APML - Ex2

Vitaly Rajev, 320929227

1 Theoretical Questions

1.1 - MEMM "Contains" HMM

- Cancelled

1.2 - Higher Order Markov Model

- Cancelled

1.3 - Energy Based Model Gradient

1.4 - MLE for HMM

- Bonus

2 Practical Exercise

2.2.1 - Processing the Data

Loding and pre-processing of the data were done in the $load\_data(test\_set\_fraction = 0.1, rare\_threshold = 5)$ function. In this function, the data is first loaded from the pickle files, then - since they are used in the process - the START, END, and RARE words were added to the list of all words, and the START and END tags were added to the PoS tags list.

Then, to each sentence, the START word was added to the beginnig of the sentence, along with the corresponding tag, and the END word was added to the end of the sentence, again with the corresponding tag. After this step, the data was split into training and test sets, according to the fraction specified in the function call/signature. After splitting the data, the training set was pre-processed and all words that appeared less times than the threshold specified in the function call/signature were replaced by the RARE word, and removed from the list of all words.

After this step, the function returned a list of all possible PoS tags, a list of words appearing in the training set, a training and test sets that were each a numpy.ndarray object containing [sentence, tags] lists (not numpy.ndarray since the sentences were of different lengths).

2.3 - Baseline Model

The Baseline class receives a list of possible PoS tags, and the list of words in the training set. The training of the class is done with the $train(self, training\_set)$ method, which trains the model over the received training set, under the assumptions that every tag appearing in the training set also appears in the possible PoS tags given to the constructor, that every word appearing in the training set appears in the list of possible words passed to the constructor. The training method calculates $\mathbb{P}(y)$ for each $y \in PoS$, and $\mathbb{P}(x|y)$ for each $(x, y) \in Words \times PoS$, and the resulting multinomial and emission probabilities are saved in the model.

The MAP method receives a list of sentences, and returns the most likely PoS tagging for the sentences, assuming the model was trained beforehand.

2.4 - HMM

2.4.1 - MLE Estimation

The HMM class receives a list of possible PoS tags, and the list of words in the training set.

The training of this model is done with the $train(self, training\_set)$ method, which trains the mdoel over the training set, assuming that the tags are all in the PoS list passed to the constructor, and that every word in the sentences appears in the list of words passed to the constructor. The training method calculates $\mathbb{P}(y_i|y_j)$ for every $(y_i, y_j) \in PoS \times PoS$, and $\mathbb{P}(x|y)$ for every $(x, y) \in Words \times PoS$. The calculated estimations are saved in the model.

2.4.2 - Sampling From a Generative Model

An example for a sentence generated by the trained HMM after training over the entire training set: tags: ['*STS*', 'NNP', 'RB', 'VB', 'NNS', 'VBD', ':', 'RB', 'VBN', 'NNS', 'VBD', 'DT', 'JJ', 'NNS', 'IN', 'NNS', 'JJ', 'IN', 'DT', 'NNS', 'CC', 'PRP', 'VBZ', 'RB', 'VBN', ',', 'DT', 'NN', 'IN', 'JJ', 'JJ', 'NN', 'CC', 'JJ', 'JJ', 'NNS', 'TO', '*ENDS*']

sentence: ['*STW*', 'Robert', 'only', 'own', 'creditors', 'predicted', ';', 'also', 'convicted', 'areas', 'said', 'The', 'subject', 'allies', 'as', 'claims', 'first', 'into', 'the', 'utilities', 'and', 'it', 'is', 'neatly', 'engineered', ',', 'a', 'look', 'of', 'changed', 'commercial', 'column', 'and', '*RARE_WORD*', 'corporate', 'advocates', 'to', '*ENDW*']

We can see that the sentence makes a little sense as a sentence in english, in certain parts and with grammatical mistakes, and it makes more sense from a PoS tag perspective.

2.4.3 - Inference

The inference is done in the $viterbi(self, sentences)$ method, which predicts the most likely PoS tagging for the sentences using the viterbi algorithm. The viterbi method calculates the cumulative probability table, such that in each 'layer' i of the table, in each PoS index (corresponding to a tag), is a list of the form [cumulative log probability, previous index], where the cumulative log probability is the maximal cumulative sum of log-probabilities such that the tag of the i'th word is the index of the list, and the previous index is the index of the PoS corresponding to the previous word. In the 0'th layer, corresponding to the START word, is the emission probability of the START word, since there is no transition into the beginning of the sentence.

In addition, the viterbi method assumes the model is trained.

2.5 - MEMM

2.5.1 - Inference

Similarly to the inference in the HMM, the viterbi method of the MEMM uses the viterbi algorithm to infer the most likely PoS tags for the given sentences. The difference between the methods if that for the MEMM, instead of calculating the probabilities using the learned transition and emission probabilities, the model uses a given feature function ϕ and a weigths vector w, to calculate the optimal probability and PoS tag.

2.5.2 - Learning w

Learning w is done using the perceptron algorithm learned in class. The perceptron is a method of the MEMM and is given a training set fitting the list of words given to the constructor of the class. The goal is for w to represent the emission and transition probabilities, normalized as seen in class by $\sum_{y'} exp(w^T \phi(y', y_k, x_i)$ for each $k \in \{1, ..., |PoS|\}$, as well as additional features we would like to add such as suffixes, capitalization, and length.

2.5.3 - Making Own Model

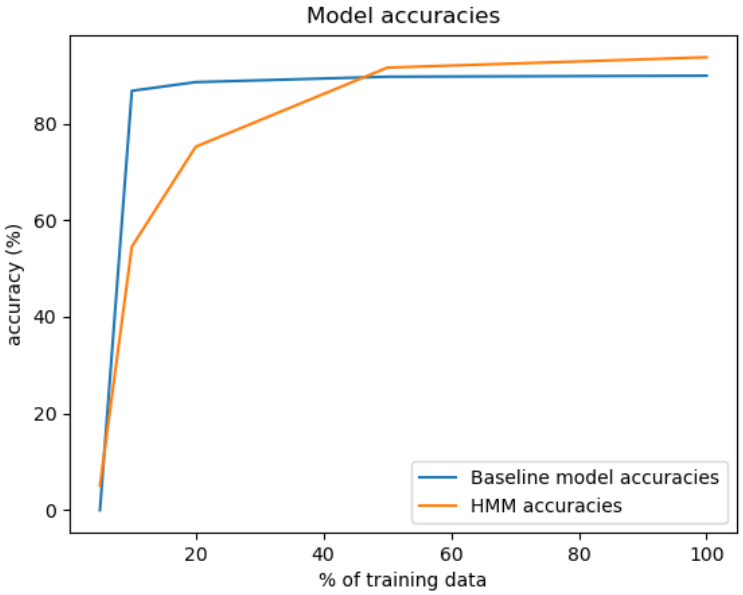
In making my own model, the feature mapping I chose consisted of the basic emission and transition probability mapping, and of 6 additional features - common suffixes for verbs, nouns, and adjectives, the length of the word, weather its first letter is capitalized, and weather the word is a number.

2.6 - Model Comparison Comparison of the models was done as follows:

The data was split into a 90% training set, and a 10% test set.

The baseline model and the HMM we trained over [5%, 10%, 20%, 50%, 100%] of the training data, and tested after each training session.

The test accuracies of each model were as follows:



As we'd expect, the test accuracy improved as we increased the number of examples. In addition, we can see that the baseline model converges faster than the HMM, but on the other hand the HMM converges to a higher accuracy (~93-94% on the HMM, and ~89% on the Bseline model).

The evaluation of the MEMMs (one without additional features, and one with additional features as described) was done using strictly 10% of the training data, since the runtime of the MEMM viterbi (and the perceptron, as a result) is significantly longer than that of the baseline model or the HMM.

The test accuracy of the MEMM without additional features is: 88.6386 %

The test accuracy of the MEMM with additional features is: 88.6386%

Adding the features in this case did not cause the test accuracy to change, this is possibly due to training the models for a small number of sentences or epochs, not adding a sufficient number of features, or adding incorrect features which did not help with training.

In addition we can see that for the same size of training data, both MEMMs outperform the Baseline Model and the HMM, which can indicate that with additional training data, the MEMMs will converge to a higher test accuracy than both other models.