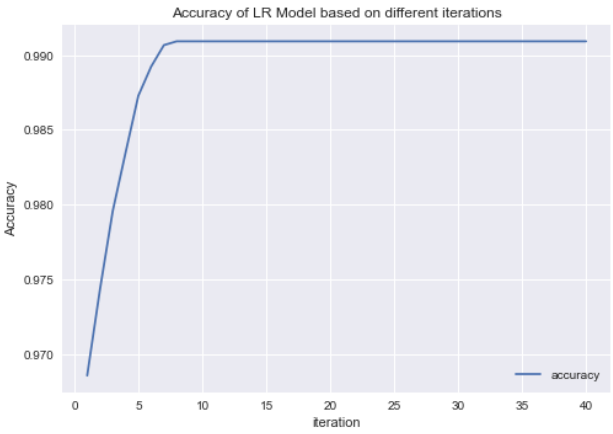
CS135 Project 1

Part One: Logistic Regression for Digit Classification

1.



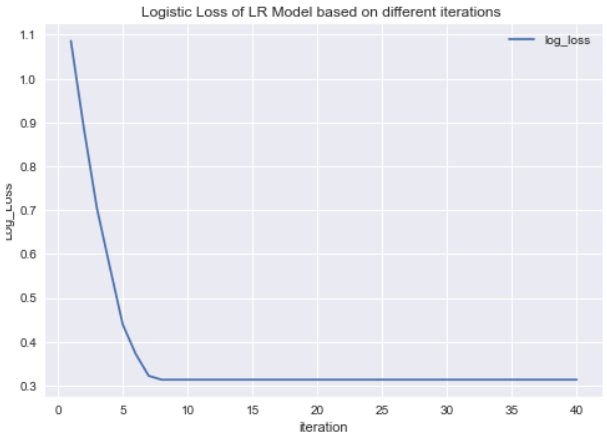


Fig.1 *Fig.2*

After 7 iterations, the accuracy climbs up from lower than 0.97 and then reaches about 0.995, Logistic Loss goes down from 1.1 to 0.32 and then they all stay on these values. Since the both accuracy and log loss are computed based on the training data set, they all finally converge and have the peak performance after certain times iterations.

2.

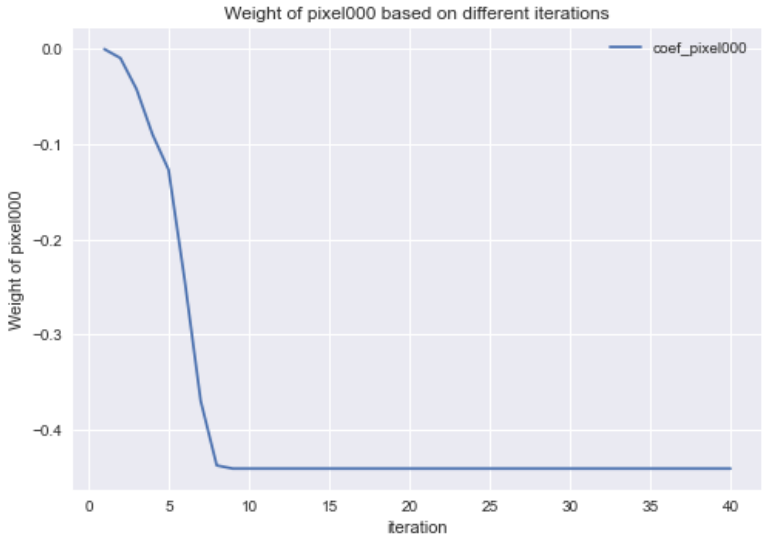


Fig.3

We can observe that the weight of pixel000 start from initial value 0, and then continue to decrease to -0.45 after 8 iterations. It shows that how the coefficient of pixel000 adjust based on Logistic Regression model.

3.

```
The minimum log loss is: 1.115
The C value based on least loss is: 0.100
The accuracy of the model is: 0.968

Confusion Matrix:

Predicted 0     1
True
0       [[942  32]
1        [ 34 975]]
```

4.

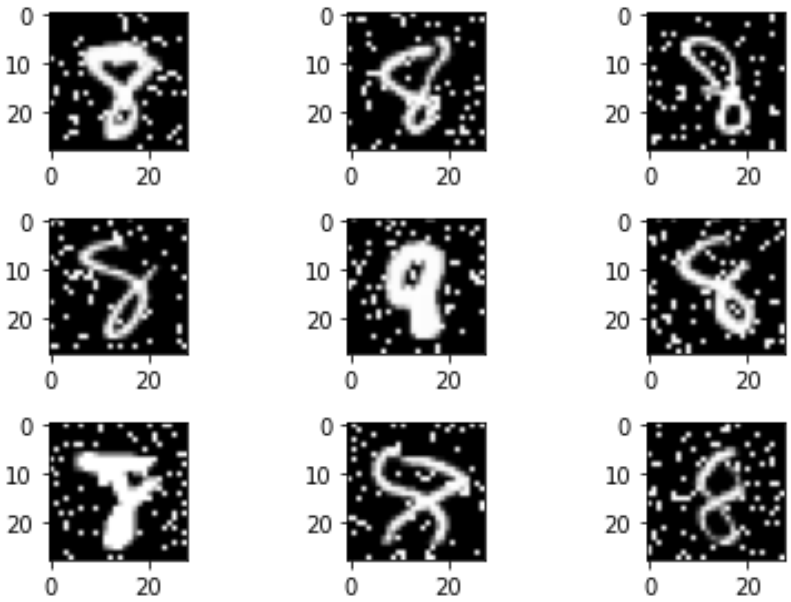


Fig.4 False Positive Images (misclassify as 9)

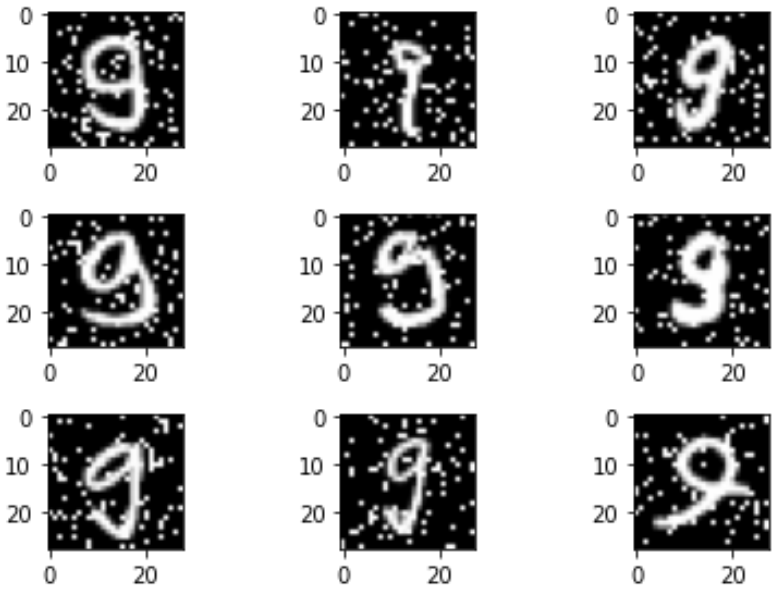


Fig.5 False Negative Images (misclassify as 8)

From Fig.4, we can see some common features that misclassify as 9. For example, the

lower “o” in the number “8” have the small part, which could be one of the reason for the model make the mistake. Also, the left second image in row 3 have the uncomplete curve, and the last image in row 3 have thin stroke have the negative effect to classify the right number.

According to Fig.5, these misclassified samples have high curve tail. From human eyes, we can easily identify it is nine since there is gap between the “o” and the tail, but for the model, it may detect the curve and then classify as “8”.

5.

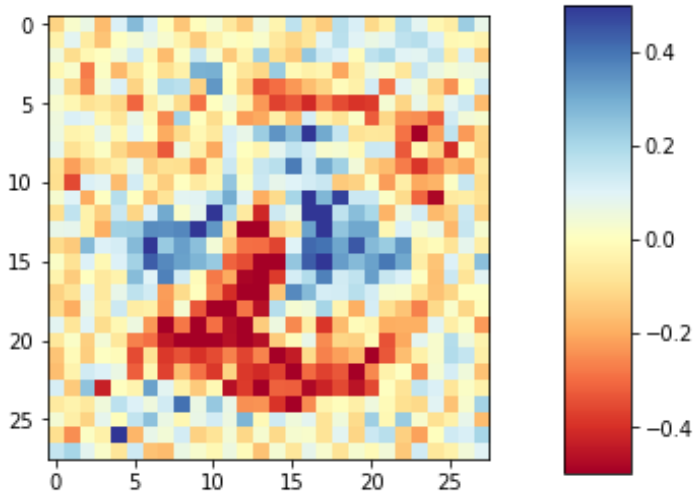


*Fig.6 Weight coefficients in 28*28 matrix*

In Fig.6, we can see the weight corresponding to every pixel, the colormap start from negative(red) to positive(blue). We assign negative to the number “8”, the positive to number “9”, thus, the red parts in Fig.6 represent the main feature of “8”, the blue parts represent the number of “9”. The darker the color, the greater the absolute value. In other words, these dark color pixels are the area that model classification reference condition.

Part Two: Shirt Classification

1. Baseline model

After importing the data and necessary libraries, I use the given files “troudess_train _x.csv” and “troudress_train_y.csv” and do the train test splitting (7:3). I build the basic Logistic Regression Model with default setting except using ‘liblinear’ solver. The accuracy on the training set is 0.972 and the logarithmic loss is 2.197. And the confusion matrix displays as below: TN: 1712, TP: 1659, FN: 110, FP: 119. The AUC is 0.978.

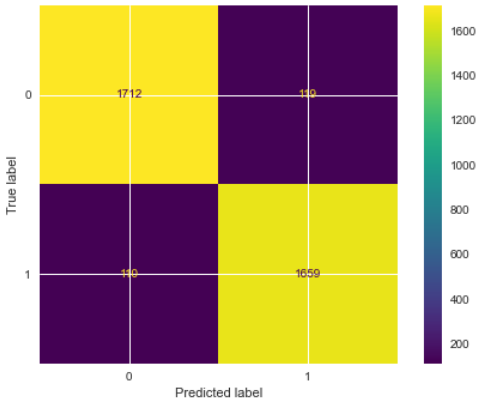


Fig.1 Confusion Matrix of basic model

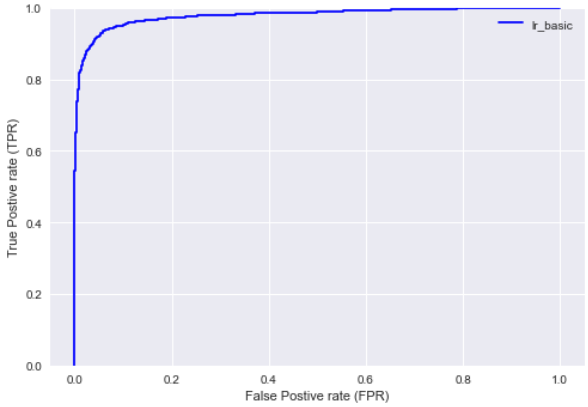


Fig.2 ROC Curve of basic model

2. Tuning the penalty parameter

In this project, I choose to tune the L1 penalty C. This dataset includes 28*28 pixels (784 features). When I plot 9 samples, I found these images are blurred and nearly half of the image is not helpful to the determine whether it is a shirt or not. So, I use L1 penalty, which can effectively help us to do the features selection and exclude these pixels.

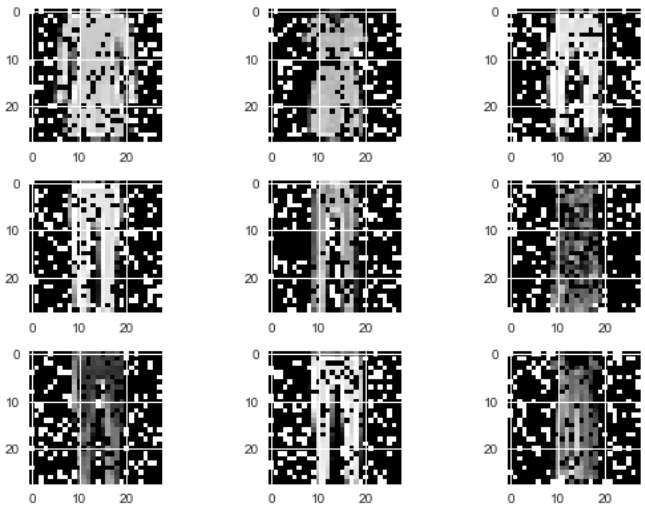


Fig.3 Samples of Fashion MNIST dataset

I assign a list of C using `np.logspace(-9, 6, 31)`, tune the model and try to find the best L1 penalty parameter.

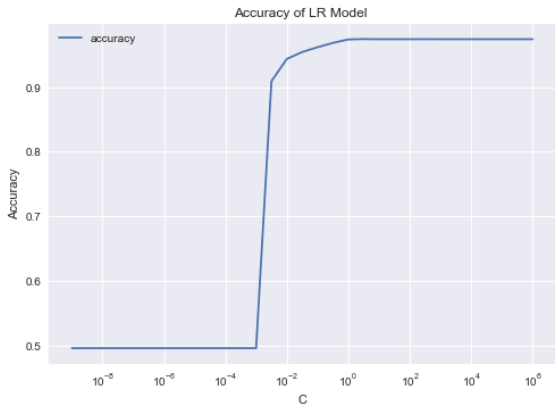


Fig.4 Accuracy based on penalty parameter

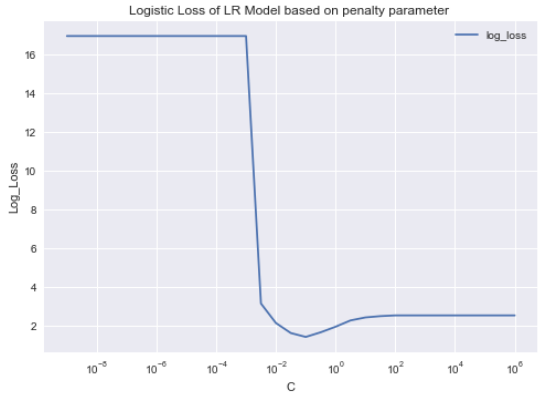


Fig.5 Log loss based on penalty parameter

As the figures show, when C increases, the accuracy also increases and log loss decreases. The best accuracy is 0.904, the C value under best accuracy is 10000. The minimum log loss is 4.816, the C value under minimum log loss is 0.316. Using the tuning result, I plot the confusion matrix again: TN: 1516, TP: 1582, FN: 197, FP: 305. The ROC curves after tuning, as the figure shows, have slightly improve compare with the basic model. The AUC is 0.936.

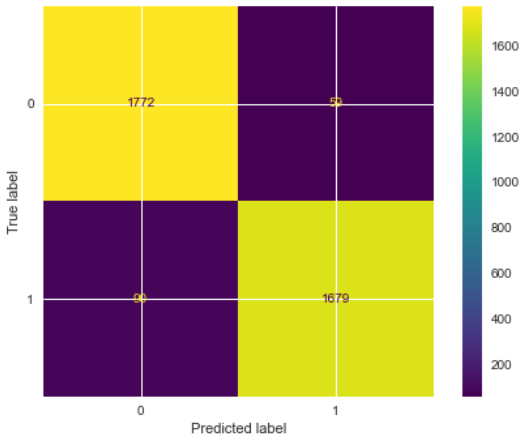


Fig.6 Confusion Matrix after tuning

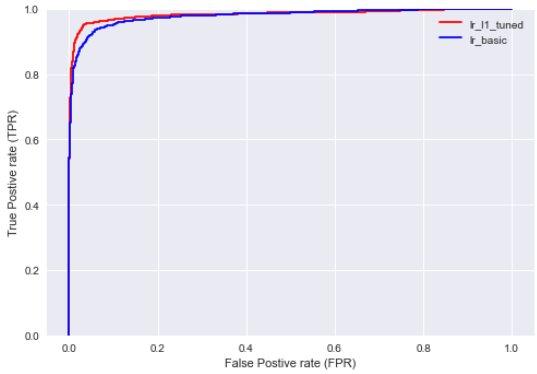


Fig.7 ROC Curve

3. Features Transformation: Explore Data Augmentation

Even the model I use has pretty good performance, in this project, I decide to explore data augmentation via flipping the image horizontally. For example, I flip the image as Fig.8 shown. Before augmentation, the given the data size for training data is 8400*784, testing data is 3600*784. After flipping the image and add them to the original dataset, the size of training data and testing data are double, which are 16800*784 and 7200*784 separately.

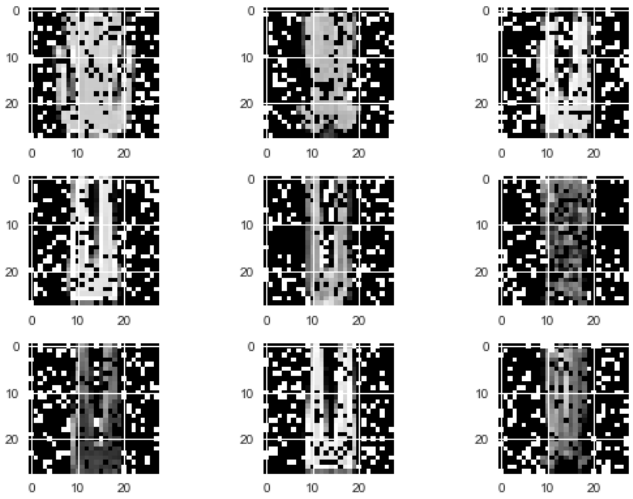


Fig.8 Samples of Flipping Fashion MNIST dataset

Next, I follow the same process of building and tuning the model as the original dataset. I build Logistic Regression Model on these augmentation data. However, the performance of the model is even worse than the original. Before tuning the penalty parameter C, the accuracy on the training set is 0.726 and the logarithmic loss is 10.093. And the confusion matrix displays as below: TN: 2623, TP: 2473, FN: 1065, FP: 1039. The AUC is 0.737.

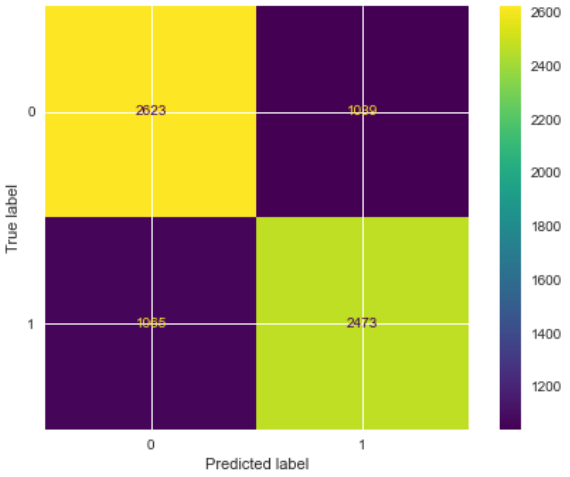


Fig.9 Confusion Matrix after Data Augmentation

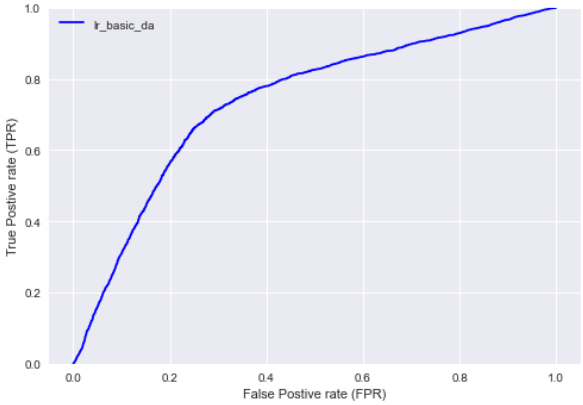


Fig.10 ROC Curve after DA

Following above tunning process, setting penalty parameter C `np.logspace(-9, 6, 31)` and record the change of accuracy and logarithmic loss.

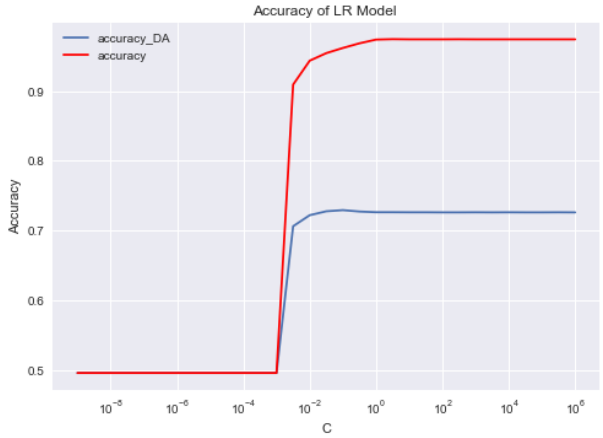


Fig.11 Accuracy after DA

Fig.12 Log Loss after DA

The pattern of line in accuracy and log loss are similar to the pattern before data augmentation though the maximum and minimum value are different. The minimum log loss after data augmentation is 9.335 and the corresponding value is 0.0316. In Fig.13, the red line represents the ROC curve before data augmentation and the blue line represents after data augmentation. The AUC before data augmentation is 0.983 and the AUC is 0.747 after data augmentation.

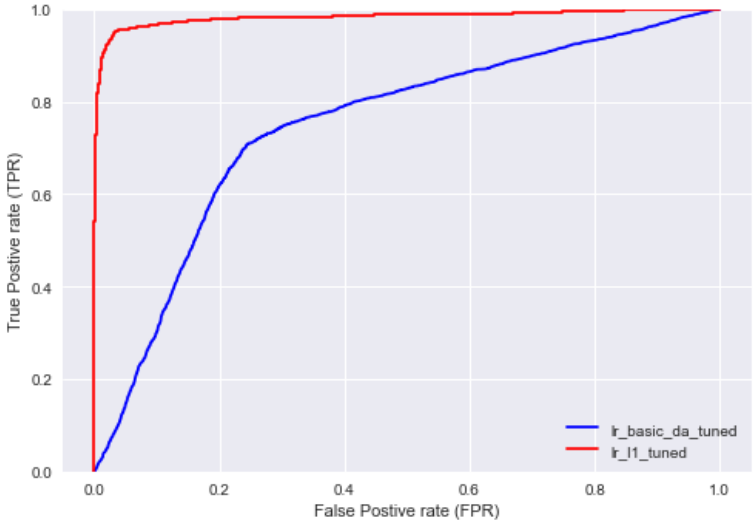


Fig.13 ROC before DA and after DA

It is inevitable to answer the question why we double the training set and have negative effect to the model performance? The model performance depends on not only the size of data but also the quality. When I flip the images and add them to the original data, it causes some confusion to the model and make it hard to differentiate whether it is a shirt. However, flipping the images can be helpful to the robustness of the model. For example, if it comes up with a flip image, the model after data augmentation may have higher accuracy to classify to the correct category.