Project 2 Report: Finding Uncertainty With Sequence Tagging

Ryan Enderby (rae83), Liane Longpre (lfl42), Jordan Stout (jds459)
Kaggle team name: rae83_lfl42_jds459

Sequence Tagging Model

Pre-Processing:

Before we could begin training our model, we had to assemble data from the text files into structures that we could use. We began by iterating through all files within a given training folder, and creating two lists from the documents, with appending each document onto the lists. One list is just of the tags (the third item in each row), and the other is a list of tuples such that each tuple is (tag, word) from a given row. At this stage, we ignored empty lines signifying breaks between sentences, and treat everything as one continuous list.

HMM:

For our sequence-tagging system, we implemented our own hidden markov model without using existing packages. In our model, the observations are word tokens. The states are B, I, and O tags. The transition probabilities are the bigram probabilities of the next tag sequence (ie P(I|B)). The emission probabilities are the probabilities of the word given the IOB tag (ie P("perhaps"|O)). We then use the Viterbi algorithm to find the most likely IOB tag sequence.

Our function that implements the Viterbi algorithm takes in input transition probabilities, emission probabilities, initial probabilities, states, and a sentence. The sentence is a string of words to be tagged. States is a list of possible states [B,I,O]. Initial probabilities are the probabilities of the first word being each of the possible states. This is created by calculating the unigram probabilities of each of the tags. Transition probabilities are calculated by creating a list of tags in order from the training data, and creating bigrams of this list. The transition probabilities are maintained as a dictionary that maps bigrams to probabilities. Emission probabilities are created by making bigrams of each word-tag pair. The emission probabilities are also maintained as a dictionary that maps bigrams to probabilities. Within the implementation of the Viterbi algorithm, the probabilities of words at intermediate steps for each word in the word sequence is maintained as a two-dimensional array. The function then outputs a string containing the most likely tag sequence, ie 'OOBIIIOOO', as well as the probability of this sequence.

Post-Processing:

To edit the test documents, we iterate through each file in the test folders, running Viterbi on each sequence of words between the blank lines in a file (e.g. a sentence). We then convert the output string from Viterbi into a list, creating a list-of-lists for each document. Then, we iterate through each file in the test folders provided, as well as creating a corresponding file in a 'test-public-edited' folder or 'test-private-edited' folder, writing our Viterbi output plus the first two items in each row from the input test document. It is important to have created the 'test-private-edited' and 'test-public-edited' folders for this output writing to work!

Baseline:

To create a preliminary baseline system, we iterate through all of the training documents and create a dictionary of cue words. Every time that we encounter a word with a cue tag, we add it to the dictionary. If the word is already in the dictionary, we add +1 to the value of that key. The dictionary contains the words as keys and their values being the number of times the word was marked as a cue. We then use a subset of the dictionary, our most successful submission using the 200 most frequent cue words, on our test data to tag uncertain words. If we encounter a word in a test document that is in the specified cue words subset, we tag it as B or I (depending on whether or not it is the first occurrence of a cue word in the sequence), and an O if it is not in the cue words list. We classify a sentence based on whether or not it has an arbitrary number of cue word tags in it. In this case, out best submission was with greater than 3 cue words as the cutoff.

Extensions

We resampled from our training data as an extension. As mentioned in the results below, we saw that our model regularly falsely predicted 'O'. As such, we chose to do an extension that could help mitigate the chances of over-guessing zero. We took two approaches, one that decreased the occurrence of documents with no uncertain cues, and one that increased the occurrence of sentences with uncertain cues. Finally, we combined these two approaches to see how it impacted our results. We predicted that both methods would improve the accuracy of our system by increasing the likelihood of prediction 'B' and 'I' tags on test cases.

To downsample training data with no uncertainty cues, we set up our code to exclude a certain fraction of training documents with no uncertainty cues. We chose to omit entire documents as opposed to only sentences with no uncertainty so that we would retain documents that have both certain and uncertain sentences, thus giving us both certain and uncertain documents within each domain. When we encounter a training doc with no uncertain cues we exclude it with an arbitrary probability with relation to a threshold we set (e.g. 0, 0.25, 0.5, 1, etc.), calculated by

using numpy.random.uniform(). We tested our model excluding different fractions of training documents with no uncertain cues, and found that our most accurate model, both in terms of sequence and sentence level prediction, was when we excluded all documents meeting this criteria. Excluding ¼ and ½ of the documents increased our Kaggle score, but the highest was still with complete exclusion of uncertainty-free documents.

To upsample, we tried doubling and tripling the occurrence of sentences with uncertain cues, e.g. if a sentence has an uncertain cue, we add multiples of it to the training data. We tested using our most successful downsampling method - complete uncertainty-free document exclusion - paired with altering the sentence multiples. The change from adding sentence multiples, both double and triple, was miniscule. It did not improve our sequence level prediction in either case, but did improve our sentence level prediction. This makes sense, as we were increasing uncertainty-bearing training examples at the sentence level.

Results

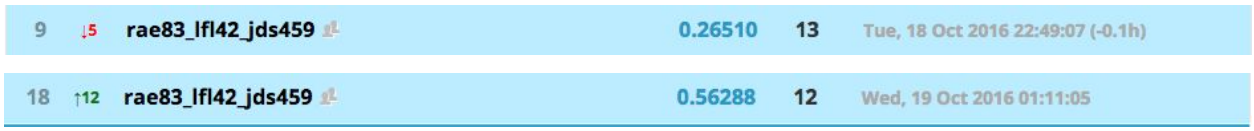
The biggest error of our initial system was in producing false negative cue tagging. Often cue phrases were tagged as O's instead of cue words. This fueled our idea for our extension of resampling. This error is most likely due to the highly imbalanced proportion of cue vs non-cue words in the training data. Because most words in the training data are not uncertainty cues, we see that our model produces a distribution of O's that is too high. It predicts O-tags much more often than it should. For this reason, we decided to experiment with resampling. We first experiment with downsampling documents. For each document that contains no uncertainty cues at all, we remove the document from training. We decided to downsample documents (as opposed to sentences) because within a document, it is reasonable to assume the document will contain words with a similar domain. Thus, in document that has uncertainty cues, it will contain words that are both certain and uncertain in the same domain, which should be helpful in creating a stronger model.

We then experiment with only removing a fraction of the documents that do not contain any uncertainty cues. The documents omitted are selected randomly based on a threshold we set. The results of these experiments are delineated below in our results table. The best results come from removing all documents with no uncertainty cues. Next, we experiment with upsampling sentences. In addition to removing all documents with no uncertainty cues, for each sentence that contains any uncertainty cue(s), we add the sentence twice into our training data. When experimenting with different multiples we find that adding the sentence twice improves our model for sentence prediction, but worsens our model for sequence prediction. We find that adding the sentence more than twice worsens performance in both prediction tasks. We find that the best combination of downsampling and upsampling is to use a threshold of 0.5 to

downsample documents with no uncertainty cues in combination with a doubling of increasing sentences with uncertainty cues. It makes sense that our model prediction improves at the sentence level rather than individual cues and sequences, because we are increasing the probability of entire sentences and not providing more data in order to detect more cues. The table below shows the results of our baseline system, our initial implementation, and experimenting with different resampling methods.

Model Variation	**Kaggle 1: Uncertain cues**	**Kaggle 2: Uncertain sentences**
Baseline, before tuning number of cue words and criteria for sentences	0.00028	0.47806
Baseline, after setting to 200 cue words and n > 3 cue words for sentence classification	0.01629	0.48348
Model trained on all documents	0.25369	0.51815
Model trained only on documents with uncertain cues	0.26510	0.55818
Model trained on documents with uncertain cues and ~½ the documents with none	0.25863	0.54089
Model trained on documents with uncertain cues and ~½ the documents with none	0.25770	0.54120
Model removes all training documents with no uncertain cues, doubles sentences with uncertainty	0.26113	0.56104
Model removes all training documents with no uncertain cues, triples sentences with uncertainty	0.25292	0.55679
Model removes ~½ documents with no uncertain cues, doubles sentences with uncertainty	0.25529	0.56288

Screenshot of top Kaggle submission:



Workflow

Code:

We wrote the code in iPython notebooks, and exported as Python 2 scripts. It requires the following packages: os, csv, itertools, operator, numpy and collections. Baseline.py contains code to write edited docs to new versions of the existing folders with '-edited' appended to the name and execute a baseline system. It is important to have created the 'test-private-edited' and 'test-public-edited' folders for this output writing to work! HMM.py uses the already-edited training docs (e.g. CUE-1 etc. have been replaced with BIO notation) to train and run our Hidden Markov Model.