Slam the Spam: A Comparison of KNN, SVM, and Naïve Bayes for SMS Spam Classification

Andrew Kim (anykim@iu.edu)

Jean Chiu (weitchiu@iu.edu)

Mitchell Thomas (mitthoma@iu.edu)

Indiana University Bloomington

Slam the Spam: A Comparison of KNN, SVM, and Naïve Bayes for SMS Spam Classification

Background

The objective of this project is to identify spam and non-spam Simple Messages Service (SMS) messages. SMS spam is any junk message delivered to a mobile phone as a text message. A common method of identifying a message as spam is to search for features and keywords often seen in known spam.

We will perform this machine learning classification task using various machine learning techniques: Naïve Bayes, support vector machine (SVM), and k-nearest neighbors (KNN). We will use the Pandas library to read and split the previously classified data from the University of California Irvine to train and test our models.

We are implementing Naïve Bayes and KNN from scratch. SVM will be implemented using svm and metrics from the sklearn library. Since Naïve Bayes and KNN are implemented without any library, one of the challenges will be to understand how to translate the theories behind these two models into program-executable code. Another major challenge will be to identify the necessary features or traits needed for each of three models. Since each entry in the dataset has a message as its only attribute, it is important to parse the most relevant information and to overcome the difficulty that is raised by not having a subject line and sender information.

After implementing each model, we will test the accuracy of each model on the test dataset. We will be comparing the results and report our findings by discussing why we think certain models return more accurate results than the other models.

Implementation

Naïve Bayes

Naïve Bayes classifiers are probabilistic classifiers that are based on Bayes' theorem and the assumption of conditional independence between each pair of traits or features given the classification.

In the case of spam message classification, we have a set $X = x_1, x_2, ..., x_n$ where

$x_1, x_2, ..., x_n$ are traits of each message. The goal is to compare P(Spam$|x_1, x_2, ..., x_n$) and P(Not Spam$|x_1, x_2, ..., x_n$) for a given message. If P(Spam$|x_1, x_2, ..., x_n$) is greater than P(Not Spam$|x_1, x_2, ..., x_n$), then the message will be classified as spam. On the other hand, if P(Spam$|x_1, x_2, ..., x_n$) is less than P(Not Spam$|x_1, x_2, ..., x_n$), then the message will be classified as non-spam.

To implement such a Naïve Bayes classifier for spam classification, we need to first find the necessary probabilities to calculate P(Spam$|x_1, x_2, ..., x_n$) and P(Not Spam$|x_1, x_2, ..., x_n$). Applying Bayes' theorem and using the conditional independence assumption, P(Spam$|x_1, x_2, ..., x_n$) = $P(Spam)P(x_1|Spam)P(x_2|Spam)...P(x_n|Spam)$ and P(Not Spam$|x_1, x_2, ..., x_n$) = $P(NotSpam)P(x_1|NotSpam)P(x_2|NotSpam)...P(x_n|NotSpam)$.

Now, to calculate P(x_i|Spam) and P($x_i|NotSpam$) for all i, we have to consider all values that x_i can takes. Suppose $x_i = x_{i_1}, x_{i_2}, ..., x_{i_j}$, we need to calculate $P(x_{i_1}|Spam), P(x_{i_1}|NotSpam), P(x_{i_2}|Spam), P(x_{i_2}|NotSpam), ..., P(x_{i_j}|Spam), P(x_{i_j}|NotSpam)$. We can find these probabilities by applying conditional probability where $P(x_i = x_{i_j}|Spam) = \frac{\text{number of messages that are spam and have the given trait}}{\text{number of total messages}}$ and $P(x_i = x_{i_j}|NotSpam) = \frac{\text{number of messages that are non-spam and have the given trait}}{\text{number of total messages}}$.

Given n as the number of entries in the training set, d as the number of features and c as the number of classes, the time complexity for training the Naïve Bayes classifier is O(nd) since all we need to do is to sum up the occurrences for each feature in the training set. The time complexity for classifying a message after we train the model is O(cd) since we need to parse information for d features for each class.

In general, this Naïve Bayes' classifier's accuracy increases as we add more features. Some traits are more relevant than others, so they increase the accuracy more than other traits. It is important to determine what traits we should use to calculate the probabilities, and sometimes, this task is the most difficult part of implementing Naïve Bayes' classifiers.

K-Nearest Neighbors (KNN)

As discussed in lecture [1], KNN works by taking a data point from the test set, comparing it to each point from the training set using a similarity function, then using the K most similar data points to determine the final classification (where K is a pre-defined integer). In our case, we defined K to be the square root of the number of training points, which is roughly 67. Our similarity function was:

$$similarity(message1, message2) = \frac{\text{words in common between message1 and message2}}{\text{total amount of words between message1 and message 2}}$$

To complete the implementation, our program took the 67 most similar messages and checked whether more of the neighbors are classified as spam (1) or non-spam (0). Ties are assumed to be spam.

The biggest limitation on KNN is its performance under scale. Because it must compare a training point to every test point, there is a substantial bottleneck. For a test of our function on a single message, the time complexity is:

$$O(W*T + Tlog(T) + K)$$

where W = the longest message length(in words) and T = the length of the test data set. The space complexity is $O(T + K)$- we just need to store the data sets and the nearest neighbors.

Support Vector Machine (SVM)

In a brief overview, the Support Vector Machine essentially splits the classifications in two sections by forming a hyperplane- or a separator- that is D - 1 dimensions, where D is the dimension of your input data. After finding a suitable hyperplane that separates the data, the model learns from training data how wide it should separate the margins of the particular separating space, which consists of the hyperplane in the middle and two support vectors that have a maximal margin in between to polarize the classifications as much as possible.

Sometimes, the classifications are not initially separable with a hyperplane, so this is where a kernel function is used within the SVM in order to increase dimensionality of

[1] Lecture 20: Nearest Neighbors

the data until a hyperplane can be formed to separate the data. The higher the dimensions of data, the more computation that is needed, but in our case we were just fine using a linear kernel function. This kernel function is essentially a mathematical function that is applied to the data. We used a linear kernel function in our case, $k(x, y) = x^T y + c$, where x corresponds to unclassified input and y corresponds to training input.

Since we chose a linear kernel for the support vector machine, the runtime complexity is O(N), where N is the number of input dimensions. Lastly, for this implementation, there was no other outside code used other than functions pulled from the scikit-learn library for SVM implementation.

Findings

The overall results for each of the algorithms can be found in 1. These results are as is, and per the instructions, we did not manipulate our algorithms to artificially improve the numbers after the initial run.

Naïve Bayes	97.76%
K-Nearest Neighbors	93.81%
Support Vector Machine	96.14%

Table 1

Accuracy on Test Data for Naïve Bayes, KNN, and SVM

The percentages are more or less in line with what we anticipated, though Naïve Bayes did much better than we anticipated. Of all our three models, Naïve Bayes is the "simplest," treating the selected factors as conditionally independent to come up with a single probability that a message is spam. One would think that KNN would have higher accuracy, since you are creating more than a binary decision tree- instead, you end up with classifications based on similarity with the known training set. The most advanced model of the three, SVM, is similar to KNN in that it can form non-linear decision boundaries. Although SVM is not *always* better than KNN, it works well here for a number of reasons. First, we have a fairly large amount of points to classify, and

SVM does not require iterating through the training set for every unlabelled point. Second, it is less prone to over-fitting compared to KNN, as the boundaries are drawn to be as far as possible from all points (see Implementation). Additionally, like Naïve Bayes, SVM works very well for high-dimensional problems like text classification. Finally, although we used a linear kernel for our implementation, there are other types (such as the radial basis function and sigmoid) available for experimentation, should the data not be linearly separable.

We came into this project with the hypothesis that Naïve Bayes, KNN, and SVM would all hold merit, and indeed, our findings suggest that all of these models could be used in a real world context (though Naïve Bayes is clearly the winner in terms of accuracy here). However, just like with other machine learning problems, none of these models are perfect, and they come with their own individual trade-offs. For instance, we observed a clear speed-accuracy trade-off. Naïve Bayes learns "on the fly," so we end up with a relatively fast solution. But because of this, it can also be error-prone if it routinely comes across message content that has been manipulated or does not already fit in the model. Particularly, it is subject to "Bayesian poisoning" attacks, where someone could maliciously influence the model by flooding spam messages with innocuous words. Meanwhile, KNN and SVM are both relatively high accuracy models that suffer from relatively low speed for classification. We now note in our Discussion section how we could further experiment with these models to improve them and perhaps reduce overhead costs.

Discussion

In our spam or ham message classification, we used five main attributes. These attributes consisted of message length, capitalization, whether or not the message contained numeric values, and whether or not the message contained one of the specified 'spam' keywords that we pre-selected. Our models in general seemed to perform quite well based on these attributes, but in order to improve the model in the future, we could add more specific attributes or modify our existing ones to be even more polarizing with the classifications.

An example of more attributes we could add in could be a rate of reply time and length of the incoming phone number. We could also strengthen the older attributes by fitting our initial models with much more data so that the models can make even more accurate predictions on new data.

Throughout the project, we learned more about the models we used for classification in greater depth, how the various models function when it comes to classification, and which model tends to perform better on our data set. Also, we received valuable experience in picking a data set that interested us and posed a problem that we wanted to solve. In this process, we got practice in analyzing the data, selecting appropriate attributes to split the data on, and pre-processing that data in order to train various models with it.

Lastly, our various implementations model how humans would classify spam text messages or not. They do this with the use of the various attributes that we selected. For instance, the spam keywords that we looked for usually show up in spam messages someone gets. A human might also look at a long text message and not read it assuming it's spam, especially if they see lots of capitalization, numbers, and/or some of the keywords we specified. Thus, we built our models to reflect this kind of human behavior.