Parallelism in Company Bankruptcy prediction

Aditya Mukundan
Dept of Electrical and Computer Engineering
Northeastern University
mukundan.a@northeastern.edu

Akanksha Agnihotri
Dept of Electrical and Computer Engineering
Northeastern University
agnihotri.ak@northeastern.edu

Nihal Mathew Sashikumar
Dept of Electrical and Computer Engineering
Northeastern University
sashikumar.n@northeastern.edu

Abstract:

Bankruptcy prediction is a crucial problem statement prevalent in the finance sector today. With the looming recession, feds hiking up interest rates with inflation; there exists a need to assess the financial as well as liquidity health of companies with time as a crucial constraint. In recent years, a plethora of machine learning techniques used for classification have been identified in solving and predicting whether a certain company is bankrupt or not. In this project, we aim to investigate and exploit parallelism in evaluating various classification algorithms used to predict bankruptcy given a large dataset. With the size of data ever increasing, we aim to achieve speedup, high accuracy as well as efficiency by leveraging the concepts that we have learnt throughout the course. With focus on scalability, optimization and feature selection, this project explores the techniques through which parallel processing can be utilized to solve large and complex problems.

I. Problem Statement

With the advent of parallel processing and distributed computing, we aim to perform Bankruptcy prediction on the large Polish Bankruptcy dataset [1] available on the UCI Machine Learning repository. We have utilized concepts learnt from the course such as feature scaling and model selection to first understand the relevant features that contribute towards Model fitting. It is also important to ensure that while training, we do not overtrain the sample which will incorporate undue bias into the model which can hinder the accuracy. By selective classification and scaling, we have decided to go ahead with five algorithms that are commonly used in classification and prediction. The crux of the problem statement lies with parallelizing the data; for this we have used Spark Data Frames to assist in scalability. With available methods such as *partitionBy()*, *repartitionBy()*, *cache()*, *persist()*, *unpersist()* etc. we aim to explore and find an optimal model that fully parallelizes the dataset that we have chosen to work on. Finally, we evaluate the models on *Northeastern's Discovery Cluster*, to study the behavior of our algorithms on multiple cores by observations on speedup and overhead.

II. Related Work

The paper [1] uses machine learning techniques for predicting the bankruptcy of a company. The research findings have implications for investors, creditors, and other stakeholders, and the authors suggest potential areas of improvement for future research in this field. This was implemented over

the Taiwan Economic Journal dataset. The authors conducted the experiment using three machine learning models such as Random Forest, Decision Tree, Logistic Regression, Support Vector Machine. However, drawback of the paper was that the dataset was a balances data. It had no missing values in the column. Hence none of the columns were dropped and data preprocessing was not so complex. Moreover, the execution time was quite large for this paper and no Spark or Parallelism was used.

Another paper [2] that we referred to helped in predicting bankruptcy in banks using artificial intelligence techniques with a focus on improving understandability. It uses artificial intelligence-based Graph AI and neural networks to perform bankruptcy predictions and estimated future improvements too. The dataset used in this paper was Federal Deposit Insurance Corporation (FDIC) data, Essex. The author has used the traditional approach of using mathematics model and fitted it within an artificial neural network. However, the drawback of this paper was that it did not use any machine learning model and was based over neural network only, hence determining that only neural network is good to use for bankruptcy prediction is not the appropriate solution to the problem. Moreover, parallelism related to graph parallelism was also not mentioned, hence spark was not considered.

III. Polish Bankruptcy dataset

In this project we analyzed data provided by Emerging Market Information service provided by UCI Machine Learning Repository [3] (Center for Machine Learning and Intelligent Systems). The Data divided for 5 years' worth, each year contains ~10,000 rows and 65 columns. Moreover, out of 65 features we had 64 features with 1 column as class label. The last column of the data defines the class which tells us whether the company is bankrupt or not. The class labels are binary in nature where 0 denotes non bankruptcy and 1 denotes bankruptcy. This is a binary classification objective. Also, the csv files can be divided into 5 years. Here, 1stYear has 271 bankrupted companies and 6756 firms that were not bankrupt in the forecasting period. In the 2nd Year there were 400 bankrupt companies and 9773 firms that were not bankrupt in the forecasting period. In the 3rd year there were 495 companies bankrupted, and 10008 firms were not bankrupt in the forecasting period. The 4th year has 515 bankrupt companies and 9277 firms that were not bankrupt in the forecasting period. 5th year has 410 bankrupt companies and 5500 firms that were not bankrupt in the forecasting period. We took 5 years dataset so that we can learn the pattern for prediction of bankruptcy. Each example represents evaluation metrics such equity, net profit, assets, liability which are important parameters to determine bankruptcy prediction.

Each variable is a float number, larger or smaller than zero. It is worth noting that the data set is multivariate data and is intended for classification as it has some missing values present. The objective of classifying the data set is to determine whether a firm will become bankrupt or not with the help of ~40,000 rows X 65 columns which is roughly 2.6 million data points.

IV. Workflow

As shown in Figure 1. The five years of data is taken in its raw CSV format. An extensive exploratory data analysis was conducted to understand the features and their contribution towards predicting

bankruptcy. Using the help of box plots, bar graphs and scatter plots we were able to narrow down the columns which were empty. Apart from being empty, we found multiple columns of data had mean values completely apart which if not scaled, would negatively affect the accuracy of prediction.

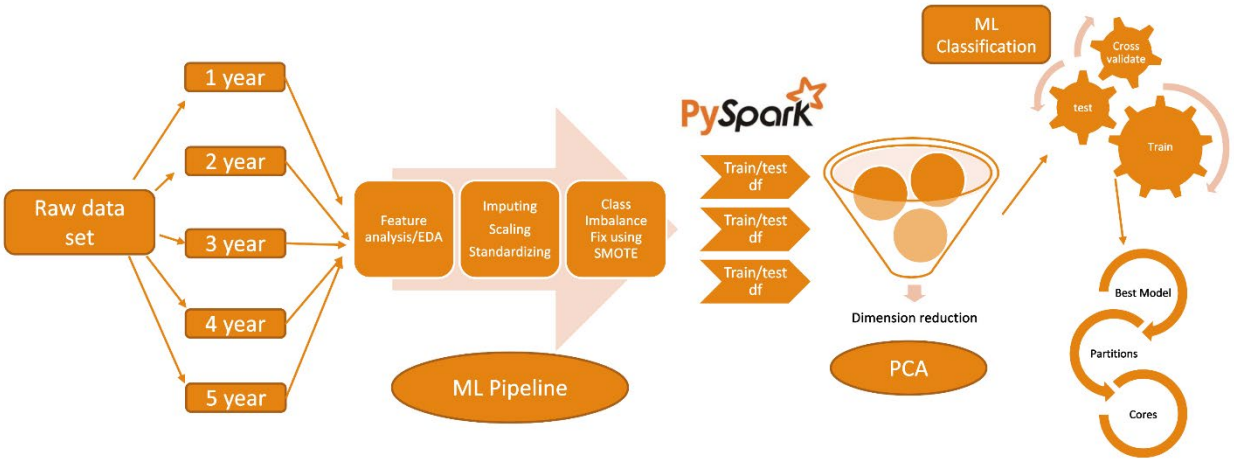

Figure 1: Flow diagram

As with any machine learning pipeline, the dataset stored in Spark DataFrames expedite parallelism; since the dataset was going through a lot of pre-processing, we decided to partition it at this stage into user defined 'N' partitions and cache the DataFrame. We applied imputation and standardization using the *StandardScalar()* library to ensure that all the features have neutral emphasis. Another potential issue was the class imbalance issue. This problem being a Binary Class one which has 2 class labels; contains features that have the majority class outweighing the minority class in terms of the sample numbers. To avoid overfit, we utilized *SMOTE* sampling to balance the class features.

At this stage we repartitioned the data since a lot of pre-processing and sampling would lead to loss of the partition information. For faster data fetch and quicker evaluation time, we `cached()` the data once again before feeding both the test and train data samples before we applied PCA to the training set. During our research, we figured that this approach would yield faster and more efficient results.

Finally, we passed the data to the five algorithms used for our experiment namely *Logistic Regression, Random Forest, Gradient Boosted Tree, Decision Trees, Multi-layer Perceptron*. We also added K-fold cross-validation as we learned in class to further validate and tune the hyperparameters. This aided in boosting the accuracy of classification. Once we evaluated the different algorithms, we decided to test the extent of parallelism in our project by increasing the number of machines (CPU) and by scaling the partitions.

V. Experimental setup - hardware and software specifications

The following table contains details of the different hardware and software that we will be using to compare the execution time, feature selection and Evaluation metrics of the different algorithms:

Software Specification	
Name	**Version**
Python	3.10
Pyspark : Apache Spark with Python	2.4.5
Hardware Specification – Discovery cluster	
CPUs	2.4 GHz Intel E5-2680 v4 CPUs
GPUs	Selection of NVIDIA K80,P100,V100,and T4 GPUs which were available while reserving
Number of CPUs used	2-40

Figure 2: Specifications used in the experiment [4]

VI. Exploratory Data Analysis and Data Preprocessing

Exploratory Data Analysis (EDA) is a critical step in every Machine Learning pipeline that helps you gain insights, clean, and pre-process the data, and make informed decisions about feature engineering, model selection, and evaluation. Skipping EDA can lead to poor model performance, increased complexity, and wasted time and resources.

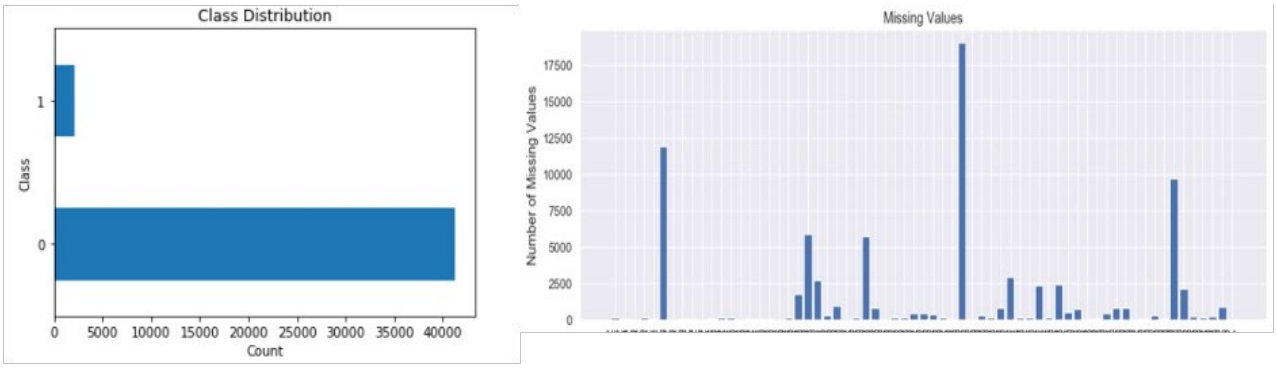

Figure 3: Class distribution and imbalance

In our project, EDA helped us to identify and address vital inconsistencies in our data. Firstly, we observe that the dataset has missing values after transforming the objects to integers. Some features, particularly column 37, are missing over half of the original data. Although we *impute* the missing data in subsequent steps, it would be preferable to just remove this feature. Further by plotting a box plot, we were able to find features with different scales and distributions which led us to scale the data by utilizing a technique called "Standard Scalar". It is a popular pre-processing technique used to standardize or normalize the features in a dataset, particularly when the features have different scales and distributions. It is important to apply scaling to ensure that all features contribute equally to the model's performance and that no feature dominates others due to its larger scale. Standard Scalar works by transforming each feature to have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean (μ) of the feature and then dividing the result by the feature's standard deviation (σ). The formula is given below [13]

$$z_{scaled} = \frac{(x - \mu)}{\sigma}$$

Here, z_{scaled} represents the standardized value of the feature, x is the original value of the feature.

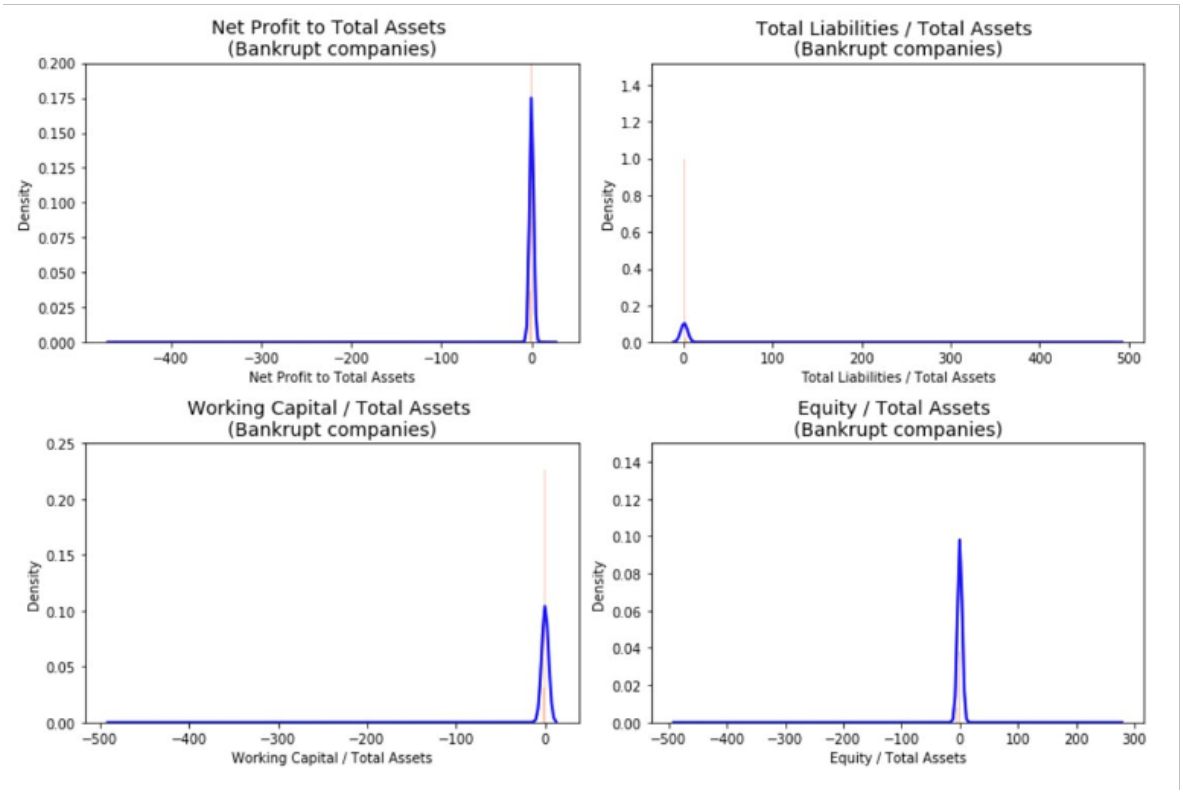

Figure 4: Feature importance based on EDA.

In figure (4), we focused on bankrupt companies to examine their distribution across various features such as Net Profit, Working Capital, Liabilities/Debt, and Equity. As expected, Profit, Working Capital, and Equity exhibit negative skewness, while Debt displays a positive skew. A company with negative earnings, negative working capital, negative revenue, and considerable debt would raise concerns about its financial health. However, it is not implied that all companies meeting these criteria have a higher likelihood of bankruptcy. Many start-ups exhibit these characteristics but eventually evolve into successful businesses with stable revenues. To provide context, the same features were plotted for companies that did not go bankrupt. The results clearly show that the tails for each distribution are much shorter, and the variances are significantly smaller.

Besides, we found that our data had class imbalance, where out of the two class labels (0-Non bankrupt and 1- Bankrupt) only 6.4% of the data indicated insolvency the next year, whereas over 90% indicated no bankruptcy. Thus, to overcome this, we leveraged a sampling technique called "SMOTE" (Synthetic Minority Over-sampling Technique). When the dataset has a significantly higher number of non-bankrupt instances compared to bankrupt instances, the model tends to be biased towards the majority class (non-bankrupt). This can result in poor performance in predicting the minority class (bankrupt), which is often the more critical outcome to predict accurately.

SMOTE generates synthetic samples for the minority class (bankrupt) to balance the class distribution. The technique is based on the K-Nearest Neighbors (KNN) algorithm. For each instance in the minority class (bankrupt), SMOTE selects k-nearest neighbors from the same class. It then generates

synthetic instances by interpolating between the selected instance and its k-nearest neighbors. The formula for generating synthetic instances is given as follows [14]

$$x_{synthetic\_instance} = x_i + \lambda(x_j - x_i)$$

Where $Xsynthetic\_instance$ is the newly generated synthetic instance, x_i is the selected instance from the minority class, x_j is a randomly chosen instance among the K-Nearest Neighbors of x_i, belonging to the same minority class and λ is a random number between 0 and 1, sampled uniformly.

V. Parallelism and Scalability

Leveraging parallelism is the main aspect of this project. By utilization of spark Data Frames to store and process the large dataset we have been able to achieve the expected parallel computing. The data was split into the training data and the testing data post which we partitioned the data to match the number of CPUs that we experimented with to the ratio 1:1 (Partition: CPU).

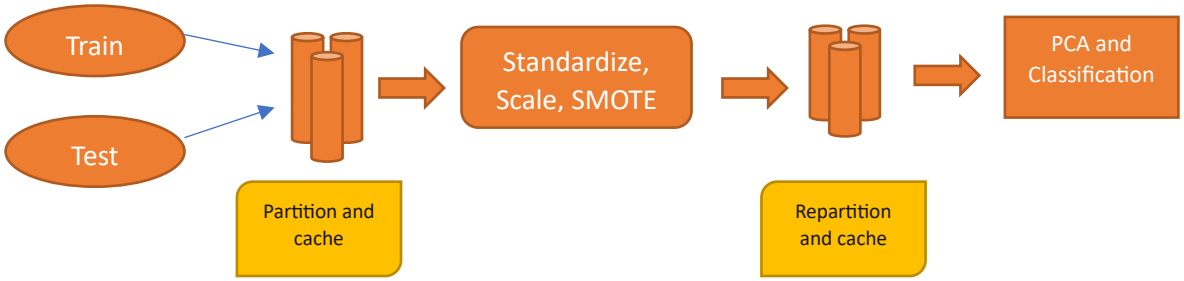

Figure 5: Parallelism workflow

We cached the data at this stage since there is a lot of pre-processing that prevails across the pipeline. After completing SMOTE we again repartitioned the data to reintroduce the partition information as it could have been lost during multiple stages. The data was cached again as each classification algorithm requires the data as input hence it would require multiple calls. Fig 5 shows the flow of parallelism. After each classification algorithm, we used $unpersist()$ to free up memory to obtain time efficient computation.

VI. Model Analysis

As part of the extensive analysis into parallelism, we decided to implement five classification algorithms.

1. Logistic Regression.

This type of statistical model (also known as logit model) is used for classification and predictive analysis. It estimates the probability of an event occurring, such as bankrupt or non-bankrupt, based on a set of independent variables. It is used where the dependent variable (target) is

categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.

Since we have the binary class problem where the class labels are 0 and 1, we have the following loss function as demonstrated in HW3:

$$L(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i) + \lambda \|\beta\|_2^2 \quad [5]$$

Over $\beta \in R^d$

Where the logistic loss is given by

$$\ell(\beta; x, y) = \log\left(1 + e^{-y\beta^\top x}\right)$$

For x, $\beta \in R^d$ *and* $y \in \{0,1\}$[5]

Setting $max\_iterations$ to 100, and setting **λ** across [0.1, 0.01, 0.001] to find the best combination of hyperparameters, we implemented the **K**-fold cross validation in this case. Setting **K** = 4 to perform 4 folds, we achieved an accuracy of **78.9%.** The beta values were considered during the training of model (inside library) hence were not explicitly specified. We have also used **RMSE** (Root Mean Square Error) [10] to evaluate the regression model with the help of regression evaluator and its value was approximately in the rand of **327.56-312.10** when we changed the **lambda** values from **0.0001- 10000** to evaluate the correctness of the model as done in the **HW4.** The RMSE was used to evaluate the performance of models that use continuous numeric values as the target variable since the Logistic Regression model uses categorical data here.

2. Random Forest.

Random Forest is a machine learning algorithm that uses an ensemble of decision trees to make predictions for classification and regression tasks. It selects random features and subsets of training data to construct each tree, which helps to reduce overfitting and increase the model's generalization ability. It is a popular algorithm due to its simplicity, robustness, and ability to handle various types of data and tasks.

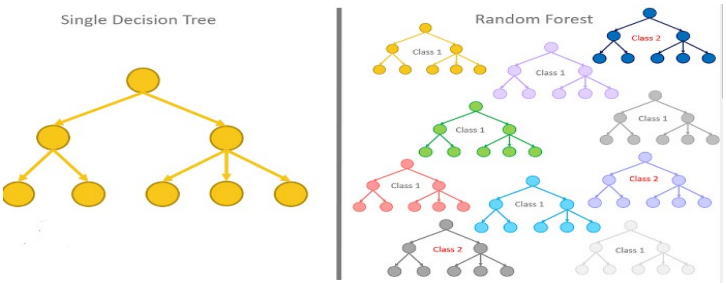

Figure 6: Decision Tree and Random Forest Tree comparison [6]

Since we use Random Forest for classification problems, it makes use of entropy to determine the nodes on a branch tree. Given below is the formula [12]:

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

We have the given parameters.

Setting number of trees to [10, 20 ,20] and max_depth [5, 10, 15] to find the optimal combination. We obtained an accuracy of **85.23%**.

3. *Decision Tree*

Decision Tree is a machine learning algorithm that is used for both classification and regression. It breaks down the data into smaller and smaller subsets, ultimately resulting in a tree-like structure where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a decision or prediction. It recursively splits the data into subsets based on feature values, creating branches and nodes until a stopping criterion is met or a pure class is achieved. For classification problems, Decision Tree uses measures like Gini impurity or information gain to determine the most suitable feature for splitting at each node. In our experiment, we tested multiple combinations of depth layers [5, 10, 15] and trees [16, 32, 64] to find the best model A simple representation of a decision tree is as shown below:

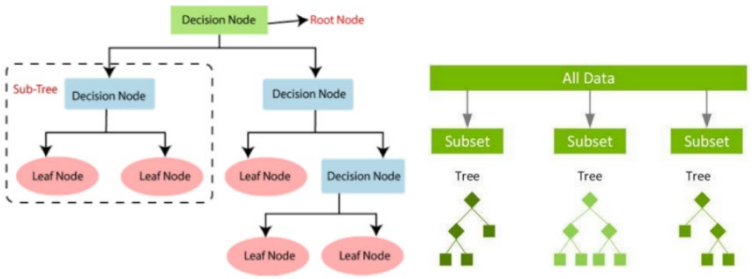

Figure 7: Decision Tree Training structure [7] and Parallelism within Decision Tree [8]

Further we did define cross validator using $MulticlassClassificationEvaluator$.
. We added our decision tree model, hypermeters in the cross validation and executed it against 4 folds to get the best model by fitting over the trained dataset which was obtained after principal component analysis. The best parameters were obtained after hypermeter tuning and 4-fold cross validation. Then the model was evaluated over a test set using cross validator to get the optimal predicted evaluation metric results. We obtained accuracy: 80.2%.

4. *Gradient Boosting Tree*

Gradient Boosting Tree (GBT) is an algorithm that combines multiple decision trees to make predictions, solve classifications and regression problems It uses the gradient descent

optimization technique to minimize the loss function and corrects the mistakes of the previous tree in a sequential manner. GBT can handle both numerical and categorical data, automatically select important features, and is robust to outliers.

Gradient for 'm' trees is shown below.

$$f_m(x) = f_{m-1}(x) + \left(\underset{h_m \epsilon H}{argmin} \left[\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + h_m(x_i)) \right] \right)(x)$$ [9]

Hence trees are given as

$$f_m = f_{m-1} - \rho_m g_m$$ [9]

The hyperparameters were defined by setting the number of trees to [4, 8,10] and *max_depth* to [2,5,10] to find the optimal combination. We obtained an accuracy of 77.2% Moreover, like other machine learning models above we did perform the 4 folds cross validation to get the optimal predicted evaluation metric for gradient boosting tree.

5. Multi-Layer Perceptron

They are very powerful neural networks for supervised learning where regression and classification are done extensively. It has three layers hidden, input and output layer. It is trained using a backpropagation algorithm to minimize the difference between predicted and actual outputs. Input layer of size 10 (features), two hidden layers of size 20 and 10, and an output layer of size 2(classes) was defined. The hyperparameter defined was used to search over the cross validation for MLP. Hyperparameters were defined as a grid with different values of the learning rate (*stepSize*)and optimization algorithm (solver) for the MLP. The values being tested are 0.01, 0.05, and 0.1 for *stepSize* (), and 'l-bfgs' and 'gd' for solver. Just like the above four models we implemented 4-fold cross validation to obtain the optimal solutions for MLP and accuracy obtained was 69.4%.

VII. Validation Analysis

For validation, the performance of all five models - *Random Forest* ***(RF)****, Decision Trees* ***(DT)****, Logistic Regression* ***(LG)****, Gradient Boost Tree* ***(GBT)****, and Multilayer Perceptron* ***(MLP)*** - were evaluated. The table below shows the accuracy, precision, recall, and F1 score for each model evaluated.

Model	accuracy	precision	recall	F1 score
RF	0.853	0.768	0.847	0.793
DT	0.802	0.718	0.798	0.746
LG	0.789	0.698	0.789	0.742
GBT	0.772	0.705	0.723	0.712
MLP	0.694	0.665	0.701	0.682

Figure 8: Models and their metrics

From figure(8), we could see that the model performance for all five models was compared, and Random Forest showed the best results in terms of recall (0.847). **Hyperparameter tuning** was performed using a **4-fold cross-validation** technique which helped **prevent overfitting** and provided more reliable estimates of model performance, as it iteratively trained and validated the model on **different subsets of the data.**

The best parameters found for RF were $\{'max\_depth': 30, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'estimators': 100\}$.

In our scenario, the primary objective is to **identify companies at risk of bankruptcy** ahead of time. Based on our analysis, the Random Forest (RF) classification model showed significant improvements in key metrics after hyperparameter tuning. Although it had some false positives, the focus here should be on minimizing false negatives and ensuring a **high recall** score. Recall across all the models is shown in the below figure (9).

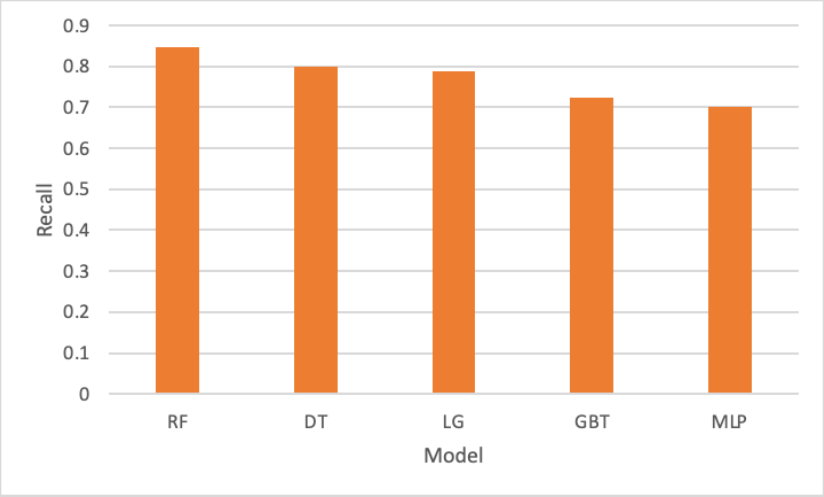

Figure 9 : Recall for All Algorithms

Besides, after hyperparameter tuning, the Random Forest model exhibited improvements in all metrics compared to the baseline, with a notable 2.01% increase in recall. All these results were **validated on the test set**, which ensured that the model was evaluated on **unseen data.**

For the Gradient Boost Tree (GBT) model, it exhibited a good balance between recall and precision. However, it underperformed in comparison to the Random Forest model, particularly in terms of recall. This **trade-off** might cause missed opportunities for investors or banks who use this model, as the GBT is cautious in its predictions.

The Multilayer Perceptron (MLP) model showed an adequate balance between recall and precision, but it was not as robust as the Random Forest model. This makes MLP a safer choice only if there is no clear preference between recall and precision.

Given the **importance of minimizing false negatives** and achieving a high recall score in bankruptcy prediction, the optimized Random Forest model emerges as the best choice. Its improvements in its performance on the test set makes it a reliable and effective model for predicting company bankruptcy.

VIII. Parallel Performance and results

As one of our primary objectives was to work with a large dataset and exploit parallel performance, below are the results of what we were able to observe achieve. Our first model analysis was *Logistic Regression.* Our initial model analysis did not yield optimal results as seen in the below Fig [10].

Although we noticed a decrease in the evaluation time, it was still ~500 seconds. We further wanted to reduce the time by further optimizing parallelism. As previously mentioned in the Parallel workflow; once we repartitioned the data and re - `cached()` the pre-processed data, there was a significant amount of speedup. Note that we have used Spark Data frames as Pandas Data frames do not support parallelism. It is important to note the visible overhead and increase in the computation time from 15 partitions to 40 which is because the model is simple so the communication cost between the partition nodes increases. Overall, however there was a drop-in time taken which shows how parallelism aided in time reduction in the suitable range. Implementing cross validation along parallelism helped us in understanding how model performance improves in terms of speed and quality.

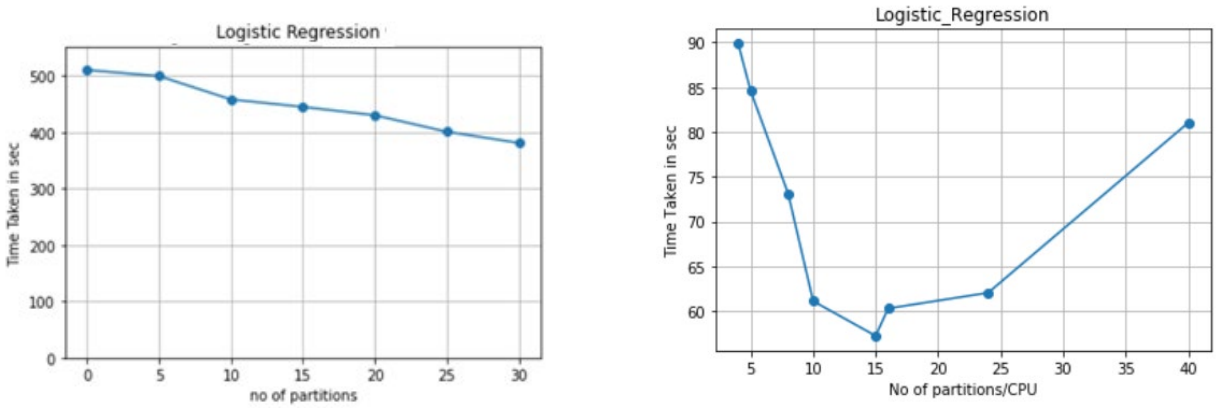

Figure 10: Logistic Regression performance

Apart from logistic regression, we also analyzed the performance of the other 4 algorithms with extra analysis on Random Forest as it was our optimal model.

From Fig [11] we can see the decreasing time in Multi-layer Perceptron; we matched each partition to the CPU to obtain the fastest execution. We did observe a steep decrease up until 15 partitions. We noticed that after 20 partitions, the time taken to train the MLP stabilized around ~20 seconds. To explain this behavior; the neural networks model that we trained was of moderate complexity. So even if the partition size increases, the communication cost does not differ a lot in terms of its increasing trend. From figures [12] we did see decision tree time decreased ~15 partition which show how this model was impacted. After the 15th partition there was an overhead observed. Due to the decision model being simple, the extra communication cost while training a simple model over large partitions was factored in it, resulting in a large overhead.

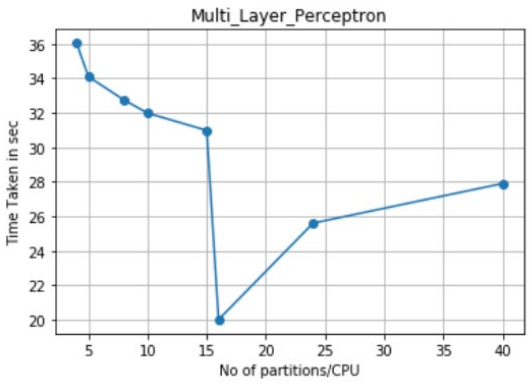

Figure 11: Multi-layer Perceptron Performance

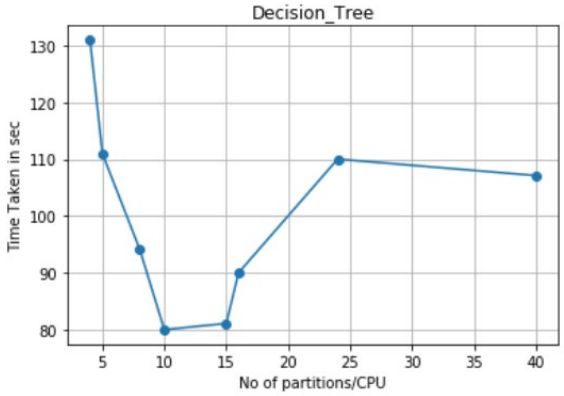

Figure 12: Decision Tree Performance

In Fig [13] we observed the performance of the branching-based algorithm GBT performance with respect to parallelism. Parallelism did benefit GBT in terms of time reduction as no overhead was observed, but due to the large tree size in GBT, we did not find a significant speedup while increasing the number of partitions.

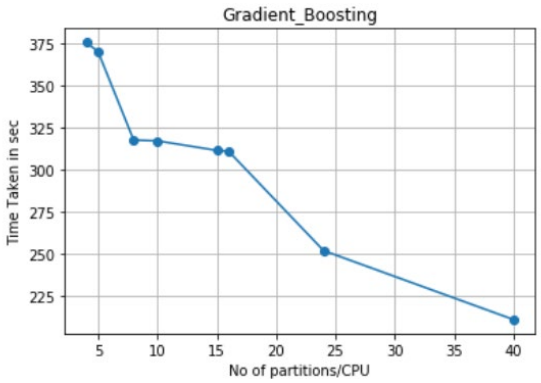

Figure 13: GBT Performance

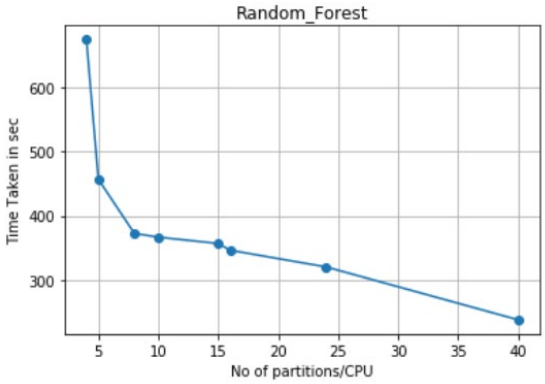

Figure 14: Random Forest Performance

As conceived in our experiments, Random Forest gave us the best accuracy at ~85%. While completely training the data and tuning the hyperparameters, there was a significant speedup achieved while exploiting parallelism. Moreover, Random Forest was a complex model projects a branching

nature. There was no overhead time observed here instead steadily decreasing with the increase in partition size. Since all the algorithms we used were cross validated against 4 folds, we were able to demonstrate healthy accuracy with significant time constraint reduction.

Fig 15 shows the time taken to compute all algorithms by varying the partitions to CPU. As mentioned earlier we matched the partition to CPU to make sure the ratio and parallelism was maintained. Based on the scalability available in the cluster, we experimented with N = 2, 5, 8, 15,16, 24 all the way to 40 partitions to check the speedup. We have visualized all the algorithms along with their respective values.

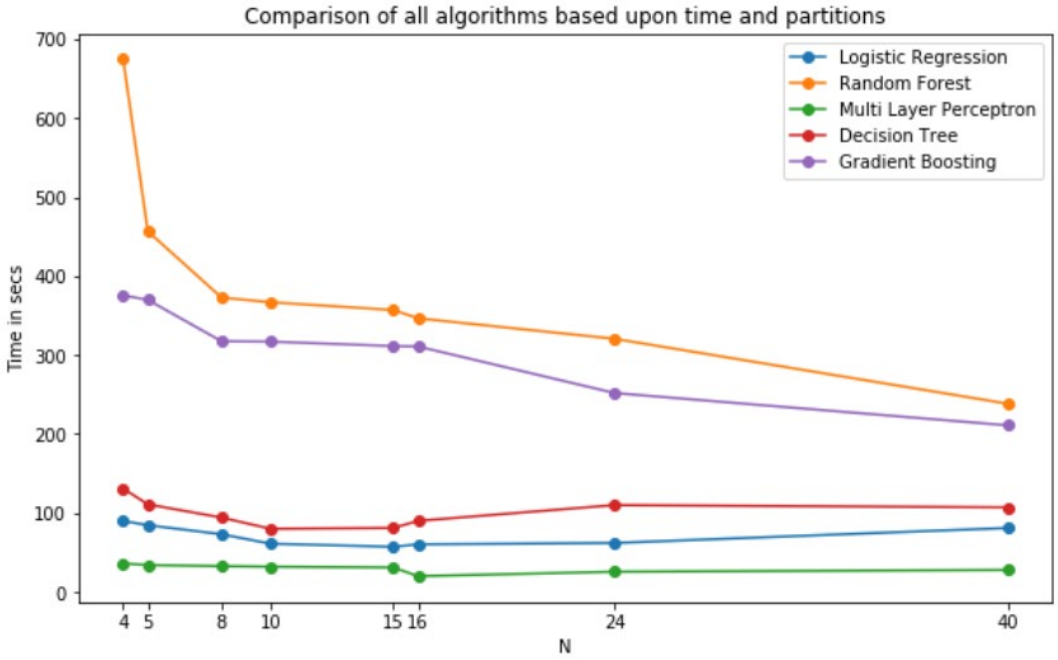

Figure 15: Comparison of Parallel Performance

On an overall analysis, we have seen that there is a nice decline in time taken up to 15 partitions which shows parallelism was effective. However, on increasing it further to 40 machines and 40 partitions, the timings slightly pick up due to extra communication cost while performing computation in the parallelism process such as while partition. A lot of memory gets allocated into performing these iterations and hence we noticed a visible overhead in cases when partition size was given more than required for respective models.

We also wanted to study the effect of parallelism on varying the size of the data set. Since we had 5 years' worth of data with each year having approximately 5000 rows, we compiled the data frames for each year in the form of a list and passed it along the Random Forest Algorithm. We decided to evaluate our model from a single machine single partition to 20 machines and 20 partitions to obtain the graph below.

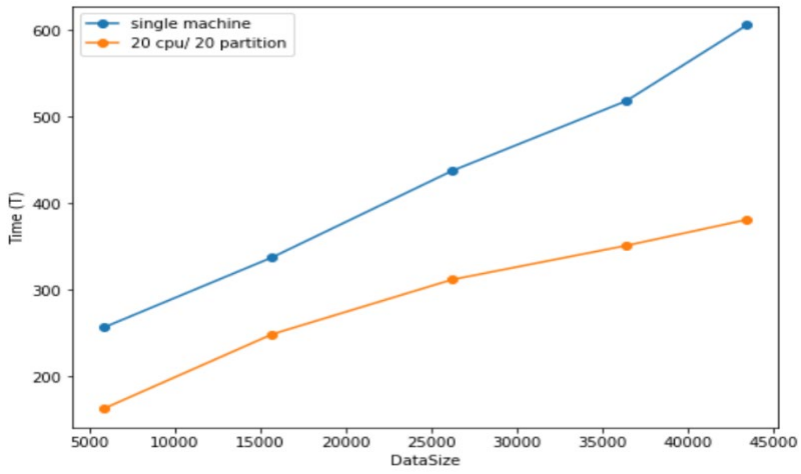

Figure 16: Dataset size vs time for RF

A significant time reduction in contrast to a single machine, single node was noticed. Therefore, the effect of parallelism in solving the problem statement was visible.

IX. Conclusion

Through the depth and extensive use of parallelism in our project, we were able to achieve good results that helped in predicting bankruptcy of the company, thus used parallelism within machine learning models to reduce company risk there by benefiting stake holders. By using key concepts from the course, we were able to successfully apply them to our problem statement and optimize the results obtained. *Random Forest* gave us the best accuracy and recall and overall showed a decreasing time curve while exploiting parallelism. We could conclude this because as per the course learning to decide which model performs the best, we did count the factors such as model complexity and feature selection by model analysis, cross validation evaluation metrics by validation analysis and time execution by parallelism and scalability analysis.

X. Contributions

The project was conducted by Akanksha, Aditya and Nihal together. Akanksha was responsible for digging dataset and converting it into usable format. Aditya and Nihal were responsible for analyzing the dataset by performing EDA. They also analyzed the binary class problem performed feature extraction along with pre-processing and designing the Machine learning pipeline including Imputing, Standardizing and SMOTE sampling. Nihal, Aditya, and Akanksha worked extensively on parallelizing the data using spark data frames and employing the parallel workflow described under V. Along with extensive EDA, Nihal was responsible for building the metrics evaluation to obtain the recall, F1 score and precision. In terms of Model evaluation, the first model was Logistic Regression that Aditya worked extensively on. This model provided hindsight into the flow described in V, which he was able to streamline and optimize. Apart from debugging and training the model, he also implemented the K-fold validation to tune the hyper-parameters. Apart from the Logistic Regression

model, Aditya also worked on implementing *Multi-layer-perceptron* along with its respective K-fold cross validation. Akanksha worked on getting the Gradient-Boosted Tree along with the Decision-Tree evaluation done along with tuning the hyperparameters, optimization, K fold cross validation and its evaluation metrics. She also worked on visualizing and performing the partition to CPU evaluation for these models. Nihal worked on setting up Random Forest along with performing critical analysis and fine tuning of its hyperparameters to obtain optimal recall and accuracy. Overall, the team contributed equally by validating and providing inputs to each other in solving this complex problem statement.

XI. Future Scope:

Given the very nature of the problem statement and its complexity in integrating real time data, the future scope involves trying to tune the hyperparameters further not just for Random Forest, but for the other algorithms as well. Also implementing this in real time would involve introducing a hybrid system capable of super-fast iterations.

XII. References:

1. Prediction of Bankruptcy of a company using machine learning techniques – IEEE 2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)
2. Bankruptcy Prediction for Banks: An Artificial Intelligence Approach to Improve Understandability – Springer (Studies in Computational Intelligence book series (SCI, volume 427)
3. http://archive.ics.uci.edu/ml/datasets/polish+companies+bankruptcy+data
4. https://rc.northeastern.edu/
5. https://northeastern.instructure.com/courses/137045/modules : Lecture Notes of EECE5645 Parallel Processing for Data Analytics lecture 8 and 9
6. https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147
7. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
8. https://www.nvidia.com/en-us/glossary/data-science/xgboost/
9. https://www.geeksforgeeks.org/ml-gradient-boosting/
10. https://northeastern.instructure.com/courses/137045/modules : Lecture Notes of EECE5645 Parallel Processing for Data Analytics lecture 10
11. https://spark.apache.org/docs/latest/ml-guide.html
12. https://medium.datadriveninvestor.com/decision-tree-and-random-forest-e174686dd9eb
13. https://www.semanticscholar.org/paper/Study-the-Influence-of-Normalization-Transformation-Raju-Lakshmi/05a4df33ffc7b2fdbf09b98d9585d80b35241425
14. https://arxiv.org/abs/1106.1813