

Supervised Machine Learning for Classifying Text with Ruby

1 Introduction

1.1 *What are Classifiers*

In the world of text, classifiers or supervised machine learning is a method for assigning a class or category to textual document. Some of the classifiers you may or may not have heard of include Naïve Bayes, Support Vector Machines, Decision Trees, K-Nearest Neighbors, and Perceptron. The “supervised” part comes from the need to have a set of training data. In the case of text classification this would be a set of documents and the class of each one (spam, non-spam, etc.).

1.2 *What they are useful for*

Text classification is useful for identifying language, news category, blog category, email spam, trackback spam, comment spam, and general document triage.

1.3 *What is needed to perform classification*

To do document classification you’ll perform at least some of the following tasks. Get some training data, clean and stem the text, represent your documents as vectors, perform feature selection, select your classifier and train it, and finally run the classification on some test data to see how well it performs.

2 The Document Vector Model

With classification we generally represent documents as a vector. This can be a vector of the terms in the document and other metadata. They can be placed into vector space so a number of machine learning algorithms can be run on them. The simplest picture would be a two-dimensional set of vectors.

2.1 *Contiguity Hypothesis*

Documents in the same class form a contiguous region and regions of different classes don’t overlap.

2.2 *Words as dimensions in the vector*

A simple document vector model is just to break on the spaces and possibly remove some punctuation.

2.3 *Lowering the Dimensionality through Stemming*

The stem of a word is the base part. Stemming will lower the dimensionality of your document by term matrix by turning cat and cats into the same term. The porter stemmer is the classic stemming algorithm. The stemmer gem will get you started. Just load the gem and require it. You’ll then be able to call stem on any string.

2.4 Vector of Features (including data other than words)

The thing to note is that the document vector is really just a vector of features. While these are often counts of stemmed words they can be other things. Words that appear in anchor text, word counts, or other things that are calculated on the fly based on domain specific information.

3 Feature Selection

Feature selection is useful for limiting the size of the vocabulary and for often increases the classification accuracy. This is because it can eliminate noisy features and avoids overfitting. Overfitting is basing decisions on rare events that occur in the training data that don't generalize to the test and real world data.

3.1 Mutual Information (MI)

Measures how much information each word contains about the class. If the word is just as likely to appear in a class as in the rest of the training set, its MI value is close to 0. Features with the highest values are returned.

3.2 Chi-squared

In statistics this is used to test the independence of two variables. In the case of feature selection the two variables are the word and the class. All of the words appearing in the training set are then ranked by their chi-squared value. Where values closer to 0 are less useful.

3.3 Frequency Based

This is the least accurate of the methods but the easiest to implement. It just selects the words that appear most often in the class. The drawback being that some words appear often in the class, but are not a valid indicator like "the" or "and".

3.4 Comparison

Chi squared and mutual information are different in their selection criteria. Chi square will include features that may occur only once in the training set. Mutual information will not include these very rare features. However, chi squared can be modified to exclude features that occur only a certain number of times or have a chi squared value below some cutoff. Both MI and chi squared are better than the more simple frequency based.

4 Brief Overview of Classifiers

There are many different classifiers. I only list a few here. Generally they are separated into linear and non-linear types. This refers to the decision boundary that the classifier makes. Non-linear classifiers can represent an arbitrarily complex decision boundary.

4.1 Linear

Linear classifiers look for a hyperplane that separates one class from the other. In the sense of the two dimensional example, it looks for a line that separates one class of documents from the other class.

4.1.1 Naïve Bayes

Also known as idiot bayes, this is one of the most popular text classification methods. This is because of its simplicity, speed, and effectiveness for text classification. It is a generative model that tells the probability that a class will generate a given word. The naïve part refers to two assumptions that the classifier makes. Positional independence: that the position of a word in a document has no bearing on its class. Conditional independence: that the probability of a word occurring in a given class and document is independent of the other words in that same document. These are obviously not correct, but making those assumptions makes things workable. NB is surprisingly accurate for text classification. One thing to note is that the probabilities it generates for the classes are completely inaccurate, but it doesn't matter because it just takes the highest one.

4.1.1.1 Binomial and Multinomial Models

Binomial model just accounts for term's presence in a document. The multinomial model counts the number of times each term occurs in the document and uses that as a value.

4.1.2 Support Vector Machines

A linear classifier that looks for a decision boundary that is maximally far away from any data points. They can get pretty complex mathematically. The thing to know is that almost all of the people using SVMs for classification tasks use a library of some kind. In ruby you can use Ruby SVM which is a ruby binding to libsvm. Libsvm is a library for running support vector machines. It's written in C++ with source also available in Java. It also supports multiclass problems.

4.1.2.1 Kernel Functions

A kernel function can map the document vectors into a higher dimensional space. This helps for when the classes are not linearly separable in the vector space they are in. The four basic kernels are linear, polynomial, radial basis function, and sigmoid.

4.1.3 Perceptron

This linear classifier uses a neural network to find the decision boundary.

4.1.4 Rocchio

Calculates centroids that lie in the center of the points for a class. The decision boundary is then based on which centroid is closer.

4.2 Non-linear

Non-linear classifiers can represent an arbitrarily complex decision boundary.

4.2.1 K Nearest Neighbor (KNN)

Selects a class of a document based on the number k documents closest to it in vector space. Generally an odd number k is used to ensure no tie exists. k=3 and k=5 are the most common choices, but test runs can be made to optimize based on training data. The thing to note about KNN is that it has no training time. It relies on memorizing all of the training data. Then when a document to be classified comes in, it can be compare to each

in the training set to find the closes. In some cases, the use of an inverted index can speed up the results of KNN.

4.3 Other

I put compression based classification in this category. An example in Ruby is Bob Aman's Squish which uses compression to make a classification decision. It trains on the test set and constructs a Huffman encoding for each classification. When it comes time to classify a document, it is compressed and the classification encoding scheme that achieves the best compression wins. This avoids the problem of tokenizing a document, representing it as a vector, and possibly removing potentially useful information like symbols. The downside to Squish is its speed. It is slow enough at this point that I'm not sure how it could realistically be used for a real world classification task.

4.4 Bias-variance tradeoff

Linear and non-linear classifiers represent two sides of the bias-variance tradeoff. Linear classifiers have a high bias and low variance. That is they represent the decision boundary as a hyperplane. Non-linear classifiers can represent a much more complex decision boundary, thus they have low bias and high variance.

5 Testing

Testing is a must for classification tasks. Ultimately you're never sure how each classification task will perform. It's possible that you're trying to classify documents that aren't linearly separable based on their terms. Through testing you can optimize the number of features used and other aspects like weighting a specific classification more than others or deciding which kernel function to use in SVM.

5.1 Accuracy

The fraction of classifications that are correct. In the test set it is how many documents were correctly classified. ($\# \text{ correctly classified} / \# \text{ total}$)

5.2 Cross Validation

Cross validation refers to splitting all of your data into a number of subsets. Called v-fold cross validation where v represents the number of subsets. A standard number to run is 10-fold cross validation. With this example, the set of documents would be split into 10 equally sized sets of documents. We would then train the classifier on 90% (9 of the subsets) of the data and test on 10%. We repeat 9 more times changing the subset that is held out for testing. The numbers are then averaged together to produce an accuracy rating for the 10 test runs.