Chapter 9

Support Vector Machines, part 1

Maximum margin classifier. Signed distance to decision frontier. Support Vectors and Support Vector Machine for linear classification

9.1 Maximum margin

In logistic regression and the perceptron we saw examples of separating different classes using a hyperplane. We saw that the squared error between the class and the distance to the hyperplane was not a good measure because it pulls the frontier towards the more distant points. By using a function that "squashes" the outputs away from the frontier, such as the logistic function, allowed us to find a better way of separating the classes. Figure 10.1 illustrates this difference. The left panel shows a linear discriminant computed by minimizing the squared errors loss function $e_{sq.}$, and the right panel the discriminant obtained with logistic regression, minimizing the loss function $E_{log.}$:

$$E_{sq.}(\widetilde{w}) = \sum_{j=1}^{N} (g(\vec{x_j}) - t_j)^2 \qquad E_{log.}(\widetilde{w}) = -\sum_{n=1}^{N} [t_n \ln g_n + (1 - t_n) \ln(1 - g_n)]$$

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

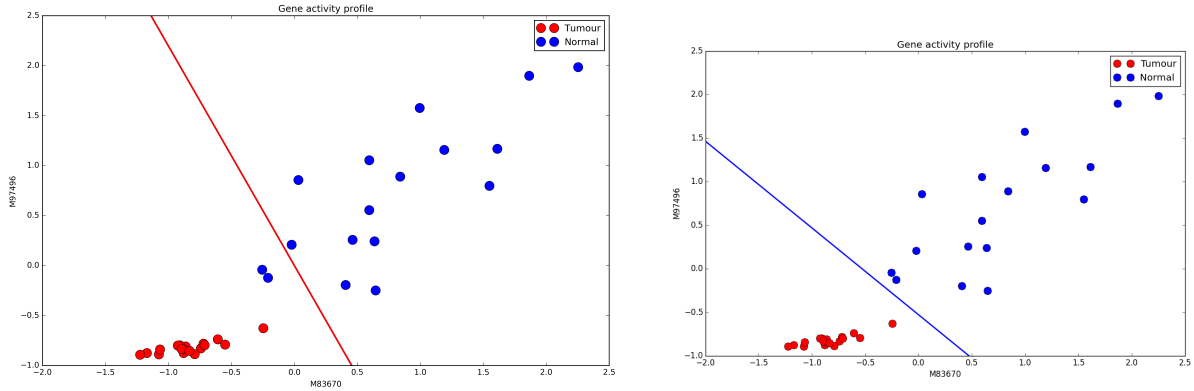

Figure 9.1: Gene activities for cancerous and normal cells. The linear discriminants were computed by least squared error (left panel) and logistic regression (right panel).

However, there is one disadvantage to an error function like the one used in logistic regression. While the quadratic error function has a well-defined minimum, because increasing the norm of vector $\widetilde{w}$ can make the decision function arbitrarily steep in the frontier, the frontier can be placed in any of a range of possible places. Figure 9.2 illustrates this problem. The left panel shows a series of results from the logistic regression minimization. Since the logistic function is flat away from the frontier, displacing the frontier makes little difference. This may result in the frontier being placed closer to some points, as shown in the right panel, increasing overfitting.

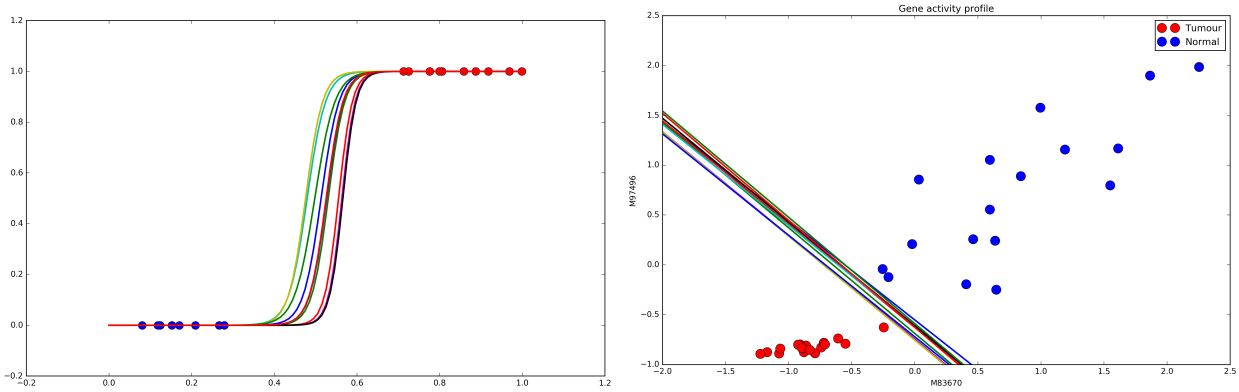

Figure 9.2: Logistic regression frontiers. Different runs of the same optimization result in different positions for the frontier because, if the logistic function is very steep at the frontier, placing it at different positions makes no difference for the loss function.

Regularization helps reduce this effect by forcing vector $\widetilde{w}$ to be shorter, smoothing the logistic function and forcing the frontier away from the closest points. Figure 9.3 shows the same logistic regression results as Figure 9.2, but this time using L2[1] regularization. This forces the optimization to always give the same result and always the same frontier, fixed farther away from the points.

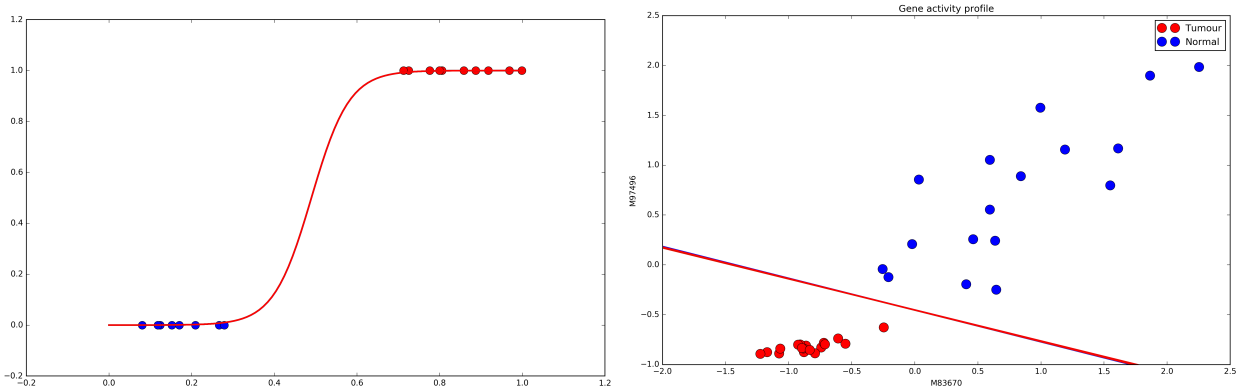

Figure 9.3: Logistic regression with L2 regularization, forcing the norm of $\widetilde{w}$ to be small, smooths the function and fixes the frontier away from the closest points.

This example shows the important concept of a *maximum margin classifier*. A *margin classifier* is a classifier that provides a measure of the distance between the frontier and the points closest to it. This is the *margin* of the classifier. A *maximum margin classifier* is a classifier that maximizes this distance. With logistic regression, we can approximate this using regularization, but this requires modifying the loss function to include the regularization term. The code below shows the loss function

[1] L2 regularization penalizes the square of the norm of $\widetilde{w}$

for the regularized logistic regression. Even though the regularization constant is small (0.00001), regularization always distorts the objective function of minimizing the error.

```
def log_cost(theta,X,y):
    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)+np.sum(coefs**2)*0.00001
```

A better option is to make margin maximization an explicit goal for our loss function. Figure 9.4 shows the margin, which is the distance between the frontier and the examples closest to it. These vectors are called the *support vectors*. To explicitly maximize the margin, we can consider the signed distance between a vector and the decision hyperplane:

$$r = \frac{\vec{w}^T x + w_0}{||\vec{w}||}$$

The value of r is positive on one side of the decision hyperplane and negative on the other, because of the value of the inner product $r = \vec{w}^T x + w_0$. Furthermore, r is invariant with respect to the norm of the vector defining the hyperplane, due to the division by $||\vec{w}||$.

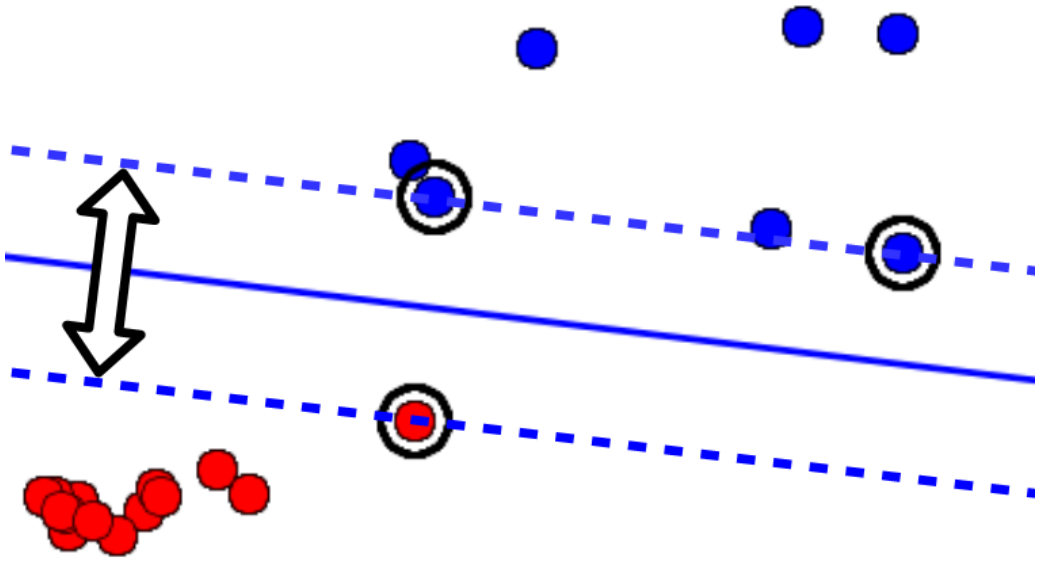

Figure 9.4: The margin is the distance to the points nearest to the decision frontier.

Now we can try to find the hyperplane that maximizes the minimum distance to points being classified:

$$\underset{\vec{w},b}{\text{argmax}} \left(\min_j \frac{y_j(\vec{w}^T x_j + w_0)}{||\vec{w}||} \right)$$

writing this loss function:

```
def closest_dist(ws, Xs, Ys):
    coefs = np.zeros((len(ws)-1,1))
    coefs[:,0] = ws.flatten()[:-1]
```

```
    dists = np.dot(Xs,coefs) + ws[2]
    norm = np.sqrt(ws[0]**2+ws[1]**2)
    return -np.min(dists * Ys / norm)

# load data
x0 = np.random.rand(3)
sol =  minimize(closest_dist, x0, args = (Xs,Ys))
```

Note the negative sign on the returned value because we are using the `minimize` function to find the maximum value. Despite the conceptual simplicity of this solution and the ease with which it can be implemented, unfortunately this does not work. As Figure 9.5 shows, the landscape of this loss function has discontinuous derivatives because, as the position of the discriminant hyperplane changes, it also changes which vectors are closest to this plane. Furthermore, the maximum value of the margin coincides with orientations for which several vectors are equidistant to the decision hyperplane, making the derivative discontinuous at the desired solution. Because of this, the optimization algorithm is unable to find the correct solution, as shown on the right panel.

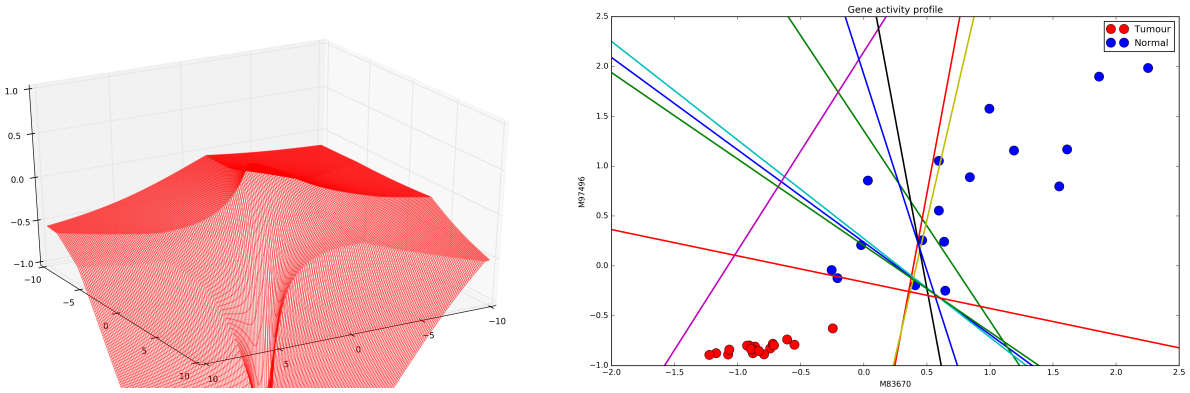

Figure 9.5: Landscape of the loss function for maximizing the minimum distance to the decision hyperplane (left panel) and the resulting hyperplanes due to the inability of the minimization algorithm to converge to the maximum margin.

To solve this problem we need a different approach.

9.2 Support Vector Machine

We start by noting that the normalized distance is invariant to scaling:

$$\frac{y_n(w^T x_n + w_0)}{||w||} = \frac{y_n(\beta\vec{w}^T x_n + \beta w_0)}{\beta||\vec{w}||}$$

So we can impose this condition by making $\vec{w}$ and w_0 as large as necessary:

$$y_n(\vec{w}^T x_n + w_0) \geq 1, \forall n \in N$$

Subject to this condition, the problem of maximizing the margin is equivalent to the problem of minimizing the norm $||\vec{w}||$. As long as the condition above is not violated, shrinking the size of $\vec{w}$ means we are effectively increasing the distance between the discriminant and the closest points. More conveniently, we can minimize a quadratic function of the norm $||\vec{w}$, since minimizing quadratic functions is computationally more convenient.

$$\underset{\vec{w},b}{\text{argmax}} \left(\min_j \frac{y_j(\vec{w}^T x_j + w_0)}{||\vec{w}||} \right) = \arg\min_{\vec{w},w_0} \frac{1}{2}||\vec{w}||^2$$

Note that w_0 is determined by the constraint. This is thus a constraint optimization problem. One method for solving constraint optimization problems is to use *Lagrange multipliers*. To illustrate the approach, consider the following example:

$$\arg\max_{x,y} (1 - x^2 - y^2) \qquad s.t. x - y - 1 = 0$$

At the maximum along the line defining the constraint, the component of the function's derivative that is parallel to the constraint line must be zero, because otherwise this would not be a local maximum. This means that, at this point, the constraint line is tangent to a contour line of the objective function. Figure 9.6 illustrates this.

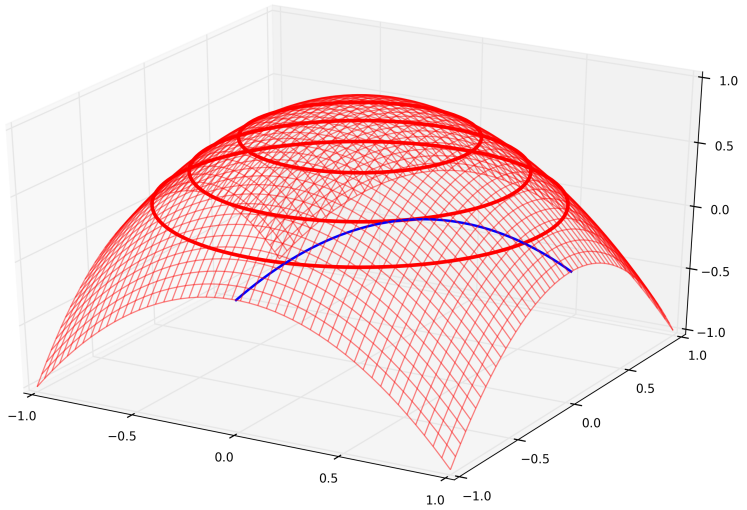

Figure 9.6: Objective function surface (in red) and the constraint line (blue). The red lines depict the contour lines of the objective function.

Since $g(x, y) = 0$ is a contour line of g, if $f(x, y)$ is a maximum subject to $g(x, y) = 0$, then the contour line of $f(x, y)$ is parallel to the contour line of $g(x, y)$. And if the contour lines are parallel then the gradient vectors are also parallel, because the gradient must be perpendicular to the contour line. So we can write:

$$\vec{\nabla}_{x,y} f(x, y) = -\alpha \vec{\nabla}_{x,y} g(x, y)$$

The negative sign is conventional and α is a *Lagrange multiplier*. We combine these in the *Lagrangian function*:

$$\mathcal{L}(x, y, \alpha) = f(x, y) + \alpha g(x, y)$$

and solve:

$$\vec{\nabla}_{x,y,\alpha} \mathcal{L}(x, y, \alpha) = 0$$

to find the critical points of the *Lagrangian*, among which the constrained optimum can be found. In our example:

$$\vec{\nabla}_{x,y,\alpha}\left(1 - x^2 - y^2 + \alpha(x - y - 1)\right) = 0$$

Can be solved by

$$\frac{\delta\mathcal{L}}{\delta x} = -2x + \alpha \qquad \frac{\delta\mathcal{L}}{\delta y} = -2y - \alpha \qquad \frac{\delta\mathcal{L}}{\delta\alpha} = x - y \breve{} 1$$

$$x - y - 1 = 0 \Leftrightarrow x = y + 1 \qquad \alpha = 2x\,, \alpha = -2y \Leftrightarrow x = -y$$

The solution is thus $\{0.5, -0.5\}$. Applying the same method to the problem of maximizing the margin of the classifier:

$$\arg\min_{w,w_0} \frac{1}{2}||\vec{w}||^2 \qquad s.t. y_n(\vec{w}^T\vec{x}_n + w_0) \geq 1, \forall n \in N$$

Noting that

$$y_n(\vec{w}^T\vec{x}_n + w_0) \geq 1 \iff -\left((y_n(\vec{w}^T x_n + w_0) - 1\right) \leq 0$$

we obtain the following *Lagrangian* :

$$\mathcal{L}(\vec{w}, w_0, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 - \sum_{n=1}^{N} \alpha_n \left(y_n(\vec{w}^T x_n + w_0) - 1\right)$$

We want to minimize the function with respect to vector $\vec{w}$ and b, while obtaining the maximum with respect to the α_n multipliers. Since at the critical point the derivatives with respect to $\vec{w}$ and w_0 are 0, we can write:

$$\frac{\delta\mathcal{L}}{\delta\vec{w}} = 0 \Leftrightarrow \vec{w} = \sum_{n=1}^{N} \alpha_n y_n \vec{x}_n \qquad \frac{\delta\mathcal{L}}{\delta w_0} = 0 \Leftrightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

Replacing, we obtain the *dual representation* of our original problem. In optimization problems, duality is a relation between two different perspectives on the problem, solving for different sets of variables. In general, the *dual* of an optimization problem only provides a bound on the optimal value but, in this case, solving the dual solves the original problem. So now we solve this *dual representation* of our problem as a function of the *lagrangian multipliers*:

$$\tilde{\mathcal{L}}(\vec{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n\alpha_m y_n y_m \vec{x}_n^T\vec{x}_m$$

$$\sum_{n=1}^{N} \alpha_n y_n = 0 \qquad \alpha_n \geq 0$$

This can be solved by standard quadratic programming algorithms.

9.3 Implementing a Support Vector Machine

As an example, we'll see how to implement a SVM using the `optimize` function from the `scipy` library. First, we compute the matrix with the inner products of all pairs of training vectors multiplied by their respective classes:

$$H = \sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \vec{x}_n^T \vec{x}_m$$

```
def H_matrix(X,Y):
    H = np.zeros((X.shape[0],X.shape[0]))
    for row in range(X.shape[0]):
        for col in range(X.shape[0]):
            H[row,col] = np.dot(X[row,:],X[col,:])*Y[row]*Y[col]
    return H
```

Now we define the function to maximize:

$$\arg\max_{\vec{\alpha}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \vec{x}_n^T \vec{x}_m$$

Intuitively, we can see that α_n will be zero (all *lagrangian multipliers* are non-negative numbers) for all vectors that are surrounded by vectors of the same class, because for these the inner products with vectors of the same class will have a greater weight in the sum, and these contribute positively to the total because the products of the class labels $y_n y_m$ will be positive. Conversely, for vectors close to vectors of the opposite class, there will be a non-zero optimal value for α_n that balances the increase of the sum of the α values and the penalty given to the sum of the inner products. However, if a point has too many neighbours of the other class, the inner products sum will be negative and the α_n value will tend towards infinity, so no solution can be found. This happens if the classes are not linearly separable.

Since we are using the `minimize` function, we need to change the sign of the result. It is also useful to provide the optimization algorithm with the jacobian matrix, which consists of the derivatives of our function with respect to each α_n. This improves the convergence of the algorithm.

```
def loss(alphas):
    return 0.5 * np.dot(alphas.T, np.dot(H, alphas)) - np.sum(alphas)

def jac(alphas):
    return np.dot(alphas.T,H)-np.ones(alphas.shape[0])
```

Now we set up the constraints and minimize the target function using the Sequential Least Squares Programming method (SLSQP).

```
H = H_matrix(Xs,Ys)
A = Ys[:,0] # sum of alphas is zero
cons = {'type':'eq',
        'fun':lambda alphas: np.dot(A,alphas),
        'jac':lambda alphas: A}
bounds = [(0,None)]*Xs.shape[0] #alpha>=0
x0 = np.random.rand(Xs.shape[0])
sol =  minimize(loss, x0, jac=jac, constraints=cons, method='SLSQP', bounds = bounds)
```

For the `minimize` function, the constraints are specified in the dictionary with the function and derivatives for the constraint line setting $\sum_{n=1}^{N} \alpha_n y_n = 0$, which corresponds to the inner product of thevector of α values and the classes of the respective points. The constraint $\alpha_n \geq 0$ is specified in the `bounds` variable.

For all examples that are distant from the margins, the α values are zero. The *support vectors*, those examples that lie at the margins, have an α value greater than zero and can be easily identified:

```
svs = sol.x>0.001
print svs
[False False False False False False  True False False False False False
 False False False False False False False False False False False  True
 False False False False False False False False False False False  True]
```

Now we can compute $\vec{w}$ and b from the *support vectors* (for b, we can average over all support vectors).

$$\vec{w} = \sum_{n=1}^{N} \alpha_n y_n \vec{x}_n \qquad b = y_n - \vec{w}^T \vec{x}_n$$

```
def svm_coefs(X,Y,alphas):
    w = np.sum(alphas*Y*X.T,axis = 1)[:,np.newaxis]
    b = np.mean(Y-np.dot(X,w))
    coefs = np.zeros(len(w)+1)
    coefs[-1] = b
    coefs[:-1] = w.flatten()
    return coefs

coefs = svm_coefs(Xs[svs,:],Ys[svs,0],sol.x[svs])
```

Figure 9.7 shows the resulting hyperplane and the support vectors found.

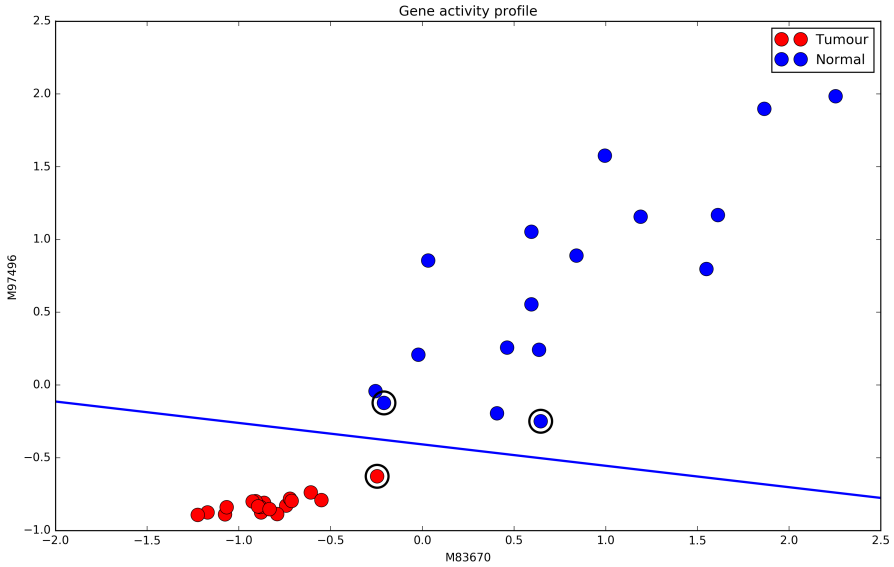

Figure 9.7: SVM classifier. The decision hyperplane is represented in blue and the support vectors are outlined with a black circle.

9.4 Further Reading

1. Alpaydin [2], Sections 13.1 and 13.2
2. Marsland [17], Section 5.1

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.