50.007 Machine Learning
Design Project
Group 9

Name	Student ID
Ayman Khan	1006313
Mateo Jalen Andrew Caledron	1006042
Patrick Phone Myat Mo	1006084
Cheong Hao Shaun	1005881

Introduction

Our primary objective in this project is to develop a robust sequence labeling model tailored for informal texts, utilizing the foundational concept of the hidden Markov model (HMM) discussed in class.

This report will delve into each component of the project, detailing our methodologies, implementations, and evaluations for each of the four parts of the Design Project, as well as the results obtained.

Instructions to Run Code

Using the submitted zip file in e-dimension that came along with this report, you can run the Python Notebook files that represent each part of the project individually. Run all cells for each Python Notebook file and note that you might have to click run all cells again, because some of the helper functions in the Notebooks, especially for part 2, are located at the bottom of the file. If there are issues with this, we strongly suggest you run the cells at the bottom to get the required emission probabilities first. In this way, when you run the Python Notebooks, you will be able to obtain the results as seen in this document.

Another way to access the code and the output files, is through this link to our github repository: https://github.com/TCReaper/ML_proj/
More instructions can be found in the repository's readMe.

Some key points before we get into the approach for each part

In order to properly go through with the requirements of each part of the project, we had to ensure that we had functions that could process input files and write output files. We noticed that the text files were separated into sentences by empty lines, so this is the convention we stuck to.

Another thing is, we made sure to properly integrate the use of unknown words in our decoding algorithms, which is why we replace all the unknown words that are not in the vocabulary learnt, with the token "#UNK#". Although this might have reduced the scores obtained when decoding, we believe that doing it this way was as intended by the design project details given to us. We point to the line that states "During the testing phase, if the word does not appear in the training set, we replace that word with #UNK#". This is why there are words that are "#UNK#" in the output files.

Part 1

To get started, we must first analyze the training data given to us and evaluate the emission parameters using the Maximum Likelihood Estimation (MLE). We created and used a helper function to get and store the number of instances of each tag (tag_count), number of instances of a word-tag pair (word_tag_count), all the words/observations in the dataset (vocabulary) as well as all the sentences in the dataset (sentences).

We then used the MLE Formula shown in Fig. 1 to calculate the emission probabilities. This does not handle the case for unknown words. Since test data is usually different from the training data, we need to create a method for handling unknown words. Thus, when calculating the emission parameters, we introduced a special token word, "#UNK#", and an assumed occurrence of k = 0.1. Everytime we find an unknown word, we treat it together with all of the other unknown words and assign corresponding emission parameters as shown in Fig. 2.

$$e(x|y) = \frac{\texttt{Count}(y \rightarrow x)}{\texttt{Count}(y)}$$

Fig 1. MLE Formula

$$e(x|y) = \begin{cases} \frac{\texttt{Count}(y \rightarrow x)}{\texttt{Count}(y)+k} & \text{If the word token } x \text{ appears in the training set} \\ \frac{k}{\texttt{Count}(y)+k} & \text{If word token } x \text{ is the special token } \texttt{\#UNK\#} \end{cases}$$

Fig 2. Emission Parameter Assignment

Lastly, we implemented a simple sentiment analysis system that would produce a tag (y*) for every word (x) in a sentence. This tag is assigned based on the tag that has the maximum likelihood of occurrence given that particular observation. Our results are as follows:

#Entity in gold data: 13179
#Entity in prediction: 17673

#Correct Entity : 9882
Entity precision: 0.5592
Entity recall: 0.7498
Entity F: 0.6406

#Correct Sentiment : 8850
Sentiment precision: 0.5008
Sentiment recall: 0.6715
Sentiment F: 0.5737

We can judge the performance of our prediction based on the F Score yielded from the evaluation script. We would ideally want a higher F score, however due to our predictions being made solely on the emission from one word to another and not considering the transition between states, it results in a lower score. This is particularly important for sentiment analysis as our model must make sense of the context of the word, since its label can differ based on it.

Part 2

In the second part of this project, we aim to build on our sentiment analysis by estimating the transition parameters from one state (label) to another. Once again we use the Maximum Likelihood Estimation (MLE) method to calculate these parameters. This time, we store these transitions between states in transition_count, a dictionary of the form (u,v): count where u represents the initial state and v represents the transition state, as well as the number of times we can see this particular transition.

For example, ('START', 'B-NP'): 4966

We then use the function estimate_all_transmission_probability to calculate the transition probabilities using the MLE formula in Fig. 3. We then store these in a variable transition_probabilities in the form (u,v) : probability.

For example, ('START', 'B-NP'): 0.6480490669450607

$$q(y_i|y_{i-1}) = \frac{\texttt{Count}(y_{i-1}, y_i)}{\texttt{Count}(y_{i-1})}$$

Fig 3. MLE Formula

Now that we have the transition parameters, we can implement an algorithm to compute the sentiment tag for each word in a given sentence. We use the Viterbi Algorithm to find the tag with the maximum likelihood of occurrence. This is a bottom-up dynamic programming method to calculate the highest probability path of transitioning between states and emitting the observed words corresponding to each state. We store the maximum probability of reaching state k, then iteratively calculate the probabilities of transitioning into state v, and store the maximum value. We repeat this until we transition into the ‘STOP’ state, then backtrack to find which label y produces the maximum probability and thus generate labels for the entire sequence of observations.

We start from the “START” label, iteratively calculate the probabilities of transitioning into state v and generating the given observation, then store the maximum value. We then use this value as the probability of reaching that state k and repeat the process for the next observations until we get to the “STOP” state. From our stored matrix of states and their probabilities, we can form a path from ‘START’ to ‘STOP’ containing all the labels for the observations, thereby labeling the sentence.

Note the way we handle the unknown words in our viterbi algorithm: We replace each unknown word in the input sentence, to #UNK# and then ensure our emission parameters have probabilities for (#UNK#, tag) pairs, which will allow our viterbi to better consider the emission

and transition probabilities when unknown words are encountered. This is why our results for this section are fairly accurate.

Upon running this algorithm on the test set our results are as follows:

```
#Entity in gold data: 13179
#Entity in prediction: 14279

#Correct Entity : 10858
Entity  precision: 0.7604
Entity  recall: 0.8239
Entity  F: 0.7909

#Correct Sentiment : 10056
Sentiment  precision: 0.7043
Sentiment  recall: 0.7630
Sentiment  F: 0.7325
```

Our F Score of 73% indicates a better performing model when compared to just using emission probabilities. Due to relatively higher transition and emission probabilities as well as shorter sentences, thereby shorter paths, we do not end up losing out accuracy to numerical underflow. Thus, our implementation does not need to consider different methods to account for it. However, if this was a more substantial issue, we could fix it using logarithmic transformation of probabilities, scaling factors or smoothing.

Part 3

For part 3, our aim was to design an algorithm that will help us compute the 2nd best label sequence for a given observation. We came up with two possible approaches to compute this. Either we find a labeling sequence that is completely different from the best output (such that none of the states in the best sequence are repeated in the 2nd best) or we consider an option where just one of the labels is different from the best case. We felt that the latter would be a better representation of a 2nd best labeling sequence, thus we implemented this with a modified version of the Viterbi algorithm.

In this algorithm, we compute the best path as usual, and it is in the backtracking step that we find the 2nd best label. Normally in the Viterbi algorithm, we just store the label for which we have the maximum probability of occurrence (argmax_y) however we now want to be able to also store the label for which we have the 2nd highest probability of occurrence, thus we store this in the variable second_best. The way we find the 2nd best label is by initially assigning it to None and updating its value with the value of the best label only when the variable storing the best label is updated as we try to find the max probability by looping through the values from the different labels.

When adding the labels into the final output, we add the first available 2nd best label within a state change when backtracking instead of the best backtrack label in the position. After traversing to one single 2nd best label, we will reroute the traversal back to the original best path, hence only absorbing the cost of rerouting to a second best label once. With this, we have generated the second best label sequence. Accuracy results for the second best sequence:

#Entity in gold data: 13179
#Entity in prediction: 24725

#Correct Entity : 7231
Entity precision: 0.2925
Entity recall: 0.5487
Entity F: 0.3815

#Correct Sentiment : 2114
Sentiment precision: 0.0855
Sentiment recall: 0.1604
Sentiment F: 0.1115

We can see here there is a significant drop in the F-Score when compared to the best sequence. This is expected since the labels are no longer ideal. However, to improve this score for the 2nd best sequence, we can consider not swapping just the last label but rather finding which label swap in any one of the states has the closest to maximum probability score and assigning that in the second best label. This proved computationally complicated in our scenario due to the number of comparisons it required, thus we only considered the first swap.

Part 4

For the design challenge, we opted to use the max-marginal forward-backward algorithm over the Viterbi algorithm to leverage its ability to provide detailed probabilistic insights into each state's likelihood at every time step, rather than merely identifying the most likely overall sequence of states.

While both algorithms are probabilistic, the Viterbi algorithm provides a deterministic output (the single best sequence), based on probabilistic calculations. In contrast, the max-marginal forward-backward algorithm offers a broader probabilistic perspective by detailing the likelihood of each possible state at every point in the sequence, making it more suitable for applications where such detailed probabilistic information is valuable. Another key thing to note is that the scores in the max_marginal algorithm, actually calculates the probability of each state, independent of the path decisions, giving a more comprehensive view of state probabilities considering all possible paths, not just the most likely path. Furthermore, the max marginal forward-backward algorithm that we implement is more versatile and can handle unknown words better for new test sets in general. In the case of this project, which entails speech tagging, the max-marginal forward backward decoding is better as with its probabilistic information, it can generally help with handling ambiguity, model confidence and can tackle unknown words more efficiently making it "better" than the Viterbi Algorithm.

To address the potential concern that this max_marginal forward-backward algorithm might not yield better results than the Viterbi implementation we had earlier on, we couple our max_marginal forward-backward algorithm with a new and better way to handle unknown words, which gives us more accurate emission parameters. In this way, we will be able to yield a more accurate sequence of tags compared to part 3. This would also have the added benefit of being able to indicate how certain we are about each state, which can be very useful for certain scenarios.

We also would like to take this chance to try implementing a model other than the Viterbi Algorithm.

New model used: Max-Marginal Forward-Backward

We make use of the max-marginal decoding algorithm that employs the forward-backward method to calculate individual state probabilities at each time step in a sequence, by computing both forward (α) and backward (β) probabilities. It selects the most likely state for each position by identifying the state that maximizes the product of these probabilities (αu(i) * βu(i)). For this, we computed the forward and backward probabilities as taught in the lectures, and stated the base cases before iteratively computing using bottom up dynamic programming approach. In this way, we can find the scores at each state and at each time step.

New method to smooth emission parameters: Absolute Discounting

Absolute discounting is a smoothing technique used in language modeling to address data sparsity issues by adjusting the counts of observed events. It works by subtracting a fixed discount factor from the counts, ensuring that no count becomes negative. This redistribution of probability mass from observed to unseen events helps the model generalize better, particularly in the presence of rare or unseen words. By allocating a portion of probability mass to unseen events, absolute discounting improves the model's ability to make accurate predictions, even in scenarios where training data is limited or incomplete. Overall, absolute discounting enhances the robustness and performance of language models by providing more reliable probability estimates and mitigating the impact of data sparsity.

```
#Entity in gold data: 13179
#Entity in prediction: 14806

#Correct Entity : 10733
Entity  precision: 0.7249
Entity  recall: 0.8144
Entity  F: 0.7671

#Correct Sentiment : 9774
Sentiment  precision: 0.6601
Sentiment  recall: 0.7416
Sentiment  F: 0.6985
```

We can see here that this implementation of max-marginal decoding does not perform as well as the Viterbi algorithm that we implemented in part 3 by a margin. Hence, while we are sacrificing some accuracy in this context, we believe this method is still better due to the probabilistic advantages mentioned above. Overall, the trade off would result in a more accurate performance over many scenarios, so we are confident in this method that we have chosen.

We can also use the emission parameters generated from this absolute discounting method to run the Viterbi algorithm as it should handle the unknown words better.

This change resulting in the following accuracy results:

#Entity in gold data: 13179
#Entity in prediction: 14270

#Correct Entity : 10856
Entity precision: 0.7608
Entity recall: 0.8237
Entity F: 0.7910

#Correct Sentiment : 10082
Sentiment precision: 0.7065
Sentiment recall: 0.7650
Sentiment F: 0.7346

Here, we can see that just by using our absolute discounting method together with our original Viterbi implementation, we are able to achieve better, more accurate results than our implementation with the given smoothing technique in part 2. However, since we are also considering the probabilistic advantages of the max marginal forward-backward algorithm, we still opt to use the new algorithm that we implement in this part 4a, as it is the better model in our opinion. Furthermore, the max marginal forward-backward algorithm that we implement is more versatile and can handle unknown words better for new test sets in general.

Conclusions

Throughout this design project, our group developed and evaluated a sequence labeling model using a Hidden Markov Model (HMM) tailored for informal texts. Our aim was to accurately predict tags for sequences of words, a key task in natural language processing for sentiment analysis. Here are our principal findings and conclusions:

Key Findings:

- Model Development and Evaluation: To manage the inevitable challenge of unknown words, we introduced a special token "#UNK#" with assumed probabilities, significantly enhancing our model's ability to generalize to unseen data. We were able to successfully and efficiently integrate it into our decoding algorithms.

- Decoding Algorithms Comparison: We assessed two decoding algorithms which are Viterbi and max-marginal forward-backward. The Viterbi algorithm yielded the most accurate predictions, while the max-marginal approach provided essential probabilistic insights at each sequence step, proving invaluable in scenarios where understanding the confidence and likelihood of each tag is crucial.

- Adaptation to New Data: We explored smoothing techniques like Absolute Discounting to refine handling of emission probabilities, improving the model’s robustness and performance by distributing probability mass to unseen events.

This project established a robust foundation for sentiment analysis of informal text using HMMs and opened paths for future enhancements in machine learning applications. The methodologies developed and insights gained not only advanced our understanding of machine learning principles but also prepared us for future challenges in data science and AI-related fields.