Assignment 5: Predicting Stock Price Jumps Using ML

Overview (Weight: 10%)

Objective: Learn and understand how various ML models can be used for stock price jump (absolute return greater than 10%) prediction

The goals of this assignment are for the students to

- Develop practical skills with scikit-learn and PyTorch for constructing various machine learning models, including logistic regression, LASSO, Ridge, K-Nearest Neighbors, and Gradient Boosted Trees.
- Gain experience with hyper-parameter tuning and model selection techniques to optimize prediction performance.
- Learn to identify patterns and trends in stock price movements and analyze how different economic indicators can serve as predictive covariates.
- Critically evaluate model outputs, focusing on drawing meaningful conclusions and insights from the predictive analysis, rather than just implementing the programming.

Required Files for Submission

- One Jupyter Notebook that includes all the data wrangling and printed computations. Unless otherwise specified, failure to have computation output will result in a loss of points
- One output in PDF/HTML format
- A well formatted report (5 pages maximum) with plots and analyses. Code is not acceptable in the report.
- Note: Please do **NOT** submit data

Data

The required dataset for the assignment is available on the OneDrive: Link

- The name of data file is *MSF_1996_2023.csv*
- Use monthly CRSP stock data for this assignment
- See the definitions of industry code provided at the end of the assignment
- It is important that you use the variables you computed from previous assignments. For example, systematic volatility, idiosyncratic volatility, beta, etc.
- Get NBER recession data. It will be useful as a crisis vs non-crisis period indicator.

Assignment

Data Construction and Analysis

Start with the full sample for initial analysis and later in **Step 3**, you can sample data to reduce the computational work.

Step 1: Construct a categorical outcome variable based on the following definition:

$$y_{i,t+1} = \begin{cases} 1 & \text{if } |Ret_{i,t+1}| > 0.1 \\ 0 & \text{otherwise} \end{cases}$$

This variable indicates whether the **absolute** return of stock i at time $t+1$ exceeds 10%, with 1 representing a stock price jump greater than 10%, and 0 otherwise.

Step 2: Plot the percentage of jumps over time along with the three macro-economic indicators from FRED. You already collected them for the SIFMA assignment. Consider using VIX as well. - What do you observe? - Can any of these indicators serve as a precursor for upcoming higher likelihood of jumps?

Step 3: Come up with a list of possible covariates that should matter for jump prediction.

- Using data from $t-w$ to t in MSF data regarding price, returns and industry. where w is look-back window.
- For different features you can use different w.

- From previous assignments (Beta from CAPM and Volatility, Estimating Neural Beta), you can use beta, volatility, etc.
- Use USREC and other macro-economic indicators from FRED. Collect variables beyond what you collected for the SIFMA assignment.

Step 4: To reduce work (computational effort), for each year, instead of analyzing the entire set of firms, select a sample of 100 firms randomly (1200 observations per year). Use industry categories as dummy feature variable.

Stock Jump Prediction

Model - 1: Logistic Regression

Do an **out-of-sample** prediction. Follow these steps

- Divide the sample into in-sample estimation period (1996-01 to 2017-12) and out of sample forecasting period (2018-01 to 2023-12)
- Estimate the model with 1996-2017 data
- Forecast jump for the 2018-2023 time period using the estimates from the 1996-2017 time period and explanatory variable data from 2018-2023 (using the predict() function)
- Note you can do forecasting three ways and try all of them and contrast the results
  - Using 2018–2023 as the out of sample period or
  - doing a rolling out of sample estimation (say, using 1996-2017 data to forecast for 2018, using 1996-2018 data for 2019 etc.)
  - using a fixed window (say, using 1996-2017 data to forecast for 2018, using 1997-2018 data for 2019, 1998–2019 for forecasting 2020 etc.)
- Plot the ROC curve and calculate AUC and KS statistics (using functions from sklearn and/or scipy etc.)
- Change feature set till you get a model with good out of sample performance

Model 2 and 3: LASSO Logistic regression and Ridge Logistic regression

Based on the practice from previous steps, do an **out-of-sample** prediction using LASSO Logistic regression and Ridge Logistic regression.

The least absolute shrinkage and selection operator (**LASSO**) is a regularization tool that penalizes model complexity in an effort to avoid overfitting the sample data. We obtain the LASSO estimates for the hazard model by minimizing the negative log likelihood function [eq:lasso]: with a penalty weight λ placed on the sum of the absolute value of covariate coefficient estimates (also called the l_1 penalty),

$$\sum_i \left(-Y_{i,t+1}\left(\beta_0 + \beta' X_{i,t}\right) + log(1 + \exp\left(\beta_0 + \beta' X_{i,t}\right)\right) + \lambda \sum_{k=1}^{p} |\beta_k|$$

where p is the number of covariates. By adding the penalty term to the likelihood estimation the LASSO shrinks the coefficient estimates toward zero, while this worsens the in-sample fit of the model it can help improve the out-of-sample performance by avoiding overfitting.

The ridge regression is very similar to the LASSO except that it replaces the l_1 penalty with a l_2 cost. The penalty function for the ridge regression is $\lambda \sum_{k=1}^{p}(\beta_k^2)$ and also serves as a regularization tool. The main difference between the two estimates is that the ridge places little penalty on small values of β but a rapidly increasing penalty on larger values, while the LASSO places a constant penalty on deviations from zero.

Steps to follow for LASSO and Ridge Regression

1. Observe that same regularization strength lambda is applied to all features, thus, scale of the features becomes extremely important for the LASSO and Ridge type of regression models. So, make sure to standardize your features before fitting LASSO and Ridge models.
2. Run a Logistic LASSO regression on the training data (1996-2017) to automatically select a good combination of covariates using **sklearn.linear_model.Lasso()** function (You may throw all relevant variables into the model and let it pick the good covariates for you)
3. Calibrate the hyperparameter λ in Logistic LASSO regression using training data (1996-2017) to find the optimal value which gives you the best model performance. Use **sklearn.linear_model.LassoCV**
4. After building the best Logistic LASSO regression (with the optimal λ), use the good covariates selected to fit a standard Logistic regression on the training data (this is so called **Post LASSO** Logistic regression)
5. Forecast jumps for testing data (2018-2023) using the Post LASSO Logistic regression model estimates from in-sample training data
6. Perform analysis similar to Model-1. You need to only use **one** of the forecasting ways from simple, rolling and fixed.
7. Compare model performance of the Post LASSO Logistic regression model with the Logistic regression model built in Model-1. Does it perform better?
8. Repeat the steps above for Ridge Regression

Model - 4: K-Nearest Neighbor (KNN)

Based on the practice from previous steps, do an **out-of-sample** prediction using K-Nearest Neighbor algorithm. Follow these steps below

1. For each observation in the test data, identify the **top K** closest observations from the training data by calculating the Euclidean distance based on the good covariates from Model-1 and classify each observation in the test data by taking a majority vote (use the **KNeighborsClassifier()** function)
2. Calibrate the **hyperparameter K** in K-Nearest Neighbor algorithm to find the optimal value which minimizes the misclassification rate on the validation data (2013-2017)
3. Compare misclassification rate of K-Nearest Neighbor algorithm with the Logistic regression, LASSO Logistic regression and Ridge Logistic regression models built in earlier steps (use the cutoff that minimizes the misclassification rate for logistic regression models). Which one has the lowest misclassification rate?

Models - 5: Pick ONE of the Gradient Boosted Tree Frameworks (XGBOOST or LIGHTGBM)

Gradient boosting trees are the most important alternative class of tree-based estimators to random forests. Boosting is a strategy that sequentially fits an estimator to a training dataset and gives more weight to misclassified observations (or errors) in successive boosting rounds. Boosted trees iteratively estimate a sequence of shallow trees, each trained to predict the residuals of the previous tree. The final prediction is an accuracy-weighted average of the estimators. Based on the practice from previous steps, do a out-of-sample prediction using **XGBoost** algorithm. Follow these steps

1. Install package **xgboost**
2. Import **XGBClassifier()** function from **xgboost**
3. Calibrate the hyperparameter **number of boosting** rounds in XGBoost algorithm to find the optimal value which minimizes the misclassification rate on the validation data (2013-2017)
4. Run the XGBoost classification model on the test data
5. Compare **misclassification rate** of XGBoost algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?

LightGBM another gradient boosted trees model. LightGBM uses a novel technique of Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value while XGBoost uses pre-sorted algorithm & Histogram-based algorithm for computing the best split. Follow these steps

1. Install package **lightgbm**
2. Import LGBMClassifier() function from **lightgbm**
3. Calibrate the hyperparameter **max depth** in LightGBM algorithm to find the optimal value which minimizes the misclassification rate on the validation data (2013-2017)
4. Run the LightGBM classification model on the test data

5. Compare **misclassification rate** of LightGBM algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?

Supplementary Details

Readings

- Empirical Asset Pricing via Machine Learning from AQR (Link)
- Machine learning in hedge fund investing from J. P. Morgan (Link)
- Benchmarking Machine Learning Models to Predict Corporate Bankruptcy (Link)

Industry code:

SIC Code	Industries
1 – 999	Agriculture, Forestry and Fishing
1000 – 1499	Mining
1500 – 1799	Construction
2000 – 3999	Manufacturing
4000 – 4999	Transportation and other Utilities
5000 – 5199	Wholesale Trade
5200 – 5999	Retail Trade
6000 – 6799	Finance, Insurance and Real Estate
7000 – 8999	Services
9000 – 9999	Public Administration