Credit Card Approval

Encheng Liu, Jiaze Cai, Yuanzhuo Li, Zaowei Dai, Chensheng Peng (Group 1300)

1 Introduction and problem description

We investigated a binary classification problem, Credit Card Approval [1]. Commercial banks receive many applications for credit cards. Many of them get rejected for many reasons, like high loan balances, low-income levels, or too many inquiries on an individual's credit report. Manually analyzing these applications is mundane, error-prone, and time-consuming. This project aims to develop a predictive model for determining the approval associated with credit card applications. The following table shows the list of factors we consider in this project (Table 1).

Feature	**Description**
Gender	The applicant's gender.
Debt	The amount of debt the applicant currently has.
Married	The marital status of the applicant.
Bank Customer	Whether the applicant is a customer of the bank.
Education Level	The education level of the applicant.
Ethnicity	The ethnicity of the applicant.
Years Employed	The number of years the applicant has been employed.
Prior Default	Whether the applicant has defaulted on a loan before.
Employed	Whether the applicant is currently employed.
Credit Score	The applicant's credit score.
Drivers License	Whether the applicant has a driver's license.
Citizen	The citizenship status of the applicant.
Zip Code	The applicant's residential zip code.
Income	The applicant's income.
Approval Status	Whether the credit card application was approved.

Table 1: Features used for credit card approval prediction

In Section 2 and 3, we provided the details of the data processing and the methodology, including k-nearest neighbors, naive Bayes, decision trees, support vector machines, boosting, logistic regression, gradient boosting decision tree, random forest, XGBoost, multilayer perceptron, and stochastic gradient descent classifier. We also provided the experimental results with an analysis of the models' performance in Section 4. Accordingly, we choose the classifier with the best performance, stochastic gradient descent classifier, for hyperparameter optimization.

2 Data Description and Preprocessing

2.1 Data Description

We use *pd.read_csv* to load the data into the pandas DataFrame and use *info()* to get the general information of it, as Figure 1 shows.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          678 non-null    object
 1   Age             678 non-null    float64
 2   Debt            690 non-null    float64
 3   Married         684 non-null    object
 4   BankCustomer    684 non-null    object
 5   EducationLevel  681 non-null    object
 6   Ethnicity       681 non-null    object
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    object
 9   Employed        690 non-null    object
 10  CreditScore     690 non-null    int64
 11  DriversLicense  690 non-null    object
 12  Citizen         690 non-null    object
 13  ZipCode         677 non-null    object
 14  Income          690 non-null    int64
 15  ApprovalStatus  690 non-null    object
dtypes: float64(3), int64(2), object(11)
memory usage: 86.4+ KB
```

Figure 1: Data Information

It is obvious that there are totally 690 samples, 15 input features, and 1 output feature called **ApprovalStatus**. According to the DataFrame, we found that the dataset utilized in our experiment consists of features indicating two distinct classes: continuous-variable(Age, Debt, Years Employed, Credit Score and Income) and categorical-variable(other features), along with some features having null entries(Gender, Age, Married, Bank Customer, Education Level, Ethnicity, Zip Code), but the percentages of missing data are small (less than 2%).

2.2 Data Preprocessing

Once the dataset is loaded and inspected, we need to preprocess the data. First, we split the raw data into training sets and test sets. We use 80% of data as the training set and 20% of data as the test set. For the training set and test set, preprocessing should be made separately since we assume they are independent. Moreover, since the features can be classified as numerical and non-numerical features, we decided to separate these two types of data and preprocess them separately.

2.2.1 Continuous-variable Features

Table 2 shows the statistical properties of continuous-variable features:

	Debt	YearsEmployed	CreditScore	Income	Age
count	690	690	690	690	690
mean	4.758725	2.223406	2.400000	1017.385507	31.110720
std	4.978163	3.346513	4.862940	5210.102598	11.714832
min	0.0	0.0	0.0	0.0	13.750000
max	28.0	28.5	67.0	100000.0	80.25

Table 2: Numerical Data and their properties

For continuous-variable, We filled in the missing values by mean imputation and finished visualization as the figure shows below:

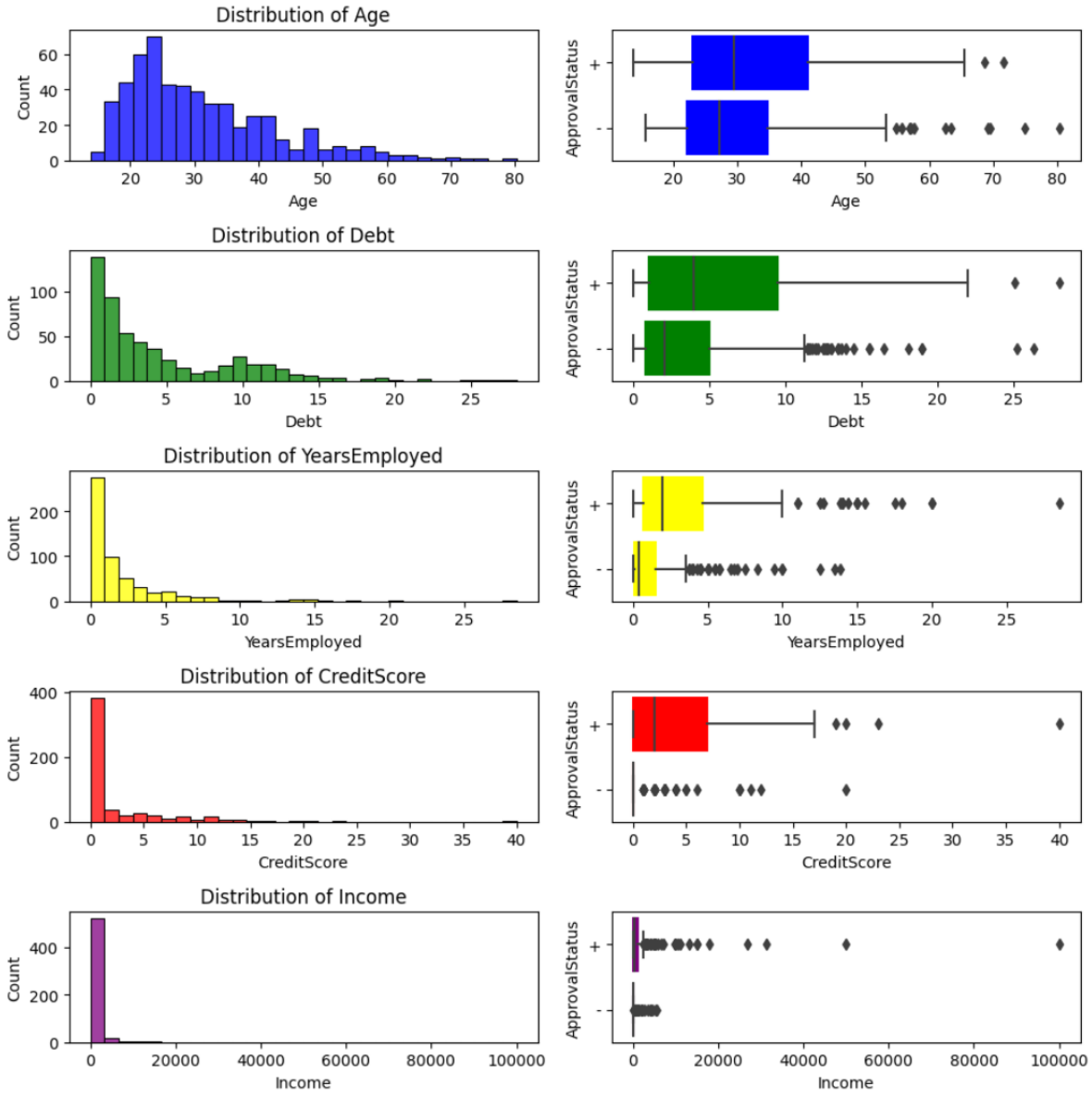

Figure 2: Histogram and Box plots of numerical features

From Figure 2, we can find that:

1. Continuous-variables distributions are highly right-skewed, with most individuals having younger ages, lower amounts of debt, fewer years of employment, lower credit scores, and lower income.

2. The Age and Debt features have the most amount of variance because the boxes are sparsely grouped about the mean. The Income feature has the least amount of variance because the boxes are tightly grouped about the mean.

3. For all five continuous features, the interquartile range for "Approved" is larger than the "Disapproved", which means that applicants whose applications are approved normally have older age, higher debt, longer working years, higher credit scores, and more income.

4. However, several outlying applicants with high feature values still were not granted credit.

Then, we calculated the correlation matrix between the Approval Status and other numerical features as below:

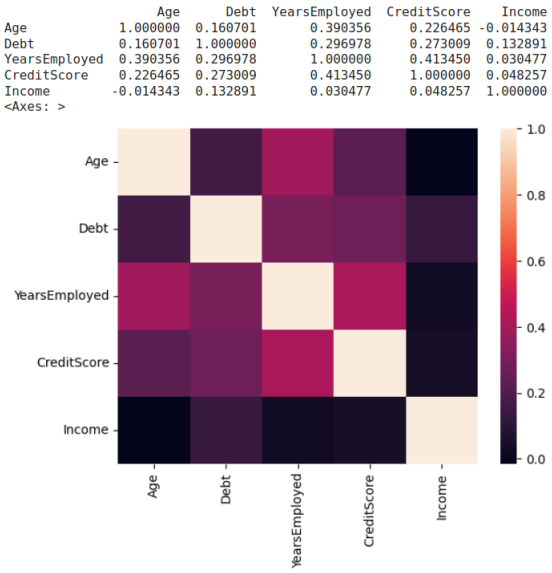

Figure 3: Heatmap of numerical features

```
SignificanceResult(statistic=0.13873426929220276, pvalue=0.0010833151973504691)
SignificanceResult(statistic=0.17584243394349006, pvalue=3.261277225637514e-05)
SignificanceResult(statistic=0.32501827875540223, pvalue=4.7813103727240735e-15)
SignificanceResult(statistic=0.44478618794229885, pvalue=3.5713546197895538e-28)
SignificanceResult(statistic=0.16919199267002194, pvalue=6.47113867928192e-05)
```

Figure 4: P-values of numerical features and output feature

From Figure 3, we discover that none of the continuous-variable features are highly correlated with others. Additionally, from Figure 4, we discover that all of these features are highly correlated with the output feature since the p-values are all very small. This indicates that all of these continuous variables are suitable for training.

2.2.2 Categorical-variable Features

For categorical variables, we first filled in the missing values by imputing these missing values with the most frequent values present in the respective columns. Then, we transformed the character symbols into numerical values and visualized their distributions as below:

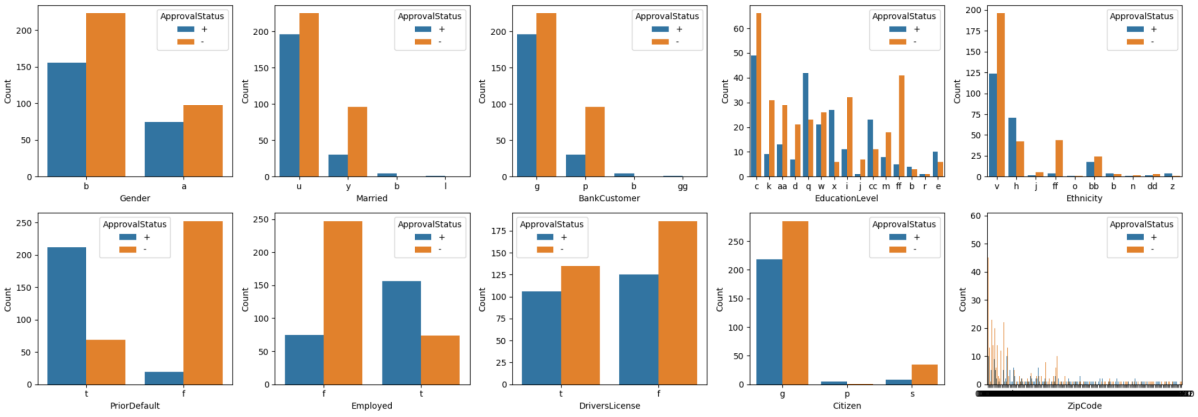

Figure 5: Bar plots of categorical features

From Figure 5, we can find that:

1. For any category in Categorical-variable (e.g., Married, Bank Customer, Prior Default, and Employed), the approval rate for a certain subcategory is much greater than for other subcategories, which is useful in future classification.

2. For Education Level and Zip Code, there are many dimensions of subcategory data, which reduces the calculation speed of the model.

We then inspected the relationship between features (inputs and output) by computing the correlation matrix as below:

	Gender	Married	BankCustomer	EducationLevel	Ethnicity	PriorDefault	Employed	DriversLicense	Citizen	ZipCode
.	0.000000e+00	5.665000e-01	5.665000e-01	1.023253e-09	1.433220e-03	5.286398e-01	2.083897e-02	0.455844	8.553959e-02	7.931622e-01
.	5.665000e-01	0.000000e+00	0.000000e+00	1.067037e-50	1.083623e-109	7.853650e-04	5.709488e-05	0.215020	2.781316e-96	8.554727e-33
.	5.665000e-01	0.000000e+00	0.000000e+00	1.067037e-50	1.083623e-109	7.853650e-04	5.709488e-05	0.215020	2.781316e-96	8.554727e-33
.	1.023253e-09	1.067037e-50	1.067037e-50	0.000000e+00	0.000000e+00	1.014386e-07	3.174949e-07	0.102246	2.480132e-37	2.213641e-02
.	1.433220e-03	1.083623e-109	1.083623e-109	0.000000e+00	0.000000e+00	6.252748e-09	7.052835e-02	0.079751	7.342771e-45	2.121361e-03
.	5.286398e-01	7.853650e-04	7.853650e-04	1.014386e-07	6.252748e-09	0.000000e+00	2.841592e-20	0.010973	1.529070e-03	1.902489e-01
.	2.083897e-02	5.709488e-05	5.709488e-05	3.174949e-07	7.052835e-02	2.841592e-20	0.000000e+00	1.000000	1.597070e-07	8.329126e-02
.	4.558445e-01	2.150197e-01	2.150197e-01	1.022456e-01	7.975068e-02	1.097269e-02	1.000000e+00	0.000000	3.248347e-01	3.032947e-01
.	8.553959e-02	2.781316e-96	2.781316e-96	2.480132e-37	7.342771e-45	1.529070e-03	1.597070e-07	0.324835	0.000000e+00	1.740911e-03
.	7.931622e-01	8.554727e-33	8.554727e-33	2.213641e-02	2.121361e-03	1.902489e-01	8.329126e-02	0.303295	1.740911e-03	0.000000e+00

Figure 6: Correlation matrix between Categorical-variable Features

```
Gender 0.6956482433490574
Married 4.34383510597917e-06
BankCustomer 4.34383510597917e-06
EducationLevel 1.4257931883226264e-12
Ethnicity 2.3410516355101748e-07
PriorDefault 4.471584228373263e-59
Employed 3.4226268728444267e-25
DriversLicense 0.4188781537069455
Citizen 0.0008049374272952941
ZipCode 0.011911728402362768
```

Figure 7: P-value of Categorical features and output feature in Chi-square test

Based on the chi-square matrix, we can discover that the Gender, Drivers License, and Zip Code features have relatively high p-values with most of the other features, which means they do not have a strong relationship with other features. We also found that Gender and Drivers License are not correlated to output features (p-values are high). Therefore, we decided to discard them.

3 Methodology

In this section, we performed preliminary training on various machine learning methods, from which we then selected the best-performing method for further tuning.

3.1 Preliminary Training

In preliminary training, We used a variety of popular methods for classification tasks via SKI Learn, such as k-nearest neighbors, naive Bayes, decision trees, support vector machines, boosting, logistic regression, gradient boosting decision tree, random forest, XGBoost, multilayer perception, and stochastic gradient descent classifier. Here is a basic comparison of those methods:

Method	**Strengths**
K-Nearest Neighbors (distance weighting, 'ball tree')	Simple, effective for small datasets
Gaussian Naive Bayes	Fast; works well with high dimensions
Bernoulli Naive Bayes	Works well with binary/boolean features
Decision Trees ('gini', 'entropy')	Easy to interpret; handles both numerical and categorical data
Support Vector Machine (C=100,000)	Effective in high dimensional spaces; robust against overfitting
AdaBoost (learning rate=2, 100 estimators,)	Improves classification accuracy; less prone to overfitting
Logistic Regression (C=10,000, stringent tolerance, 'sag' solver)	Fast; provides probabilities for outcomes
Gradient Boosting Decision Tree (depth=6, 100 estimators)	Handles different types of predictor variables; robust to outliers
Random Forest ('gini', 'entropy')	Handles large datasets with higher dimensionality; reduces overfitting
XGBoost	Fast; handles different types of predictor variables; reduces overfitting
Multilayer Perceptron (L2 penalty, alpha=0.01)	Can model non-linear relationships; robust to outliers
Stochastic Gradient Descent Classifier (L2 penalty, alpha=0.01)	Efficient for large datasets; supports different loss functions

Table 3: Comparison of Machine Learning Methods

For each classifier in our list, the following steps were performed:

- Fit the model to the training data using the default or specified configuration settings.

- Predict the outcomes of the test data using the fitted model.
- Calculate the evaluation metrics for the test data using the predicted outcomes and the true labels.
- Log the metrics in a Pandas DataFrame for comparison.

Classifier	Accuracy ↑	Recall ↑	F1 Score ↑	ROC AUC ↑
KNN	84.058	78.947	84.507	0.846
GaussianNB	53.623	19.737	31.915	0.574
BernoulliNB	85.507	78.947	85.714	0.862
DecisionTree gini	84.058	84.211	85.333	0.840
DecisionTreeEntropy	84.058	82.895	85.135	0.842
SVM	81.159	78.947	82.192	0.814
AdaBoost	86.957	85.526	87.838	0.871
Logistic	84.058	81.579	84.932	0.843
GBDT	84.058	85.526	85.526	0.839
RandomForest entropy	87.681	86.842	88.591	0.878
RandomForest gini	87.681	85.526	88.435	0.879
XGBoster	84.058	84.211	85.333	0.840
MLPClassifer	84.783	81.579	85.517	0.851
SGDClassifier	**89.855**	**94.737**	**91.139**	**0.893**

Table 4: Comparison of performance between different models.

After performing preliminary training on each method, we compared the performance of each model and selected the best-performing model for further optimization. As we learned from the course material, the evaluation matrics including Accuarcy, Recall, F1 Score, and ROC AUC, were used to evaluate the performance of each model. The performance of each method under the evaluation matrices is shown in Table 4. After comparison, we chose SGD classifier for our fine-tuning in the next part.

3.2 Stochastic Gradient Descent and Random Forest Method

From the figure above and through unpacking the best parameters, We can see that SGD Classifier is better than others in terms of all metrics. Therefore, the SGD classifier was chosen for further fine-tuning.

As we learned in class, Stochastic Gradient Descent (SGD) is an iterative optimization algorithm that minimizes an objective function with suitable smoothness properties.

Mathematically, the optimization problem is defined as follows,

$$\text{Given } D = \{y_i\}_N \overset{\text{iid}}{\sim} Y, \text{ find} \qquad \underline{\theta^*} = \arg\min_{\underline{\theta}} E[L(Y\,;\,\underline{\theta})] \tag{1}$$

The dataset D is an *iid* sample of a random variable Y. In our case, this Y is a Bernoulli random variable corresponding to our model's prediction of approval results (approval or rejection). The decision variable θ is an array of parameters we used for our model. For each value of θ = (θ_1, θ_2, ...), the variations in Y produce variations in L(Y ; θ).

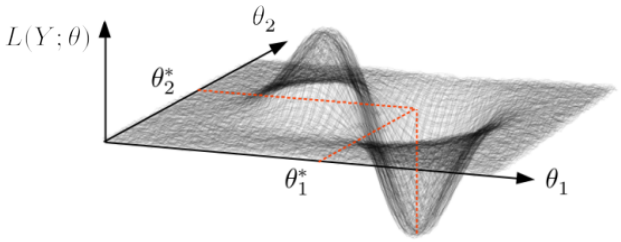

Figure 8: Stochastic cost function

For Stochastic Gradient Descent (SGD) method, the following algorithm is used to approach the optimal feature θ^* iteratively as shown in Figure 8:

$$\underline{\theta}_{k+1} = \underline{\theta}_k - \gamma \sum_{y_i \in B} \nabla_{\underline{\theta}} L(y_i;\, \underline{\theta}) \tag{2}$$

It is worth mentioning that the Random Forest Method also results in high accuracy in our application. The idea of a Random Forest Classifier is to use a bagging technique, bootstrap aggregating, to create different subsets of data from the original dataset and train a decision tree on each subset. The final prediction is then made by taking the majority vote of the individual trees.

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x) \tag{3}$$

, where $\hat{y}$ is the predicted class, B is the number of trees in the forest, and $T_b(x)$ is the prediction of the b-th tree for the input x. This equation shows that the random forest classifier is an ensemble method that averages the predictions of many decision trees.

4 Results and Analysis

In regards to fine-tuning, a grid search over ranges of different hyperparameters is adopted to optimize the Stochastic Gradient Descent classifier.

Specifically, the following list of hyperparameter ranges is adopted:

1. The regularization function penalty: l1, l2, elasticnet (combined l1 and l2)
2. The constant that multiplies the regularization term, which is also the learning rate α: np.logspace(-4, -1, 10)
3. The number of iterations with no improvement to wait before stopping fitting: [4,5,6,7,8,9]
4. The loss function to be used: hinge (linear SVM), log-loss (logistic regression), perception (linear loss used by the perceptron algorithm)
5. The Elastic Net mixing parameter l1-ratio: np.linspace(0.5, 0.8, 20)

From the grid search, the resulting accuracy for each set of discrete parameters is plotted with respect to α on a log scale:

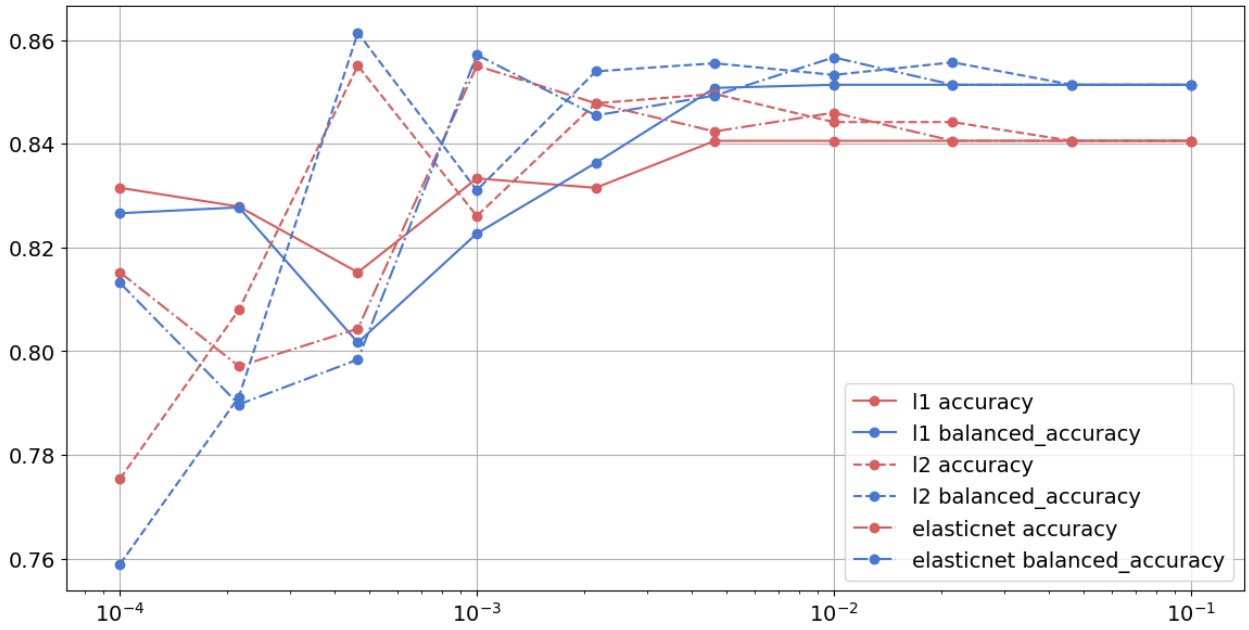

Figure 9: Fine Tuning with Alpha and Penalty

From the figure above, the optimal hyperparameters we obtained are as follows:

- alpha: 0.002154434690031882
- l1-ratio: 0.5631578947368421
- loss function: log-loss (shown in Figure 10). This gives a logistic regression model (a probabilistic classifier).
- The number of iterations with no improvement to wait before stopping fitting: 8
- penalty: l2

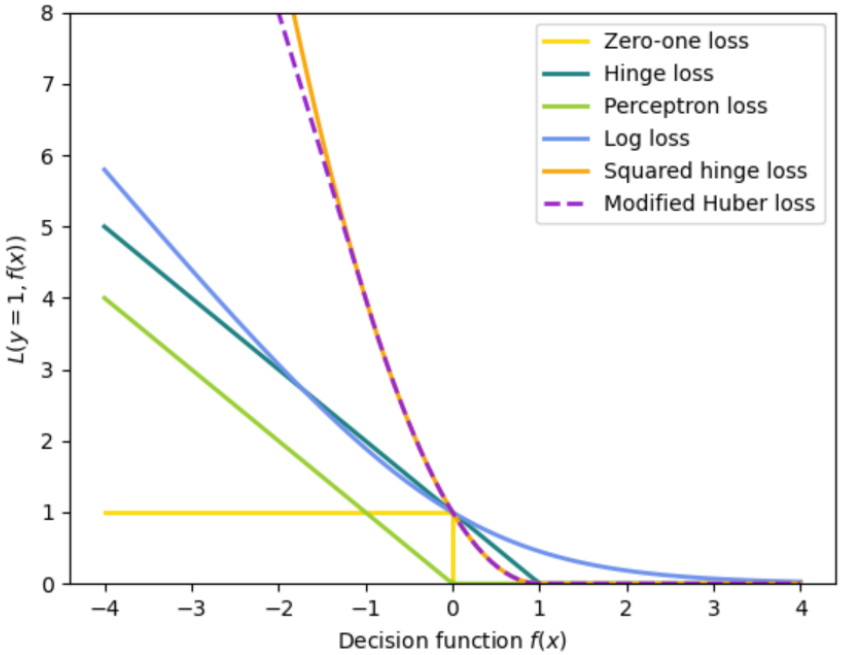

Figure 10: Loss functions for Stochastic Gradient Descent method[2]. The blue curve is the optimal loss function we found from hyperparameter search.

It is worth pointing out that the accuracy of the best model is around 84.05%, and the balanced accuracy of the best model is around 84.48%, which are lower than the result from the preliminary training. According to this page[3], the best score in the training dataset does not ensure a better performance in the test dataset. Although the test score in the test dataset is lower than in the training dataset, we believe our tuning has achieved the best outcomes.

References

[1] https://www.kaggle.com/datasets/youssefaboelwafa/credit-card-approval.

[2] https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_loss_functions.

[3] https://stackoverflow.com/questions/68952967/the-test-accuracy-score-is-higher-than-the-best-score-in-gridsearchcv