Project Report

Viterbi decoding – For parts of speech tagging and an application.

Arvind Krishna Parthasarathy (axp133230)

Introduction:

For classifying text, a basic text processor using a bag of words model would just look at the words that appear on the document and classify based on the probabilities computed in the training set and applying them on the test set.

A different task altogether, would be to find the structure of the sentences in a document, by finding the parts of speech of words in each sentence. Here I implement the Viterbi algorithm – a dynamic programming approach to find the most probable parts of speech sequence for a sentence. I have also used these parts of speech data onto another dataset to give you an application and usage of parts of speech in Machine Learning.

At the end of this report, I would be presenting to you, the accuracy of my implementation on a test set and also use the found tags sequences to give an overview for an application of these data.

Problem Definition:

In this project I am using a corpus that has been downloaded from the internet, called Brown corpus which contains a list of files with tagged sentences. Each word will be of the form "word/POS", and there is a line break after each sentence.

There are 493 unique tags that can be assigned to each word.

Once the model is built, the model can be used to tag any sentence that has been put forth. I have added two ways of implementing it. One is to ask the user for an input sentence in a particular format and then give out the parts of speech sequence for it. The other one is to use a separate test set from on it to give you the accuracy.

As I discussed before, there are a lot of applications in language processing using tags of the words in a sentence. Finding a parts of speech sequence that is accurate therefore becomes a core problem that needs to be solved so that it can be used in the next steps.

Algorithm used:

I have implemented an algorithm known as the **Viterbi decoding** which is **a dynamic programming** approach, to solve this problem.

In the next few lines, I describe the algorithm in greater detail for clarity. I will also put forth the various parameters that are computed from the training set that would be used in the test set.

Given a sentence say, **"I want to play football tomorrow",** what is the most probable sequence of tags that can be associated with it? One way is to find the probabilities of all sequences comprising of n tags and finding the sequence t with the maximum probability among all such sequences. But this is computationally really slow or even impossible.

We therefore look for two parameters, by applying Bayes assumptions.

These two parameters are the **prior probability of a tag sequence P (t1)** and the **likelihood of the word string P (w1/t1).**

$$\hat{t_1}^n = argmax_{t_1{}^n} P(w_1^n|t_1{}^n)P(t_1{}^n)$$

Where $P(w_1^n|t_1{}^n)$ is the likelihood and $P(t_1{}^n)$ is the prior.

Unfortunately it is still computationally hard. HMM taggers therefore make two simplifying assumptions.

1. **Prior probability of tag sequence** is simplified by assuming that, a tag appearing on a sentence only depends upon its previous tag. Bigram assumption.
$$P(t_1{}^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$
2. **The likelihood of a word** string is simplified by assuming that, the word only depends on its parts of speech and is independent of other words and other tags around it.
$$P(w_1^n|t_1{}^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

Therefore our equation becomes

$$\hat{t_1}^n = argmax_{t_1{}^n} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

Which contains two kinds of probabilities, the tag transition probabilities and the word likelihoods.

Computation of probabilities:

1. **The tag transition probability –** This is nothing but the number of times the tags t_i and t_{i-1} appear together divided by the number of times t_{i-1} appears in the training set.
$$P(t_i|t_{i-1}) = \frac{Count(t_i, t_{i-1})}{Count(t_{i-1})}$$
2. The other probability is easy to compute too. It is nothing but the number of times the tag and the word appear together divided by the number of times the tag appears.

$$P(w_i|t_i) = \frac{Count(t_i|w_i)}{Count(t_i)}$$

Viterbi Decoding process:

The task of determining which sequence of variables is the underlying source of some sequence of observations is called the decoding task. It is a standard application of dynamic programming algorithm and looks a lot like the minimum edit distance algorithm.

It sets up n columns, one each to each word in a sentence. In the first column, we set the Viterbi value in each cell to the product of tag transition probability (from start to all other tags), and the word likelihood. All back-pointers are to the start tag.

We then move column by column and for each tag present there we create the probability of transitions from all available tags and assign the maximum to it. The transition to the new tag with the most probability will get a back-pointer back to the old tag.

Pseudo-code for the algorithm: Source Wikipedia.

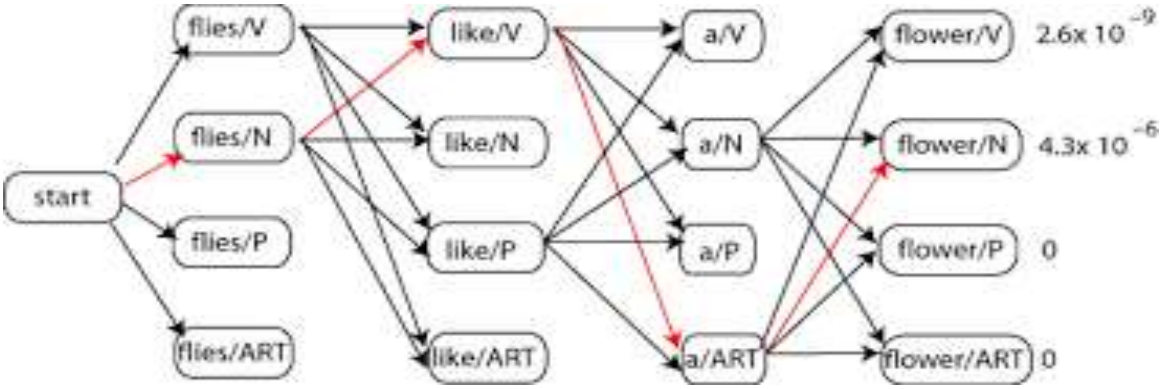
INPUT: The observation space $O = \{o_1, o_2, \ldots, o_N\}$,
the state space $S = \{s_1, s_2, \ldots, s_K\}$,
a sequence of observations $Y = \{y_1, y_2, \ldots, y_T\}$ such that $y_t == i$ if the
observation at time t is o_i,
transition matrix A of size $K \cdot K$ such that A_{ij} stores the transition
probability of transiting from state s_i to state s_j,
emission matrix B of size $K \cdot N$ such that B_{ij} stores the probability of
observing o_j from state s_i,
an array of initial probabilities π of size K such that π_i stores the probability
that $x_1 == s_i$

OUTPUT: The most likely hidden state sequence $X = \{x_1, x_2, \ldots, x_T\}$

```
A01 function VITERBI( O, S,π,Y,A,B ) : X
A02     for each state sᵢ do
A03         T₁[i,1]←πᵢ*Bᵢy₁
A04         T₂[i,1]←0
A05     end for
A06     for i←2,3,...,T do
A07         for each state sⱼ do
A08             T₁[j,i]← max_k (T₁[k,i−1] · Akj · Bjyi)
```

```
A09              T2[j,i]← arg max_k (T1[k, i − 1] · A_kj · B_jyi)
A10          end for
A11      end for
A12      zT← arg max_k (T1[k, T])
A13      xT←SzT
A14      for i←T,T-1,...,2 do
A15          zi-1←T2[zi,i]
A16          xi-1←Szi-1
A17      end for
A18      return X
A19  end function
```

Example:



Implementation of an application using these parts of speech:

I have implemented a simple feature selection algorithm which will take as input, a review and tells us if the review is a deceptive or a truthful review. Deceptive reviews are reviews that are not posted by a customer. Instead, the product owner can create reviews to persuade customers to purchase a product. It turns out that we can obtain about 75 – 80% accuracy by just using feature selection on counting the number of adjectives and personal pronoun usage in a review.

I have implemented a training on truthful and deceptive hotel reviews and computed the number of adjectives and personal pronouns that are present in these two classes of documents. We can clearly see that the deceptive reviews have a lot more adjectives and personal pronouns used on average when compared to the truthful reviews.

The accuracy of such a classifier can be increased by adding additional feature selections and having a weighted average over all of them to give us a more accurate classification.

Data set used:

1. I used a dataset called the Brown tagged dataset which has over 42000 tagged sentences.
2. It is open source and I had to provide my university name to obtain this dataset.
3. The data set has 493 unique tags that have been used over words of these 42000 sentences.

Evaluation:

1. For the Viterbi decoding algorithm –
    a. The algorithm trains on almost 50000 sentences.
    b. Each sentence has words with the associated parts of speech.

    I performed 10 fold cross validation on 45000 training and 5000 testing.

Evaluation criteria:

Number of words correctly classified/total number of words gives us the accuracy of the algorithm

Since there are almost 493 unique tags, testing on 5000 sentences for one fold would take close to 5 minutes. So I am not turning in code with the 10-fold cross validation but the output of the algorithm on a test set.

2. The second part – which is an application, is just going to train on two classes of reviews and do a feature selection to report its learning. We can clearly differentiate between the two classes given these two features.

Results:

Viterbi decoding – I was able to achieve 92% accuracy on predicting the parts of speech tags. This includes the accuracy with even the unknown words.

Unknown words - I could achieve close to 40% accuracy on unknown words, but given the number of tags that are available, I think this is a pretty good achievement. Since there are 493 tags, actual chance probability would be close to 1/493. But my implementation could bring it up to close to 1/2.5 which is a huge difference.

My unknown word accuracy was 32% the first time I ran the algorithm. I searched for the unknown words and found that over 60% of the words had a capital first letter or the word was capitalized. I then ran a few checks and found that over 75% of the words that were capitalized belonged to the noun (nn) category. So I made the tag of those unknown words to nn and found that the accuracy went up by around 5%.

Since there are 493 tags and the dynamic programming approach calculates the Viterbi values for all the tags for each word, the algorithm slows down a bit.

I tested the algorithm on a whole different data set with 47 unique tags and the algorithm can find tag sequences for 1000s of sentences in mere seconds with an accuracy of 96%.

Data set containing	Accuracy	Unknown word accuracy
47 unique tags	96%	51.6%
493 unique tags	91.7%	36.4%

*Accuracy = Number of correctly classified tags / Total number of tags.

Application results: Just the training. The detailed implementation is going to be my future work.

Deceptive review		Truthful reviews	
Average usage of adjectives	Average usage of Personal Pronouns	Average usage of adjectives	Average usage of Personal Pronouns
68.8 / review	40.2 / review	30 / review	22.8 / review

As you can see, a basic feature selection can help in finding deceptive reviews, even though just this information is not enough for us to come to a conclusion.

What do these results mean?

Given that there are 493 unique tags, a sentence of length n will have 493^n possible combinations of tag sequences. For a 5 word sentence this number is 2.9*10^13 combinations.

The fact that this algorithm performs so well and produces 92% accuracy indicates the soundness and the correctness of the algorithm.

Related Work:

As I discussed before, these parts of speech can be used to define a structure to sentences and reduce the ambiguity of a sentence. This helps a great deal in speech recognition and other similar concepts.

I have implemented an overview of an extension to show you an area in which the parts of speech can be used. I used them to find if a review is truthful or not. To be more precise, some reviews may be added by the product owner to persuade a user to buy his product.

Once we detect these deceptive reviews and remove them from the list of reviews, the rating for that product becomes more accurate.

Conclusion:

My implementation of the algorithm could obtain these accuracies.

1. For 47 unique tags – 96% accuracy
2. For 493 unique tags – 91.7% accuracy (no of tags correctly classified).

These tagged sentences have a lot of applications in the real world. Therefore, I would consider this as the base of all other speech and text processing.

References:

1. Viterbi algorithm –
    a. Speech and Language Processing by Daniel Jurafsky & James H. Martin.
    b. Wikipedia
2. HMM POS tagging –
    Speech and Language Processing by Daniel Jurafsky & James H. Martin.
3. Seeing through deception: A computational approach to deceit detection in written communication – By Ángela Almela, Rafael Valencia-García, and Pascual Cantos, University of Murcia (Spain).
4. Method and system for the automatic recognition of deceptive language, by Joan C. Bachenko, Michael J. Schonwetter.
5. About algorithm and its implementations for unknown words – Wikipedia.
6. Viterbi decoding - http://www.inf.ed.ac.uk/teaching/courses/icl/lectures/2006/pos3-lec.pdf.

***** I have added all the tags and their abbreviation to a new file called "tag-information.docx". Please review the file for information regarding the tag. *****