Estimating the Probability of Default for Credit Card Clients

Vamsi Desu

CS 4641 – Final Project in Supervised Learning

04/19/2020

**Introduction to the Dataset**

This dataset contains the information of Taiwanese credit card customers. It includes amount of credit, gender, age, education level, marital status, repayment status per month over six months in 2005, bill statement per month over six months in 2005, amount paid per month over 6 month in 2005, and whether the customer defaulted (i.e. 0 means has not defaulted, 1 means default). Using this data on credit card customers’ information, the following question can be asked: can we predict the probability of default for a certain customer? This can be turned into a supervised learning question. Supervised learning is the machine learning task of learning a function that maps an input to an output given example inputs and output pairs. In this dataset’s case, the example input would be all the customer’s credit card information data which was mentioned above and the example output would be whether or not the customer defaulted on their credit card (labeled as 0 or 1 for modeling purposes). This example input and output pairs can help us generate a function that maps a new customer to their probability of defaulting. We can generate this function by using supervised learning techniques such as random forest, support vector machines, and neural networks. Furthermore, this is a classification problem, which means the output is discrete and has labels(in this case, 0 and 1). This problem is important to me because my father worked in risk management during the 2008 financial crisis and subsequently lost his job. One of the issues during that recession was the amount of creditors that were defaulting. If machine learning techniques were used on the data during that time, faulty creditors would be found and my father and many others may have not lost their job during that time. The accuracy/score will be used as a metric to measure performance.

**Description of the Algorithms**

Since this problem is a classification one, we will use random forests with bagging, support vector machines, and a neural network. The scikit-learn library was used for these algorithms.

**Random Forests with Bagging**

Algorithm

A random forest consists of a large number of individual decision trees fitting to the training data, where each decision tree produces a class prediction and the class with the most votes becomes the entire model’s prediction. Usually, from prior knowledge of a decision tree, the “best” tree is usually produced from the training dataset. However, bagging solves this problem and allows us to produce multiple trees. Bagging does this by creating randomly sampling our data and splitting points randomly instead of optimally. Bagging aims to reduce the complexity and overfitting of the training data.

Hyperparameters

To implement a random forest with bagging, the RandomForestClassifier was used from the sklearn.ensemble module. The classifier uses bagging by default. The hyperparameters tuned are in a list below. Each hyperparameter’s performance

* n\_estimators: This specifies the number of trees in the forest of the model.

A screenshot of a map

Description automatically generated

The complexity of the model increases as n\_estimators is increased and it could result in overfitting. However, at a certain number of trees, the error rate will stabilize and not improve.

* max\_depth: This specifies the maximum depth of each tree in the forest.

The complexity of the model increases as max\_depth is increased and it could result in overfitting.

* min\_samples\_split: The minimum number of samples required to split an internal leaf node.
* min\_samples\_leaf: The minimum number of samples required to be at a leaf node.

The complexity of the model decreases as min\_samples\_leaf is increased, as it is easier for it to generalize.

**Support Vector Machine**

Algorithm

Support Vector Machines(SVMs) are based on the idea of finding a decision boundary that best divides a dataset into two classes by maximizing the margin. If the data is linearly separable, a hyperplane can be used as the decision boundary. However, if not, the kernel trick can be used to transform the data into a higher dimensionality space, where a nonlinear decision boundary can be formed.

Hyperparameters

To implement a random forest with bagging, the RandomForestClassifier was used from the sklearn.ensemble module. The classifier uses bagging by default. The hyperparameters tuned are in a list below.

* n\_estimators: This specifies the number of trees in the forest of the model.
* max\_depth: This specifies the maximum depth of each tree in the forest.
* min\_samples\_split: The minimum number of samples required to split an internal leaf node.
* min\_samples\_leaf: The minimum number of samples required to be at a leaf node.

**Tuning Hyperparameters**

**Comparing Algorithm Performance**

**Conclusion**

**Acknowledgments**

C gamma kernel Accuracy

0 0.1 1.000 rbf 0.809208

1 0.1 1.000 poly 0.812667

2 0.1 1.000 sigmoid 0.687875

3 0.1 0.100 rbf 0.778875

4 0.1 0.100 poly 0.778708

5 0.1 0.100 sigmoid 0.778708

6 0.1 0.010 rbf 0.778708

7 0.1 0.010 poly 0.778708

8 0.1 0.010 sigmoid 0.778708

9 0.1 0.001 rbf 0.778708

10 0.1 0.001 poly 0.778708

11 0.1 0.001 sigmoid 0.778708

12 1.0 1.000 rbf 0.817708

13 1.0 1.000 poly 0.818958

14 1.0 1.000 sigmoid 0.662750

15 1.0 0.100 rbf 0.795167

16 1.0 0.100 poly 0.780375

17 1.0 0.100 sigmoid 0.790500

18 1.0 0.010 rbf 0.778708

19 1.0 0.010 poly 0.778708

20 1.0 0.010 sigmoid 0.778708

21 1.0 0.001 rbf 0.778708

22 1.0 0.001 poly 0.778708

23 1.0 0.001 sigmoid 0.778708

24 10.0 1.000 rbf 0.821208

25 10.0 1.000 poly 0.820250

26 10.0 1.000 sigmoid 0.660292

27 10.0 0.100 rbf 0.812375

28 10.0 0.100 poly 0.801542

29 10.0 0.100 sigmoid 0.724042

30 10.0 0.010 rbf 0.787292

31 10.0 0.010 poly 0.778708

32 10.0 0.010 sigmoid 0.780292

33 10.0 0.001 rbf 0.778708

34 10.0 0.001 poly 0.778708

35 10.0 0.001 sigmoid 0.778708

36 100.0 1.000 rbf 0.818625

37 100.0 1.000 poly 0.818250

38 100.0 1.000 sigmoid 0.660000

39 100.0 0.100 rbf 0.821167

40 100.0 0.100 poly 0.812708

41 100.0 0.100 sigmoid 0.703750

42 100.0 0.010 rbf 0.798917

43 100.0 0.010 poly 0.778708

44 100.0 0.010 sigmoid 0.809500

45 100.0 0.001 rbf 0.786042

46 100.0 0.001 poly 0.778708

47 100.0 0.001 sigmoid 0.780125

SVC(C=10, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma=1, kernel='rbf', max\_iter=-1,

probability=False, random\_state=None, shrinking=True, tol=0.001,

verbose=False)

|  |  |  |  |
| --- | --- | --- | --- |
| **C** | **gamma** | **kernel** | **Accuracy** |
| 0.1 | 1.000 | rbf | 0.809208 |
| 0.1 | 1.000 | poly | 0.812667 |
| 0.1 | 1.000 | sigmoid | 0.687875 |
| 0.1 | 0.100 | rbf | 0.778875 |
| 0.1 | 0.100 | poly | 0.778708 |
| 0.1 | 0.100 | sigmoid | 0.778708 |
| 0.1 | 0.010 | rbf | 0.778708 |
| 0.1 | 0.010 | poly | 0.778708 |
| 0.1 | 0.010 | sigmoid | 0.778708 |
| 0.1 | 0.001 | rbf | 0.778708 |
| 0.1 | 0.001 | poly | 0.778708 |
| 0.1 | 0.001 | sigmoid | 0.778708 |
| 1.0 | 1.000 | rbf | 0.817708 |
| 1.0 | 1.000 | poly | 0.818958 |
| 1.0 | 1.000 | sigmoid | 0.662750 |
| 1.0 | 0.100 | rbf | 0.795167 |
| 1.0 | 0.100 | poly | 0.780375 |
| 1.0 | 0.100 | sigmoid | 0.790500 |
| 1.0 | 0.010 | rbf | 0.778708 |
| 1.0 | 0.010 | poly | 0.778708 |
| 1.0 | 0.010 | sigmoid | 0.778708 |
| 1.0 | 0.001 | rbf | 0.778708 |
| 1.0 | 0.001 | poly | 0.778708 |
| 1.0 | 0.001 | sigmoid | 0.778708 |
| 10.0 | 1.000 | rbf | 0.821208 |
| 10.0 | 1.000 | poly | 0.820250 |
| 10.0 | 1.000 | sigmoid | 0.660292 |
| 10.0 | 0.100 | rbf | 0.812375 |
| 10.0 | 0.100 | poly | 0.801542 |
| 10.0 | 0.100 | sigmoid | 0.724042 |
| 10.0 | 0.010 | rbf | 0.787292 |
| 10.0 | 0.010 | poly | 0.778708 |
| 10.0 | 0.010 | sigmoid | 0.780292 |
| 10.0 | 0.001 | rbf | 0.778708 |
| 10.0 | 0.001 | poly | 0.778708 |
| 10.0 | 0.001 | sigmoid | 0.778708 |
| 100 | 1.000 | rbf | 0.818625 |
| 100 | 1.000 | poly | 0.818250 |
| 100 | 1.000 | sigmoid | 0.660000 |
| 100 | 0.100 | rbf | 0.821167 |
| 100 | 0.100 | poly | 0.812708 |
| 100 | 0.100 | sigmoid | 0.703750 |
| 100 | 0.010 | rbf | 0.798917 |
| 100 | 0.010 | poly | 0.778708 |
| 100 | 0.010 | sigmoid | 0.809500 |
| 100 | 0.001 | rbf | 0.786042 |
| 100 | 0.001 | poly | 0.778708 |
| 100 | 0.001 | sigmoid | 0.780125 |

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=25, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=100,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_jobs=None, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)