

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

2007 Apr 14 Sat

Week 10

1. Machine Learning Intro

Wikipedia: As a broad subfield of artificial intelligence, machine learning is concerned with the development of algorithms and techniques that allow computers to "learn". Most of machine learning works on extracting rules and patterns out of massive data sets.

Some parts of machine learning are closely related to data mining and statistics. Machine learning research is focused on the computational properties of the statistical methods, such as their computational complexity.

Machine learning has a wide spectrum of applications including natural language processing, syntactic pattern recognition, search engines, medical diagnosis, bioinformatics and cheminformatics, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing, and robot locomotion.

2. Basic Probability

First we will cover some basic probability.

Conditional probability:

$$P(A | B) = P(A \text{ and } B) / P(B)$$

Definition of independence:

$$P(A \text{ and } B) = P(A) * P(B)$$

Bayes Rule:

$$P(A | B) = (P(B | A) * P(A)) / P(B)$$

3. Classification

Classification is a major task within machine learning. The general problem is as follows: we are given a dataset where each example within that dataset belongs to a certain class. For example, we could be given a set of programs that are either viruses or non-viruses (i.e. they are benign files). We wish to train our system on this dataset so that given a new, unknown example, we can with high accuracy predict the correct class.

This classification task can be generalized to more than 2 classes, but for the purpose of this lecture we will be limiting ourselves to 2 classes; the extension to n classes should be obvious.

Note that oftentimes we are given a large dataset, and it is our task to partition the set into a training subset and a testing subset. Thus we know the correct class for both subsets, which allows us to calculate the accuracy of our classifier.

4. Naïve Bayes: Overview and Derivation

The Naïve Bayes algorithm (hereon referred to as “NB”) uses basic probability rules to build a *model* for each of the classes. Then, given a new example whose class is unknown (or hidden), it calculates the *a posteriori* probability for each class, and selects the highest one as its prediction. That is, given a new example, NB calculates the probability that this example is a member of each of the classes, and chooses the most likely one as its guess. Mathematically speaking:

$predicted_class = class\ that\ gives\ the\ highest\ P(this_class\ | \ unknown_example)$

So, how do we calculate $P(this_class\ | \ unknown_example)$? Remembering Bayes Rule, this expression becomes:

$P(unknown_example\ | \ this_class) * P(this_class)$

Note that we dropped the denominator because all we care about is the one that gives the maximum value, and the denominator is the same for all classes.

Now we have two expressions. Clearly we cannot know the “true” value of these probabilities (if they even exist), and so what we do is guess the values of each; that is, we wish to calculate the *likelihood estimate* of each class, and select the class that gives the *maximum likelihood estimate* as our prediction.

First, we assume that the training set is representative of the ratios of the classes of the entire example space. Given this assumption, a reasonable estimate of $P(this_class)$ would be the ratio of the number of examples in a given class (in the training set) to the total number of examples. That is,

$P(this_class) \sim (num\ examples\ in\ this\ class) / (total\ number\ of\ examples\ in\ all\ classes)$

Next we have to estimate $P(unknown_example\ | \ this_class)$. How do we do this? Here we must tangent into a brief discussion of example representation. For our virus classifier example, each program (example) is represented as a stream of bytes. For a text classifier, each document (example) would be represented as a sequence of words. These components of each example are generally called *features* in the context of text classification, but we will use that term here in the same way.

At this point the NB algorithm makes two very strong (and often unreasonable) assumptions about the data. The first assumption is the *conditional independence assumption*. It says that all of the possible features happen independently of one another. That is, in the text classification context, every single word has a probability of occurring that is independent of every other word. Making this assumption allows us to express $P(\text{unknown_example} | \text{this_class})$ into:

$$P(\text{unknown_example} | \text{this_class}) = P(\text{first_feature} | \text{this_class}) * P(\text{second_feature} | \text{this_class}) * \dots * P(\text{nth_feature} | \text{this_class})$$

That is, the probability of an unknown file given a particular class is equal to the product of the probabilities of each feature given this particular class. The probability of a particular feature given a particular class is simply the frequency that this feature occurs in all the examples of that class divided by the total frequency of all features (of that class). That is:

$$P(\text{this_feature} | \text{this_class}) = \text{num_occurrences_of_this_feature} / \text{num_occurrences_of_all_features}$$

The second assumption NB makes is the *positional independence* assumption. This assumption says that the position of a given feature within a document has no effect on its probability, and is a natural consequent of our first assumption (because we said that every example is independent of every other example).

The last issue we must deal with is features that occur in the unknown file but not in the training set; the probability of this will be 0, which means that it will zero out the entire expression. The standard solution to this is known as *smoothing*: simply assume that every single feature in the unknown file occurred at least once in the training set. If the training set is big enough, it would have very little effect on the features that occur often, and circumvents the zero'ing out issue.

Note that what we are describing here is the *multinomial document generation model*; in English, we are counting the number of occurrences of features. The opposite of this would be the *binomial document generation model*: we only consider the presence or absence of features.

5. Analysis and Discussion of NB

The two assumptions NB makes are quite strong and oftentimes unreasonable. It is easiest to see this in the context of text classification. Suppose we wish to classify a set of newsgroup postings into two classes, baseball and hockey. The conditional independence assumption implies that the probability of words like “baseball”, “pitcher”, “puck”, “goal” are the same regardless of what class the document is. Clearly this is not true. The positional independence implies that “the Yankees beat the Braves” and “the Braves beat

the Yankees” should have the same meaning, when they clearly do not.

It is because of these assumptions that NB is sometimes referred to as “Idiot Bayes”. However, despite these (and other) problems, NB works quite well in practice, generally achieving at least 80-90% accuracy on text classification tasks. It is also relatively fast and simple to implement.

5. Naïve Bayes: Algorithm

Here is NB in algorithmic form:

- i. Train the models
 1. For each class...
 1. For each file in that class...
 1. Scan through the file and count the occurrence of each feature
- ii. Test a new unknown example
 1. Build a model only on that example
 2. Calculate the likelihood estimate for each class
 3. Select the class with the highest score as the predicted class

What is the expression to calculate the likelihood estimate for some class c ? It is:

*the_product_of (the probability of each feature in that class, including duplicates, that occurs in the unknown file) * probability_of_the_class*

Note that because of limited machine precision and the multiplication of vast amounts of very small numbers, in practice the logarithm of the above expression is used:

*sum_of (frequency in unknown file * log(probability of each feature)) + log (probability_of_class)*

6. Assignment

This week's assignment will not involve much (or any) coding. Instead, you will be using the provided code to perform experiments on a given corpus. Read BOTH tasks first before starting, as one does NOT depend on the other, and it would be nice if the class split itself between the two tasks so that we can share and compare results.

Preparation

Begin by downloading “virus-classifier.zip” from the webpage:
<http://www1.cs.columbia.edu/~mwc2110/shp/10/virus-classifier.zip>

After you expand it, you will see the following files/directories:

```
NBModel.java
VirusClassifier.java
virii/
non-virii/
```

First you will need to compile the .java files. To start the system, call VirusClassifier as the main class:

```
java VirusClassifier
```

You will be presented with a menu which should be relatively clear.

One extra detail not previously discussed is the “window size”, which is the length of the features being considered. In general, smaller window sizes results in better accuracy because you see more examples of each type.

Task A

First, randomly select 10 random files as your testing set (you can build this set however you like, 5 virus and 5 non-virus, or 1 virus and 9 non-virus, etc).

Then, training the system with 1 virus and 1 non-virus, and see how well it classifies your testing set. Repeat the experiments with 5/5, 10/10, 30/30, and whatever other training sets you consider interesting. Record the accuracy of your experiments, as well as a (rough) estimate of the amount of time taken in training and testing. Also try uneven training sets, like 1/30, 15/30, 22/22, etc.

Remember that the Naïve Bayes algorithm does not “memorize” the training set; that is, even if you trained a model on a given file, there's no guarantee that it will correctly predict the class of that file in testing.

Task B

Again select 10 random files as your testing set as in Task A.

This time, try varying the window size, and see its effect on classification accuracy. Again keep track of accuracy and performance. Start off with a window size of 1 up to 3.

“Extra Credit”

The code provided is general enough that it can be used on any dataset. If you find this interesting, speak to me about other experiments you can perform on your own, such as using Naïve Bayes to perform authorship detection or topic categorization.

7. References

The lecture materials were liberally adopted from the forthcoming text on information retrieval “Introduction to Information Retrieval”, by Manning, Raghavan and Schuetze,

Chapter 13 “Text classification and Naive Bayes”.

<http://www-csli.stanford.edu/~schuetze/information-retrieval-book.html>

The virus/non-virus dataset was taken from the Columbia CS3134 “Data Structures in Java” course Spring 2007, taught by Professor Shlomo Hershkop.

<http://www.cs.columbia.edu/~sh553/teaching/w3134-s07/>