# SVM

## The idea of SVM is simple, the algorithm creates a hyperplane (line) which separates the data into classes of ‘Loan payers’ and ‘Charged off’. According to the SVM algorithm we find the points closest to the line from both the classes. These points are called support vectors. Then, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.

## **Feature Selection**

After pre-processing of the dataset, we selected the features according to the k highest scores using sklearn.feature\_selection.SelectKBest module. After trying with different values of ‘K’ I decided to go with k=9, for which we got better results. Split the data into training set (70%), and test set (30%). Training set will be used to fit the model, and test set will be to evaluate the best model to get an estimation of generalization error. Instead of having validation set to tune hyperparameters and evaluate different models, we’ll use 10-folds cross validation because it’s more reliable estimate of generalization error.

**Model Validation**

There is no silver bullet for choosing a model—often it comes down to the goals of implementation, the tradeoff between identifying more delinquent loans at the cost of misclassification. In many real-world settings, accuracy or error are not the best quality metrics for classification. For the model evaluation, we have choosen, three of the most popular classification metrics that can be used to compare the predicted and actual values, which are recall, precision, and the f1-score.

SVM

**With default values**

Classification Report:

precision recall f1-score support

0 0.84 1.00 0.91 5795

1 0.00 0.00 0.00 1099

micro avg 0.84 0.84 0.84 6894

macro avg 0.42 0.50 0.46 6894

weighted avg 0.71 0.84 0.77 6894

accuracy: 0.8405860168262257

Confusion Matrix:

[[5795 0]

[1099 0]]

**After Hyper parameter tuning on f1**

Classification Report:

precision recall f1-score support

0 0.87 0.57 0.69 5795

1 0.20 0.56 0.29 1099

micro avg 0.57 0.57 0.57 6894

macro avg 0.53 0.56 0.49 6894

weighted avg 0.76 0.57 0.63 6894

accuracy: 0.5673049028140412

Confusion Matrix:

[[3298 2497]

[ 486 613]]

**Boosting**

Build models sequentially. That means each model learns from the residuals of the previous model. In our project, we decided to use XGBoost and AdaBoost Classifier with Decision Trees as base estimators in them. We choose Decision Tree as the base estimator because results of Decision Tree have been promising on the given dataset.

XGBoostClassifier

Classification Report:

precision recall f1-score support

0 0.85 0.93 0.89 5795

1 0.31 0.16 0.21 1099

micro avg 0.81 0.81 0.81 6894

macro avg 0.58 0.55 0.55 6894

weighted avg 0.77 0.81 0.78 6894

accuracy: 0.8099796924862199

Confusion Matrix:

[[5409 386]

[ 924 175]]

AdaBoostClassifier

Classification Report:

precision recall f1-score support

0 0.85 0.93 0.89 5795

1 0.30 0.16 0.21 1099

micro avg 0.81 0.81 0.81 6894

macro avg 0.58 0.54 0.55 6894

weighted avg 0.77 0.81 0.78 6894

accuracy: 0.8086742094574992

Confusion Matrix:

[[5403 392]

[ 927 172]]

**Strategies to deal with imbalanced datasets**

* Under-Sample: Under-sample the majority class with or w/o replacement by making the number of positive and negative examples equal. One of the drawbacks of under-sampling is that it ignores a good portion of training data that has valuable information. However, it’s very fast to train.
* Over-Sample: Over-sample the minority class with or w/o replacement by making the number of positive and negative examples equal. It’s a lot more computationally expensive than under-sampling. Also, it’s more prune to overfitting due to repeated examples. We used SMOTE, as it over-samples the minority class but using synthesized examples.

For the model training, we have used the combination of both under-sampling and over-sampling, which in practice yields a better result.

|  |  |  |
| --- | --- | --- |
| Dataset Ratio (After train\_test\_split) | Loan Status = Paid | Loan Status = Not Paid |
| Original Dataset | 19411 | 3682 |
| After under-sampling | 10000 | 3682 |
| After over-sampling | 10000 | 10000 |

**XGBClassifier**

**XGBClassifier(learning\_rate =1,**

**n\_estimators=500,**

**max\_depth=4)**

Classification Report:

precision recall f1-score support

0 0.86 0.82 0.84 9561

1 0.25 0.33 0.29 1814

micro avg 0.74 0.74 0.74 11375

macro avg 0.56 0.57 0.56 11375

weighted avg 0.77 0.74 0.75 11375

accuracy: 0.7390769230769231

Confusion Matrix:

[[7814 1747]

[1221 593]]

**AdaBoostClassifier**

**AdaBoostClassifier(base\_estimator=DecisionTreeClassifier(max\_depth=3), n\_estimators=500)**

Classification Report:

precision recall f1-score support

0 0.86 0.81 0.83 9561

1 0.23 0.30 0.26 1814

micro avg 0.73 0.73 0.73 11375

macro avg 0.54 0.56 0.55 11375

weighted avg 0.76 0.73 0.74 11375

accuracy: 0.7265934065934065

Confusion Matrix:

[[7713 1848]

[1262 552]]

**SVM Classifier**

**With default values**

Classification Report:

precision recall f1-score support

0 0.91 0.45 0.61 9561

1 0.21 0.77 0.33 1814

micro avg 0.51 0.51 0.51 11375

macro avg 0.56 0.61 0.47 11375

weighted avg 0.80 0.51 0.56 11375

accuracy: 0.5050549450549451

Confusion Matrix:

[[4340 5221]

[ 409 1405]]