15 Support vector machines and kernel functions

Improving classifier performance has been an area of intensive machine-learning research for the last two decades, and this work has led to a new generation of state-of-the-art classifiers, such as support vector machines, boosted decision trees, regularized logistic regression, neural networks, and random forests. Many of these methods, including support vector machines (SVMs), the main topic of this chapter, have been applied with success to information retrieval problems, particularly text classification. We will initially motivate and develop SVMs for the case of two-class data sets that are separable by a linear classifier (Section 15.1), and then extend the model to non-separable data (Section 15.2) and nonlinear models (Section 15.3). The chapter then concludes with more general discussion of experimental results for text classification (Section 15.4) and system design choices and text-specific features to be exploited in all text categorization work (Section 15.5). Support vector machines, otherwise referred to as large-margin classifiers, are not necessarily better than other methods in the above group (except perhaps in low data situations), but they perform at the state-of-the-art level and have much current theoretical and empirical appeal. No one ever got fired for using an SVM.

15.1 Support vector machines: The linearly separable case

For some training data sets, such as the one in Figure 14.8 (page 226), there are lots of possible linear separators. Intuitively, the gray/green one seems better than the black/red one because it draws the decision boundary in the middle of the void between the data. While some learning methods such as the perceptron algorithm just find any linear separator, others search for the best linear separator according to some criterion. The SVM in particular defines the criterion to be looking for a decision surface that is maximally far away from any data points. This distance from the decision surface to the closest data point determines the margin of the classifier.
This seems like a good thing to do because points near the decision surface represent very uncertain classification decisions: there is almost a 50% chance of the classifier deciding either way. A classifier with a large margin makes no very uncertain classification decisions. Another intuition motivating SVMs is shown in Figure 15.1. By construction, an SVM classifier insists on a large margin around the decision boundary. Compared to a decision hyperplane, if you have to place a fat separator between classes, you have fewer choices of where it can be put. As a result of this, the capacity of the model has been decreased, and hence we expect that its ability to correctly generalize to test data is increased (cf. the discussion of the bias-variance tradeoff in Chapter 14, page 226).

SVMs are inherently two-class classifiers. To do multiclass classification, one has to use one of the methods discussed in Section 14.4 (page 232).

An SVM is constructed to maximize the margin around the separating hyperplane. This necessarily means that the decision function for an SVM is fully specified by a (usually small) subset of the data which defines the position of the separator. These points are referred to as the support vectors.
Figure 15.2 shows the support vectors for a sample problem. Other data points play no part in determining the decision surface that is chosen.

Let us try to formalize this notion with algebra. A decision hyperplane (page 227) can be defined by a decision hyperplane normal vector \( \vec{w} \) which is perpendicular to the hyperplane. Because of this perpendicularity, all points \( \vec{x} \) on the hyperplane satisfy \( \vec{w}^T \vec{x} = 0 \). To choose among all the hyperplanes that are perpendicular to the normal vector, we also specify an intercept term \( b \). Now suppose that we have a set of training data points \( \{ \vec{x}_i \} \) with corresponding classes \( \{ y_i \} \). For SVMs, the two data classes are always named \(+1\) and \(-1\) (rather than 1 and 0), and the intercept term is always explicitly represented as \( b \) (rather than being folded into the weight vector \( \vec{w} \) by adding an extra always-on feature). The math works out much more cleanly if you do things this way, as we will see almost immediately in the definition of
15 Support vector machines and kernel functions

Figure 15.3 The geometric margin of a linear classifier.

The functional margin of a dataset is then twice the minimal functional margin of any point in the data set (the factor of 2 comes from measuring across the whole width of the margin, as in Figure 15.2). However, there is a problem with this definition: we can always make the functional margin as big as we wish by simply scaling up $\vec{w}$ and $b$. For example, if we replace $\vec{w}$ by $5\vec{w}$ and $b$ by $5b$ then the functional margin $y_i(5\vec{w}^T\vec{x}_i + 5b)$ is five times as large. This suggests that we need to place some constraint on the size of the $\vec{w}$ vector. To get a sense of how to do that, let’s look at the actual geometry.

What is the Euclidean distance from a point $\vec{x}$ to the decision boundary? Look at Figure 15.3. Let us call the distance we are looking for $r$. We know that the shortest distance between a point and a hyperplane is perpendicular...
linear to the plane, and hence, parallel to $\vec{w}$. A unit vector in this direction is $\vec{w} / \|\vec{w}\|$. Then, the dotted line in the diagram is a translation of the vector $r \vec{w} / \|\vec{w}\|$. Let us label the point on the hyperplane closest to $\vec{x}$ as $\vec{x}'$. Then we know from the above discussion that:

$$\vec{x}' = \vec{x} - y r \frac{\vec{w}}{\|\vec{w}\|}$$  \hspace{1cm} (15.2)

where multiplying by $y$ again just changes the sign for the two cases where $\vec{x}$ is on either side of the decision surface. Moreover, $\vec{x}'$ lies on the decision boundary and so satisfies that $\vec{w} \vec{x}' + b = 0$. Hence:

$$\vec{w}^T (\vec{x} - y r \frac{\vec{w}}{\|\vec{w}\|}) + b = 0$$

Solving for $r$ gives:

$$r = y \frac{\vec{w}^T \vec{x} + b}{\|\vec{w}\|}$$  \hspace{1cm} (15.3)

Again, the points closest to the separating hyperplane are support vectors. The geometric margin of the classifier is the maximum width of the band that can be drawn separating the support vectors of the two classes. That is, it is twice the maximal $r$ defined above, or the maximal width of one of the fat separators shown in Figure 15.1. The geometric margin is clearly invariant to scaling of parameters: if we replace $\vec{w}$ by $5\vec{w}$ and $b$ by $5b$, then the geometric margin is the same, because it is scaled by the length of $\vec{w}$. This means that we can impose any scaling constraint we wish on $\vec{w}$ without affecting anything. Among other choices, requiring $\|\vec{w}\| = 1$ would make the geometric margin the same as the functional margin.

Since we can scale the functional margin as we please, let us require that the functional margin of all data points is at least 1 and that this bound is tight. Then for all items in the data:

$$y_i (\vec{w}^T \vec{x}_i + b) \geq 1$$

and for the support vectors the inequality is an equality. Since each example’s distance from the hyperplane is $r_i = y_i (\vec{w}^T \vec{x}_i + b) / \|\vec{w}\|$, the geometric margin is $\rho = 2 / \|\vec{w}\|$. Our desire is still to maximize this geometric margin. That is, we want to find $\vec{w}$ and $b$ such that:

- $\rho = 2 / \|\vec{w}\|$ is maximized
- For all $\{(\vec{x}_i, y_i)\}$, $y_i (\vec{w}^T \vec{x}_i + b) \geq 1$

1. Recall that $\|\vec{a}\| = \sqrt{\vec{a}^T \vec{a}}$. 

Preliminary draft (c) 2007 Cambridge UP
Now noting that maximizing $\frac{2}{\|\vec{w}\|}$ is the same as minimizing $\|\vec{w}\|/2$, we have the standard final formulation as a minimization problem:

(15.4) Find $\vec{w}$ and $b$ such that:

- $\frac{1}{2}\vec{w}^T\vec{w}$ is minimized
- and for all $\{((\vec{x}_i, y_i))\}$, $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$

This is now optimizing a quadratic function subject to linear constraints. Quadratic optimization problems are a standard, well-known class of mathematical optimization problem, and many algorithms exist for solving them. We could in principle build our SVM using standard quadratic programming (QP) libraries, though in practice there are more specialized and much faster libraries available especially for building SVMs. There has been much research in this area and many fast but intricate algorithms exist for this problem. However, we will not present the details here. It is enough to understand how the problem is set up as a QP problem. Thereafter, there are only about 20 people in the world who don’t use one of the standard SVM software packages to build models.

Nevertheless, to understand the mechanism for doing nonlinear classification with SVMs, we need to present a fraction more of how SVMs are solved although we will omit the details. The solution involves constructing a dual problem where a Lagrange multiplier $\alpha_i$ is associated with every constraint in the primary problem:

(15.5) Find $\alpha_1, \ldots, \alpha_N$ such that $\sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$ is maximized, and

- $\sum_i \alpha_i y_i = 0$
- $\alpha_i \geq 0$ for all $1 \leq i \leq N$

The solution is then of the form:

(15.6) $\vec{w} = \sum \alpha_i y_i \vec{x}_i$

$b = y_k - \vec{w}^T \vec{x}_k$ for any $\vec{x}_k$ such that $\alpha_k \neq 0$

In the solution, most of the $\alpha_i$ are zero. Each non-zero $\alpha_i$ indicates that the corresponding $\vec{x}_i$ is a support vector. The classification function is then:

$$f(\vec{x}) = \text{sign}(\sum \alpha_i y_i \vec{x}_i^T \vec{x} + b)$$

Notice that both the term to be maximized in the dual problem and the classifying function involve an inner product between pairs of points ($\vec{x}$ and $\vec{x}_i$ or $\vec{x}_i$ and $\vec{x}_j$), and that is the only way the data is used – we will return to the significance of this later.
15.2 Soft margin classification

For the very high dimensional problems common in text classification, sometimes the data is linearly separable. But in the general case it is not, and even if it is, we might prefer a solution that better separates the bulk of the data while ignoring a couple of weird outlier points.

If the training set is not linearly separable, the standard approach is to introduce slack variables $\xi_i$ which allow misclassification of difficult or noisy examples. In this model, the fat decision margin is allowed to make a few mistakes, and we pay a cost for each misclassified example which depends on how far away from the decision surface it is. See Figure 15.4.

The formulation of the optimization problem with slack variables is then:

\begin{equation}
\begin{aligned}
\text{Find } \vec{w}, b, \text{ and } \xi_i \geq 0 \text{ such that:} \\
\quad & \frac{1}{2} \vec{w}^T \vec{w} + C \sum_i \xi_i \text{ is minimized} \\
\quad & \text{and for all } \{(\vec{x}_i, y_i)\}, \ y_i (\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i \\
\end{aligned}
\end{equation}
The optimization problem is then trading off how fat it can make the margin versus how many points have to be moved around to allow this margin. The margin can be less than 1 for a point $\mathbf{x}_i$ by setting $\xi_i > 0$, but then one pays a penalty of $C\xi_i$ in the minimization for having done that. The parameter $C$ is a regularization term, which provides a way to control overfitting: as $C$ becomes large, it is unattractive to not respect the data at the cost of reducing the geometric margin, while when it is small, it is easy to account for some data points with the use of slack variables and to have the fat margin placed so it models the bulk of the data.

The dual problem for soft margin classification becomes:

$$\text{(15.8)}$$

Find $\alpha_1, \ldots, \alpha_N$ such that $\sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized, and

- $\sum_i \alpha_i y_i = 0$
- $0 \leq \alpha_i \leq C$ for all $1 \leq i \leq N$

Note that neither the slack variables $\xi_i$ nor Lagrange multipliers for them appear in the dual problem. All we are left with is the constant $C$ bounding the possible size of the Lagrange multipliers for the support vector data points. Again, the $\mathbf{x}_i$ with non-zero $\alpha_i$ will be the support vectors, and typically they will be a small proportion of the data. The solution of the dual problem is of the form:

$$\text{(15.9)}$$

$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

$b = y_k (1 - \xi_k) - \mathbf{w}^T \mathbf{x}_k$ for $k = \arg \max \alpha_k$

Again $\mathbf{w}$ is not needed explicitly for classification, which can be done simply in terms of the dot product of data points:

$$f(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

Given a new point $\mathbf{x}$ to classify, the classification function is computing the projection of the point onto the hyperplane normal. This will determine which class to assign to the point. If the point is within the margin of the classifier (or another confidence threshold $t$ that we might have determined to avoid classification mistakes) then the classifier can return “don’t know” rather than one of the two classes.

### 15.3 Nonlinear SVMs

With what we have presented so far, data sets that are linearly separable (perhaps with a few exceptions or some noise) are well-handled. But what are we going to do if the data set is just too hard – in the sense that it just doesn’t allow classification by a linear classifier. Let us look at a one-dimensional case
for motivation. The top data set in Figure 15.5 is straightforwardly classified by a linear classifier but the middle data set is not. We instead need to be able to pick out an interval. One way to solve this problem is to map the data on to a higher dimensional space and then to use a linear classifier in the higher dimensional space. For example, the bottom part of the figure shows that a linear separator can easily classify the data if we use a quadratic function to map the data into two dimensions (a polar coordinates projection would be another possibility). The general idea is to map the original feature space to some higher-dimensional feature space where the training set is separable. Though, of course, we would want to do so in ways that preserve relevant notions of data point relatedness, so that the resultant classifier should still generalize well. Kernels can make a non-separable problem separable, and they can map data into a better representational space.
SVMs, and also a number of other linear classifiers, provide an easy and efficient way of doing this mapping to a higher dimensional space, which is referred to as “the kernel trick”. It’s not really a trick: it just exploits the math that we have seen. The SVM linear classifier relies on an inner product between data point vectors. Let $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$. Then the classifier is

$$f(\vec{x}) = \sum_i a_i y_i K(\vec{x}_i, \vec{x}) + b$$  

Now consider if we decided to map every data point into a higher dimensional space via some transformation $\Phi: \vec{x} \mapsto \phi(\vec{x})$. Then the inner product becomes $\phi(\vec{x}_i)^T \phi(\vec{x}_j)$. If it turned out that this inner product (which is just a real number) could be computed simply and efficiently in terms of the original data points, then we wouldn’t have to actually map from $\vec{x} \mapsto \phi(\vec{x})$. Rather, we could simply compute the quantity $K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T \phi(\vec{x}_j)$, and then use the function’s value in Equation (15.10). A kernel function $K$ is such a function that corresponds to an inner product in some expanded feature space.

For example, for 2-dimensional vectors $\vec{x} = (x_1, x_2)$, let $K(\vec{x}_i, \vec{x}_j) = (1 + x_i^T \vec{x_j})^2$. We wish to show that this is a kernel, i.e., that $K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i)^T \phi(\vec{x}_j)$ for some $\phi$. Consider $\phi(\vec{x}) = (1, x_1^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2)$. Then:

$$K(\vec{x}_i, \vec{x}_j) = (1 + x_i^T \vec{x_j})^2 = 1 + x_1^2 + 2x_1x_2 + x_2^2 + x_1^2x_2^2 + 2x_1x_2x_1^2 + 2x_1x_2x_2^2$$

$$= [1 x_1^2 \sqrt{2}x_1x_2 x_2^2 \sqrt{2}x_1 \sqrt{2}x_2]^T [1 x_1^2 \sqrt{2}x_1x_2 x_2^2 \sqrt{2}x_1 \sqrt{2}x_2]$$

What kinds of functions are valid kernel functions (sometimes more precisely referred to as Mercer kernels, because they satisfy Mercer’s condition)? The function $K$ must be continuous, symmetric, and have a positive definite gram matrix. Such a $K$ means that there exists a mapping to a reproducing kernel Hilbert space (a Hilbert space is a vector space closed under dot products) such that the dot product there equals the value of $K$. If you know a fair amount of functional analysis, those last two sentences might have made sense; if you don’t but would like to, you should consult the books on SVMs in the references; and if you’re in neither of those two groups, you can content yourself with knowing that 90% of work with kernels uses one of two straightforward families of functions of two vectors, which do define valid kernels.

These two common families of kernels are polynomial kernels and radial basis functions. Polynomial kernels are of the form $K(\vec{x}, \vec{z}) = (1 + \vec{x}^T \vec{z})^d$. The case of $d = 1$ is a linear kernel, which is what we had before we started talking about kernels (the constant $1$ just changing the threshold). The case
of $d = 2$ gives a quadratic kernel, and is very commonly used. We illustrated the quadratic kernel above. The most common form of radial basis function is a Gaussian distribution, calculated as:

$$K(\vec{x}, \vec{z}) = e^{-\|\vec{x} - \vec{z}\|^2 / (2\sigma^2)}$$

A radial basis function is equivalent to mapping the data into an infinite dimensional Hilbert space, and so we can’t illustrate the radial basis function in the same way as we did for a quadratic kernel. Beyond these two families, there has been interesting work developing other kernels, some of which is promising for text applications. In particular, there has been investigation of string kernels.

The world of SVMs comes with its own language, which is rather different from the language otherwise used in machine learning. The terminology does have deep roots in mathematics, even though most people who use SVMs don’t really understand those roots and just use the terminology because others do and it sounds cool. It’s important not to be too awed by that terminology. Really, we are talking about some quite simple things. A polynomial kernel allows you to model feature conjunctions (up to the order of the polynomial). Simultaneously you also get the powers of the basic features – for most text applications, that probably isn’t useful, but just comes along with the math and hopefully doesn’t do harm. A radial basis function (RBF) allows one to have features that pick out circles (hyperspheres) – although the decision boundaries become much more complex as multiple such features interact. A string kernel lets you have features that are character subsequences of words. All of these are straightforward notions which have also been used in many other places under different names.

15.4 Experimental data

Experiments have shown SVMs to be a very effective text classifier. Dumais et al. (1998) compared a Rocchio variant, Naive Bayes, a more general Bayes Net classifier, Decision Trees, and SVM classifiers on the 10 largest classes in the Reuters-21578 Test Collection, which was introduced in Chapter 13 (page 210). Some of their results are shown in Table 15.1, with SVMs clearly performing best. This was one of several pieces of work that established the strong reputation of SVMs for text classification. Another comparison from around the same time was work by Joachims (1998). Some of his results are shown in Table 15.2. Joachims uses a large number of word features and reports notable gains from using higher order polynomial or RBF kernels. However, Dumais et al. (1998) used MI feature selection (Section 13.5.1, page 204) to build classifiers with a much more limited number of features (300) and got as good or better results than Joachims with just linear SVMs.
Support vector machines and kernel functions

Table 15.1 SVM classifier break-even performance from Dumais et al. (1998). Results are shown for the 10 largest categories and over all categories on the Reuters-21578 data set. Micro- and macro-averaging were defined in Table 13.6 (page 211).

<table>
<thead>
<tr>
<th>Category</th>
<th>NB</th>
<th>Rocchio</th>
<th>BayesNets</th>
<th>Trees</th>
<th>SVM (poly degree = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>earn</td>
<td>92.9%</td>
<td>95.9%</td>
<td>95.8%</td>
<td>97.8%</td>
<td>98.0%</td>
</tr>
<tr>
<td>acq</td>
<td>64.7%</td>
<td>87.8%</td>
<td>88.3%</td>
<td>89.7%</td>
<td>93.6%</td>
</tr>
<tr>
<td>money-fx</td>
<td>46.7%</td>
<td>56.6%</td>
<td>58.8%</td>
<td>66.2%</td>
<td>74.5%</td>
</tr>
<tr>
<td>grain</td>
<td>67.5%</td>
<td>78.8%</td>
<td>81.4%</td>
<td>85.0%</td>
<td>94.6%</td>
</tr>
<tr>
<td>crude</td>
<td>70.1%</td>
<td>79.5%</td>
<td>79.6%</td>
<td>85.0%</td>
<td>88.9%</td>
</tr>
<tr>
<td>trade</td>
<td>65.1%</td>
<td>63.9%</td>
<td>69.0%</td>
<td>72.5%</td>
<td>75.9%</td>
</tr>
<tr>
<td>interest</td>
<td>63.4%</td>
<td>64.9%</td>
<td>71.3%</td>
<td>67.1%</td>
<td>77.2%</td>
</tr>
<tr>
<td>ship</td>
<td>49.2%</td>
<td>85.4%</td>
<td>84.4%</td>
<td>74.2%</td>
<td>85.6%</td>
</tr>
<tr>
<td>wheat</td>
<td>68.9%</td>
<td>69.7%</td>
<td>82.7%</td>
<td>92.5%</td>
<td>91.8%</td>
</tr>
<tr>
<td>corn</td>
<td>48.2%</td>
<td>65.3%</td>
<td>76.4%</td>
<td>91.8%</td>
<td>90.3%</td>
</tr>
<tr>
<td>micro-Avg Top 10</td>
<td>64.6%</td>
<td>81.5%</td>
<td>85.0%</td>
<td>88.4%</td>
<td>92.0%</td>
</tr>
<tr>
<td>micro-Avg All Cat</td>
<td>61.7%</td>
<td>75.2%</td>
<td>80.0%</td>
<td>N/A</td>
<td>87.0%</td>
</tr>
</tbody>
</table>

Table 15.2 SVM classifier break-even performance from Joachims (1998). Results are shown for the 10 largest categories on the Reuters-21578 data set.

This mirrors the results discussed in Chapter 14 (page 226) on other linear approaches like Naive Bayes. At a minimum, it seems that working with simple word features can get one a long way. It is also noticeable the degree to which the two papers’ results for other learning methods differ. In text classification, there’s always more to know than simply which machine learning algorithm was used.

These and other results have shown that simple classifiers such as Naive Bayes classifiers are uncompetitive with classifiers like SVMs when trained

Preliminary draft (c) 2007 Cambridge UP
and tested on independent and identically distributed (i.i.d.) data, that is, uniform data with all the good properties of statistical sampling. However, these differences may often be invisible or even reverse themselves when working in the real world where, usually, the training sample is drawn from a subset of the data to which the classifier will be applied, the nature of the data drifts over time rather than being stationary, and there may well be errors in the data (among other problems). For general discussion of this issue see Hand (2006). Many practitioners have had the experience of being unable to build a fancy classifier for a certain problem that consistently performs as well as Naive Bayes.

15.5 Issues in the categorization of text documents

15.6 References and further reading

There are now a number of books dedicated to SVMs, large margin learning, and kernels of which currently the two best are probably Schölkopf and Smola (2001) and Shawe-Taylor and Cristianini (2004). Well-known, good article length introductions are Burges (1998) and Chen et al. (2005), the latter of which introduces the more recent $\nu$-SVM, which provides an alternative parameterization for dealing with inseparable problems, whereby rather than specifying a penalty $C$, one specifies a parameter $\nu$ which bounds the number of examples which can appear on the wrong side of the decision surface. For the foundations by their originator, see Vapnik (1998). Other recent, more general books on statistical learning also give thorough coverage to SVMs, for example, Hastie et al. (2001).

The kernel trick was first presented in (Aizerman et al. 1964). For more about string kernels and other kernels for structured data, see (Lodhi et al. 2002, Gaertner et al. 2002). The Advances in Neural Information Processing (NIPS) conferences have become the premier venue for theoretical machine learning work, such as on SVMs. Other venues such as SIGIR are much stronger on experimental methodology and using text-specific features to leverage performance.

A recent comparison of most current machine learning classifiers (though on problems rather different from typical text problems) can be found in Caruana and Niculescu-Mizil (2006). Older examinations of a more limited set of classifiers on text classification problems can be found in (Yang 1999, Yang and Liu 1999, Dumais et al. 1998). Joachims (2002a) presents his work on SVMs applied to text problems in detail. Zhang and Oles (2001) have insightful comparisons of Naive Bayes, regularized logistic regression and SVM classifiers.