CSE628 Assignment 2

Name – Gaurav Ahuja

SBU Id – 111500318

1) **Viterbi Implementation:**

   Say L is the total number of labels and N is the length of sentence. I created two 2D arrays of size LxN named dp and parent.

   - dp[i,j] stores the maximum score if the j^{th} word gets i^{th} tag.
   - Parent[i,j] stores the previous tag for which dp[i,j] is maximized. This parent information is required to find the tag sequence with the highest score.

   Filling the dp array:

   - The columns 2 to N are filled according to eq 5 in the assignment pdf.
   - First column is initialized with start_scores. Emission scores of the first word given each tag are added to the first column.
   - In the last column, the end_scores for each tag are also added to account for the condition that the sentence ends a particular tag.

   After filling the dp array, the parent pointers are traced back to find out the sequence with highest score.
   The time complexity for Viterbi is $O(L^2N)$. The performance of Viterbi can be slightly improved by replacing one for loop with a vectorized addition.
   To fill dp[i,j], we need to consider (dp[k,j-1] + trans_score[k,i]) for all k in 1 to L. This for loop can be replaced by vectorized addition and taking a max of the resulting vector.

   Challenges faced: The dp table should be filled column by column to get the correct result as dp[i,j] depends on dp[k,j] for all k. I was filling the dp table row by row and was getting nearly 50% to 60% accuracy in Viterbi_test. I found out the case where the result of my implementation was incorrect, walked through it step by step and finally figured out the error. I figured out the vectorized version of Viterbi during this debugging phase.

2) **Engineered Features:**

   Basic features: isAlphaNumeric, isNumeric, isDigit, isUppercase, isLowercase, sentStart, sentEnd.

   Features tried:

   - If a word start with capital letter, it is likely to be a noun. Hence I added the feature “STARTS_WITH_CAPS”. The absence of “SENT_BEGIN” and presence of “STARTS_WITH_CAPS” is a strong indicator for noun.
   - As a first step, I tested whether the word ends in a common verb suffix like **ATE** (validate, complicate), **IFY** (quantify, magnify), **ISH** (publish, vanish), **ING** (coming, going).

If the word ended in any of these suffixes, the feature **“ENDS_WITH_[SUFFIX]”** was added.

- Similarly, for adjectives, **LY** (friendly, daily), **ISH** (foolish, childish) were used.

I realized that I was hardcoding a lot of things.

Since the last 3 or 2 or 1 character of a word is an important feature, I decided to add the following 3 features for all words:
“ENDS_WITH_[last 3 character of word]”, “ENDS_WITH_[last 2 character of word]”, “ENDS_WITH_[last character of word]”. This way, words ending in same suffix share a feature.

- **X category specific features:**
  Using the features described above lead to overall increase in token level accuracy. I observed that adjective and X category still had low F1 scores. So, I printed all the words in training set belonging to these 2 categories. The words in adjective category were as per my expectations. The words in X category were surprising. The **‘#[something]’, ‘@[someone]’, URL’S** and **smileys (‘:)’, ‘’;)’)** were marked as X. To incorporate these, I added the following features:
    - “STARTS_WITH_URL” : If the word starts with ‘www’ or ‘http’
    - ‘CONTAINS_[SYMBOL]’ : If the word contains any of the following symbols ['@', '#', '!', ':', '.', '?', ',', ';', '&', '_', "'"]
    - ‘STARTS_WITH_[SYBMOL]’ : If the word starts with the above symbols.

The X specific features lead to improve in F1 score of this category.
Then I focused on ADJ category.

- **Clustering on word embeddings**
  In Prof Geoffery Hinton’s course on coursera, the following plot of words arranged on 2d plane based on word vectors is shown.

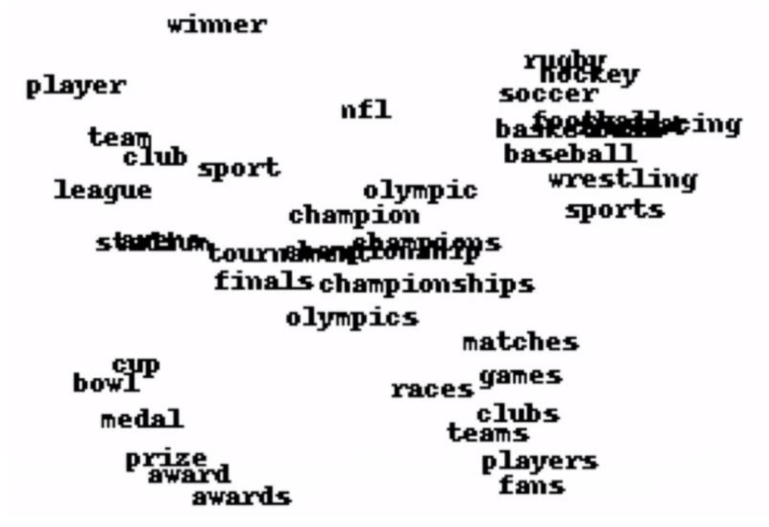

Figure 1 Words related to sports

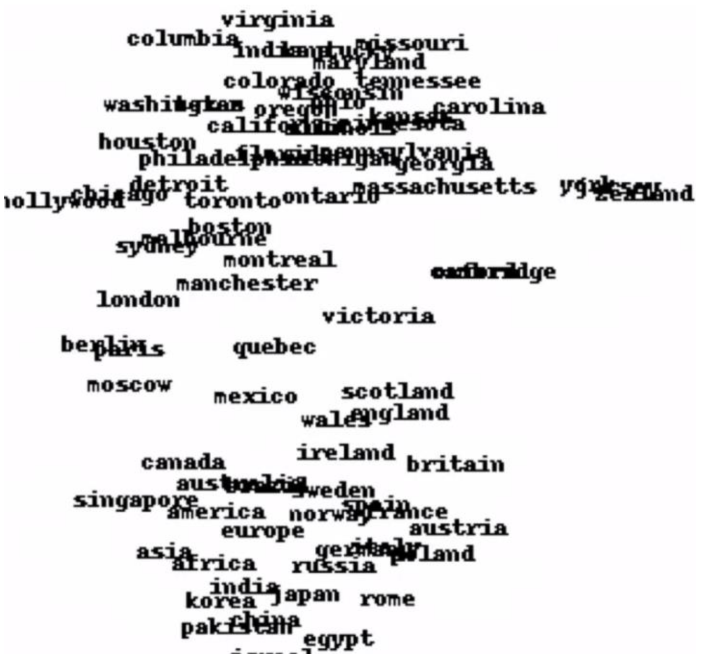

Figure 2 Cities

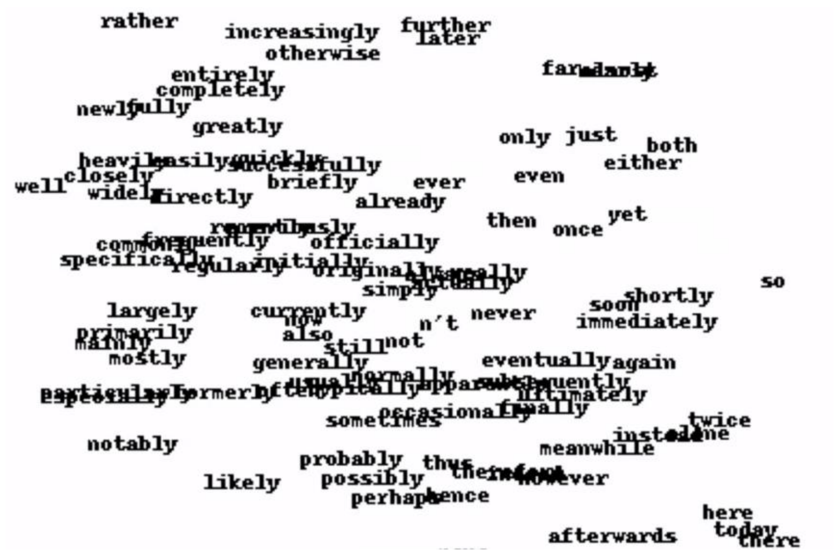

Figure 3 Adjectives

It is clear from these figures that in word vectors, similar words are clustered together. I used the word vectors of 100k words from assignment 1 and used k means clustering to try to group similar words together.
Unfortunately, when I used 100 clusters, the clustering did not finish even in few hours. So I used 10 clusters, but the improvement in token wise accuracy was not observed when cluster number of a word was used as a feature (e.g. 'CLUSTER_[cluster no.]').

To achieve clustering with 100 clusters, I used the MiniBatchKMeans in scikit package. It converges a lot faster, but the clustering results are not very good as it is just an approximate method. When I used 200 clusters with MiniBatchKMeans, 134 clusters contained only 1 element. The words were mainly distributed on 66 clusters.

An observation: Out of nearly 20k words in train sentences, only nearly 5k words were found in the dictionary supplied in assignment 1. It seems that nearly 75% of the words will be unknown and end up in same cluster, thus the clustering may not seem helpful. On further inspection, nearly all the words with tag ADJ are present in dictionary. So the clustering appears to be useful for these kinds of words which are written properly (no slang, no shorthand) on twitter. This should also be true for many verbs.

- **Manual Clustering:**
  Since clustering on word embeddings did not go well, I wanted the words with same POS tag to share a feature. I searched the common suffixes for verbs, adjectives, adverbs, nouns and if the word ended in the common suffix, the feature 'POSSIBLE_[POS TAG]' was added (e.g. friendly - 'POSSIBLE_ADJ'). The common suffix list

was taken from https://dictionary.cambridge.org/grammar/british-grammar/word-formation/suffixes

3) Results

- The token wise accuracy and accuracy on ADJ category are shown with various features in the figure below.

Table 1 Comparison of LR and CRF on features on Dev set

	Logistic Regression			**CRF**		
Features	**Token wise accuracy**	**ADJ**		**Token wise accuracy**	**ADJ**	
		Precision	**Recall**		**Precision**	**Recall**
Basic	84.39%	0.73	0.36	84.29%	0.64	0.55
Suffix	86.42%	0.75	0.46	85.19%	0.69	0.56
Start	86.47%	0.77	0.46	85.85%	0.63	0.53
X specific	87.27%	0.76	0.47	86.42%	0.64	0.54
	F1 on X goes from 0.80 to 0.86			F1 on X goes from 0.81 to 0.84		
Auto Clustering	87.13%	0.73	0.49	86.47%	0.6	0.56
Manual clustering	87.32%	0.75	0.49	85.90%	0.66	0.55
	Auto clustering better for CRF, manual clustering better for LR					
Both Clustering	87.03%	0.72	0.51	86.56%	0.67	0.59

- Effect of adding features on Logistic Regression and CRF on Dev set is shown in Table 2, 3 and 4. For logistic regression, best results are obtained by using features upto 'X specific' and using manual clustering.
  For CRF, best accuracy was obtained by using all the engineered features.

Table 2 Effect of adding features for both LR and CRF

	Logistic regression		**CRF**	
	Basic features	**Engineered features**	**Basic features**	**Engineered features**
Token-wise accuracy	84.38%	87.41%	84.29%	86.80%

Token-wise F1 (macro)	83.33%	87.27%	83.21%	86.04%
Token-wise F1 (micro)	84.38%	87.14%	84.29%	86.80%
Sentence-wise accuracy	8.92%	18.75%	11.60%	15.17%

Table 3 Effect of adding features on each token in LR

Logistic regression										
	Basic features						Engineered features			
	precision	recall	f1-score	support			precision	recall	f1-score	support
.	0.94	0.98	0.96	254		.	0.96	1	0.98	254
ADJ	0.73	0.36	0.49	99		ADJ	0.75	0.49	0.6	99
ADP	0.92	0.88	0.9	151		ADP	0.89	0.92	0.91	151
ADV	0.94	0.59	0.72	129		ADV	0.85	0.72	0.78	129
CONJ	1	0.93	0.96	42		CONJ	0.97	0.93	0.95	42
DET	0.99	0.92	0.95	130		DET	0.98	0.92	0.95	130
NOUN	0.73	0.9	0.8	479		NOUN	0.8	0.88	0.84	479
NUM	0.85	0.68	0.75	34		NUM	0.91	0.91	0.91	34
PRON	0.99	0.92	0.96	194		PRON	0.96	0.95	0.95	194
PRT	0.89	0.88	0.88	57		PRT	0.89	0.89	0.89	57
VERB	0.8	0.85	0.82	362		VERB	0.83	0.88	0.85	362
X	0.81	0.77	0.79	183		X	0.9	0.82	0.86	183

Table 4 Effect of adding features on each token for CRF

CRF										
	Basic features						Engineered features			
	precision	recall	f1-score	support			precision	recall	f1-score	support
.	0.95	0.98	0.97	254		.	0.98	0.99	0.98	254
ADJ	0.64	0.55	0.59	99		ADJ	0.65	0.55	0.59	99
ADP	0.86	0.87	0.87	151		ADP	0.86	0.91	0.88	151
ADV	0.83	0.62	0.71	129		ADV	0.86	0.69	0.76	129
CONJ	0.95	0.93	0.94	42		CONJ	0.91	0.93	0.92	42
DET	0.96	0.91	0.93	130		DET	0.97	0.91	0.94	130
NOUN	0.79	0.86	0.82	479		NOUN	0.8	0.87	0.84	479

NUM	0.85	0.68	0.75	34		NUM	0.88	0.85	0.87	34
PRON	0.99	0.93	0.96	194		PRON	0.95	0.95	0.95	194
PRT	0.84	0.84	0.84	57		PRT	0.88	0.88	0.88	57
VERB	0.79	0.84	0.82	362		VERB	0.84	0.87	0.85	362
X	0.8	0.78	0.79	183		X	0.9	0.83	0.86	183

Example sentence where engineered features help:

"@john lovely seeing u this afternoon #lunch": Here, @,#, -ing, -ly features help in getting better accuracy.

Sentence where CRF is better than LR:

"Really hope I can get to @glasgowfilm for Winter's Bone - need to get on with job applications tonight then! Trailer : http://bit.ly/bhUlum"

In this sentence, the only difference between outputs of LR and CRF is that CRF correctly predicts trailer as Noun, whereas LR predicts it as an Adjective. CRF takes into account the "goodness" of the tag sequence and hence even though the features say that Trailer is Adjective, CRF is able to model that Trailer should be a noun in this context.

Another example: "even better than cocktail friday! http://is.gd/ffDb9"

Here, CRF correctly classifies the whole sentence, while LR makes an error on 'than'.
The ability to model goodness of transitions and that of whose sequence makes CRF a better classifier.

With more training data, I believe CRF will be able to overtake LR in terms of accuracy.