246), (india cricket world cup,1969.600976874869), (kashmir
conflict,1872.7356090180956), (british raj,1753.6885725870266), (economy
india,1660.9863993350964)),ArrayBuffer((bmtc route,5626.605037101496), (hedge
fund,927.910528925824), (subprime mortgage crisis,913.3155783771828), (exchange-traded
fund,869.9138106820249), (monopoly,685.4620290895868)),0.27308003735142544)
```

Test 2 Result 2 :

```
features2: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((bmtc route,5703.856092232246), (india cricket world cup,1969.600976874869), (kashmir conflict,1872.7356090180956)),ArrayBuffer((bmtc route,5626.605037101496), (hedge fund,927.910528925824), (subprime mortgage crisis,913.3155783771828)),0.33394055032768827)
```

Test 3 - Actual Class: 0

```
normalized cosine (5 docs): 0.5463439687877142
normalized cosine (3 docs): 0.5231472645168105
```

```
val features3 = generateFeatures(Seq("best", "digital", "marketing", "institution", "banglore"), Seq("best", "digital", "marketing", "institute", "pune"))
```

```
features3: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((brand,871.0486408058304), (kodak,665.7966542754972), (brand awareness,516.3605799595025), (social media,499.3396275587404), (digital rights management,495.68284540473115)),ArrayBuffer((institute technology,1576.655347210678), (brand,910.0700377576435), (nagpur,718.9810805132878), (think tank,678.340702560648), (kodak,660.280011819962)),0.5463439687877142)
```

```
features3: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((brand,871.0486408058304), (kodak,665.7966542754972), (brand awareness,516.3605799595025)),ArrayBuffer((institute technology,1576.655347210678), (brand,910.0700377576435), (nagpur,718.9810805132878)),0.5231472645168105)
```

Test 4 - Actual Class: 0 Error!

```
normalized cosine (5 docs): 0.9999999999999976
normalized cosine (3 docs): 0.999999999999997
```

```
val features4 = generateFeatures(Seq("law", "change", "status", "student", "visa", "green", "card", "us", "compare", "immigration", "law", "canada"), Seq("law", "change", "status", "student", "visa", "green", "card", "us", "compare", "immigration", "law", "japan"))
```

```
features4: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((debit card,7705.677849305977), (common law,5605.809750876164), (credit card,5334.985444579405), (schengen area,3789.853793825895), (law,3646.4775327880975)),ArrayBuffer((debit card,7644.531631644547), (common law,5562.20212565562), (credit card,5338.431401216598), (schengen area,3788.2557519335137), (law,3645.3735341867614)),0.9999999999999976)
```

```
features4: (Seq[(String, Double)], Seq[(String, Double)], Double, Double) = (ArrayBuffer((debit card,7705.677849305977), (common law,5605.809750876164), (credit card,5334.985444579405)),ArrayBuffer((debit card,7644.531631644547), (common law,5562.20212565562), (credit card,5338.431401216598)),0.999999999999997)
```

Summary

I plan to pursue this work further. I am currently working on a means of providing a more scalable approach by using the outer product of the matrix US and a column vector of document vectors representing each question. While quicker results can be obtained with a much lower rank of 100 and fewer wikipedia terms, the best that I tested come from passing SVD a k value of 1000, and using 200,000 terms. The only place in which my LSA implementation does not perform well is for long questions with only a single word distance. This can be observed in the fourth test above. In general, It performs astonishingly well. All the requisite functions to run the code are in the svd folder. The necessary term-document matrix is included as well.

Sources:

General:
* Advanced Analytics with Spark, Josh Wills, Sandy Ryza, Sean Owen, Uri Larson.
* Introduction To Datamining, Pang-Ning Tan, Michael Steinback, Vipen Kumar

Word2vec:
* (Chris McCormick - Word2Vec Tutorial):
  http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
* Apache Documentation:
https://spark.apache.org/docs/2.1.0/mllib-feature-extraction.html#word2vec

Breeze Library Intro:
* https://github.com/scalanlp/breeze/wiki/Quickstart

Mllib:
http://spark.apache.org/docs/latest/ml-guide.html

Using spark-submit:
https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-submit.html

Stanford NLP
https://stanfordnlp.github.io/CoreNLP/index.html

Wiki Data Dump
https://meta.wikimedia.org/wiki/Data_dumps