Hidden Markov Models for Part of Speech Tagging

Emre Yurtbay

February 17, 2022

Abstract

Part of Speech (POS) tagging is a classic computational linguistics problem. Consider a sequence of words $x_1, x_2, \ldots, x_n$, and a ”tag set” Q, which contains all the possible parts of speech for a given language. The goal of POS tagging is to output the sequence of part of speech tags $y_1, y_2, \ldots, y_n$, such that tag y_i corresponds with word x_i and $y_i \in Q \forall i$. Unfortunately, this task is challenging due to potentially ambiguous words, such as the word ”saw”, which can function as a verb or a noun, depending on the context in which it is used. POS tagging can be viewed as a disambiguation task; a model that accurately predicts the parts of speech for each word in a sentence must effectively consider each word’s context in a sentence. In this project, we utilize Hidden Markov Models (HMMs) to solve the POS tagging problem. HMMs belong to a class of models called probabilistic sequence models. That is, given a sequence of words, these types of models compute a probability distribution over possible sequences of labels and choose the best label using a dynamic programming decoding procedure known as the Viterbi algorithm. We will show that HMMs are particularly adept at solving the POS tagging problem using standard evaluation metrics in the literature. Furthermore, we provide a comparison to a benchmark in the literature. We will demonstrate our methods using a portion of the WSJ corpus, a collection of news articles where each word is tagged with its part of speech. Finally, we will test the sensitivity of our method by showing how test accuracy increases as a function of the percentage of the corpus used to train the HMM, and also compare out of sample performance as we change the order of the model.

Introduction

Consider the following sentence:

The woman ate the orange.

Any sufficiently bright fifth grader could tell you the part of speech of each word in the above sentence: ”The” is a determiner, ”woman” is a noun, ”ate” is a verb, ”the” is a determiner, and ”orange” is a noun. The ”tag sequence” for this sentence is the following:

D N V D N

At first, it would seem simple enough to train a model to build a tag sequence for any given sentence. However, consider another sentence:

I saw the orange cat.

In the first sentence, the word ”orange” functions as a noun, but in the second sentence, ”orange” is an adjective. It is easy to see that the part of speech of many words is not constant; it depends on the context in which it appears in a sentence. In part of speech (POS) tagging, our goal is to build a model where the input is a sequence of words (that is, a sentence), such as the ones above, and the output is a sequence of tags representing the parts of speech of each of the words of the input sequence. We can formalize the POS tagging problem the following way. Let the inputs be a sequence of words $x_1, x_2, \ldots, x_n$, and a ”tag set” Q, which contains all of the possible parts of speech for the language. Our output is the sequence of part of speech tags $y_1, y_2, \ldots, y_n$, where y_i corresponds with x_i, and each $y_i \in Q$. What makes this particular problem challenging to solve is something we have already alluded to: the problem of ambiguity. Many words in the the English language, and many other languages, can take different parts of speech. Part of speech tagging can therefore be thought of as a disambiguation task; a model that accurately predicts the parts of speech for each word in a sentence must effectively take into account each word's context in a sentence. In this paper, we will use Hidden Markov Models (HMMs) to solve the POS tagging problem, and demonstrate how they are able to achieve extremely high accuracies.

The Model

Hidden Markov Models (HMMs) belong to a class of models called probabilistic sequence models: given a sequence of words, they compute a probability distribution over possible sequences of labels and choose the best label. HMMs require that there exists an observable ”process” X whose outcomes are influenced by the outcomes of some unobservable ”process” Y. In our example, the observable process may be the sequence of words that make up the sentence, while the unobserved process Y is the sequence of parts of speech. We do not see the parts of speech directly; they are the latent or ”hidden” states that we are ultimately trying to recover. The HMMs we will use here consist of several components, each of which will be discussed below. The first component is $Q = \{q_1, q_2, \ldots, q_n\}$, a set of n states. These states correspond with the different parts of speech, (the hidden states) that each word in a sentence can be tagged as. In grade school in America, many of us are taught that there are 9 basic parts of speech in the English language: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection. However, English is a much richer language than this, and we can define more granular parts of speech as well, such a singular proper nouns, predeterminers, gerund verbs, modals, and many, many more. *In toto*, we will be using a set Q with 36 possible ”states” when building our model. Next, we have an $n \times n$ transition probability matrix A, which contains the probabilities from moving from one state (that is, part of speech) to another. The emission probability matrix E gives us the probability an observation w is generated from state q. In our case, the observations are words and the states are parts of speech. Finally, we have an initial probability distribution π over all states. This gives us the probability of each part of speech beginning a sentence. One other important consideration is the ”order” of the HMM. Recall that a model that accurately predicts the parts of speech for each word in a sentence must effectively take into account each word's context in a sentence. To predict the part of speech of a word, the model will look at a number of words before it to help determine parts of speech. If the order of the HMM is 2 and the model is trying to tag word i, it will use the tag of word $i-1$ to help tag word i. (w_i, w_{i-1}) form what is known as a bigram, and thus order 2 HMMs are also known as ”bigram HMMs”. This is also where the ”Markov property” of HMMs comes into play: in bigram HMMs, the tag for w_i only depends on the tag for w_{i-1}, and no other words before or after it. If we want to consider even more context, we can use trigram HMMs, in which tags for w_i depend not only on w_{i-1} but also w_{i-2}. Increasing the order of the HMM may lead to more accurate tags, but also to a much greater computational burden while training. HMMs can achieve remarkably accurate results even when their order is just 2 or 3.

Decoding with the Viterbi Algorithm

For a model with latent states, the task of determining the hidden variable sequence corresponding to the sequence of observations is called ”decoding.” To formalize this somewhat, given as an input an HMM and a sequence of observations $o_1, o_2, \dots o_T$ (i.e. a sentence), we want to find the most probable sequence of states $q_1, q_2, \dots q_T$. In the part of speech tagging example, the goal is to choose the tag sequence $t_1 \dots t_n$ that is most probable given the observation sequence of n words, $w_1 \dots w_n$. This will be achieved using the Viterbi algorithm.

Experimental Setup and Metrics

The data we will be using in this demonstration is called the WSJ corpus, a text file of sentences containing over 1 million words in total. Each line has exactly 1 word and exactly 1 tag. We call this a large collection of text a ”corpus.” Each word is tagged with 1 of 36 parts of speech by a human tagger (presumably a grammar expert). These tags are taken to be the ground truth - though one should note there could be errors made by the human taggers. Our test set is a small set of natural text, untagged. We also have a version of the test set that is tagged by a human tagger that is in the same form as the training corpus. This will be used as our validation set to determine test accuracy. The metric of importance here will be test accuracy, that is, what percent of words in the test data are labelled correctly by the HMM.

We will have 4 experiment types. We will test the performance of trigram HHMs with smoothing, trigram HMMs without smoothing, bigram HMMs with smoothing, and bigram HMMs without smoothing. This setup will allow us to compare the performance of trigram HMMs and bigram HMMs as well as the effect of smoothing, a standard NLP statistical method for combating overfitting.

For convenience, we explain here the experiment for trigram HMMs with smoothing; all of the other experiments will proceed similarly. First, we we load in 1 percent of the training data, and train a trigram HMM on the data using smoothing. Then we will use this trained HMM to predict on the test data, and get a prediction accuracy score. Next, we will load in 5 percent of the entire training data, and train a trigram HMM on the data using smoothing. Again we will use this trained HMM to predict on the out of sample data, and get a prediction accuracy score. We repeat this for 10, 25, 50, 75, and 100 percent of the data. This will give us a sense of how increasing the amount of training data increases the accuracy of our model.

Experimental Results

Table 1 and Table 2 show the results of our experiments.

Table 1: Bigram HMM Results

Percent of Data	No Smoothing Accuracy	Smoothing Accuracy
1%	0.691	0.758
5%	0.827	0.860
10%	0.869	0.890
25%	0.905	0.919
50%	0.927	0.939
75%	0.949	0.954
100%	0.957	0.959

From Table 1, we see that the smoothed versions of the bigram HMM performs significantly better than its unsmoothed counterparts when the training data is small, but this advantage decreases as we use more training data. This points to the fact that bigram HMMs do not overfit to a high degree, so they benefit less from smoothing. When we use all of the training data, the bigram HMM achieves 96

Table 2: Trigram HMM Results

Percent of Data	No Smoothing Accuracy	Smoothing Accuracy
1%	0.646	0.745
5%	0.790	0.866
10%	0.835	0.899
25%	0.859	0.928
50%	0.895	0.949
75%	0.917	0.964
100%	0.932	0.969

percent accuracy, which is very impressive. Table 2 gives us the results for trigram HMMs. The effect of smoothing is much more apparent for trigram HMMs; smoothed trigram HMMs outperform their unsmoothed counterparts by a significant margin. Interestingly, the unsmoothed version of the bigram HMM actually achieves higher accuracies than the unsmoothed version of the trigram HMM. Still, the smoothed trigram HMM seems to have the best performance, achieving 97 percent accuracy when trained on the full data. This accuracy is very impressive; a trigram HMM could probably do better than the vast majority of literate English speakers. Finally, we see that as we increase the percentage of data on which we train each HMM, the accuracy improves, but the rate of increase in accuracy tapers off as the percent of the training data used increases.