Chapter 5

Overfitting Logistic Regression

Classification errors. Cross validation. Model selection with cross validation and Logistic Regression. Regularization in Logistic Regression

5.1 Scoring binary classifiers

In Chapter 3, we saw the difference between the *training error*, measured on the set of points used to fit the model; the *validation error*, measured outside the training set to estimate the error of each of a number of hypotheses in order to select the best one; and the *test error*, measured in an another set of points and remaining an unbiased estimator of the *true error* because it is never used to fit or select an hypothesis. In all these cases, we always measured the quadratic error between the predicted and the target values:

$$E(\theta|\mathcal{X}) = \sum_{t=1}^{n}[y^t - g(x^t|\theta)]^2$$

In regression, we used this function to fit the data, validate and test the regression hypotheses. However, in Chapter 4, we saw that, for classifying data using a linear discriminant defining a hyperplane, the quadratic error measured as the distance to the discriminant was not the best cost function for minimization. In Logistic Regression, we used a logistic function to estimate the probability of each class and then obtained, by maximum likelihood, a cost function to minimize:

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

with

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

Since g_n is our predicted probability of example n belonging to class $t = 1$, this is actually the cross-entropy between the probability distribution of our data and the probability distribution of our predictions. Averaging over all samples, we get the average cross-entropy. This is called the *logistic loss* or *log loss* function:

$$L(\widetilde{w}) = \frac{1}{N}\sum_{n=1}^{N}H(p_n,q_n) = -\frac{1}{N}\sum[t_n \ln g_n + (1-t_n)\ln(1-g_n)]$$

The lower the *log loss*function the better our hypothesis is at predicting the training data.

Another possible measure is the quadratic error between the probability prediction given by our hypothesis g_n and the class $t \in 0, 1$. This is called the *Brier score*:

$$E(\widetilde{w}) = \frac{1}{N}\sum_{n=1}^{N}[t_n - g_n]^2$$

Figure 5.1 shows the surface of the predicted probabilities of each point belonging to class 1 and the points used to fit this model. The quadratic error will be the sum of the squared differences between this surface and the class value for each point. Note that, in this case, the error is measured not from the distance to the frontier but from the difference between the class and the estimated probability of the point belonging to class 1.

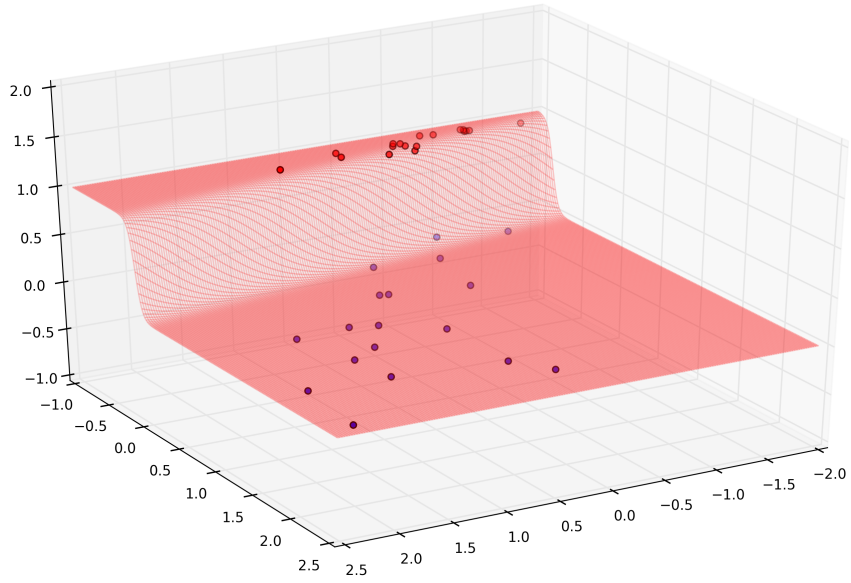

Figure 5.1: Surface representing the predicted probability and the points plotted in their classes with z=0 or z=1.

Another possible error measure is the *accuracy* of the classifier. Let us suppose we consider that any point with $g_n \geq 0.5$ is predicted to be in class $t = 1$ and any point with $g_n < 0.5$ is predicted to be in class $t = 0$. We can consider four different possibilities:

1. *True positive*: the example belongs to class 1 and was predicted to belong to class 1.
2. *False positive*: the example belongs to class 0 and was predicted to belong to class 1.
3. *True negative*: the example belongs to class 0 and was predicted to belong to class 0.
4. *False negative*: the example belongs to class 1 and was predicted to belong to class 0.

Schematically, we can represent these four possibilities with a *confusion table*:

		Examples	
		Class 1	Class 0
Predictions	Class 1	True Positive	False Positive
	Class 0	False Negative	True Negative

The *accuracy* of a binary classifier over a set of N points is thus:

$$accuracy = \frac{true\ positives + true\ negatives}{N}$$

Considering this classification, we can also define the *precision* and *recall* of the classifier:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \qquad recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

In other words, *precision* is the fraction of correctly classified positive examples in all examples classified as positive (correctly or not), whereas *recall* is the fraction of correctly classified positive examples from the set of all positive examples. This gives us another useful measure of the performance of a classifier, the *F1* score, which is the harmonic mean of *precision* and *recall*:

$$F1 = \frac{2 \times true\ positives}{2 \times true\ positives + false\ positives + false\ negatives}$$

$$F1 = 2\frac{precision \times recall}{precision + recall}$$

Although the usual approach is to consider that $g_n \geq 0.5$ predicts a point in class 1 (positive), we can change the value of this threshold and consider a more general approach of predicting the positive class at $g_n \geq \alpha,\ \alpha \in [0, 1]$. Figure 5.2 shows the effect of drawing the frontier at different values of α and then plotting the number of true and false positives as a function of α. If α is too small, all points will be classified as being in the positive class, so there will be a maximum number of true and false positives. As α increases, the false positives should start decreasing first. When α is too high, then all points are classified as being in the negative class, which means there are no false positives but no true positives either.

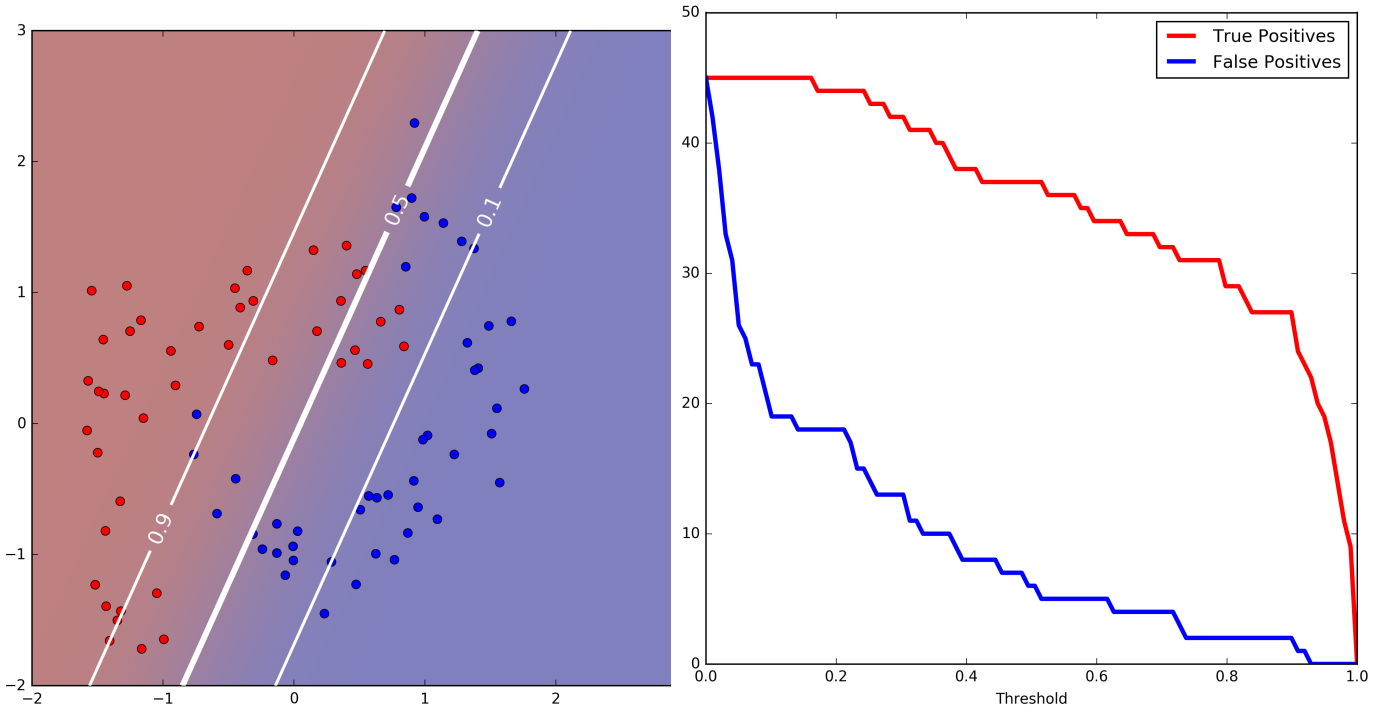

Figure 5.2: The left panel shows different contours of the probability of finding class 1 (in red). The right panel shows the true positives and false positives as a function of the threshold α.

Using this variation in the fraction of true positives and false positives as a function of the threshold, we can also evaluate a binary classifier by plotting a *receiver operating characteristic* curve, or *ROC* curve[1]. The *ROC* curve is plotted by computing the fraction of *true positives* and *false positives* at different score thresholds. A classifier performs all the better the greater the fraction of *true positives* relative to the *false positives* for different threshold levels. In other words, the larger the area below the *ROC* curve the better the classifier's performance. Figure 5.3 shows an example of a *ROC* curve.

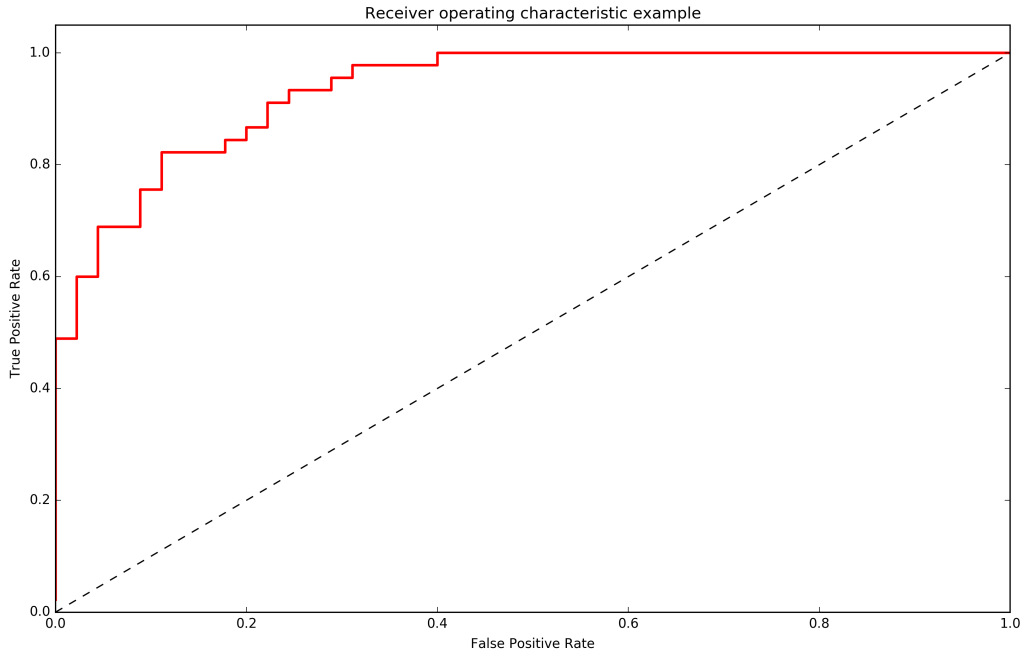

Figure 5.3: A ROC curve.

Classifiers in the Scikit-Learn offer a `score(X,Y)` method that returns the *accuracy* score for the classifier computed on the given data. From this score, we can also compute the error as one minus the *accuracy*.

```
1 from sklearn.linear_model import LogisticRegression
2
3 reg = LogisticRegression()
4 reg.fit(X_r,Y_r)
5 test_error = 1-reg.score(X_t,Y_t)
```

5.2 Cross-Validation and Model Selection

In Chapter 3 we saw a simple way to solve the overfitting problem, which was to select the hypothesis that had a smaller *validation error*. To do this we split our data set into a *training set* and *validation set* (and, if desired, a *test set* to estimate the *true error* of the selected hypothesis). However, all these estimates are random samples from some probability distribution and we can improve them by averaging over several repetitions. Furthermore, doing validation in that way only allowed us to evaluate specific hypotheses and not the models themselves. Cross-validation solves these problems.

[1]The name comes from the original use of this method, which was to optimize the detection rate of aircraft in radar signals during the second world war

To do cross-validation, we partition our data into a number of disjoint *folds*. For example, if we have 50 points and want to use 5-fold cross-validation, we place 10 points into each fold. Then we train our model with all folds but one, validate on the fold that was left out, and repeat for all folds. In the end we average the validation error and this gives us an estimate of the true error that, on average, hypotheses generated from this model will have on this type of data. Figure 5.4 shows an example of 5-fold cross validation using the gene expression data. Each panel shows an hypothesis obtained by fitting the model to four of the folds (indicated by the smaller points) and then validating using the fold left out.

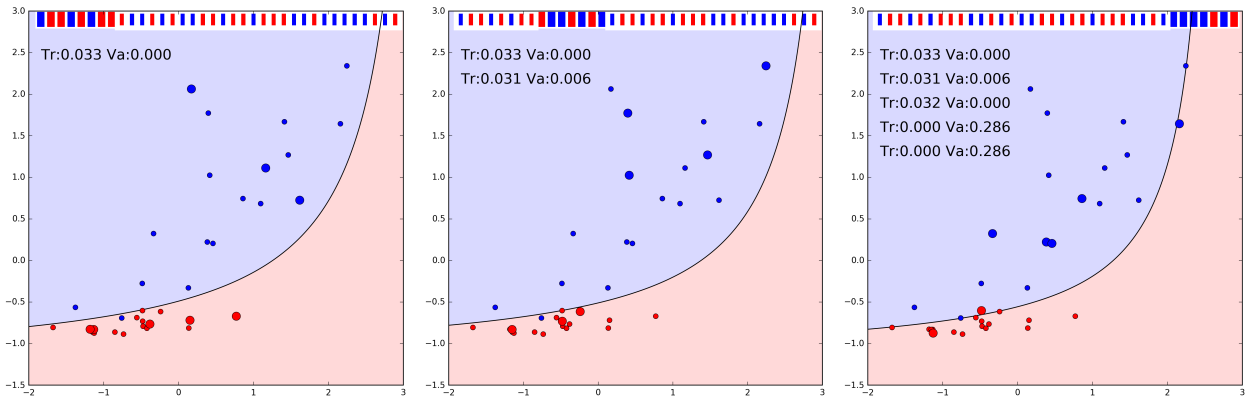

Figure 5.4: Example of 5-fold cross validation, showing the plots for folds 1, 2 and 5. In each panel, one of the folds is left out for validation, the other folds are used for training. The larger points are those used for validation in each fold. The training and validation errors are kept for each fold and then averaged in the end.

In general, *k-fold cross-validation* can be done with any number of folds from two to the number of data points. In this last case, it is called *leave-one-out cross-validation*.

To illustrate this, consider the data set in Figure 5.5. We want to find the best model to separate these data with a logistic regression. First, we load the data, shuffle the order of the points randomly, and then we set aside a third of the data points for the final error evaluation (the *test set*). Ordering the points at random is often necessary to eliminate any correlations in the data set. For example, in this case, all positive class examples are first in the file. We also standardize the features.

```
1 import numpy as np
2 from sklearn.utils import shuffle
3
4 mat = np.loadtxt('dataset_90.txt',delimiter=',')
5 data = shuffle(mat)
6 Ys = data[:,0]
7 Xs = data[:,1:]
8 means = np.mean(Xs,axis=0)
9 stdevs = np.std(Xs,axis=0)
10 Xs = (Xs-means)/stdevs
```

We will select the best model using 10-fold cross-validation on the training set. The different models to consider are different expansions of the original data, $x_1, x_2, x_1, x_2, x_1 \times x_2, x_1, x_2, x_1 \times x_2, x_1^2, ...$, each resulting in a model with a different number of features. To do this, we expand the original features polynomially into a matrix of 16 features. Please note that this is not a common use of logistic regression. Explicitly expanding features like this is not very efficient; there are better algorithms for

this that we will see later. However, this exercise is useful to help understand the idea of transforming the examples so that they become linearly separable.

```
def poly_16features(X):
    """Expand data polynomially"""
    X_exp = np.zeros((X.shape[0],X.shape[1]+14))
    X_exp[:,:2] = X
    X_exp[:,2] = X[:,0]*X[:,1]
    X_exp[:,3] = X[:,0]**2
    X_exp[:,4] = X[:,1]**2
    #... rest of the expansion here
    return X_exp
```

As we saw previously, the larger the dimension into which we expand the original data, the easier it is to separate the classes in the training set but the more likely the model is to overfit the data. So, we partition the training set into 10 folds, train each model 10 times, leaving out one fold for validation and average the training and validation error. In this case, we estimate the error using the *Brier score*, which is the average square difference between the class value and the predicted probability of each point being in class 1. This is easy to do with the `sklearn` library. First we create a function that returns the training and test error given a data set and the indexes of the training and test points, using the number of features indicated.

```
from sklearn.linear_model import LogisticRegression

def calc_fold(feats, X,Y, train_ix,test_ix,C=1e12):
    """return classification error for train and test sets"""
    reg = LogisticRegression(penalty='l2',C=C, tol=1e-10)
    reg.fit(X[train_ix,:feats],Y[train_ix,0])
    prob = reg.predict_proba(X[:,:feats])[:,1]
    squares = (prob-Y[:,0])**2
    return (np.mean(squares[train_ix]),
            np.mean(squares[test_ix]))
```

This function fits the logistic regression classifier to the training set, then predicts the probabilities for all the set and returns the mean squared error for the training and test sets. Note that the computation of the Brier score by subtracting the predicted probability and the class assumes that classes are 0 and 1. If the class labels are not 0 and 1 we must convert them to these values.

Now we use the `KFold` class to generate an iterator for the training and validation sets. Here is an example of how a `Kfold` object works:

```
from sklearn.model_selection import KFold

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
kf = KFold(n_splits=4)
for train, valid in kf.split(x):
    print (train, valid)

[ 3  4  5  6  7  8  9 10 11] [0 1 2]
[ 0  1  2  6  7  8  9 10 11] [3 4 5]
[ 0  1  2  3  4  5  9 10 11] [6 7 8]
[ 0  1  2  3  4  5  6  7  8] [ 9 10 11]
```

We expand the data to the maximum number of features, leave out one third of the data for testing and then loop through the range of features. For each model, we iterate through the 10 different folds to do the cross-validation, printing the average training and validation errors:

```
from sklearn.model_selection import train_test_split, StratifiedKFold

Xs=poly_16features(Xs)
X_r,X_t,Y_r,Y_t = train_test_split(Xs, Ys, test_size=0.33, stratify = Ys)
folds = 10
kf = StratifiedKFold(n_splits=folds)
for feats in range(2,16):
    tr_err = va_err = 0
    for tr_ix,va_ix in kf.split(Y_r,Y_r):
        r,v = calc_fold(feats,X_r,Y_r,tr_ix,va_ix)
        tr_err += r
        va_err += v
    print(feats,':', tr_err/folds,va_err/folds)
```

Figure 5.5 illustrates the ten hypotheses obtained for each of two models, with 2 and 6 features, and the mean training and validation errors.

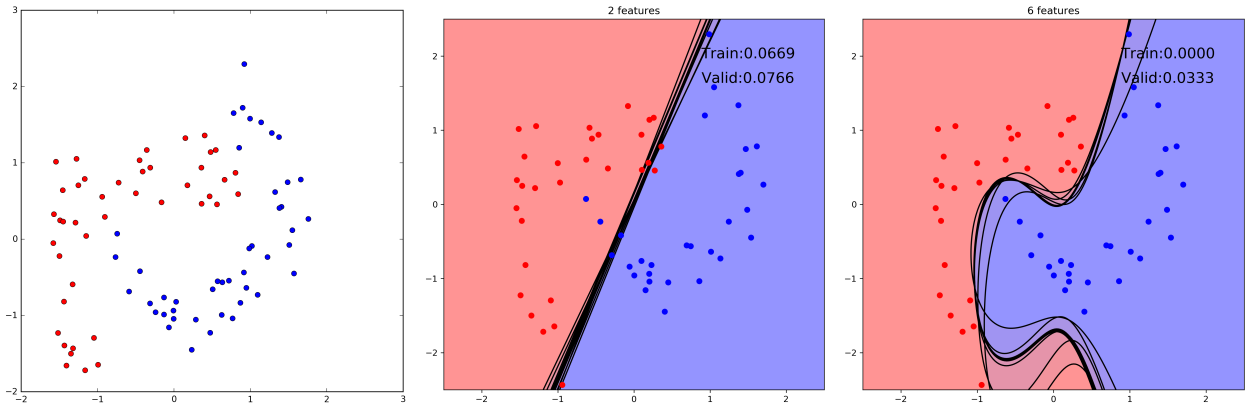

Figure 5.5: Example dataset. The first panel shows the full data set, with positive class (1) in red, negative class (0) in blue. The next two panels show the training set, used in cross-validation. The lines show the 10 different hypotheses obtained during the 10-fold cross-validation. The training and validation errors are the average for the 10 folds.

Figure 5.6 shows the average training and validation error measured for the set of models considered, with expansions of up to 15 features. In this case, we can see that the best model appears to be the one with 9 features. We now train this model with all the points in the training set and estimate the *true error* with the *test set*. Even though the cross-validation error, by itself, is not biased, since we used cross-validation to select the best model we now need to estimate the true error of the best model with examples outside the training set. The hypothesis obtained from the best model trained on the whole training set and the error estimate obtained with the test set is shown on the right panel of Figure 5.6.

5.3 Cross-Validation and Regularization

We can also use cross-validation to find parameters, such as when optimizing regularization. The `LogisticRegression` class can take a regularization parameter, `C`, which, when using a L2 type

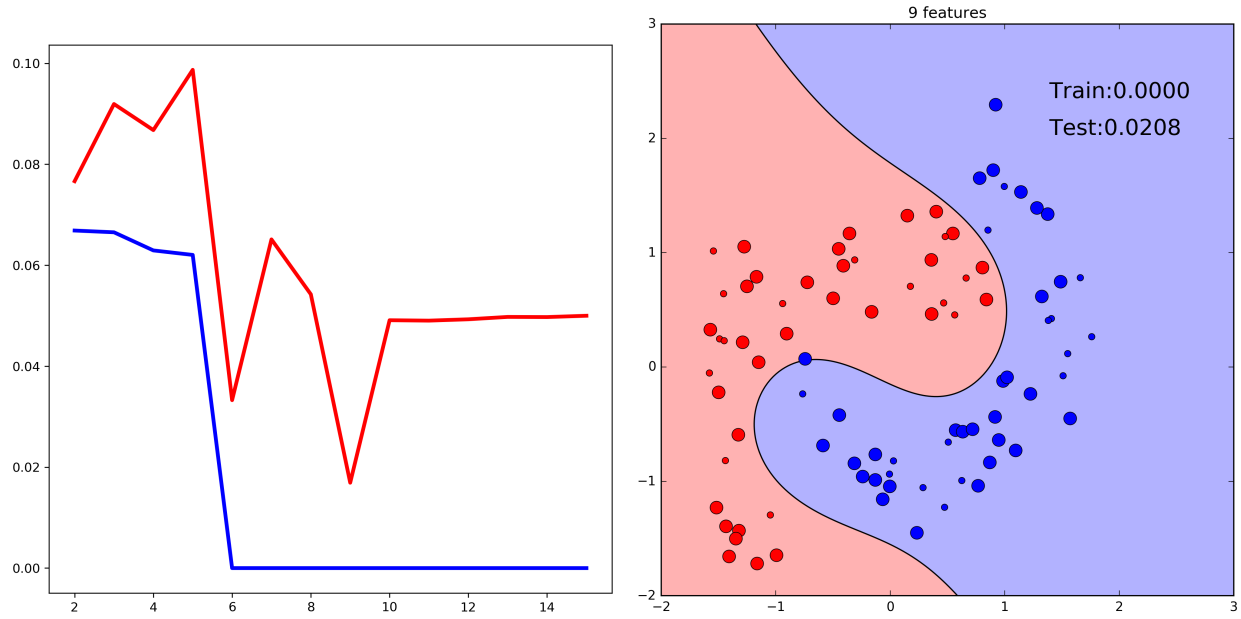

Figure 5.6: The left panel shows the average training and validation error as a function of the number of features in the model. The right panel shows the best model (9 features) trained with the whole training set and tested with the test set (larger points).

regularization, as we saw before with *ridge regression*, corresponds to $\frac{1}{\lambda}$. So our regularization term in this case will be

$$\frac{1}{C}\sum_{j=1}^{m} w_j^2$$

where w_j are the coefficients for the hyperplane separating the classes.

Penalizing the hyperplane will force the coefficients of $\widetilde{w}$ to be smaller. This affects the slope of the logistic function:

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

Without regularization, $\widetilde{w}$ can be as large as necessary and the logistic function can be very sharply sloped, allowing the discriminant to be placed very close to the data points. In this example below we can see that, without regularization, the logistic function is steep enough to separate these classes perfectly:

However, this is probably over-fitting the data since the sole blue point close to the red class is likely to be an outlier and not representative of the data in general. If we regularize the logistic regression classifier, the regularization will force $\widetilde{w}$ to be smaller and thus decrease the slope of the logistic function. This forces the margins around the discriminant to be wider, which in turn forces the discriminant away from the larger number of red class points:

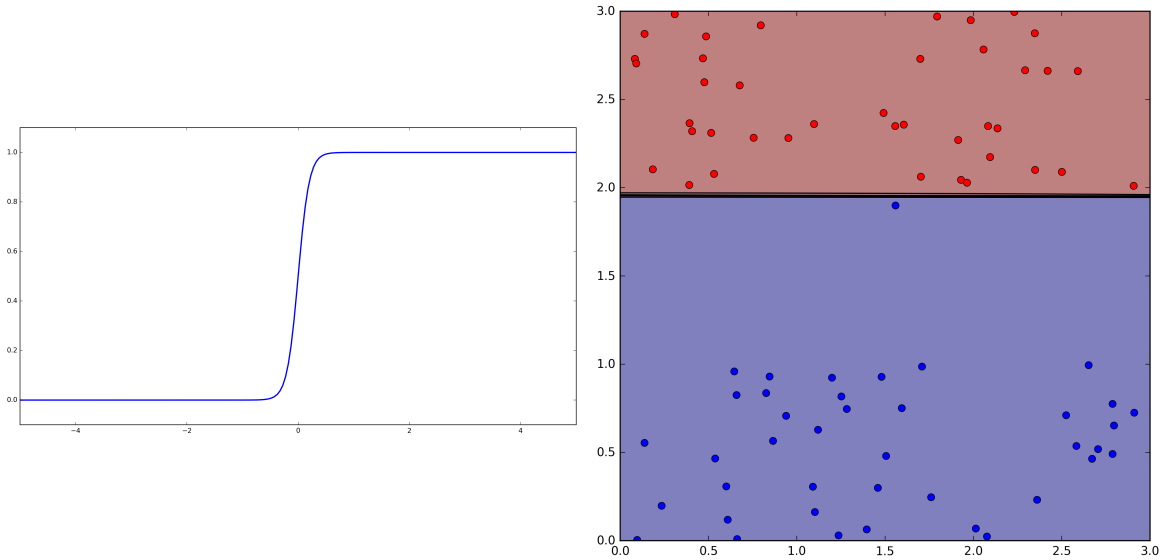

Figure 5.7: Arbitrarily steep logistic function, without regularization.

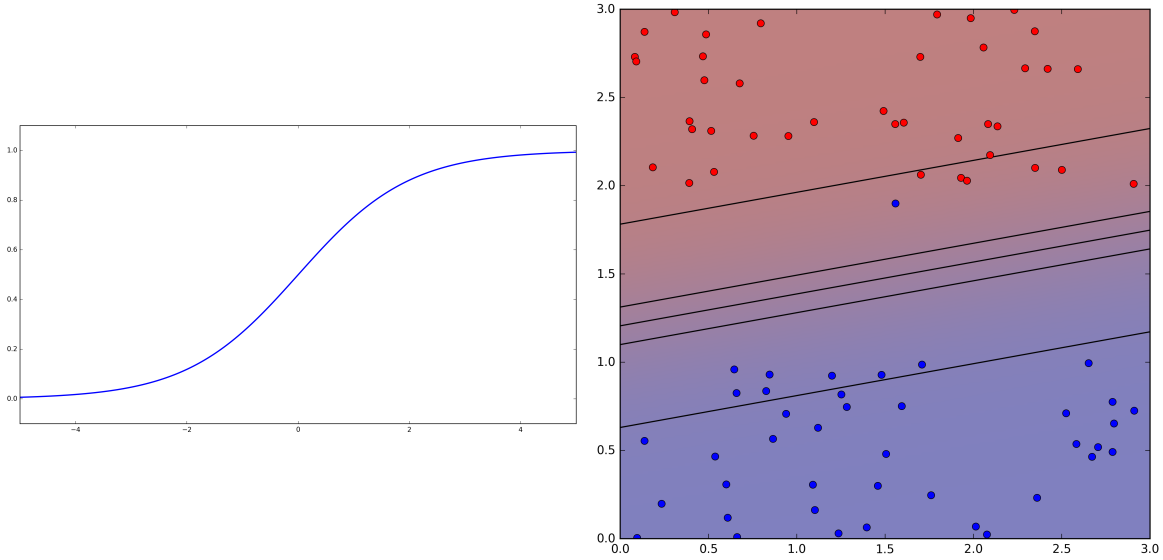

Figure 5.8: Regularization reduces the size of coefficients, making the logistic function less steep.

Figure 5.9 shows the average training and validation error measured for different values of `C`, from 10^{-5} to 10^{15}, always using the model with 15 features, for the original data set shown in Figure 5.5. The right panel shows the result of fitting the model using a `C` value of 10^{5} to all the points in the training set and then testing with the test set we left out in the beginning.

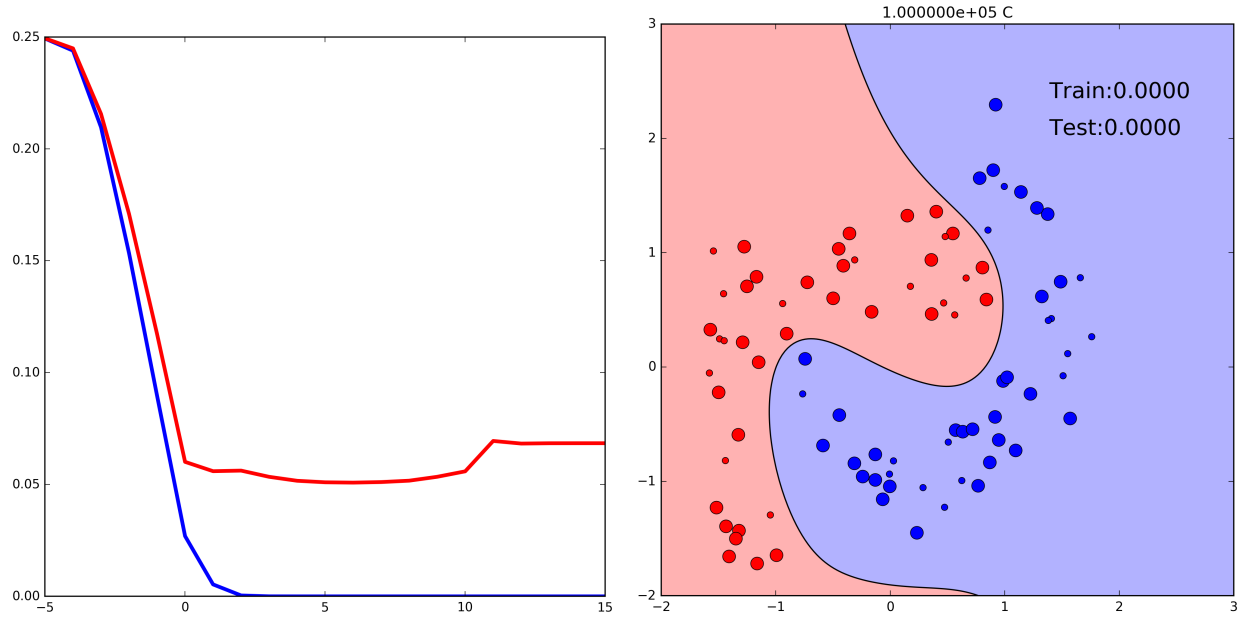

Figure 5.9: The left panel shows the cross-validation error plotted against $log_{10}(C)$ for the 15 features model. The right panel shows the 15 features model regularized with $C = 10^5$ trained on the complete training set.

5.4 Summary

In this chapter we saw different ways of scoring classifiers, covered model selection with *cross-validation* and also saw how to use *cross-validation* to find the best regularization parameter.

5.5 Further Reading

1. Alpaydin [2], Section 2.7

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.