Predicting the SPY with Sectors

FE595 Final Project

December 18, 2020

By: Thomas Cifelli Jr., Zenya Koprowski, Katelin Prazdnik, Vasundhra Srevatsan

Purpose

The purpose of this project is to predict the movement of the SPY using the SPDR ETFs through various machine learning algorithms. The tickers/sectors that we used from SPDR are XLE - Energy Select Sector, XLU - Utilities Select Sector, XLK - Technology Select Sector, XLB - Materials Select Sector, XLP - Consumer Staples Select Sector, XLY - Consumer Discretionary Select Sector, XLI - Industrial Select Sector, XLV - Health Care Select Sector, and XLF - Financial Select Sector. The movements are classified by an upward (+1), downward (-1) or neutral (0) movement. A movement of 0.5% or less would indicate a neutral movement.

KNN Model:

The K-Nearest Neighbor Algorithm, also known as KNN, is a model that classifies data points based on the points that are most similar to it. It uses test data to make an "educated guess" on what an unclassified point should be classified as. Some positive attributes of this algorithm is that it's incredibly easy to use, it has a quick runtime, and it doesn't make any assumptions about the data, which means it's considered to be non-parametric. The model runs purely based on the data it's given instead of making any assumption about the data's structure. Some potential cons to this method are that the accuracy depends on the quality of the data, which may not be ideal, and the optimal k (number of nearest neighbors) value must be found in order to select a final model.

Going into the details of the code, "SPY_data.csv" is first read in and transformed into a dataframe of returns. From the returns, a matrix of -1,0, and 1s is built based on negative, neutral or positive returns with a neutral threshold of 0.5% in either direction. Then, the predictors and target are defined and the training and testing sets are distinguished. Next, it performs the K-Nearest Neighbors algorithm for a range of k values (1-26) to decide which k value results in the highest accuracy rate through the use of a plot, and then runs a final model with the best k value. For this case, 19 was the optimal value, therefore the final model used 19 for the number of neighbors. Finally, the script returns the Accuracy rate and Confusion matrix for the final model. The final model produced an accuracy score of 51.6%, which is decent but not amazing. This could be due to some of the cons discussed above. It's possible that the data is not ideal for working with such a model, or the best k value could actually be outside of the 1-26 range but there are too many options so it had to be limited in some way.

LDA & QDA Model:

Linear Discriminant Analysis and Quadratic Discriminant Analysis are qualitative classification methods based on Bayes' Theorem; LDA uses a linear decision boundary while QDA uses a quadratic decision boundary. The "SPY_data.csv" data was taken in and after taking the lagged difference, each percent change was classified as positive (“1”), negative (“-1”), or neutral (“0”).

LDA had a much higher accuracy than QDA which could mean that a linear decision boundary was a better fit and that QDA was too flexible of a classifier. The QDA method was also giving a “Variables are collinear” warning which means our variables were correlated which also would have resulted in a lower accuracy. We also looked at trying to increase the LDA accuracy it was giving by testing different shrinkage values for types of “lsqr” and “eigen”. A higher accuracy usually occurs with “lsqr” but we found that the accuracy is not much higher than the accuracy it was giving without using “lsqr” and shrinkage values.

Decision Tree, Random Forest, and Cross-Validation:

Decision Tree Classifiers are one of the few types of ML models that are easy to interpret the results of. Much like a series of if-else statements classification is performed by starting at the root node containing all the data and splitting it into two sub-nodes based on some kind of boolean check. Each of these sub-nodes may be split again on a separate check, and this process is repeated until all the data in a tail node is of the same class. Because the classification is done as a series of boolean checks it is very easy to start at the top and visualize the classification process, making this model particularly useful for presentations and models that will be pitched to ML-laypeople.

In order to determine how best to split at each node you start by splitting the data on any attribute and calculate the ‘entropy’, a measure of how ‘messy’ the data is. Our ultimate goal with the decision tree is to decrease entropy, so on each split we can add the entropy of the sub-nodes and subtract from the root to get that split’s ‘information gain’, a measure of how well that node splits the data. By using the split with the max information gain at each node we can optimally construct a decision tree that reduces entropy and hope that the rules determined for that reduction in the training set also apply to the testing set.

Much like the name implies, a Random Forest classifier is a collection of decision trees. The idea behind this collective model, often called an ‘ensemble model’, is that a large collection of uncorrelated weak classifiers will outperform a single more-complex model. As an example, say we have 100 decision trees that all act independently of each other, each with 60% accuracy. Any given tree has a 40% chance to be wrong on any given test, but if we aggregate the classifications of the entire population then we can be much more confident that the majority classification is the correct one. The key to this working is that the individual trees are uncorrelated, if they all fall victim to the same errors then they may as well just be a single model. To help ensure this property we can use Bootstrap Aggregation (Bagging), in which individual trees sample randomly from the entire training set with replacement so no two trees are trained on exactly the same data.

Building good decision trees and random forests is a tough task because there are so many possible options to tweak. Tree depth and splitting criteria are just a couple of the dozen possible values to tweak, and often there isn't a clear 'best choice' for a given setting. Rather than manually test hundreds of combinations, a common methodology for determining these kinds of things (known as Hyperparameter Tuning) is to use a combination of randomized and grid-based cross-validation. Both operate on the same principle: establish a wide range of possible model settings and test various combinations of them to see what makes the best model. The difference between them is that randomized cross-validation samples randomly from the possible set of settings whereas grid-based runs all possible combinations. To avoid wasting time common practice is to use randomized cross-validation to get close to good values, and then use grid-based cross validation to hone in on the best values in that area.

In practice the Decision Tree did not perform well on our dataset, averaging about 46% accuracy over a randomized 80-20 train-set split. While disappointing, this shouldn't come as too much of a surprise because if a simple decision tree was able to accurately predict market movements it would be a poster hanging in the wall of every financial classroom and office in the world. The Random Forest fared slightly better, averaging 56% accuracy over the same train-test split, but it tended to accomplish this by never predicting downward movement and almost always predicting neutral movement. Because neutral movement makes up about 55% of the dataset it's hard to call this a useful model. Much like an earthquake prediction model that always says it's safe, it may be right 99% of the time, but that 1% is really the part we care about.

Support Vector Machine:

The Support Vector Machine, also known as SVM, is another machine learning algorithm used to classify data points. The SVM uses linear algebra to create optimal hyperplanes throughout the data in order to classify each of the points. As mentioned in the introduction, the classes that are used in this project are an increase, decrease or neutral movement in daily return. These were calculated by looking at the percentage price change of the SPDR ETFs from the previous day. If the percent change was 0.5% up or down, then it was classified as a "1" or "-1" respectively. If the change was less than 0.5%, then it was classified as "0".

After running the daily returns of the SPDR ETF's onto the SPY, we saw that the SVM returned an accuracy score of 56.4%. Interestingly, the SVM confusion matrix only predicted neutral days for the SPY, and returned zero for the increase and decrease. This could be why that accuracy score is so low.

Conclusion:

Overall, we found that most of our models were able to predict above 50% of the SPY's movement. As seen in the chart included below, the SVM had the highest accuracy at 56.4% while the QDA had the lowest at 35.2%. Another interesting observation to point out is that LDA performed significantly better than QDA. This is alluding to the fact that there is a stronger linear relationship between our nine sector ETFs and SPDR S&P500 ETF Trust than there is a quadratic one. Considering that our best performing model only reached an accuracy level of 56.4% and had only predicted neutral values (none that were positive or negative), we can conclude that none of our models are truly ideal. There is definitely potential for growth, especially in LDA, Random Forest and SVM with some tweaking and further analysis. The other algorithms are probably not worth continuing to explore with regards to this dataset. That being said, part of the process of finding models that work well is finding the models that can be eliminated.

Model	**Accuracy Score**
KNN (K-Nearest Neighbors)	51.6%
LDA	55.3%
QDA	35.2%
Decision Tree	≈46% (on average)
Random Forest	≈54% (on average)
SVM (Support Vector Machine)	56.4%

Further Research:

As was said in the Conclusion, one possible area for further research is to explore the three best performing models we have found through our project: LDA, Random Forest and SVM. With more time, we could potentially find new methods and ideas to increase their accuracy. Something else that could potentially create better results is playing around with the threshold. We used between -0.5% and +0.5% as our neutral space, but if we made it smaller or larger, there's a chance our results would be improved. We could also experiment with not using a neutral range at all, and only consider positive and negative returns. While we didn't have enough time to follow through with these ideas, they would be a great starting point of taking this project one step further.