Machine Learning Fundamentals

HIDEN MARKOV MODEL

Sentiment Analysis

Islem BEGGARI
Idriss BENGUEZZOU
Iness BOUABID
Ghilas MEZIANE

02/04/2023

Table des matières

1 Data description

The project is based on a Named Entity Recognition (NER) dataset that contains sentences, words, their associated Part-of-Speech (POS) tags, and named entity tags. The dataset is used to train a Hidden Markov Model (HMM) to recognize named entities in text and classify them into appropriate categories.

The dataset contains 4 columns :

— **sentence** : sentence number to which each word belongs
— **Word** : the actual word in the sentence
— **POS** : the Part-of-Speech tag associated with the word
— **Tag** : the named entity tag associated with the word

2 Pre-processing

The pre-processing stage involves converting the text data to a format that is suitable for analysis. We decided to convert all words to lower case and remove punctuation, special characters, and numerical values. We also removed stop words, which are commonly occurring words that do not add much meaning to the text.

However, we found that using stemming or lemmatization, which are techniques for reducing words to their base form, resulted in a significant decrease in the log likelihood of the Hidden Markov Model. Therefore, we decided not to use these techniques in our pre-processing pipeline.

Overall, the pre-processing steps aimed to clean the data and make it ready for training the HMM.

3 Implementation of each task

3.1 TASK 1

3.1.1 The transition probability matrix between the words

In this task, we were required to create a transition probability matrix that stores the probability of each word transitioning to the next word in a sentence. The input data was a set of sentences, which was split into individual words. The **word2id** dictionary was used to map each word to a unique ID.

A count matrix was created that stores the frequency of transitions between words. The count matrix was then normalized to obtain the transition probability matrix. The resulting matrix can be used to predict the probability of a given word transitioning to the next word in a sentence.

3.1.2 Computation of the log likelihood given a sentence

The **calculate_log_likelihood** function was implemented to calculate the log likelihood of a given sentence based on a word transition matrix. The function first converts the sentence to a list of word IDs using the mapping dictionary (**word2id**). Then, it iterates through the list of word ids, calculating the log likelihood of each transition based on the word transition matrix. Finally, it returns the total log likelihood of the sentence.

3.2 TASK 2

3.2.1 Implementation of Viterbi Algorithm

We implemented the Viterbi algorithm from scratch to find the best probable path in a Hidden Markov Model (HMM). The algorithm takes four input parameters : initial probabilities (π), transition probabilities (a), emission probabilities (b), and a list of observations. The algorithm returns an array of the indices of the best hidden states.

We initialized the trellis table with zeros and backpointers with integers. Then we calculated the scores for all possible previous states for each observation and chose the highest score. Finally, we backtrack to find the most probable sequence of hidden states.

We tested our implementation using an HMM model trained on the NER dataset. We compared our implementation's output to the output of the HMM model's built-in decode method from the hmmlearn library. The two paths were identical, confirming the correctness of our implementation.

3.2.2 The function Decode Path

We also defined a function **decode_path** that takes two arguments : *path*, which is an array of integer indices representing the sequence of hidden states obtained from running the Viterbi algorithm, and *hidden_states*, which is an array of the actual hidden state labels used in the model.

Our function returns a list of the actual hidden state labels corresponding to the sequence of integer indices in path. This is done by iterating through each index in *path*, and using it as an index to obtain the corresponding hidden state label from *hidden_states*.

This function is useful in converting the integer indices outputted by the Viterbi algorithm into more interpretable labels that correspond to the actual hidden states used in the model.

3.3 TASK 3 : Form our own HMM

The third task required us to develop our own Hidden Markov Model (HMM) by defining the hidden and observable states and generating our own observations.

For our example, we chose to create an HMM that related facial expressions to emotions. The hidden states corresponded to five emotions : *'happy', 'angry', 'sad', 'calm'*, and *'disgusted'*. Since emotions cannot be directly observed, they are considered as hidden states.

Also, we choosed five distinct facial expressions which are : *'smiling', 'frowning', 'crying', 'neutral',* and *'grimacing'*. Such facial expressions can potentially offer indications about a person's emotional state.

To generate our observations, we randomly selected an observable state 100 times using the Python **random.choice()** function. We then mapped the observable states to integers (0-4) for further processing.

This toy example demonstrates how HMMs can be used to model the relationship between hidden and observable states in real-world situations. For instance, HMMs can be applied to infer a person's emotional state based on their facial expressions.

3.4 TASK 4

3.4.1 Baum-welch : Forward-Backwards functions

We are asked to implement the forward and backward probabilities functions as part of the Baum-Welch algorithm for estimating the HMM parameters.

The Baum-Welch algorithm is an iterative procedure used to estimate the parameters of an HMM given a set of observations. The forward and backward probabilities are intermediate calculations required for estimating the parameters of the model.

The forward probabilities function, named **forward__probs**, takes as input the observations, observation vocabulary, number of hidden states, and the estimated transition and emission matrices (*a__* and *b__*). It returns the refined forward probabilities (**alpha__**), which are calculated recursively for each time step t using the previous alpha values and the estimated transition and emission matrices. The **alpha__ matrix** is initialized with a starting probability value of **1/n__hidden__states** for each hidden state for the first time step, and subsequently refined using the estimated transition and emission matrices.

The backward probabilities function, named **backward__probs**, also takes as input the observations, observation vocabulary, number of hidden states, and the estimated transition and emission matrices (*a__* and *b__*). It returns the refined backward probabilities (**beta__**), which are also calculated recursively for each time step t using the next beta values and the estimated transition and emission matrices. The **beta__ matrix** is initialized with a value of 1 for each hidden state for the last time step, and subsequently refined using the estimated transition and emission matrices.

Both *forward* and *backward* probabilities are used to calculate the intermediate variable called gamma (**gamma__**) which is the product of the corresponding forward and backward probabilities, and is required for estimating the transition and emission probabilities. These functions are called iteratively as part of the Baum-Welch algorithm to refine the estimated transition and emission matrices until convergence.

3.4.2 Testing with our formulated HMM

After formulating our Hidden Markov Model (HMM) and training it using the Baum-Welch algorithm, we can now test it with some sample sentences using the Viterbi algorithm. The Viterbi algorithm will predict the most probable path of hidden emotions that generated the observable facial expressions present in each sentence.

We defined a **test__sentence** function that takes a sentence as input, maps the observable facial expressions to their corresponding indices using the **observable__map__face** dictionary, and initializes the Viterbi algorithm with start probabilities for each hidden emotion. The function then runs the Viterbi algorithm to find the most likely path, prints the sentence and the predicted path, and decodes the predicted path to provide the sentiment of the sentence.

We tested the model using several sample sentences with different emotions, such as happy, sad, angry, disgusted, and calm. We generated the observation sequence randomly for each sentence, so the predicted emotions may not always be accurate or meaningful.

Overall, our formulated HMM and the Viterbi algorithm have shown promising results in predicting the most likely sequence of hidden emotions from observable facial expressions in sentences. However, further testing and optimization may be necessary to improve the accuracy and applicability of the model.

4 Further Work

Based on the results and limitations of our work, we identified several point that we could improve :

1. **Incorporating more observable states :** Currently, our HMM only considers five observable states : *'smiling', 'frowning', 'crying', 'neutral',* and *'grimacing'*. Including additional observable states, such as eyebrow movement or lip biting, could potentially improve the accuracy of our model.
2. **Increasing the complexity of the model :** While our HMM includes hidden states to account for different emotional states, it does not consider the context in which these emotions are expressed. Incorporating contextual information, such as the speaker's tone of voice or the situation in which the emotions are being expressed, could lead to a more nuanced understanding of facial expressions.
3. **Expanding the dataset :** Our current dataset is relatively small and only includes a limited range of emotions. Collecting additional data that includes a wider range of emotions and a more diverse set of individuals could help to improve the generalizability of our model.

5 Conclusion

Overall, this project focused on implementing Hidden Markov Models and applying them to various tasks such as Part-of-Speech tagging. We also developed a toy HMM model for facial expressions and emotions.

One of the key challenges we encountered was selecting the appropriate pre-processing techniques. While removing stop words and converting all words to lower case was necessary, using stemming or lemmatization resulted in a significant decrease in the model's performance. This emphasizes the importance of experimentation and testing different approaches to find the most suitable ones.

Another challenge was implementing the Viterbi algorithm and the forward and backward function of the Baum-Welch algorithm. However, this exercise helped us to understand the underlying principles of the algorithms and to appreciate the advantages of using pre-existing libraries such as hmmlearn.

In conclusion, this project provided an opportunity to explore Hidden Markov Models and their applications.