Assignment 6

Assignment 6: Predicting Corporate Defaults Using Hazard and ML Models (Python)

Submission Details

1. This assignment is **individual** assignment. Assignment weight is **10%**
2. Submit through Canvas; Deadline is 11:59pm on **Oct 20, 2025**
3. Use Python Jupyter Notebook
4. All the necessary data is available on Canvas
5. Please get started early as there are many models to estimate
6. You have to submit ONLY:
    - Jupyter Notebook file (source codes)
    - Output in HTML/PDF format (source code in readable format)
    - 5-page PDF with your analyses (your findings)
    - Short and concise explanation (1-3 lines) for the specification used in the regressions (economic rationale for each variable and expected sign). For example, I chose **leverage** as one of the explanatory variables because it determines the default boundary and higher leverage is expected to result in a higher (+) likelihood of default
    - Do not run the regressions or ML models first, see the sign and then input this sign in the writeup. Provide an economic explanation for why the variable should matter for default prediction and how (positive or negative) it would effect the default likelihood

6.0. Motivation

Recent corporate failures, like the bankruptcy of auto-parts supplier First Brands and the liquidation of subprime lender Tricolor, show how hidden leverage, tricky receivables financing, and unstable funding can quickly destabilize firms and spill over to lenders, suppliers, and borrowers. These episodes highlight the need for timely early-warning systems that flag rising default risk before problems become public, guiding capital allocation and counterparty risk management.

Evidence from recent research (Alanis, Chava and Shah, 2022) benchmarks hazard and modern ML models and finds that tree-ensemble methods, especially gradient-boosted trees, deliver the strongest one-year-ahead bankruptcy forecasts; market-based firm signals (excess returns, idiosyncratic risk, relative size) are top predictors, while broad macro/industry variables and filing-text features add little out-of-sample lift. This motivates our side-by-side estimation of hazard and ML approaches (LASSO/Ridge, RF/Survival RF, XGBoost/LightGBM) on lagged CRSP-COMPUSTAT data, evaluating both statistical accuracy (ROC/AUC/KS) and practical credit usefulness. We also recommend the article as background reading for model choice, variable importance, and out-of-sample evaluation details.

6.1. Data Extraction and Analysis

1. You should use the provided bankruptcy data between 1964 and 2020, and the same datasets you have used in previous assignments – CRSP and COMPUSTAT.
2. Merge CRSP-COMPUSTAT with the bankruptcy data to create a dataset with features.
3. There would be multiple observations per firm, with an indicator variable that takes one in the year of bankruptcy and zero otherwise.
4. Make sure that the accounting and market data is lagged (so that there is no **look ahead bias**).
5. Run the classification models as detailed in the following section. Compute the required statistics.
6. Save the results in HTML/PDF format. We don't want to see hundreds of pages of output or datasets.
7. In your report, provide the analysis and describe the insights you got from the assignment.
8. For more detailed information, you can browse through **An Introduction to Statistical Learning with Applications in Python**.

6.2. Model Training

Model - 1: Logistic Regression

Please follow the steps below.

1. Come up with a list of possible covariates that should matter for default prediction.
2. Compute these explanatory variables.
3. Do an **in-sample** estimation and prediction. Follow these steps
    - Use the entire time period 1964-2020 for estimation
    - Run a Logistic regression model with bankruptcy as the LHS variable and the variables in step 1 as explanatory variables
    - Present the output and fit statistics for the model (output of summary() function)
4. Do an **out-of-sample** prediction. Follow these steps
    - Divide the sample into in-sample estimation period (1964-1990) and out of sample forecasting period (1991-2020)
    - Estimate the model with 1964-1990 data
    - Forecast default for 1991-2020 time period using the estimates from 1964-1990 time period and explanatory variable data from 1991-2020 (using predict() function)
    - Note you can do forecasting three ways and try all of them and contrast the results
        - Using 1991–2020 as the out of sample period or
        - doing a rolling out of sample estimation (say, using 1964-1990 data to forecast for 1991, using 1964-1991 data for 1992 etc.)
        - using a fixed window (say, using 1964-1990 data to forecast for 1991, using 1965-1991 data for 1992, 1966–1992 for forecasting 1993 etc.)
    - Rank the default probabilities into deciles (10 groups).
    - Compute the number (and percentage of defaults) in each of the 10 groups during 1991-2020 time period.
    - A good model is one that has the majority of defaults in decile 1 or 2 and very few in other deciles
    - Plot the ROC curve and calculate AUC and KS statistics (using functions from package sklearn and/or scipy etc.)
5. Iterate steps 1-4 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 in-sample training data 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 in-sample data 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 in-sample training data (this is so called **Post LASSO** Logistic regression)
5. Forecast default for out-of-sample testing data using the Post LASSO Logistic regression model estimates from in-sample training data
6. For out-of-sample predictions, do analysis similar to step 4 of Model-1
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 above steps 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 out-of-sample data, identify the **top K** closest observations from the in-sample data by calculating the Euclidean distance based on the good covariates from Model-1 and classify each observation in the out-of-sample data by taking a majority vote (use **KNeighborsClassifier()** function
2. Calibrate the **hyperparameter K** in K-Nearest Neighbor algorithm to find the optimal value which minimizes the misclassification rate on the out-of-sample data
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?

Model 5 and 6: Random Forest and Survival Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Based on the practice from previous steps, do a **out-of-sample** prediction using Random Forest. Follow these steps

1. Run the Random Forest classification model for **out-of-sample** using the **sklearn** package in Python. You can load the model by calling the function **RandomForestClassifier()** from **sklearn.ensemble** module. You can learn more about Random Forests from research article of Leo Breiman).
2. Calibrate the hyperparameter max depth and number of trees in Random Forest algorithm to find the optimal value which minimizes the misclassification rate on the out-of-sample data
3. Compare misclassification rate of Random Forest algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?

A Survival Random Forest ensures that individual trees are not correlated. To achieve that it builds each tree on a different bootstrap sample of training data, and at each node, it only evaluates the split criterion for a randomly selected subset of features and thresholds. Follow these steps

1. Install package **scikit-survival**
2. Import **RandomSurvivalForest()** function from sksurv.ensemble module of **scikit-survival** package

3. Calibrate the hyperparameter **number of trees** in Survival Random Forest algorithm to find the optimal value which minimizes the misclassification rate on the out-of-sample data
4. Run the Survival Random Forest classification model for **out-of-sample** prediction
5. Compare misclassification rate of Survival Random Forest algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?

Models 7 and 8: Gradient Boosted Trees (XGBOOST and 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 out-of-sample data
4. Run the XGBoost classification model for **out-of-sample** prediction
5. Compare **misclassification rate** of XGBoost algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?

Now, build **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 out-of-sample data
4. Run the LightGBM classification model for **out-of-sample** prediction
5. Compare **misclassification rate** of LightGBM algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?