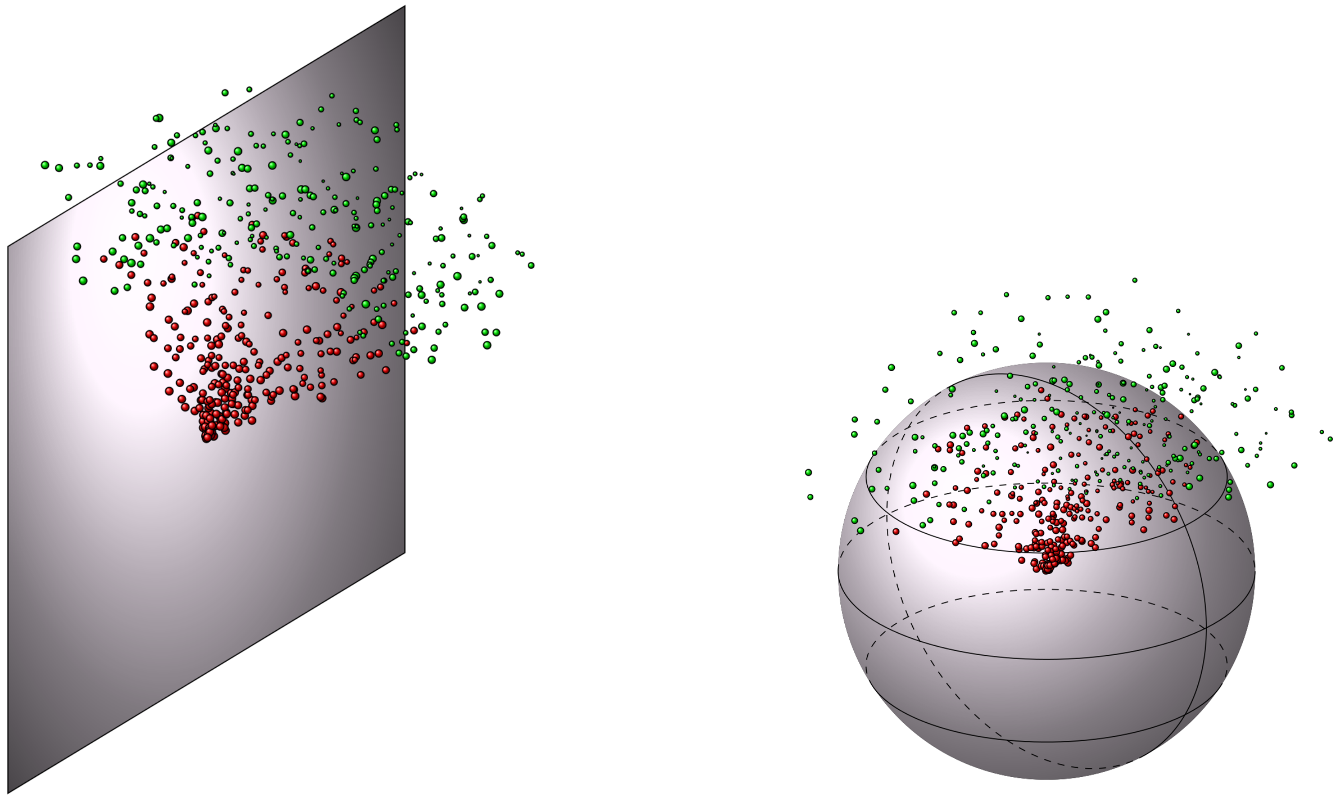
Support vector machine parameter description

Israel Vaughn

July 13 , 2011

1 Support vector machines (what are they?)

Support vector machines are conceptually simple. Suppose we have data points in some n dimensional space, each one assigned to a class in the set $\{+1, -1\}$. Now suppose that we want to construct an $n-1$ dimensional surface (called a manifold) that separates the data in the two classes as best as possible. The first surface to try is just an $n-1$ dimensional plane. The plane is flat, so will often be a poor separating surface (if for example a spherical surface better separates the classes).



(a) Hyperplane separation

(b) Sphere separation

Figure 1: A set of points for two different classes in three dimensions, (🟢) is the -1 class and (🔴) is the $+1$ class. Note that for this example the sphere is the best possible surface which separates the two classes, while a plane can never separate the classes very well, no matter the orientation of that plane.

Figure 1 demonstrates this issue, notice that with the data displayed no plane can separate the two classes, only a spherical surface (or something similar) can separate the classes.

It may be difficult to derive a surface that separates data classes, especially in spaces with greater than three dimensions as visualization is difficult. Support vector machines (SVMs) are one technique with which to derive a suitable surface. SVMs construct this surface in the data space by employing a kernel function, which maps an n dimensional point into an (usually) infinite dimensional space. This is denoted the dual space. Essentially points can almost always be separated in the infinite dimensional space, and the inverse map of that separation surface generates a complicated surface in the data space. The "support" part of support vector machines comes from the fact that SVMs use the vectors themselves to define the surface. I will not delve into details about how SVMs are constructed or why they are mathematically valid, see Cortes [3] for the SVM implementation (or Burges [2] for a detailed analysis) and Lax [4] or Stakgold [6] for the general mathematical functional analysis details.

Note that algorithms like principal components analysis (PCA), linear discriminant analysis (LDA), and empirical orthogonal function (EOF) are all examples of algorithms that can only separate data with a plane as shown in Fig. 1a. These are all examples of *linear* classifiers. If a finite set of parameters is being estimated (i.e. more than two classes but not an infinite number of classes) the most general linear estimator is called the *Wiener estimator* which includes statistical classification. PCA, matched filters, EOF, and LDA are all special cases of the Wiener estimator.

2 SVM parameters

First, there are two ways to compute SVMs. SVMs can be reduced to a quadratic programming problem :

$$\frac{1}{2}\left\|\sum_{i=1}\alpha_i\phi(\mathbf{x}_i)\right\|^2 - \sum_i y_i\alpha_i \text{ s.t.} \begin{cases} \sum_i \alpha_i = 0 \\ 0 \le y_i\alpha_i \le C \end{cases} \tag{2.1}$$

the above equation is minimized over $\boldsymbol{\alpha}$, where the inner product $\big(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\big)$ is the kernel function, the $y_i \in \{-1, 1\}$s are the labels, the $\mathbf{x}_i$s are the data vectors or points, and C is the box constraint. Note that this minimization is done in the dual (or kernel) space rather than in the data space. SVM constructs a surface which maximizes the distance between the two classes, this is denoted the *maximal margin hyperplane*, and this hyperplane is in the dual or kernel space. Note that the hyperplane corresponds to some *non-planar* surface (manifold) in the data space.

The two parameters that can then be adjusted are the box-constraint, C, and the kernel function, $\big(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\big)$. Nearly always, given enough time, SVM training can find a surface in an infinite dimensional kernel space which completely separates two classes. This, however, often leads to poor performance on subsequent data (which was not trained on originally with SVM) because data always has statistical errors and SVM is a non-probabilistic learning machine. Basically, SVM finds a surface which works perfectly for the given data, but is too specific, and results in many mis-classifications on subsequent data. This is called over-training, and is a result of a low number of training samples not properly representing the underlying true probability distribution. An analogy from Vapnik [7] elucidates this point :

> A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist's lazy brother, who declares that if it's green, it's a tree. Neither can generalize well. The exploration and formalization of these concepts has resulted in one of the shining peaks of the theory of statistical learning (Vapnik, 1979).

Where "capacity" corresponds to overtraining or undertraining.

This issue is what the C parameter is used for, it basically determines the penalty for mis-classified data points during the SVM training phase. If C is small, then the penalty is small, and many points can be mis-classified, if C is large then fewer points will be mis-classified, but the margin distance may shrink. This is equivalent to a kind of regularization parameter. This description is not completely correct, as I failed to discuss how the *soft margin* parameter enables the mis-classification, but it complicates the description. For details, see [3].

The other parameter is the kernel function. This function determines the kinds of shapes that are generated in the data space. In real world use, most people have good results using the Gaussian kernel :

$$\big(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\big) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2} = e^{-(\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i\cdot\mathbf{x}_j)/2\sigma^2} \tag{2.2}$$

also commonly denoted a radial basis function (RBF) in the literature. The Gaussian is parameterized with σ alone, which makes it easy to characterize.

These parameters are often searched over to generate a reasonable or best-possible SVM for classification later on. A Gaussian kernel is often used, which results in a two dimensional parameter search space : (σ, C).

Note that the input into a trained SVM is a set of vectors or points with some dimension, n. The SVM then maps each input vector to some number. When SVM is trained, it attempts to map points to -1 or $+1$ using a decision surface, but in general the output is a real number, i.e. when an arbitrary point is input into the trained SVM, the output is some real number denoted the *test statistic*, λ. To classify a point, some threshold is defined, c_t, and when $\lambda > c_t$ the point is considered to be in class $+1$ and when $\lambda \le c_t$ the point is considered to be in class -1. An ROC curve can then be generated by taking a set of points, with known classes, and varying c_t while plotting the *true positive rate* (aka the probability of detection, the sensitivity, the hit rate, etc.) vs the *false positive rate* (aka the false alarm rate, the fall out, 1 - specificity, etc.).

3 SVM training and implementation

In order to prevent the over-training described in the above section the following procedure is usually followed :

1. The original dataset is separated into two sub-sets, a holdout set and the training set. This is done by a uniformly distributed random assignment. Typically the holdout set is 20% of the original dataset and the training set is 80% of the original training set.

2. The training set (80% of the original data) will be subsequently divided into k smaller sets, again by a uniformly distributed random assignment. These are commonly referred to as folds. Typically $k = 10$ is used since this number generally yields the best results, from the SVM community's empirical experience.

3. Once the k folds are created, $k - 1$ folds are combined into one set for training, while the remaining fold is used for testing the trained SVM (i.e. no points in the remaining fold are used in SVM training), this can be done for each fold so an SVM is trained a total of k times. Each trained SVM can then be used to classify the withheld fold, and some performance measure can be applied to the classified points. The performance measure can then be averaged over the k classified folds to get a reasonably unbiased performance measure for particular parameters used for the SVM.

4. The above is done for a set of parameters $\left(\{\sigma_1, \sigma_2, ..., \sigma_m\}, \{C_1, C_2, ..., C_n\}\right)$, and the parameters with the best average performance measure are selected for use. The SVM is then re-trained on the entire training set using the chosen parameters.

5. The trained SVM from above is then used on the 20% of the data held out at the beginning, and the performance measure calculated. This performance measure is the most likely real world performance measure.

4 Iterative SVM

The quadratic programming problem posed above uses a Hessian matrix in practice. Suppose that the number of data points in the training set is N, then the size of the Hessian matrix will be N^2. This severely limits the size of the training database, with 8GB of RAM only $\sim 13,000$ sixteen dimensional data points can be used in the training set for the Matlab SVM training implementation.

Most classification algorithms and statistical observers can be recast as an iterative problem. Sequential minimal optimization (SMO) using the Karush-Kuhn-Tucker (KTT) conditions is used to train an SVM iteratively. Google yields plenty of papers on these two topics, I won't go into details here.

The iterative implementation introduces three other parameters, the maximum allowed iterations, the tolerance, and the KTT violation parameter. The maximum allowed iteration parameter is self explanatory. The tolerance is the number used to compare the difference between the previous iteration to the current iteration, if the SVM is not changing much then the algorithm can be terminated, i.e. it is the convergence tolerance parameter. The KTT conditions are a set of conditions (inequalities) that many locally linear optimization problems must satisfy, whether they are globally linear or not (see `http://www.svms.org/kkt/`), also see [5] for details. Basically the amount of violation of the KTT conditions and the margin of the hyperplane result in a tradeoff. Note that setting a maximum iteration value is equivalent to regularizing the optimization problem.

5 Performance measures

For statistical two set classification, an ROC curve displays the most general performance information. There does exist a "best" ROC curve, which is the ROC curve produced by the theoretical ideal observer (theoretically optimal classifier), for details see [1]. It is provable that no other ROC curve will be higher, and to the left, of the ideal observer's ROC curve, see Fig. 2. Ideal observers are, however, difficult to find or compute, as they require knowledge of statistical parameters that are usually unknown. In the case where an ideal observer cannot be computed, or if it is not known if a particular classifier is the ideal observer, some performance measure must be used to compare observers (classifiers).

Table 1 gives an overview of most of the common performance measures. In light of the specific task at hand (high sensitivity to false positives), I suggest we use something similar to AUC, but with a false positive allowance. That is, the performance measure would be to integrate the ROC curve, similar to computing the AUC, but instead of stopping the integration at $FPF = 1$, the integration can be stopped at some acceptable false positive rate, e.g. $FPF = 0.3$. This would allow for maximization of performance of the ROC curve before the $FPF = 0.3$ point. So we would have

$$AUC_a = \int_0^a ROC(FPF), \tag{5.1}$$

for some a used as the performance measure. I believe that this kind of performance measure will be more robust than using positive or negative predictive values or the Neyman-Pearson measure because it uses more than a single point on the ROC curve.

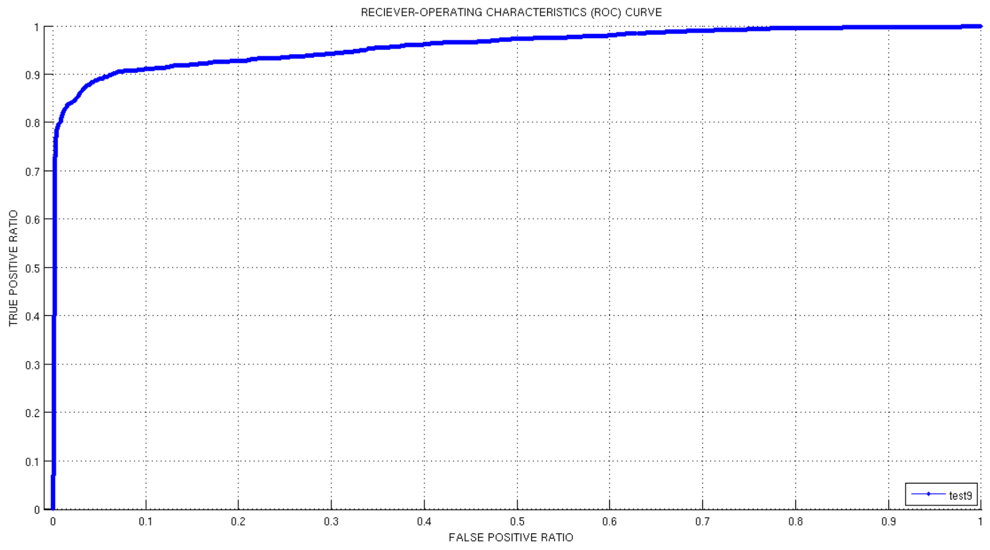


Figure 2: An example ROC curve. If this was the ROC curve for an ideal observer, no other observer (or classifier) could have a y value greater than the graph at any x value.

Performance Measure	Definition	Comments
Accuracy	$\lim_{N\to\infty} \frac{N_{\mathrm{TP}} + N_{\mathrm{TN}}}{N}$	Poor measure; generally only works well for datasets with approximately equal class sizes
Positive predictive value	$\lim_{N\to\infty} \frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}} + N_{\mathrm{FP}}}$	A Bayesian style measure
Negative predictive value	$\lim_{N\to\infty} \frac{N_{\mathrm{TN}}}{N_{\mathrm{TN}} + N_{\mathrm{FN}}}$	A Bayesian style measure
Neyman-Pearson criterion	Maximizes the true positive fraction (TPF) at a specific false positive fraction (FPF) $TPF = \lim_{N\to\infty} \frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}} + N_{\mathrm{FN}}}$ and $FPF = \lim_{N\to\infty} \frac{N_{\mathrm{FP}}}{N_{\mathrm{FP}} + N_{\mathrm{TN}}}$	A reasonable measure for any given false positive fraction (rate). This is a single point on the ROC curve.
Area under the ROC curve (AUC)	$\int_0^1 ROC(FPF)$	Here FPF is the false positive fraction and $ROC(FPF)$ is the true positive fraction value on the ROC curve at a specific FPF value. A reasonable measure that is often used. Corresponds exactly to $\frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{SNR}{2}\right)$ when the test statistic is normally distributed for both classes.

Table 1: Performance measures for binary classification, mostly taken from Barrett and Myers [1]. N is the total number of points, N_{TP} is the number of true positives, N_{TN} is the number of true negatives, N_{FN} is the number of false negatives, N_{FP} is the number of false positives, and SNR is the signal to noise ratio.

References

[1] Harrison H. Barrett and Kyle J. Myers. *Foundations of Image Science*. Wiley-Interscience, Hoboken, New Jersey, United States, 2004.

[2] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[3] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[4] P. D. Lax. *Functional Analysis*. Wiley-Interscience, New York, New York, 2002.

[5] Alex J. Smola and Bernhard Schlkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004. 10.1023/B:STCO.0000035301.49549.88.

[6] Ivar Stakgold. *Boundary value problems of mathematical physics*. SIAM, 2000.

[7] Vladimir Naumovich Vapnik. *Estimation of Dependences Based on Empirical Data [translated from Russian]*. Springer Verlag, New York, 1982.