Assignment 10

Assignment 10: Predicting Corporate Defaults Using Hazard and ML Models (Python)

Submission Details

1. This assignment is **individual** assignment. Assignment weight is **30%**
2. Submit through Canvas https://canvas.gatech.edu; Deadline is 6pm on **Dec 03, 2023**; Late submission without penalty is allowed till 6pm on **Dec 10, 2023**. As we need to grade and submit final grades by Dec 14th, no late submissions beyond 6pm on **Dec 12, 2023** would be entertained.
3. Use Python Jupyter Notebook
4. Please get started early as there are many models to estimate
5. You have to submit ONLY
    - Jupyter / Qaurto Notebook file
    - Output in PDF format (Hide Code)
    - You don't need to submit any datasets
    - 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.

Assignment 10.1

Model - 1: Logistic Regression

Please follow the steps below.

1. Come up with a list of possible covariates that should matter for default prediction. Some possible sources are

    - Lecture by Dr. Reguly would go through the models in more detail
    - Alanis, Chava and Shah (2023)
    - Lecture on Merton Model
    - Lecture on credit scoring in optional readings
    - Lecture on credit rating agencies (see Fitch rating guidelines in optional readings)
    - Chava and Jarrow (2004), Chava, Stefanescu and Turnbull (2012), Alanis and Chava (2021) - will be put in the optional resources folder
    - Variables computed in assignments 1, 4, 6, 7 and assignment 9

2. Compute these explanatory variables

3. Do an **in-sample** estimation and prediction. Follow these steps

    - Use the entire time period 1964-2019 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 (1991-2019)
    - Estimate the model with 1964-1990 data
    - Forecast default for 1991-2019 time period using the estimates from 1964-1990 time period and explanatory variable data from 1991-2019 (using predict() function)
    - Note you can do forecasting three ways and try all of them and contrast the results
        - Using 1991–2019 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 1991-2019 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 (https://link.springer.com/article/10.1023/a:1010933404324){Random Forests}).
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?

Extra Credit: Model 9: Artificial Neural Network (ANN)

Neural networks (NNs) represent a wide class of learning methods. A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). An MLP has three types of layers of nodes, an input layer, hidden layers (there can be multiple hidden layers) and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP model learns using a supervised learning technique called backpropagation. Follow these steps to build a ANN model

1. Install **tensorflow**
2. Import **keras** module from **tensorflow**
3. Build neural network model using **tensorflow.keras.Sequential()**
4. Try different number of hidden layers with different number of neurons.
5. Also try different **activation functions** (Sigmoid, ReLU, Tanh etc.).
6. Find the optimal configuration of neural network which minimizes the misclassification rate on the out-of-sample data
7. Run the MLP classification model for **out-of-sample** prediction
8. Compare **misclassification rate** of MLP algorithm with models built in earlier tasks. Which one has the lowest misclassification rate?

Assignment 10.2: Sentiment Analysis

In this assignment, we will extend our work from Assignment 10.1 to perform sentiment analysis on financial texts. You will use the best performing model from 10.1 and apply it to a specific set of data.

1. **Model Selection**: Based on your work in 10.1, select the best performing model for sentiment analysis.
2. **Analyzing False Negatives**:

- Identify the sample of false negatives from the selected model. False negatives refer to the firm-year observations where firms declared bankruptcy, but the model predicted no bankruptcy.

3. **Sentiment Analysis of SEC 10-K Filings**:

- **Data Location**: Retrieve SEC 10-K filings for all false negatives firms starting year 2000, focusing on the 5 years prior to bankruptcy. These are located in the `Q:/Data-ReadOnly/SurvivalAnalysis/10K` folder.
- **Relevant Text Selection**:
  - Not all sections in the SEC 10-K filings are relevant for bankruptcy prediction. Choose the relevant items for analysis and justify your selection.
  - The files are in JSON format, facilitating the extraction of specific sections.
  - **Notes on reading data from JSON files:**
  - If a firm bankrupts in year t, then the filings data for year t may or may not be present in the folder
  - Not all firms will have full 5 years of data before bankruptcy. Only use the data that is available to compute distributions.
  - Some of the JSON files in the data are corrupt (contains " " (empty string) in its contents), you can skip those files and use 0.0 as a sentiment score for those files.
- **Sentence Tokenization**:
  - Break down the text into individual sentences.
  - Use the NLTK package for tokenization.
- **Sentiment Analysis Using FinBERT**:
  - Calculate the sentiment for each tokenized sentence using the FinBERT model.
  - Utilize the `transformers` library as demonstrated in the provided Python script.
  - Classify sentiments into positive, neutral, or negative categories.

```
# LABEL_2 is positive, LABEL_1 is neutral, LABEL_0 is negative

from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("ipuneetrathore/bert-base-cased-finetuned-finBERT")
model = AutoModelForSequenceClassification.from_pretrained("ipuneetrathore/bert-base-cased-finetuned-finBERT")

# Initialize sentiment classifier
classifier = pipeline('sentiment-analysis', model=model, tokenizer=tokenizer, framework="pt")

# Analyze sentiments of example sentences
results = classifier([
    'The revenue increased above and beyond over last quarter.',
    'We did poorly on sales',
    'The inflation increased over last year.',
```

```
    'It is a normal trading day'
], batch_size=2, truncation="only_first")

# Print results
print(results)
```

- **Document-Level Sentiment Measure**: Use labeled sentences in each document to generate a document-level measure of sentiment for document i using the following formula:

$$Measure_i = \frac{\#Positive_i - \#Negative_i}{\#Total_i}$$

4. **Time Series Analysis**:

- Compute the descriptive stats - N, mean, standard deviation, skewness, kurtosis along with the minimum value, maximum value, 1%, 5%, 25%, 50%, 75%, 95%, 99% percentiles.
- Plot them for the 5-year period leading up to bankruptcy (t-5, t-4,..., t-1). Where t is the bankruptcy year.
- Analyze and interpret the trends observed in these plots.

5. **Evaluating Text-Based Features**:

- Consider whether adding text-based features would enhance the model's performance.
- Discuss under what conditions these additional features might be beneficial.

Additional information

BERT (Bidirectional Encoder Representations from Transformers) is a model proposed by the researchers at Google AI Language in their paper titled BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Check their GitHub repository BERT GitHub for detailed information on BERT.

FinBERT is a pre-trained NLP model to analyze sentiment of financial text based on BERT model. You can find information on how to use pre-trained FinBERT model on HuggingFace. HuggingFace also has a pipeline for sentiment analysis, making it easier to implement various models in the library. Please check pipeline interface details.

Important Resources and References

- Kaggle: Python NLTK sentiment analysis
- NLTK official documentation, with examples (most relevant):
  - http://www.nltk.org/api/nltk.sentiment.html
  - http://www.nltk.org/howto/sentiment.html
- Foundational Papers (for deeper understanding): Encourage to at least watch YouTube videos.

- Word2Vec (dated, but useful to understand the concept of embeddings, which is very important and for historical reasons)
  - https://arxiv.org/pdf/1301.3781.pdf
  - https://jalammar.github.io/illustrated-word2vec/
- Attention is All You Need
  - https://arxiv.org/abs/1706.03762
  - https://jalammar.github.io/illustrated-transformer/
  - https://www.youtube.com/watch?v=OxCpWwDCDFQ
  - https://www.youtube.com/watch?v=UPtG_38Oq8o
- BERT
  - https://arxiv.org/abs/1810.04805
  - https://jalammar.github.io/illustrated-bert/
  - https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Data Extraction and Analysis

1. You can use the same datasets that you have used in previous assignment - CRSP and COMPUSTAT data
2. You may be able to use many of the features that you have computed in Assignments 1, 4, 6, 7, and 9.
3. In particular, you can **adapt** the data extraction steps in **Assignment 9**
4. Bankruptcy data starting from 1964 in **Q:/Data-ReadOnly/ SurvivalAnalysis/BR1964_2019.csv**
5. Merge CRSP-COMPUSTAT with the bankruptcy data to create a dataset that I outlined in the class.
6. There would be multiple observations per firm, with an indicator variable that takes one in the year of bankruptcy and zero other wise
7. Make sure that the accounting and market data is lagged (so that there is no **look ahead bias**)
8. Run the classification models discussed earlier
9. Compute the required statistics
10. Save the results in PDF format. I don't want to see hundreds of pages of output or datasets
11. Upload the Jupyter notebook file and results (in PDF form) to Canvas
12. For more detailed information, you can browse through http://faculty.marshall.usc.edu/gareth-james/ISL/, **An Introduction to Statistical Learning with Applications in R/Python**