jpund2ihr

December 7, 2023

1 Name: Xian Jia Le,Ben

EIDs: 56214537

Kaggle Competition: Optiver - Trading at the Close

Kaggle Team Name: Glanceman

2 CS5489 - Course Project (2023A)

Due date: See canvas site.

2.1 Possible Projects

For the course project, you may select **one** of the following competitions on Kaggle **or** define your own course project:

2.1.1 LLM - Detect AI Generated Text: Identify which essay was written by a large language model

In recent years, large language models (LLMs) have become increasingly sophisticated, capable of generating text that is difficult to distinguish from human-written text. In this competition, we hope to foster open research and transparency on AI detection techniques applicable in the real world.

This competition challenges participants to develop a machine learning model that can accurately detect whether an essay was written by a student or an LLM. The competition dataset comprises a mix of student-written essays and essays generated by a variety of LLMs.

2.1.2 Optiver - Trading at the Close: Predict US stocks closing movements

In this competition, you are challenged to develop a model capable of predicting the closing price movements for hundreds of Nasdaq listed stocks using data from the order book and the closing auction of the stock. Information from the auction can be used to adjust prices, assess supply and demand dynamics, and identify trading opportunities.

2.1.3 Linking Writing Processes to Writing Quality: Use typing behavior to predict essay quality identify which essay was written by a large language model

The goal of this competition is to predict overall writing quality. Does typing behavior affect the outcome of an essay? You will develop a model trained on a large dataset of keystroke logs that have captured writing process features.

Your work will help explore the relationship between learners' writing behaviors and writing performance, which could provide valuable insights for writing instruction, the development of automated writing evaluation techniques, and intelligent tutoring systems.

2.1.4 Child Mind Institute - Detect Sleep States: Detect sleep onset and wake from wrist-worn accelerometer data

Your work will improve researchers' ability to analyze accelerometer data for sleep monitoring and enable them to conduct large-scale studies of sleep. Ultimately, the work of this competition could improve awareness and guidance surrounding the importance of sleep. The valuable insights into how environmental factors impact sleep, mood, and behavior can inform the development of personalized interventions and support systems tailored to the unique needs of each child.

2.1.5 Enefit - Predict Energy Behavior of Prosumers: Predict Prosumer Energy Patterns and Minimize Imbalance Costs.

The goal of the competition is to create an energy prediction model of prosumers to reduce energy imbalance costs.

This competition aims to tackle the issue of energy imbalance, a situation where the energy expected to be used doesn't line up with the actual energy used or produced. Prosumers, who both consume and generate energy, contribute a large part of the energy imbalance. Despite being only a small part of all consumers, their unpredictable energy use causes logistical and financial problems for the energy companies.

2.1.6 Student-defined Course Project

The goal of the student-defined project is to get some hands-on experience using the course material on your own research problems. Keep in mind that there will only be about 4 weeks to do the project, so the scope should not be too large. Following the major themes of the course, here are some general topics for the project: - *regression* (supervised learning) - use regression methods (e.g. ridge regression, Gaussian processes) to model data or predict from data. - *classification* (supervised learning) - use classification methods (e.g., SVM, BDR, Logistic Regression, NNs) to learn to distinguish between multiple classes given a feature vector. - *clustering* (unsupervised learning) - use clustering methods (e.g., K-means, EM, Mean-Shift) to discover the natural groups in data. - *visualization* (unsupervised learning) - use dimensionality reduction methods (e.g., PCA, kernel-PCA, non-linear embedding) to visualize the structure of high-dimensional data.

You can pick any one of these topics and apply them to your own problem/data.

- *Can my project be my recently submitted or soon-to-be submitted paper?* If you plan to just turn in the results from your paper, then the answer is no. The project cannot be be work

that you have already done. However, your course project can be based on extending your work. For example, you can try some models introduced in the course on your data/problem.

Before actually doing the project, you need to write a **project proposal** so that we can make sure the project is doable within the 3-4 weeks. I can also give you some pointers to relevant methods, if necessary.
- The project proposal should be at most one page with the following contents: 1) an introduction that briefy states the problem; 2) a precise description of what you plan to do - e.g., What types of features do you plan to use? What algorithms do you plan to use? What dataset will you use? How will you evaluate your results? How do you define a good outcome for the project? - The goal of the proposal is to work out, in your head, what your project will be. Once the proposal is done, it is just a matter of implementation! - *You need to submit the project proposal to Canvas 1 week after the Course project is released.*

2.2 Groups

Group projects should contain 2 students. To sign up for a group, go to Canvas and under "People", join an existing **"Project Group X"**, where X is a number. *For group projects, the project report must state the percentage contribution from each project member. You must also submit the contribution percentages to the "Project Group Contribution" assignment on Canvas.*

2.3 Methodology

You are free to choose the methodology to solve the task. In machine learning, it is important to use domain knowledge to help solve the problem. Hence, instead of blindly applying the algorithms to the data you need to think about how to represent the data in a way that makes sense for the algorithm to solve the task.

2.4 Kaggle: Kaggle Notebooks

The Kaggle competitions have Kaggle Notebooks enabled, which provide free GPU/TPU computing resources (up to a limit). You can develop your model in the Kaggle Notebook, CS5489 JupyterHub (Dive), or on your own computers.

2.5 Kaggle: Evaluation on Kaggle

For Kaggle projects, the final evaluation will be performed on Kaggle. Note that for these competitions you need to submit your code via the Kaggle Notebook, which will then generate the submission file for processing.

2.6 Project Presentation

Each project group needs to give a presentation at the end of the semester. You will record your presentation and upload it to FlipGrid. The presentation is limited to 5 minutes. You *must* give a presentation. See the details in the "Project Presentations" Canvas assignment.

2.7 What to hand in

You need to turn in the following things.

The following files should be uploaded to "Course Project" on Canvas:

1. This ipynb file `CourseProject-2023A.ipynb` with your source code and documentation. **You should write about all the various attempts that you make to find a good solution.** You may also submit .py files, but your documentation should be in the ipynb file.
2. A **PDF** version of your ipynb file.
3. Presentation slides.
4. (Kaggle projects) Your final submission file to Kaggle. Note that most competitions require you to submit the code, and Kaggle will run it on the hidden test set.
5. (Kaggle projects) A downloaded copy of your Kaggle Notebook that is submitted to Kaggle. This file should contain the code that generates the final submission file on Kaggle. This code will be used to verify that your Kaggle submission is reproducible.

Other things that need to be turned in: - Upload your Project presentation to FlipGrid and the submit the URL to the "Project Presentations" assignment on Canvas. See the detailed instructions in the assignment. - Enter the percentage contribution for each project member using the "Project Group Contribution" assignment on Canvas. - (Student-defined projects only) submit your project proposal to the "Project Proposal" assignment on Canvas. The project proposal is due 1 week after the course project is released. Kaggle projects do not need to submit a proposal.

2.8 Grading

The marks of the assignment are distributed as follows: - 40% - Results using various feature representations, dimensionality reduction methods, classifiers, etc. - 25% - Trying out feature representations (e.g. adding additional features, combining features from different sources) or methods not used in the tutorials. - 15% - Quality of the written report. More points for insightful observations and analysis. - 15% - Project presentation - 5% - For Kaggle projects, final ranking on the Kaggle leaderboard; For student-defined projects, the project proposal.

Late Penalty: 25 marks will be subtracted for each day late.

Group contribution: marks for a group member with less than equal contribution will be deducted according to the following formula: - Let A% and B% be the percentage contributions for group members Alice and Bob. A%+B%=100% - Let x be the group project marks. - If A>B, then Bob's marks will be reduced to be: x*B/A

3 YOUR METHODS HERE

3.1 Documentation

3.1.1 First Attempt (Feature engineering)

I have first do some basic feature engineerings with the exisiting features and model them in linear regression. And the result is around **11.4** which is not ideal. The features are mainly constructed by comparing two attributes. Here is the code snippets:

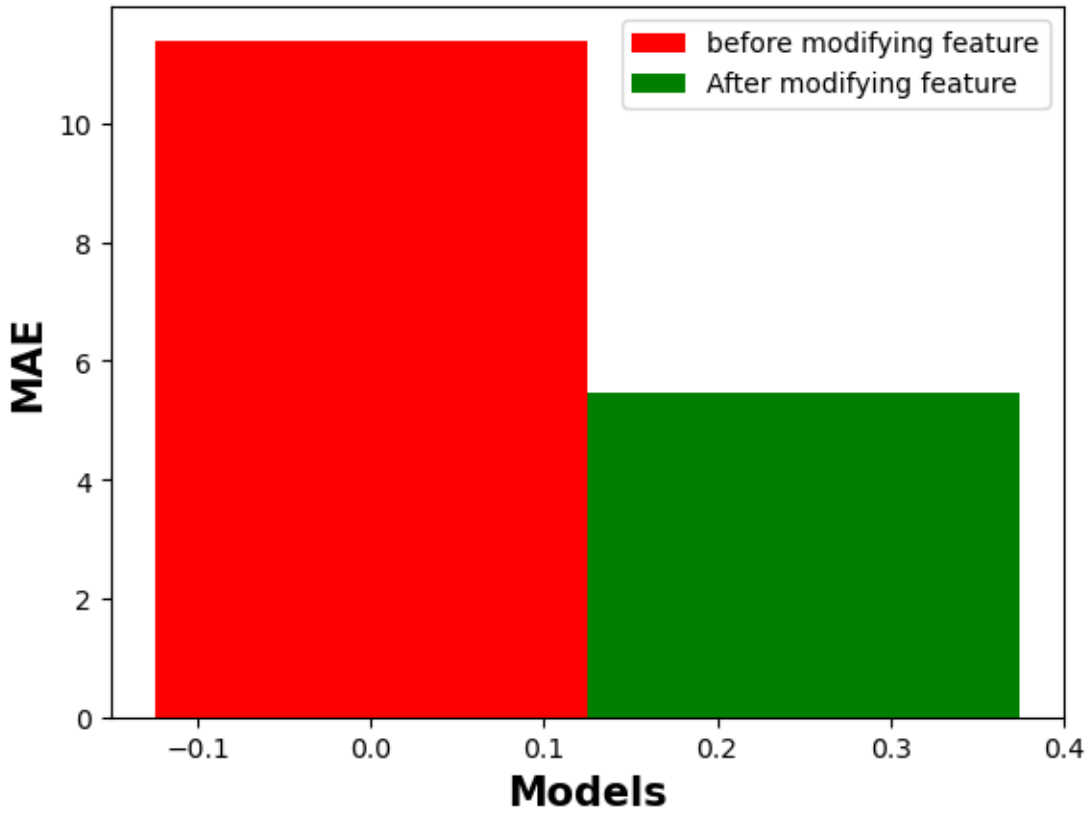
```
df['imbalance_ratio'] = df['imbalance_size'] / df['matched_size']
df['trend']=df['imbalance_size']*df['imbalance_buy_sell_flag']
df['mid_price'] = (df['ask_price'] + df['bid_price']) / 2
df['price_diff'] = df['reference_price'] - df['wap']
df["bid/ask_price_ratio"] = df['bid_price']/df['ask_price']
df["bid/ask_size_ratio"] = df['bid_size']/df['ask_size']
df['bid_ask_diff'] = df['bid_price'] - df['ask_price']
```

After that, i believed poor result is due to the features. As such, i continued to add more features. In this time, I added the combination between two different prices. And the result is boosted to **5.46** MAE score. Here is the code snippets:

```
# create different combination of imbalance prices
prices = ["reference_price", "far_price", "near_price", "ask_price", "bid_price", "wap"]
for c in combinations(prices, 2):
    df_copy[f"{c[0]}_{c[1]}_imb"] = df.eval(f"({c[0]} - {c[1]})/({c[0]} + {c[1]})")
```

[2]:
```
import matplotlib.pyplot as plt
import numpy as np

barWidth = 0.25
# different models
labelx=['Linear Regression']
trainError =[11.4]
testError=[5.46]
br1=np.arange(len(trainError))
br2 = [x + barWidth for x in br1]
plt.bar(x=br1,height=trainError, color ='r',width=barWidth,label ='before␣
 ↪modifying feature')
plt.bar(x=br2,height=testError, color ='g',width=barWidth,label ='After␣
 ↪modifying feature')
plt.xlabel('Models', fontweight ='bold', fontsize = 15)
plt.ylabel('MAE', fontweight ='bold', fontsize = 15)
plt.legend()
plt.show()
```

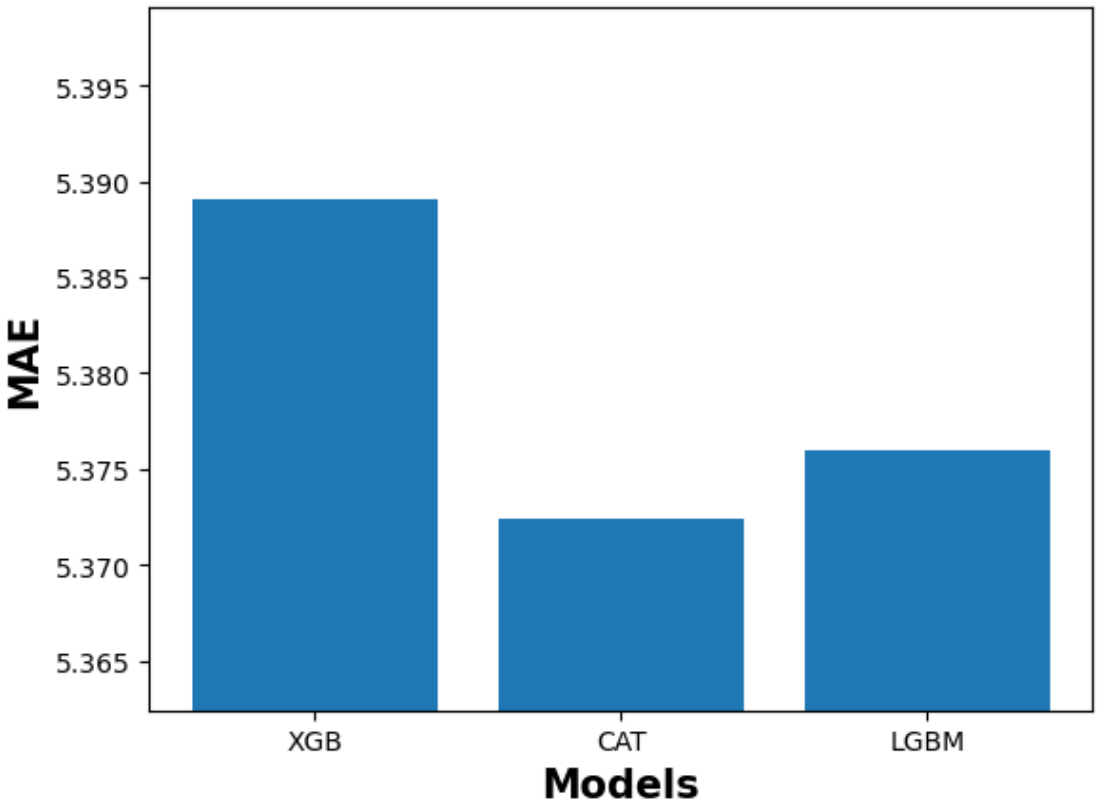


3.1.2 Second Attempt (Try different models)

After the some experiments with linear Regression, I soon realized there is no any improvement with linear regression model no matter how i modify the features. Therefore, i decided to try different models. The first model i have tried is **support vector machine regression (SVR)**. Quickly, I found out that training with SVR with cross validation is too slow which causes ran out of time limit. As such, i decided to focus on some models which allows gpu to compute and the candidate models are **LGBMRegressor (LGBM)**, **XGBoostRegressor (XGB)** ,**CatBoost(CAT)** . All the models are belong to ensemble methods. The result are as the following:

[3]:
```
barWidth = 0.25
# different models
labelx=['XGB',"CAT","LGBM"]
testError=[5.3891,5.3724,5.376]
plt.bar(labelx,testError)
plt.xlabel('Models', fontweight ='bold', fontsize = 15)
plt.ylabel('MAE', fontweight ='bold', fontsize = 15)
plt.ylim(min(testError) - 0.01, max(testError) +0.01)

plt.show()
```



Based on the above result, we can find out that Catboost perform the best here. However, there are some noticeable issues of this model. First of all, Catboost consumes a lot of memory when compared to other methods. It causes the kaggle notebook fail when using GPU. Due to this issues, Catboost is used CPU to run and also take longer time to complete the training process. Meanwhile, LGBM has the similar result of the Catboost but take less time for training. Although XGB have poor result among these two models but it run effenctly. Therefore, in the following experiments, i tend to use XGB for check the performance.

3.1.3 Third attempt (Timeseries cross validation + more features)

After the above explorations, I first decided to change the strategy of doing cross validation. Since the data is time related, doing cross-vaildation in order manner will be more representative and reason.

```
numJobs=cv
if(timeSeries==True):
    tscv = TimeSeriesSplit(n_splits=cv)
    cv=tscv
match modelType:
    case "LGBM":
            print("Model LGBM")
            paramgrid={
```

```
            "objective": ["mae"],
            "n_estimators": [750,1000],
            "num_leaves": [256,128],
            "subsample": [1],
            "colsample_bytree":[0.6],
            "learning_rate": [0.05,0.01],
        }
        model = lgb.LGBMRegressor(n_estimators= 500,random_state=4487,objective='mae',
        modelCV= GridSearchCV(model,param_grid=paramgrid,n_jobs=numJobs,verbose=True,cv
        modelCV.fit(x, y)
        print(modelCV.best_params_)
        return modelCV.best_estimator_
```

Compared to the last time, i have added time related features such as the difference bewteen price in current day and price in last day.

```
# time related feature
for col in ['matched_size', 'imbalance_size', 'reference_price', 'imbalance_buy_sell_flag']
    for window in [1, 2, 3,10]:
        df_copy[f"{col}_shift_{window}"] = df.groupby('stock_id')[col].shift(window) # kee
        df_copy[f"{col}_ret_{window}"] = df.groupby('stock_id')[col].pct_change(window) #
```

In addition to that, added some statictistical data to the record.

```
for func in ["mean", "std", "skew", "kurt"]:
    df_copy[f"all_prices_{func}"] = df[prices].agg(func, axis=1)
    df_copy[f"all_sizes_{func}"] = df[sizes].agg(func, axis=1)
```

Unfortunately, the result after adding new features become worser. Result from catboost, xgboost and LGBM are all **higher than 5.4**.

3.1.4 Forth attempt (Ensemble)

Based on the above attempt, i decided to combine mulitple LGBM models into one single models. Each model from the first 5 models learn different protion of data from all data. For instance, the first model will learn the first 25% of data, the second model will learn the second 25% of data. The last model will learn all the data. When doing perdict, the result is the mean of results from all models. I have perform different varaint onto the traning.

- **Variation 1** (5.415)
- – Dataset is divided into 5 pieces. Each of model is trained with 1 piece data. 98% data from that data is used for training and the remaining is for valiging.
- **Variation 2** (5.4088)
- – Dataset is divided into 5 pieces. First of model trained with data which is up to the first pieces. Second of model trained with data which is up to the second pieces. The following model keep the same training rule.
- **Variation 3** (5.4105)

- – More than half amount of data is used for training. Each model trains with data in increasing amount.

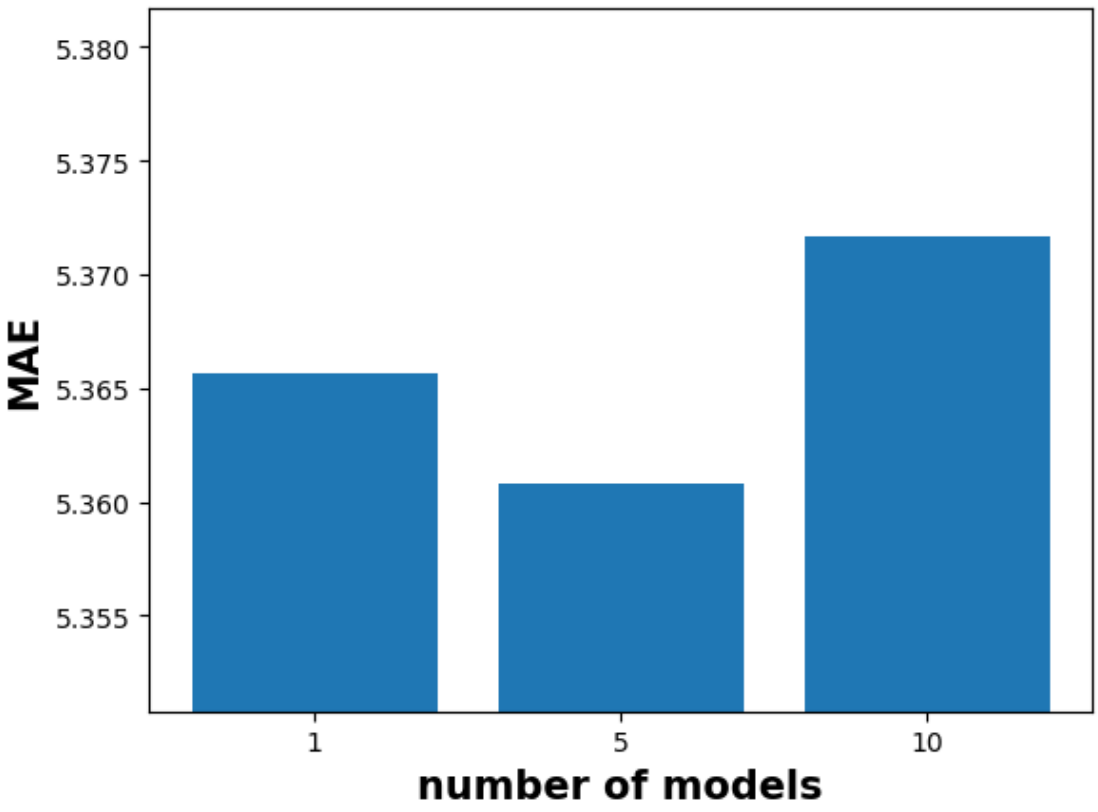
Unfortunately, the result are same with the previous one which **higher than 5.4**.

Then i decided to modify the data in predict. Since prediction is run by loop, i can collect all the data from previous loop and leave the last data. After modifying, the result is boost to average of **5.363**. Their result are as follow: - **Variation 1** (3.713) - **Variation 2** (3.630) - **Variation 3** (3.613)

After some hypertuning and small adjustment, the final score is manage to 5.3608. The following figures shows the public score with different number of LGBM models

[4]:
```
barWidth = 0.25
# different models
labelx=['1',"5","10"]
score=[5.3657,5.3608,5.3717]
plt.bar(labelx,score)
plt.xlabel('number of models', fontweight ='bold', fontsize = 15)
plt.ylabel('MAE', fontweight ='bold', fontsize = 15)
plt.ylim(min(score) - 0.01, max(score) +0.01)

plt.show()
```



3.2 Conclusions

In summary, the exploration of stock price forecasting has revealed some important insights that can guide future work in this area. The key takeaways from the analysis are as follows:

1. **Decreasing returns with additional features:**
   - The idea that adding more features does not necessarily improve accuracy emphasises the importance of feature selection. The key is to focus on the quality and relevance of features rather than the number of features. Unnecessary or irrelevant features may introduce noise and hinder model performance.
2. **Feature quality is critical:**
   - The performance of a model depends heavily on the quality of the selected features. If a feature lacks predictive power or fails to capture meaningful patterns in the data, the predictive power of the model is likely to suffer. Therefore, it is critical to carefully consider and evaluate the contribution of each feature.
3. **Results vary according to training strategy:**
   - From the ensemble attempt, using different training strategics can cause the result have significant change which is obervious than single model.

3.2.1 Reasons for unsatisfactory results:

Unsatisfactory results can be attributed to several factors: - **Inadequate quality of features:** Since lacking of domain knowledge relating to finance, it is diffcult to construct meaningful features. Random features can not give good quality. - **Overfitting or underfitting:** The model may have difficulty in generalising patterns in the training data to unseen data, resulting in poor predictive performance.

3.2.2 Future Improvement Strategies:

To address the identified shortcomings and improve the predictive ability of the model, we propose the following improvement strategies:

1. **Rolling features:**
   - Explore the construction of rolling features, including the calculation of averages or other statistics for subsets of data. This approach captures the dynamic trends and temporal patterns of stock prices, thus providing more informative features for the model.
2. **Diversify training strategies**:
   - Experiment with different training strategies, including incorporating a variety of regression models. A diverse mix of models helps to build a more robust and generalised forecasting framework. By combining the strengths of different models, the overall prediction performance can be improved.
3. **Temporal analysis:**
   - A comprehensive temporal analysis is conducted to identify trends, seasonal or cyclical patterns in stock prices. This helps develop features that better capture the inherent dynamics of financial markets.
4. **Feature engineering**
   - Invest more effort in feature engineering, focusing on extracting relevant information from raw data. This may involve exploring alternative transformations, creating interaction terms, or incorporating external data sources that enhance the model's understanding of the underlying factors that affect stock prices.

5. **Regularisation techniques:**
    - Regularisation techniques are used to address overfitting or underfitting problems. techniques such as L1 or L2 regularisation help to control the complexity of the model and improve its ability to generalise to unseen data.
6. **Study Domain knowledge**
    - having domain knowledge may help to construct meaningful feature

In conclusion, predicting stock prices is an ongoing process. By refining feature selection, exploring novel feature engineering techniques, and employing diverse training strategies, future models can better navigate the complexity of financial markets and provide more accurate predictions.

3.3 Final Code

[ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from itertools import combinations  # For creating combinations of elements
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit,train_test_split
import xgboost as xgb
from sklearn import svm
import catboost as cbt
import lightgbm as lgb
from numba import njit, prange

# tool functions
def reduce_mem_usage(df, verbose=0):
    """
    Iterate through all numeric columns of a dataframe and modify the data type
    to reduce memory usage.
    """
    # Calculate the initial memory usage of the DataFrame
    start_mem = df.memory_usage().sum() / 1024**2

    # Iterate through each column in the DataFrame
    for col in df.columns:
        col_type = df[col].dtype

        # Check if the column's data type is not 'object' (i.e., numeric)
        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()

            # Check if the column's data type is an integer
            if str(col_type)[:3] == "int":
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).
↪max:
                    df[col] = df[col].astype(np.int8)
```

```
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.
 ↪int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.
 ↪int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.
 ↪int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                # Check if the column's data type is a float
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.
 ↪float16).max:
                    df[col] = df[col].astype(np.float32)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
 ↪float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float32)

    #  Provide memory optimization information if 'verbose' is True
    if verbose:
        print(f"Memory usage of dataframe is {start_mem:.2f} MB")
        end_mem = df.memory_usage().sum() / 1024**2
        print(f"Memory usage after optimization is: {end_mem:.2f} MB")
        decrease = 100 * (start_mem - end_mem) / start_mem
        print(f"Decreased by {decrease:.2f}%")

    # Return the DataFrame with optimized memory usage
    return df

# drop the unneccassary columns
def record_filter(df:pd.DataFrame)->pd.DataFrame:
    if('target'in df):
        df.dropna(subset=['target'],inplace=True) # drop record target =null
        df.reset_index(drop=True, inplace=True)
    return df

def feature_selection(df:pd.DataFrame,NotIncludeAtrs=['row_id',␣
 ↪'time_id','date_id','stock_id'])->pd.DataFrame:
    cols = [c for c in df.columns if c not in NotIncludeAtrs]
    tmp = df[cols]
    return tmp


@njit(parallel=True) #faster
```

```python
def compute_triplet_imbalance(df_values, comb_indices):
    num_rows = df_values.shape[0]
    num_combinations = len(comb_indices)
    imbalance_features = np.empty((num_rows, num_combinations))

    #  Loop through all combinations of triplets
    for i in prange(num_combinations):
        a, b, c = comb_indices[i]

        # Loop through rows of the DataFrame
        for j in range(num_rows):
            max_val = max(df_values[j, a], df_values[j, b], df_values[j, c])
            min_val = min(df_values[j, a], df_values[j, b], df_values[j, c])
            mid_val = df_values[j, a] + df_values[j, b] + df_values[j, c] -␣
 ↪min_val - max_val

            # Prevent division by zero
            if mid_val == min_val:
                imbalance_features[j, i] = np.nan
            else:
                imbalance_features[j, i] = (max_val - mid_val) / (mid_val -␣
 ↪min_val)

    return imbalance_features

def calculate_triplet_imbalance_numba(price:[], df:pd.DataFrame):
    # Convert DataFrame to numpy array for Numba compatibility
    df_values = df[price].values
    comb_indices = [(price.index(a), price.index(b), price.index(c)) for a, b,␣
 ↪c in combinations(price, 3)]

    # Calculate the triplet imbalance using the Numba-optimized function
    features_array = compute_triplet_imbalance(df_values, comb_indices)

    # Create a DataFrame from the results
    columns = [f"{a}_{b}_{c}_imb2" for a, b, c in combinations(price, 3)]
    features = pd.DataFrame(features_array, columns=columns)

    return features

# feature engineering
def feature_engineering_ver1(df:pd.DataFrame):
    df = df.copy()
    #df_copy['trend']=df['imbalance_size'] * df['imbalance_buy_sell_flag']
    df["volume"] = df["ask_size"] +df["bid_size"]
    df["mid_price"] = (df['ask_price']+df['bid_price'])/2
```

```
    df["liquidity_imbalance"] = (df['bid_size']-df['ask_size'])/
↪(df['bid_size']+df['ask_size'])
    df["matched_imbalance"] = (df['imbalance_size']-df['matched_size'])/
↪(df['matched_size']+df['imbalance_size'])
    df["size_imbalance"] = df['bid_size'] / df['ask_size']


    # create different pariwise combination of imbalance prices
    prices = ["reference_price", "far_price", "near_price", "ask_price",␣
↪"bid_price", "wap"]
    for c in combinations(prices, 2):
        #df_copy[f'{c[0]}_-_{c[1]}'] = (df_copy[f'{c[0]}'] -␣
↪df_copy[f'{c[1]}']).astype(np.float32) ## difference between the different␣
↪prices
        df[f"{c[0]}_{c[1]}_imb"] = df.eval(f"({c[0]} - {c[1]})/({c[0]} +␣
↪{c[1]})")

    #create triplet of imbalance
    sizes = ["matched_size", "bid_size", "ask_size", "imbalance_size"]
    for c in [['ask_price', 'bid_price', 'wap', 'reference_price'], sizes]:
        triplet_feature = calculate_triplet_imbalance_numba(c, df)
        df[triplet_feature.columns] = triplet_feature.values


    df["imbalance_momentum"] = df.groupby(['stock_id'])['imbalance_size'].
↪diff(periods=1) / df['matched_size']
    df["price_spread"] = df["ask_price"] - df["bid_price"]
    df["spread_velocity"] = df.groupby(['stock_id'])['price_spread'].diff()
    df['price_pressure'] = df['imbalance_size'] * (df['ask_price'] -␣
↪df['bid_price'])
    df['market_urgency'] = df['price_spread'] * df['liquidity_imbalance'] #␣
↪showing the how hit of the stock in the market
    df['depth_pressure'] = (df['ask_size'] - df['bid_size']) / (df['far_price']␣
↪- df['near_price'])

    # Calculate various statistical aggregation features
    for func in ["mean", "std", "skew", "kurt"]:
        df[f"all_prices_{func}"] = df[prices].agg(func, axis=1)
        df[f"all_sizes_{func}"] = df[sizes].agg(func, axis=1)

    # time related feature
    for col in ['matched_size', 'imbalance_size', 'reference_price',␣
↪'imbalance_buy_sell_flag']:
        for window in [1, 2, 3,10]:
            df[f"{col}_shift_{window}"] = df.groupby('stock_id')[col].
↪shift(window) # keep track for previous value
```

```
            df[f"{col}_ret_{window}"] = df.groupby('stock_id')[col].
 ↪pct_change(window) # percentage change with previous value

    for col in ['ask_price', 'bid_price', 'ask_size', 'bid_size',␣
 ↪'market_urgency', 'imbalance_momentum', 'size_imbalance']:
        for window in [1, 2, 3,10]:
            df[f"{col}_diff_{window}"] = df.groupby("stock_id")[col].
 ↪diff(window) # difference with previous value


    # adding day for haveing trading
    df["dow"] = df["date_id"] % 5                 # Day of the week
    df["seconds"] = df["seconds_in_bucket"] % 60  # Seconds
    df["minute"] = df["seconds_in_bucket"] // 60  # Minutes
    #global feature
    global_stock_id_feats = {
        "median_size": df.groupby("stock_id")["bid_size"].median() + df.
 ↪groupby("stock_id")["ask_size"].median(),
        "std_size": df.groupby("stock_id")["bid_size"].std() + df.
 ↪groupby("stock_id")["ask_size"].std(),
        "ptp_size": df.groupby("stock_id")["bid_size"].max() - df.
 ↪groupby("stock_id")["bid_size"].min(),
        "median_price": df.groupby("stock_id")["bid_price"].median() + df.
 ↪groupby("stock_id")["ask_price"].median(),
        "std_price": df.groupby("stock_id")["bid_price"].std() + df.
 ↪groupby("stock_id")["ask_price"].std(),
        "ptp_price": df.groupby("stock_id")["bid_price"].max() - df.
 ↪groupby("stock_id")["ask_price"].min(),
    }
    for key, value in global_stock_id_feats.items():
            df[f"global_{key}"] = df["stock_id"].map(value.to_dict())

    df.replace([np.inf, -np.inf], 0,inplace=True)
    return df


def preproccess(df:pd.DataFrame,NotIncludeAtrs=['row_id',␣
 ↪'time_id','date_id','stock_id']):
    df=record_filter(df)
    df=feature_engineering_ver1(df)
    df=feature_selection(df,NotIncludeAtrs)
    df=reduce_mem_usage(df)
    return df

# input data
df_train = pd.read_csv("/kaggle/input/optiver-trading-at-the-close/train.csv")
```

```
print(f"original shape {df_train.shape}")
print(df_train[0:1])
# reduce mem

df_train=preproccess(df_train,NotIncludeAtrs=['row_id', 'time_id','stock_id'])

models=[]
def trainEnsembleModels(folds=5,data:pd.DataFrame=None):
    models=[]
    data=data.copy()
    max_date=data['date_id'].max()
    folds=folds-1
    date_bins = pd.cut(data['date_id'], bins=folds, labels=False)
    for fold in range(folds):

        if fold >= folds-1:
            subData = data.loc[(date_bins == fold)] #block
            date = subData['date_id'].max()-3 # use 3 date for validation
            trainingData=subData[subData['date_id']<=date]
            validData=subData[subData['date_id']>date]
            del subData
        else:
            trainingData = data.loc[(date_bins == fold)] #block
            temp = data.loc[(date_bins >= (fold+1))]
            minDate = temp["date_id"].min()
            validData=temp[temp["date_id"]<(minDate+3)]

        trainingData=trainingData.drop(columns='date_id')# give date_id
        validData=validData.drop(columns='date_id')

        x_train=trainingData.drop(columns='target')
        y_train=trainingData['target'].values

        x_valid=validData.drop(columns='target')
        y_valid=validData['target'].values

        print(f"train.shape {x_train.shape} valid.shape {x_valid.shape}")
        #train model
        model = lgb.LGBMRegressor(n_estimators=␣
 ↪5500,objective='mae',learning_rate=0.00871,num_leaves=256,subsample=0.
 ↪6,colsample_bytree=0.8,max_depth=11,importance_type= "gain",verbosity=␣
 ↪-1,device="gpu",seed= 42)
        model.fit(x_train,y_train,
                  eval_set=[(x_valid,  y_valid)],
                  callbacks=[
                        lgb.callback.early_stopping(stopping_rounds=100),
                        lgb.callback.log_evaluation(period=100),
```

```
                    ],
                )
            models.append(model)
            print(f"append model of fold {fold}")
            del x_train,y_train,x_valid,y_valid,trainingData,validData

        #train all data again
        average_best_iteration = int(np.mean([model.best_iteration_ for model in␣
 ↪models]))
        print(f"average_best_iteration {average_best_iteration}")
        all_y_train = data['target'].values
        all_x_train = data.drop(columns=['target','date_id'])
        print(f"all train.shape {all_x_train.shape}")
        model = lgb.LGBMRegressor(n_estimators=␣
 ↪average_best_iteration,objective='mae',learning_rate=0.
 ↪00871,num_leaves=256,subsample=0.6,colsample_bytree=0.
 ↪8,max_depth=11,importance_type= "gain",verbosity= -1,device="gpu",seed= 42)
        model.fit(all_x_train,all_y_train,
                    callbacks=[
                        lgb.callback.log_evaluation(period=100),
                    ],
                )
        models.append(model) # total is k models + 1 final model
        return models

y_min = df_train['target'].min()
y_max = df_train['target'].min()
models=trainEnsembleModels(folds=5,data=df_train)

def zero_sum(prices, volumes):
    std_error = np.sqrt(volumes)  # Calculate standard error based on volumes
    step = np.sum(prices) / np.sum(std_error)  # Calculate the step size based␣
 ↪on prices and standard error
    out = prices - std_error * step  # Adjust prices by subtracting the␣
 ↪standardized step size
    return out

def zero_sum_prediction(predY,data):
    predY = zero_sum(predY, data['bid_size'] + data['ask_size'])
    clipped_predictions = np.clip(predY, -64, 64)
    return clipped_predictions

import optiver2023
env = optiver2023.make_env()
iter_test = env.iter_test()

cache = pd.DataFrame()
```

```
counter = 0
for (test, revealed_targets, sample_prediction) in iter_test:
    if("currently_scored" in test):
        test=test.drop("currently_scored",axis=1)

    cache = pd.concat([cache, test], ignore_index=True, axis=0)
    if counter > 0: # take 4 week as windows
        cache = cache.groupby(['stock_id']).tail(28).sort_values(by=['date_id',␣
 ↪'seconds_in_bucket', 'stock_id']).reset_index(drop=True)

    print(f"------counter :{counter}-----------")
    print(cache.shape)
    print(test.shape)
    test_df=preproccess(cache)# pick the last 200 record need to be predict
    test_df=test_df[-test.shape[0]:]
    pred = np.mean([model.predict(test_df) for model in models], axis=0)
    pred= zero_sum_prediction(pred,test)
    print(pred.values)
    sample_prediction['target'] = pred

    env.predict(sample_prediction)

    counter += 1
```