ML HMM Project

https://github.com/jeroee/ML-Project-50.007
Jeremy Ng 1003565
Suhas Sahu 1003370
Ong Li Chang 1003328

Hidden Markov Model Results [Part 2 - 4]

Entity Recognition

		Datasets		
		EN	SG	CN
Part 2	Precision	0.5116	0.1903	0.0787
	Recall	0.7240	0.5466	0.4771
	F1-Score	0.5996	0.2824	0.1351
Part 3	Precision	0.8325	0.5995	0.3759
	Recall	0.8389	0.4618	0.2271
	F1-Score	0.8357	0.5217	0.2832
Part 4	Precision	0.2251	-	-
	Recall	0.2372	-	-
	F1-Score	0.2310	-	-
Part 5	Precision	0.42	-	-
	Recall	0.41	-	-
	F1-Score	0.41	-	-

Sentiment Recognition

		Datasets		
		EN	SG	CN
Part 2	Precision	0.4534	0.1176	0.0360
	Recall	0.6416	0.3378	0.2186
	F1-Score	0.5313	0.1745	0.0619
Part 3	Precision	0.7995	0.5174	0.2719

	Recall	0.8057	0.3985	0.1643
	F1-Score	0.8026	0.4502	0.2048
Part 4	Precision	0.0369	-	-
	Recall	0.0388	-	-
	F1-Score	0.0378	-	-
Part 5	Precision	0.42	-	-
	Recall	0.41	-	-
	F1-Score	0.41	-	-

Modified Viterbi Algorithm implementation for 3rd best sequence [Part 4]

For the modified Viterbi Algorithm, we made changes to both the forward and backward propagation. The base case stays the same.

First let us talk about the Forward algorithm. For the first layer, we keep the same values as the original viterbi algorithm. For all subsequent layers besides the last layer, for every node of the layer we will store the 3 highest probabilities from transitioning from the previous layer nodes. We will use these probabilities to calculate the subsequent 3 highest probabilities for future nodes. Alongside each probability we have also attached the index for use in backtracking. At the final layer (State -> Stop), out of all our scoring paths calculate, the final node will take the 3rd highest probability of the scoring paths and then the backward propagation algorithm will begin.

For the backward propagation, we will look at the index included in the given probability and will use it to find the previous layer state. At that state we divide our current probability with our chosen transmission and emission parameters and that is how we will find the next probability value to use and index. Using this index we will backtrack again. At every backtrack we will note down the state before moving. We stop at the 2nd layer after the START node as this is where our last index lies, which will give us our last state. This is how we retrieved the third best path.

Design Challenge [Part 5]

For the design challenge we tried to implement a Bidirectional LSTM. 'Bidirectional LSTMs are an extension of traditional LSTMs that can improve model performance on sequence classification problems.

In problems where all timesteps of the input sequence are available, Bidirectional LSTMs train two instead of one LSTMs on the input sequence. The first on the input sequence as-is and the second on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.'[1]
To implement this Bidirectional LSTM, we followed this guide here: https://towardsdatascience.com/pos-tagging-using-rnn-7f08a522f849

Initially we believed that the model was working very well as based on our EN/dev.in we managed to get an accuracy of 90.4%. However due to a lack of time, we were not able to implement the model well and neither were we able to output a file with its attached labels. We believe that this is due to our use of word embeddings, we used word2vec, which resulted in an obscuring of data. Thus, to retrieve the precision, recall and F1 score from the inbuilt scikit learn evaluation method 'classification report'.

Also we used a jupyter notebook instead of making a python file for speed of development. Unfortunately, we ran out of time before we could properly convert it to a python file. Included in our zip is both the python file and jupyter notebook. For these reasons we are not able to include part 5 question 2 results.

---

[1] https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/