overview

January 3, 2019

0.1 # Machine Learning Project

0.1.1 Two Sigma: Using News to Predict Stock Movements

On my journey to transition from a software developer to a machine learning role, I decided to apply what I have learned so far to a Kaggle competition. This was my first attempt at a Kaggle competition and I decided to challenge myself. I chose to work through a Two Sigma challenge which involves using market and news data to predict stock movements.

There are 3 notebooks for this project. The first includes all of the EDA for both datasets. In the second, I walk through feature preprocessing, exploration and engineering. Finally, in the third notebook, I build, test and tune multiple machine learning models.

Part 1: EDA
Part 2: Features
Part 3: Modeling
Accompanying Slides

Description and Evaluation Can we use news analytics and market data to predict stock price performance? There is no doubt that the ubiquity of data today enables investors at any scale to make better investment decisions but to truly harness this power, we must be able to distinguish signal from noise.

The end result of this project is a model that predicts a signed confidence of an assets fluctuation over a ten-day window.

$$\widehat{y}_{ti} \in [-1, 1]$$

Initially, I was a little confused with the evaluation process here. Most people think of stock market predictions as being regression problems but this seemed like a binary classification problem to me. An asset either has a positive or a negative return, with the signed confidence being used to indicate both the direction and the magnitude of this move.

as mentioned above, the signed confidence interval needs to be between [-1 and 1]. Binary classification models are going to output a probability, naturally being a number between 0 and 1. So in order to get the output of my model to conform to this structure, I decided I would multiply my predicted value by 2 and then subtract 1. If the predicted probability is 0, this will cause the output to be -1 and if the predicted probability is 1, then the output will be 1.

For each day in the evaluation time period, we calculate:

$$x_t = \sum_i \widehat{y}_{ti} r_{ti} u_{ti}$$

$$score = \frac{\bar{x}_t}{\sigma(x_t)}$$

where r_{ti} is the 10-day market-adjusted leading return for day t for instrument i, and u_{ti} is a $0/1$ universe variable that controls whether a particular asset is included in scoring on a particular day.

The submission score is then calculated as the mean divided by the standard deviation of your daily x_t values:

If the standard deviation of predictions is 0, the score is defined as 0.

Data Two sources of data for this competition:

Market data (2007 to present) provided by Intrinio - contains financial market information such as opening price, closing price, trading volume, calculated returns, etc.

News data (2007 to present) Source: Thomson Reuters - contains information about news articles/alerts published about assets, such as article details, sentiment, and other commentary.

0.2 ## Summary

EDA (Part 1) Overall, I learned a lot with this Kaggle competition, perhaps most importantly, I learned how to apply my machine learning knowledge to a challenging real-world dataset. So many courses and books use datasets such as the Titanic or Iris and it was refreshing to experiment with something much more challenging.

I started by performing EDA on the dataset and identifying trends, outliers, correlations, identifying the most positively and negatively viewed companies through sentiment analysis and examining peculiar looking situations, such as stocks splits.

> "It will fluctuate" ~ J. P. Morgan

This is one of my favourite quotes related to the financial industry and it illustrates the most important and obvious thing I learned through the EDA process, everything simply fluctuates and there are rarely definitive correlations or patterns to its madness.

Feature Preprocessing, Exploration and Engineering (Part 2) I then moved on to feature preprocessing, exploration and engineering utilizing the knowledge gained during the EDA process. I ran a chi-squared test to determine if some of the categorical features were significant in relation to the ground truth labels. This proved to be somewhat of a redundant step though because I realized that when dealing with large datasets, the P-value will begin to drift towards passing a significant threshold.

Processing and feature engineering:

- Date features such as the day-of-week, month, year and quarter.
- 14, 30, 50 and 200 day moving averages and Relative Strength Indexes (RSI's)
- Standardization and Imputing
- Determining the proportion of the news item discussing the asset
- Determining the relative position of the first mention of the asset
- TF-IDF features for the headline text
- Examining features importances with random forest models

Model Tuning and Selection (Part 3) I started experimenting by training my models on a small sample of the data. I think it's important to be able to iterate quickly in the early phases and this enabled me to experiment quickly. Jeremy Howard states in his Fast.ai course that he's an advocate of iterating quickly and that he typically wants his models to run in under 10 seconds while experimenting. Since I am dealing with temporal data here, I also need to take careful steps when creating my training and validation splits. Shuffling the splits will destroy the temporal nature so the splits should be in sequence. The validation set should also come after the training set.

- Random Forrest
- I find that RF is a good starting point for most problems
- Makes almost no assumptions about the data
- Ease of use/interpretability
- Learn non-linear decision boundaries
- XGBoost
- Achieved better results than random forest
- LGBM
- Achieved better results than random forest
- Ran much faster than both of the above models

After experimenting with the models listed above, on an empirical basis, as well as running random search, I was getting the best results from LGBM.

Overall, XGBoost and LGBM performed better than Random Forest. XGBoost and LGBM both had similar results, however, training and prediction time for LGBM was much faster. For instance, fitting an LGBM model after a random grid search was 9 times faster than for XGBoost. LGBM also overfit less. So while the XGBoost model was 0.002% higher in accuracy, I think the other benefits of the LGBM model outweighed this slight increase in accuracy.

0.3 Challenges I Faced

I quickly realized that this competition was going to be quite an endeavor for my first attempt at Kaggle but I learned a lot.

Memory The datasets were big and this was a Kernel only competition which meant I was limited to one instance on Kaggle's environment.

market_train_df shape: (4072956, 16)

news_train_df shape: (9328750, 35)

This represents 13+ million records. The data was not available in CSV files either, it was loaded into the Kaggle kernel environment directly through a call to a local Two Sigma function:

```
env = twosigmanews.make_env()
(market_train_df, news_train_df) = env.get_training_data()
```

Both dataframes are stored as feather format, so after calling **get_training_data()**, 13+ million rows (7GB) were dumped into RAM and I effectively lost half of the available RAM on the instance. Manually calling the garbage collector only does so much and I realized that Python doesn't actually release all memory back to the OS after deleting objects in the Python environment, it does, however, do this for Numpy arrays. I found myself having to be very diligent to avoid running out of RAM, which happened many times.

Features There were 16 features for the market data and 35 for news. While this doesn't represent a seemingly problematic number of features, a few had very high cardinality. I also found it far more difficult to find predictive power in the news data compared to the market data.

get_training_data() Calling the Two Sigma functions were also somewhat annoying. I could only load the data once per kernel instantiation. Meaning if I accidentally made a mistake during preprocessing, I would have to restart my kernel before I could load the data in again to revert back to its original state. Same goes for loading in the test set.

0.4 Results

** _ Training Accuracy Score: 0.55_ **

** _ Validation Accuracy Score: 0.54_ **

** *Test Set Accuracy Score: 0.60 -> The test set was provided by Kaggle* **

I ended up getting a score of 60.41% on the test set provided by Kaggle, which put me in the top 50% of the competition. The discrepancy between my validation accuracy and the test set score is a result of how the test set score is calculated, which is described at the top of the notebook.

My next steps to improve my model would be to spend more time engineering features for the news dataset, experimenting with stacking multiple models and experimenting with a custom loss function (the function used to calculate the score on the test set).