Principal Text 01-2 Final Report

Aubree Larson

[aubree-larson@uiowa.edu](mailto:aubree-larson@uiowa.edu)

Conner Buchanan

[conner-buchanan@uiowa.edu](mailto:conner-buchanan@uiowa.edu)

Qianyu Ma

[qianyu-ma@uiowa.edu](mailto:qianyu-ma@uiowa.edu)

Zhounan Wang

[zhounan-wang@uiowa.edu](mailto:zhounan-wang@uiowa.edu)

May 3, 2019

# Abstract

We performed textual analytics scoring and validation for Principal Global Investors (PGI). We annotated text, engineered features, and built models that predict relevance based off our engineered features. These models will aid analysts in timely and thorough analysis of large data sets of publicly traded securities. The four financial topics that we analyzed were opportunity, growth, stability, and strategy. We engineered features to add the top 10 most relevant words from the training set, the top 5 most common occurring pairs of words from the data set, weighted importance, and a column indicating if there was a question mark. We built a Classification Tree, Neural Network, Random Forest, K-Nearest Neighbors (KNN), and an ADA Boost model for each of the topics. We used AUC values to determine how well the models were performing against one another. The best performing model for growth was a Neural Network model, the best for opportunity was a Classification Tree model, the best for strategy was a Random Forest model, and the best for stability was a Neural Network model. We recommend that PGI moves to the prototype phase focusing simply on the Random Forest model because its average AUC value across all four topics was higher than the other four models. We also recommend that PGI adds observations to their data sets so that their Random Forest model can react accurately to variability in unstructured data.

# Table of Contents

1. Abstract
2. Table of Contents
3. Executive Summary
4. Exploratory Analysis
   1. Data Exploration
   2. Data Cleaning
5. Feature Engineering and Selection
   1. Word Counts
   2. Cooccurrence Matrices
   3. Own Knowledge
   4. Correlation Matrix
6. Modeling
   1. Pre-Modeling
   2. Classification Trees
   3. Random Forest
   4. Neural Network
   5. Center Based Clustering
   6. ADA Boost
   7. K Nearest Neighbors
7. Results
   1. Growth
   2. Opportunity
   3. Strategy
   4. Stability
8. Conclusions and Recommendations
   1. Challenges
   2. Key Takeaways
   3. Recommendations

# **3 Executive Summary**

This project is being completed for Principal Financial Group located in Des Moines, Iowa. Principal has four lines of business: Retirement and Income Solutions (RIS), Principal International (PI), US Insurance Solutions (USIS), and Principal Global Investors (PGI). For the duration of the project we will be consulting with Principal Global Investors. The Principal Global Investors team is comprised of a network of investment teams whose specialties span a broad array of assets and strategies. They cater to a diverse client base, including some of the largest companies in the world.

Principal Global Investors (PGI) approached our group needing help with understanding the relevance of a variety of text towards certain financial topics. They want help creating textual classifiers that will identify topics from unstructured sources. The ability to create these identifiers will improve PGI’s ability to parse textual data and leverage it as part of their intelligence systems.

The need stems from the sheer time it takes to sift through textual data manually. Currently, their analysts are trying to read through analyst notes, stock narratives, social media, blogs, websites, emails, and various other information sources. The analysts need to be able to identify all of this information in order to predict trends in the markets and accurately advise their clients. Developing an artificial intelligence will aid analysts in timely and thorough analysis of large datasets of publicly traded securities.

PGI wanted to begin the project by scoring textual snippets to determine their relevance to a given topic as well as the sentiment of the snippet. PGI then provided the data generated from scoring the text to our group so that we could engineer features that we believed would help predict relevance. We then created processes to drop features from the data set that did not add any unique value to the model. Once the appropriate features were in the data set, we built five models for each of the four topics.

The creation of classifiers and their relationship to relevance will be discussed below. Further along in the report different types of models will be introduced and recommendations will be made with how to proceed with this project.

# **4 Exploratory Analysis**

# 4.1 Data Exploration

The first portion of our exploration was understanding the data we had. We were given data broken down by topics: Growth, Opportunity, Strategy, and Stability. Within each data set there were sheets for relevant data and non-relevant data. The relevance was determined from the results of annotating sentences through our KTM accounts. Within each sheet there are columns for sentences, features, relevance/ non-relevance, and sentiment.

PGI provided features for us, but we cannot speak to how these features were chosen. Their features are single words and they are binary variables. For example, in the Growth data set there was a column labeled “grow” and there was a 1 if the sentence in that corresponding row contained the word grow and a 0 if the sentence did not contain it.

The next data provided to us were the relevant and non-relevant columns. These columns detailed the percentage of people (represented through decimals) who ranked the sentence the given amount of relevance. The options for deciding levels of relevant were non-relevant, relevant, very relevant, and extremely relevant. If there was 0.65 listed in row 5 under the “very relevant” column, then 65% of the students who saw the sentence on their KTM account said that the sentence was very relevant for the given topic.

The sentiment columns refer to how students ranked relevant sentences through their KTM account. If a student chose that the sentence was some level of relevant (relevant, very, extremely), they were prompted with a second question. The question asked if the sentiment of the relevant sentence was negative or positive. The columns list the percentage of students (represented as a decimal) who ranked the sentiment either negative or positive.

# 4.2 Data Cleaning

Once we understood the data, we began to clean it. Cleaning the data included fixing punctuation, removing unnecessary characters, combined relevant columns, and removed sentiment columns.

We looked through sentences to see if certain symbols had gotten changed between our KTM accounts and the .csv file we were provided with. The only punctuation error we found was with the “ ‘ “ symbol. It had changed the punctuation to a combination of foreign symbols. We used the grepl function to replace the combination of the foreign symbols with the correct “ ‘ “ in the sentence column.

To remove unnecessary characters, we again combed through the data manually. The largest problem we noticed within the sentences was that many sentences began with something like “13).” To remove this beginning, we created an if statement that searched for two numbers followed by a closed parenthesis and replaced it with nothing. This effectively removed all of the numbers from the beginning of sentences without deleting relevant information such as sentences that started with years or percentages. The largest problem we noticed within the data set itself was that random columns were added when the relevant and non-relevant sheets were bound together. We had to comb through each of the data sets to determine which columns were filled with NAs and contained no other information. Once we finished this cleaning, we could start combining columns.

Since PGI stated they were most concerned with models that could correctly predict relevance, we combined the relevant, very relevant, and extremely relevant columns. This gave us a total percentage of students who ranked the sentence at any level of relevant. We then changed the column to a binary variable. The relevance column would have a value of one if the percentage of people who ranked it some level of relevance was above 60%, it would have a value of 0 otherwise.

Lastly, we took out the sentiment columns. These columns were removed because PGI expressed that they do not care to know the sentiment of a sentence. In order to remove noise from the data set we felt it was necessary to remove the positive and negative sentiment columns.

# **Feature Engineering and Selection**

After the data was cleaned we moved which is going to help us create a model that is the most successful for predicting relevance. We focused on word counts, word cooccurrence, and our own knowledge from annotating in order to add features into each data set. I will explain the process we went through for adding features to one data set instead of all four because the process is the same for each data set.

# 5.1 Word Counts

The first technique we used to add features were word counts. These word counts produced a table which listed the most commonly occurring words first and least frequent words last. We first created word counts from the sentences as they appeared in the data set, and then we created word counts after using a stemming package. The stemming package took off endings of words like -ed and -ing. We wanted to run both sets of word counts to see if the tense of the word mattered, and it did for certain topics. Since tense mattered, we only kept the non-stemmed data when running the word counts. To standardize this for any data set run through we followed a three-step process.

The first step for adding word counts was to store the current column names in a vector. We then created word counts and pulled the top ten most occurring words from the data set. We ran the top most occurring words in a loop to check them against the vector of current column names. If the most occurring word was not already in the data set, it was added as a column. The value for each corresponding sentence was either a 1 if the word appeared (no matter how many times), and a 0 if the word never occurred.

A close up of text on a white background

Description generated with very high confidence

Figure 1: The stemmed word count for the growth data set.

# 5.2 Cooccurrence Matrices

The second tool we used to engineer features for the data sets was a cooccurrence matrix. This showed us words that commonly occurred together within a certain distance. For our specific code, we wanted to identify word pairs that appeared within three words of each other. The thicker the red line between words means that the words occurred more commonly. Below is an example of a cooccurrence matrix for the growth data set.

A close up of a map

Description generated with high confidence

Figure 2: A cooccurrence matrix for words within 3 words distance for the growth data set.

We went through a three-step process to add words from the cooccurrence matrix as well. We first ran a cooccurrence matrix to identify words that commonly occurred within 3 words distance from each other. We added these word pairs to the data set in two different ways. The first way we chose to add them was identifying if the word pair appeared in consecutive words. If the words were directly next to each other, the column would populate with a value of 1. The second way we added these word pairs was to identify if the two words appeared in the same sentence at any point. If the two words appeared, no matter how far apart, the column would populate with a value of 1. We hoped that adding the pairs in these two separate ways would help us determine if word distance had an effect on predicting relevance.

# 5.3 Own Knowledge

The last tool we used to create features was our own experience from annotating. We discussed as a group patterns we noticed that didn’t appear in the word counts but that we felt were important. For example, we all experienced that passages with one or multiple question marks were usually non-relevant. These sentences were just questions from people that appeared but did not provide any relevant information. We added a column to identify if a sentence included a question mark to help predict non-relevance as the target variable.

Another issue we noticed was that not all sentences were the same length. Because of the length, some sentences naturally had more featured words. To negate this difference, we added in a feature called weighted importance. To get weighted importance, we first summed all the featured word columns for each sentence. Then, we calculated the sentence length. We took the sum of featured words divided by sentence length to give us weighted importance.

# 5.4 Feature Removal

After we engineered features for all of the data sets, we needed to determine which features to keep when moving into the modeling phase. We used a high correlation matrix to determine which features were unnecessary for the model to perform well. The correlation matrix shows which pairs of variables had a correlation above 0.75 as well as which variable in the pair could be removed. The matrix determines which can be removed by testing the variables against all others in the data set. If one of the variables in the pair had less of an effect on the model, the matrix suggests that this is the variable that should be dropped. We followed the correlation for each data set and dropped the features it provided for us.

Once the data had been cleaned, the features engineered, and the features selected, we were ready to move into the modeling phase of this project.

# **6 Modeling**

For each of the four data sets we ran five different models. We felt that these combination of five models would help us determine the best possible model to determine relevance. In order to determine model performance, we will be evaluating AUC values. An AUC value specifies the area under the ROC curve. A ROC curve is receiver operating characteristic curve and graphs true positive rate against false positive rate. The true positive rate is determined by taking the number of true positives in a model divided by the sum of true positives and false negatives in a model. The false positive rate is determined by taking the number of false positives in a model divided by the sum of the false positives and true negatives. The greater the AUC value, the better the model performed.

# 6.1 Pre-Modeling

Before we could start modeling, we had to run code on our data to make sure that it would provide accurate results during modeling. The first portion of this pre-modeling code was to center and scale the data. This meant that skewness and other outliers would be removed. This gave us a more accurate representation of what information the data provided.

After the data had been centered and scaled, we created a “stop” dictionary. This ensured that meaningless words like the, as, and if would not be taken into account for the feature engineering portion of the code.

We then split this processed data into training and testing sets. 80% of the data was binned into the training sets and the remaining 20% was placed into the testing set. We ran the word clouds, word counts, and cooccurrence matrices on the training data set. Once we had engineered these features, we added them to the testing set as well. We ran our code this way so that the features weren’t biased before they ran through the model. Adding them to the testing set later ensured that these features would be tested for accuracy without bias. The data was then ready to be run through the models.

We ran five models for each of the four data sets: classification trees, random forest, neural network, ADA Boost, and K-nearest neighbors.

# 6.2 Classification Trees

This is a basic model that we felt was necessary to include when evaluating the other models. We did not expect it to have the best performance, but it is the easiest model to interpret. It also has a very low run time so it produces model results quickly.

A classification tree functions much like a decision tree. Since we have a continuous target variable, our tree is specifically a regression tree. The tree consists of a root node, a terminal node, and branches. The root node is the first decision made in the decision tree. For example, it could be that the first decision for the Growth data set was whether or not the sentence contained the word increase. The terminal node is what predicts the outcome. In our case, the terminal node predicts whether the sentence was relevant or not. The branches contain countless other decisions that are made between the root and the terminal node.

# 6.3 Random Forest

Random Forest models are easy to interpret because you can print out the weights assigned to each of the features. The weights are determined after many decision trees have been created and an average is determined from the combination.

Random Forest regression is built off the same fundamental principles as decision trees. The “random” portion introduces a component that improves predictive performance as well as decreasing the variance of a single tree’s prediction accuracy. Since there are a plethora of decision trees created through random forest, there is often overlap between trees. Since all predictors are evaluated at each split in a tree, the trees are not entirely independent from one another. This means that correlation can be high between trees. Although the high correlation is a downfall, we still felt that Random Forest models would be an accurate model for textual analytics. Below is an example of the weights assigned to predictors for the growth data set:

A close up of a map

Description generated with very high confidence

Figure 3: A visual representation of the decision tree that the Random Forest model created with the growth data set.

# 6.4 Neural Network

A neural network functions like the human nervous system because information passes between interconnected units much like information passes between neurons in the body. The network has three layers which it uses to process information. The first layer takes raw input data, the input is then passed to hidden layers which determine connections, and then the hidden layers pass information to the last layer which produces and output.

The advantage of neural networks is the ability of the models to learn as they go. They establish weights to relationships within the hidden layers to determine how large of an effect those relationships have on the output. The weights are optimized through learning rules: least mean square error, gradient descent, conjugate gradient, newton’s rule, etc. These learning rules are used simultaneously with the backpropagation error method. Backpropagation means the backwards propagation of errors since an error is calculated after the output layer and then distributed back through the network’s layers. The errors are backpropagated so that the error at each unit is proportional to that unit’s contribution towards the total error at the output layer. The weights of each unit are then optimized to reduce total error at the output layer. The process of a neural network is shown in the graphic below:

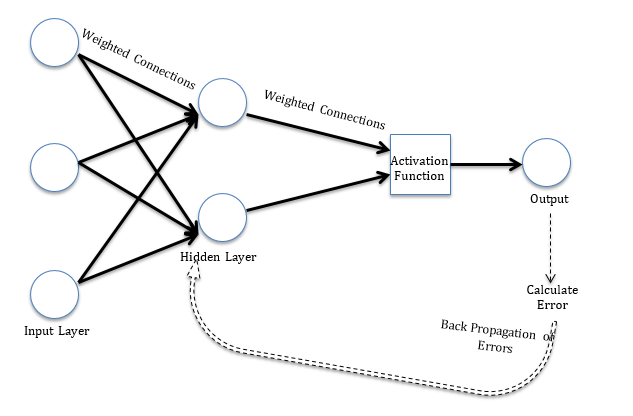


Figure 4: Visualization of a neural network (image from analyticsvidhya.com).

When picking models, we expected that the neural network would be one of the best performing models. Its ability to learn as it goes made it an easy choice to include. However, a neural network is what we call a “black box” model. Because the model learns as it goes, we cannot see the hidden decision layers within the model. If this model was run to determine if a person should be approved for a loan, it would be hard to explain how the model came to its conclusion. We included it in the models, but with caution of its interpretability.

# 6.5 ADA Boost

ADA Boost is short for adaptive boost algorithm. ADA boost models are like neural networks in the sense that they assign weight to certain points in the model. ADA boost models look at each classifier individually and calculates errors between actual relevance and predicted relevance. The model then back propagates this information into the model to assign weight to each classifier.

# 6.6 K Nearest Neighbors

The final model that we ran for each data set was a K nearest neighbors (KNN) model. KNN models use training sets to identify categories for each specific data point. The categories are determined by how similar each data point is to another. The idea is that data points within a category are more similar to each other than to data points in other categories. When the testing set is run through a KNN model, the model determines output by identifying the output of the most similar data points. That is, the testing data point is put into a category and its output is predicted based off of the known output of the existing category. We tuned our model so that each would identify the optimal value of K.

This model is easy to explain and very easy to visualize. We believed it would be a good addition to this project because it is not an ensemble method like three of our other models.

# **7 Results**

All five models were run for each of the four data sets. AUC values were calculated to determine model performance. The model with the highest AUC value predicted relevance with the highest accuracy compared to the other models. We also added in a weight to deter the models from predicting false positives. This helped tune the models while the training set was running.

# 7.1 Growth

The Neural Network model was the best performing model for the growth topic. Since the highest AUC value indicates the best performing model, it can be seen that Neural Network was the best model for predicting relevance using the Growth data set. After tuning the model to negate false positives while simultaneously maximizing AUC, the model produced an AUC value of 0.677. Random Forest was close behind the Neural Network model with an AUC value of 0.677.

Figure 5: Graph of AUC values for the models run with the growth data set.

# 7.2 Opportunity

To our surprise, the Classification Tree model was the best performing model for the opportunity topic. We included this model as a baseline, since it is the simplest model, and did not expect it to outperform the other four more complex models. The graph below shows how each of the five models’ AUC values compared to the other models. Since the highest AUC value indicates the best performing model, Classification Tree was the best model for predicting relevance using the Opportunity data set. After tuning the model to negate false positives while simultaneously maximizing AUC, the model produced an AUC value of 0.775. The Neural Network was very close to the Classification Tree model with an AUC value of 0.774.

Figure 6: Graph of AUC values for the models run with the growth data set.

# 7.3 Strategy

The best performing model for the strategy topic was the Random Forest model. The graph below shows how each of the five models’ AUC values compared to the other models. Since the highest AUC value indicates the best performing model, Random Forest was the best model for predicting relevance using the Strategy data set. After tuning the model to negate false positives while simultaneously maximizing AUC, the model produced an AUC value of 0.693. The next closest model was the Neural Network model with an AUC value of 0.65.

Figure 7: Graph of AUC values for the models run with the strategy data set.

# 7.4 Stability

The best performing model for the stability topic was the Neural Network model. The graph below shows how each of the five models’ AUC values compared to the other models. Since the highest AUC value indicates the best performing model, Neural Network was the best model for predicting relevance using the Opportunity data set. After tuning the model to negate false positives while simultaneously maximizing AUC, the model produced an AUC value of 0.95. We can see that the stability topic had the best performing models compared to the other three topics by a significant amount. We assume that this performance is due to the fact that the non-relevant sentences for stability were pulled from a different topic. This may have helped over train our models considering the lowest AUC value was 0.899.

Figure 8: Graph of AUC values for the models run with the strategy data set.

# **8 Conclusions and Recommendations**

From the models we created, Neural Network performed the best for Growth and Strategy, Classification Tree performed the best for Opportunity, and Random Forest performed the best for Stability. During the process of finding these results we faced many challenges and learned many lessons.

# 8.1 Challenges

The first challenge we faced was understanding relevant information when annotating. Although handouts were provided to inform us of the topics, we were still fairly new to most of the terminology being used. We felt that this led us to guess on some of the sentences and therefore did not provide entirely accurate data sets.

The second challenge we faced was cleaning the data topic by topic. As we mentioned before, when we bound the relevant and non-relevant data together some random columns were created. Each topic had different names for the columns created, so we could not standardize this portion of cleaning. Similar to the random column were the stop words. Some words were relevant and frequent for data sets that were irrelevant and frequent for others. This means that the stop dictionary needs to be adjusted per data set as well.

The last major challenge we faced was pre-processing the data before we ran our models. It was difficult for us to figure out how to process the data before we created folds for each of the models. Only certain models required processing, but we had to do it to all of the data so that some of our models weren’t biased.

# 8.2 Key Takeaways

As mentioned previously in this report, it is very important to spend time cleaning each data set before running it through a model. The data cleaning process cannot be standardized between data sets because textual analytics is so unique. We strongly suggest that PGI spends thorough time cleaning data sets when continuing this project.

Another takeaway that we realized was how vital writing code in the Machine Learning Pipeline structure was. Writing code in a standardized fashion allows for the opportunity to run numerous data sets through the same script with very little change.

# 8.3 Recommendations

We have two recommendations for PGI before they move out of the prototype phase of this project.

First, we recommend that PGI adds observations to their data sets. The larger the data sets are, the better they will perform. Also, the better they will react to variability when unstructured text is run through them. We feel that having the analysts perform annotations will assure accurate data sets because of their expertise.

Second, we recommend that PGI spends more time modeling, specifically with the Random Forest model. On average, the Random Forest model performed the best across all four of the financial topics. We recommend that PGI uses a Random Forest model to aid their analysts if they are only going to focus on one model.

Table 1: Performance Summary for all models across all topics.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Stability | Strategy | Growth | Opportunity | Average AUC |
| Classification Tree | 0.939 | 0.605 | 0.645 | 0.775 | 0.741 |
| Random Forest | 0.943 | 0.693 | 0.664 | 0.761 | 0.765 |
| Neural Network | 0.95 | 0.65 | 0.677 | 0.774 | 0.763 |
| KNN | 0.899 | 0.642 | 0.631 | 0.732 | 0.726 |
| ADA Boost | 0.946 | 0.646 | 0.658 | 0.769 | 0.755 |

With these recommendations, we feel that PGI will be able to build a sufficient form of artificial intelligence for their analysts. We have built a solid base of annotating and feature engineering, but they will need to focus heavily on modeling moving forward.

# Appendix

Figure 9: Words added to each of the four topics from the word count method.

Figure 10: Word pairs added to each of the four topics from the cooccurrence matrices.

Figure 11: Words dropped from each of the four topics using the correlation matrix method.