Declaration of Time Spent

Edward

Edward completed these parts of the project

- Implemented
    - First Order Viterbi Algorithm
    - Second Order Viterbi Algorithm
    - Hidden Markov Model
- Wrote these sections
    - Viterbi First Order
    - Viterbi Second Order
    - Code

Edward spent 15 hours on the project

Spencer

Spencer completed these parts of the project

- Coded the first and second order Language Models
- Wrote these sections
    - Language Model- First Order
    - Language Model- Second Order
    - Confusion Matrices

Spencer spent 10 hours on the project

Parts of Speech Tagging Lab Write-Up
by Edward Ekstrom and Spencer Weight

Introduction

In this project, we were asked to make a Parts of Speech analyzer using the Viterbi algorithm. The analyzer would be trained on one set of words with parts of speech tags and then tested on how well it could predict the part of speech that each word was in a test set of words and tags. We were also asked to compare the result of the Viterbi algorithm with the results from a First and Second order Markov chain. Below is the discussion of how well the Language Model Markov chains faired against the Viterbi algorithm.

Language Model - First Order

Initial

The first implementation of the First Order Language Model was based on the code that we went over in class. The original code was unwieldy for large bodies of text, was set up such that each "context" (the previous word) was stored in a list of size one, and was set up to read all the text from the document at once. These three characteristics are the aspects that we improved upon to help the algorithm have better results

Improvements

Our first improvement was to store word counts instead of storing all the words in a giant list for each context. After some coding, we added a block of code that would convert all the word counts into probabilities. We chose to store the word counts as probabilities so that we could condense the word counts into a more manageable data structure that would be faster to store and use for the text predictions.

The second improvement was to change the context variable from a list to a string. Since there was only one word going into the list at a time, it seemed like a better idea to just store the variable in a simpler data structure. This way there didn't have to be any conversions when attempting to use the list variable version of context as a string.

The third improvement made to the initial algorithm was to read all the lines of the document into a list instead of one giant blob of text. This way we could calculate the probability of what word a sentence would start with and then use the starting word as the initial context for the rest of the algorithm to predict with . This seemed like a good idea, but we're not sure if it was much of an improvement over the original algorithm from class.

Result

The final polished algorithm turned out to not be much of an improvement over the class algorithm. Only about half the text was coherent enough to make decent sentences. Reading the text felt similar to reading the writings of a schizophrenic trying to learn English. Calculating the probability of the first word of a sentence proved to not be as helpful as we thought. It merely acted as a springboard for each sentence to start off on, but didn't improve the sentence structure much more than what was seen in class. The result of this algorithm was drastically improved in the implementation of the Second Order Language Model.

Language Model - Second Order

Initial

The initial algorithm for the Second Order language model was created by copying the code from the First Order Language Model and then making some modifications. The context string variable became difficult to use to keep track of the last two words seen. After realizing this, the appropriate improvements were made

Improvements

The major improvement that had to be made to this algorithm over the First Order Algorithm was that now two words had to be kept track of instead of just one. Initially we tried to write the code such that there would be a contextA and a contextB which would be combined to make the key for the Dictionary structure that kept track of contexts and their words. Eventually after some struggle, it was decided that we should use the original method of keeping track of context that was presented in class. This proved to be much easier to code and so this implementation was kept.

Result

The final result of the Second Order Language Model proved to be much better. The text produced flowed better and felt more like it was written by someone who know English much better than the First Order algorithm. We ran the algorithm several times just to be sure that the text being produced made sense multiple times.

Viterbi - First Order

Initial

It took me a long time to understand this algorithm, so my first attempts did not work out very well. At first I did not know how to calculate the emission probabilities or the transition probabilities.

Calculating Probabilities

To compute these two probabilities, I decided to put the data in a format that would help me compute them easier. For the emission probabilities part I put all of the (parts of speech, word) tuples on separate lines with the counts of how many times they occurred together next to the tuple. Then I put all of the parts of speech on lines next to their numbers of occurrences. Then I could just divide the number of times a word was a particular part of speech by the total for that type of part of speech to get the emission probability.

Next I put all of the two part of speech combinations on lines with their number of occurrences. This was so I could calculate the transition probabilities. To calculate the transition probabilities from one part of speech to another I would take the number of occurrences of the two parts of speech in order and divide it by the number of occurrences of the first part of speech.

Road Block

For the algorithm to work we had to decide what to do with transition probabilities that never appear in the training data. We decided just to assign those probabilities to be 0.001 to get our algorithm to work. After we did this, our algorithm worked perfectly.

Viterbi Algorithm

We used the code from the Viterbi Wikipedia page as a starting point for our algorithm. We did have to tweak it considerably to get it to work with our HMM model and our functions for getting emission and transition probabilities.

Result

To get an idea of how our algorithm was performing, we wrote a Python script that would go through our predicted tagging of all of the sentences in the test document and compare them with the actual parts of speech. We had two metrics, namely the number of individual words for which we tagged its part of speech correctly divided by the total number of words in the document, and the number of whole sentences we tagged perfectly correctly divided by the number of total sentences in the test document. We were very pleased with both of these results. On the word by word basis, we were able to predict parts of speech with an accuracy of 0.929916560436. We were able to predict whole sentences flawlessly with an accuracy of 0.251114413076. Using both of these metrics together we concluded that for sentences we did not get perfectly correct, we likely only misclassified one or two words in the sentence.

Viterbi - Second Order

Computing Probabilities

To compute the emission and transition probabilities for the second order Viterbi algorithm we used very similar methods as the first order. For the transition probabilities we needed to add some data to our file with all of the totals. We added all of the combinations of occurrences of three parts of speech in a row. To calculate the transition probabilities in the second order, we would take the possible three parts of speech in order and divide the number of occurrences of those three in that order by the number of occurrences of the first two in their order. Luckily, we already had counts for two parts of speech in their order because we used that for the first order Viterbi algorithm.

Results

Using the second order Viterbi algorithm it took a lot longer to go through each sentence. Seeing as we were running these tests only hours before we had to submit this paper, we were only able to get through 37 of the 1346 sentences with our second order Viterbi. We did the same tests as with our first order Viterbi algorithm and achieved an accuracy of 0.825806451613 on the word by word basis and 0.216216216216 for whole sentences. Since this is based on only 37 sentences, we do not think these accuracies are statistically significant. Had we been able to let it run longer and finish all of the sentences, we hypothesize that the second order Viterbi would have done better than the first order.

Code

To see the data that we used to populate our HMM class, look at the "dataForHMMclass" file in the HMMdata directory of the src we submitted. There you will see what we used to calculate our transition and emission probabilities. To see how we actually did the calculations, look at "HMM.py" in the root directory. To see our implementation of the Viterbi algorithm see "viterbi.py" in the root directory. To see how we calculated accuracies, look at "statistics.py" in the root directory.

Confusion Matrix

We saved our confusion matrix as a separate .pdf file in our submitted zip file. Looking at the confusion matrix, one can see that our first order Viterbi algorithm is accurate. The code we implemented to track our statistics showed us that we had about 93% accuracy for predicting the correct part of speech given a specific word. This is evident in that there are large numbers across the diagonal. and a few smaller numbers scattered everywhere else.

We could not get an accurate second order confusion matrix since we ran out of time while it was running and only finished 37 of the 1346 sentences.

Conclusion

Based on the accuracies that our first and second order Viterbi algorithms produced at labeling words in sentences with their parts of speech, we conclude that our implementations is of high quality. With over 90% accuracy, we think that our implementation could be applied to real world parts of speech tagging and would perform reliably under most circumstances.

Feeback

Edward

I thought this project was interesting but a little too difficult to do in just one week. I spent way too many hours to finish the Viterbi algorithm. I think next time you should go over it a few more times in class before they have to code it up.

Also, this algorithm seems like it could be applied to something much more interesting that parts of speech tagging. That is what was so awesome about the Kalman filter lab. It would be cool if we could implement the Viterbi algorithm on something more interesting.

Spencer

I mostly worked on the Language Model algorithms. I felt like those were a fun project and good standard to compare the Viterbi algorithm to. The result of these algorithms was fun to see, especially when the algorithm came up with a hilarious line of text like this, "lift-ticket sales have been pressuring the Wellington government to impose barriers to a boil in Newark , Del".

I feel like the project description should have shown the formula for the Viterbi algorithm, but I satisfied with the fact that it told me where to find it in the book.