Penn CIS 5300- Speech and Language Processing - Chapter 4 Notes

Jonathon Delemos - Dr. Chris Callison Burch

June 17, 2025

This course provides an overview of the field of natural language processing. The goal of the field is to build technologies that will allow machines to understand human languages. Applications include machine translation, automatic summarization, question answering systems, and dialog systems. NLP is used in technologies like Amazon Alexa and Google Translate.

0.1 4.0 - Naive Bayes, Text Classification, and Sentiment

Classification lies at the heart of all intelligence. Deciding how to interpret symbols, words, and actions is an important step in our decision making process. In this chapter, we will discuss the Naives Bayes algorithm and how to apply it to *text categorization.* This involves determining sentinment, the positive or negative orientation of a remark. The most common form of achieving text classification in language processing is through **supervised machine learning**. This is where we have a data set, each bit associated with some correct output. The goal of the algorithm is to learn to map the new observation to the correct output.

Example: Imagine you're a mailroom assistant. Every day, you receive letters with no return address, and your job is to guess whether each letter is a love letter (positive) or a complaint letter (negative) — just by scanning the words it uses.

- Y = (y_1, y_2, y_3, y_3, etc - Set of correct Inputs)
- $y \in$ Y - specific input is in set of inputs
- c = *Class* - This is how you might group a word
- d = *Document* - Think of this as our x input

0.2 Understanding the Problem

We call Naive Bayes a generative model because we can infer an answer based off the given information. These are the variables we will use: we can represent a document d as a set of features $f_1, f_2, ...f_n$: We represent a document as if it were a bag of words. We only keep tracks of the frequency of the words. Instead of x, we will use a document *d.* Instead of an output f(x), we will use c (for "class"). Here, we use the *hat* notation $\hat{c}$ to represent our estimate of the correct class. We also use the arg max operator to mean an operation that selects the argument (in this case, the class c) that maximizes a function (in this case, the probability $P(c \mid d)$):

$$\hat{c} = \arg\max_{c \in C} P(c \mid d)$$

Bayes' Rule:

$$P(x \mid y) = \frac{P(y \mid x) \cdot P(x)}{P(y)}$$

Then we substitute using bayes rule.

$$\hat{c} = \arg\max_{c \in C} \frac{P(d \mid c) \cdot P(c)}{P(d)}$$

$$\hat{c} = \arg\max_{c \in C} \frac{P(f_1, f_2, ...f_n \mid c) \cdot P(c)}{P(d)}$$

Naive Bayes Assumption : this is conditional independence assumption that the probabilities $P(f_i \mid c)$ are independent given the class c and hence can be naively multiplied.

$$\hat{c} = \arg\max_{c \in C} P(c) \cdot \prod_{f \in F} P(f \mid c)$$

Here, we are multiplying the word values in the document to receive a product vector. This result will allow us to evaluate the *sentiment* of the document. Pretty neat stuff.

0.3 4.2 - Training The Naive Bayes Classifier

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c)}{\sum_{v \in W} count(w, c)}$$

Here, the vocabulary V consists of the union of all the word types in all classes. We are summing the count of times this word has been used in a "positive" way across all documents.

0.4 Example:

Let's say you have 1000 total word tokens in all negative documents. The word "horrible" might appear 20 times in those documents. Therefore, we have:

$$P(\text{"}horrible\text{"} \mid negative) = \frac{20}{1000} = 0.02$$

This can be a problem however. If we find a zero as the result, the product of all the word combinations will be zero. A solution offered is to use a *Laplace smoothing* and add one to the numerator.

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{v \in W} count(w, c) + \mid V \mid}$$

Here is the idea in Python.

0.5 4.3 - Worked Example

```
def train_naive_bayes(D, C):
    # D: list of (document, class) pairs
    # C: list of all possible classes
    logprior = {}
    loglikelihood = {}
    V = set()  # vocabulary

    Ndoc = len(D)

    for c in C:
        docs_in_class = [d for (d, label) in D if label == c]
        Nc = len(docs_in_class)
        logprior[c] = math.log(Nc / Ndoc)

        # Flatten all words in class c into one big document
        bigdoc = []
        for d in docs_in_class:
            bigdoc.extend(d)
        # Build vocabulary
        V.update(bigdoc)

        word_counts = {}
        for w in V:
            word_counts[w] = bigdoc.count(w)

        total_wc = sum(word_counts[w] + 1 for w in V)  # Laplace smoothing

        for w in V:
            loglikelihood[(w, c)] = math.log((word_counts[w] + 1) / total_wc)

    return logprior, loglikelihood, V
```

Testing Naive Bayes

```
def test_naive_bayes(testdoc, logprior, loglikelihood, C, V):
    scores = {}
    for c in C:
        scores[c] = logprior[c]
        for word in testdoc:
            if word in V:
                scores[c] += loglikelihood.get((word, c), 0)
    return max(scores, key=scores.get)
```

0.6 4.4 - Optimizations for Sentiment Analysis

Still, we have optimizations to make. A very simple baseline commonly used in sentiment analysis to deal with negation is the following: during text normalization, prepend the prefix NOT to every word after a token of logical negation (n't, not, no, never) until the next punc- tuation mark. Thus the phrase didn't like this movie becomes didn't NOT like NOT this NOT movie , Newly formed 'words' like NOT like, NOT recommend will thus occur more often in negative document and act as cues for negative sentiment, while words like NOT bored, NOT dismiss will acquire positive associations. Furthermore, we can apply a **binary naive bayes** that will discard the word count used in a document. Instead, it will create a binary count of whether a word was used or not. Let's say if a word was used, it receives a 1. Else, a 0. We can add other values to represent specific information. For example, 2 might indicates the value appears in multiple documents.

0.7 4.5-4.7 - Other Classification Tasks

We also need to consider human labels that we are trying to match; i.e. whether or not something is spam. Example: Let's say you are in control of marketing at a company. You might be curious as to what people are writing about your product online. You might use **gold labels** to mark what is considered to be a target response. In both cases, we would need a metric to help us determine how well our machine is performing. This is called the **confusion matrix**, which is a stupid name for a punnett square. The information gathered from these punnett squares help use determine both **precision** and **recall**.

$$Precision = \frac{truepositives}{truepositives + falsepositives}$$

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

The single metric that incorporates both precision and recall that we are looking for to determine the effectiveness of our work is the **F-measure.** It employs a weighted harmonic mean to balance the precision and recall. Using the symbol beta as the the modifier to balance both precision and recall, we can find a reasonable metric to measure success by.

$$F = \frac{(\beta^2 + 1) \cdot PR}{\beta^2 P + R}$$

0.8 4.8-4.9 Test Sets and Cross Validation/Statistical Significance

10 fold cross validation is where we delegate a percentage of our data to be the test set and train on the rest of it. The hypothesis H0, called the null hypothesis, supposes that δ (x) is actually nega-null hypothesis tive or zero, meaning that A is not better than B. We would like to know if we can confidently rule out this hypothesis, and instead support H1, that A is better.

$$\delta(x) = M(A, x) - M(B, x)$$

We would like to know if δ (x) ¿ 0, meaning that our logistic regression classifier has a higher F1 than our naive Bayes classifier on x. δ(x) is called the effect size; aeffect size bigger δ means that A seems to be way better than B; a small δ means A seems to be only a little better.
In this statistical analysis, we can consider the **null hypothesis** to be the case where δ is negative. The **p-value** is the threshold for rejecting the null hypothesis. I like to think of it as the "probability level" of something occurring. Simplified, how likely the null is to be true. If we find a high P-value, that means we can reasonably accept the null hypothesis. If we find a low p-value, something like .001, then it is more likely the null is false. To evaluate all cases for these tests, we can normalize a set of data and calculate if 99 percent of data is less than the threshold p-value. That's just a simple CDF. The most common form of statistical analysis performed to evaluate the effectiveness of the trials is he **Paired Bootstrap test.** This happens naturally when we are comparing the performance of two systems on the same test set. We can pair the performance of system A on an individual observation x_i with the performance of system B on the same x_i.

0.9 Pair Bootstrap Explained

Bootstrapping refers to repeatedly drawing large numbers of samples with replacement from a set. Later, I will implement a table to show the differences between algorithms A and B on a single data set. To measure exactly how surpising the observed δ(x) is, we can use the following formula.

$$p - value(x) = \frac{1}{b} \sum_{i=1} 1_x(\delta(x^i) - \delta(x) \geq 0)$$

This formula looks a little weird. Notice we have both an equal sign and greater than. We use the notation 1(x) to mean "1 if x is true, and 0 otherwise".

0.10 Avoiding Harms in Classifiers

An example of this sort of detection is **hate speech detection**. While the goal of the classifier itself is to reduce hate speech, creating a probabilistic model to detect hate speech can cause significant harm. Incorrectly generating biases, gender sensitivity detection, or racist detection could result in false positives that do more harm than good.

0.11 Questions?

This chapter is also going really slow.. would be a lot easier with a teacher.
I think I get it for the most part.

0.12 Summary

In this chapter, we discusssed Naive Bayes theorem for classification and applied it to text categorization and sentiment analysis.

- Sentiment analysis classifies a text as reflecting the positive or negative orientation that a writer expresses.
- Naive Bayes is a generative model that makes the bag-of-words assumption (positive doesn't matter) and conditional independence assumption.
- Classifiers are based on precision and recall.
- Classifiers are trained using distinct training, included cross-validation in the training set.
- Statistical significance should be used to determine whether we can be confident that one version of a classifier is better than another.
- Engineers should be careful when generating techniques to accomplish objectives. It's very possible to cause more harm than good.