Intelligent Systems

ITCS 6150-051

Credit Scoring with Machine Learning

Submitted by

Kha Tran (801222949)

Jonathan Lorray (801045949)

Zachary Pacello (801054565)

Srilakshmi Neha Yalamanchili (801254203)

Submitted to

Dr. Ali Sever

<u>Abstract</u>

When it comes to credit risk assessment, credit scoring is a critical classification task to discriminate "bad" applicants from "good" applicants for financial institutions. Here we present our attempt to create a web based application for quick and easy estimation of one's credit risk. While many financial institutions use logistic regression as a means of estimating an applicant's credit risk, our application makes use of other machine learning methods to deliver a more accurate and comprehensive estimate of one's credit risk. We found that gradient boosting classification and neural network classification performed the best. The neural network model was implemented in our web application.

<u>Introduction</u>

Default loss is one of the major concerns for banks when lending loans. To process large volumes of credit agreements, it takes a lot of human resources and does not scale well. Our goal is to build a model able to provide an estimate of probability risk score given information from a credit applicant. The application will then send this data as input to the model, via API call, and the output of the model/probability risk score will be provided to the user. This probability risk score will help businesses determine the risk associated with their financial decisions

The users of the system will be businesses who are interested in obtaining their probability risk score. The system will be incorporated into a web based application that users can interact with. The interaction will include forms for the user to complete and these forms will include information relevant to obtaining the probability risk score.

Data

The dataset was sourced from Kaggle Database which consists of 150000 borrowers for the training dataset and 101503 samples for the testing dataset. Features in the dataset include age, serious delinquency in 2 years, number of times the borrowers are 30-59, 60-89, and more than 90 days past due, debt ratio, monthly income, and so on. Overall, there are 12 initial features.

Data processing

From the initial training dataset, we observed and removed column "unnamed" which was intended as an id number and unnecessary for training the model. There are null data in some feature columns which were replaced using technique of imputing missing data with either median or most frequent values. The same was done on the test samples as well. In addition, we observed the distribution of borrowers' age mostly over 20 which makes sense as there's age restriction for borrowers. Going through features "NumberOfTimes90DaysLate", “NumberOfTimes60-89DaysPastDueNotWorse”, and “NumberofTime30-59DaysPastDueNotWorse”, we saw values of 96 and 98 are same for all three features which did not make sense and can affect how the model can confidently predict. Next, values for ‘RevolvingUtilizationOfUnsecuredLines’ are distributed over wide range, but 25% of delinquency is concentrated between 0.4 and 10. Similar, we also observed values of ‘Debt Ratio’ is mainly distributed over small range. These features contained noisy data; hence we would remove or replace them.

Performance metrics

Since the label is skewed heavily with roughly 94% with label of 0 and 6% with label of 1, we recognized that prediction accuracy is quite a poor metric to measure the model performance. In fact, most of our models achieved greater than 90% accuracy. It made sense since the model could just predict 0 and still got 94% accuracy. Therefore, we included other metrics such as precision, recall, f1 score, and AUC score to compare models’ performance.

- **Precision**: the proportion of correct positive identification
- **Recall**: the proportion of correct actual positive identification
- **F1 score**: balance between precision and recall score
- **AUC score**: measure the ability of a classifier to distinguish between classes (1 and 0 in this case) with respect to true positive and false positive. Higher score means that the classifier is more likely to predict positive as true positive.

Specifically, AUC score was the primary metric we used to determine how confidently the model can predict the probability of a borrower default.

Model selection

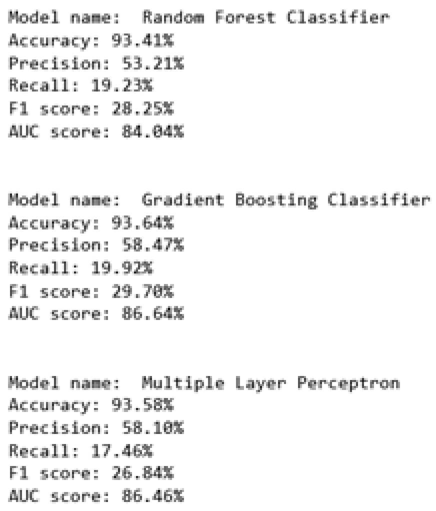
We tested multiple models in order to select the best performing model. Random forest, logistic regression, gradient boosting, and multiple layer perceptron (MLP), or neural network, classifiers were selected initially. Imputed data were split and fed to the models for training. Initial results showed logistic regression and MLP were the worst performers. The MLP classifier, however, performed much better after we standardized the training data. We decided to remove the logistic regression classifier and went ahead with the rest.

Feature Extraction

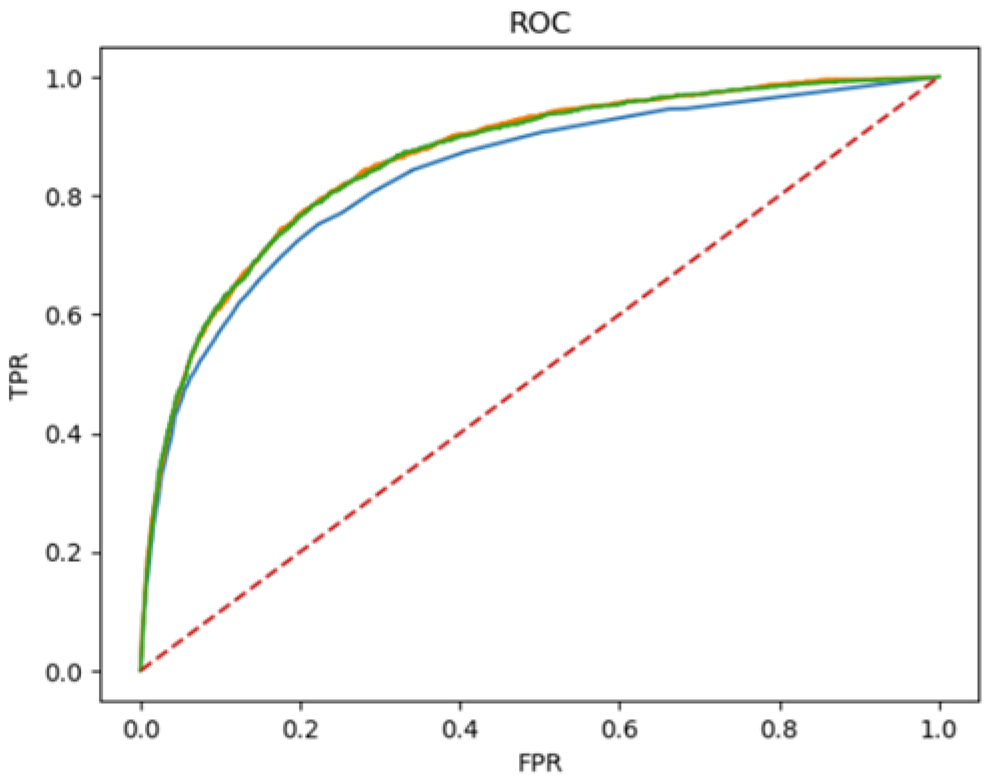
Based on our data analysis, we decided to manipulate data certain features in order to increase our model's performance. We replaced all value of over 95 in the features "NumberOfTimes90DaysLate", "NumberOfTimes60-89DaysPastDueNotWorse", and "NumberofTime30-59DaysPastDueNotWorse" to 20. We also replaced values in 'RevolvingUtilizationOfUnsecuredLines' that were greater than 10 to only 10 in order to lower the noise. Only samples with age of over 20 were kept as well as debt ratio in the quantile of 0.95. In addition, new feature was introduced by combining "NumberOfTimes90DaysLate", "NumberOfTimes60-89DaysPastDueNotWorse", and "NumberofTime30-59DaysPastDueNotWorse" together.

Model performance

We noticed an increase in all three models after the process of feature extraction. The AUC score increased from 84.11% to 84.17%, 86.49% to 86.64%, and 83.58% to 86.46% for random forest classifier, gradient boosting classifier, and MLP classifier respectively. Specifically, MLP classifiers saw significant boost in performance after feature extraction and achieved comparable performance to that of gradient boosting classifiers. Since we are more familiar with neural network models, we decided to implement the MLP model in our web application. It was, however, altered somewhat since we used tensorflow and keras libraries rather than sklearn library which was used for the model selection process.



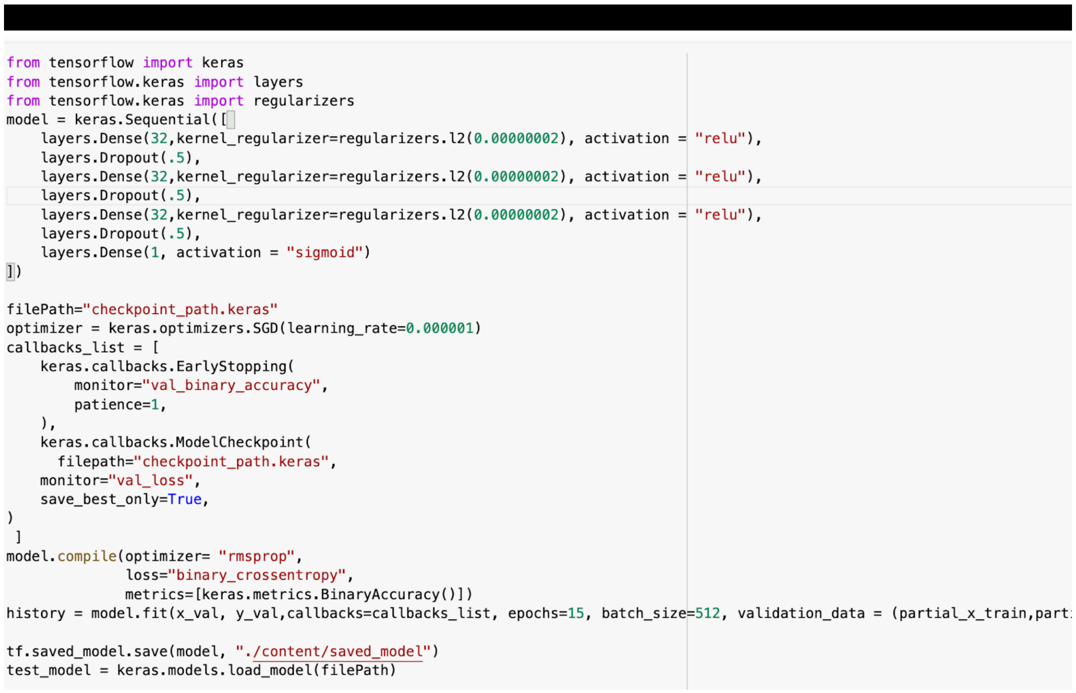
```
Model name:  Random Forest Classifier
Accuracy: 93.41%
Precision: 53.21%
Recall: 19.23%
F1 score: 28.25%
AUC score: 84.04%

Model name:  Gradient Boosting Classifier
Accuracy: 93.64%
Precision: 58.47%
Recall: 19.92%
F1 score: 29.70%
AUC score: 86.64%

Model name:  Multiple Layer Perceptron
Accuracy: 93.58%
Precision: 58.10%
Recall: 17.46%
F1 score: 26.84%
AUC score: 86.46%
```



```
AUC score of Random Forest Classifier: 83.52%
Confusion Matrix
[[26291   287]
 [ 1574   349]]


AUC score of Gradient Boosting Classifier: 86.24%
Confusion Matrix
[[26352   226]
 [ 1558   365]]


AUC score of Multiple Layer Perceptron: 86.08%
Confusion Matrix
[[26330   248]
 [ 1571   352]]
```

Deep Learning Model

The model architecture we chose for the neural network was a sequential model with a stack of 3 dense layers, each followed by a dropout layer and finally a dense layer with one unit and a sigmoid activation. We faced many obstacles while building the model, for example, the loss value remaining constant and not decreasing. This was due to many issues, the main one being the choice of optimizer. Initially, we tried SGD, however, RMSprop was much more suited to the problem. Also, the constant loss value resulted because of the learning rate being too low. Another issue that we encountered was overfitting to the validation data. Furthermore, this was the most difficult problem to overcome because a small network resulted in poor accuracy, but a large network ended up overfitting to the validation data. Additionally, the training accuracy began to stall because the network did not have enough capacity to model all relevant patterns in the training data. Eventually, we decided on a middle ground and have a small model, in order to prevent overfitting to the validation data.

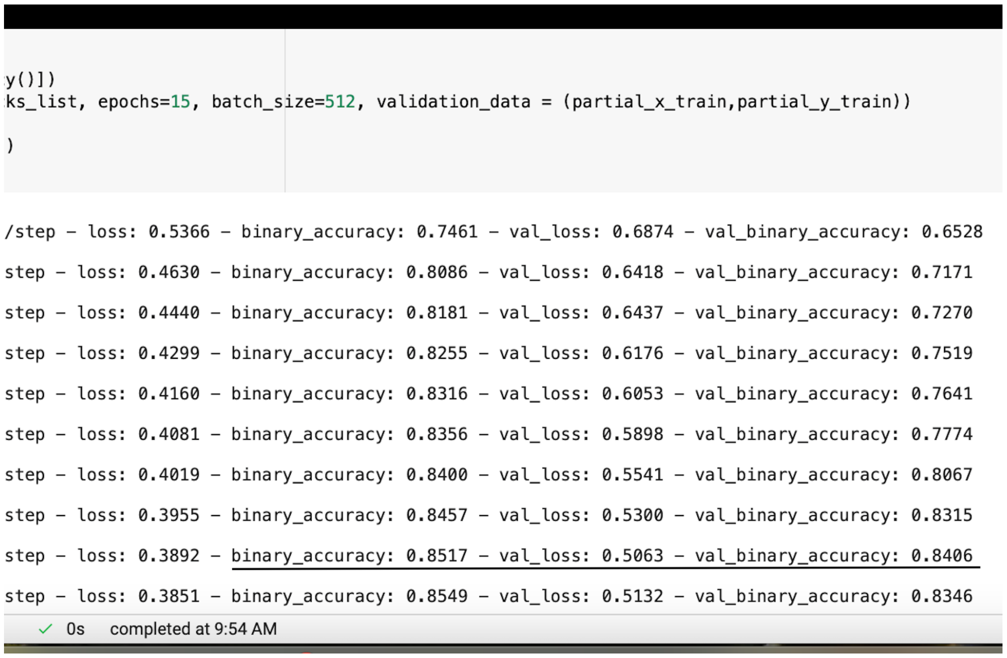


```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(32,kernel_regularizer=regularizers.l2(0.00000002), activation = "relu"),
    layers.Dropout(.5),
    layers.Dense(32,kernel_regularizer=regularizers.l2(0.00000002), activation = "relu"),
    layers.Dropout(.5),
    layers.Dense(32,kernel_regularizer=regularizers.l2(0.00000002), activation = "relu"),
    layers.Dropout(.5),
    layers.Dense(1, activation = "sigmoid")
])

filePath="checkpoint_path.keras"
optimizer = keras.optimizers.SGD(learning_rate=0.000001)
callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor="val_binary_accuracy",
        patience=1,
    ),
    keras.callbacks.ModelCheckpoint(
      filepath="checkpoint_path.keras",
    monitor="val_loss",
    save_best_only=True,
)
 ]
model.compile(optimizer= "rmsprop",
              loss="binary_crossentropy",
              metrics=[keras.metrics.BinaryAccuracy()])
history = model.fit(x_val, y_val,callbacks=callbacks_list, epochs=15, batch_size=512, validation_data = (partial_x_train,part

tf.saved_model.save(model, "./content/saved_model")
test_model = keras.models.load_model(filePath)
```

Querying the Models Predictions

To create the web application we used the Javascript framework, express. Also, we used TensorFlowJS to load the model from local storage and then call the model's predict method on the user's input data. Since the course is not focused on web application development the app only consists of a form that the user can query the model with. Once the input data is received on the server, the data is normalized just like we did for the training data. Finally, an asynchronous function call is computed to load the graph model file from local storage and query the model's predictions. Once the prediction has been computed, the result is sent back to the client/user.

Deep Learning Model Accuracy:



```
y()])
ks_list, epochs=15, batch_size=512, validation_data = (partial_x_train,partial_y_train))

)

/step - loss: 0.5366 - binary_accuracy: 0.7461 - val_loss: 0.6874 - val_binary_accuracy: 0.6528
step - loss: 0.4630 - binary_accuracy: 0.8086 - val_loss: 0.6418 - val_binary_accuracy: 0.7171
step - loss: 0.4440 - binary_accuracy: 0.8181 - val_loss: 0.6437 - val_binary_accuracy: 0.7270
step - loss: 0.4299 - binary_accuracy: 0.8255 - val_loss: 0.6176 - val_binary_accuracy: 0.7519
step - loss: 0.4160 - binary_accuracy: 0.8316 - val_loss: 0.6053 - val_binary_accuracy: 0.7641
step - loss: 0.4081 - binary_accuracy: 0.8356 - val_loss: 0.5898 - val_binary_accuracy: 0.7774
step - loss: 0.4019 - binary_accuracy: 0.8400 - val_loss: 0.5541 - val_binary_accuracy: 0.8067
step - loss: 0.3955 - binary_accuracy: 0.8457 - val_loss: 0.5300 - val_binary_accuracy: 0.8315
step - loss: 0.3892 - binary_accuracy: 0.8517 - val_loss: 0.5063 - val_binary_accuracy: 0.8406
step - loss: 0.3851 - binary_accuracy: 0.8549 - val_loss: 0.5132 - val_binary_accuracy: 0.8346
```

<u>Challenges</u>

As a starting point, the main challenge we faced was querying the models predictions via API call. The process to configure the server and enable the web application to query the models predictions was extensive. There were many dependencies that needed to be installed and some had to be a particular version. Not to mention, the server had to be started locally and by the time we diagnosed most of the errors we had limited time remaining. Nevertheless, we did set up a web application that queries the models predictions, however, it is done locally by using the model.json file in the local file system.

Next, another challenge that we encountered was the model configuration process. As a matter of fact, all of us were fairly new to deep learning and did not have much experience with implementing a deep learning model. A large portion of the model configuration process consisted of testing different values for the batch size, epochs, learning rate, number of layers, units per layer, etc. Furthermore, it required a lot of time to understand how all these parameters are correlated. For instance, if the batch size is too small and the learning rate is too high, then the gradient updates of the model may be very large, which would prevent convergence. Once we understood how different values for these parameters affected the model accuracy, then we were able to find an efficient model configuration.

Conclusion

The problem of our project was basically a classification problem. Many classification models were implemented for comparison. Our selected neural network classification model performed well with about 86% confidence that it could accurately predict the probability that the borrower would default in the next 2 years.