Evaluation of Kernel Functions for Online Multiclass Classification

Benjamin Lambert
Department of Computer Science
University of Illinois at Urbana-Champaign

May 10, 2004

Abstract

As it is not clear how to choose an appropriate kernel function for a specific learning problem, this paper examines the performance of three kernel functions with varied parameters for online handwritten character recognition. The three kernel functions evaluated are the polynomial, complete polynomial, and radial basis functions. Results show that the three functions perform comparably with the appropriate parameters.

1 Introduction

Kernel functions enable learning algorithms to efficiently learn linear discriminants in very high dimensional spaces. The choice of kernel function defines the feature space in which the inner product is computed. Since the examples being classified may or may not be separable in the higher dimension space the choice of kernel function is important. Some kernel functions simulate higher dimension spaces where the classifier is able to generalize better. This paper empirically examines the performance of several commonly used kernel functions.

The learning algorithm presented in this paper learns linear functions. Linear functions can be used for classification. Binary classification of points, say as positive or negative examples, is very naturally modeled by a linear function. If a point lies on the positive side of the function then it is classified as a positive example, otherwise it is classified as a negative example. For example, in the two dimensional space of a white-board, the function $y < 7ft.$ could be the linear function that classifies portions of a white-board that I can reach.

More interesting classifications can also be made by linear discriminants. For example, handwritten character recognition has been modeled effectively with linear discriminant functions. For this type of classification instances are mapped by d feature functions into some d dimensional space. For character recognition, a feature function might be $f \rightarrow \Re$, where $\Re$ is the light intensity of pixel #12. In the d dimensional space, a linear discriminant for classifying the letter *m*, ideally

would be a linear function where examples of the letter *m* are mapped to points on one side of the function and all other examples map to points on the other side of the function.

For complex classifications, there may not exist a linear function that separates two classes of points. One solution to this problem is to map the examples into a higher dimensional space where the examples are separated by a linear function. One way to map the examples into a higher dimensional space is to engineer new features. Experts could look at the examples and observe that most examples of the letter *m* have very high light intensity at both pixel 12 and pixel 29. Based on this observation, a new dimension could be introduced into the feature space; perhaps the new dimension would be the product of the intensities of pixels 12 and 29.

Engineering features manually is difficult, expensive, and may be unfeasible in some situations. Instead, generic features could be used with the hope that some are indeed meaningful. For example, in addition to the original features, the product of each pair of features could be used as a feature. These generic features can be defined by a function that maps examples into a higher dimensional space. The function that maps examples from features to products of pairs of feature is $f : (x_1, \ldots, x_n) \rightarrow (x_1x_2, x_1x_3, \ldots, x_2x_3, x_2x_4, \ldots, x_nx_{n-1})$.

To make the examples linearly separable, it may be necessary to map the features into a space with very high dimensionality. Some of the functions described in this paper effectively map features into a space with dimensionality well over 10^{20}. Regardless of time complexity of the learning algorithm, even the space requirement for storing the hypothesis as a weight vector in such a high dimensional space makes this feature mapping infeasible. For algorithms like perceptron, the hypothesis is computed as the vector sum of the misclassified examples. Classifications are made by taking the inner product of the weight vector with the example to be classified. Since the weight vector is fully specified by the misclassified examples, the algorithm does not need to store the weight vector in memory. Some mappings of feature spaces allow dot products in the high dimensional space to be computed without actually mapping examples into the space. The dot products are instead computed with a kernel function.

A given kernel function specifies a particular mapping, so choosing a kernel function is equivalent to choosing a mapping to new space. The selection of a kernel function is important as the data might not be separable by a linear function in the new space. Even if the data are separable in the new space, some kernel functions will yield hypotheses that generalize better on future examples. So choosing the right kernel function is key.

For many learning problems, the best choice for a kernel function is often not known. Many classes of kernel functions also have parameters that affect separability and generalization. The experiments described in this paper test the accuracy of three commonly used classes of kernel functions on handwritten character recognition. As there is unlikely to be some panacea kernel function that makes all learning problems easily linearly separable, the choice of kernel is application specific. The goal of this paper is to find which kernel and what parameters are best for handwritten character recognition.

The results show that the three function classes perform comparably. Polynomial and complete polynomial kernel function perform best with degree between three and six, and the radial basis function performs best with smoothing parameter 2.5. Results also show that simple online learning for the particular data set is not good enough, and would benefit from being taken offline.

2 Online Multi-class Classification

The goal of this paper is to find which kernel functions best perform recognition of handwritten characters. However, the techniques used to do this can be applied to classification of an arbitrary entity that can be represented as a vector of real numbers. Generally speaking, classifications are usually among objects that are all members of some common larger class (e.g. handwritten characters). The larger common class can be broken down into two or more subclasses. These subclasses are the classifications that are desired.

Instances of the encompassing common class are represented as feature vectors. Each dimension of the vector corresponds to some aspect of the instance (e.g. the light intensity of pixel 35) and has a real numbered value. Instances all have the same number of features, denoted by n.

Definition 1 (Instance) *An instance or example is an n dimensional vector.*

$$\overrightarrow{x} = e \in \Re^n$$

For a particular classification problem, each instance belongs to a class. A class is a multiset that contains zero or more examples.

Definition 2 (Class) *A class c is multiset of examples.*

$$e \in c \quad \textit{if and only if e is an example and the class of e is c}$$

For a particular problem, there is generally some finite set of classes, denoted as C. Thus $e \in c \in C$.

A classifier is a mapping from an instance to a class. For example, a classifier of handwritten characters would take an image of a handwritten character as input and map that image to the name of a letter (e.g. *m*). The classes in the range of the classifier function are unique human-determined class.

Definition 3 (Classifier) *A classifier is a function which maps each example $\overrightarrow{x}$ to a class c.*

$$m : \overrightarrow{x} \rightarrow c$$

Ideally, the learning algorithm would learn a classifier that maps each example to the class it belongs, which this definition of classifier does not require. The goal is to learn an approximation of a perfect classifier. The actual class to which a class belongs is denoted as y_i.

Definition 4 (Perfect Classifier) *A perfect classifier maps each example to the class it is a member of.*

$$p(\overrightarrow{x}) = c \quad \textit{if and only if} \quad \overrightarrow{x} \in c$$

equivalently,

$$p(\overrightarrow{x}) = y$$

A perfect classifier that maps from images of handwritten characters to character names would be impossible to program manually. An approximation of this mapping can be induced using machine learning algorithms by looking at many examples of handwritten characters where the class labels are known.

The definition of online classification is that examples are classified one-by-one, and after each classification is made the correct label is given to the classifier. If the predicted class was incorrect the hypothesis is updated before classifying the next example. This is opposed to batch classification where a large set of examples are all classified at once, then class labels are provides after all the classifications have been made.

The classifications described in this paper are multi-class which simply means that there are more than two classes or $|C| > 2$. The distinction is important because binary classifications are made very naturally with linear discriminants, which divide the feature space into exactly two subspaces, whereas a single function is not sufficient to divide a space into more than two subspaces.

Online Multi-class Classification without Kernels

The standard offline perceptron algorithm learns a linear discriminant after making no more than R^2/γ^2 mistakes assuming the data are linearly separable (Novikoff). The classifier learned by the perceptron algorithm is a binary classifier. The form of the classifier produced by the perceptron algorithm is

$$f(\overrightarrow{x}) = sign(\overrightarrow{x} \cdot \overrightarrow{w} + \beta)$$

for a learned weight vector w and a learned bias β.

The perceptron decision rule is inherently binary as the range of the $sign$ function is size two. The perceptron learning algorithm can be extended to do multi-class classifications by maintaining a weight vector for each class and updating all vectors for every mistake. This is effectively learning $|C|$ binary classifiers, each discriminating its corresponding class from all the rest (sometimes called "one versus all"). The classifiers are not perfect so for a given example it is possible that none of the classifiers map to positive or that more than one classifier maps to positive. So the decision rule must be modified. One rule which is often used, is to compute which hyperplane is nearest to the example. This is formulated as:

$$argmax_i \, (\overrightarrow{x} \cdot \overrightarrow{w}_i + \beta_i)$$

where $\overrightarrow{w}_i$ is the weight vector corresponding to class i and β_i is the corresponding bias weight. The perceptron update rule can be modified to not update all of the weight vectors but only some of the vectors [CS01].

Online Multi-class Classification with Kernels

If the data are not linearly separable, then a function can be used to map the data into a space where the data are separable. The function used to map examples into a high dimension space is referred to as $\phi(\overrightarrow{x})$. If the function $\phi(\overrightarrow{x})$ is appropriately chosen then the inner product $\phi(\overrightarrow{x}) \cdot \phi(\overrightarrow{y})$ can

be computed with a kernel function $K(\overrightarrow{x}, \overrightarrow{y})$ without actually mapping the examples into the new space.

Definition 5 (Kernel Function)

$$K(\overrightarrow{x}_i, \overrightarrow{y}_i) = \phi(\overrightarrow{x}_i) \cdot \phi(\overrightarrow{y}_i)$$

The weight vector computed by the standard perceptron algorithm can be written as a sum over the examples that were misclassified. Specifically the weight vector w output by the perceptron algorithm is:

$$\overrightarrow{w}_i = \sum_{x_i \in mistakes} y_i \overrightarrow{x}_i$$

where y_i is the actual class which $\overrightarrow{x}_i$ belongs to and the classes are the integers 1 and -1.

The key observation is that by distributing the inner product between $\overrightarrow{x}_i$ and $\overrightarrow{w}_i$ over the sum, the inner product with the weight vector can be computed by summing the the inner product with each misclassified example. Thus the perceptron decision rule can be rewritten as:

$$\beta \leq \sum_{x_i \in mistakes} y_i \overrightarrow{x} \cdot \overrightarrow{x}_i$$

In this form the inner products inside the summation can be replaced with vectors mapped into a higher dimension:

$$\beta \leq \sum_{x_i \in mistakes} y_i \phi(\overrightarrow{x}) \cdot \phi(\overrightarrow{x}_i)$$

The inner product can then be substituted with the kernel function as it was defined.

$$\beta \leq \sum_{x_i \in mistakes} y_i K(\overrightarrow{x}, \overrightarrow{x}_i)$$

With this new decision rule for perceptron, the multi-class classification method can be reformulated as:

$$argmax_i \sum_{m_j \in mistakes} y_i K(\overrightarrow{x}, \overrightarrow{m}_j) + \beta_i$$

Three Classes of Kernel Functions

Learning Kernel Classifiers [Her02] presents four commonly used classes of kernel functions: polynomial, complete polynomial, radial basis, and Mahalanobis. The first three kernels are examined in this paper. The Mahalanobis kernel is similar to the radial basis kernel except there is are n parameters, scaling factors for each dimension.

The polynomial kernel function simply takes the inner product of the two vectors and raises the result to the power p. Each dimension in the feature space that the kernel's implied $\phi(x)$ is the product of p of the features. The feature space has $\binom{n+p-1}{p}$ dimensions. The degree to use depends on the function that is being learned. Experiments in section 3 evaluate the performance with degrees 1 through 12 for handwritten character recognition.

Definition 6 (Polynomial Kernel Function)

$$K(\overrightarrow{x}, \overrightarrow{y}) = (\overrightarrow{x} \cdot \overrightarrow{y})^p$$

The complete polynomial kernel is the same as the polynomial kernel function except that there is a constant added to the dot product. The constant can be tweaked to improve performance. The experiments described in this paper c was fixed at 1. The feature space corresponding to this kernel function has $\binom{n+p}{p}$ dimensions. Again the degree depends on the function being learned, section 3 evaluates the performance with degrees 1 through 12 for handwritten character recognition.

Definition 7 (Complete Polynomial Kernel Function)

$$K(\overrightarrow{x}, \overrightarrow{y}) = (\overrightarrow{x} \cdot \overrightarrow{y} + c)^p$$

The radial basis function is quite a bit different from the last two classes of kernels. The value for σ depends on the function that is being learned. Experiments in section 3 evaluate the performance with sigma between 1.0 and 10.0 for handwritten character recognition. The dimension of the space corresponding to this kernel is infinite.

Definition 8 (Radial Basis Kernel Function)

$$K(\overrightarrow{x}, \overrightarrow{y}) = e^{-\frac{|\overrightarrow{x} - \overrightarrow{y}|^2}{2\sigma^2}}$$

It is not clear what to expect. The training examples are linearly separable for the polynomial kernel function with degree greater than two [Vap98], so they would also be linearly separable for the complete polynomial function with degree greater than two.

3 Kernel Functions for Handwritten Character Recognition

Even though digital communication continues to prevail over analog communication (e.g. e-mail replacing handwritten letters), manual handwriting is often more natural than using a digital computer input. One example of this is when writing a letter. For most people it is far easier to take a pen to an envelope than set up a printer to print on envelopes. Even if sending postal mail were conveniently automated, other sources of handwriting, such as taking notes in class, might never cease to exist.

The are many situations where automatic translation from written language to computer readable character strings is beneficial. For example the United States Postal Service uses character recognition to identify handwritten zip codes to route mail. Another reason to convert written language to a digital character representation is that written language cannot be indexed and thus cannot be searched.

Machine learning algorithms have been used, often with high accuracy, to construct classifiers of handwritten character. Russell and Norvig in their AI textbook [RN03] report that the most successful attempt to automate handwritten digit (0-9) recognition was with a "virtual support

vector machine." The virtual SVM's error rate on the test set of the NIST handwritten digit database was 0.56%.

The performance on character recognition of the kernel functions described in the previous section was evaluated with the USPS handwritten digit data set. This data set consists of images of handwritten numbers (0-9), so there are 10 classes (one for each decimal digit). Each image is grayscale sixteen pixel by sixteen pixel image. Each pixel is a feature that has a real valued number, corresponding to the light intensity of each pixel.

Although kernel perceptron is an online algorithm, to evaluate accuracy, online algorithms are usually taken offline and tested on a test set where mistakes do not cause a hypothesis update. The USPS data set consists of 7,291 training images and 2,007 testing images.

The primary measure of performance for each kernel and parameter is the accuracy on the training set and on the testing set. The number of misclassified examples, also called support vectors, after the training stage are also shown for each algorithm. Fewer support vectors is better because to classify an new image, the algorithm must compute the kernel function for each support vector. The time to classify an image is proportional to the number of support vectors. The last quantification of performance is the learning rate. This is an online algorithm, so the performance while the algorithm is online (during training) is more important than the performance when taken offline (for testing). It is the learning rate that we would like to maximize. We would like the algorithm to make most of the mistakes early, and have better accuracy for the remainder of the time online. The learning rates are plotted by how many of the classification have been erroneous throughout the duration of training.

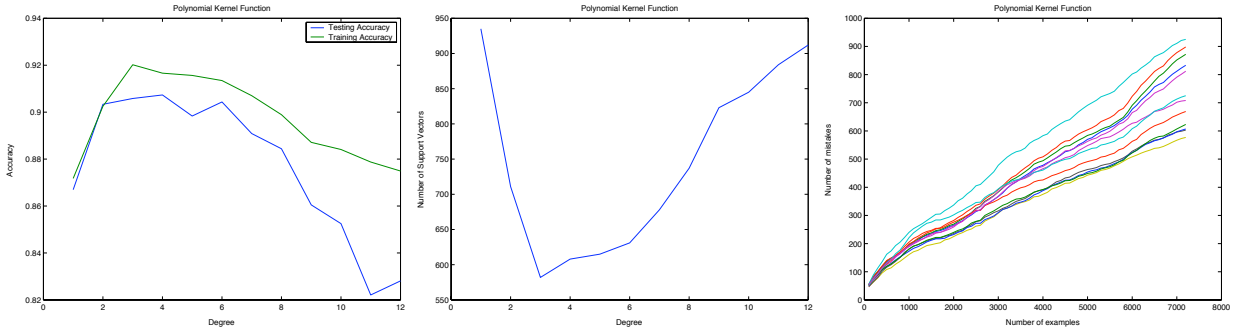
The three kernel functions used, polynomial, complete polynomial, and radial basis, for the most part were comparable to eachother. The performance of each kernel function varied for different parameters.

Polynomial Kernel

The highest accuracy on the testing set for perceptron with a polynomial kernel was for a polynomial kernel function with degree between two and six. Accuracy on the training data peaked at degree three. Likewise the number of support vectors is minimized for degree three. This indicates that the additional support vectors for other degrees are extraneous and detrimental to classification. It is not clear why accuracy is better for degree between 2 and 6, but it is clear that performance is impaired at high degree. For example, when using a polynomial kernel of degree eleven the testing accuracy is nearly ten percent less than for degree four.

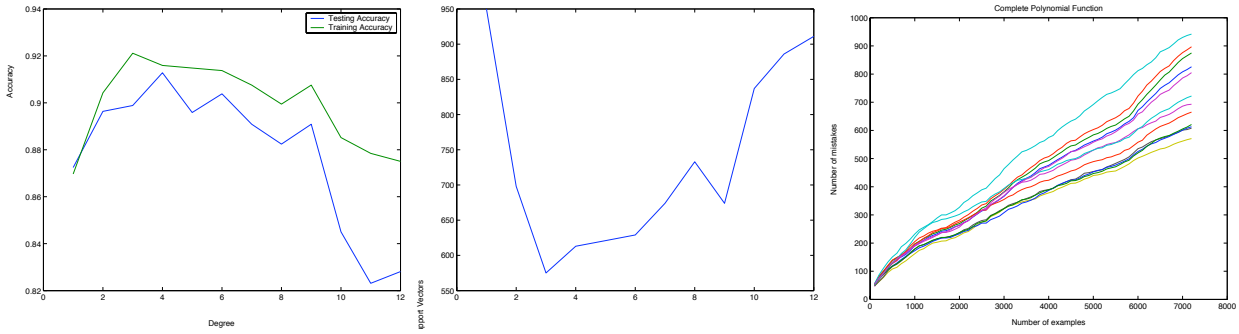
Another interesting phenomenon in the data is large difference in the number of support vectors for each degree. The number of support vectors is minimized when the degree is three. The number of support vectors nearly doubles for degrees one and twelve.

Using a polynomial kernel of degree one is the same as normal perceptron. Testing accuracy at degree four is over three percent higher than degree one, so the kernel is clearly helping with accuracy.



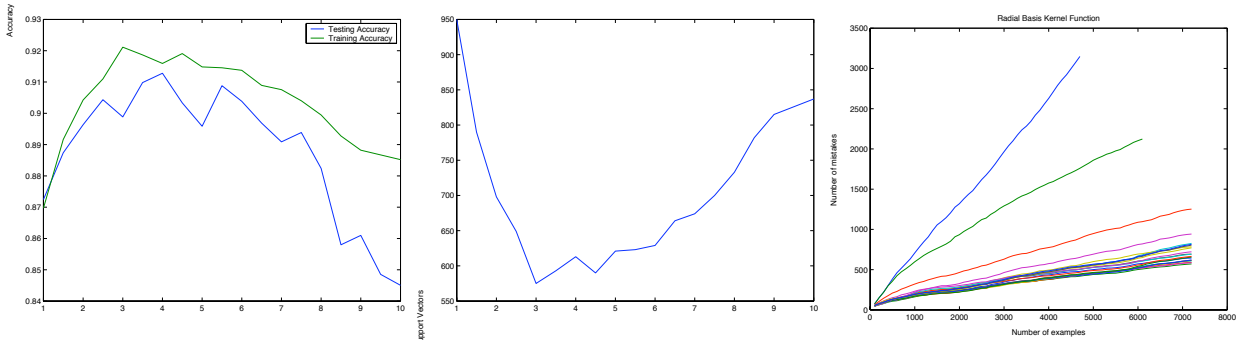
Complete Polynomial Kernel

The results when using a complete polynomial kernel function were largely the same as for the regular polynomial kernel function. This may be because of the choice of the c parameter. There are a few noticeable differences, such as a spike in accuracy at degree nine. There curves are not as smooth as the curves for the polynomial kernel.



Radial Basis Function Kernel

The parameter that can be adjusted for the radial basis function is the divisor in the exponent. Herbrich describes this value as a smoothing parameter, where larger values make the function smoother. On the testing data, this function has the highest accuracy for $\sigma = 3.5 - 4.0$. Accuracy for $2.5 \leq \sigma \leq 5.5$ was comparable, but fell for higher and lower values.



Summary of Results

The main conclusion that can be drawn from these results is that to classify handwritten digits with kernel perceptron low degree (around four to six) is best for polynomial and complete polynomial kernel functions, and a smoothing parameter around three is appropriate for radial basis kernels. The linearity of the learning rates indicates that the algorithm is not close to convergence after classifying all 7,291 examples in the training set. As described in the following section, convergence could be achieved by taking the algorithm offline.

Another way to increase the performance is to lower the update threshold. Regular kernel perceptron only updates the hypothesis when a mistake is made. Alternatively, the hypothesis can be updated whenever an example is within some margin of the current classifier; the idea is to proactively find a hyperplane with a large margin. Crammer et. al. use this strategy to achieve better accuracy [CSK04].

4 Related Research

Other research achieves higher accuracy on the same data set by taking the algorithm offline. With the data offline, the training set can be classified several times before testing. Freund and Schapire show that training for several epochs improves performance [FS98].

Another way to find a good linear discriminant with the examples offline is to formulate a linear program where the solution is an optimal (by margin or margin distribution) linear discriminant. This approach, used in support vector machines, can classify the testing set with error as low as 4.0% (for a degree three polynomial kernel function) [Vap98]. Vapnik also reports that the training set is not linearly separable, as an optimal classifier misclassifies 340 examples. The optimal linear discriminant in the space corresponding to a polynomial kernel of degree two misclassifies four examples. The data are separable in the space corresponding to a kernel of degree three.

If the learning is done offline, the support vectors specifying the hyperplane can be found by solving a linear or quadratic program. These support vectors fully specify the hyperplane, so all other examples can be discarded. If the learning is online, we do not have the luxury of discarding irrelevant examples. Since the time to make a classification is proportional to the number of support vectors it may be necessary to discard examples if classifications are to be made quickly. Crammer and Singer discard examples that are more than some threshold distance away from the current hypothesis hyperplane [CSK04].

5 Future Research

It is not clear how best to perform multi-class classifications. The solution to this problem used in this paper was to learn a linear discriminant for each class, then classify examples depending on which hyperplane they are closest to. Each weight vector can be though of as a prototype or ideal instance of that class. This assumes that all examples are derivatives of some ideal example, and are thus in geometric proximity. For character recognition, it may be the case that there are two

particular styles of writing some character. If this is indeed the case, then there is not a one-to-one correspondence between the human notion of a character and subspace of the feature space.

A statistical analysis of the distribution of known class members could be performed to determine if the human-defined class maps to multiple regions of the feature space. Some criteria, such as a that for splitting clusters in the k-means algorithm (i.e. ISODATA) could flag the need to create two versions of a class. The examples could then be clustered into two classes both representative of the original human-defined class. This clustering could be performed by k-means or some other clustering algorithm appropriate to the geometry of the feature space.

Splitting classes that appear to be mixtures of distributions would seem particularly suited for use in algorithms using kernel methods. Since class hypotheses are sets of misclassified examples, on each misclassification the class can be checked for high variance and then clustered with a clustering algorithm. The new clusters are then assigned new class labels, and a mapping is created that maps those class labels back to the original class labels.

6 Acknowledgements

This project would not have been possible without the help of the UIUC architecture cluster which is funded by grant XYZ.

References

[CS01] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. In *Conference on Computational Learning Theory COLT*, volume 2111, pages 99–115. Springer, Berlin, 2001.

[CSK04] Koby Crammer, Yoram Singer, and Jaz Kandola. Online classification on a budget. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[FS98] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 209–217, 1998.

[Her02] Ralf Herbrich. *Learning Kernel Classifiers*. The MIT Press, Cambridge, Massachusetts, 2002.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Sadle River, New Jersey, 2003.

[Vap98] Vladmir N. Vapnick. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.