Textual Analysis (590.06): Midterm Project

Aaron Williams – Due: 10/12/2018

**Introduction:**

For this project, we’ll be looking at an alternative (and original) algorithm for tagging based off Viterbi. This method incorporates a different ideology in terms of text-state analysis and attempts to make improvements on Viterbi through assumptions and bidirectional transition states. We will not be making use of advanced neural network techniques or any special libraries beyond numpy. Rather, we will be developing additional functionality which we’ll evaluate through validation on numerous corpus sets. The goal isn’t necessarily to beat the basic Viterbi algorithm, but to determine whether the reasoning behind that functionality is sound. Before we discuss this method further, let’s go over how the problem is being set up.

**Setup:**

The corpuses that we will be using are from the NLTK brown tagged sets. We’ll be using the universal tagset as it makes things simpler and the problem converges a little easier. For training, we’ll stick to a single corpus, in this case ‘adventure’, and use it for all training purposes. The reason that we don’t use multiple corpuses for training is that it simplifies our problem to use only one, and it takes significantly less time. Before training, we’re going to map all unique words and tags to a set of indices and transfer our training to unsigned space. This drastically reduces computation time. For training, we’ll be collecting probabilities for transition states and word states:

Word State:

wprobs[WORD][TAG] = Probability of a specific word having a specific tag

Transition States:

tprobs[0][WORD][TAG][TAG(-1)] = Probability given specific word and previous tag

tprobs[1][WORD][TAG][TAG(+1)] = Probability given specific word and next tag

First these values will be tallied based on our training material, and then converted to probabilities along the TAG axis. A set of these probability matrices will be developed for Viterbi by adding one to all counts (before probability conversion) and converting them to a log scale. For the bidirectional method, we’ll simply add 0.01 to the sum of our counts to guarantee that python doesn’t have issues, but we’ll keep probabilities at 0 if applicable. Additionally, a more basic set of transition probabilities will be developed for words where the probabilities are not known which will be composed from a sum of tprobs along the WORD axis.

Wordless Transition States:

Tsquash[0][TAG][TAG(-1)] = Probability for tag given previous tag

Tsquash[0][TAG][TAG(+1)] = Probability for tag given next tag

**Testing:**

We will be using every category for nltk.brown as a test corpus, including our training corpus ‘adventure’. Each test corpus will be mapped to unsigned space based and pushed through both the Viterbi and the Bidirectional Method. The output set of tags will be compared against the correct tags and accuracy will be determined from that. We’ll also be determining a total of results by tag as well as a confusion matrix for each of our two methods.

**The Bidirectional Method:**

I think it’s important that we go over the bidirectional method and the differences from the Viterbi algorithm before we get in to the results.

Step 1: Create a predicted tag array with tags already filled in which are “guaranteed” based on word array probabilities. Anything with a probability of over 0.9 in the word array is guaranteed.

Step 2: Iterate through these tags. If a non-guaranteed tag is detected, create a subset of it along with all proceeding non-guaranteed tags. Additionally, include the guaranteed tags on either side in this subset if not at the start or end of the corpus.

In the subset:

Step 2a: If not at the start of the corpus, use the first tag which is guaranteed as an anchor and starting state. Iterate through the subset, determining the next tag from a maximum probability composed of transition to (curr tag and word), transition from (prev tag and word), and word probabilities (curr word). Only 1 next tag is determined at each iteration. The final probability is determined using the transition to the guaranteed tag at the end of the subset.

Step 2b: We do the same thing as in Step 2a, but we iterate backwards using the final guaranteed tag as our starting tag. We use our probabilities as before but in reverse order.

Step 2c: We select the better probability of the two and reinsert that set of tags back into the primary set of tags.

Step 3: Tada, we have collected a set of tags. A join probability is not needed for bidirectional.

**Why do what we do in the Bidirectional Method?**

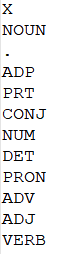
The main reason I decided to use the bidirectional method was that when I analyzed the Viterbi method I noticed that after only a few iterations all possible starting states had converged. By analyzing the best option at each step, the best tag was often the same, and with tens of thousands of words per corpus, I felt the algorithm wasn’t getting the best look at all possible transition states. If a specific word probability was already extremely high, I felt that the Viterbi algorithm would converge anyway, so we might as well get different looks from those anchor points. Additionally, Viterbi seemed too dependent on former state determinations. I think with vaguer word probabilities Viterbi would produce better results than the bidirectional method.

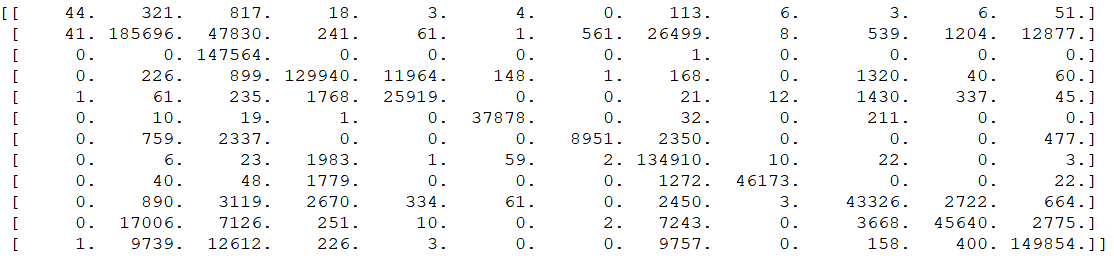
**Results:**

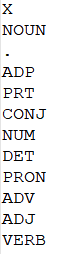
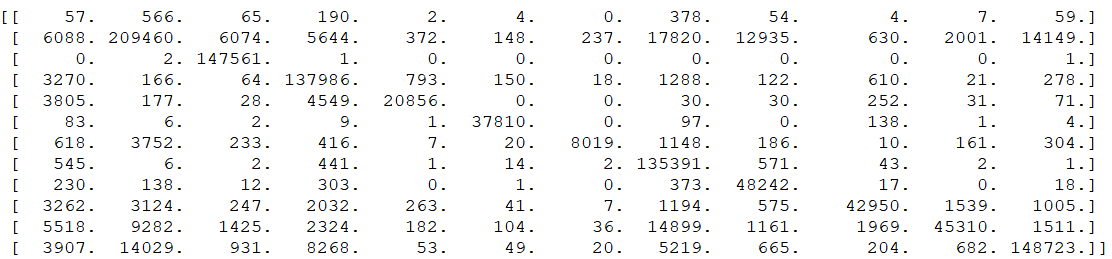
Here are our accuracy results for both methods:

|  |  |  |
| --- | --- | --- |
| **Category** | **Viterbi** | **Bidirectional** |
| Adventure (training cat) | 0.959 | 0.987 |
| Belles Lettres | 0.825 | 0.845 |
| Editorial | 0.807 | 0.822 |
| Fiction | 0.874 | 0.887 |
| Government | 0.768 | 0.800 |
| Hobbies | 0.793 | 0.817 |
| Humor | 0.848 | 0.865 |
| Learned | 0.784 | 0.810 |
| Lore | 0.820 | 0.842 |
| Mystery | 0.879 | 0.894 |
| News | 0.773 | 0.805 |
| Religion | 0.825 | 0.844 |
| Reviews | 0.790 | 0.816 |
| Romance | 0.875 | 0.895 |
| Science Fiction | 0.865 | 0.875 |

Our training category performs best, but in this report, we’re not particularly interested in which categories perform better than others. It’s clear, however, that more similar topics generally produce better results. What’s easy to notice is that the bidirectional method performs on average about ~0.01-0.03 points better on any given test set.

Here is the Confusion Matrix for Viterbi:



And the confusion Matrix for Bidirectional:

One thing that’s worth noting is that is that the bidirectional method seems to misclassify things into less common categories (such as X and PRON). However, it does a better job in classifying the most common category, NOUN, which is where it beats Viterbi in accuracy almost unilaterally. One reason for this might be because better probabilities are given in reverse for things like nouns so more often an unknown word will be guessed to be a NOUN. Here is the breakdown for correct guessed by tag:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tag** | **Viterbi** | **Bidirectional** | **miss Viterbi** | **miss Bidirectional** |
| X | 0.0317 | 0.0411 | 0.494 | 0.998 |
| NOUN | 0.674 | 0.760 | 0.135 | 0.130 |
| . | 1.00(sig) | 1.00(sig) | 0.337 | 0.0580 |
| ADP | 0.897 | 0.953 | 0.0644 | 0.149 |
| PRT | 0.869 | 0.699 | 0.323 | 0.0743 |
| CONJ | 0.993 | 0.991 | 0.00716 | 0.0138 |
| NUM | 0.602 | 0.539 | 0.0595 | 0.0384 |
| DET | 0.985 | 0.988 | 0.270 | 0.239 |
| PRON | 0.936 | 0.978 | 0.000844 | 0.252 |
| ADV | 0.770 | 0.768 | 0.145 | 0.0828 |
| ADJ | 0.545 | 0.541 | 0.0935 | 0.0893 |
| VERB | 0.820 | 0.814 | 0.102 | 0.105 |

The Bidirectional method is more accurate at determining NOUN and ADP tags, but less accurate with NUM and PRT tags. The other tag values are similar between classes.

In terms of misclassification, the Viterbi algorithm misclassifies ‘.’, PRT, and ADV higher rates. Bidirectional on the other hand misclassifies X, NOUN, ADP, and PRON at higher rates. Bidirection seems to be better overall at classifying NOUNS, ‘.’, and ADV, but worse overall at classifying X and NUM considering both categories. This is likely due to the application of back backwards probability terms which are better for separating NOUNs and adverbs but give falsely good probabilities for things that in truth are unidentified numbers.

One thing that Bidirection really doesn’t do well on is the X tag, which it classifies at a relatively high rate but still only gets a tiny fraction right. I’m not sure if this is an artifact from the training set or possibly a bug in the algorithm, but the X class should not be receiving the classification rate that it does from Bidirection.

**Conclusion:**

By changing the methodology of Viterbi, we were able to improve the tag rate a minor bit for this specific problem. We did this by making assumptions of certainty for specific words and by attempting both forward and backward propagation on subsets of the tags. I don’t think that this solution would work for all problems, but it works well for tagging words where there is a decent chunk of words with a specific tag. In terms of operating time, the both methods perform in the O(N) frame and would likely operate interchangeably depending on the specifications of the set. However, unless the training set is not accurate to the target test set, there should be an improvement for Bidirectional.

I think there are still several improvements to be made with the Bidirectional algorithm. For one, the cause of the ‘X’ tag misclassification rate should be identified, and if possible, fixed. I have not tried a variety of training sets which might better identify a solution to this problem as well as further clarify what more needs to be improved. I could also modify the cutoff rate for guaranteed tags to see if that improves the algorithm further. Smoothing is another thing that should be considered for the Bidirectional method as currently ‘guarantees’ can be made by as little as a single tag for a specific word. Perhaps the difference in smoothing between the two methods has a large effect on the final accuracy results.

More training data would have improved both algorithms considerably and training on multiple corpuses would likely improve accuracy. Also, we could have incorporated additional probability matrices to help boost the classification rate just a bit more. Regardless, I imagine a neural network or advanced machine learning algorithm would have done a much better job at making predictions across an entire set. Both the Viterbi and the Bidirectional classification methods, while fast, still do a poor job at getting total classification.