Loan Default Prediction

Youssef Ashraf Kandil
Computer Engineering
The American University in Cairo
youssefkandil@aucegypt.edu

Abdullah Mohamed Kassem
Computer Engineering
The American University in Cairo
Abdullahkassem@aucegypt.edu

Analysis criteria

In this project we are required to predict the loan defaults based on some features, which means that we have a classification machine learning problem composed of two classes 0 and 1 (where 1 label indicates a default). By analyzing the data we realized that the defaulters are the minority class in the dataset, implying that we need to consider the recall rather than the accuracy and precision, when measuring the models' performances.

In other words, in the banking field, the bank needs to minimize the percentage of fallaciously accepted loans. Which means that we need to consider minimizing the false negative predictions of our chosen models.

As discussed in the previous phase, we had a main issue with our chosen dataset which is the highly imbalanced nature of the data. This issue is very common with this type of classification problems, where the intended prediction class has a low probability of occurrence in real life. In our problem, it is rare for a loan default to happen. Specifically, the dataset is highly imbalanced, and has a minority class (defaulters) of about 25% of the data.

The issue of the class imbalance in classification problems is that any machine learning model tends to learn patterns and trends associated with the majority class, and almost totally neglects the minority class. A very common example to this is the email spam classification problem, where spam emails occur with a very low probability in real life. Our problem unfortunately has this nature as well.

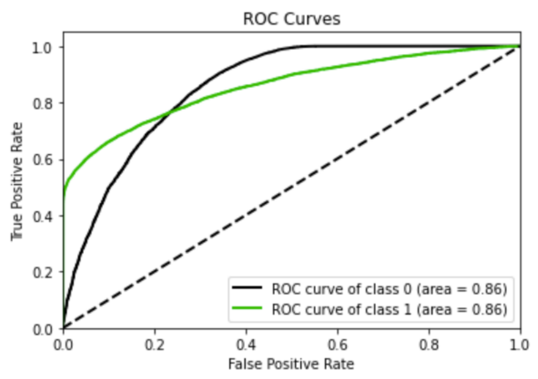
In the previous phase, we intensively analyzed the performance of all the classification supervised learning models we learnt in this course, and reported their performance based on the specified above metrics. Consequently, and based on the recall, precision, and roc curves, we found that the best performing models are :

1. Random forests
2. Decision trees
3. Neural networks

Choosing Model:

The Random forests showed the best results in the analysis phase, in terms of highest recall values for the minority class, and best ROC curves. However, the problem is that the best recall value was about 0.65, which is very low. This is definitely due to the highly imbalanced nature of the dataset as discussed before.

```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92     22319
           1       0.94      0.52      0.67      7415

    accuracy                           0.87     29734
   macro avg       0.90      0.75      0.80     29734
weighted avg       0.88      0.87      0.86     29734

array([[22070,   249],
       [ 3551,  3864]])
```



By the advice of the professor, we had to refactor the loss function of our chosen model to count for this abnormality, and to give a higher chance for the model to recognize patterns produced by the minority class. Moreover, this modification should reduce the tendency of the model to overfit, especially towards the minority class.

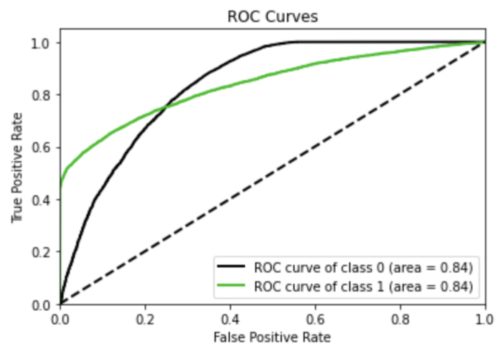
Our main plan was to give higher weight to the minority class's contribution to the learning process.

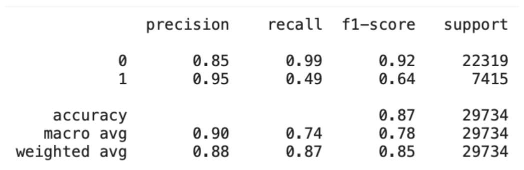
Random forests:

By searching the internet and referring to the cited literature, we realized that modifying the loss function of any random forest model, either ID3, CART, OR C4.5, is a very challenging mission. Since all these models are based on probabilistic learning algorithms, we were not able to introduce any weighted factors to their formulas in a way that only affects the contribution of the minority class. At the end of the day, we are not planning to change the probabilistic distribution of the data. The disadvantages of using SMOTE or near miss methods were discussed in the previous phase.

Thus we decided to shift to a neural network model, since that a gradient descent loss function better fits our classification problem.

Here are the analysis results of neural network model in the previous phase:





```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92     22319
           1       0.95      0.49      0.64      7415

    accuracy                           0.87     29734
   macro avg       0.90      0.74      0.78     29734
weighted avg       0.88      0.87      0.85     29734
```

Neural network:
As mentioned before, we want to introduce weights to each class that accounts for its proportion in the dataset. To do so, we introduced the following piecewise function, modifying the learning rate based on the class.

Loss function:

<u>Original loss function:</u>

$$W_i(t+1) = W_i(t) + \eta(Y - \overline{Y})X$$

Where:
ɳ is the learning rate

<u>Modified loss function:</u>

$$W_i(t+1) = \begin{cases} W_i(t) + \alpha(Y - \overline{Y})X & Y=0 \\ W_i(t) + \beta(Y - \overline{Y})X & Y=1 \end{cases}$$

Where:
α is the learning rate for the majority class
β is the learning rate for the minority class

The ratio between the new learning rates (**α** and ***β***) should be close to the ration between the classes distribution in the dataset. Thus we assigned the following ratio:

α	:	***β***
1	:	4

This new loss function should be biased towards learning more information from the minority data points (defaulters). The values for **α** and ***β*** will initially be (0.01, and 0.04) respectively, however, we will do cross validation over their values in the next phase after modifying the loss function to reach optimality.

<u>**Cross validation:**</u>

In the last phase, we did a brief cross validation to find the optimal set of parameters for the neural network, and in this phase we decided to do a comprehensive cross validation, experimenting with a wider set of parameters. Aiming to find the best activation function, number of hidden layers, and number of nodes. The following set of parameters where permutated:

```
parameters_two={
'Hidden_layer_sizes':[
(2,2,2),(2,3,4),
(3,3,3,3),(3,3),
(5,5,5,5),(2,10,2),
(8,8,8), (5,6,7)
],
'max_iter': [500, 800],
'activation':['logistic','tanh','relu],
'learning_rate': ['constant'] # set as
constant since we will change the loss
function and learning rate, thus no need
to check optimality here}
```

Hyperparameters:

The best performing set of parameters was:

```
SMLPClassifier ( activation = 'tanh' ,
                 hidden_layer_sizes = ( 8, 8, 8),
                 max_iter = 800 )
```

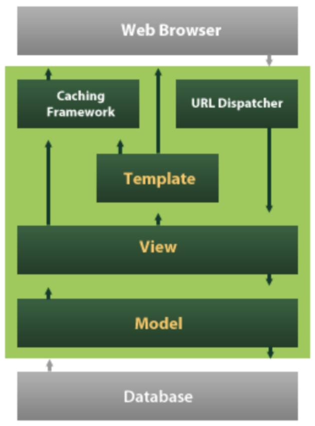
```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92     22319
           1       0.94      0.49      0.65      7415

    accuracy                           0.87     29734
   macro avg       0.90      0.74      0.78     29734
weighted avg       0.88      0.87      0.85     29734
```

These parameters will be guidance thresholds for our implementation in the next phase, however they are not final. We expect the loss function modification to affect the optimality of these parameters, so another cross validation step has to be done in the next phase.

Utility Application architecture:

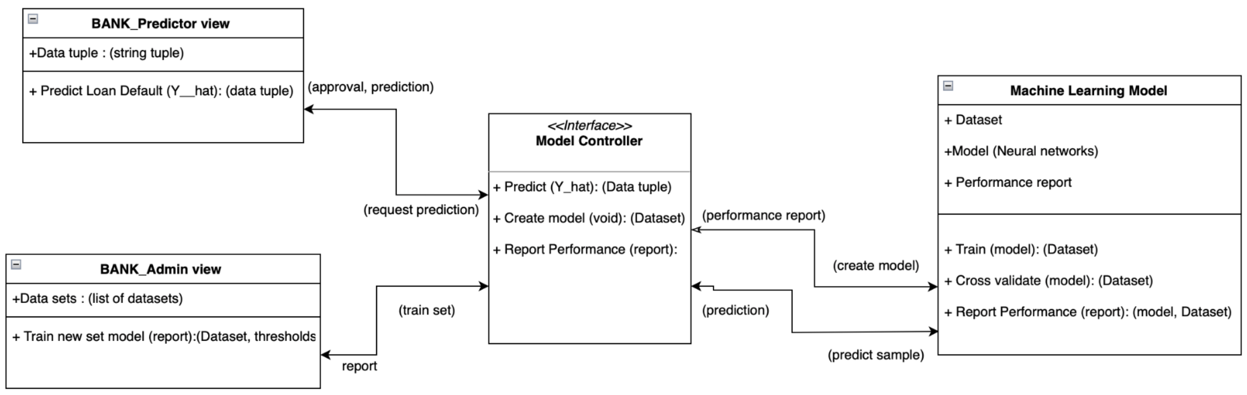
For the application, we decided to implement it as a web application that is deployed on a server and has an MVC architecture (Model View Controller). The reason behind using MVC is the advantage of layering architectures, which guarantees separation of concerns of each layer. The model layer is responsible for the backend and the machine learning model. We can create multiple web application views for Bank clients and admins. And we have a controller layer that acts as an interface between the other layers. We chose the django framework because it is based on an MVC architecture, and satisfies these aspects.

Django architecture diagram:

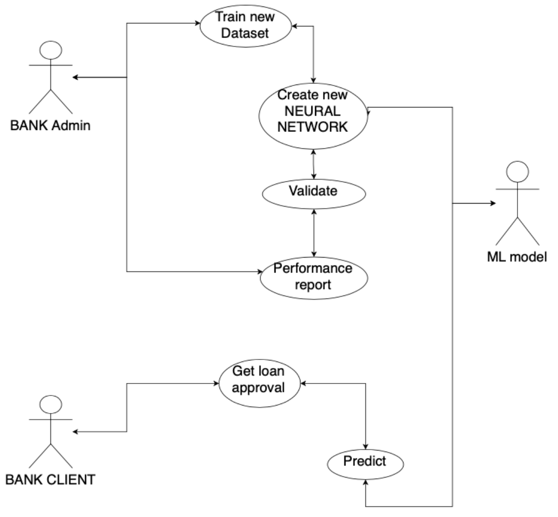


Use Case diagram:

MVC Architecture


BANK_Predictor view
+Data tuple : (string tuple)
+ Predict Loan Default (Y__hat): (data tuple)
(approval, prediction)
(request prediction)
<<Interface>>
Model Controller
+ Predict (Y_hat): (Data tuple)
+ Create model (void): (Dataset)
+ Report Performance (report):
(performance report)
(create model)
(prediction)
(predict sample)
Machine Learning Model
+ Dataset
+Model (Neural networks)
+ Performance report
+ Train (model): (Dataset)
+ Cross validate (model): (Dataset)
+ Report Performance (report): (model, Dataset)
BANK_Admin view
+Data sets : (list of datasets)
+ Train new set model (report):(Dataset, thresholds
(train set)
report

Use Case Diagram



References:

- Bhattacharyya, S. (2021, February 22). *A loss function suitable for class imbalanced data: "Focal loss"*. Medium. Retrieved April 8, 2022, from https://towardsdatascience.com/a-loss-function-suitable-for-class-imbalanced-data-focal-loss-af1702d75d75
- JohnJohn 5311 silver badge44 bronze badges, oW_♦oW_ 5, Dhruv MahajanDhruv Mahajan 34811 silver badge1111 bronze badges, & PaulVDPaulVD 3111 bronze badge. (1965, April 1). *Is Gini coefficient a good metric for measuring predictive model performance on highly imbalanced data*. Data Science Stack Exchange. Retrieved April 8, 2022, from https://datascience.stackexchange.com/questions/19755/is-gini-coefficient-a-good-metric-for-measuring-predictive-model-performance-on
- Brownlee, J. (2020, August 20). *Cost-sensitive decision trees for imbalanced classification*. Machine Learning Mastery. Retrieved April 8, 2022, from https://machinelearningmastery.com/cost-sensitive-decision-trees-for-imbalanced-classification/
- Brownlee, J. (2020, August 20). *How to develop a cost-sensitive neural network for imbalanced classification*. Machine Learning Mastery. Retrieved April 8, 2022, from https://machinelearningmastery.com/cost-sensitive-neural-network-for-imbalanced-classification/
- Rolin, P., Paul RolinPaul Rolin , & Marcin MożejkoMarcin Możejko . (1964, May 1). *Neural network - working with a imbalanced dataset*. Stack Overflow. Retrieved April 8, 2022, from https://stackoverflow.com/questions/38664487/neural-network-working-with-a-imbalanced-dataset
- *Architecture¶*. Architecture - MyTARDIS 1.99 documentation. (n.d.). Retrieved April 8, 2022, from https://pythonhosted.org/MyTARDIS/architecture.html