Chapter 4

Logistic Regression

Classification and linear separability. Normalization and standardization. The logistic regression classifier. Linear separability and dimensionality

4.1 Linear separability

In two dimensions, a pair of sets of points is *linearly separable* if there exists a line that can separate the two sets. Figure 4.3 shows two pairs of sets of points. These are the plots of the activity of gene pairs in tumour (red) and normal tissue (blue). The data is from [1]. In the first panel, the sets are *linearly separable*, as shown by the line dividing them (the *decision boundary*). In the second panel they are not *linearly separable*, as there is no straight line that can divide the two sets.

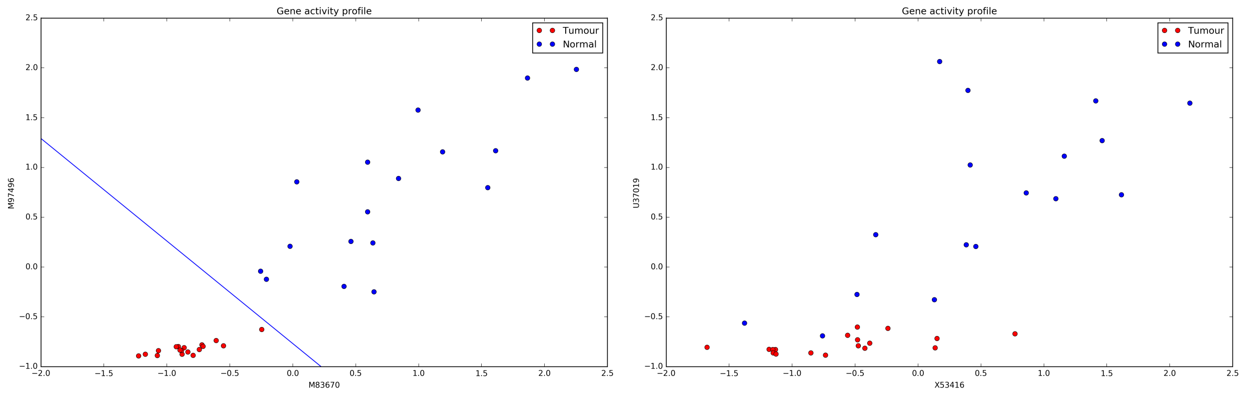

Figure 4.1: The left panel shows a linearly separable pair of point sets. The pair of sets shown on the right is not linearly separable.

In the previous chapters we saw that we can define a straight line with two parameters, in two dimensions. However, for classification we also need to distinguish between the two sides of the line, since we want to separate sets of points with the line as a frontier. We can do this by defining the line with a vector. In this way, the line will consist of all vectors perpendicular to our chosen vector and the vector will indicate a "positive" side of the line. Generalizing for N dimensions, given a vector $\vec{w}$ perpendicular to the desired plane and a constant w_0, the points belonging to the plane can be found by this equation:

$$\vec{w}^T \vec{x} + w_0 = 0$$

Figure 4.2 shows a plane separating two sets of points and the corresponding $\vec{w}$, plotted on the plane. Note that the different axis scales make it appear that the vector is not perpendicular to the plane (but it is).

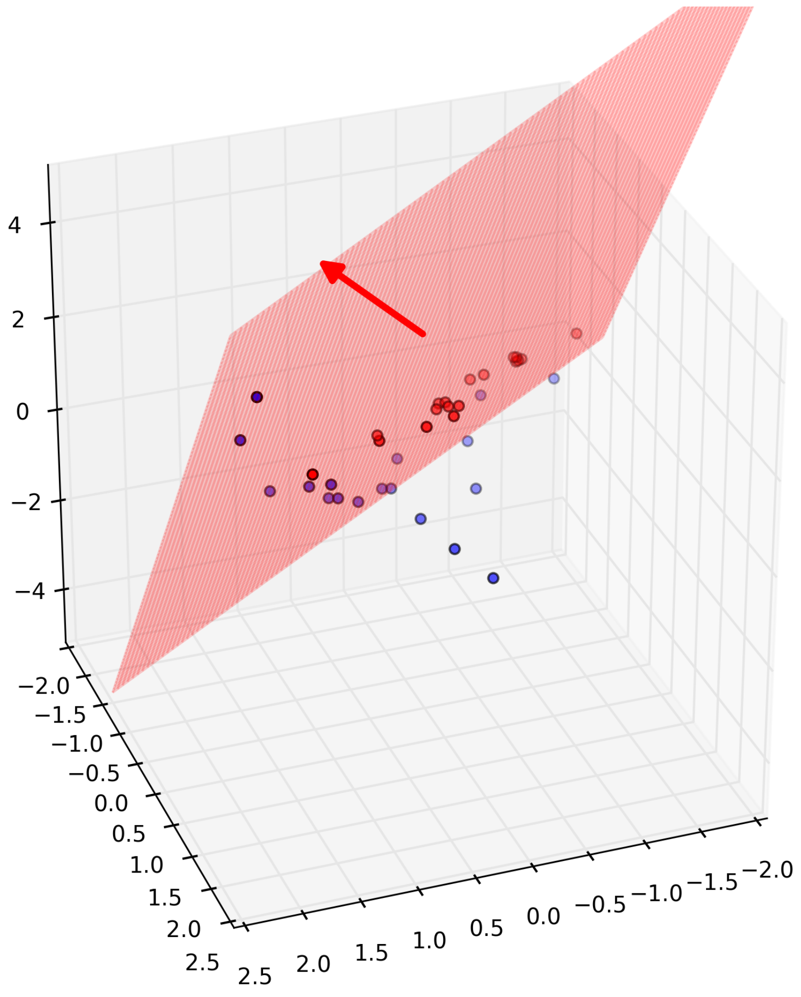

Figure 4.2: A plane separating the two sets of points. The arrow shows the vector normal to the plane (note the distortion due to the different axis scales).

The function

$$y(\vec{x}) = \vec{w}^T\vec{x} + w_0$$

has a positive value on one side of the dividing hyperplane and a negative value on the opposite side. This gives us a way to model a *linear discriminant* separating the two classes. Now we need to find a way to determine the coordinates for $\vec{w}$ and the constant w_0.

4.2 The wrong way: least mean squares

Since the function $y(\vec{x}) = \vec{w}^T\vec{x} + w_0$ is positive on one side of the hyperplane and negative on the other, if we assign a value of 1 to one class and -1 to the other, we can try to find the place of the best dividing hyperplane by minimizing the squared error:

$$E = \sum_{j=1}^{N} (y(\vec{x_j}) - t_j)^2$$

where $\vec{x_j}$ are the points in our training set and t_j the respective class of each point (1 or -1). Note that **this is the wrong way to solve this problem**. However, the result teaches an important lesson about a difference between regression and classification.

To simplify the computation of $y(\vec{x}) = \vec{w}^T\vec{x} + w_0$, we can include w_0 in $\vec{w}$ and simply add a 1 to each $\vec{x}$:

$$\widetilde{w} = (w_0, \vec{w}), \widetilde{x} = (1, \vec{x}), y(\vec{x}) = \widetilde{w}^T\widetilde{x}$$

We will use the data set shown in Figure 4.3, which shows a plot of the activity of the guanylate cyclase activator 2A gene (M97496) versus the activity of the carbonic anhydrase IV gene (M83670) in normal cells (blue) and tumour cells (red) [1]. Our goal is to find the best frontier between these two *linearly separable* sets.

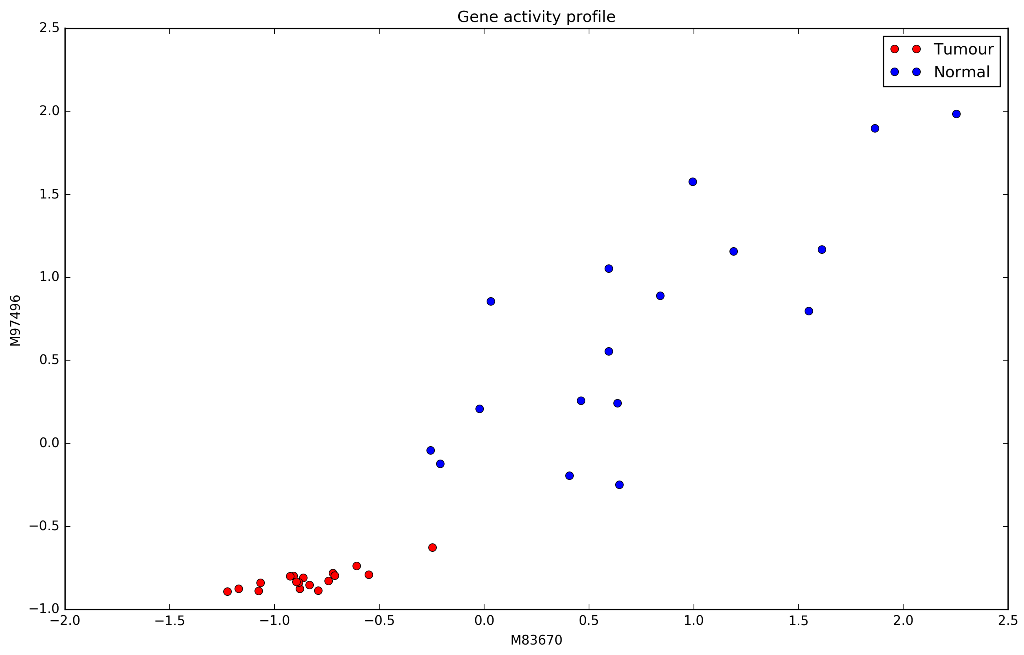

Figure 4.3: Data set for the two classes, tumour and normal, plotting the activity of genes M83670 and M97496 (standardized, see text).

These data is given in a text file, with the activity of each gene and the class, with 0 for normal and 1 for tumour cells:

```
-81     10      1
-30     60      1
...
320   1231      0
172    700      0
...
```

First we read the data and adjust the range of the input values. This is important to bring all values into similar scales in order to prevent the solver from having to deal with different scales. There are two main methods of preprocessing data. *Normalization* is a linear transformation that brings the values of features into the range [0,1]:

$$x_{new} = \frac{x - min(X)}{max(X) - min(X)}$$

where X is the set of all values.

Standardization is a linear transformation that sets the average of the values of features to zero and the standard deviation of these values to 1:

$$x_{new} = \frac{x - \mu(X)}{\sigma(X)}$$

When using real data, we should always do some preprocessing. In this case, we will *standardize* the data. Note that in real applications we must retain these values because we need to process any future examples in exactly the same way as we process the training set, using the same scaling factors.

```python
import numpy as np

mat = np.loadtxt(input_data,delimiter='\t')
Ys = mat[:,[-1]]
Xs = mat[:,:-1]
means = np.mean(Xs,0)
stdevs = np.std(Xs,0)
Xs = (Xs-means)/stdevs
```

Now we create that function that adds the column of ones to the input vector, which originally contains two columns for the gene activity. We add this column of ones at the end instead of at the beginning, but the exact placement is indifferent as long as we remember it.

```python
import numpy as np

def expand_features(X):
    """append a columns of 1
    """
    X_exp = np.ones((X.shape[0],X.shape[1]+1))
    X_exp[:,:-1] = X
    return X_exp
```

Then the function that computes the quadratic error comparing the signed distance to a frontier line and the classes of the examples. Since we want class values of -1 and 1 but the data originally has classes of 0 for normal and 1 for tumour cells, we need to do some simple algebra to convert this in the last line. Otherwise, we simply compute the inner (dot) product of the two vectors and compute the mean squared error from the result.

```python
def quad_cost(theta,X,y):
    """return  error value comparing signed distance with y
       theta is a column of coefficients
       X is a matrix with one example per row, and a 1 in the last column
       y is a vector of classes 0 or 1
    """
    coefs = np.zeros((len(theta),1))
    coefs[:,0] = theta
    vals = np.dot(X,coefs)
    return np.mean((vals-(2*y-1))**2)
```

Now we just add the column of ones and minimize the quadratic error:

```python
from scipy.optimize import minimize
```

```
import matplotlib.pyplot as plt

X_exp = expand_features(Xs)
coefs = np.ones(X_exp.shape[1])
opt = minimize(quad_cost,coefs,(X_exp,Ys),tol=0.00001)
coefs = opt.x
# plot the chart
```

The result, however, is less than ideal, as shown in Figure 4.4. The reason for this is that point farther from the decision line will have a larger distance value. Since we are minimizing the squared error, the learning algorithm will try to reduce the distance between the decision line and the farthest points. This displaces the decision line away from the actual frontier between the two sets, resulting in some points being misclassified.

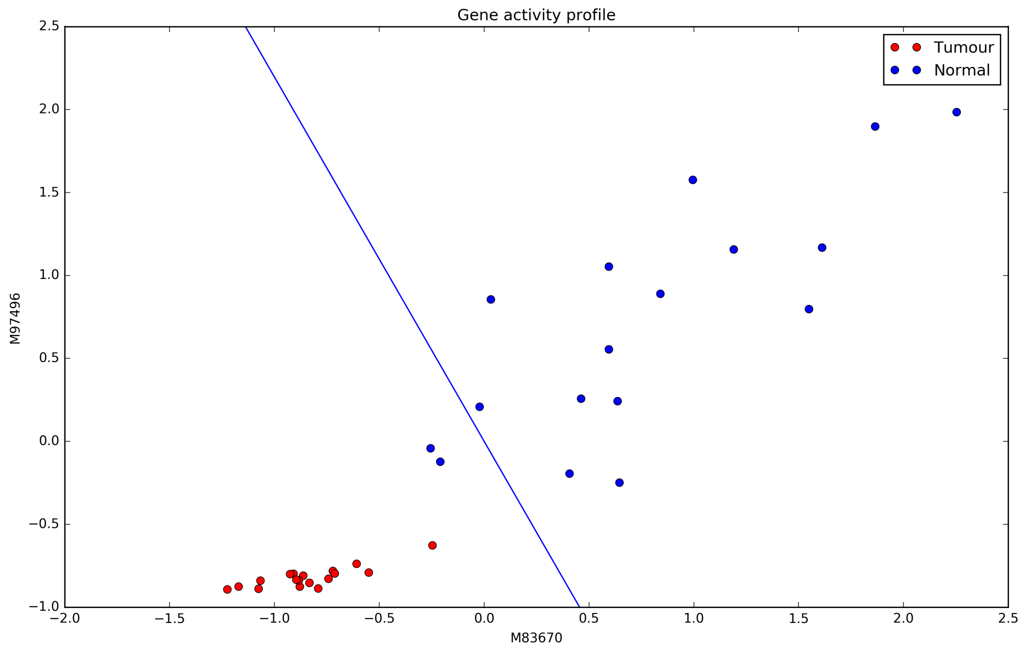

Figure 4.4: Tentative decision line between these two sets computed by minimizing the squared error between the signed distance and the class label. Note how the decision line is incorrectly placed due to the effect of the farthest points.

This is a fundamental difference between regression and classification. In regression problems, it is not desirable to have points distant from the prediction. In classification problems, having points distant from the decision surface (or line, in two dimensions) is not a problem as long as they fall on the correct side of the frontier. So for classification we need a different approach.

4.3 Classification by Logistic Regression

We will be using *logistic regression* as a classifier, with the purpose of obtaining a decision hyperplane that separates different classes. However, *logistic regression* is, at heart, a regression model too, as we will see below.

Let us assume we can find a function of our parameters $\widetilde{w}$[1] and the features $\vec{x}$ that can tell us the

[1]Remember that $\widetilde{w} = (w_0, \vec{w})$.

probability of an example with features $\vec{x}$ belonging to class C_1:

$$g(\vec{x}, \widetilde{w}) = P(C_1|\vec{x})$$

We want to find a decision hyperplane where the probability of finding an example of class C_1 equals the probability of finding an example of class C_0, assuming there are only these two classes.

$$P(C_1|\vec{x}) = P(C_0|\vec{x}) = 1 - P(C_1|\vec{x})$$

which is equivalent to:

$$ln \frac{P(C_1|\vec{x})}{1 - P(C_1|\vec{x})} = 0$$

Thus, we can write

$$ln \frac{P(C_1|\vec{x})}{1 - P(C_1|\vec{x})} = ln \frac{g(\vec{x}, \widetilde{w})}{1 - g(\vec{x}, \widetilde{w})} = \vec{w}^T \vec{x} + w_0$$

Rearranging, we obtain

$$g(\vec{x}, \widetilde{w}) = \frac{1}{1 + e^{-(\vec{w}^T \vec{x} + w_0)}}$$

The function

$$f(x) = \frac{1}{1 + e^{-k(x - x_0)}}$$

is called the *logistic function* and is represented on the right panel of Figure 4.5, for $x_0 = 0$ and $k = 1$.

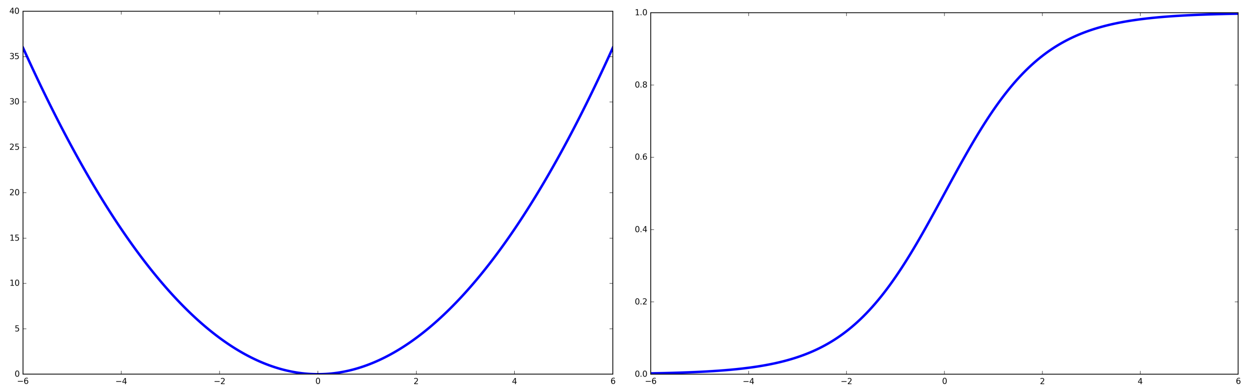

Figure 4.5: Comparing the quadratic and the logistic functions.

This function has the useful feature of varying around a threshold value but being nearly constant away from this threshold. This is what we need to solve the problem we had with the minimization of the squared error, in which the points farther away were having an undesired effect on the decision boundary. Note also that this function attempts to approximate the probability of each point $\vec{x}$ being in class C_1. This is why *logistic regression* is a regression model; when fitting the model we are trying to approximate this continuous probability function. However, because we also choose a cut-off value where we separate the two classes — where $P(C_1|\vec{x}) = P(C_0|\vec{x})$ — we turn this regression model into a classifier. This is a common occurrence in classification, with the classifier model being trained, at bottom, as a regression model. Since the class of hypotheses we are considering in *logistic regression* to separate the classes is the set of all hyperplanes (or planes in 3D and lines in 2D) — *i.e.* linear combinations of the features — this is a linear classifier, when used as a classifier.

Now, given that:

$$g(\vec{x}, \widetilde{w}) = P(C_1|\vec{x})$$

the likelihood of our parameters $\widetilde{w}$, which is the product of the probabilities of the classes of our examples given our hypothesis, will be

$$\mathcal{L}(\widetilde{w}|X) = \prod_{n=1}^{N} \left[g_n^{t_n} (1 - g_n)^{1-t_n} \right]$$

and the logarithm of the likelihood is

$$l(\widetilde{w}|X) = \sum_{n=1}^{N} \left[t_n \ln g_n + (1 - t_n) \ln(1 - g_n) \right]$$

where g_n is the result of $g(\vec{x_n}, \widetilde{w})$ and t_n is the true class of point n, which can be 1 or 0.

Thus, the maximum likelihood solution to this problem is the minimum of this cost function:

$$E(\widetilde{w}) = -\sum_{n=1}^{N} \left[t_n \ln g_n + (1 - t_n) \ln(1 - g_n) \right]$$

with

$$g_n = \frac{1}{1 + e^{-(\vec{w}^T \vec{x_n} + w_0)}}$$

To implement this, we write the logistic and logistic cost functions:

```python
def logistic(X):
    """return logistic function of vector X"""
    den = 1.0 + np.e ** (-1.0 * X)
    return 1.0 / den

def log_cost(theta,X,y):
    """return  logistic error value
      X is a matrix with one example per row, and a 1 in the last column
      y is a vector of classes 0 or 1
    """
    coefs = np.zeros((len(theta),1))
    coefs[:,0] = theta
    sig_vals = logistic(np.dot(X,coefs))
    log_1 = np.log(sig_vals)*y
    log_0 = np.log((1-sig_vals))*(1-y)
    return -np.mean(log_0+log_1)
```

And then minimize this function instead of the quadratic error. The result is much better, as shown in Figure 4.6.

4.4 Linear separability, revisited

If the classes are not *linearly separable*, it is impossible to find a straight line that can separate the two classes. This is illustrated in Figure 4.7, where the standard logistic regression approach of the last section results in a decision boundary that cannot discriminate between the two classes. In this case,

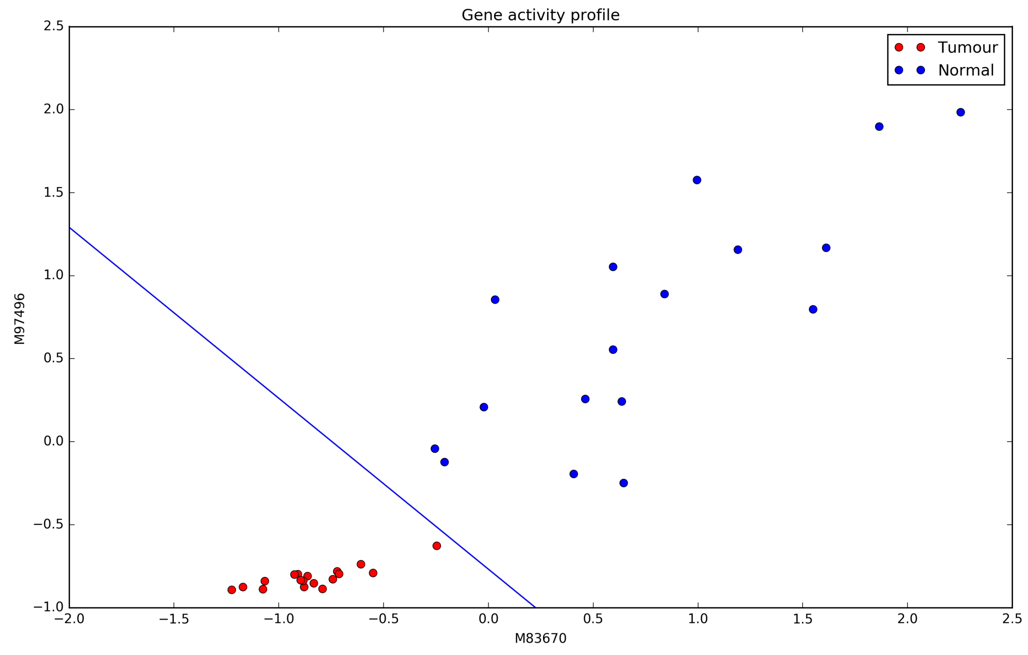

Figure 4.6: Logistic regression on the activity of M83670 and M97496 genes.

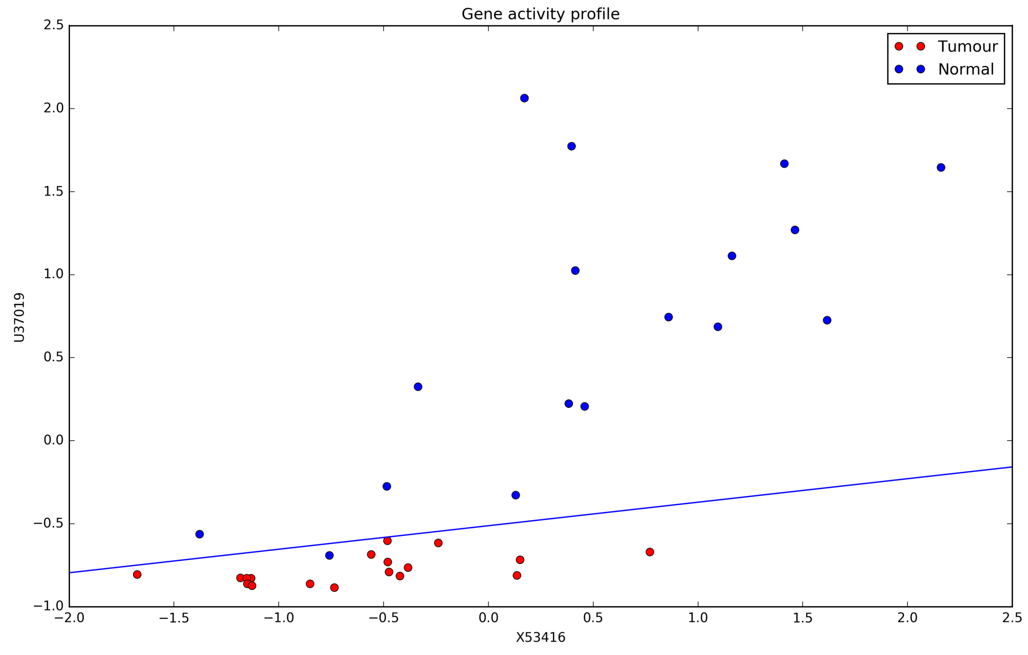

Figure 4.7: Attempting to separate classes that are not linearly separable (activity of X53416 and U37019 genes).

the features are the activity of the actin-binding protein gene (X53416) and the smooth muscle cell calcium binding protein gene (U37019).

However, we can expand our data with an additional feature obtained by (for example) the product of the two original gene activity values. This function is easy to implement.

```
def poly_3features(X):
    """append a column with the product of the two first features
    """
    X_exp = np.zeros((X.shape[0],X.shape[1]+1))
    X_exp[:,:-1] = X
    X_exp[:,-1] = X[:,0]*X[:,1]
    return X_exp
```

Now we load the data, transform it and use the `LogisticRegression` class from the `sklearn.linear_mode` module to find the best plane separating the expanded data.

```
mat = np.loadtxt('gene_data_2.txt',delimiter='\t')
Ys = mat[:,[-1]]
Xs = mat[:,:-1]
means = np.mean(Xs,0)
stdevs = np.std(Xs,0)
Xs = (Xs-means)/stdevs
X_exp = poly_3features(Xs)
reg = LogisticRegression(C=1e12, tol=1e-10)
reg.fit(X_exp,Ys[:,0])
```

Now, our *linear discriminant* will no longer be a line in two dimensions but a plane in three dimensions, corresponding to the expanded data, as shown in Figure 4.8

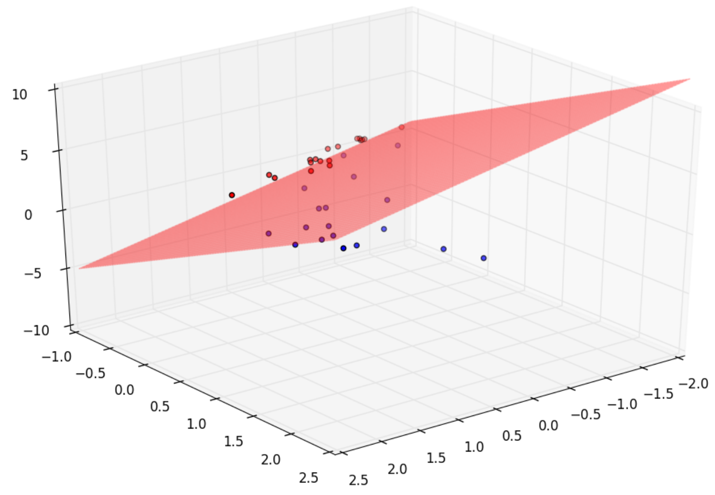

Figure 4.8: Gene activity (X53416 and U37019 genes) data expanded to three dimensions and the decision plane found by logistic regression.

Projecting this decision plane back into the two dimensional space of the original data, we can find the corresponding decision boundary which is no longer a straight line. This is a similar approach to the one we saw with linear regression.

This is still not enough to completely separate the two classes, but we can increase the discrimination power of the classifier by increasing the dimensions used by the logistical regression. For example, using 7 dimensions, as shown in the code below and Figure 4.10.

```
def poly_7features(X):
    """append a five columns with the product,
        square and cube of the first two features
    """
    X_exp = np.zeros((X.shape[0],X.shape[1]+5))
```

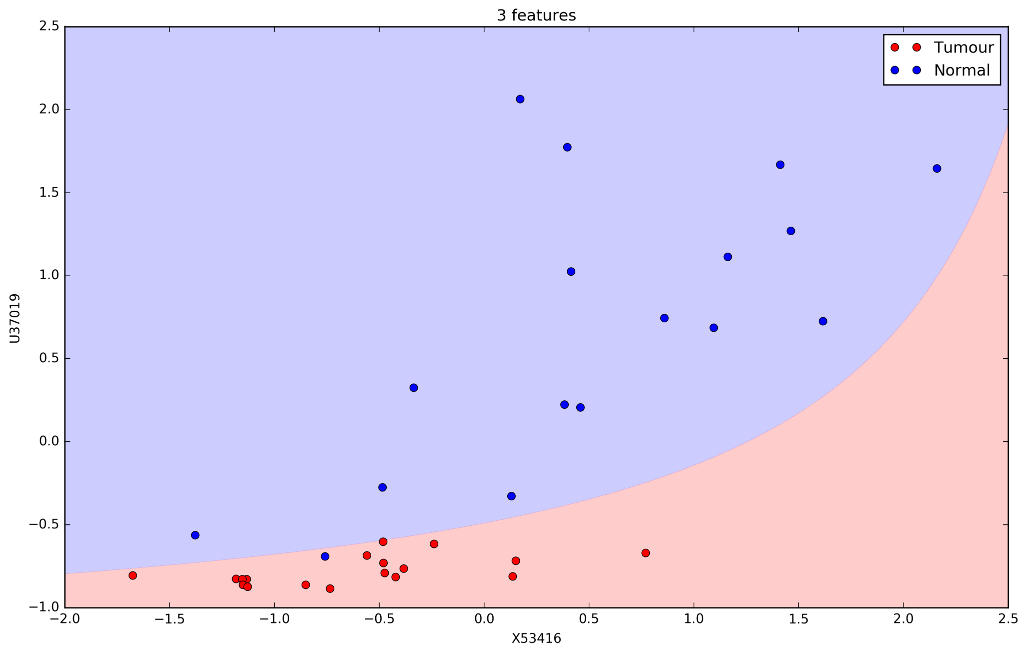

Figure 4.9: Gene activity (X53416 and U37019 genes), decision boundary obtained using a third feature value obtained from the product of the first two.

```
    X_exp[:,:-5] = X
    X_exp[:,-3] = X[:,0]*X[:,1]
    X_exp[:,-2] = X[:,0]**2
    X_exp[:,-1] = X[:,1]**2
    X_exp[:,-5] = X[:,0]**3
    X_exp[:,-4] = X[:,1]**3
    return X_exp
```

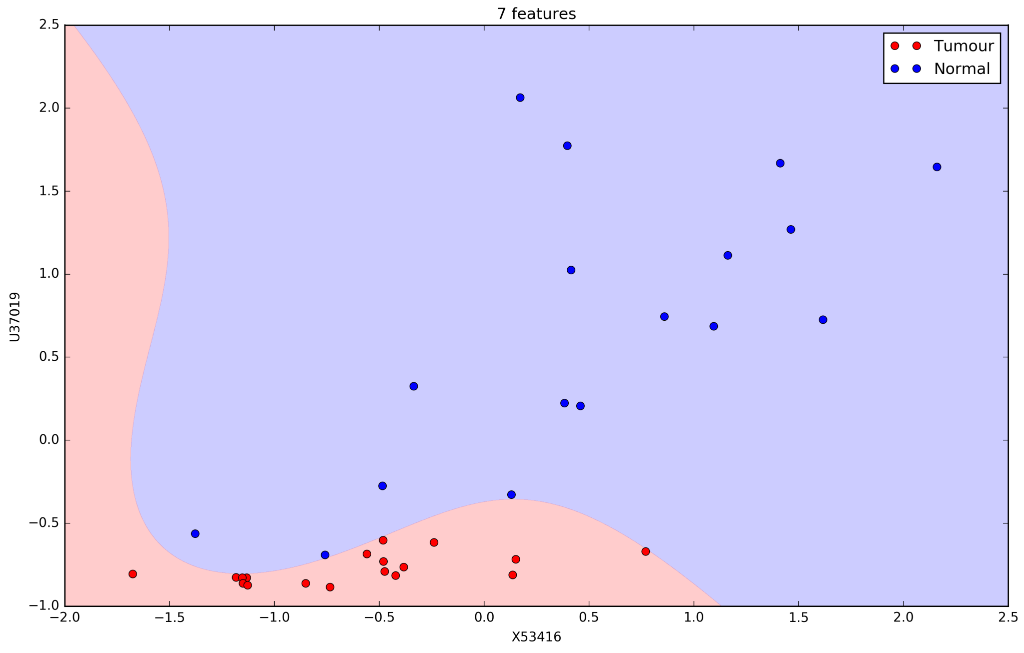

Figure 4.10: Gene activity (X53416 and U37019 genes), decision boundary obtained using a five additional features computed from the original two.

Once again, we need to face the possibility of overfitting our data. This will be covered in the next lecture.

4.5 Summary

In this chapter we covered the notions of *linear separability* and *linear discriminants*. We also saw that finding linear discriminant by minimizing a simple quadratic error did not place the boundary in the right position. However, we saw that using a logistic function to model the probability of obtaining examples of each class as a function of the feature vectors and the discriminant allows us to find the best discriminant by maximum likelihood. We also saw how expanding our features into a feature space with more dimensions can allow us to use linear discriminants in higher dimensions to separate classes that are not linearly separable in the original feature space.

4.6 Further Reading

1. Bishop [4], Sections 4.1.1, 4.1.3 and 4.3.2

Bibliography

[1] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] David F Andrews. Plots of high-dimensional data. *Biometrics*, pages 125–136, 1972.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, 1st ed. edition, oct 2006.

[5] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual. Association for Computing Machinery, Inc., October 2004.

[6] Guanghua Chi, Yu Liu, and Haishandbscan Wu. Ghost cities analysis based on positioning data in china. *arXiv preprint arXiv:1510.08505*, 2015.

[7] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning. Stanford CA Morgan Kaufmann*, pages 231–238, 2000.

[9] Hakan Erdogan, Ruhi Sarikaya, Stanley F Chen, Yuqing Gao, and Michael Picheny. Using semantic analysis to improve speech recognition performance. *Computer Speech & Language*, 19(3):321–343, 2005.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley. Dna visual and analytic data mining. In *Visualization'97., Proceedings*, pages 437–441. IEEE, 1997.

[14] Chang-Hwan Lee, Fernando Gutierrez, and Dejing Dou. Calculating feature weights in naive bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1146–1151. IEEE, 2011.

[15] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[16] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[17] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[20] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques for Multimedia*, Cognitive Technologies, pages 159–187. Springer Berlin Heidelberg, 2008.

[21] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[22] Jake VanderPlas. Frequentism and bayesianism: a python-driven primer. *arXiv preprint arXiv:1411.5018*, 2014.