Estimating the Probability of Default for Credit Card Clients

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 forests, 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 many jobs would have been saved. The accuracy/score, which is the fraction of correct prediction, will be used as a metric to measure performance. [anything else?]

**Description of the Algorithms**

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

Eighty percent of the data was considered as the training dataset and the other twenty percent was considered as the testing dataset.

**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 individual hyperparameter’s performance and their relation to the complexity of the hypothesis class is discussed below.

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

A screenshot of a map

Description automatically generated

This graph displays the training and test data’s accuracy scores as the n\_estimators parameter increased. 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.

A close up of a map

Description automatically generated

This graph displays the training and test data’s accuracy scores as the max\_depth parameter increased. The complexity of the model increases as max\_depth is increased and it could result in overfitting with a max\_depth too high, as seen in the graph above. As the max\_depth increases, the model accommodates well for the training data, but testing data accuracy scores falter, indicating overfitting and poor generalization.

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

A screenshot of a map

Description automatically generated

This graph displays the training and test data’s accuracy scores as the min\_samples\_split parameter increased. The complexity of the model decreases as min\_samples\_split increases. This makes sense intuitively, as the minimum samples required to split a leaf increases, the fewer number of splits that are possible in each tree. This makes the complexity decrease and generalization increase too. A lower value for this hyperparameter makes sense. The default value for the parameter is 2 and choosing very high values will make the nodes too general.

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

A screenshot of a cell phone

Description automatically generated

This graph displays the training and test data’s accuracy scores as the min\_samples\_split parameter increased. The complexity of the model decreases as min\_samples\_split increases, similar to the parameter above. This makes sense intuitively, as the minimum samples required at a leaf increases, the shorter the tree would be. Since the default for this parameter is 1, the tuning for this hyperparameter may result in a number close to 1. The model should not generalize too much.

**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 SVM, the SVC (Support Vector Classification) was used from the sklearn.svm module. The hyperparameters tuned are in a list below. Each individual hyperparameter’s performance and their relation to the complexity of the hypothesis class is discussed below. Only non-linear kernels were used.

* **C**:  A parameter that controls the trade-off between the achieving a low training error and a low testing error that is the ability to generalize your classifier to unseen data. For instance, a larger C will choose a smaller-margin hyperplane if it is does a better job of getting all the training points classified correctly.

A close up of a map

Description automatically generated

This graph displays the training and test data’s accuracy scores as the C parameter increased. The complexity of the model increases as C increases. This makes sense intuitively, as a higher C will try to classify as many points in the training set correctly, which means it will begin to overfit. C should be chosen such that there is enough generalization and no overfitting of the data.

* **Gamma**: This parameter determines how far influences the calculation of the decision boundary. A high gamma only considers nearer points.

A screenshot of a cell phone

Description automatically generated

This graph displays the training and test data’s accuracy scores as the gamma parameter increased. The complexity of the model increases as gamma increases. As one can see in the graph, an increase in gamma will result in overfitting and the model will be unable to generalize well. Therefore, a lower gamma will be chosen.

* **Kernel**: This parameter takes a low dimensional input space and transforms it into a higher-dimensional space.

A screenshot of a cell phone

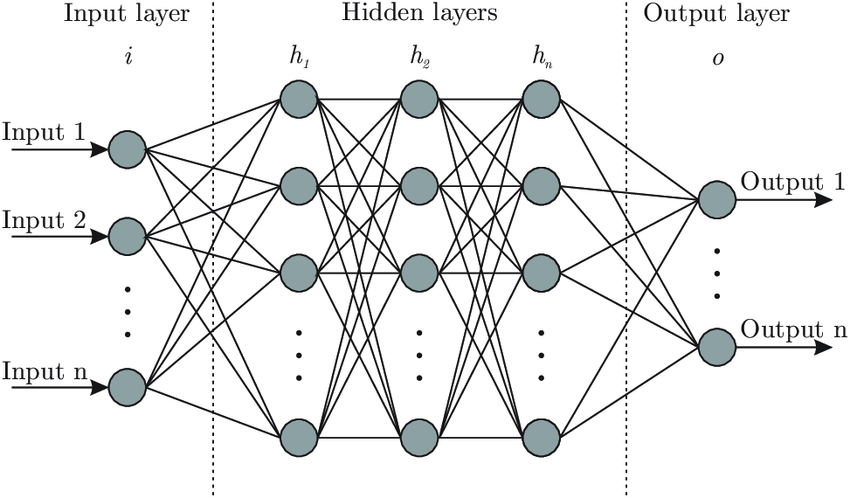
Description automatically generated

This graph displays the training and test data’s accuracy scores for each kernel. All the accuracy scores are extremely similar for both test and train data, so this will not be selected as a final hyperparameter to be tuned. I will use the default value for the kernel in the classifier, which is rbf.

**Neural Network**

Algorithm

Neural networks are organized in layers. Layers are made up of a number of interconnected nodes which contain an activation function. The input layer communicates to one or more hidden layers where processing is done using weights. The hidden layers link to an output layer. For a better understanding, the diagram below is a neural network with 3 hidden layers.



Hyperparameters

To implement a neural network, the MLP Classifier from the sklearn.nerual\_network module was used. The hyperparameters tuned are in a list below. Each individual hyperparameter’s performance and their relation to the complexity of the hypothesis class is discussed below.

* **hidden\_layer\_sizes**: This parameter determines the number of hidden layers and the number of nodes in each.

A picture containing drawing

Description automatically generated

This graph displays the training and test data’s accuracy scores for the hidden layer sizes. No clear pattern is seen in the graph, but the 2 hidden layers with 20 nodes each, 3 hidden layers with 50 nodes each, and 3 hidden layers with 50 nodes seems to produce the highest accuracy.

* **Alpha**: regularization parameter

A screenshot of a cell phone

Description automatically generated

This graph displays the training and test data’s accuracy scores as the alpha parameter increased. The complexity of the model decreases as alpha increases. As one can see in the graph, the smaller alpha is, the more complex the model is. As alpha increases, the model is able to generalize better.

* **Learning Rate**: The parameter controls how the initial learning rate value changes as the model learns

**A screenshot of a cell phone

Description automatically generated**

This graph displays the training and test data’s accuracy scores as the alpha parameter increased. The best accuracies are seen when the model uses a constant or adaptive learning rate. There seems to be no overfitting, as the testing scores are similar to the training scores.

**Tuning Hyperparameters**

My dataset was confirmed to be a non-trivial distribution, as I used support vector machine with a linear kernel to test this. The accuracy score of this model was only 0.739, which is a very low score compared to the other classifiers used in this report.

GridSearchCV was used to produce the best parameters for the all classifiers, which evaluates a model for each combination of hyperparameters in the grid. GridSearchCV is from the sklearn.model\_selection module. The results of tuning for each classifier are shown below.

**Random Forests**

Using the default hyperparameters, the accuracy of the random forest classifier with bagging was 0.805.

The following values were used for hyperparameter tuning using GridSearchCV with 3 cross-validation folds.

n\_estimators = [50, 100, 150, 200]

max\_depth = [5, 10, 15, 25]

min\_samples\_split = [2, 5, 10,15]

min\_samples\_leaf = [1, 2, 5]

The best accuracy came with n\_estimators as 200, max\_depth as 10, min\_samples\_split as 15, and min\_samples leaf as 2. The accuracy score was 0.8215. The total time for tuning was 26.2 minutes. Below are three tables, consisting of all the possible combinations tested. In each table, the first column is line number, the second column is the max\_depth, the third column is min\_samples\_leaf, the fourth column is min\_samples\_split, the fifth column is n\_estimators, and the sixth column is accuracy.

A close up of a logo

Description automatically generatedA picture containing boat, computer, white

Description automatically generatedA picture containing boat, white

Description automatically generated**Support Vector Machines**

Using the default hyperparameters, the accuracy of the support vector machine was 0.739.

The following values were used for hyperparameter tuning using GridSearchCV with 3 cross-validation folds. The default kernel ‘rbf’ was used since the performance for all three kernels tested earlier had similar accuracy scores.

C = [1/16,1/8,1/4,1/2,1,2,4,8,16]

gamma = [1/16,1/8,1/4,1/2,1,2,4,8,16]

The best accuracy came with C as 8 and gamma as 1. The accuracy score on the training data was 0.820. The total time for tuning was 46.4 minutes. Below are two tables, consisting of all the possible combinations tested. In each table, the first column is line number, the second column is the C, the third column is gamma, and the sixth column is accuracy.

A close up of a piece of paper

Description automatically generated A close up of a piece of paper

Description automatically generated

**Neural Networks**

Using the default hyperparameters, the accuracy of the support vector machine was 0.778.

The following values were used for hyperparameter tuning using GridSearchCV with 3 cross-validation folds.

learning\_rate= ['constant', 'invscaling', 'adaptive']

alpha=[0.1, 0.01, 0.001, 0.0001]

hidden\_layer\_sizes=[[20,20],[20,20,20], [30,30], [30,30,30], [40,40], [40,40,40], [50,50], [50, 50, 50]

The best accuracy came with a learning\_rate as invscaling, hidden\_layer\_size as 3 hidden layers with 50 nodes each, and an alpha 0.1. The accuracy score was 0.823. The total time for tuning was 35 minutes. Below are two tables, consisting of all the possible combinations tested. In each table, the first column is line number, the second column is the alpha, the third column is hidden\_layer\_sizes, the fourth column is learning\_rate, and the fifth column is accuracy.

A picture containing computer

Description automatically generated**A close up of text on a white background

Description automatically generated**

**Comparing Algorithm Performance**

A screenshot of a cell phone

Description automatically generated

The graph above shows the accuracy scores of the three models tuned when run on the testing dataset. The SVM performs slightly better than the other two models on the held-out testing data with a score of 0.827. The random forest and neural network have accuracy scores of 0.815 and 0.821, respectively. Overall, all classifiers have a relatively similar accuracy score, but the SVM beats out the other two models by a small margin. Since they all relatively have the same accuracy score, further comparison of algorithms can be done using the classification reports for each model.

**Random Forest - Classification Report**

**A screenshot of a cell phone

Description automatically generated**

**SVM - Classification Report**

A picture containing clock

Description automatically generated

**Neural Networks - Classification Report**

A screenshot of a cell phone

Description automatically generated

* **Precision** is the ability of a classifier not to label an instance positive that is actually negative. All reports do relatively for classifying datapoints/people who will not default correctly (i.e. have a label of 0). However, the SVM does the best for classifying defaulters correctly, with precision of 0.73. The other two models only have a precision of 0.68.
* **Recall** is the ability of a classifier to find all positive instances. For a label of 0, the SVM had the highest score of 0.97. For the label of 1, the neural network had the highest score of 0.40.
* **F1 score** is a weighted harmonic mean of precision and recall. For a label of 0, the SVM had a slightly higher score than the other two models at 0.9. However, for the label of 1, the neural network had the highest score at 0.51, which is better than the other two models which were at 0.46.

Since the training dataset (24,000 points) and the testing dataset (6,000 points) were both large, these are reliable experiments with valid and significant results. Overall, the accuracy scores for all three tuned models are extremely similar. However, when taking a deeper dive into the classification report, the tuned neural network seems to work best with this dataset. This is due to its high F1 scores for both labels, 0.89 and 0.51 respectively. F1 scores are more significant than accuracy scores, as the F1 score takes into account precision and recall. Although the SVM has a slightly higher accuracy and precision scores, the F1 score definitely takes precedence and therefore, the neural network seems like the best option.

**Conclusion**

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifier** | **Accuracy Score** | **Training Time** | **Hyperparameters Tuned** |
| Random Forest | 0.815 | 5.7 seconds | 4 |
| Support Vector Machine | 0.827 | 23.43 seconds | 2 |
| Neural Network | 0.821 | 16.36 seconds | 3 |

The table above provides a summary of the tuned models and their attributes. In today’s day and age, speed is everything. From devices such as IBM’s supercomputer Watson to iPhones, companies and individuals are trying to train and run models on these devices, small or large. Since the accuracy scores for all three models are relatively similar, looking at training time and hyperparameters tuned is a path for finding a model that will work best in the real world. Specifically, in the domain of finance, there’s data constantly being processed at all times and speed is key. With this in mind, I would choose the random forest in the real world, as the model only needs 5.7 seconds to train on 24,000 datapoints. Additionally, the model will retrain as more data is added so that time adds up if using the other two models, which take more than 15 seconds per train. Although the random forest has the most hyperparameters to tune, it took the least time to tune those hyperparameters out of all the models at only 26 minutes. Therefore, the random forest is the best for the real world in the finance domain due to its speed for both training and tuning the hyperparameters.

**Acknowledgments**

Random Forests:

<https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74>

<https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6>

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

Support Vector Machine:

<https://towardsdatascience.com/svm-hyper-parameter-tuning-using-gridsearchcv-49c0bc55ce29>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

Neural Networks:

<https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier>

<https://datascience.stackexchange.com/questions/36049/how-to-adjust-the-hyperparameters-of-mlp-classifier-to-get-more-perfect-performa>

<https://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html/2>

Dataset and Other Information/Libraries:

<https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients#>

<https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf>

<https://matplotlib.org/3.2.1/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py>