CSE574 Introduction to Machine Learning

Programming Assignment 3

Classification and Regression

Shi Yan, Weibin Ma

Part1. Binary logistic regression

Here is the comparison between training error and test error when using Logistic Regression. It clearly shows that their performance trend is similar. With the increase of training error, test error increases as well.
Moreover, we can easily find that the test error is slightly larger than training error in each category, which proves our model fit new data very well. No overfitting in our model.

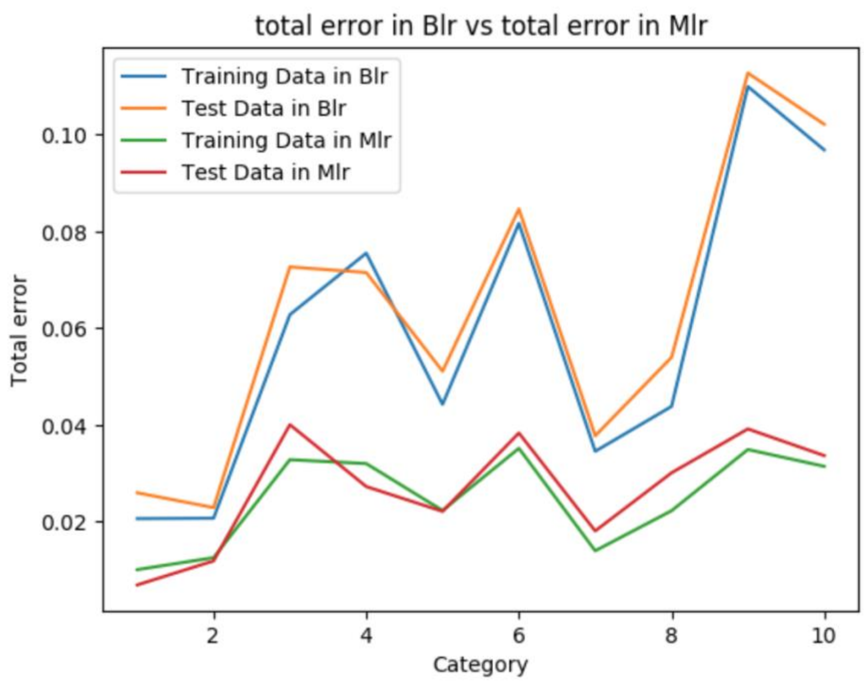
Here is the final prediction accuracy in our model with respect to training data, validation data and test data when using Logistic Regression.

Training set Accuracy: 92.742%
Validation set Accuracy: 91.53999999999999%
Testing set Accuracy: 92.01%

Thus, we generated a pretty good classification model using logistic regression method with more than 92% prediction accuracy in our test data. Additionally, from the total error of training data and test data in each category, we can easily find that the model performs very good in final prediction without obvious overfitting.

Part2. Multiple logistic regression

From the plot above, we can easily find that the total error trend between training error and test error using Multi-Class Logistic Regression is similar again. In other words, with respect to each category, the increase or decrease of training error, test error rises or descends as well. Moreover, the total training error is larger than total test error in each category. The reason for the difference is that training data has sample size up to 50000 while test data only has 10000 samples.



After normalization the total error, we can easily find similar output with Binary Logistic Regression, which means these two methods could generate similar prediction accuracy in our large dataset. Moreover, the total error in BLR is larger than the total error in MLR. Finally, if we only consider the time-consuming, the Multi-Class Logistic Regression performs much better than BLR as well.

Here is the final prediction accuracy in our model with respect to training data, validation data and test data when using Multi-Class Logistic Regression.

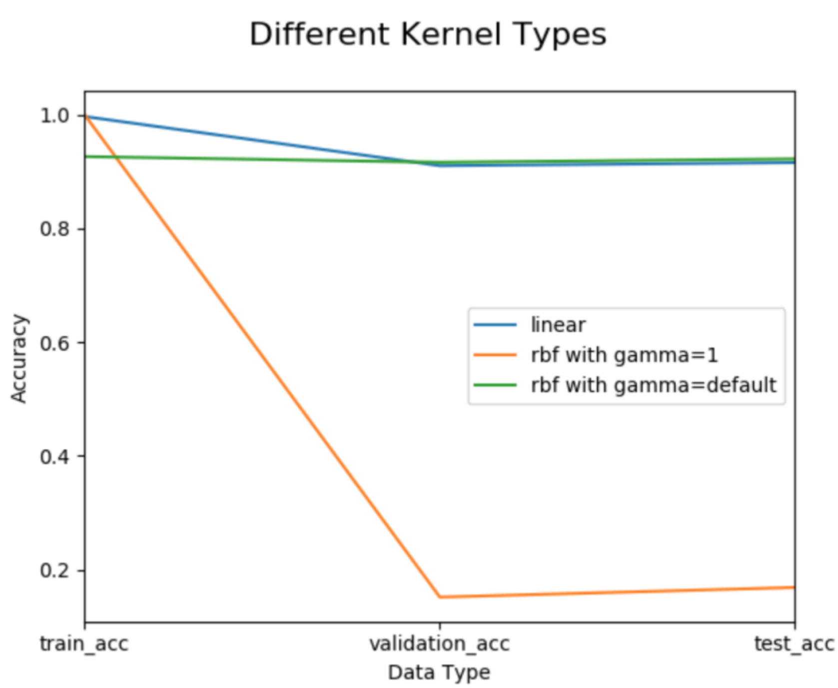
Training set Accuracy: 93.112%
Validation set Accuracy: 92.39%
Testing set Accuracy: 92.54%

By using Multi-Class Logistic Regression, we generated more than 92% accuracy both in training data and test data, which means our model performs good in class classification. Additionally, the performance difference between multi-class strategy and one-vs-all strategy is that we just need to compare (C-1) times if we want to find the highest probability of total C category, while we will have to compare (C-1) square times in one-vs-all strategy. Thus, using multi-class strategy will help save more time cost than one-vs-all strategy in large training data size.

Part3. SVM

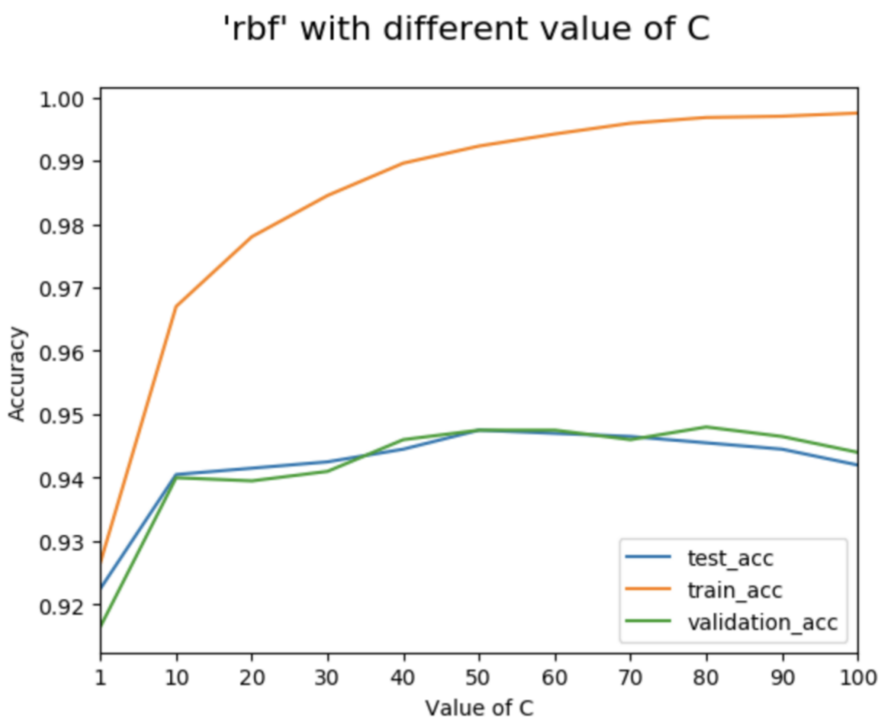
For different kernel:

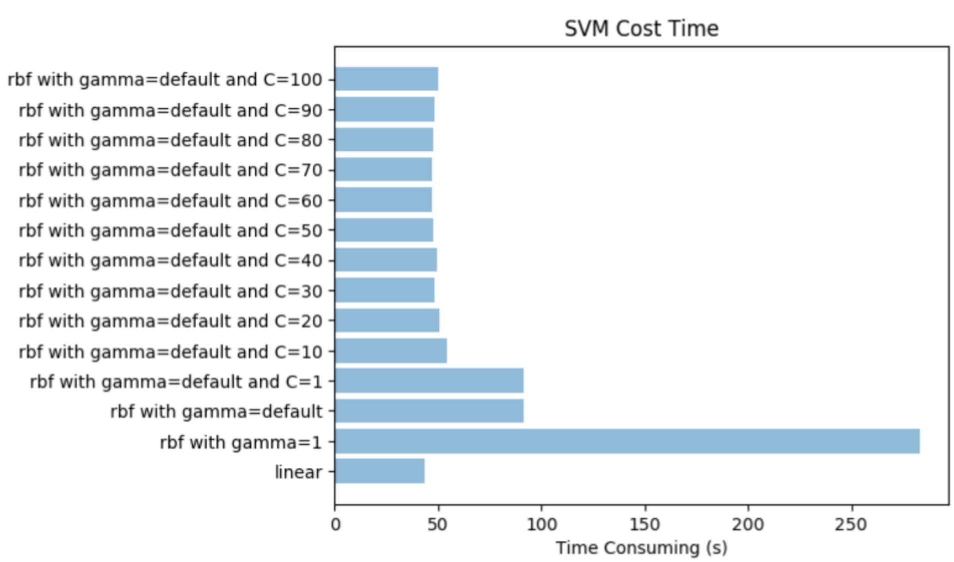
	linear	rbf with gamma=1	rbf with gamma=default
train_acc	0.9971	1.0000	0.9266
validation_acc	0.9105	0.1515	0.9165
test_acc	0.9160	0.1685	0.9225



From above, we could find that rbf kernel with gamma =1 performs worst and rbf kernel with default gamma performs best. When gamma is 1, it is overfitting, since the curve of decision boundary is too high. Thus, we can improve the test accuracy by control the value of gamma when using rbf kernel.

For rbf with different C:





C is penalty parameter of the error term. When C is small, there exists some misclassified data points, when C is large, the misclassified data was heavily penalized. So, when the value of C increase, the classifier preforms better, until it increases to some value, the performance archives the best and if you still increase C value, the performance start going down.
Additionally, from the time-consuming plot, it clearly shows that the time cost is highest when using rbf with gamma=1. Meanwhile, the time-consuming decrease as C begins to increase when using rbf with gamma=default. Moreover, the time-consuming is smallest when using linear kernel.