Company Bankruptcy Prediction

Hui Zheng Kaiyan Xu Seung Min Lee Chun Cheng
Department of Computer Science
Boston University
Boston, MA 02215
{tommzy,stevenhh,cicheng}@bu.edu
seungminlee1998@gmail.com

Code: https://colab.research.google.com/drive/1cF-aT9qI0L3MVWHeccklNjjIe0x_11WL?usp=sharing

Abstract

This study examines machine learning techniques to predict company bankruptcy. A 2009-2019 financial record dataset of 6819 companies from Taiwan Economic Journal was used for training and testing the models. LDA (Linear Discriminative Analysis) was selected as the feature extraction methods after comparing with other methods. The classifiers and algorithm used to classify the data included logistic regression, Perceptron, Naive Bayes classifier, shallow neural network, and support-vector machine. Among all the classifiers, Perceptron and Naive Bayes had highest average f1 score and recall, so the 2 classifiers were thought as the best two models in predicting bankruptcy in this project.

1 Data preprocessing

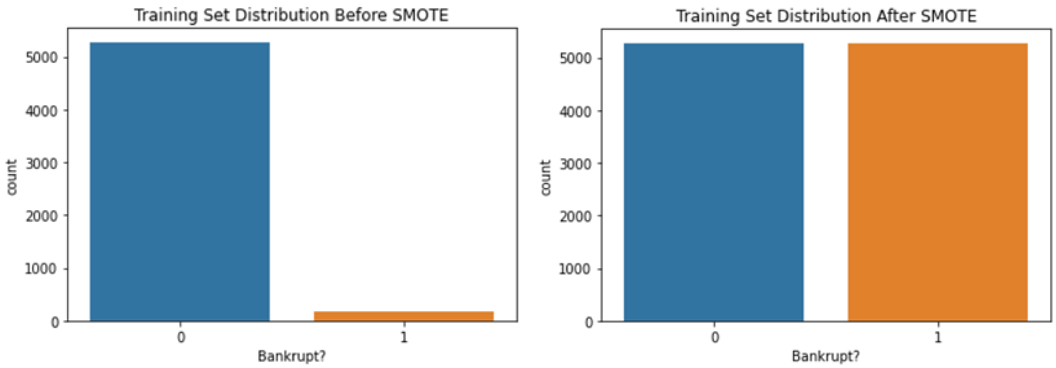
Our dataset contains 95 features. Among 6819 features, 220 went bankrupt and 6599 did not go bankrupt. There are two main issues in the dataset. Firstly, range of values are significantly different among different features. Most of the features varies from 0 to 1, but in 24 features, data have magnitude in millions. Large variance across features is problematic for classifiers based on distance measurement, like support vector machine. The effects of some features are completely dwarfed by others with larger scales, so the model is dominated by several features that may be statistically insignificant [1]. Also, when we later use feature extraction methods like PCA and LDA and classifiers like Naive Bayes, it is required that the data input should be normalized. Therefore, it is necessary to unify the scale of the features. Because most features follow Gaussian distribution, we standardize all the features by calculating their z-score:

$$z = (x - \mu)/\sigma$$

After standardization, each feature was distributed with mean 0 and standard deviation 1.

The second issue was the imbalanced number of target variables. Among 6819 cases, only 220 companies went bankruptcy. If the algorithm is trained on the original imbalanced data, it tends to ignore minority cases (bankruptcy) and predict more majority cases (not bankruptcy). In extreme case, if our models just return majority class as the prediction, the accuracy will still be 96.8%, but the model cannot draw attention of those companies that have financial crisis to its potential bankruptcy. Therefore, such a high accuracy model is not useful and accuracy is not a good metric to evaluate the model. To conquer the problem, it is required to balance the data so that the training process takes equal weight on both bankruptcy and not bankruptcy cases. In this case, an oversampling method, SMOTE (synthetic minority oversampling technique), was applied on the training set. By selecting pairs of minority class observations and creating synthetic points that lie on the lines between the

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) (b)

Figure 1: (a) Training set distribution before SMOTE; (b) Training set distribution after SMOTE

pairs, SMOTE increases the number minority class examples and enables the imbalanced dataset to become balanced. Figure 1 shows the training samples before and after using SMOTE.

2 Feature selection and extraction

To decide the input of training model, the biggest issue we need to consider is dimensionality. Training with more features in a high dimension space tends to decrease loss on the training set but increase parameter uncertainty, and finally increasing dimensions leads to overfitting. Therefore, we should include as many features or components as possible but stop right before the dimension leads to overfitting. In this project, we try one feature selection method and two feature extraction methods. Feature selection methods pick the best one or several features and discard the rest of the original features, while feature extraction methods consider all the features and find components that best represent the combination of all the original features. By picking the best set of components, feature extraction methods reduces the dimension to speed up training speed and avoid overfitting. Another benefit of conduting dimensionality reduction is that it makes the prediction more stable so that a minor change in feature will not influence the prediction a lot.

To test which method is best at determining input, each method with different hyperparameters is trained and validated with the most simple logistic regression for 10 iterations. The final method is chosen based on the highest average f1 score.

2.1 SelectKBest

SelectKBest is a function in the feature_selection module of scikit-learn package. It selects top k features that have highest mutual information with target. Higher mutual information means higher dependence between the features and the target. Therefore, the selected features are likely to be statistically significant for prediction. From 1 to 25 highest mutual information features, the highest validation f1 score was at 9 features. After 9 features, the model began to overfit. Therefore, the best number of features was 9 features. The average f1 score at this hyperparameter was 0.291.

2.2 PCA

PCA (Principal Component Analysis) is an unsupervised learning dimensionality reduction method that aims to find components that best summarize the original datasets. By maximizing variances and minimizing the reconstruction error by pairwise distances between features, original data is projected into a set of orthogonal axes ranked by importance. By keeping the most important components, PCA keeps the most valuable part and uses a smaller dimension to represent the original features. This way, PCA gives prediction with high accuracy and significantly shorten the training time. Another benefit of using PCA is that each new component is independent of each other, so there is no multicollinearity issue to worry about.

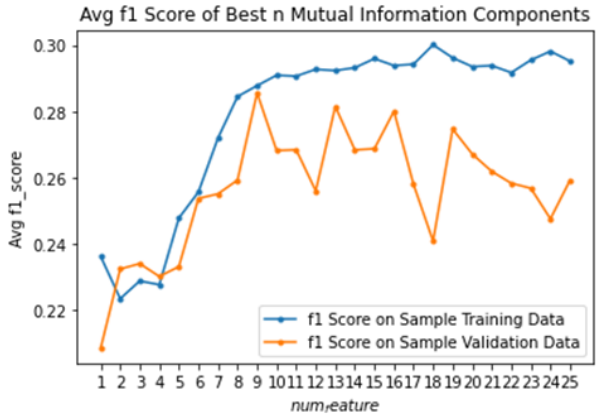


Figure 2: Avg F1 score vs. best n nutual information components

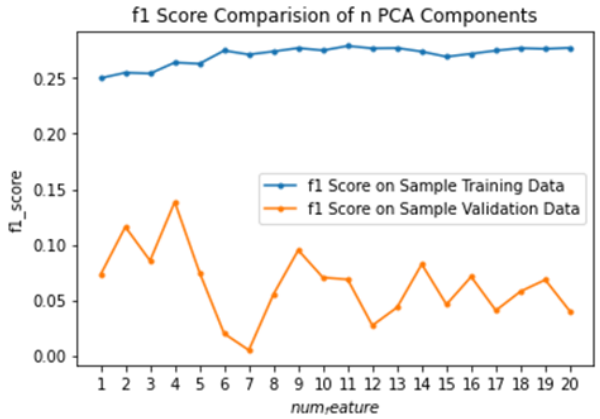


Figure 3: F1 score vs. n PCA components

We validate on the number of components from 1 to 20. When the number of components equals to 4, the validation set has the highest average f1 score in 10 iterations. Under this best hyperparameter, PCA has the f1 score 0.145.

2.3 LDA

LDA (Linear Discriminative Analysis) is a supervised learning dimensionality reduction method that finds the linear combination of the predictors such that between-class variance is maximized relative to the within-class variance [2]. To do so, when there is no linear hyperplane that separates the classes completely in high dimension, LDA projects high dimensional data into lower dimensional space. Because the number of components in LDA has to be lower than the minimum between number of features and number of classes, the only number of components that we can choose is 1. After running Logistic regression with the input from LDA for 10 iterations, the average f1 score at validation set is 0.364.

2.4 Feature selection method choice and workflow

Therefore, among all the three feature selection and feature extraction methods, LDA with number of component 1 returned the highest average f1 score. To make our comparison consistent when testing all the following classifiers, we would use LDA to generate input for all the classifiers.

Each model was trained and tested with the same workflow. Inside each model, we changed different hyperparameters and tested them using validation set to pick the best hyperparameters. Each

model and each hyperparameter were trained and validated in ten iterations. Among each iteration, we splitted training set and validation set again. In the end, we used testing set to compare the performance of different classifiers with their best hyperparameters. We picked the best classifier based on f1 score and recall for minority class. The reason why we specifically focused on recall of bankruptcy cases instead of precision was that recall gave the proportion of actual bankruptcy correctly classified. We wanted to be very cautious and classified more problematic companies as bankruptcy rather than missed a real bankruptcy company. Even if this criteria favors classifiers that tend to mislabel non-bankrupted companies as bankrupted, but after receiving the bankruptcy prediction, accountants and managers can further investigate the financial status of the company and clarify if the risk indeed exist. However, if we miss a bankrupted company, the loss will be giant and harmful.

3 Linear models

3.1 Logistic regression

The training data was fitted using the logistic regression classifier. The class, "sklean.linear_model.LogisticRegression," was used to implement the classifier [3]. LogisticRegression calculates w^* that mininmizes the cost function,

$$\underset{w}{\text{argmin}} C \sum_{i=1}^{n} (-y_i \log(p(X_i)) - (1 - y_i) \log(1 - p(X_i))) + r(w)$$

and then uses w^* to make predictions $\hat{y}$ [4]. The process of calculating w* differs depending on the hyperparameters, including the choice of the regularization term and the optimization algorithm. In order to select the hyperparameters that optimize the performance of our model, each combination of compatible hyperparameters was evaluated using a validation data set. The hyperparameters that were included for evaluation were the optimization algorithm, "solver," the inverse of regularization strength, "C," and the regularization term, "penalty." The compatible combinations of "solver" and "penalty" included "newton-cg" with penalty options "l2" and "none," "lbfgs" with penalty options "l2" and "none," "liblinear" with penalty options "l1" and "l2," "sag" with penalty options "l2" and "none," and "saga " with penalty options "l1," "l2," "elasticnet," and "none" [3]. The inverse of regularization strength was evaluated with values 1.0, 0.1, 0.01, and 0.001.The performance of each combination of hyperparameters was evaluated using the F1 score. After comparing the performance of different hyperparameters, an inverse of regularization strength of 0.001, the optimization algorithm, "libfgs," with penalty "l2" were chosen as the best hyperparameters. By fitting the training data on the logistic regression model with the best hyperparameters chosen, an accuracy of 0.9326, a precision of 0.3125, a recall of 0.9091, and an F1 score of 0.4651 were achieved. Among the companies that did not go bankrupt, 93.3333% was correctly predicted. Among the companies that went bankrupt, 90.9091% was correctly predicted. The resulting confusion matrix is shown in Figure 4.

3.2 Perceptron

The training data was fitted using the Perceptron classifier. The class, "sklean.linear_model.Perceptron," was used to implement the classifier. Perceptron utilizes stochastic gradient descent to calculate the w^* that mininmizes the loss, and then uses w^* to make predictions $\hat{y}$. The gradient descent algorithm for Perceptron has a learning rate of 1 [3]. In order to select the hyperparameters that optimize the performance of our model, each combination of compatible hyperparameters was evaluated using a validation data set.. The hyperparameters that were included for evaluation were the regularization term, "penalty," and the constant that multiplies the regularization term, "alpha." "alpha" was evaluated with values 1.0, 0.1, 0.01, and 0.001, and the regularization terms evaluated were "l1," "l2," "elasticnet," and "none." The performance of each combination of hyperparameters was evaluated using F1 score. After comparing the performance of different hyperparameters, penalty "l1" and alpha 0.000001 were chosen as the best hyperparameters. By fitting the training data on the Perceptron model with the best hyperparameters chosen, an accuracy of 0.9619, a precision of 0.4524, a recall of 0.8636, and an F1 score of 0.5938 were achieved. Among the companies that did not go bankrupt, 96.5152% was correctly predicted. Among the companies that went bankrupt, 86.3636% was correctly predicted. The resulting confusion matrix is shown in Figure 5.

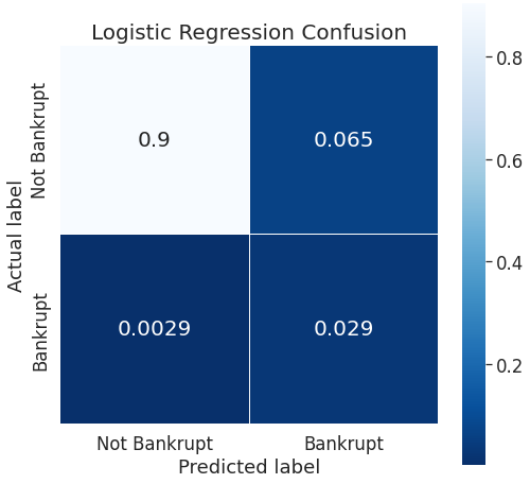
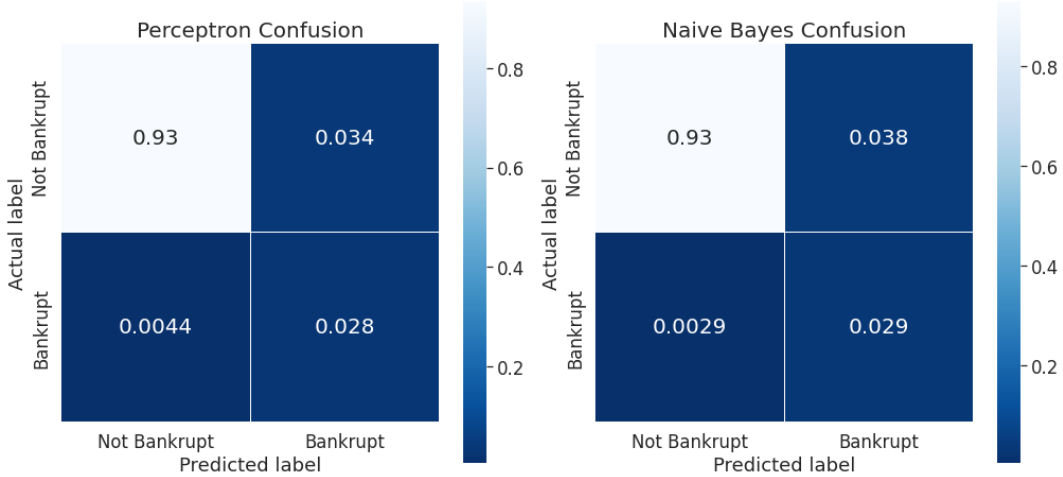


Figure 4: Logistic regression confusion matrix.



(a) (b)

Figure 5: (a) Perceptron confusion matrix; (b) Naive Bayes confusion matrix

3.3 Naive Bayes

The training data was fitted with the Gaussian Naive Bayes algorithm. The class, "sklean.naive_bayes.GaussianNB," was used to implement the classifier. For GaussianNB, the likelihood was assumed to exhibit a Gaussian distribution with the mean and variance estimated using MLE [4]. The prior was adjusted according to the training data. The likelihood and the prior were applied to Bayes' rule to maximize the posterior [4]. By fitting the training data on the Naive Bayes model, an accuracy of 0.9589, a precision of 0.4348, a recall of 0.9091, and an F1 score of 0.5882 were achieved. Among the companies that did not go bankrupt, 96.0606% was correctly predicted. Among the companies that went bankrupt, 90.9091% was correctly predicted. The resulting confusion matrix is shown in Figure 5.

3.4 Comparison of the linear models

The Perceptron was the most effective model in predicting if a company would not go bankrupt. The Logistic Regression and Naive Bayes models were the most effective in predicting that a company

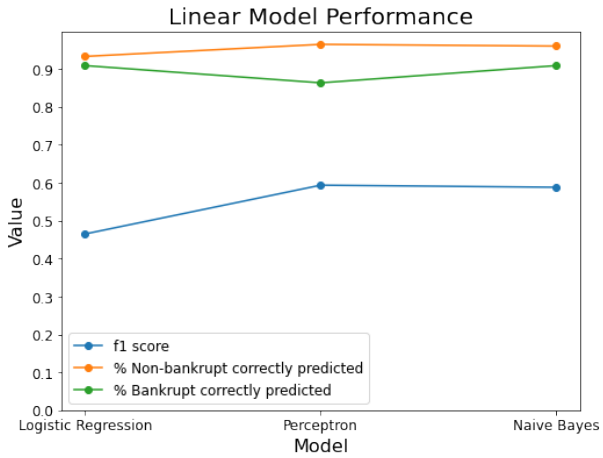


Figure 6: Linear model performance.

would go bankrupt. Overall, all of the linear models performed well in bankruptcy prediction. The performance of the linear models is shown in Figure 6.

4 Shallow neural network

The neural network implementation is based on TensorFlow Keras API.

A 4-layer neural network would be enough for this problem. The first 3 layers use relu as activation function. Since the problem is about binary classification, last layer activation applied sigmoid as activation function.

4.1 Prepare for model parameters tuning

Random variables play an essential role when training a model. Therefore, during model parameter tuning, it would be ideal to fix the random generated variables and get reproducible results each run.

In order to do that, it's necessary to set three random seed at the beginning of the code: Python random seed, Numpy random seed, and Tensorflow random seed [5].

4.2 Model parameters tuning

There are 4 parameters that need to be tuned: epochs number, batch size, optimizer, learning rate.

4.2.1 Epochs Number

Epochs means how many times we processed the dataset. From 1 to N, if the trend of the loss of the epoch is approximate to constant. Then epochs number, which to the minimum value that the trend is approaching to flat, would be the ideal value. As shown in Figure 7, after epoch = 50 the loss is approaching the minimum.

4.2.2 Batch size

Batch size represents how many small packages that the training date gets divided into. Batch sizes of 50,100,200,300,400,500 were used for experiment. Apply the same strategy like what we use in finding Epochs Number. We plotted all figures from the experiment together to identify which size produced the best performance over our data. According to Figure 8, we found that the orange line(batch size = 100) loss distribution was better, and it had fewer oscillations.

4.2.3 Optimizer

Optimizer was chosen from SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, and Ftrl.

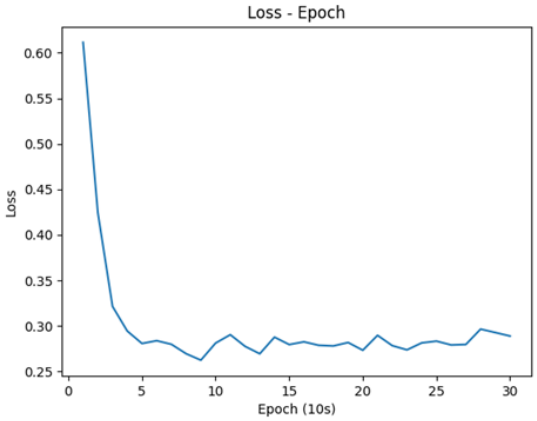


Figure 7: Loss vs. epoch

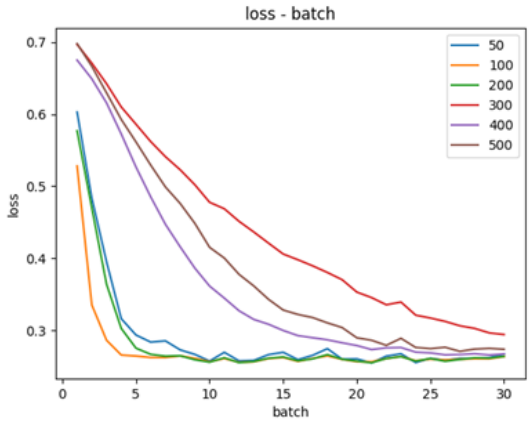
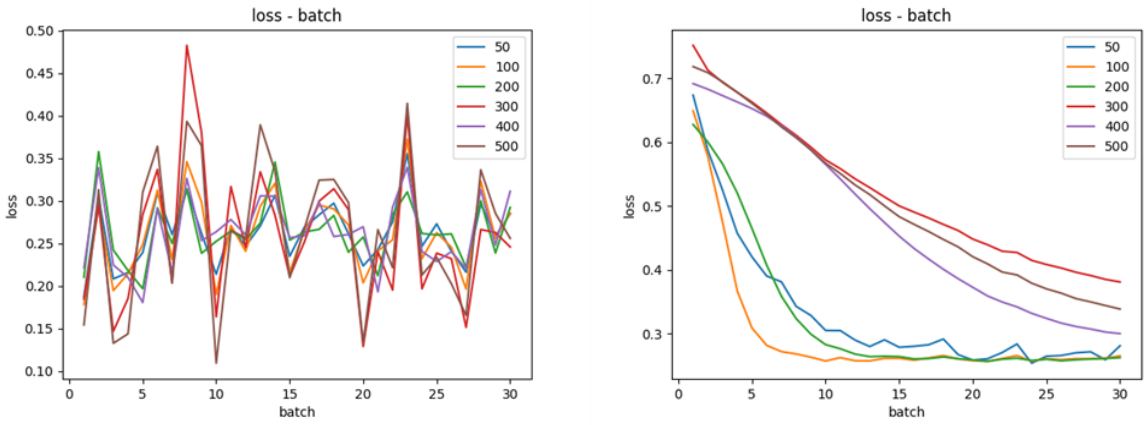


Figure 8: Loss vs. batch with Adagrad and learning rate=0.002



(a) (b)

Figure 9: (a) loss vs. batch with RMSprop; (b) loss vs. batch with SGD

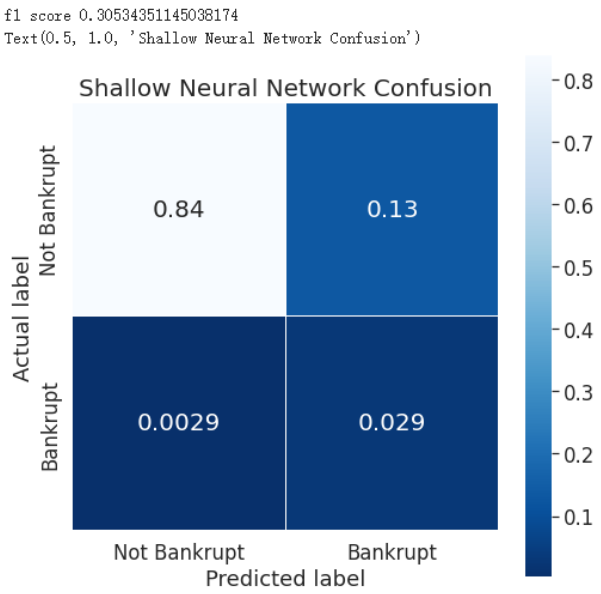


Figure 10: Shallow neural network confusion

The ideal optimizer would yield less oscillation and better loss distribution. RMSprop, Adam, Adadelta,, Adamax, Nadam, Ftrl all produced heavy oscillations.

As shown in Figure 9 and Figure 8, Adagrad and SGD produced fewer oscillations. Adagrad produced better loss distribution with batch size = 100.

4.2.4 Learning rate

As shown in Figure 8, using certain value of learning rate yielded better loss distribution.

When Learning Rate = 0.002, it had less oscillation and acceptable loss distribution.

The values chosen for the parameters are: Optimizer=Adagrad, Learning rate = 0.02, epochs number = 50, batch size = 100. The model was then tested using the chosen parameters and the testing set. Results are shown in Figure 10.

5 Support-vector machine

The training data was fitted using the Support Vector Machine (SVM) classifier. The class, "sklean.svm.SVC" was used to implement the classifier. The SVM classifier employs the following kernels for classification: linear model, polynomial, and sigmoid. The main objective of a kernel is to perform transformation on the input data of a dataset and convert it into a particular form that is more easily separable, generally in higher dimension. Linear models produce linear maximum-margin hyperplanes that are more useful when working with a smaller number of features while polynomial kernels are more beneficial in separating classes when using a non-linear hyperplane [6]. The sigmoid model is generally preferred in neural networks and is an equivalent to a two-layer perceptron model that utilizes sigmoid as its activation function. There are other hyperparameters that could also be fine-tuned other than the kernel such as the regularization parameter in python's Scikit-learn C parameter or the gamma term which is the kernel coefficient for "poly" and "sigmoid" [3]. In general, SVM classifications offer higher accuracy and compute predictions earlier with less memory compared to the Naive Bayes model. However, SVM models, especially when using the Scikit-learn package, are less useful when the dataset gets significantly larger, which in turn renders the training time of SVM to become considerably higher than that of a Naive Bayes.

Given our dataset, the sigmoid kernel resulted in the highest f1 score of 0.492 while the linear kernel resulted in 0.300 and the polynomial kernel resulted in 0.378. Thus, the sigmoid kernel was selected

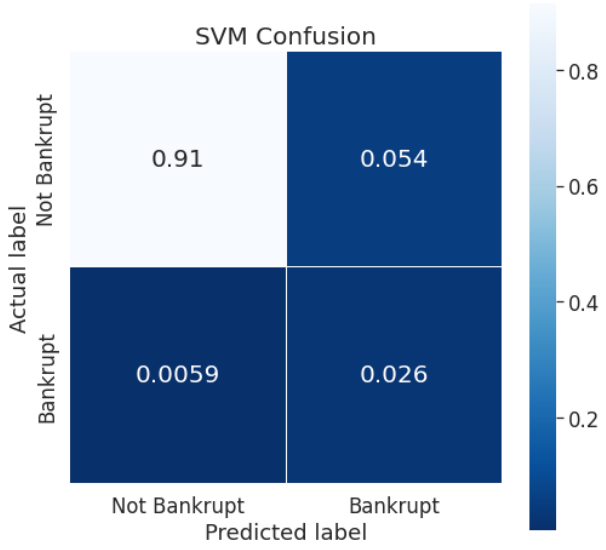


Figure 11: SVM confusion matrix.

as our hyperparameter. By fitting the training data on the Support Vector Machine (SVM) model with the best hyperparameter chosen, an accuracy score of 0.940, a precision score of 0.327, a recall score of 0.818, and an f1 score of 0.468 were achieved. The resulting confusion matrix is shown in Figure 11.

6 Conclusion

Based on our data set and goal, linear models demonstrated pretty good performance. Shallow neural network had decent performance, but it was not as effective as linear models according to the F1 score and recall score. Even though the neural network was a more complex model, with good data preprocessing methodology and result, linear models provided better results. It explains the reason why they are popular for so many years.

7 Future work

Our focus of future work is to try better feature extraction method. Since our dataset is not complicated and many classical ML classifiers do not perform worse than neural network, we speculate that adding more advanced algorithm may not significantly improve the performance. Therefore, to gain higher performance, we have to refine our preprocessing and feature extraction method.

References

[1] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 2016.

[2] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning

software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Scikit-learn svm tutorial with python (support vector machines).