June 27, 2017 https://github.com/mkewls/buytherumor

Buy the Rumor, Sell the Fact:
A Predictive Model for the S&P 500 Index Based on "Business" Headlines

BY MICHAEL E. WILLIAMS*

I. Project Definition

Overview

How do I predict stocks, let me count the ways. Stock prediction is a domain chock full of models of all shapes and sizes. Indeed, this project is certainly not the first to use headlines to make predictions on stock prices. Academics, students, and technically-savvy investors have extensively examined the problem of using news, which I define as fact-based information describing a current event, to drive investment decisions. Whether it's pulling from the Twitter Firehose or using the front page of the Wall Street Journal, for the past decade this has been a problem many have looked to solve using machine learning (it can obviously be a very valuable solution). So why tackle the problem again in this project? First, based on our† review of publicly available models, we think there's still room for improvement. Second, this is a problem rich in data. We can easily obtain stock prices and there is a great deal of data either already in usable form or accessible through web-scraping or API's. It's a great combination for experimentation.

Problem Statement

So how can we improve on what's been done before? A good start is to use some of the best parts of models that have shown success and then extend them toward better performance. In that vein, rather than predict a particular stock price, we're going to use an aggregate or index of prices to represent the performance of the market for that day. This is something that others have done before to avoid the problem of overfitting or narrowing a model to the particulars of a specific company's stock performance. It's also a great way to apply an aggregate set of inputs to an aggregate outcome, i.e., how the market is doing on the whole for that day. Another helpful idea comes from work done in sentiment analysis, or the assignment of a mood ("happy" or "sad") or other label ("positive" or "negative") to a body of text. Rather than use classification modeling to assign a mood to the day's headlines, however, we'll instead create categories for

---

* The author provides the predictive model and analytic data contained in this report without any explicit or implied guarantee regarding the efficacy of the model or analysis for investment decisions. Markets are tricky. Don't bet the bank on any one investment tactic or strategy!

† Author assumes the royal we throughout this paper in order to sound more regal. Just kidding, but it does sound better than I and my.

positive, negative, and neutral to identify the S&P's movement. Or in other words, we'll leverage similar classification techniques to sentiment analysis to learn the S&P results as labels and then make predictions as to what a body of text equates to in terms of the S&P's movement.

With that higher level overview, let's dive a little deeper. The Standard & Poor's 500 index (S&P 500 or S&P index) is an highly-used representative for the U.S. stock market's performance and is often a benchmark against which investment strategies are measured. It is a basket of stock prices for 500 large capital U.S. stocks (equities traded on U.S. exchanges) created and maintained by a private company (Standard & Poor's). Although we're not going to build or advocate a particular investment strategy as part of this project, let's just say that if we can predict the S&P 500's close at the end of the market day relative to its open, we can make money by buying a stock (and here I mean any equity available on a public market) linked to the S&P index at a low open and selling at a high close. We're likewise going to use the S&P 500's daily performance as our label (the result we're trying to predict) for our model. We're going to start by trying to predict the actual value at which the index gains or loses for the day, which would be the most highly valuable prediction, and then abstract that by trying to predict simply whether it will gain, lose, or have neutral movement that day (so, classification not a continuous magnitude of the gain or loss).

Alright, that all sounds good, but how are we going to turn headlines into an S&P index value prediction? This brings us back to sentiment analysis. Our hypothesis is that we can train a model to learn what words or pairs of words (bi-grams) lead to a given result. We'll thus use "supervised learning," which basically is using known values (features) and results (labels) to train (construct through automated iteration) a model that is able to predict a certain label based on new features that it encounters after its training. Under the hood, this is essentially breaking down all the headlines for a particular day into their words and bi-grams, aggregating them together to create a snapshot of word and bi-gram frequency, and creating a calculus that is optimized for producing a result based on the snapshot it trained on. For example, if our model is trained on just one day of headlines and many of them have the bi-gram, "unemployment up," then if our known result for that day is a drop in the S&P 500 index, that simple model is going to have a calculus which predicts similar drops when it sees that bi-gram again.

There remains the problem, however, that headlines, especially good ones, can be deceiving. Often there are mixed ideas in the same phrase and often the news is straight-forward, lacking easy indicators like "doom" or "gloom" to guide the model's training. The hope of this project is to leverage a large set of headline data, already categorized, to assist the learning process and thus improve our predictions.‡ Even slicing-off the "business" category headlines, we're going to be able to use more than 115,000 headlines over a six-month period to build our models. This large data set from myriad news sources will help us address the bias-variance tradeoff, ideally by not

---

‡ Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

creating too high a bias (a higher bias is an overly generalized model produced by missing relevant relationships) and importantly avoiding high variance (which is a model that is not generalized for data and works mostly for the data on which it trained, produced by fitting too closely to a noisy data set).

Metrics

As this is a classification exercise, we'll look primarily at the F1 score of the model, but also its general accuracy. An F1 score is simply another measure of accuracy, with accuracy defined as the average number of correct predictions. F1 extends that calculation by multiplying the model's precision by its recall, dividing that by the sum of those two measures, and multiplying by two to create a 0 to 1 scale of performance.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

To better define those terms: Precision can be defined as all correct predictions of a result over all predictions of the result made (e.g., correctly predicted gains over all the gain predictions made, even where a loss occurred). Recall can be defined as all correct predictions of a result made over all actual occurrences of that result (e.g., correctly predicted gains over all actual gains). In both cases, general accuracy and F1, we'll be able to get a sense of how well our classification models made predictions, which is certainly useful to measure our performance. For both regular accuracy and F1, the closer to 1 we get, the better the model.

II. Analysis

Data Exploration

Let's look first at the headlines data set, which is a massive collection of over 420,000 timestamped headlines from March 2014 to August 2014, consuming several categories of news from sources all across the web. A big thanks to the University of California, Irvine Machine Learning Repository for making this available on Kaggle. For our purposes, we're going to limit the headlines to just those categorized as business and in that process, eliminate some of the columnar information we don't need, such as the publisher. What we're left with is the following:

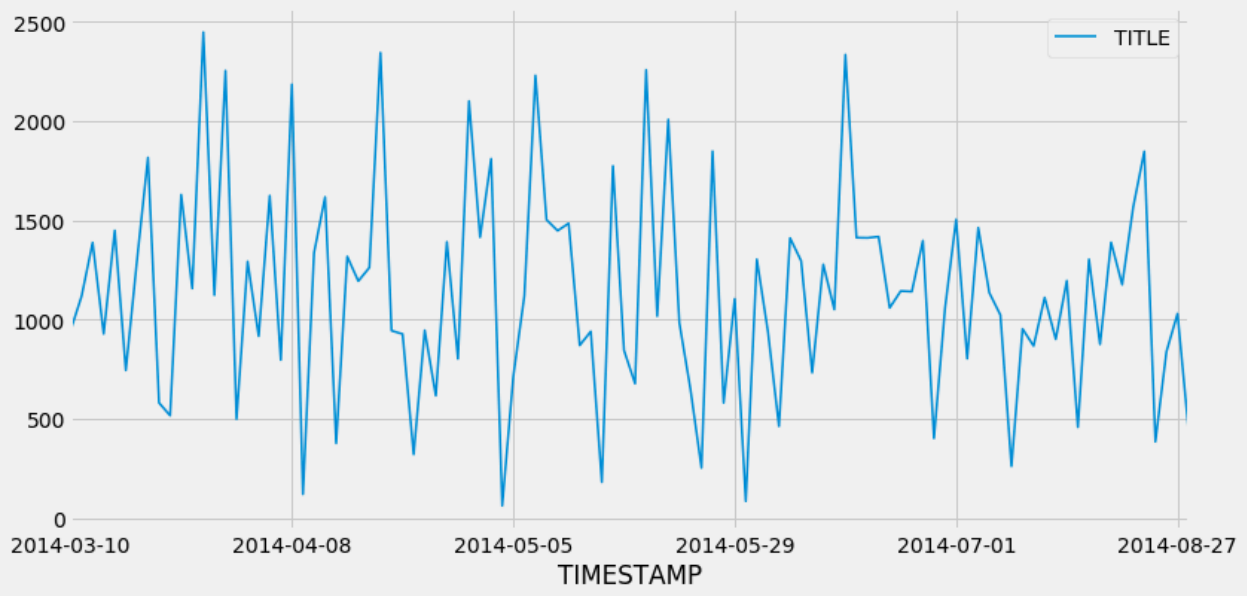
```
Feature      Number of Data Points
-------------------------------
ID           115967
TITLE        115967
TIMESTAMP    115967
```

Some examples of these headlines from the top of the data set include the following:

```
"Fed official says weak data caused by weather, should not
slow taper" "Fed's Charles Plosser sees high bar for change
in pace of tapering" "US open: Stocks fall after Fed
official hints at accelerated tapering"
```

Note that these headlines are often redundant and some are clearly not from the morning (the third one describes how the stock market opened for the day). This raises an interesting question of whether to remove redundant headlines and whether to eliminate post-market open headlines (the timestamps are Unix, so come with granular time info beyond the date). In the end, we've opted to keep all the headlines in place for our modeling. On the first issue, the redundancy may actually help us, as the frequency of significant words and bi-grams will increase (as it would in the investor psyche given an onslaught of repeated good or bad news from U.S. media). On the second issue, I consider news to be a real-time driver of the modern stock market, and likewise, if bad news comes out after the market open and causes a drop, that should be a part of our model.

A visual of the frequency of headlines by date:

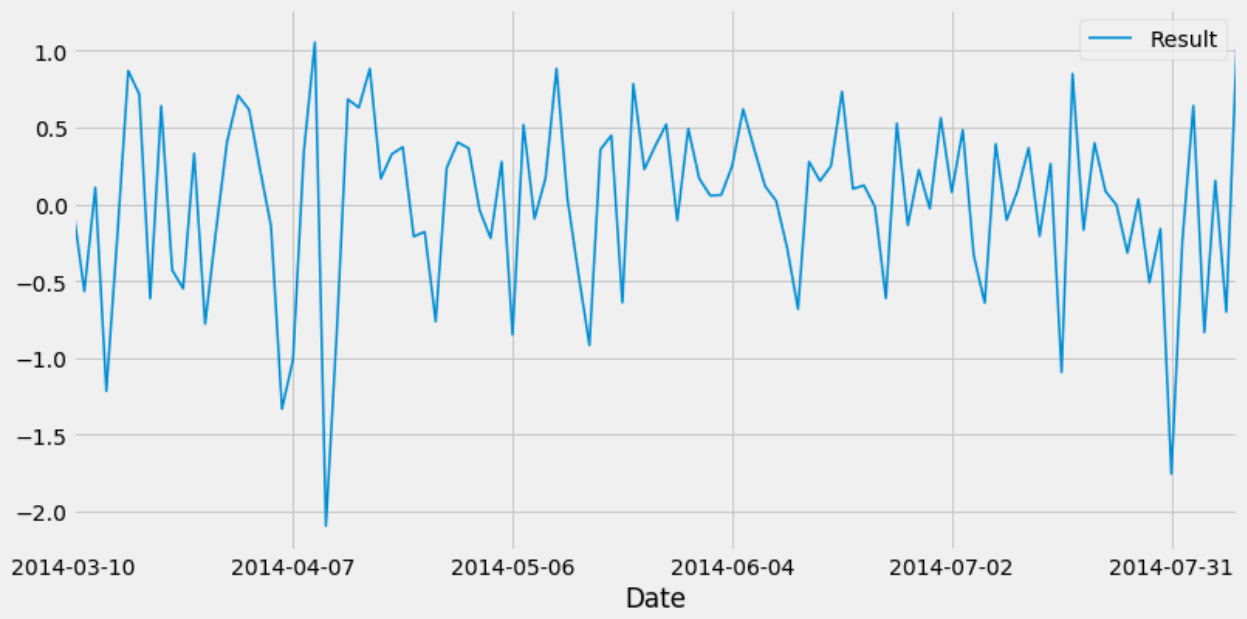



```
Total Number of Headline Dates:  102
Maximum Number of Headlines on One Day:  2449
Minimum Number of Headlines on One Day:  61
```

Looking at the visual and confirming with the statistics below it, we can see that some work is needed to even out the distribution of headlines per day. This is an important

step for two main reasons: (1) it provides our models an even distribution of per day data from which to train and test (else we could have a test set with very few or very many of one day's data), and (2) it avoids over-loading our model with one day's big headline (e.g., if the day with nearly 2500 headlines was all about Alibaba's IPO). Additionally, we know that we do not have business headlines for every day in the time period (102 days does not equal six months of headlines). That's ok, we can sample 60 headlines (one less the per day minimum) from each day to even our distribution *and* skip non-existent days when extracting labels for our headlines. Furthermore, we'll turn those pesky Unix timestamps into dates that conform with the S&P 500 indexes stringified dates, so that we can meld the two together.

A visual of the S&P 500 by date:



Modeling: Algorithms and Techniques

Although our intent is to create an effective classification model, we're going to start as part exploration, part sense-making of the data with some linear modeling. This makes sense as a starting place because it may be useful to see if such models find a correlation or accurate means of predicting the continuous results of the S&P (e.g., a 1% gain as opposed to just a "gain"). We'll start with a simple regression known as Ordinary Least Squares (OLS), and then use a somewhat more advanced regression technique known as Ridge Regression.

As helpfully noted in the scikit learn documentation, from which we'll apply the model, OLS uses coefficient assignments (represented with a *w*, which I like to think of as "weight") to optimize a linear plot by "minimiz[ing] the residual sum of squares between the observed responses . . . and the responses predicted by the linear approximation." That linear approximation is the line (curve) that our model will plot through the data points. Mathematically, the documentation shows us that this is represented as the following:

$$\min_w ||Xw - y||_2{}^2$$

Ridge steps it up a notch and optimizes its linear plot the same way as OLS plus a complexity penalty, represented by the Greek letter alpha. This penalty helps maintain a limit on the size of the *w* coefficient used to minimize the residual sum of squares. Again from the scikit learn documentation, we have a helpful mathematical representation for this algorithm clearly showing how it extends OLS and avoids huge weights:

$$\min_w ||Xw - y||_2{}^2 + \alpha||w||_2{}^2$$

The Performance Models: Classification

Whereas linear models are made for the continuous values we're seeking to produce from our headlines (a percentage gain or loss on the S&P index), the use of classification algorithms is useful both to more precisely measure our performance (x number of correct gain predictions as opposed to an error from a continuous value) and to inform trading strategies that do not necessarily require a precise numeric result, but simply a hint in the right direction. As mentioned in the project overview, this is akin to sentiment analysis in that we're bucketing our results and using the tokenized frequency of the combined headlines' words and bi-grams. Although our buckets are subject to granular modification (e.g., is a gain > 0.1% or 0.05%) and experimentation, a coherent rule is that positive values will represent positive, negative represent negative, and values at or close to 0 will represent neutral.

For our classification algorithms, we'll start with Naive Bayes, which put simply, applies Bayes' theorem to the data with an assumption of independence between the features (i.e., the presence of the bi-grams "unemployment down" and "companies hiring" is assumed to be independent). This is helpful, as we don't have a good prior probability (some probabilistic assumptions made about the features in the training data as a class) to apply in the use of Bayes' theorem. Given that we're using more than one

category, we'll apply the Multinomial Naive Bayes algorithm from scikit learn, which simply means it can output more than two classifications.

Our second classification algorithm is often used as a standard in sentiment analysis: Support Vector Machine (SVM). An SVM can be a bit difficult to succinctly describe, but essentially divides the features into groups (categories) in a multi-dimensional plot (each feature being a dimension) by applying a hyperplane that best minimizes the distance between values on each side of the plane (the decision boundary). Something known as the "kernel trick" allows for the elevation of the hyperplane to a dimension where it can be fit to an otherwise nonlinear data function. It's thus a hyper dimensional linear plot! To really make for a fun model, we'll step it up a notch and use the SVM algorithm with Stochastic Gradient Descent (SGD). SGD uses a given loss (optimization) function and applies a penalty value to that function on every iteration of its optimization by sampling some number of values from the results of that iteration to determine whether it improved the model or not. Or in other words, if the sampled results show an improvement, the model optimizes in a similar direction (e.g., increasing the weights by minimizing the penalty) and vice-versa.

For our SGDClassifier from scikit learn, we'll use the "hinge" loss function, which means we're using an SVM classification, and an "l2" penalty which grades the model weights toward 0 using the squared Euclidean norm (the distance between feature points and the decision boundary). After using this algorithm, we'll also experiment with other values, including non-SVM loss functions to see whether SVM is truly the best model with our SGDClassifier.

Benchmarks

While no true benchmark can exist for our modeling without an associated trading strategy, certain basic assumptions might be made regarding the efficacy of our algorithms:

(1) Choosing whether the S&P gains, loses, or stays neutral blindly over time with a constant execution should be equivalent to being accurate about 33% of the time.

(2) In our classification models, achieving an F1 or accuracy score above 0.33 should improve upon a coin flip investment strategy by improving our ability to arbitrage losses and gains with accurate predictions.

(3) The above may simply be adding a layer of complexity that lowers our benchmark. For example, if our random choice was simply positive or negative, we'd have to be right 50% of the time to beat a coin flip.

While the above are not incredibly useful benchmarks, our goal remains the same, beat random execution by as much as possible, which then provides the most flexibility in developing an investment strategy to produce the most gains possible.

III. Methodology

Data Preprocessing

As with most modeling, data transformation is both highly time-consuming and highly important. Here are the steps we took to preprocess the datasets, which fortunately, were clean and well-formatted.

Headlines

(1) Read and reshaped the .csv format data as a Pandas dataframe with columns for the headline's original ID, text, category, and timestamp.
(2) Extracted "business" category headlines and reformatted the timestamp from Unix time to a string in the form "YYYY-MM-DD".
(3) Sampled the data evenly over the days for which headlines existed using the minimum number of headlines as the bound, producing 60 headlines for 102 days.

```
Number of Data Points: 6120
--------------------------------
ID                                                TITLE   TIMESTAMP
1  Fed official says weak data caused by weather,...  2014-03-10
2  Fed's Charles Plosser sees high bar for change...  2014-03-10
3  US open: Stocks fall after Fed official hints ...  2014-03-10
4  Fed risks falling 'behind the curve', Charles ...  2014-03-10
5  Fed's Plosser: Nasty Weather Has Curbed Job Gr...  2014-03-10
```

S&P Data

(1) Extracted daily S&P index fundamentals from csv and reshaped to a Pandas dataframe containing the date in "YYYY-MM-DD" string form and result as float calculated from that day's open (float value) subtracted from its close divided by its open multiplied by 100 (thus a percentage gain or loss).

```
         Date    Result
2014-03-10 -0.036741
2014-03-11 -0.565950
2014-03-12  0.109848
2014-03-13 -1.215589
2014-03-14 -0.213539
```

Combined Data Set

(1) We then iterated through our headline rows and appended the result that matched the timestamp of the headline (`headline['TIMESTAMP'] == snp['Date]`). This added the day's result to the headline. We later made a separate data set that appended the following day's result, so that we could examine any change between 'day of' and 'day after' models.
(2) With the data set in good shape, we split about 80/20 between training and testing data using a time-based split of the data frame, so that we weren't mixing very similar headlines from our training set into our testing set. If we had, for instance, just used a random split function, similar headlines that carried over days, i.e., longer-lasting news stories, would likely have snuck into both training and testing data, thus creating better results that would be useless to a generalized model. We did shuffle the data following the split, although that is unlikely to have any impact on the model results.

Feature Extraction

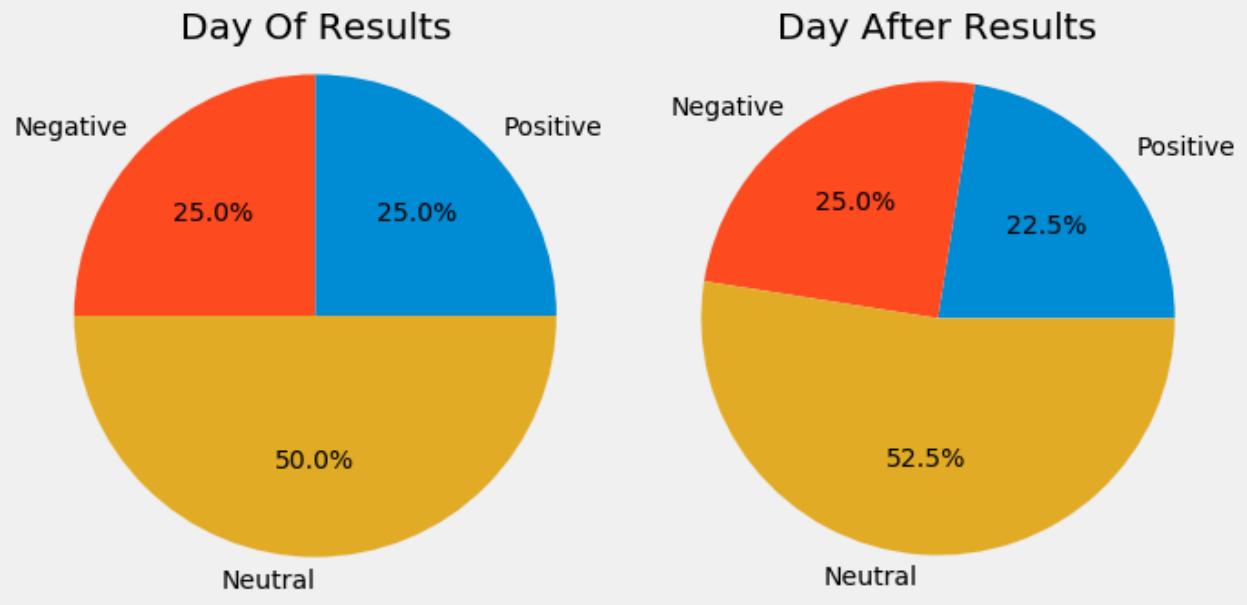
(1) To make the data useful (i.e., numeric) to our algorithms, we converted the text of the headlines in both the training and testing sets into a vectorized count of both words and bi-grams using scikit learn's CountVectorizer function. We explored using just words and just bi-grams, but the combination of both proved best for our models' performance.
(2) We then took the counts and ran them through scikit learn's TfidfTransformer function to turn the "contextless" counts into a fractional form, representative of the frequency of the word or bi-gram in the context of the training and testing data sets, respectively. The "tfidf" stands for "text frequency inverse [to] document frequency" and is a common transformation for natural language processing.

```
Training Set Size & Num. Frequencies:  (80, 25816)
Testing Set Size & Num. Frequencies:  (22, 25816)
```

Classification Categories

(1) The above feature extraction allows the linear models to train and test, but we needed to take another step and bucket our labels into categories to allow the classification models to run. To do so, we experimented by relabeling our results with varying buckets (e.g., “positive” > 0.1, “negative” < -0.1, and “neutral” >= -0.1 or <= 0.1), but decided that a bucket amounting to four-tenths of a percent around zero (0 to 0.4) for our neutral category was best. This judgment was based not only on good performance at that threshold, but because it created a roughly even split between positive and negative categories at the expense of creating a dominant neutral category. Still, for investment strategy purposes, we can imagine that a dominant neutral category would be better than a predictive outcome favoring positive (or negative) errors.

Visualization of categorization:



Implementation

Linear Model Exploration

When implementing our OLS and Ridge Regression algorithms for data exploration, two problems arose: for both “day of” and “day after” data, there were empty values found in the result labels that needed to be translated to zeroes for the algorithm to work, and specifically for “day after” data, a NaN value snuck its way into the labels

based on a quirk in how the transformation of the phase shift was interpreting a string (non-numeric) value. After fixing the preprocessing algorithms to include error checking and reassignments for these unusable values, the models worked "out of the box" without further tuning (the Ridge Regression algorithm had built-in tuning using a cross-validation method).

Classification Models

The classification models benefitted from the reformed preprocessing algorithm used with the linear models for applying the S&P results to the headlines, which ensured the category labels would all be accurate and properly formatted when converting result values to result categories. As mentioned above, we experimented with the size of the categories during the implementation and settled on the 0.0 threshold for negative results and 0.5 for positive, for the reasons already discussed. Otherwise the models worked out of the box.

Refinement

To test the multitude of parameter options for our best-performing classification model, an SVM model that uses the Stochastic Gradient Descent classifier, we used GridSearchCV, which implements a grid search cross-validation method of testing each of the various parameter values provided to the function in a matrix of models. The best performing model, as assessed by its accuracy score, is then made available as an attribute of the function's return value, among myriad other attributes of the matrix. (Note that other types of scoring could have been provided, but we stuck with accuracy as the deciding factor.) Interestingly, we also were able to use other loss functions beyond SVM in implementing GridSearchCV with the SGDClassifier, given the SGDClassifier's allowance for a variety of algorithm inputs. This was more of a curiosity than a belief that another algorithm would beat SVM, but we did look at both 'log' and 'perceptron', which are the logistic regression and the perceptron algorithm respectively. More pointedly to our refinement, we tested the following:

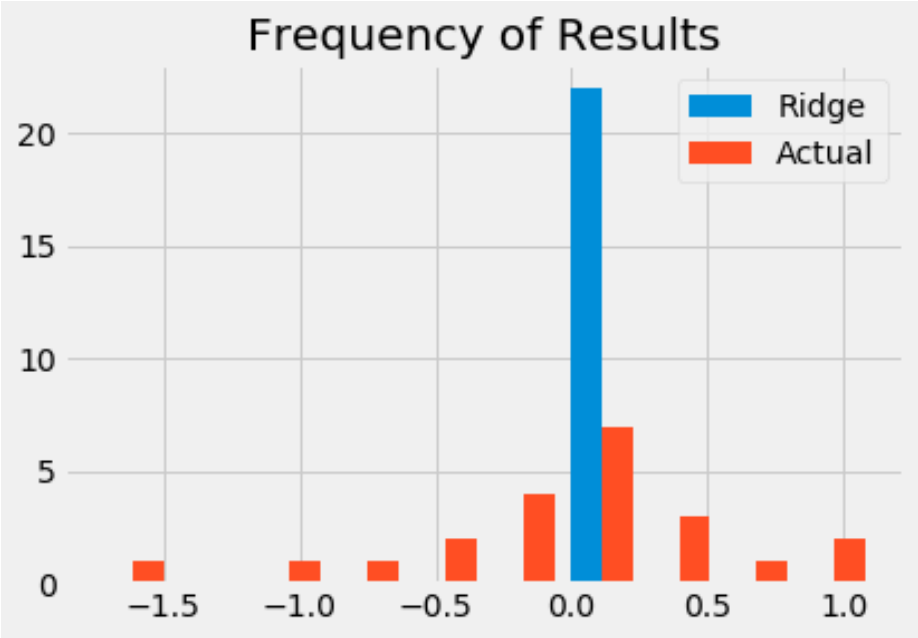
* penalty functions: 'l2' (*default*), 'l1' (*better for feature selection*), and 'none'
* alpha (*the multiplier of the penalty function*): 0.1, 0.01, 0.001, and 0.0001
* n_iter (*the number of learning iterations*): 3, 5, and 8

IV. Results

Model Evaluation and Validation

Before we get to evaluating and validating our classification models, it's worth touching upon the performance of our linear models, which we used for exploratory purposes. In sum, whether it was the OLS or Ridge Regression method, our linear models plotted a nearly horizontal line to minimize Mean Squared Error (MSE). This was a bit surprising, but it also should served as a warning of what was to come in evaluating our classification models. The below visual shows in histogram form exactly what we mean.

A histogram showing the best version of the Ridge Regression predictions. In all our linear models, the correlation of coefficient showed a correlation _worse_ than a horizontal line.



And now on to the close failure of our classification models to beat the (low bar) of performing better than random choice. Our performance can be mostly addressed by three observations: (1) The Day After method performed much better for both models (Naive Bayes & SVM) as compared to the Day Of method, (2) Although SVM performed better than Naive Bayes for Day After and thus was best overall, it performed worse in Day Of, and (3) Although several models produced aggregate F1 scores over 0.33 and thus beat our benchmark, none of our models scored an F1 score over 0.33 on

positive/negative predictions, meaning any investment strategy implemented on these models would mostly “hold” the money given a neutral categorization.

This result actually makes some sense. Our data was fairly small data, tracking only 80 training points, even though there were hundreds of features per label. Given that we could not increase the number of data points (this was, in fact, the largest data set of news headlines I could locate in public sources), we did vary the categorization threshold from the settled four-tenths of a neutral category to smaller and larger sizes for that category to validate that we had a strong prediction bias to the largest category. The results of that test proved that iby increasing the size of the positive category and decreasing the negative (e.g., leaving negative with only one-tenth of a percent, 0.0-0.1), then the model would bias toward making positive predictions.

```
================  Naive Bayes Day Of ====================
             precision    recall  f1-score   support
   Positive       0.00      0.00      0.00         4
   Negative       0.00      0.00      0.00         9
    Neutral       0.41      1.00      0.58         9

avg / total       0.17      0.41      0.24        22
Overall Accuracy: 0.41%

=================     SVM Day Of    ====================
             precision    recall  f1-score   support
   Positive       0.00      0.00      0.00         4
   Negative       0.00      0.00      0.00         9
    Neutral       0.29      0.44      0.35         9

avg / total       0.12      0.18      0.14        22
Overall Accuracy: 0.18%

================  Naive Bayes  Day After ====================
             precision    recall  f1-score   support
   Positive       0.00      0.00      0.00         3
   Negative       0.00      0.00      0.00         8
    Neutral       0.50      1.00      0.67        11

avg / total       0.25      0.50      0.33        22
Overall Accuracy: 0.50%
```

```
=================     SVM Day After    ====================
             precision    recall  f1-score   support

   Positive       0.25      0.33      0.29         3
   Negative       0.50      0.12      0.20         8
    Neutral       0.50      0.73      0.59        11

avg / total       0.47      0.45      0.41        22
Overall Accuracy: 0.45%
```

Tuning the models, namely the Day After model given its better performance, did not significantly change the results.

```
          ========  Tuned Day After SVM  ============
             precision    recall  f1-score   support

   Positive       0.00      0.00      0.00         3
   Negative       0.00      0.00      0.00         8
    Neutral       0.50      1.00      0.67        11

avg / total       0.25      0.50      0.33        22
Overall Accuracy: 0.50%

Parameters Used:
penalty: l2
loss: hinge
n_iter: 3
alpha: 0.1
```

In sum, our Day After SVM was the best model for making predictions across categories with some level of accuracy. Our other models simply chose neutral and hoped for the best. This actually worked to produce aggregate F1 scores greater than random choice, but when broken-out into categorical predictions, we can see that the model adds little value to random prediction-making.

Justification

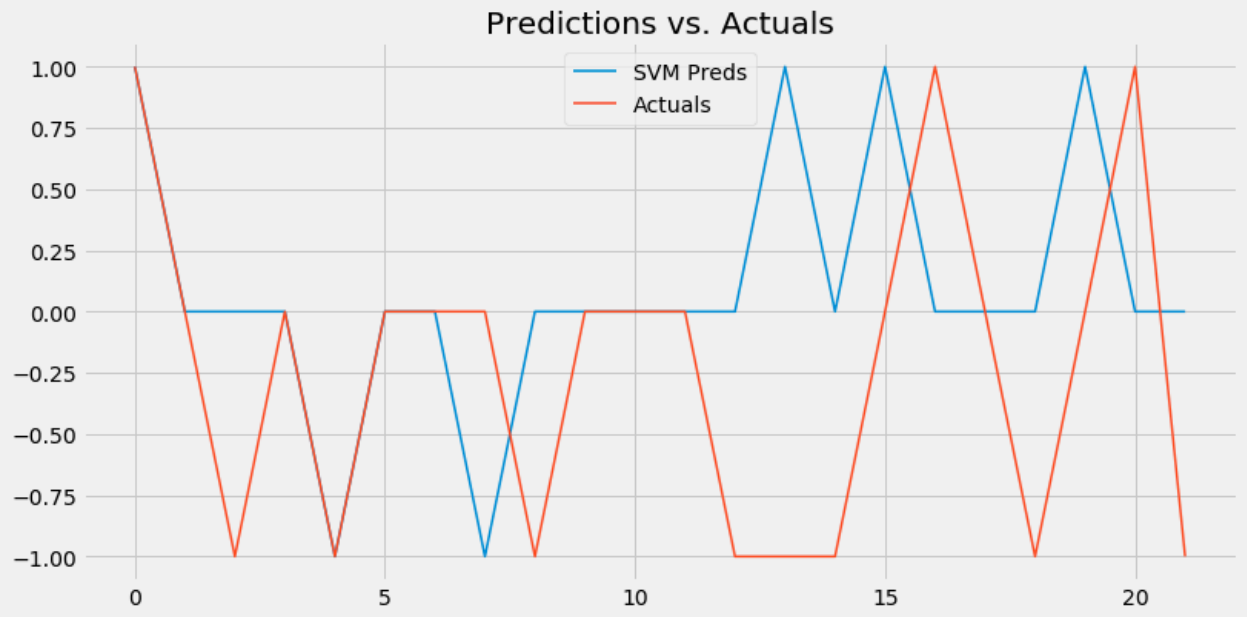
After considering the above model evaluation, iterating through modifications, validating, experimenting, and otherwise attempting to improve performance, we conclude that the model simply is not better than random choice and would not add value to an investment strategy. Or in other words, it only beats the benchmark on

aggregate and not on actual categorical predictions beyond whichever category is largest by training data. What this leads us to conclude is that choosing a constant prediction, e.g., positive, has approximately the same value as our model. That is to say, not much. What we've proven, possibly, is that: (1) News headlines lack good features for assessing a result, and (2) If they do, we probably need a lot more data points to learn from, ideally with labels that are well-distributed among categories (as opposed to heavily weighted to one, as in this case.)

V. Conclusion

Visualization

A visualization of our predicted values vs actual in our test set



The big takeaway from this visualization is that the model does appear to be flipping a coin, hitting some points correctly, but missing most. The visualization likewise supports the model's average F1 score of 0.33 for the three predictable categories. That said, there are two interesting observations to make:

(1) The model predictions performed fairly well at the outset of the testing data, getting the 1st, 2nd, 4th, 5th, and 6th predictions right, so if we stop right there, we'd have a 5 out of 6 or over 80% accuracy! But things get fuzzy after that, and with few exceptions, the predictions miss the mark. As we know, over time, even if

we guess the coin flip right several times in a row — a streak — it will eventually average out to 50% or 33% here for our three-sided coin.

(2) The trend of the model predictions appears generally negative at the outset and moving positive (with a bunch of neutral thrown in of course). In further exploration of the model's improvement, that seems like a trend we'd like to unpack at the feature level.

Reflection

Our end-to-end solution had a variety of sub-problems to solve, most of which occurred in the preprocessing stage. We extracted a very large data set, sliced it down to those features that our hypothesis included (i.e., the headlines, category, and date), and then transformed that data to play nice with our labels, namely date-specific results from the S&P. That transformation phase had a lot of moving parts, including first sampling the data set to evenly distribute our headlines and not overweight one day with way more headlines than another, breaking the headlines down into a bag of words and bi-grams, counting those words, and then assigning them a frequency proportional to all the headline words and bi-grams in our model. Finally, we got to the modeling!

In our modeling, we first explored with linear regression to examine correlations between our features and labels, which proved less than fruitful. Classification was better, but our models still had trouble optimizing on the data. The features were many, but the data was few, meaning our predictions tended toward whichever label was most prominent in the classifier. This stayed true, albeit with improved performance, when we shifted our labels by a day to produce a Day After set of models. Tuning didn't help much, as cross-validation couldn't find hyper parameters to beat our untuned Day After SVM model. But that model did hit the benchmark, albeit proving only that we could do as well as random choice.

Nevertheless, this was a valuable problem to solve, not just because of the variety of ways to explore modeling a solution, but because there was a great deal of data preprocessing both before and during the modeling implementation. It truly showed that even with very clean and well-formatted data — we used a pre-processed data set for both headlines and S&P data! — the ETL part of the data science process is extremely time-consuming. I was exposed in great detail to aspects of python's fantastic scientific libraries, from scikit-learn to pandas to numpy, in the process, learning in great detail how various methods can be leveraged to transform the data into what we need for a good model implementation. Indeed, even with all those

helpful out-of-the-box tools, I still ended up creating utility functions to aid in the preprocessing work.

Improvement

What this project set out to achieve, predicting the S&P index based on headlines, was not a success. With that result, there's certainly a lot to improve. Here are some of the most important ways we might do so:

First, get more data. I know that may sound crazy given that we began with a 420,000 headline set, but as we transformed the data, the reduction of useful data points was astounding: 420,000 to 102! That, of course, involved the merging of hundreds of data points into single points for the purpose of labeling, but even when we used separate points each having the same label (e.g., 100+ headlines all with the label of the day after S&P result), our models (obviously) made no improvement. It's effectively the same optimization problem. So when we say get more data, we do mean longitudinal data. Those 420,000 headlines only covered six months. What if we had six years? That might mean 10x more data and, potentially, a better training set for our model.

Second, consider deeper data. Or in other words, our hypothesis was that headlines carried the sentiment needed to drive investors one way or another. But what if its also the text of the article, or possibly, what commentators on cable news are saying about the headlines. If we can deepen the substance of the conversation about the news, we'll likely get a whole different world of n-grams on which to train our models and that may significantly improve our outcome.

Finally, consider other labels to the S&P 500 index. Although this really is a standard-bearer for the performance of the U.S. equity markets on any given day, the S&P is surprisingly steady. Using a label with such little variance likely helps create the bias we found in predicting those categories that had the largest number of data points. The model, understandably, gravitated toward the biggest category given the steadiness of the labels. A volatility index or some other high-variance measure might serve to better our modeling.