Boğaziçi University

IE 582

Statistical Learning for Data Mining

Project Report

“The Miners”

Burak Tabak - 2019702177

Enes Ata Oruç - 2020702120

Kaan Bilgin - 2019706009

Instructor: Asst. Prof. Mustafa Gökçe Baydoğan

Submitted: 02.15.2021

Fall 2020-2021

Table of Contents

1. Introduction

1.1. Problem Description

The aim of this project is building a classification model by using the given train and test sets. Train and Test sets consist of 61 columns, including 60 features and 1 target column. These sets did not have labels. In order to measure the performance of the model established for classification, Balanced Error Rate (BER) and Area Under the Curve (AUC) were used as performance measures and the model was expected to predict the target values in the Test data.

1.2. Exploratory Data Analysis

Before applying the classification approaches, Exploratory Data Analytics applications were included in order to have more information about data. While conducting these analyzes, the SmartEDA package in R was used. In order to conduct initial exploratory analysis on any dataset describing the context and the relationships present in the data, SmartEDA contains several custom functions. In both summary and graphical type, the created output can be obtained. Variables are automatically categorized according to the appropriate data types.

1.2.1. Overview of the Data

Understanding the dimensions of the dataset, variable names, overall missing summary and data types of each variable is important. Information about the properties of the variables in Train data can be seen in Table 1.

Table 1: Overview of the Data

Overview of the data

Descriptions <fctr>	Value <fctr>
Sample size (nrow)	2074
No. of variables (ncol)	61
No. of numeric/interger variables	60
No. of factor variables	1
No. of text variables	0
No. of logical variables	0
No. of identifier variables	0
No. of date variables	0
No. of zero variance variables (uniform)	2
%. of variables having complete cases	100% (61)

As it can be seen in Table 1, while the 60 feature is in numeric / integer structure, one of the features is held as a factor. This factor column is the variable where Target values are kept under the name "y" in the data.

After looking at the variable types of the data, the structure of the data was also examined. In other words, how many different values are found in the relevant feature. For example, if the number of distinct values is 2, it means there are only 2 different values in that column, (like binary 1-0 values). Structure of the data for the first 10 features can be seen in Table 2.

Table 2: Structure of the Data

Structure of the data

Index <dbl>	Variable_Name <chr>	Variable_Type <chr>	Per_of_Missing <dbl>	No_of_distinct_values <int>
1	x1	integer	0	36
2	x2	integer	0	2
3	x3	integer	0	2
4	x4	integer	0	2
5	x5	integer	0	19
6	x6	integer	0	19
7	x7	integer	0	19
8	x8	integer	0	37
9	x9	numeric	0	1302
10	x10	numeric	0	1302

1-10 of 61 rows Previous 1 2 3 4 5 6 7 Next

In the y column, which is the target feature, there are categorical a and b values in factor type. The distribution percentage of a and b values can be seen in Figure 1.

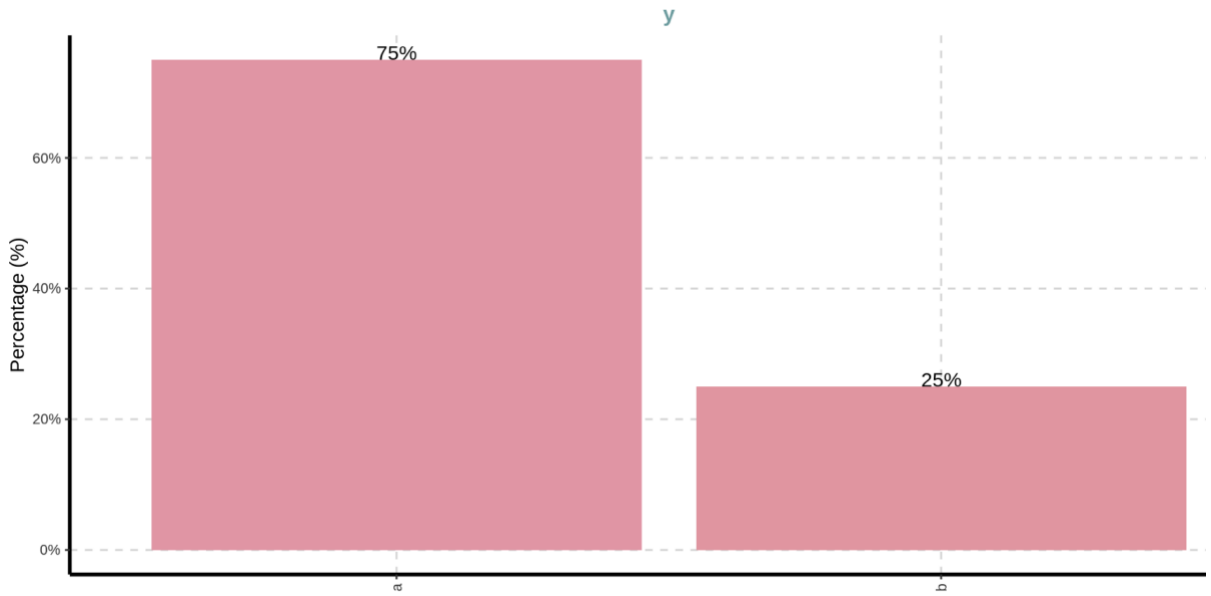

Figure 1: Distribution of Categorical Variables

1.3. Summary of Approaches

After the overview of the data step, the classification task has started. At this step, first of all, appropriate methods and approaches to be followed were decided. The applied methods are Penalized Logistic Regression, Bagging Approaches with Random Forest and Boosting Approaches with Extreme Stochastic Gradient Boosting.

The application of these methods was first started with data preprocessing. Unnecessary features with 0 or low-variance were cleaned here. Standardization has been applied to variables other than the target variable. Out of bag predictions from the Random Forest method applied to whole data, this was the feature engineering step. As the last step of data preprocessing, improvements were made to Oversampling for Class Imbalance. After the modeling step of the selected approaches, the post-processing and selection of models step is to decide which one will be applied as the final model.

2. Literature Review

Mulay and Joshi (2015) define data mining as “Analytic process to explore data in search of consistent patterns and systematic relations between variables” [1]. Their study focuses on predictive data mining. The main problem being faced in the whole world is analyzing and working with large data which causes some challenges and waste of time. This simply explains the need for better data mining techniques. Data mining includes in itself both statistical and machine learning techniques to determine patterns, behaviors in large data sources. For classification it includes several options such as Artificial Neural Networks (ANN), decision tree analysis, rune induction, K-nearest neighboring techniques.

With its classification and regression properties, trees are used in real life in a broad area of machine learning. While doing analyses, it provides a clear, visual and explicit understanding [2]. In data mining decision trees are used very often to build some classification system. Trees make their classification based on multiple variables and also offer the chance to develop some prediction algorithms [3].

By sequentially fitting a simple parameterized function (the basic learner) to the least squares and current "pseudo" residues on each iteration, gradient boosting generates additive regression models. In each training data point evaluated in the current stage, the pseudo residuals are the gradient of the loss function minimized according to the model values [4].

It has been shown that it is possible to significantly increase both the estimated accuracy of gradient enhancement and the speed of administration by using randomization in the process [4]. Boosting was shown to increase the accuracy and decrease the variance of the base classifier when applied to low-dimensional data [5].

3. Approach

After doing the necessary literature research about the classification problem and analyzing data descriptively for given task, the following steps are followed as the final method in this report. The details about each step will be given separately with their motivations.

Data Preprocessing:

Feature Selection

- Find near zero variance features (Features with low variance value.)
- Find feature importance from Random Forest model applied to full data.
- Remove intersection of zero variance and unimportant 30 feature from feature importance.

Standardization of the features

- (X - mean) / sd standardization is applied for each feature excluding target.

Feature Engineering

- Out-of-bag predictions of Random Forest model applied to whole data are used as new features.

Oversampling for Class Imbalance

- SMOTE method is used for balancing class sizes.

Modelling:

Methods

- Penalized Logistic Regression approaches using GLMNET
- Bagging approaches using Random Forest
- Boosting approaches using extreme stochastic gradient boosting (Xgboost)

Parameter Selection

- 10-fold Cross Validation is applied for 3/4 of data to choose best parameters of tuning.
- AUC is used in tuning.

Model Evaluation

- 1/4 of the data is separated as test data.
- Performances are compared on this hold-out data.
- AUC, Balanced Error Rate and Accuracy are reported.

Post-processing and Selection of Models

- Average of best performing models on test data used as final prediction.
- **Boosting is excluded.** LASSO and Random Forest models are used as final models.

3.1 Data Preprocessing

There are 61 features with 2074 observations in the given classification task including target feature. Beside from the target, all other features are numeric. In order to avoid the curse of dimensionality, high-dimensional feature sets with small observations may prevent finding meaningful relationships between features, feature selection is applied on data. The main aim is to reduce feature dimension by excluding unimportant features and features with low variances.

In order to find features with low variances, “nearZeroVar” function of caret package is used with default settings. For feature importance, “ranger” function is used from the ranger package. Ranger is the efficient implementation of random forest method. As a feature importance measure, Gini impurity is used. Because there are some features (x42 is an example) with low variance but high feature importance, the intersection of low variance and 30 least significant feature set are removed. The importance of features is given below in Figure 2.

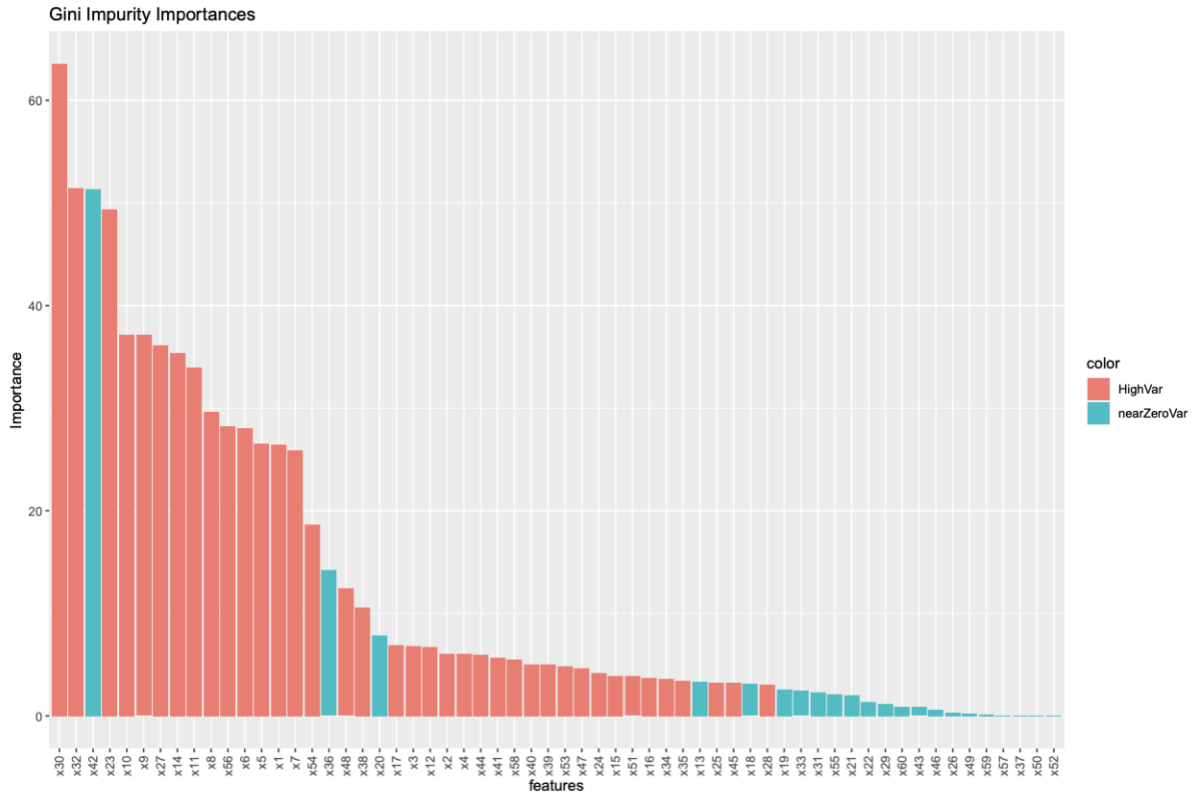

Figure 2: Gini Impurity Importance

After removing insignificant features, 42 features remained including target. At this step, standardization is applied because there are some features (x36, x42) with mostly 0 value but for less observation, they take huge values.

They have highly left skewed distributions. In order to prevent possible issues because of large ranging values in the modelling phase, standardization is applied by using the “scale” function. The histogram of x36 and x42 is given below in Figure 3.

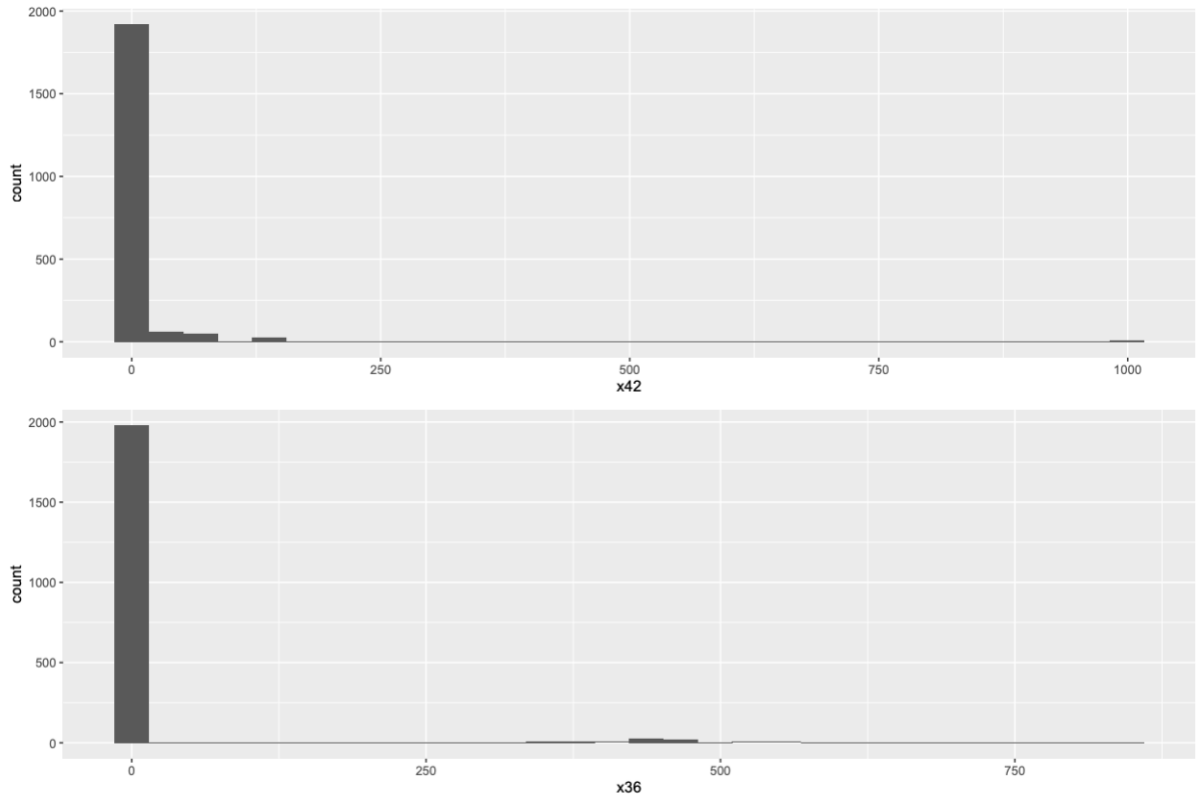

Figure 3: Sample Features Before Standardization

After standardization, out of predictions of Random Forest integrated into the feature set. In the bagging approach, at each step, 2/3 of the data used as training and 1/3 of them used as out of bag. So, out of bag observations are not included in the training set of the corresponding tree. Because of this property, out of bag predictions can be used as features because there are less likely to over fit.

In this part, two features are integrated into the feature set. The first feature is the out of bag predictions of regression trees of “ranger” method with minimum node size equals to 5. The second feature is the out of bag predictions of classification trees of “ranger” with minimum node size equals to 3. For classification trees, it is meaningful to use smaller minimum node size to allow further splits. Lastly, in order to avoid class imbalance problem; SMOTE approach is used for oversampling. In given task, a/b ratio ~ 3/1 and this situation may create problems in the modeling phase. Also, a balanced error rate metric uses 0.5 as threshold so it is necessary to calibrate probabilities if unbalanced data is used in training. Balanced error rate with 0.5 threshold assumes balanced class problem.

AUC shows the ordering capability of the model but balanced error rate needs strict assignment according to threshold. So, to prevent possible class imbalance problems, SMOTE is applied by using the “SMOTE” function from the DMwR package. SMOTE is an oversampling method which creates artificial observations from minority classes using the k-nearest neighborhood algorithm. The class sizes of target before and after SMOTE are shown below in Figure 4.

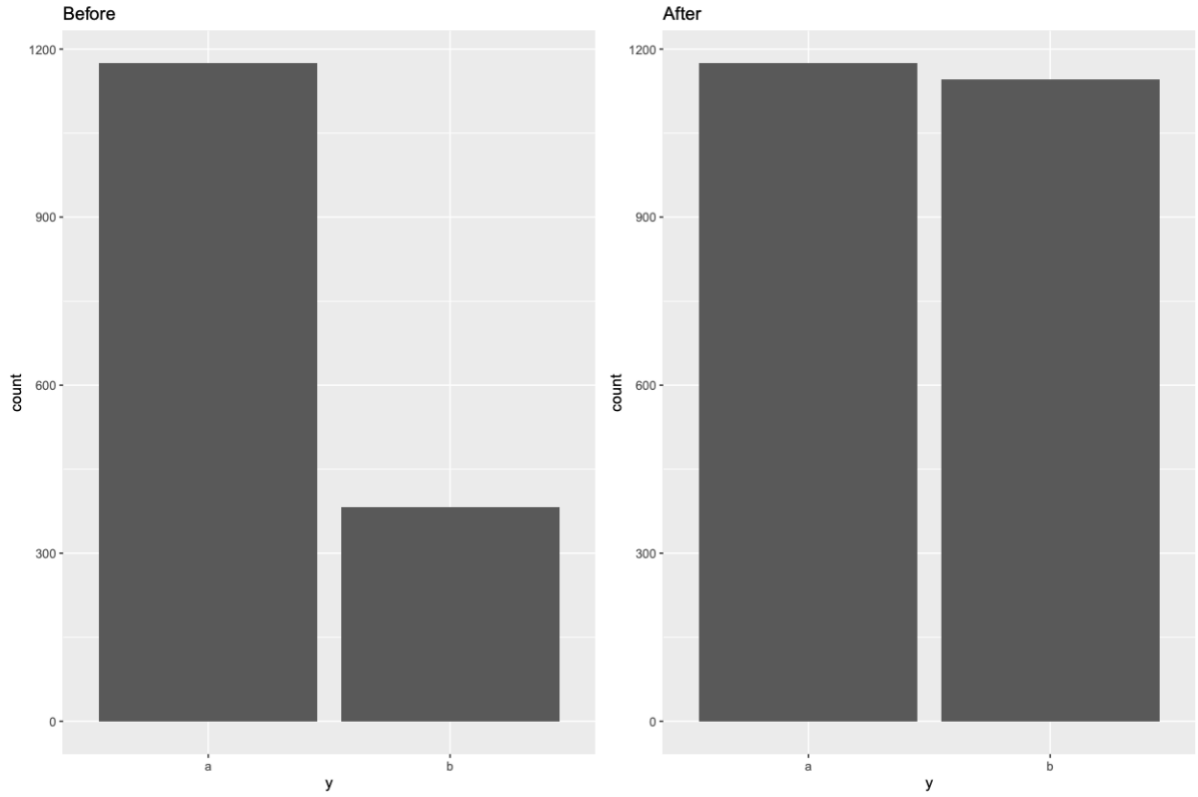

Figure 4: Class Sizes of Target

3.2. Modelling

After data preprocessing operations, the following machine learning methods are applied to data for classification task:

1) **Linear Approaches**
   - Penalized Logistic Regression using L1 penalty (LASSO)

2) **Nonlinear Approaches**
   - Bagging with Random Forest
   - Boosting with extreme gradient boosting (Xgboost)

The main of the selection of methods is to compare both linear and non-linear methods. Also, both bagging and boosting approaches are analyzed in order to study problems from different perspectives.

Before parameter tuning, 1/4 of the data split as hold-out test data. In order to keep distribution balance of target in train and test data, test split is done via "stratified" function over target. 3/4 of the data is used for parameter tuning. For parameter tuning, the best parameters are selected according to results of stratified 10-fold Cross Validation with single repetition (Figure 5.). As a comparison metric in cross validation, Area Under Curve (AUC) is used for classification tasks; AUC is generally built-in performance metrics in model functions. Integrating custom performance metric causes performance deficiency in implementation. Also, introducing two objective functions at the same time causes optimization convergence problems. Because balanced error rate and AUC are correlated, AUC is used for cross validation metric on behalf of two metrics.

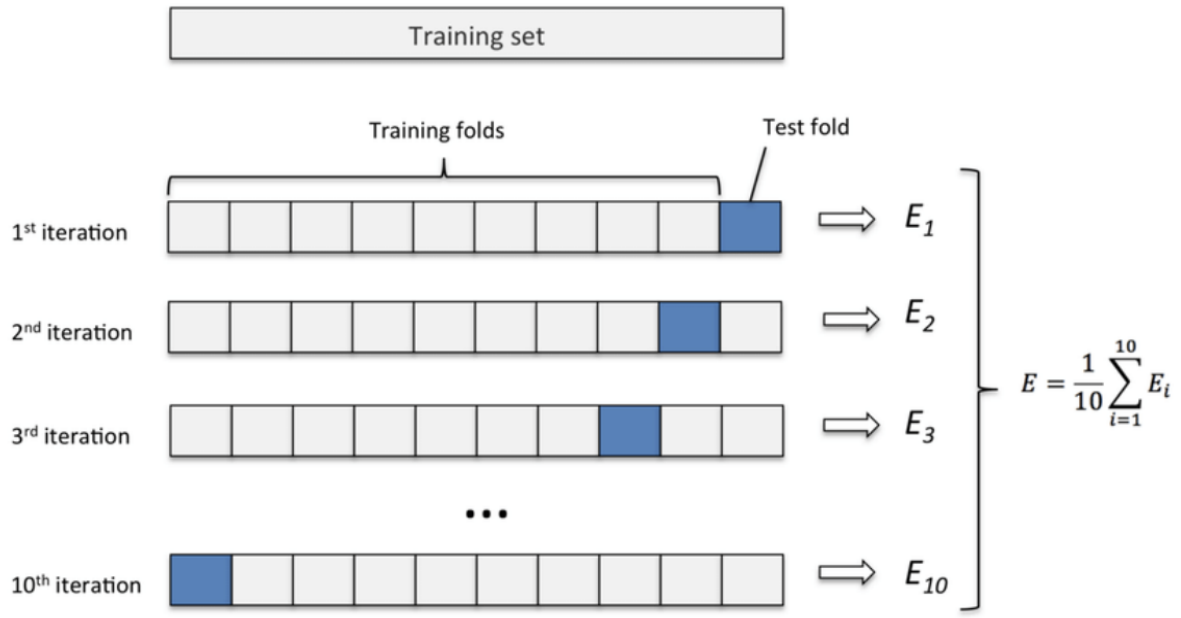

Figure 5: Cross Validation on Train Set

For Penalized Logistic Regression (Figure 6.) using L1 penalty, “cv.glmnet” function from glmnet package is used. The tuning parameter is lambda, which is the penalization coefficient.

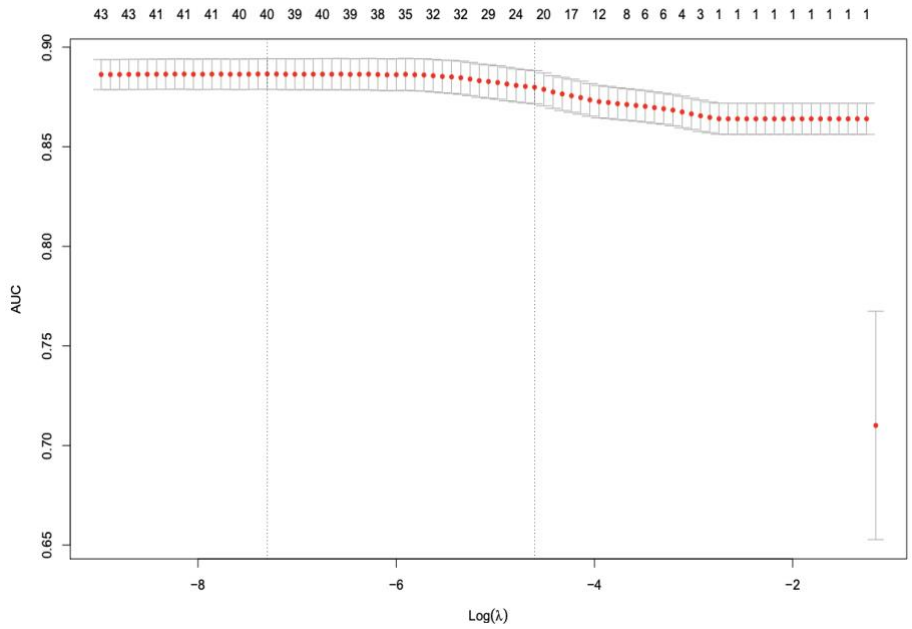

Figure 6: Lasso Cross Validation Results

The best parameter lambda.min = 0.002094 with 0.8864 AUC according to cross validation result. For random forest, caret package is used for parameter tuning. The tuning parameter is mtry, which is the number of variables considered in each tree. (2,4,6,8,10) set is used for tuning the grid. The number of trees 5 generated at each step is 1000 with minimum node size equals to 5.

The mtry=6 is chosen with 0.9635020 AUC (Figure 7) according to cross validation result. For stochastic gradient boosting, caret package is used for parameter tuning winf “xgbTree”. The tuning parameters are learning rate (eta), maximum depth of trees (max_depth) and number of iterations (nrounds). Other parameters are kept at default levels for example gamma, colsample_bytree, min_child_weight, subsample as 0,1,1,1 respectively.

The final values used for the model were nrounds = 1000, max_depth = 6, eta = 0.05, gamma = 0, colsample_bytree = 1, min_child_weight = 1 and subsample = 1 according to cross validation results. (Figure 8.)

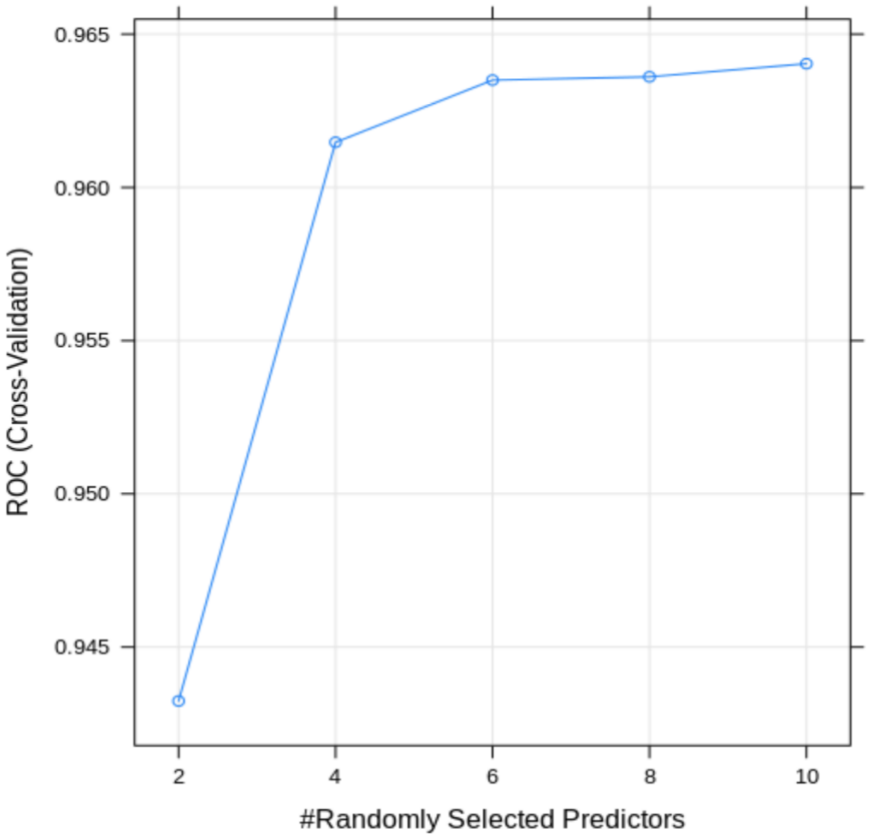

Figure 7: Random Forest Cross Validation Results

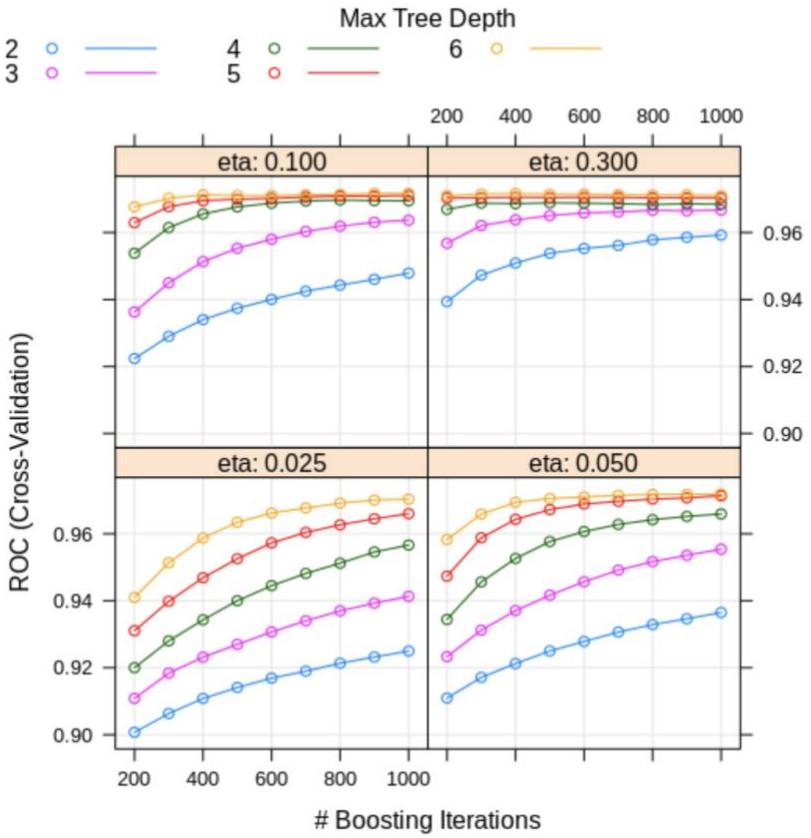

Figure 8: Boosting Cross Validation Results

3.3. Post-processing and Selection of Models

After cross validation, the corresponding parameters are selected. In order to decide final selection; the performances on hold-out test data will be analyzed. Because there are artificial observations in train set due to SMOTE, test performance gives more consistent and close results to actual test data with unknown target. The final method for submission will be selected according to both AUC and Balanced Error Rate metrics on hold-out test data (Table 3.)

Table 3: Model Performances

model	AUC	Accuracy	Balanced Accuracy
Xgboost	85.80%	78.57%	71.45%
LASSO	88.55%	77.61%	78.79%
Random Forest	87.20%	80.89%	75.91%

It can be seen that boosting performs the worst according to both metrics however for Balanced Accuracy, it seems worse than AUC performance. In this point, we will analyze the probabilities given by three models.

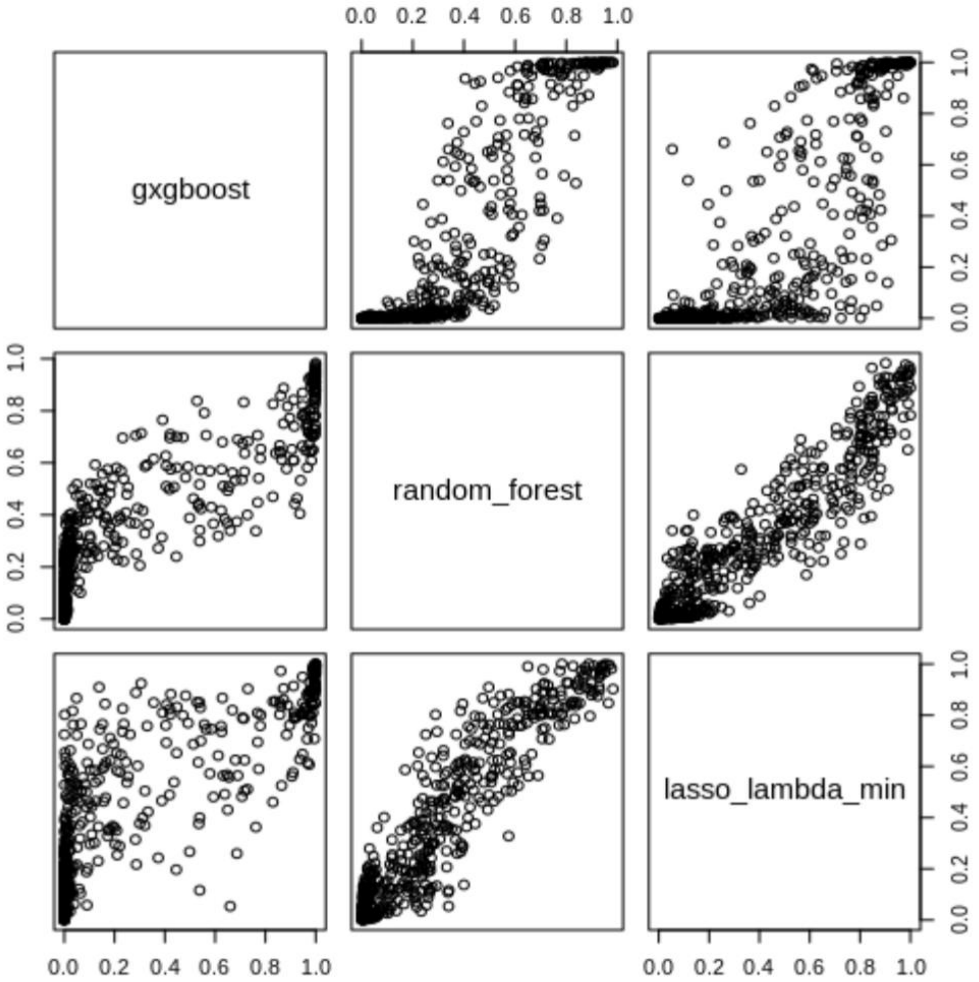

Figure 9: Probabilities of Models

It can be observed that boosting gives probabilities closer to 0 and 1 compared to random forest and LASSO. So, because balanced error rate is calculated via 0.5 threshold, probability results of boosting don't represent true probabilities. Boosting performs ordering but doesn't give correct probability estimates. There is a calibration needed. However, for LASSO and Random Forest, probability results seem highly linearly correlated (Figure 9.)

It is known that performance of method can be measured by Biasˆ2 + Variance. So, in case of unbiased estimators; variance can be reduced by increasing estimator number because variance = sigmaˆ2/n. So, ensembles of the models may decrease variance and perform better compared to single models. Below, in Table 4, you can find ensemble results on test data.

Table 4: Ensemble results

model	AUC	Accuracy	Balanced Accuracy
Xgboost	85.80%	78.57%	71.45%

model	AUC	Accuracy	Balanced Accuracy
LASSO	88.55%	77.61%	78.79%
Random Forest	87.20%	80.89%	75.91%
Lasso-RandomForest	88.38%	81.47%	80.28%
Lasso-RF-XGB	87.71%	81.27%	76.70%

As expected, the ensemble of LASSO and Random Forest performs best compared to both LASSO and Random Forest. Introducing boosting into ensemble decreases performance due to probability calibration reasons.

In the light of these results, the average of Lasso and Random Forest probabilities are submitted as the final model.

4. Results

From the previous part, the final performances on hold-out test data is below (Table 5.)

Table 5: Performances on hold-out test data

model	AUC	Accuracy	Balanced Accuracy
Xgboost	85.80%	78.57%	71.45%
LASSO	88.55%	77.61%	78.79%
Random Forest	87.20%	80.89%	75.91%
Lasso-RandomForest	88.38%	81.47%	80.28%
Lasso-RF-XGB	87.71%	81.27%	76.70%

As the final model, Lasso-Random Forest average is chosen for final submission. The same data preprocessing is applied to whole data and models with best parameters according to cross validation results are built on the whole dataset. The different ensembles and single models are also submitted throughout the competition but the best result of initial evaluation phase is achieved by our final approach as follows (Group: The Miners):

Table 6: Ranking of the Project

Username	Best score
R'siz Rakunlar	0.9001
Data Miners	0.8972
ChE-origin	0.8949
Miners	0.8936
The Miners	0.8907
HNY	0.8891
Runtime Terror	0.8855
Datatata	0.8832
Los Galacticos	0.8775
TOYGAR - MERT	0.8736
Frankenstein	0.8649
Error_Loading	0.8630
papatya grubu	0.8599
Oversitting	0.8591
Cagan-Hakan	0.7764
admin-m	0.0000
admin	0.0000
admin-t	0.0000

5. Conclusion and Future Work

For the unknown test data:

- Feature selection and standardization done for the dataset
- Oversampling done for imbalanced class
- Various models of various methods modelled and analyzed
- AUC performance ~ 92% and Balanced Accuracy ~ 86% for our final method.
- AUC 9 performance seems good but Balanced Error Rate is the bottleneck for our approach. So, an increased number of unbiased estimators with good probability representations may improve Balanced Accuracy performance. Also, some probability calibration methods such as Platt Scaling, Isotonic regression may increase Balanced Accuracy metric.

Main issues addressed in this project are:

1) Preprocessing and selecting the right features
2) Class imbalance

Although we proposed proper solutions for overcoming these problems, we have one more solution in general for the overall project, Deep Neural Networks [6].

Deep learning has a long history and many aspirations. Some of the proposed approaches have not yet fully borne fruit. Some ambitious goals remain to be achieved. Modern deep learning provides a powerful framework for supervised, unsupervised, reinforced and transfer learning. Main objective is to generate a general representation of the data. There were Artificial Neural Networks(ANN) before DNN whose working principle was to take the input, multiply them with weights and generate an output. This type of networks called feed forward algorithms and they usually work for linear problems. Looking out from here one can directly think that can it represent a non-linear problem and the answer is no, it can't represent non-linear problems. To be able to generate a representation for non-linear problems, scientist examined nature and came up with an interesting idea, brain. Neural Networks have their name thanks to their similarity to brain functions of living, which was on purpose. Every deep learning architecture, regardless of their depth (depth = number of layer) or width (width = number of neurons in each layer), is built with three things, with this hierarchical order (Figure 10):

1) Input layer(s)
2) Hidden layer(s)
3) Output layer(s)

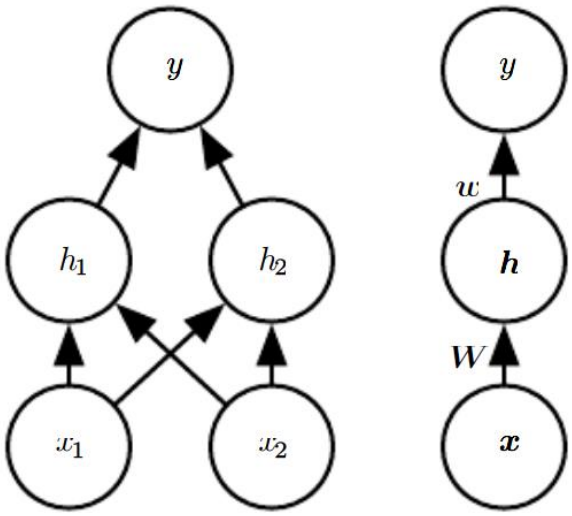

Figure 10: A simple deep network with an input layer, a hidden layer and an output layer [6].

Superiority of the deep neural networks to ANN is that deep neural networks have back propagation (Figure 11.). Back propagation is a way to adjust weights and biases of the layers to learn and predict an output value close to the given (or not given, depends on the problem e.g. unsupervised learning) output. This adjustment is calculated with the simple chain rule of derivation. Adjustment is done based on cost, which is a measure of error. For C is the cost function, A is the activation function and Z is the weighted input, the cost for a network with a single neuron is:

$$Cost=C(A(Z(input\times Wi)))$$

And from this cost, Information gained for whether to change or not to change the weights and biases for the next iteration (if needed). If it's necessary to change the weights and biases, it requires the derivate of the cost with respect to any weight. This derivation done for gaining information about how much each weight contributes to the total error and adjusting direction to reduce the total error.

One can observe a problem, a paternal problem in terms of computational engineering. For the exact terms in different steps are calculating again and again for each step, by that it decreases computational efficiency and increases the computation time by nearly to exponential time. To solve the issue there is a simple solution method in this particular area: " Memoization "

Memoization is the term produced from Dynamic Programing branch which roughly can be explained as:

- We have more powerful computer in both terms of memory and processing
- To do more efficient computing, a repeated expression can be stored in an array (which located in the memory) and can be used after, any time the propagation calls that specific stored expression.
- By so, it is a trade between higher computing speed and computer memory, which is a kind of trade that makes the exponential computing time (for deeper and wider networks) decreases significantly.

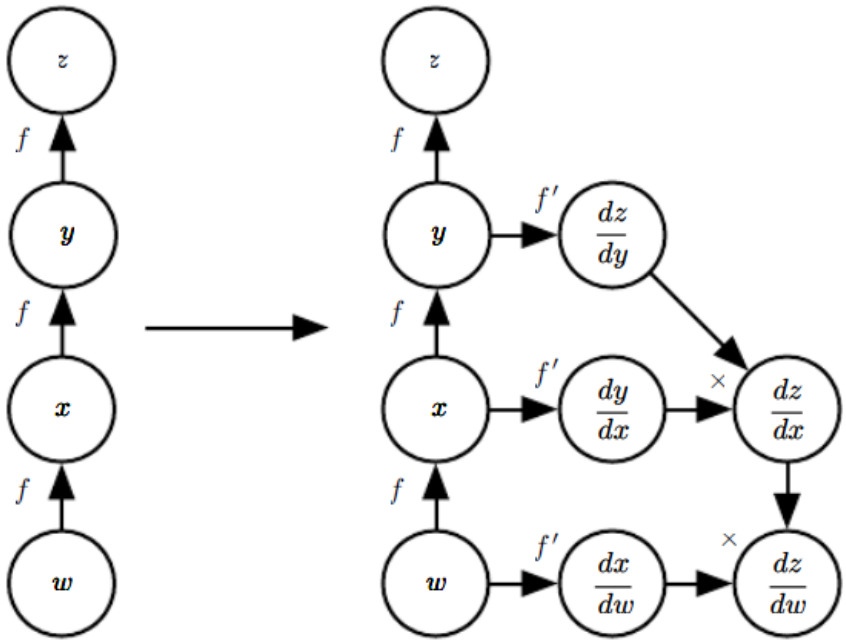

Figure 11: Back propagation future of DNN [6]

By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most of the tasks of mapping an input vector to an output vector that are quick for a person to accomplish can be done by deep learning when there are sufficiently large models and sufficiently large data sets with training examples labeled. So again we are facing an old enemy, not sufficient data and imbalanced classes (Deep learning methods have a bad reputation to over fit rapidly for not sufficient data and imbalanced classes)

5.1. Proposed Dense Neural Network Model and Steps

1) Same dataset used for the training and testing of the ML methods
2) Output variables (‘a’ and ‘b’) encoded as integers with LabelEncoder class from sklearn.preprocessing package
3) Data normalized with StandardScaler function from sklearn.preprocessing package
4) Model will be a very simple dense neural network and will be tested for both raw and processed data.
5) 1. Model will not include any regularization methods like dropout or weight decay etc.
6) This model will be a dense network with linearly rectified unit(relu) activation

Function at hidden layers and sigmoid activation function at the output layer.

7) Binary classfication exist with predicting the probability of a label as true (1) or not (0) as probabilities lies between [0,1] and the logistic function of sigmoid is the superior choice here since it values between (0,1).

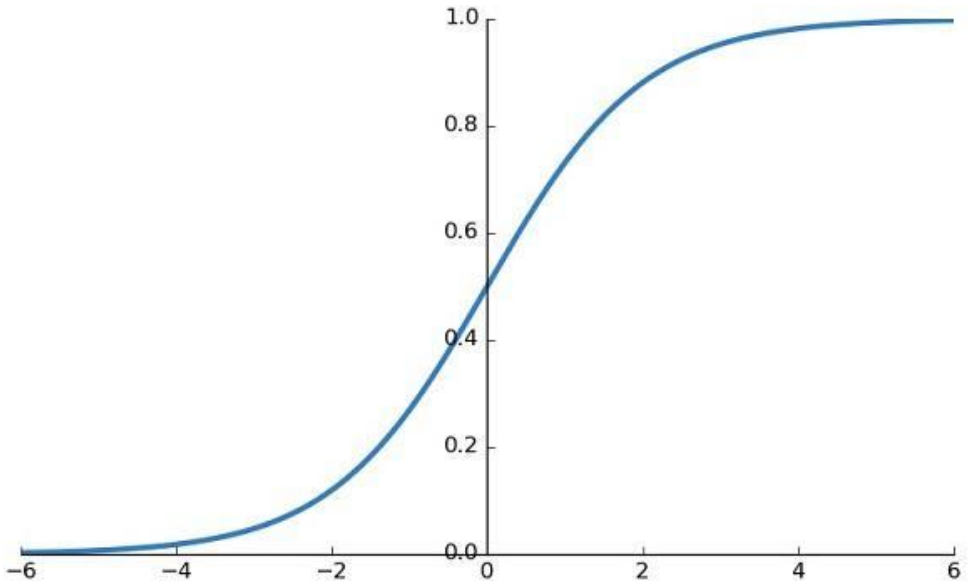

Figure 12: Sigmoid Function [6]

8) Loss will be calculated true binary_crossentropy which needs probalistic results as input, hence makes the sigmoid choice also relatable.

9) Optimizer will be stochastic gradient descent (SGD)

10) Metrics will be accuracy (it also can be binary accuracy)

11) Early Stopping will be applied to prevent over fit

12) Detailed architecture can be found in our google collaboratory file

13) Model 2 network will be so much shallow than the first model but with l2 norm penalty and dropout features.

5.2. Results

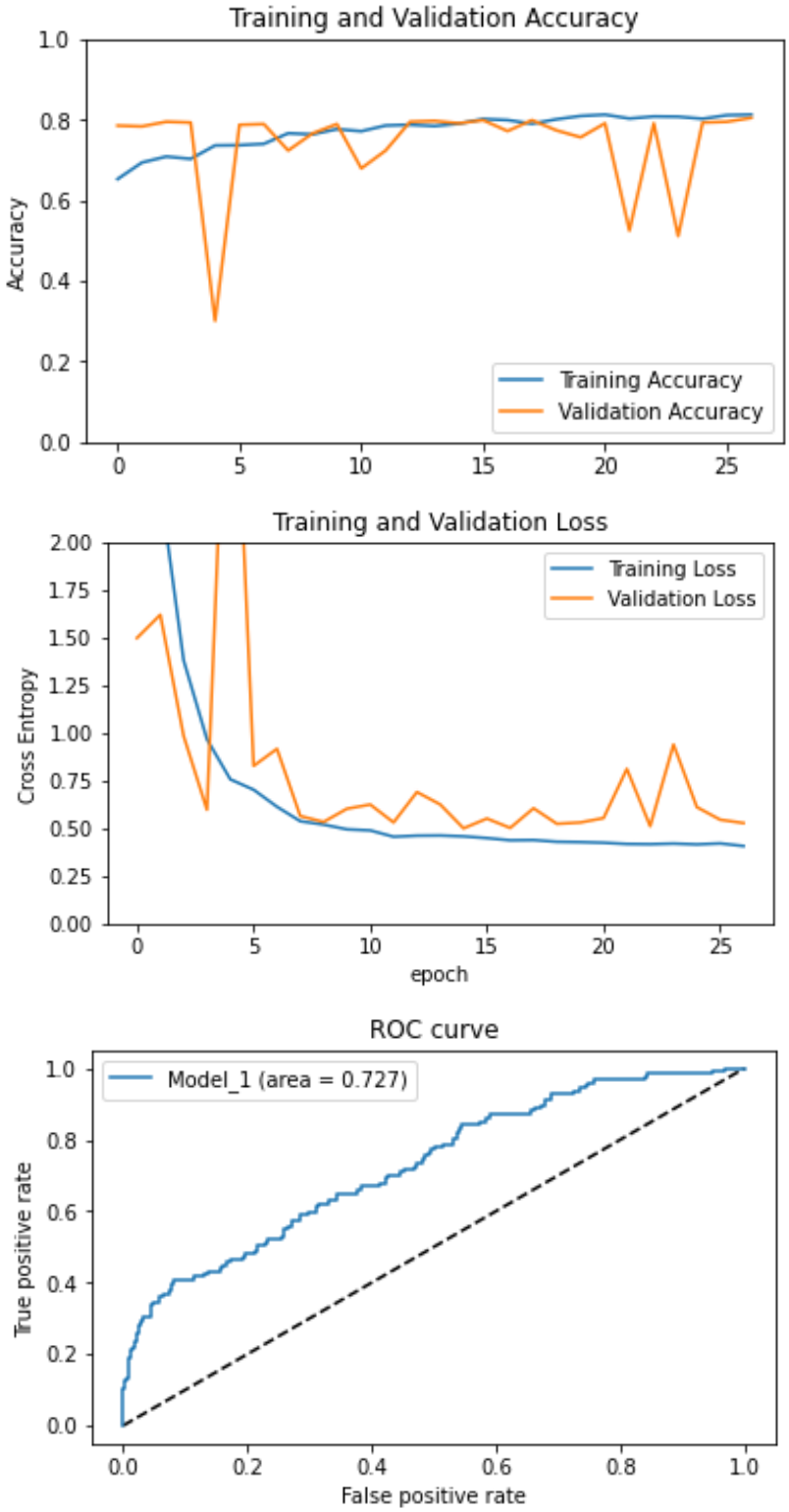

Figure 13: Model 1 results with raw, non-processed data

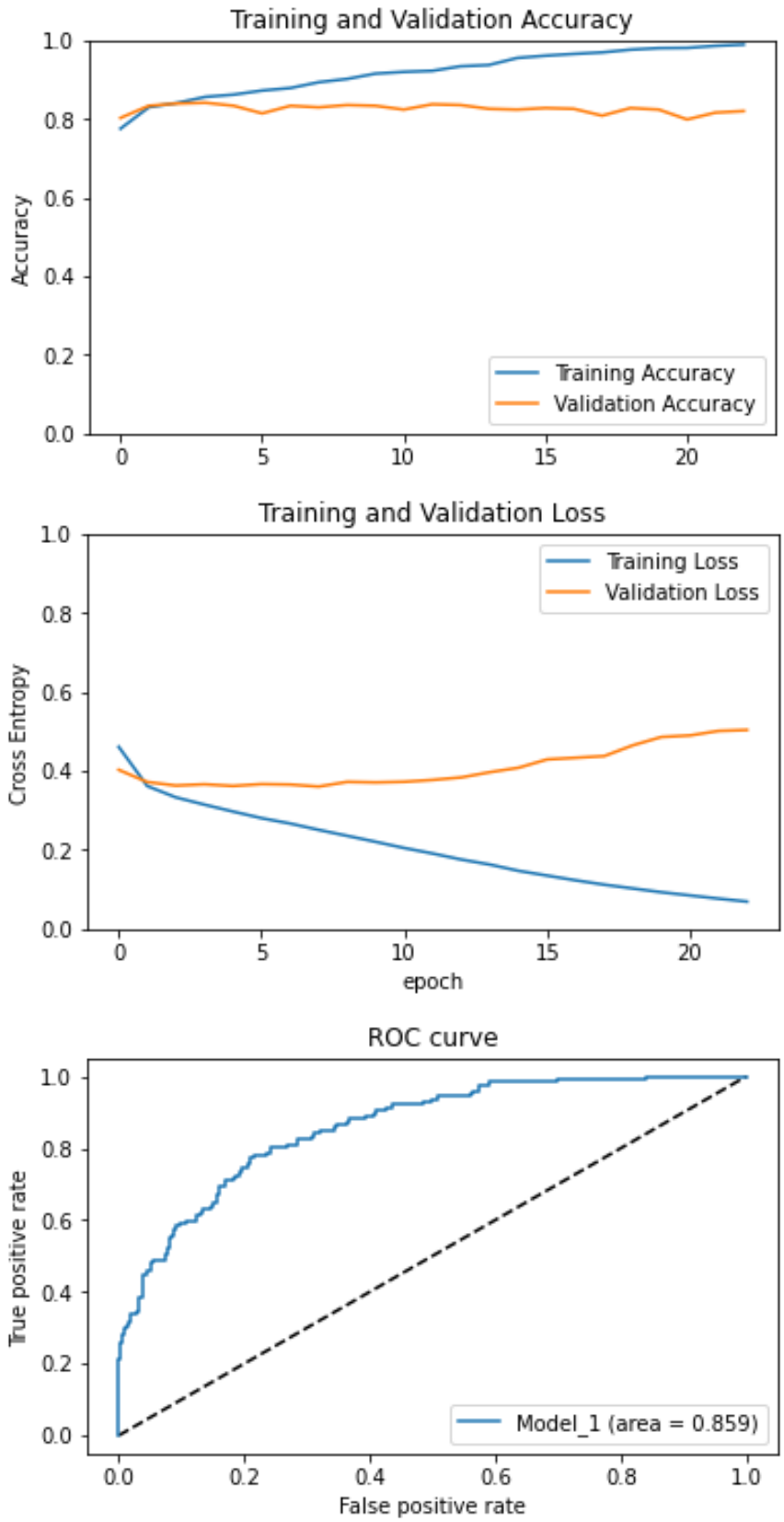

Figure 14: Model 1 results with normalized data

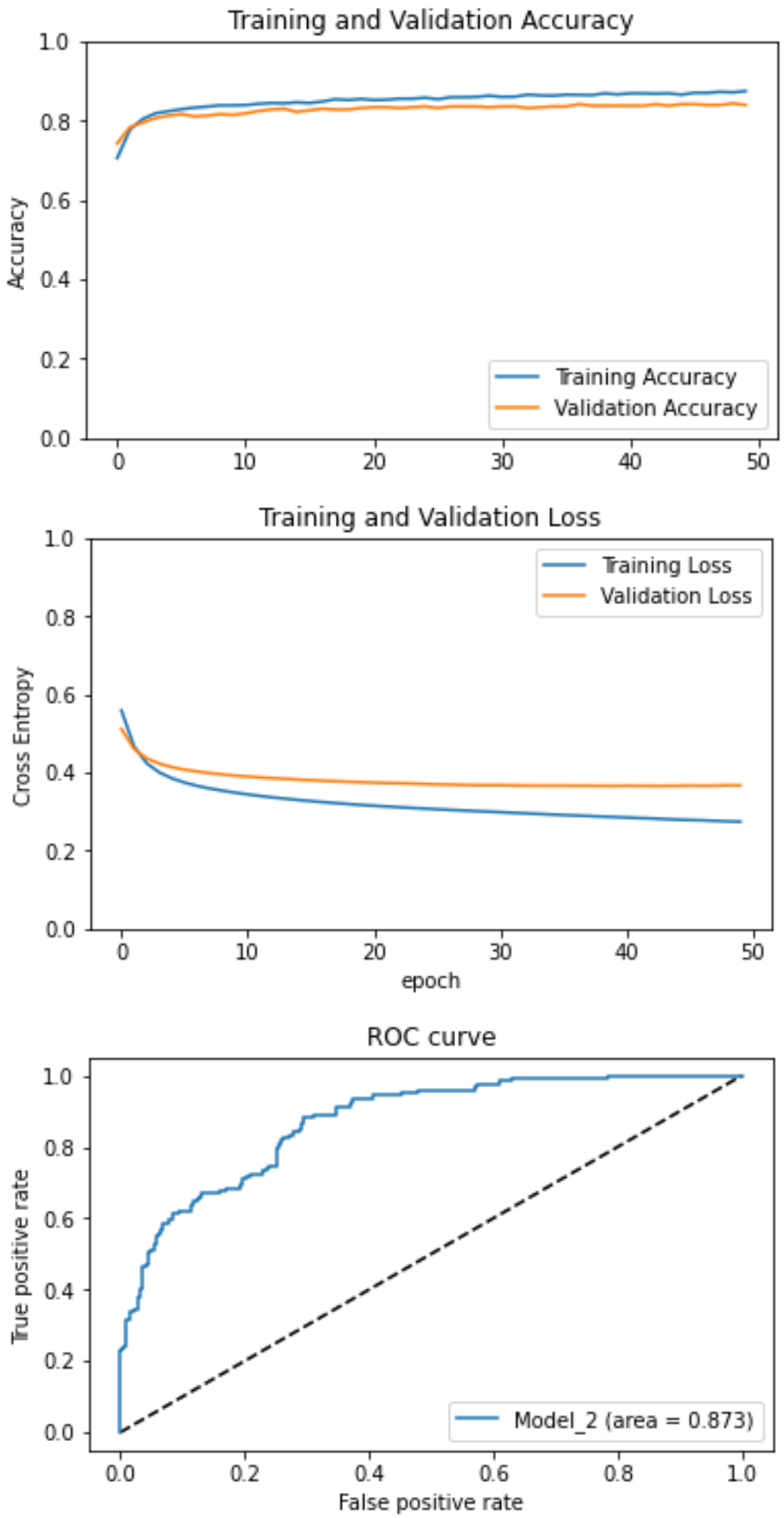

Figure 15: Model 2 results with normalized data and regularization

As can be seen from roc curve values in figure 13 and 14, preprocessing still does an immense job at generalization of the representation in this problem.

- Rapid overfitting can be observed for each result.
- More shallow but l2 penalized model 2 showed the best accuracy between all three.

There are still things to address:

- On the contrary of ML methods, in deep learning, networks do not show which features are important
- Dealing with class imbalance can increase accuracy since it prevents to over fit through the dominant class
- Parameter tuning (with regarding the regularization methods like we do) is referred to art more than science in deep learning because of that there is no 'guide' to help you get maximize output. Yes, there are regularization methods and different architectures like CNN and RNN for various kind of problem but in the very core parameter tuning is a posteriori and changes for each new type of problem. But the most important thing like CNN and dense neural networks are that **we do not need to choose which feature will be in our model, model is deciding this for itself.** This is a huge bonus for data where we don't know where to look. There are preprocessing techniques and feature extraction tools such as PCA, but not thinking about these for **"big data"** is a win-win situation.

6. References

[1] Muley, P., & Joshi, D. (2015). Application of Data Mining Techniques for Customer Segmentation in Real Time Business Intelligence. International Journal of Innovative Research in Advanced Engineering (IJIRAE),106-109. Retrieved from http://www.ijirae.com/volumes/Vol2/iss4/19.APAE10122.pdf

[2] Gupta, P., & Gupta, P. (2017, May 17). Decision Trees in Machine Learning. Retrieved from https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

[3] Song, Y. Y., & Lu, Y. (2015). Decision tree methods: applications for classification and prediction. Shanghai archives of psychiatry, 27(2), 130–135. doi:10.11919/j.issn.1002-0829.215044

[4] Jerome H. Friedman, Stochastic gradient boosting, Computational Statistics & Data Analysis, Volume 38, Issue 4, 2002, Pages 367-378, ISSN 0167-9473, https://doi.org/10.1016/S0167-9473(01)00065-2.
(https://www.sciencedirect.com/science/article/pii/S0167947301000652)

[5] Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., Herrera, F., 2012. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 42, 463–484.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. The MIT Press.

7. GitHub Repositories

Burak Tabak:
https://bu-ie-582.github.io/fall20-Burakt94/codes/Project.R

Enes Ata Oruç:
https://bu-ie-582.github.io/fall20-AtaOruc/Files/IE582_ProjectCodes.R

Kaan Bilgin:
https://bu-ie-582.github.io/fall20-kaanblgn/files/Project/codes_final.R

Future Work Google Collab Link:
https://colab.research.google.com/drive/1_H1EBVa4dZrIucEZIaL3Eb7BEJQzDRmd?usp=sharing