A Comparison of the Accuracy of Support Vector Machine and Naïve Bayes Algorithms In Spam Classification

Rita McCue
University of California at Santa Cruz

Nov 29, 2009

Abstract

Four implementations of the Naïve Bayes classification algorithm were compared to those of the four different kernels available in an implementation of a Support Vector Machine (SVM) classifier. The two algorithms (and the four versions of each) had their accuracy, speed and effectiveness compared against each other. The Naïve Bayes implementation was done in Matlab, and all plots were created by the built in utilities in the Matlab program. The Support Vector Machine implementation used is the SVMLib application, and included are comparisons of the four available kernel choices.

1 Introduction

Spam has long been known to be a huge drain on office productivity, costing businesses great expense in lost employee time. Because of the obvious need, there has been a great deal of work done in the past decade in creating ever more effective spam-fighting products. Modern spam fighting products are required to be versatile, and are designed so they are able to adapt to detect the ever-changing content of spam from the ”ham”, i.e. the valid email received.

The primary method that these problem emails are detected is by means of detecting the spam-like nature of their content. All emails - in fact all text-based documents - can be represented by a binary set of features; a simple example of one of these features could be whether the email included a specific word. While many possible features using common words (such as ”the”) would be worthless for spam detection, a careful selection of relevant features, including likely spam indicators (such as the word ”viagra” and terms such as ”lose weight now”) and likely ham indicators (which might indicate terms such as ”visio diagram” for business users, or the acronym ”SOE” for UCSC students) is essential for effective spam detection.

Two commonly used algorithms genres in machine learning include the Naïve Bayes and Support Vector Machines (SVM) algorithms. Naïve Bayes algorithms are relatively easy to implement probability based classifiers with strong (aka naïve) assumptions that all features are completely independent. While algorithms using this assumption are unable to do complex associations between features (such as the high probability of spam when the feature ”doctor” is coupled with the features ”free” and ”list”), they can still be highly effective spam classifiers. In comparison to the simplistic Naïve Bayes algorithms, Support Vector Machines are exceedingly difficult to implement. Support Vector Machines work by translating non-linearly classifiable feature data into a higher-dimensional hyperplane where it is possible to classify the data in a simple, linear manner. Both algorithms are known to be highly effective classifiers, and are able to achieve impressive accuracy in spam classification.

2 Data Choice

For the algorithm comparisons made in this study, a ready-made data set of spam/ham and their associated features was utilized. At the time of this paper was written, this data set could be found on UCSC's class website for CMPS 242 (Introduction to Machine Learning). This data consists of a matrix with 2000 email rows, each with 2000 feature columns, resulting in a 2000 by 2001 matrix. Thus, each email in this data set has 2000 features, each of which is a binary representation (aka a "yes" or "no") of a word's existance within that email. As certain machine learning algorithms base their updates only on features that are "positive" results, all 2000 features of each email were then negated and concatenated onto the end of the original feature set in order to derive the maximum amount of information possible. This results in a 2000 by 4001 matrix, where each of the 2000 emails has 4000 features for use by the learning algorithms. For the purposes of this experiment, we are not actively dealing with text emails, and are not attempting to modify which common words to use as our features within the data set. This data set was separated into two portions: the first 75% of the data set was used as a training set for the data, and the final 25% was set aside to be used as the testing set.

2.1 Further Data Modifications Necessary for Naïve Bayes Implementations

The original data set as given above must be modified to be usable by Naïve Bayes learning algorithms. The training data was separated into two matrices based on the value of the first column (spam/ham), which (in the data set given) resulted in 1021 "Ham" emails in a 1021 by 4000 matrix and 479 "Spam" emails in a 479 by 4000 matrix.

3 Naïve Bayes Classifiers

Naïve Bayes classification algorithms are currently one of the most highly popular algorithms used in spam filtering applications. This popularity is likely due to the extremely high accuracy and quick training speeds they attain despite their relative simplicity to implement.[3] Though most papers on the subject refer to "the Naïve Bayes algorithm" in singular rather than plural form, there are in actuality several different versions of Naive Bayes that differ significantly. Because the data set used in these comparisons is in binary form, only three of the five algorithms are expected to provide accurate results, and one of the five algorithms could not be implemented at all.

Despite the differences in how they are implemented, all versions of Naïve Bayes generally result in roughly the same level of accuracy as the basic Naïve Bayes classifier, with which it's possible to work with both binary and non-binary data; the alternate algorithms value instead lies in the fact that they sometimes allow "the same level of accuracy to be reached with fewer attributes".[5]

3.1 Basic Naïve Bayes Classifier

The Basic Naïve Bayes Classifier is derived from Baye's theorem; when applied to spam filtering, gives the probability that a message with vector (aka the feature set) $\vec{x} = \langle x_i, ..., x_m \rangle$ belongs in category c as:

$$\frac{p(c_s) \cdot p(\vec{x}|c_s)}{p(c_s) \cdot p(\vec{x}|c_s) + p(c_h) \cdot p(\vec{x}|c_h)} > T$$

(Where the vector $\vec{x}$ are the features of the message we're testing and T being an even split of the 2 possible weights - aka 0.5.)

3.2 Multi-Variate Bernoulli Naïve Bayes

The Multi-Variate Bernoulli Naïve Bayes algorithm represents each message of a category (either spam or ham) as a set of tokens, containing (only once) each token that occurs in the message as the result of the number-of-features Bernoulli trials, where at each trial we decide whether or not each feature will occur in the message.

$$\frac{p(c_s) \cdot \prod_{i=1}^{m} p(t_i|c_s)^{x_i} \cdot (1 - p(t_i|c_s))^{1-x_i}}{\sum_{c \in \{c_s, c_h\}} p(c) \cdot \prod_{i=1}^{m} p(t_i|c)^{x_i} \cdot (1 - p(t_i|c))^{1-x_i}} > T,$$

(Where T remains the same, $p(t|c)$ is estimated using a Laplacean prior such as $p(t|c) = \frac{1 + M_{t,c}}{2 + M_c}$, m is the number of features, x_i is feature number i of the current email we're testing, and T being set at an even split of 0.5 .)

3.3 Multi-Variate Gauss Naïve Bayes

The Multi-Vairate Gauss Naïve Bayes algorithm is a modified version of the Multi-Variate Bernoulli equation which is designed to use real value attributes (rather than the boolean attributes of our data set) by assuming each attribute follows a normal distribution $g(x_i; \mu_{i,c}, \sigma_{i,c})$ in each category c. This algorithm could not be accurately tested with the boolean data set used, but was implemented anyway for the sake of curiosity; though the results will be less accurate than if we had a true data set, we can assume for the purpose of this equation that each feature simply appears a maximum of once in each email.

$$\frac{p(c_s) \cdot \prod_{i=1}^{m} g(x_i; \mu_{i,c_s}, \sigma_{i,c_s})}{\sum_{c \in \{c_s, c_h\}} p(c) \cdot \prod_{i=1}^{m} g(x_i; \mu_{i,c}, \sigma_{i,c})} > T$$

(where $g(x_i; \mu_{i,c}, \sigma_{i,c}) = \frac{1}{\sigma_{i,c}\sqrt{2\pi}} * e^{\left(-\frac{(x_i - \mu_{i,c})^2}{2\sigma_{i,c}^2}\right)}$ and with T being set at an even split of 0.5.)

3.4 Multinomial Naïve Bayes, Term Frequency (TF) Attributes and Multinomial Naive Bayes, Boolean Attributes

These two algorithms use the same equation, but are kept distinct in that they each require a different type of data sets. Technically, the Multinomail Naïve Bayes with Term Frequency Attributes algorithm could not be implemented, as our data set did not include the number of times that a term existed within an email, only a binary indicator of the terms presence. The version of Multinomial Naïve Bayes that uses Boolean Attributes, however, was of course included in our comparisons.

$$\frac{p(c_s) \cdot \prod_{i=1}^{m} p(t_i|c_s)^{x_i}}{\sum_{c \in \{c_s, c_h\}} p(c) \cdot \prod_{i=1}^{m} p(t_i|c)^{x_i}}$$

(where $p(t|c)$ is estimated using a Laplacean prior such as $p(t|c) = \frac{1 + N_{t,c}}{m + N_c}$.)

3.5 Flexible Bayes

Flexible Bayes was another Naïve Bayes algorithm which could not be tested using our data set. FB models $p(x_i|c)$ as the average of $L_{i,c}$ normal distributions with the same typical deviation as mentioned above but different mean values.

$$p(x_i|c) = \frac{1}{L_{i,c}} \cdot \sum_{l=1}^{L_{i,c}} g(x_i; \mu_{i,c,l}, \sigma_{i,c}) > T$$

(where $L_{i,c}$ is the number of different values X_i has in the training data of category c, $\mu_{i,c}$ is the mean of a normal distribution in that category, $\sigma_{i,c}$ as the single typical deviation of that category c, and with T being set at an even split of 0.5.)

4 Support Vector Machine Classifiers

Simple linear classifiers, being unable to deal with non-linearly separable data or noisy data, are not always sufficient data classifiers. If necessary, the more complex Support Vector Machine (SVM) learning algorithms are there to fill the gap. Support Vector Machines, though exceedingly complex to implement, offer a solution to the above limitations; by mapping data into a richer feature space including non-linear features, it is then possible to classify the data in a simple linear fashion, something that seemed impossible just moments before.

Support Vector Machines are based on Mercer's theorem, which states that any continuous, symmetric, positive semi-definite kernel function can be expressed as a dot product in a high-dimensional space. As a side effect of this theorem, the resulting problems have no local minima, meaning that all have the property of convexity. If a machine learning algorithm can be rewritten into any higher dimensional space so that they so that the mathematical computations it uses are based solely on inner (dot) products between the data features, a Support Vector Machine can be created by replacing every dot-product with the chosen SVM kernel, a computational short cut called the Kernel Trick. The higher-dimensional non-linear algorithm is equivalent to the original lower dimension linear algorithm; the new higher-dimensional representation is simply operating in a feature space mapped from the original. Because of this kernel trick, the new feature space function - which has the potential of being highly complex - is never explicitly computed. This is highly desirable, as it makes this calculation solvable in merely polynomial - rather than exponential - computation time.

Merely finding one hyperplane that separates the training data is not enough to result in an accurate learning machine; many such hyperplanes will exist, and - as it is extremely easy to overfit in high dimensional spaces - merely using any random hyperplane will likely result in poor accuracy against the test data set. In order to choose the best possible hyperplane and minimize the risk of overfitting, it is necessary to find the one with the maximal margin between the classes of items being identified. The transformation from Primal to Dual form is therefore done by means of a Lagrangian, whose constraints each place an upper bound on the linear combination of the Lagrangian variables and thus limit the amount it is possible to fit the training data within the hyperspace created by that Lagrangian. Additionally, if the kernel and it's parameters were chosen carefully, the margin will be larger and better generalization can be achieved. Once this maximum margin is found, only the points nearest to the hyperplane will be given a positive weight, resulting in sparsely weighted features within the within the higher-dimensional feature space. These points are rather appropriately titled support vectors. [1]

For the comparisons done in this paper, we utilized a freeware Support Vector Machine application called SVMLib. This application has four available kernels built in; linear, polynomial, radial basis function, and sigmoid. Like many other learning algorithms,

Support Vector Machines require parameter tuning in order to improve the accuracy of their test results. To do this parameter tuning, 5-fold cross validation was performed for each of the four available kernels in SVMLib's Matlab implementation on the training data. A logarithmic "grid search" method was performed to find the best choices for c and γ; those that had the best average cross-validation accuracy are the ones chosen for use on the test data.

With the installation of a couple of additional programs (C++, Python, GnuPlot) and the additional requirement of translating your data from Matlab's default array data format to SVMLib's "dense" format, SVMLib's pre-written code could be used for selecting the best parameter values for each kernel type. Though rough manual grid-search parameter tuning had already been performed, the default parameters set by SVMLib and best parameters found by the automated method was also implemented for curiosity's sake. The results of all three tunings were run on the test data to see how a "quick and dirty" parameter choice would compare to ones carefully tuned. The other choice available in most SVM applications - which kernel to choose - is a highly complex subject which were not truely researched for this paper. However, in the descriptions of each kernel (below) a cursory attempt was made to give some hints gleaned during the research done.

Default parameters chosen by SVMLib (Table 1) [2]

Parameter	Default Choice
c	1
γ	1/numFeatures
polynomial degree d (polynomial kernal only)	3
coef r (polynomial and sigmoid kernals only)	0

4.1 Linear Kernel

Unlike the rest of the kernels, the Linear Kernel has only one tunable parameter (c). It is

a special case the Radial Basis Kernel, of one of the other 4 kernels available. This kernel performs well if the number of features is large compared to the size of the data, and if is unnecessary to map to a higher dimensional space. It deals poorly, however, with "noisy data".

$$K(x_i, x_j) = x_i^T x_j$$

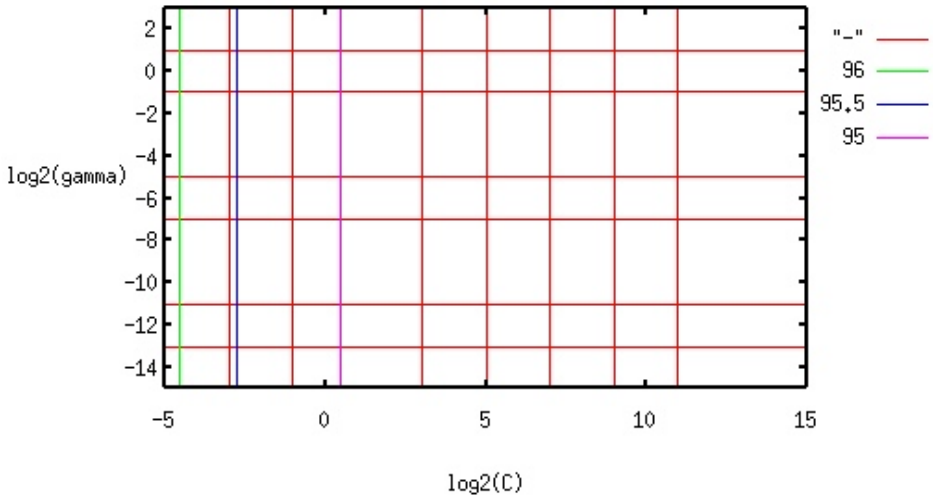

Figure 1: Support Vector Machine Kernels - Linear Kernel Parameter Tuning.

4.2 Polynomial Kernel

The polynomial kernel has the most tunable parameters of all the kernels available in the SVMLib application. In addition to the basic c and γ parameters, it additionally has 2 extra parameters; the polynomial degree d and the degree coefficiant r. It is important to be restrained in the choice of d, as kernel values may go to infinitiy or zero when the degree is large. In the testing done for this paper, only the more common parameters c and γ were tuned; the variables d and r remained at their default values (3 and 0, respectively).

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$$

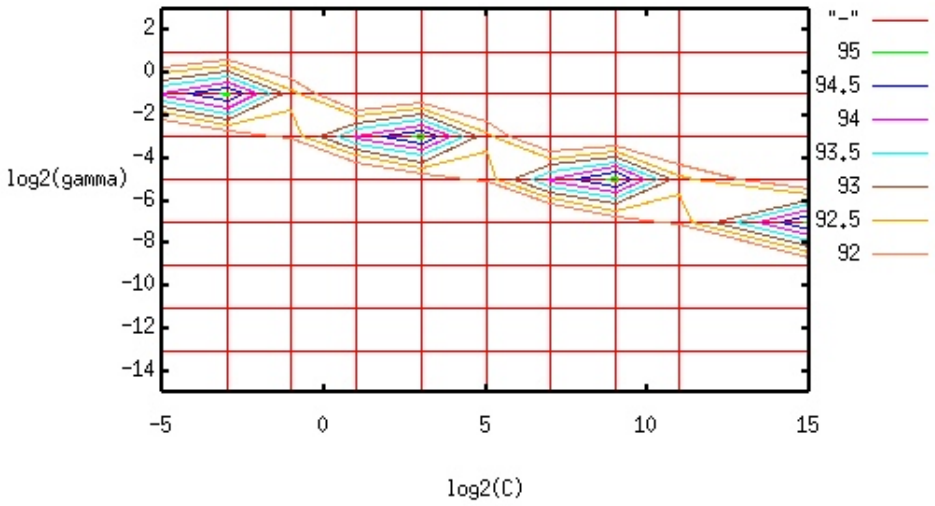

Figure 2: Support Vector Machine Kernels - Polynomial Kernel Parameter Tuning.

4.3 Radial Basis Function (RBF) Kernel

The RBF kernel maps samples into the higher dimensional space in a non-linear fashion, so (unlike the linear kernel), it can handle cases where the class labels and attributes are related in a non-linear manner. However, unlike the Linear Kernel, it isn't suitable when number of features is "very large". This is the default kernel chosen by the SVMLib application, and is suggested as the best kernel to "start with" by the SVMLib Guide. [2]

$$K(x_i, x_j) = e^{(-\gamma||x_i ? x_j||^2)}, \gamma > 0$$

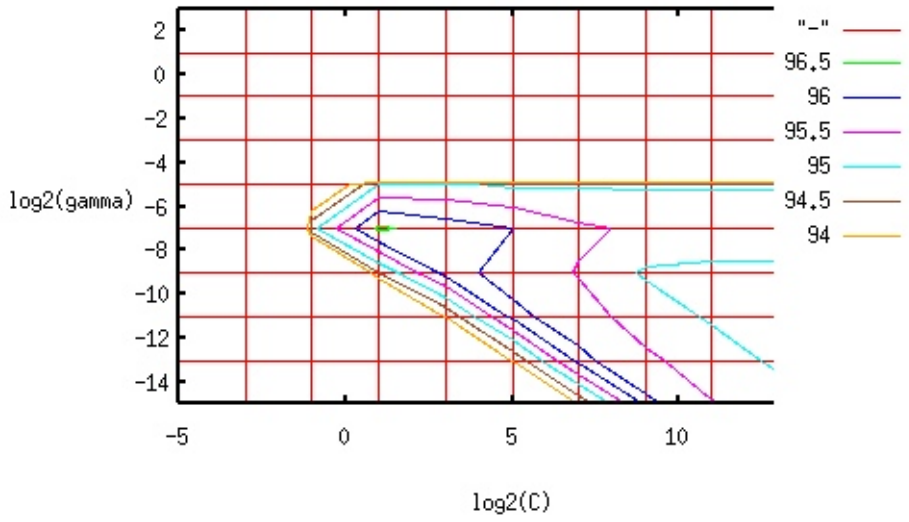

Figure 3: Support Vector Machine Kernels - Radial Basis Function Kernel Parameter Tuning.

4.4 Sigmoid Kernel

The Sigmoid Kernel, though a popular Support Vector Machine commonly found in neural networks, has several properties that remain not fully studied. It is known, however, that unlike the other Support Vector Machines Kernels available in SVMLib, the Sigmoid Kernel is only conditionally a positive semi-definite function[4]; for this reason, guides of using SVMLib give a warning that this kernel is not valid for some parameter choices.

$$K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$$

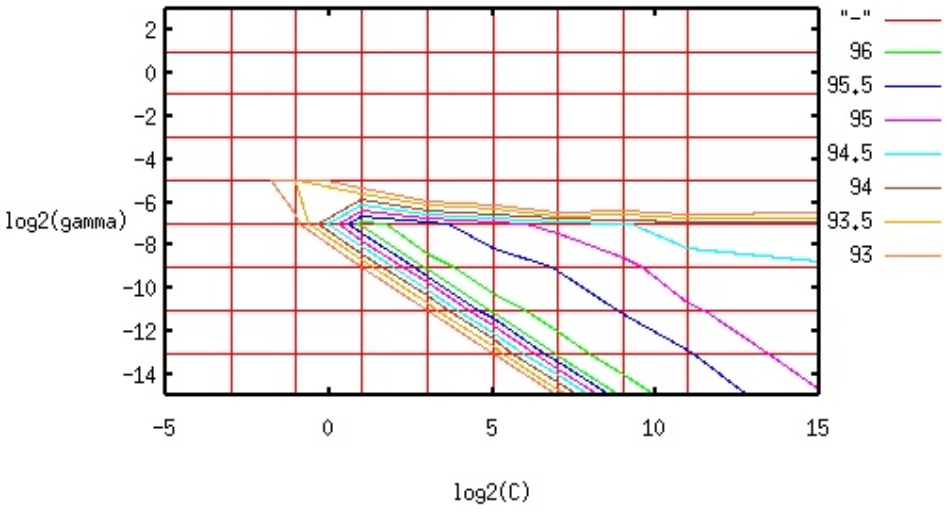

Figure 4: Support Vector Machine Kernels - Sigmoid Kernel Parameter Tuning.

5 Results and Conclusions

5.1 Testing Data Collected

The main loss comparator used in order to evaluate these different machine learning algorithms was their accuracy at predicting the ”Spam”/”Ham” property of the emails in the test data set. Though misclassified ”Ham” examples in reality would likely be given more weight than misclassified ”Spam” examples, our accuracy calculations equally weight the two misclassifications. Despite it’s simplicity, it was found that the Basic Naïve Bayes classification algorithm gave the best prediction results on our testing set, coming in at a respectable 97.8%. However, assuming accurate parameter tuning (when relevant), all Support Vector Machines and Naive Bayes algorithms that were designed to utilize binary imput data resulted in an accuracy well above the 90% range. As mentioned when the Naïve Bayes algorithms were first described above, the Multivariate Gauss Naïve Bayes algorithm is not designed to work with binary data; it instead expected counts of the occurrences of each feature in the email. It’s far lower accuracy of 70%, while still far better than random guessing, would likely improve greatly if it received data of the correct form.

5.1.1 Naïve Bayes Classification Results

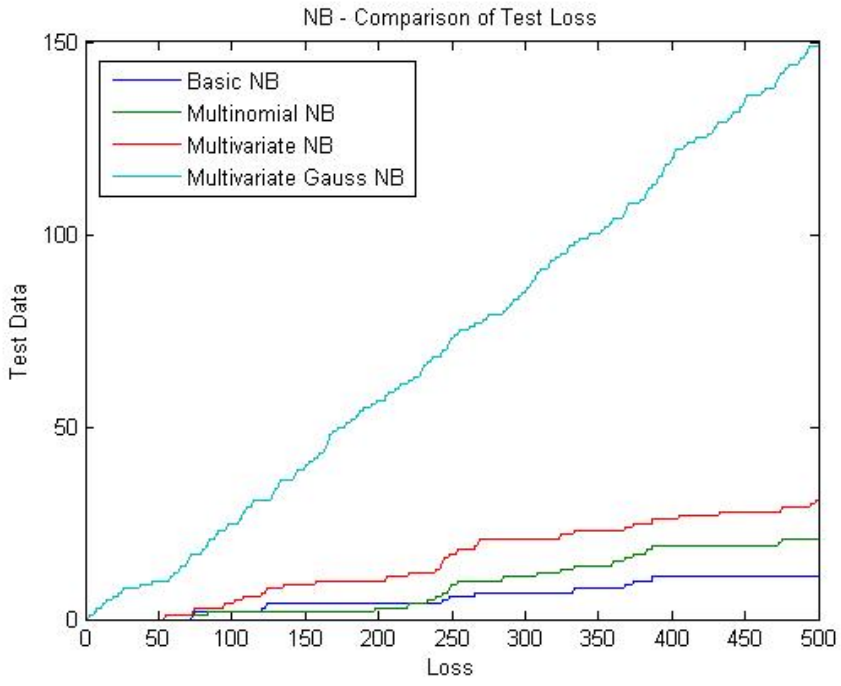

Figure 5: Naive Bayes Algorithms - Comparison of Test Loss.

Naï Bayes Accuracy (Table 2)

NB Algorithm	Accuracy	# Correct in Testing Set
Basic Naïve Bayes	97.8%	$\frac{489}{500}$
Multinomial Naïve Bayes	95.8%	$\frac{479}{500}$
Multivariate Naïve Bayes	93.8%	$\frac{469}{500}$
Multivariate Gauss Naïve Bayes	70.2%	$\frac{351}{500}$

5.1.2 Support Vector Machines Classification Results

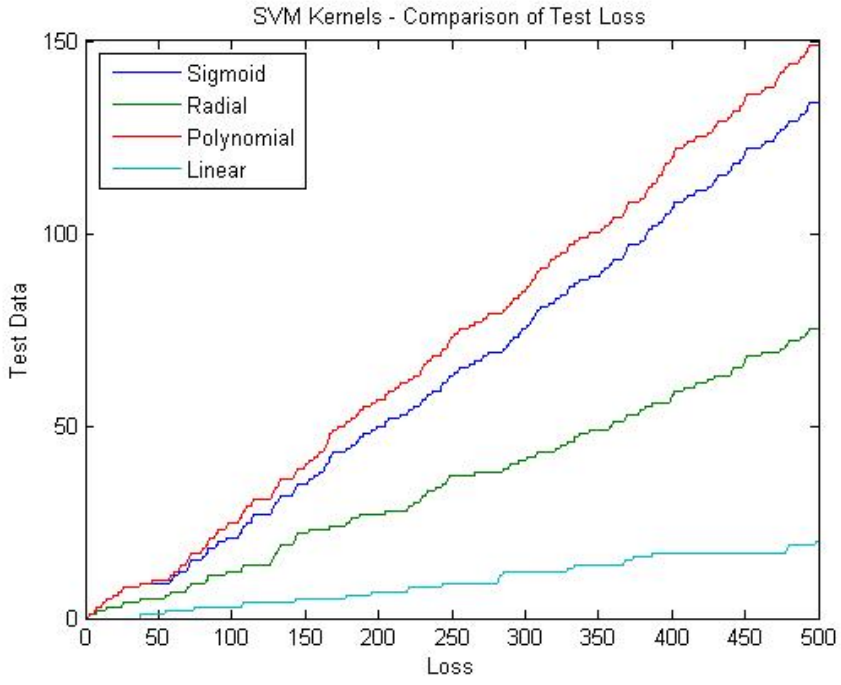

Figure 6: Support Vector Machine Kernels - Comparison of Test Loss (Pre-Tuning, using default SVMLib parameter choices).

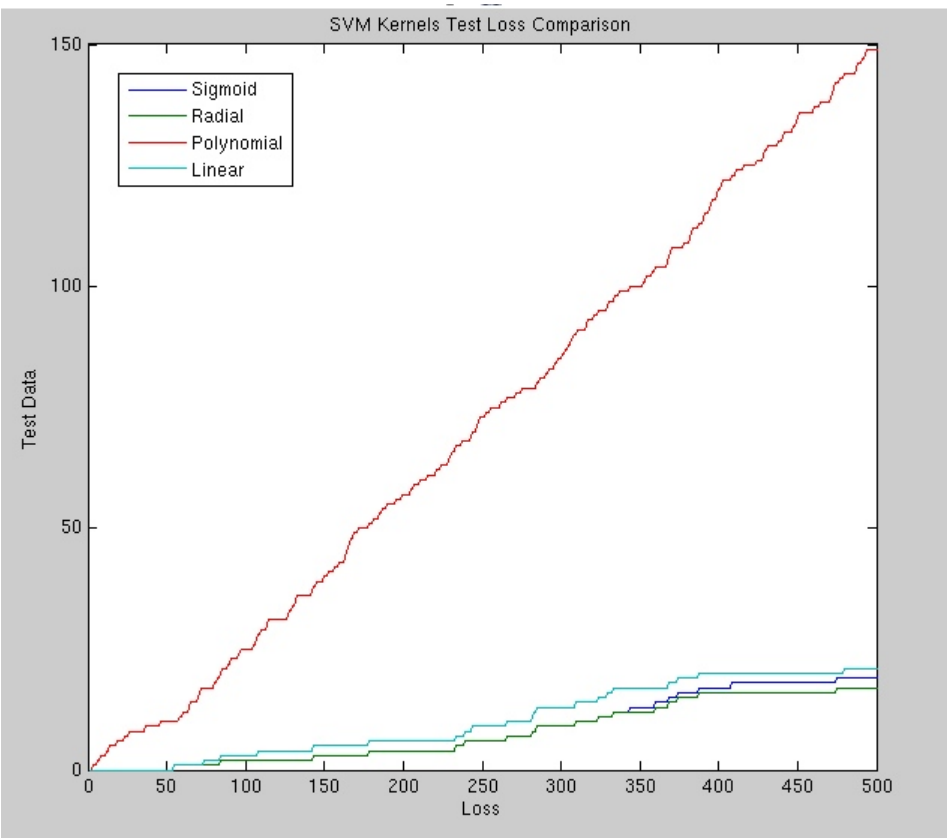

Figure 7: Support Vector Machine Kernels - Comparison of Test Loss (Post-Tuning, Manually written 5-fold cross-validation with a basic grid search).

SVM Accuracy, default parameters (Table 3)

SVM Kernel Type (Pre-Tuning, using default SVMLib parameter choices)	Accuracy	# Correct, Testing Set
Linear Kernel	96%	$\frac{480}{500}$
Radial Basis Kernel	85%	$\frac{425}{500}$
Sigmoid Kernel	73%	$\frac{366}{500}$
Polynomial Kernel	68%	$\frac{351}{500}$

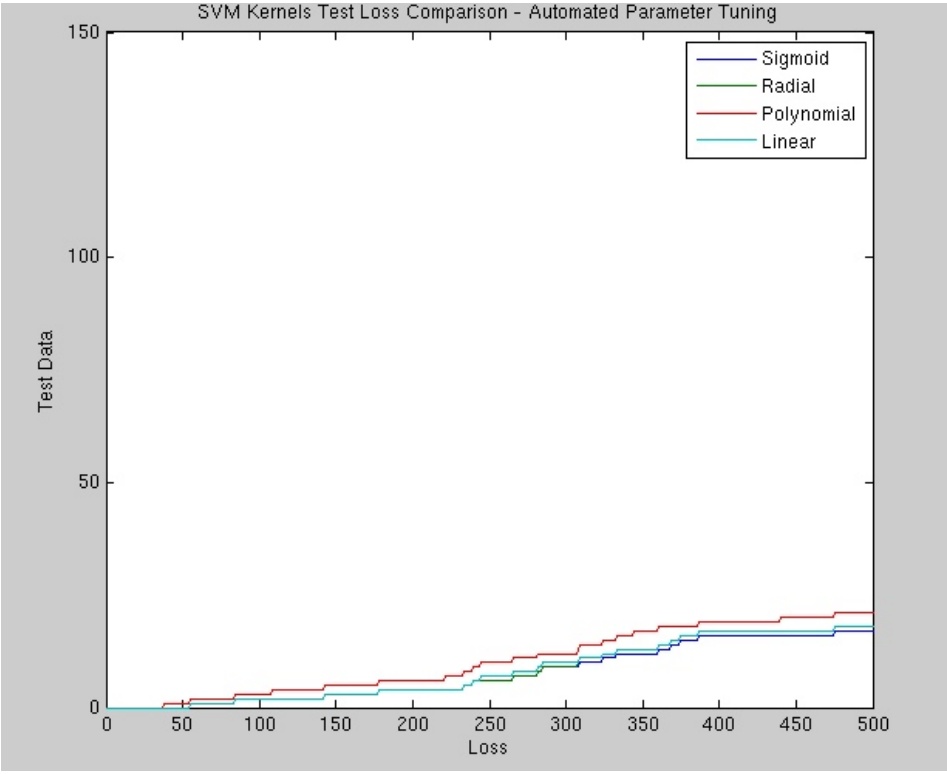

Figure 8: Support Vector Machine Kernels - Comparison of Test Loss (Post-Tuning, Automated).

SVM Accuracy, tuned parameters (Table 4)

SVM Kernel Type (Post-Tuning)	Accuracy (manual tuning)	# Correct, Test Set (manual tuning)	Accuracy (auto-tuning)	#Correct, Test Set (auto-tuning)
Sigmoid Kernel	96.2%	$\frac{481}{500}$	96.6%	$\frac{483}{500}$
Radial Basis Kernel	96.6%	$\frac{483}{500}$	96.4%	$\frac{482}{500}$
Linear Kernel	95.8%	$\frac{479}{500}$	96.4%	$\frac{482}{500}$
Polynomial Kernel	70.2%	$\frac{351}{500}$	95.8%	$\frac{479}{500}$

1

[1]Created by LaTeX 2_ε on Nov 29, 2009.

References

[1] Nello Cristianini. Support vector and kernel machines. PDF Online, Presentation.

[2] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classication. PDF Online, May 2009.

[3] Course lecturer: Amos Storkey and notes written by David Barber. Machine learning and pattern recognition: Naive bayes. PDF Online, edited class notes.

[4] Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. PDF Online.

[5] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes – which naive bayes? PDF Online, 2006.