Data Preparation and Preprocessing (1 hour)

I started the project by creating a DataFrame with all records and features from the years 2021 to 2024. In the DataFrame, I found NaN values and decided to handle them first. The DataFrame contained both numerical and categorical features, so I decided to handle each type differently. There are many ways to handle NaN values within datasets. Some common practices are disregarding records with NaN values and filling NaN values with the mean or mode of the feature. Since there were so many features, I assumed almost all the records would have at least one NaN value, meaning disregarding all records with NaN values would greatly reduce important data. As a result, I decided to fill means for numerical feature NaN values and modes for categorical feature NaN values.

When dealing with categorical features, I discovered that categorical features contained both numerical and categorical values. Therefore, I decided to split each categorical feature into two features - one for numerical values and one for categorical values (i.e. “feature285” was split into “feature285n” for numerical values and “feature285c” for categorical values). Also when splitting categorical features, I found that features “feature308c” and “feature417c” contained date information. Since we already had a variable present for date and time and those features would add an unnecessary amount of columns when one-hot encoding, I decided to disregard these features.

The goal of the project is to try to make the model lose less money or more profitable. Therefore, I decided to use supervised learning models with the “profit_loss” feature as the label (where a profit is 1 and a loss is 0). Most supervised learning models require all data to be numerical. Therefore, I decided to one-hot encode all categorical features, making them representable with numbers. With this, I was now ready to build my models.

Type 1 Features (1 hour)

When examining Type 1 features, I noticed that all records have the same value for the same date, only differing on different dates. Furthermore, the description explains this phenomenon, stating to use Type 1 features to decide whether or not to trade on a given day. As a result, I decided to ensemble a list of DataFrames, with each DataFrame being grouped by “date_time”. Next, I calculated the net profit/loss for each DataFrame to see if a given date reported a net profit or a net loss. I used this to decide whether or not to keep the DataFrame of that day - if there was a net profit that day, we should trade that day, if there was a net loss that day, we shouldn’t trade that day.

Other than distinguishing dates, Type 1 features don’t offer any other information. From the problem description, Type 1 variables are meant to be used to determine to trade on a specific day, which we have already done by grouping by “date_time”. Therefore, we are now done with Type 1 variables. I separated Type 1 features from Type 2 and Type 3 features to set the new features to just those from Type 2 and Type 3. I cleaned up the database with the new results and moved on to Type 2 features.

Type 2 Features (1 hour)

Before dealing with Type 2 features, I decided to split Type 2 and Type 3 features. In the program description, Type 2 variables are non-negative and categorical, and Type 3 variables can be positive or negative. I created the split on features based on whether the numerical

features contained any negative numbers - in which case it would be classified as a Type 3 feature. Finally, I added all categorical features to Type 2 features.

Type 2 features contain both numerical and categorical features. Since I have already converted the categorical features into numerical features with one-hot encoding, I can use a supervised machine learning model. I decided to use a neural networks model since neural networks are excellent with large datasets and complex patterns - which seem to be present in this dataset. This proved to be successful, as the model boosted approximately 94% accuracy. However, a limitation I ran across was the dataset was too large to develop a model in a reasonable amount of time. As a result, I decided to sample 5% of all records each time I tested the model (which still utilized 14,400 records). I used the SKlearn library to run the neural networks - using features such as preprocessing, decomposition, pipeline, and cross-validation to prepare the model. Finally, I stored the records predicted to be profits by the model in their own DataFrame.

Type 3 Features (3 hours)

Type 3 features had a lot of interpretation and assumptions needed. In my assumptions, I gathered that what was required from Type 3 features was the creation of filters for each feature. These features needed to be symmetric, which I assumed meant if there was a filter for a long in a feature, then there needed to be a filter for a short in a feature, and vice versa. With these assumptions, I decided the best way to find filters would be through an aspect of the decision tree algorithm.

In the decision tree algorithm, a tree is built by decisions taken while traversing through the tree, which leads you to the prediction. These decisions are calculated by sorting each feature and splitting the tree at every point. The point where the feature is split is called the threshold. At a given threshold, the left side's and right side's entropy is calculated. Entropy is calculated using the number of ones and zeros present at that record for the predicted label. The goal is to have an unbalanced number of ones compared to zeros - since the more unbalanced a result, the more impactful the split is. The decision tree algorithm takes the entropies of the left and right sides and finds the best gain with them. The maximum gain is the threshold where a split gave the most unbalanced results on both the left and right sides - meaning splitting at that threshold can help accurately predict the label. The entropy part of the decision tree algorithm is perfect for finding effective filters, which is why I decided to create my custom version of the algorithm to find filters.

Since I need to create symmetric filters I decided to use the "entry_side" feature to create two DataFrames - one with long positions and one with short positions. The limitation of time was once again met with the decision tree algorithm. To compensate, I decided to sample the Type 2 sample with 100 values but with 5 different samples to avoid underfitting. These 100 values are run through the algorithm with cross-validation to ensure the best accuracy. When working with decision trees, the max depth is limited to avoid having too many decisions to make. This is because with too many decisions, the result can be too overfitted to the model. This also applies to filters and as a result, I set the threshold for accepting a filter to 62.5% accuracy for both long and short positions, which yields approximately 2 to 6 filters depending on the sample (any lower percentage could yield too many filters, which would select too few

records i.e. 60% accuracy yields approximately 15 to 25 filters). Once these filters are found, all records in the original sample are run through the filters and stored in their own DataFrame.

Testing (2 hours)

Each time the entire solution is run from the beginning, it takes approximately 30 minutes. The time-consuming sections are when I replace NaN values for all numerical features, when I run the neural net model, and when I run the decision tree model. Since I took a sample of the data for Type 2 and Type 3 features, I decided to run the model multiple times to make sure the results were consistent and accurate.

Results

Now that the Type 2 and Type 3 models have predicted where they would profit, we can combine them and see the result. While Type 2 and Type 3 models use a sample of the dataset, they use the same dataset. This means their intersection will ensemble records that follow both models. The sample itself is created from a DataFrame that has already accounted for Type 1 features. This means the final model gives the results after applying the models for Type 1, Type 2, and Type 3 features for the 5% sample taken. The final model found that approximately 95% of its trades were profits, ultimately proving success. The total amount of time I spent on this project was 8 hours.

Limitations and Alternative Considerations

The largest limitation faced while developing my solution was the models taking too much time for a large dataset. Due to hardware constraints, I wasn't able to run the entire dataset into my models and instead had to opt for samples of the dataset. With stronger hardware or less computation-heavy models, I would be more likely to use a larger sample or the entire dataset.

An alternate consideration for this model was to use unsupervised learning or reinforcement learning. Since the goal of this solution was to make it more profitable, it made sense to select instances where the model was profitable to train on. However, another approach could have been to use the quantity of the profits, where a larger emphasis would be placed on gravitating towards large profits and avoiding large losses. Clustering algorithms such as k-means could map all the data points and try to select trades that belong in the most profitable clusters while filtering out trades that belong in the clusters displaying the largest losses. Another great model would be reinforcement learning. Reinforcement learning can utilize a reward system to decide what feature filters work best for profit, where the reward and penalty can be the "profit_loss" feature. This ensures that the model recognizes how to make a highly profitable trade (with a high reward) and how to avoid large losses (with a high penalty). As a bonus, the reinforcement learning algorithm's exploitation aspect can allow the model to adapt to new trends as markets shift over time.