ml-classifier

More Information on the Naive Bayes Classifier

This document provides a more complete walkthrough of the natural language processing (NLP) and machine learning (ML) techniques behind the classifier implemented in this project, in case you're interested. Nothing in this document should be required to implement the project - it's just extra information if you're interested. There is also considerable overlap between this document and what is described in the main specification.

At a high level, the classifier we implement works by assuming a probabilistic model of how Piazza posts are composed, and then finding which **label** (e.g. our categories of "euchre", "exam", etc.) is the most probable source of a particular post.

There are many different kinds of classifiers that have subtle differences. The classifier we describe here is a version of a "Multi-Variate Bernoulli Naive Bayes Classifier".

Background: Conditional Probabilities and Bayes Theorem

We write to denote the probability (a number between 0 and 1) that some event will occur. denotes the probability that event will occur given that we already know event has occurred. For example, if event is "it will rain today", we might guess that (assuming we hadn't looked outside or at the weather forecast). However, if we were to look outside and observe event , "it is cloudy", then we might believe it is more likely that it will rain. So we could say something like .

The Bag of Words Model

We will treat a Piazza post as a "bag of words" - each post is simply characterized by which words it includes. The ordering of words is ignored, as are multiple occurrences of the same word. These two posts would be considered equivalent:

- "the left bower took the trick"
- "took took trick the left bower bower"

Thus, we could imagine the post generation process as a person sitting down and going through every possible word and deciding which to toss into a bag.

Which Label is Most Likely?

Given a post , we must determine the most probable label from which it could have been generated. This would be whatever label **maximizes the probability** of . Using Bayes' theorem to rewrite the probability, we have:

The denominator does not depend on , so it will be the same for all labels and can be ignored. Thus we want to find the label that maximizes the **probability score**:

These two quantities have intuitive meanings for our application.

- is the **prior probability** of label . It is the probability a randomly selected post has label as . (i.e. How common is each label?)
- is the **likelihood** of post given label . If someone sat down and wrote a post for label this is the probability that post would result.

Let's consider the likelihood in more detail. According to our generative process and bag of words model, we could think of the probability of the whole post as the probability of the particular combination of words chosen to put into the bag:

This is the **joint probability** of several individual events for each word that was included in the post. Technically, we could also consider events for each word that was left out, but this turns out not to matter in many cases (and specifically for our Piazza classifier).

However, we have a problem - the joint distribution is difficult to learn from data. An informal way to think about this is that the compound event of several different words occurring is quite rare. Although we may have seen many posts about euchre, we may never have seen precisely the post "the left bower took the trick".

To solve this problem we introduce the **Naive Bayes assumption**, which posits that the occurrence of each word is **conditionally independent** from the occurrence of other words, given a particular

label. That is, for posts within a particular label, the presence of one word doesn't influence the chance of seeing another. We can rewrite the likelihood of the whole post simply as the product of the likelihoods of its words:

Of course, this is not completely true, and this is why the assumption is called "naive". For example, even given the class "euchre", occurrences of the words "left" and "bower" are going to be related. That said, the assumption is close to true for many words (e.g. "dealer" and "trick" given the class "euchre"), and Naive Bayes classifiers work well for many applications in practice. It is also much easier to estimate individual word likelihoods by learning from data. We'll address this in the next section.

But first, one crucial detail. Our goal is now to find the label that maximizes:

But computing this product can be problematic due to the limited precision of floating point numbers. The probability of a particular word occurring is generally pretty low (e.g.), and when we multiply many of them together the result becomes very close to zero and problems with limited floating-point precision can occur.

We avoid this problem using a neat trick: **work with the natural logarithm of probabilities instead of the probabilities themselves**. (In C++, `std::log()` gives the natural logarithm.) Because the logarithm is a monotonically increasing function, our goal is still to find the label with the highest log-probability, but we can remove the multiplications that can cause underflow problems with this property of logarithms:

So we must find the label with the highest **log-probability** score given the post:

(Note: If multiple classes are tied, predict whichever comes first alphabetically.)

Important: Because we're using the bag-of-words model, the words are only the unique words in the post, not including duplicates!

Learning Classifier Parameters (Training the Classifier)

To compute the log-probability score, we need concrete values for and for each word . These are called the **classifier parameters**. To find these, we train the classifier on a dataset of already labeled Piazza posts (this is called supervised learning). Based on observations from the training dataset, the classifier can estimate the parameters. If the dataset is large enough, these estimates should be quite good.

To estimate the **log-prior probability** of a label , the classifier should observe the proportion of posts with that label in the training set:

To estimate the **log-likelihood** of a word given a label , the classifier should observe the proportion of posts with label that contain the word :

If we need to compute , but was never seen in a post with label in the training data, we get likelihood of and a corresponding log-likelihood of . So instead, use this alternate formula for occurrences of the word through the entire training set:

(Use when does not occur in posts labeled but does occur in the training data overall.)

If the word has never been seen anywhere in the entire training set we just pretend the word had been seen in one document:

(Use when does not occur anywhere at all in the training set.)

Return to the main specification.