Shira Rozenthal & Alex Olteanu

CMPS4010 – Milestone 2

**Predicting Volatility – Leading Kaggle Comp. Submissions**

**1st Place Submission: Nearest Neighbors**

This was the only leading solution that published a well-explained solution with the accompanying snippets of implementation. As a result, other top contenders cited this submission for inspiration, and largely derived their solutions from this initial, leading approach.

Their first order of business was to **reverse engineer time-id**, so that there is a chronological timeline in relation to price.

* They explained that this information was useful in creating a time-series cross validation, which acts as a strong proxy for leaderboard performance.

They next began **feature engineering**.

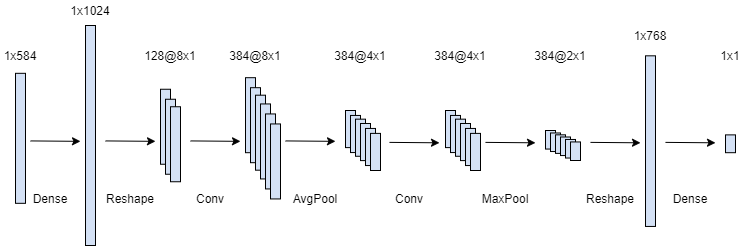
* In a random walk model, the best predictor of metric X at time T is metric X at time T-1. Accordingly, the best predictor of where volatility will realize in 10 minutes is the realized volatility (RV) of the 10 minutes that just passed.
* Accordingly, adjacent time IDs for stock XYZ are the most powerful features in their model.
* One step further, time IDs that are “nearby” can be just as powerful (think how the feature data / RV of a stock XYZ at 10am could relate to the RV of stock XYZ at 1pm, 2pm, etc.).
* They used a Nearest Neighbor model to find the **nearby N time IDs to calculate the average of features like RV and stock size**.

Next comes **feature processing**.

* With the now chronologically-sorted time IDs, they performed adversarial validation to detect if and how certain features change/evolve over time.
* They also applied np.log1p to features that have a large skew, but noted that this didn’t really impact their performance.

For the **model**, they used three simple blends for prediction: LightGBM, 1D-CNN, and MLP.

* They didn’t submit a pre-trained model in order to incorporate the test data into feature calculations (all model training was under one .ipynb file).
* Their diagram of the 1D-CNN architecture is below:



**3rd Place Submission: 300-second Model**

This submission was heavily inspired by the one described above. As a matter of fact, their solution was a mere set of modifications to the 1st-place solution. Though they provided limited descriptions of their technical approach and no code excerpts of their actual implementation, below are our takeaways from the written overview they published.

They also began by using **Nearest Neighbors on their chronological time IDs**. Key differences in their doing so from the above solution were:

* Adding a distance ration of the adjacent (1st) time ID and the kth nearest time ID.
* No time-series CV as they did not reverse-engineer the time IDs (this was noted with a sad face beside it).

Further, they included **target transformation**.

* In markets, volatility behaves as a non-stationary series (i.e. volatility levels may change over time but the behavior of volatility is likely to remain similar).
* Accordingly, they changed their target to be the ratio of the target to the realized volatility of 0~600 seconds.

Because they projected a large time interval between training and test data, they implemented a **300-seconds model** by doing the following.

1. Concatenate train + test data
2. Sliced the 600 seconds in half
3. Used the first half (0-300) to create features that would predict the volatility of the second half (300-600)
4. Used seconds 300-600 to create features that would predict the volatility for seconds 600-900

Finally, they used **macro-estimation** because they believed that an individual stock’s volatility was contingent upon the more holistic environment.

* Essentially trained a model that predicts the volatility of all stock IDs at each individual time ID.