3F8: Inference
Short Lab Report

Chia Jing Heng

March 4, 2020

Abstract

This lab aims to implement and apply a classifier to a simple dataset. A logistic classifier was first considered and its performance was evaluated based on the fraction of true negatives, false positives, false negatives and true positives detected. A non-linear feature expansion was then considered to improve the performance of the classifier. An RBF expansion with width $l = 1$ was able to improve the fraction of true negatives and true positives detected.

1 Introduction

Logistic regression is very useful for binary classification and it gives soft decisions instead of hard boundaries. However, its performance is very limited to classifying linear data. This is because the prediction probability contours it generates are linear and thus making a logistic classifier less suitable for classifying data with non-linear decision boundaries. One way to curb this is to introduce some non-linear feature expansion on the input data. This enables the classifier to give non-linear decision boundaries. This report outlines the methods used for a regular logistic classifier, the results and analysis on the results. Next, the methods and results for the RBF feature expansion are then outlined. The results are then compared against the regular classfier.

2 Exercise a)

In this exercise we have to consider the logistic classification model (aka logistic regression) and derive the gradients of the log-likelihood given a vector of binary labels $\mathbf{y}$ and a matrix of input features $\mathbf{X}$. The gradient of the log-likelihood can be written as

$$\frac{\partial \mathcal{L}(\beta)}{\partial \beta} = \sum_{n=1}^{N} (y^{(n)} - \sigma(\beta^T \tilde{x}^{(n)}))\tilde{x}_i^{(n)}.$$

3 Exercise b)

In this exercise we are asked to write pseudocode to estimate the parameters β using gradient ascent of the log-likelihood. Our code should be vectorised. The pseudocode to estimate the parameters β is shown below:

```
Function estimate_parameters:

   Input:  feature matrix X, labels y
   Output: vector of coefficients w

   Code:
      eta = 0.001 #learning rate
      w = random()
      sigmoid = (logistic(dot(X_tilde, w)))
      w = w + eta*(dot(transpose(X_tilde), (y - sigmoid_value)))

      return w
```

The learning rate parameter η is chosen to be 0.001. This value was obtained through experimentation to ensure the likelihood converges within a reasonable number of steps.

4 Exercise c)

In this exercise we visualise the dataset in the two-dimensional input space displaying each datapoint's class label. The dataset is visualised in Figure 1. By analysing Figure1 we conclude that a linear classifier is not suitable for this dataset since the class boundaries are not linear.

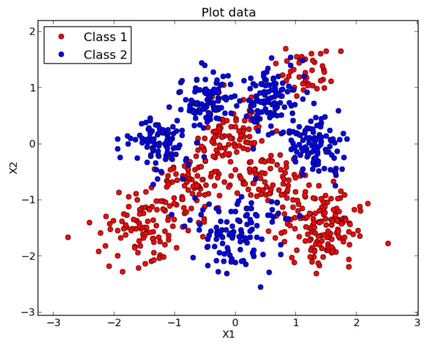

Figure 1: Visualisation of the data.

5 Exercise d)

In this exercise we split the data randomly into training and test sets with 800 and 200 data points, respectively. The pseudo code from exercise a) is transformed into python code as follows:

```
def fit_w(X_tilde_train, y_train, X_tilde_test, y_test, n_steps, alpha):
    w = np.random.randn(X_tilde_train.shape[ 1 ]) #(3 x 1)
    ll_train = np.zeros(n_steps)
    ll_test = np.zeros(n_steps)
    for i in range(n_steps):
        sigmoid_value = predict(X_tilde_train, w)

        w = w + alpha*(np.dot(np.transpose(X_tilde_train), (y_train - sigmoid_value)))

        ll_train[ i ] = compute_average_ll(X_tilde_train, y_train, w)
        ll_test[ i ] = compute_average_ll(X_tilde_test, y_test, w)
        print(ll_train[ i ], ll_test[ i ])

    return w, ll_train, ll_test
```

We then train the classifier using this code. We fixed the learning rate parameter to be $\eta = 0.001$. The average log-likelihood on the training and test sets as the optimisation proceeds are shown in Figure 2. By looking at these plots we conclude that the log-likelihoods are increased with each step, converging to a maximum value. This shows that the classifier was trained appopriately.

Figure 2 displays the visualisation of the contours of the class predictive probabilities on top of the data. This figure shows that the contours are linear and do not capture the classification of the data effectively, as expected.

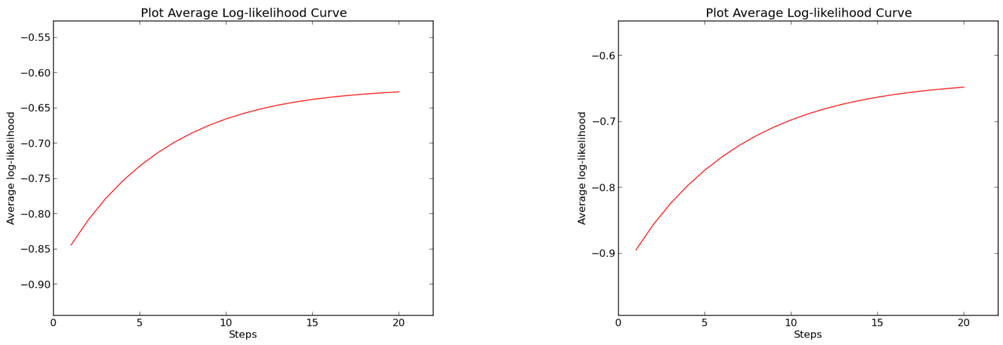

Figure 2: Learning curves showing the average log-likelihood on the training (left) and test (right) data sets.

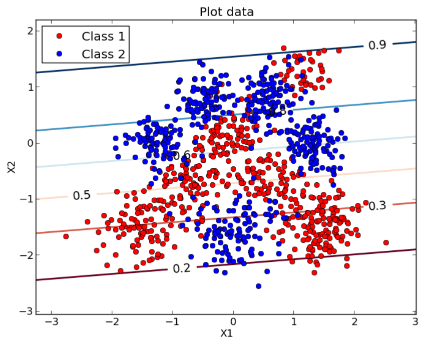

Figure 3: Visualisation of the contours of the class predictive probabilities.

6 Exercise e)

The final average training and test log-likelihoods are shown in Table 1. These results indicate that the model did not overfit, which is desirable. The 2x2 confusion matrices on the and test set is shown in Table 2. By analysing this table, we conclude that the classifier, despite having linear boundaries, was still able to classify the data to a reasonable accuracy.

7 Exercise f)

We now expand the inputs through a set of Gaussian radial basis functions centred on the training data points. We consider widths $l = \{0.01, 0.1, 1\}$ for the basis functions. We fix the learning rate parameter to

Avg. Train ll	**Avg. Test ll**
-0.62696	-0.64748

Table 1: Average training and test log-likelihoods.

$$\begin{array}{cc|c|c} & & \multicolumn{2}{c}{\hat{y}} \\ & & 0 & 1 \\ \hline y & 0 & 0.722 & 0.278 \\ \hline & 1 & 0.282 & 0.718 \end{array}$$

Table 2: Confusion matrix on the test set.

be $\eta = \{0.01, 0.005, 0.0001\}$ for each $l = \{0.01, 0.1, 1\}$, respectively. Figure 4 displays the visualisation of the contours of the resulting class predictive probabilities on top of the data for each choice of $l = \{0.01, 0.1, 1\}$.

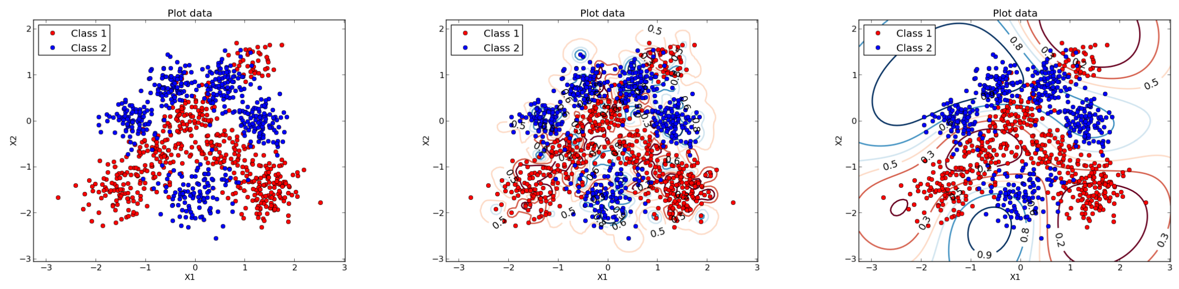

Figure 4: Visualisation of the contours of the class predictive probabilities for $l = 0.01$ (left), $l = 0.1$ (middle), $l = 1$ (right).

8 Exercise g)

The final training and test log-likelihoods per datapoint obtained for each setting of $l = \{0.01, 0.1, 1\}$ are shown in tables 3, 4 and 5. These results indicate that there is overfitting for when $l = 0.01$. The 2×2 confusion matrices for the three models trained with $l = \{0.01, 0.1, 1\}$ are show in tables 6, 7 and 8. After analysing these matrices, we can say that the performance improves as l increases. At $l = 0.01$, the classifier tends to classify the data as class 0. This is due to the overfitting of the model. This could also be seen from figure 4 for $l = 0.01$. The probability contours are not visible because they are overfitting each data point. This problem goes away as l is increased. When we compare these results to those obtained using the original inputs we conclude that there is improvement in performance. The decision boundaries are also non-linear and more effectively capture the nature of the data.

Avg. Train ll	**Avg. Test ll**
-0.737	-0.693

Table 3: Results for $l = 0.01$

Avg. Train ll	**Avg. Test ll**
-0.535	-0.567

Table 4: Results for $l = 0.1$

Avg. Train ll	**Avg. Test ll**
-0.435	-0.454

Table 5: Results for $l = 1$

$$\begin{array}{cc|c|c} & & \multicolumn{2}{c}{\hat{y}} \\ & & 0 & 1 \\ \hline y & 0 & 0.98 & 0.019 \\ \hline & 1 & 0.939 & 0.061 \end{array}$$

Table 6: Conf. matrix $l = 0.01$.

$$\begin{array}{cc|c|c} & & \multicolumn{2}{c}{\hat{y}} \\ & & 0 & 1 \\ \hline y & 0 & 0.784 & 0.215 \\ \hline & 1 & 0.336 & 0.664 \end{array}$$

Table 7: Conf. matrix $l = 0.1$.

$$\begin{array}{cc|c|c} & & \multicolumn{2}{c}{\hat{y}} \\ & & 0 & 1 \\ \hline y & 0 & 0.8265 & 0.1735 \\ \hline & 1 & 0.1765 & 0.824 \end{array}$$

Table 8: Conf. matrix $l = 1$.

9 Conclusions

The lab investigated a logistic regression model with the weights updated using gradient ascent on the log-likelihood. Without feature engineering, the classifier has a limited performance as the probability contours do not capture the non-linearity of the dataset. RBF feature expansion was used to improve upon the classifier. When $l = 0.01$, there is significant over-fitting. However, this was improved as l increases. At $l = 1$, the classifier produces a greater fraction of true negatives and true positives than without the RBF feature expansion.