3F8 Classification

Harvey Hughes • Emmanuel College
Lab Date: 13/2/19
hh458@cam.ac.uk

February 27, 2019

Abstract

Different classification techniques were used to identify what class previously unseen datapoints belong to. This classification was done by training the model on a training data set. It was shown that η the learning rate must be carefully chosen for convergence to occur. A linear classifier performs badly so RBF functions were used to improve accuracy. The width of these functions was found to be very important to avoid overfitting and extrapolation and should be in the range 0.1-1.

I. Introduction

During the course of this lab several classification techniques were used to identify the two classes in the dataset shown in figure 1. The performance of each classifier is measured by testing on an unseen test dataset once training has been completed. The classifiers used were a linear logistic function and non linear radial basis functions.

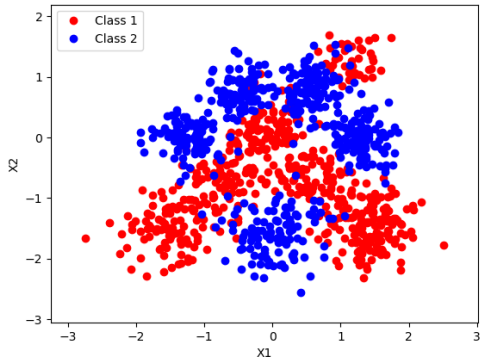


Figure 1: *The datasets to be classified*

II. 1. Preparation exercises

The classification aims to maximise the likelihood of the data by optimising the weights, β. For this the log-likelihood: $\mathcal{L}(\beta) = logP(y|X,\beta))$ is maximised. This is a non trivial optimisation problem therefore gradient ascent is used. This involves an initial guess at β values then updating in the direction of increasing log-likelihood. See equations 1 and 2 for the form of this process.

$$\beta^{(new)} = \beta^{(old)} + \eta \frac{\partial \mathcal{L}}{\partial \beta}\Big|_{\beta^{(old)}} \tag{1}$$

Where η is a hyper-parameter called the leaning rate. A low value for η is likely to cause slow convergence to the maxima, however a high value is often worse as it leads to oscillations about the maxima or never reaching it. Therefore, η needs to be chosen carefully often by trialling a few values.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \beta} &= \sum_{n=1}^{N} y_n(1-\sigma(\beta^T\tilde{x}^{(n)}))\tilde{x}^{(n)} \\ \sigma(\beta^T\tilde{x}^{(n)}) &= \frac{1}{1+exp(-\beta^T\tilde{x}^{(n)})} \end{aligned} \tag{2}$$

$\tilde{x}^{(n)}$ in the linear classifier is equal to ($x^{(n)}$,1) in order to handle biases, therefore β is a (3,1) vector of weights. The implementation of this gradient ascent method used a vectorised form of the parameters to increase speed. This can be seen in appendix 1.1.

III. 2. Linear Model

Initially a linear model for separating the data groups was investigated. The data groups aren't in distinct groups that can be separated by one linear line see figure 1 therefore a linear model wasn't expected to be very successful.

i. Training

Using 80% of the available data the linear model was trained using gradient ascent. 100 iterations were run with $\eta = 0.001$, the convergence to maximum log-likelihood can be seen in figure 2. After training $\beta = (-0.108, 0.904, 0.338)^T$

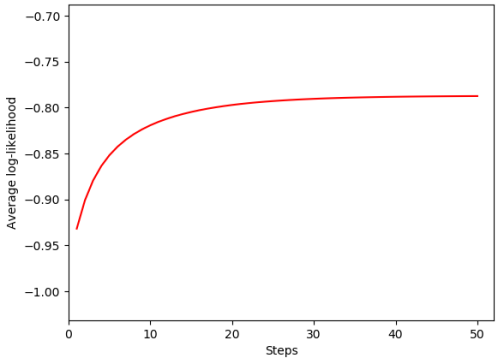


Figure 2: *Log-Likelihood being maximised for the linear model*

ii. Probability contours

In order to visualise the classification process probability contours can be drawn on the data set. These indicate the probability of a certain (x_1, x_2) residing in the blue class. Figure 3 shows these contours. It's easily seen here that the accuracy of a linear classifier won't be too high on this data set due the two classes being non linearly separable by the central linear contour.

iii. Confusion Matrix

To analyse the accuracy of classification more precisely a confusion matrix can be calculated. The confusion matrix is calculated using the

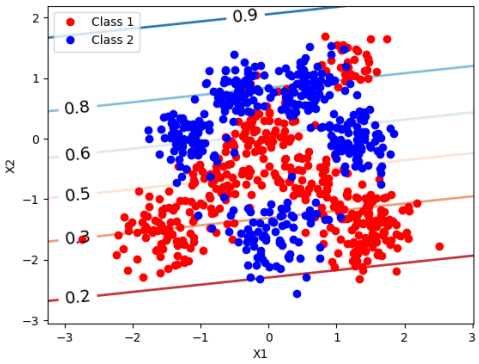


Figure 3: *Probability contours of residing in the blue class for the linear model*

unseen test set using the optimum weights previously found. The on diagonal terms represent correctly classifying the data point, and off diagonal terms are incorrect classifications. The code for computing $\mathcal{C}$ is shown in the appendix 1.2.

$$\mathcal{C} = \begin{bmatrix} 0.35 & 0.155 \\ 0.18 & 0.315 \end{bmatrix}$$

The confusion matrix shows that 66.5% of the test data was correctly classified.

IV. 3. Radial basis functions

To improve classification accuracy non-linear techniques need to be used. This is done through the use of radial basis functions shown in equation 3.

$$\tilde{x}^{(n)}_{m+1} = exp(\frac{-1}{2l^2}\sum_{d=1}^{2}(x^{(n)}_d - x^{(m)}_d)^2) \quad (3)$$

The result of this is extra inputs in $\tilde{x}^{(n)}$ which are 2D quadratics (radial basis functions) centred on each training data point with a certain width l. These extra inputs allow the model to classify using non linear regions meaning separate clusters can be identified therefore reducing the error. The widths being tests are l =[1, 0.1, 0.01].

i. Training

Once again 80% of the data was used to train on in order to find the optimum value for the extended (800,1) weight vector. Figures 4 - 6 show the training progress across iterations.

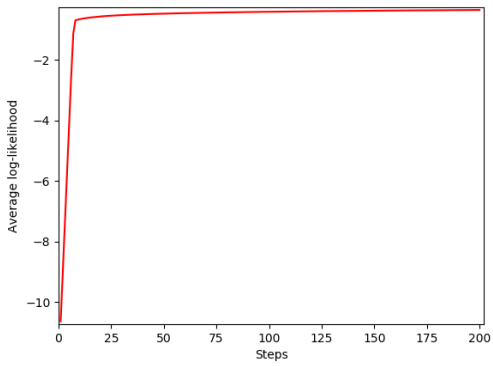


Figure 4: *Log-Likelihood being maximised when l=1*

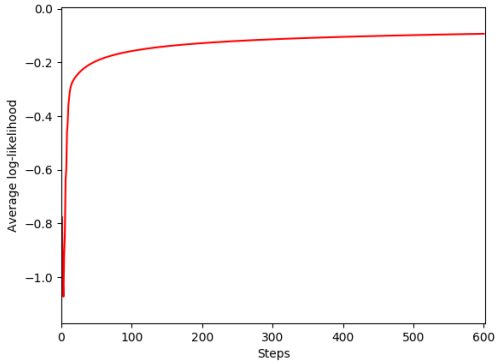


Figure 5: *Log-Likelihood being maximised when l=0.1*

Learning rates of η =[0.0001, 0.015, 0.011] for width's of l =[1, 0.1, 0.01] were used. Smaller widths were more sensitive to oscillations about the maxima causing divergence. Convergence for the radial basis functions was slower than for a linear model, hence a smaller magnitude starting weight vector and more iterations were used. The likelihood plots for l=1 and 0.1 show that a maxima has been reached at the end of training. However in order to minimise over fitting and shorten training times the maxima wasn't reached when l=0.001. The log-likelihood on the training set with small widths can reach values close to 0, however

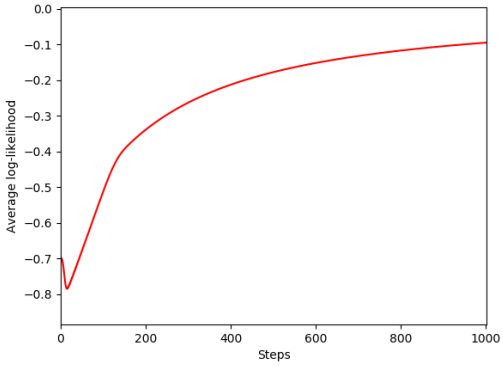


Figure 6: *Log-Likelihood being maximised when l = 0.01*

this is because the weight vector's magnitude becomes very large as it learns almost exactly the training set. This exact learning reduces generality and success on an unseen test set.

ii. Probability contours

Similarly to the linear case probability contours can be drawn for the three widths of RBF classifiers. These are shown in figures 7 - 9.

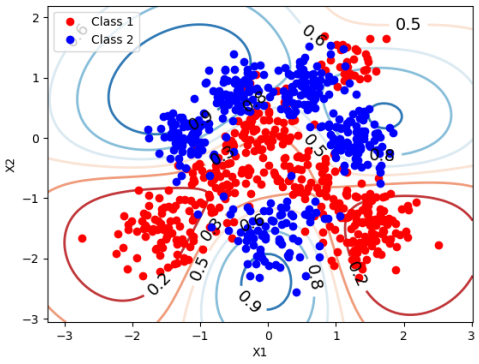


Figure 7: *Probability contours of residing in the blue class for the RBF model of width 1*

figures 7 - 9 show that the width of RBF function is directly correlated to the proximity of the probability contours to the training data. With a large width of 1 the classes are separated well by the contours. However, the contours include large areas where no data was measured. This extrapolation could result in incorrect classification if the test data introduces a new cluster in a previously unseen region

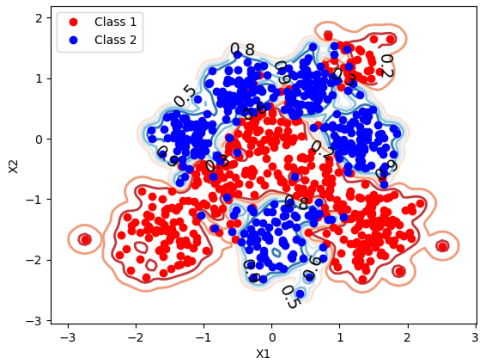


Figure 8: *Probability contours of residing in the blue class for the RBF model of width 0.1*

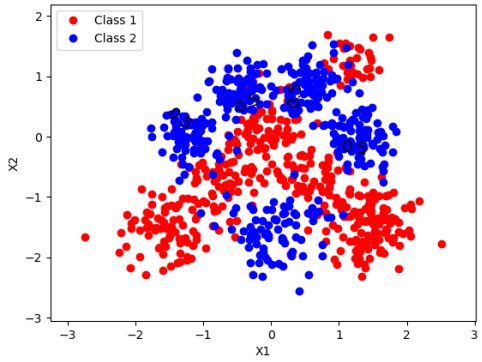


Figure 9: *Probability contours of residing in the blue class for the RBF model of width 0.01*

of the space (x_1, x_2), a better estimate for unseen regions is a probability nearer to 0.5. The adverse is true for small widths, the training has learnt where all the training points are and drawn small bubbles around them shown in figure 10. This causes over fitting and means classification of new points is likely to be no better than a coin toss unless a test data point lies very close to a training point of the same class. Overfitting also means any noise in the inputs will be likely to cause an incorrect classification. For this data the optimum width is between 0.1 and 1.

iii. Confusion Matrix

Confusion matrices can once again be calculated for the three RBF models in order to quantify performance.

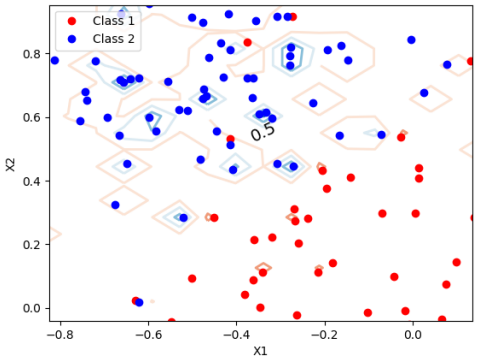


Figure 10: *Zoomed in probability contours of residing in the blue class for the RBF model of width 0.01*

$$\mathcal{C}_1 = \begin{bmatrix} 0.42 & 0.085 \\ 0.05 & 0.445 \end{bmatrix}$$

$$\mathcal{C}_{0.1} = \begin{bmatrix} 0.435 & 0.07 \\ 0.05 & 0.445 \end{bmatrix}$$

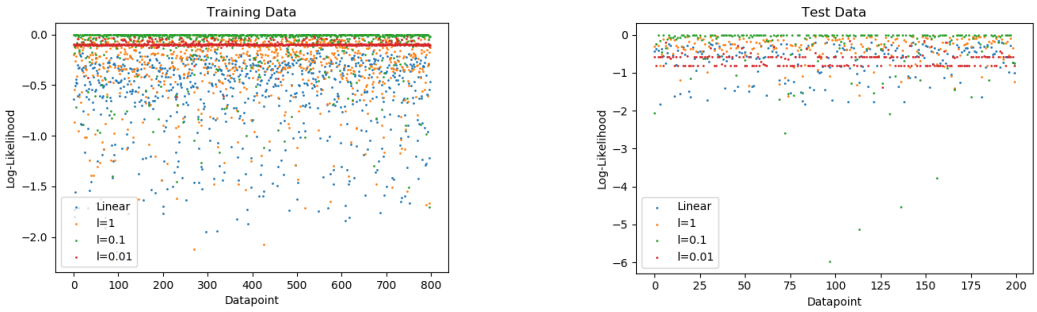
$$\mathcal{C}_{0.01} = \begin{bmatrix} 0.495 & 0.01 \\ 0.45 & 0.045 \end{bmatrix}$$

By looking at $\mathcal{C}_{0.01}$ it's shown that 94.5% of test data is classified as class 1 (red). This is a result of the over fitting and causes very low success probability on the unseen test set. The classification becomes like a coin toss and is incorrect 46% of the time. Both $\mathcal{C}_{0.1}$ and $\mathcal{C}_1$ show a high success rate of 88% and 86.5%. This is a large improvement on the linear classifier.

iv. Log Likelihood per data point

Additional comparisons between the classifiers can be done by looking at the log-likelihood of each data point for the training and test sets. Figure 11 show these log-likelihoods on a scatter plot in order to gain a sense of magnitudes, the positions of outliers can be seen on figures 12 and 13.

Figure 11 shows that all points in the training set for l=0.01 reach an almost constant likelihood. This is a sign of overfitting. For the test set the likelihood with l-0.01 is one of two values corresponding to the two classes, the likelihood is on general lower than the other classifiers. When l=0.1 the likelihood is lowest



(a) *Training data* **(b)** *Test Data*

Figure 11: *Log Likelihood per point for each classifier*

for the training data however large outliers appear on the test set due to data points in the test set residing slightly too far away from the observed training set. Both the linear classifier and RBF with l=1 show quite noisy log-likelihoods possibly due to the larger sizes of region created.

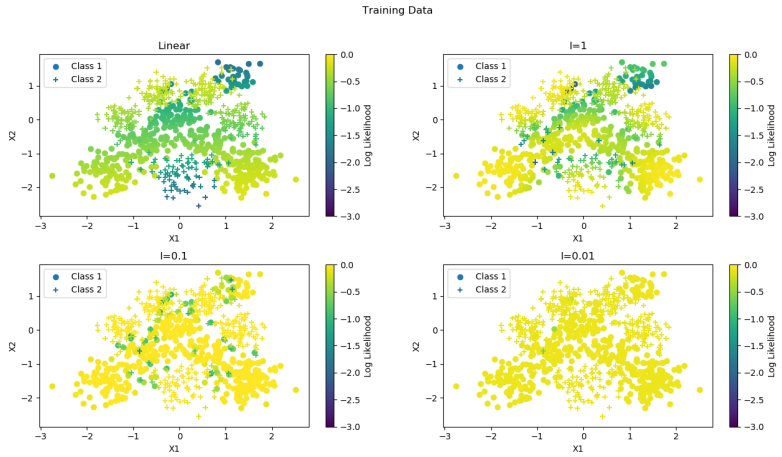


Figure 12: *Gradient plot showing the training log-likelihood for each classifier with the position in the data set*

The same patterns can be seen by looking at the log-likelihoods with position. The RBF classifier with l=0.001 is a constant colour in the training set and two shades in the test set. The outliers when l=0.1 are shown to be on the edges of clusters of one group or when a data point of one class sits amongst lots of the other class. The more random nature of the linear and RBF with l=1 is shown to be due to certain groups in the (x_1, x_2) space. Low likelihoods are assigned to any group on the incorrect side of the linear divide for the linear classifier. For l=0.1 these low likelihoods come from boundaries between cluster groups in particular the class 1 group at the top right, which in figure 7 has class 2 probability contours running through.

V. Conclusion

- Care must be taken when choosing η in order for the gradient ascent algorithm to converge.
- RBF classifiers perform far better than linear classifiers on datasets with intermingled classes.
- Using a small width of RBF function results in overfitting the data and a classifier that performs no better then a coin toss.

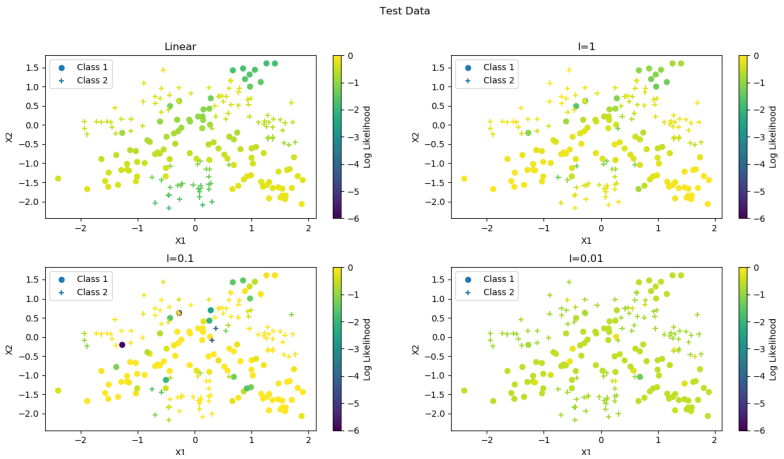


Figure 13: *Gradient plot showing the test log-likelihood for each classifier with the position in the data set*

- A large width results in extrapolating the data outside of observed regions giving high probability of certain classes being present where little information is known.

VI. Appendix

1.1 Training Data

```
def gradient_ascent(beta,x_tilde,y):
    sigma = 1/(1+np.exp(-np.dot(x_tilde, beta))) # creates a vector of sigma values
    gradient=np.matmul((y-sigma.T),x_tilde).T   creates a gradient vector
    return beta + lrate*gradient

def train_data(X,y,beta,n):
    aveLike=np.zeros(n)
    for i in range(n):
        beta = gradient_ascent(beta,X,y)
        aveLike[i]=compute_average_ll(X,y,beta)
    plot_ll(aveLike)
    return beta
```

1.2 Computing Confusion Matrix

```
def flip_numbers(x):
    return -x+1

def compute_confusion(testxtilde,testy,w):
    test_prob = logistic(np.dot(testxtilde, w))
    ytilde = test_prob //0.5
    confusion = np.array([[0,0],[0,0]])
    confusion[0,0] = np.dot(flip_numbers(testy),flip_numbers(ytilde)) # guess = 0
     and value is 0
    confusion[0,1] = np.dot(flip_numbers(testy),ytilde) #guess = 1 , true = o
    confusion[1,0] = np.dot(testy,flip_numbers(ytilde))# guess = 0 , true value = 1
    confusion[1,1] = np.dot(testy,ytilde) #guess = 1 and true value = 1
    confusion = confusion / testy.shape[0]
    return confusion
```