50.007 Machine Learning

PROJECT REPORT

Members:
Victoria Yong | 1004455
Joshua Samjaya | 1004423
Sarah Wong | 1004659

Date of submission: 13 December 2021

Table of Contents

Overview

This project can be found on GitHub at https://github.com/nugglet/ML-Project

In this project, we attempt to build a Hidden Markov Model (HMM) for sentiment analysis with text data taken from Spanish and Russian tweets. The HMM's functions have been written into a wrapper class, **HMM()**. In the following sections, we will highlight the relevant functions used in each part of the assignment and explain our approach and implementation.

Our answer for parts 1-3 can be found in the file *hmm.py*
Our answer for part 4 can be found in the file *hmm_ls.py*
Output files can be found in the /ES and /RU directories for each dataset.

Project Usage and Dependencies

This project utilises python's built-in library and numpy as an external library.

General Notations

K: number of unique classes
T: number of unique tokens
n: length of the dataset

General Functions

get_train_data()
This function takes in the path of the training data. The format of the training data is given as such:

```
Así O
lo O
hizo O
y O
el O
resultado O
fue O
espectacular O
. O
```

Fig.1 Sample from ES/ES/train

The function reads this input, and separates the data into two separate 2D ndarrays, one containing the text input and the other containing the corresponding label. From the data, we generate additional arrays containing unique tokens and labels using the get_ unique() function, which is explained in detail below. These arrays are stored as class attributes:

self.tokens : 2D array where the row and column headers are the tokens in each sentence.

self.labels : 2D array where the row and column headers are the corresponding labels for each word in each sentence.

self.unique_tokens: 1D array of unique input text from the training dataset, alphabetically sorted.

self.unique_labels: 1D array of possible labels, alphabetically sorted. In this case, the categories are ['B-negative', 'B-neutral', 'B-positive', 'I-negative', 'I-neutral', 'I-positive', 'O']

get_test_data()

This function takes in the path of the validation data used to make predictions on. The format of the training data is given as such:

```
La
comida
estuvo
muy
sabrosa
.

Servicio
esmerado
.
```

Fig.2 Sample from ES/ES/dev.in

The function reads this input, and separates the data into a 2D array of tokens, where each 1D array contains one sentence.

get_unique()

This function takes in a 2D array of strings and returns a 1D array of unique strings in the given input array.

```
def get_unique(self, nested_list):
    flattened_list = [item for sublist in nested_list for item in sublist]
    return sorted(list(set(flattened_list)))
```

Fig.3 Function implementation of get_unique()

evaluate()

The *evaluate* function utilizes the evalResult() function from the provided EvalResult.py script. This function takes in 2 arguments, the path of the 'gold' output file, dev.out, and the path of the generated predictions file from our HMM model. The function compares the two files and returns the confusion matrix scores of the model's performance for its entity tagging and sentiment tagging.

Part 1

1.1 Estimating Emission Parameters

The implementation for the generation of the emission parameter table can be found in the function estimate_e().

This function iterates over the training data and counts the number of occurrences of the label (u) and the number of occurrences of each emission pair of label and token (u → o). The function then calculates the ratio of emission: $\frac{count(u->o)}{count(u)}$, and returns a numpy array of size (K, T+1).

Realising that some of the test data will not have appeared in the train data, we added the unknown token #UNK# as the default value for unknown test data. Hence, we calculate our emission parameter for each word as $\frac{count(u->o)}{count(u)+k}$ and the emission parameter for the unknown token as $\frac{k}{count(u)+k}$

The emission table is stored as a class attribute, **self.e_table**.

	La	**comida**	**[...]**	**#UNK#**
B-negative				
[...]				
O				

Fig.4 Visual representation of emission table structure

1.2 Sentiment Prediction with Emission Parameters

Using the emission table, label predictions were generated on the dev.in files and saved to the dev.p1.out file.

The implementation of making predictions using the emission table is written in the *predict_p1()* function, which predicts the label for each token separately based solely on the highest valued emission parameter for that particular token.

1.3 Evaluating Results

The observed F score for both datasets is very low, which is largely attributed to the large number of predictions leading to lower precision (even though the recall is high). In summary, the result is as follows:

ES dataset:
Entity F: 0.2062
Sentiment F:0.1137

RU dataset:
Entity F: 0.2627
Sentiment F: 0.1067

Result for ES dataset:	**Result for RU dataset**

`#Entity in gold data: 255`
`#Entity in prediction: 1733`
 	
`#Correct Entity : 205`
`Entity  precision: 0.1183`
`Entity  recall: 0.8039`
`Entity  F: 0.2062`
 	
`#Correct Sentiment : 113`
`Sentiment  precision: 0.0652`
`Sentiment  recall: 0.4431`
`Sentiment  F: 0.1137`
	`#Entity in gold data: 461`
	`#Entity in prediction: 2089`
	
	`#Correct Entity : 335`
	`Entity  precision: 0.1604`
	`Entity  recall: 0.7267`
	`Entity  F: 0.2627`
	
	`#Correct Sentiment : 136`
	`Sentiment  precision: 0.0651`
	`Sentiment  recall: 0.2950`
	`Sentiment  F: 0.1067`

Fig.5 Classification report generated by evaluate() on the ES and RU dataset

Part 2

2.1 Estimating Transition Parameters

The implementation for the generation of the transition parameter table can be found in the function estimate_q().

This function iterates over the training data and counts 2 things: the number of occurrences of the label (u) and the number of occurrences of each transition pair of labels (u → v). The 'START' and 'STOP' labels are also included. The function then calculates the ratio of transition: $\frac{count(u->v)}{count(u)}$, and stores the values in a numpy array of size (K+1, K+1).

The transition table is stored as a class attribute, **self.q_table**.

	B-negative	[...]	O	STOP
START				
B-negative				
[...]				
O				

Fig.6 Visual representation of the Transition table

2.2 The Viterbi Algorithm

Using the estimated emission and transition parameters, we implemented the viterbi algorithm to compute for the following (for a sentence with n words):

$$y1^*, ..., yn^* = \text{argmax } p(x1, ..., xn, y1, ..., yn)$$

2.2.1 Initialisation

We begin by initialising our length of sentence, number of unique labels and an (n+2) by m array, where n is the length of a sentence and m is the number of unique labels. We also set Pi to be an empty (n+2) by m array.

2.2.2 Forward Algorithm

Then, we iterate through the number of words in each sentence. To check to ensure we do not consider #UNK tokens, for each word n, only if the next word n+1 is in our list of unique tokens, do we proceed to consider it. Then we iterate across the number of unique labels m, to calculate the emission probability.

2.2.3 Termination

To terminate, we iterate across the label and calculate the transition probability from the start label to the current label. If the current probability is greater than the max probability, we update the max probability.

2.2.4 Backward Algorithm

To perform the backward algorithm, we iterate across the m unique labels, and for each m, we calculate the current transition probability and current probability. Checking backwards from the n-1th node, and iterating across, we check if our current probability is greater than the max probability, and update the max probability accordingly.

2.3 Evaluating Results

Results

There is a significant improvement from part 1, where the entity predicted is significantly lesser, hence allowing higher F score. In summary, the result is as follows:

ES dataset:
Entity F: 0.5092
Sentiment F: 0.4304

RU dataset:
Entity F: 0.4353
Sentiment F: 0.3044

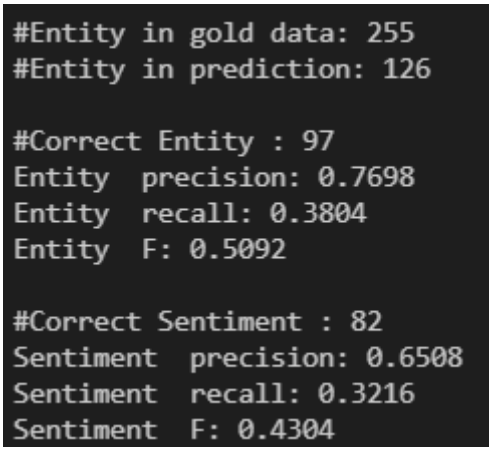
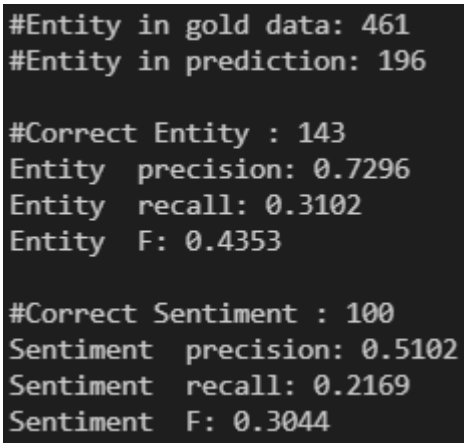
Result for ES dataset:	**Result for RU dataset**
	

Fig.7 Classification report generated by evaluate() on the ES and RU dataset

Part 3

3.1 Viterbi Algorithm: Finding the 5th Best Result

We implemented the 5th best result by tweaking part 2's Viterbi algorithm:

1. Instead of recording one pi value per node, we recorded the best 5 pi value in the forward step. The pi array is now a 3D array, where each node contains five values indicating the top 5 highest pi values sorted in descending order.
2. In the backward step, we select 5 possible parents for each possible parent token. For every possible parent out of the five, we calculate the pi value and select the top five parents for the previous node, storing the possible sequence . At the end of the backward step, we have a 2D array containing 5 label sequences, sorted in descending order. The fifth best sequence is the last sequence in the array.

3.2 Evaluating Results

As we can see, the value of the entity in prediction decreased quite significantly. Both the entity prediction and sentiment prediction suffers, although the sentiment F score seems to suffer the most. In summary, the result is as follows:

ES dataset:
Entity F: 0.2670
Sentiment F:0.0218

RU dataset:
Entity F: 0.1472
Sentiment F: 0.0201

Result for ES dataset:	**Result for RU dataset**
#Entity in gold data: 255 #Entity in prediction: 112 #Correct Entity : 49 Entity precision: 0.4375 Entity recall: 0.1922 Entity F: 0.2670 #Correct Sentiment : 4 Sentiment precision: 0.0357 Sentiment recall: 0.0157 Sentiment F: 0.0218	#Entity in gold data: 461 #Entity in prediction: 137 #Correct Entity : 44 Entity precision: 0.3212 Entity recall: 0.0954 Entity F: 0.1472 #Correct Sentiment : 6 Sentiment precision: 0.0438 Sentiment recall: 0.0130 Sentiment F: 0.0201

Fig.8 Classification report generated by evaluate() on the ES and RU dataset

Part 4

4.1 Laplace Smoothing

We implemented a Laplace Smoothing to deal with transition and emission parameters which have a value of zero. We modified the emission and transmission parameter to include an constant alpha value as a regularisation parameter, thus preventing any of these two parameters to be zero, even if such an instance is never seen in the training data. The calculation becomes as such:

1. Emission parameter: $\frac{count(u->o)+alpha}{count(u)+k+alpha*K}$
2. Transmission parameter: $\frac{count(u->v)+alpha}{count(u)+k+alpha*K}$

However, there is a limited impact on the final F score, most probably due to the small size of the dataset. Even though the recall is higher, the precision is lower, which overall affects the F score negatively. We believe a larger dataset with more entities will see larger improvement from Laplace Smoothing. In summary, the result is as follows:

ES dataset:
Entity F: 0.3207
Sentiment F: 0.2546

RU dataset:
Entity F: 0.4382
Sentiment F: 0.2874

Result for ES dataset:	**Result for RU dataset**

`#Entity in gold data: 255`
`#Entity in prediction: 562`
 	
`#Correct Entity : 131`
`Entity  precision: 0.2331`
`Entity  recall: 0.5137`
`Entity  F: 0.3207`
 	
`#Correct Sentiment : 104`
`Sentiment  precision: 0.1851`
`Sentiment  recall: 0.4078`
`Sentiment  F: 0.2546`
	`#Entity in gold data: 461`
	`#Entity in prediction: 534`
	
	`#Correct Entity : 218`
	`Entity  precision: 0.4082`
	`Entity  recall: 0.4729`
	`Entity  F: 0.4382`
	
	`#Correct Sentiment : 143`
	`Sentiment  precision: 0.2678`
	`Sentiment  recall: 0.3102`
	`Sentiment  F: 0.2874`

Fig.9 Classification report generated by evaluate() on the ES and RU dataset