CS4740/5740 Fall 2019 Project 2
Metaphor Detection with Sequence Labeling Models

Saurabh Kumar Yadav **(sky36)**
Sudhanshu Khoriya **(sdk225)**
Soham Ray **(sr2259)**

Kaggle Team : **Team Rocket Returns**

The following document reports the work done in designing and implementation of two models that identifies and labels "Metaphors". The first model is based on implementing **HMM model using Viterbi algorithm**. The second model is decided after experimenting with couple of classifiers. We experimented with **Naive Bayes** and **MaxEnt** classifiers.

1 Sequence Labelling Models

1.1 Implementation Details

1.1.1 HMM (Hidden Markov Model) with Viterbi:

1. **Hidden Markov Model** is a probabilistic sequence model, whose job is to assign labels to each unit in sequence. Our task is to assign labels Metaphor, Literals / Non-Metaphors using HMM model.

2. **Design**

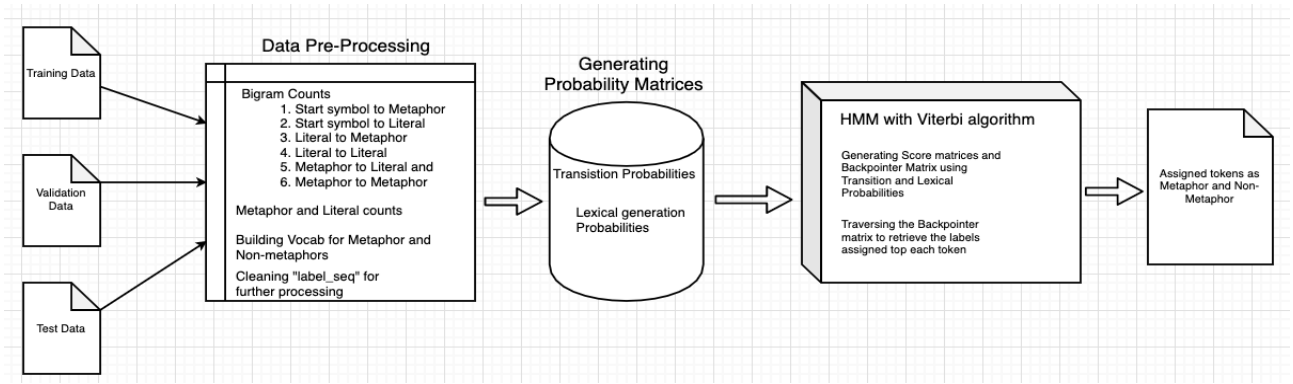

Figure 1: Using HMM with Viterbi in our Model-1

3. **Viterbi Algorithm** provides an efficient way of finding the most likely state sequence in the maximum a posteriori probability sense of a process assumed to be a finite-state discrete-time Markov process.
   We used Viterbi Algorithm to find the best probable sequence by reducing the time complexity for using HMM to C^2

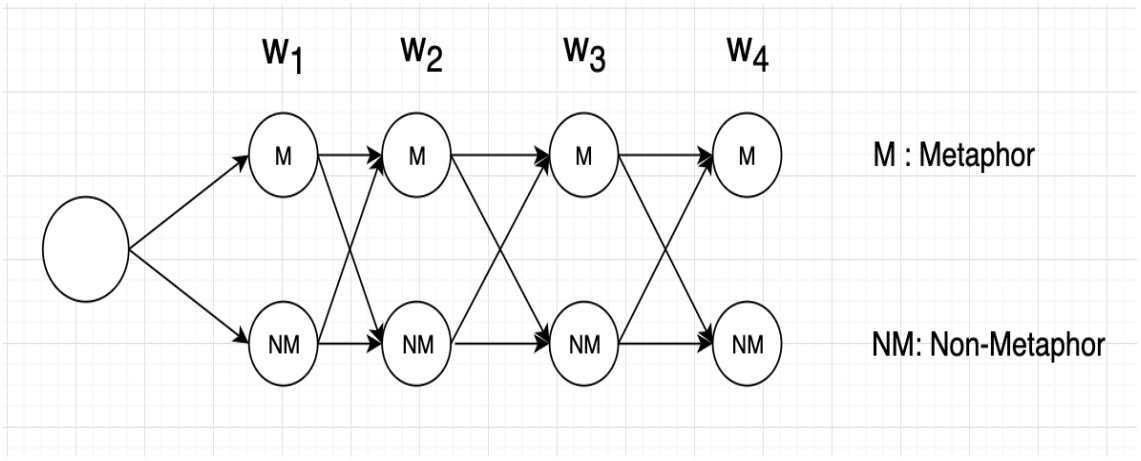

Figure 2: Viterbi

4. **Equation Used**

$$t_n = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) = exp \sum_{i=1}^{n} log(P(t_i|t_{i-1}) + log(P(w_i)|t_i))$$

5. **Feature Generation**

   - Bi-gram counts for:
     - Start symbol followed by Literal
     - Start symbol followed by Metaphor
     - Metaphor followed by Literal
     - Metaphor followed by Metaphor
     - Literal followed by Metaphor
     - Literal followed by Literal
   - Metaphor and Literal counts plus their BOW (Bag of Words).

6. **Code Implementation Details**

   - **Packages used**: Numpy and Pandas
   - **Algorithms developed from scratch**:
     - Viterbi Algorithm
     - Data Pre-processing and
     - Generating Probability Matrices

7. **Viterbi Algorithm : Code Snippet for HMM**

Viterbi Algorithm

In [85]:
```
def viterbi(sent):

    words = sent.split()

    score=np.zeros((2,len(words)))
    backtrack = np.zeros((2,len(words)))

    score[0][0]= np.log(s_m_prob) + np.log(emission_prob.get(words[0],(3.7833502322977045e-06,0))[0])
    backtrack[0][0]=0
    score[1][0]= np.log(s_nm_prob) + np.log(emission_prob.get(words[0],(0,4.799010252125602e-07))[1])
    backtrack[1][0]=0

    for i in range(1,len(words)):
        score_m_m=score[0][i-1] + np.log(m_m_prob)+np.log(emission_prob.get(words[i],(3.7833502322977045e-06,0))[0])
        score_nm_m=score[1][i-1]  +np.log(nm_m_prob) +np.log( emission_prob.get(words[i],(3.7833502322977045e-06,0))[0]

        score_m_nm=score[0][i-1]+np.log(m_nm_prob) +np.log( emission_prob.get(words[i],(0,4.799010252125602e-07))[1])
        score_nm_nm=score[1][i-1] + np.log(nm_nm_prob) + np.log(emission_prob.get(words[i],(0,4.799010252125602e-07))[1

        if score_m_m > score_nm_m:
            backtrack[0][i] = 0
            score[0][i]=score_m_m
        else:
            backtrack[0][i]=1
            score[0][i]=score_nm_m

        if score_m_nm > score_nm_nm:
            backtrack[1][i] = 0
            score[1][i]= score_m_nm
        else:
            backtrack[1][i]=1
            score[1][i]= score_nm_nm

    return score,backtrack
```

Figure 3: Code Snippet: Viterbi

1.1.2 MaxEnt classifier with Viterbi decoding

Maximum Entropy(MaxEnt) The maximum entropy principle (MaxEnt) states that the most appropriate distribution to model a given set of data is the one with highest entropy among all those that satisfy the constrains of our prior knowledge.
The entropy of a probability distribution is calculated as:

$$H(p) = -\sum_i p(i)log(p(i))$$

1. **Design**

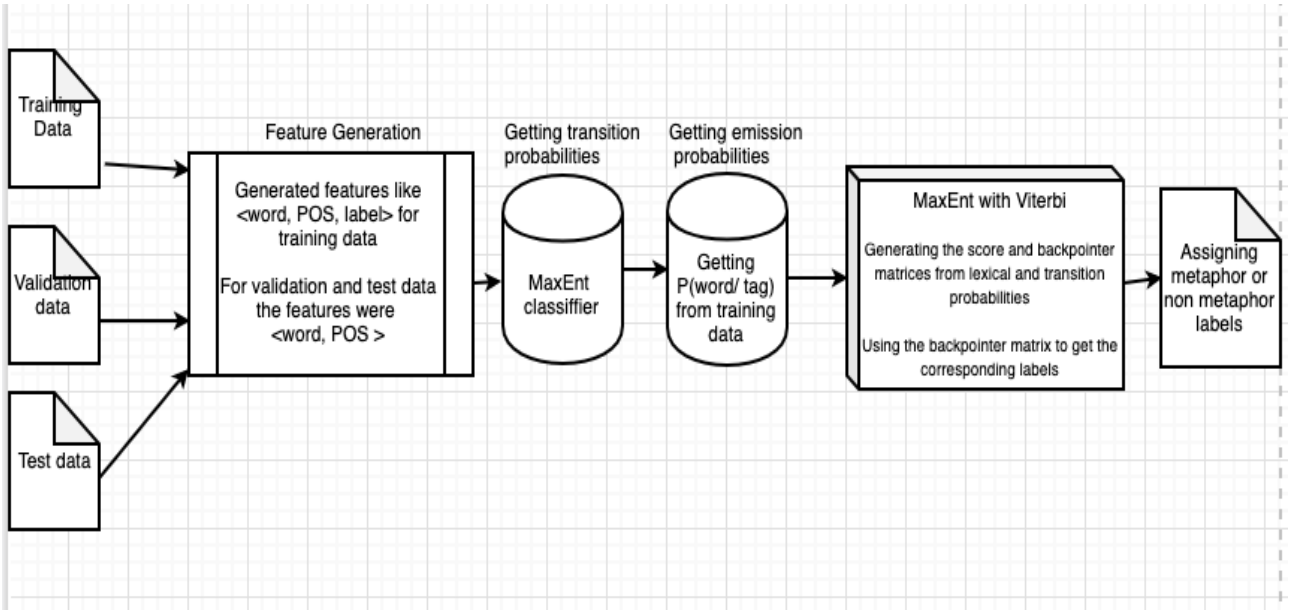

Figure 4: MaxEnt with Viterbi

2. **Feature engineering on Training set**:
   The features that were included for training set were a vector which included the word, it's POS tag and the label(1 for metaphor and 0 for non metaphor). We used a dictionary (python dict) to store a word it's POS tag and created a tuple of this dictionary and it's label. We used these features as our metaphor detection also depends on the word and it's POS tag and we could leverage our MaxEnt model to use all these features to predict the transition probabilities to give us maximum context. The validation and test set were also converted to similar feature vectors.
   We used start symbol followed by literal ,start symbol followed by metaphor and word given the POS tag for finding the emission probabilities for viterbi decoding.

3. **Code implementation details**:

   - **Packages used**: NLTK, sklearn, Numpy and Pandas
   - **Algorithms developed from scratch**:
     - Feature Genration for MaxEnt
     - Data Pre-processing
     - Generating lexical generation probabilities
     - Viterbi decoding

4. **Data structures used**: Python dict, numpy arrays, list and list[list]

5. **MaxEnt : Code Snippet for Viterbi**

   The highlighted part shows the difference in Viterbi for HMM model and MaxEnt model

```
for i in range(1,len(words)):
    score_m_m=score[0][i-1] *prob_matrix[i][1] *emission_prob.get(words[i],(3.7833502322977045e-06,0))[0]
    score_nm_m=score[1][i-1] * prob_matrix[i][1]* emission_prob.get(words[i],(3.7833502322977045e-06,0))[0]

    score_m_nm=score[0][i-1]*prob_matrix[i][0]* emission_prob.get(words[i],(0,4.799010252125602e-07))[1]
    score_nm_nm=score[1][i-1] * prob_matrix[i][0]* emission_prob.get(words[i],(0,4.799010252125602e-07))[1]
```

Figure 5: Code Snippet: Viterbi for MaxEnt

6. **MaxEnt : Code Snippet for Generating features and classification**

Generating feature vectors

```
train_vector = []
for i in range(len(features)):
    train_vector.append((features[i],y_train[i]))

# train_vector
```

```
features_val = []
y_val = []
# sample_weight = []
for _,row in val.iterrows():
    words = row[0].split()
    pos_tags = row[1].split(',')
    for i in range(len(words)):
        y_val.append(int(row[2][i]))
        feat_dict ={}
        feat_dict[words[i]]= pos_tags[i]
        features_val.append(feat_dict)
# features_val
```

Training MaxEnt classifier

```
encoding = maxent.TypedMaxentFeatureEncoding.train(train_vector, count_cutoff=3, alwayson_features=True)
classifier = maxent.MaxentClassifier.train(train_vector, bernoulli=False, encoding=encoding, trace=0)
```

Figure 6: Code Snippet: Feature Generation and MaxEnt classifier

1.2 Pre-processing

Throughout the project, we employ different approaches to preprocessing.

Hidden Markov Models with Viterbi: Here, some text pre-processing is helpful. We convert all text to lower case, however, we do not perform stemming or lemmatization to preserve the use of words as different parts of speech, metaphor or non-metaphor. For example, drown could be used as a non-metaphor in our corpus, whereas drowning could be a metaphor. To preserve this distinction, we perform minimal text pre-processing.

Maximum Entropy Classifier with Viterbi: Here, we perform no textual pre-processing as our features are dependent on the parts of speech tags of the words, and not the words themselves.

Naive Bayes Classifier with Viterbi: For Naive Bayes, again, we rely on the parts of speech sequences, and not the words themselves. Therefore, there is no need for textual preprocessing.

In both approaches, we perform minimal pre-processing to extract the features and their labels and convert them to the required formats in python.

1.3 Experiments

To obtain optimal performance and results, we experimented with multiple approaches to both Hidden Markov Models, as well as Classifier based approaches.

1.3.1 Hyperparameter Tuning for HMM:

To increase our accuracy on both validation and test datasets, we experimented with 2 hyperparameters:

1. **Add-k Smoothing:** We set the value of k for smoothing. When we see a word in val/test dataset, that we didn't see during training, we perform add-k smoothing. Initially, we set the value of k to 0.1. However, since our probability values for metaphor/non-metaphor are very small, 0.1 is a very large bias. To optimize performance, we started using smaller and smaller values of k, and got best results with k=0.00000001

2. **lambda for Transition probability:** In our HMM model, the number of non-metaphors far outnumber the number of metaphors. This drastically effects transition probabilities for non-metaphor to metaphor and metaphor to metaphor. To reduce generalization error, we add a hyperparameter lambda. This gives lesser weightage to transition probabilities and emission probabilities can now make more of a difference. We select lamda as 0.8.

For classifiers, we first try **Naive Bayes** Classifier.

1.3.2 Feature Generation:

For Naive Bayes, we created a **feature matrix**. For every word token in the corpus, we create a feature vector. The feature vector consists of the parts of speech of n-words before it to n-words after it.
For example, in the sentence "The lazy dog is sleeping", for n=1 and word ='dog', we consider feature vector as [pos tag of lazy, pos tag of dog, pos tag of is]. We do this for every word.
The label of every word is its metaphor/non-metaphor label. Here, we experiment with the hyperparameter n and get best results with n=1.
Once the model is trained, we use predict-proba function to get metaphor, non-metaphor probabilities for every word.
We use this **classification probability to replace the transition probability in HMM**.
While Naive Bayes provides us with satisfactory results (F1 score: 54), we switched to MaxEnt Classifier to explore better options.
While implementing **MaxEnt model** we tried **using a feature vector consisting of a pair of two words (word1:pos1, word2:pos2)** but this gave us F1 score of 0.3507 which was less than the F1 score of features containing (word:pos) . So we stuck with the latter as our final model.
Using MaxEnt, we were able to achieve better results and an F1 score of 64. Feature Engineering and experiments are mentioned in detail in the Model-2 section of the document.

Feature Generation from POS tags

```
ngrams=3
# feature_vector = [[0 for j in range(ngrams)] for i in range(len(words))]
feature_vector = np.zeros((len(words),ngrams))

k=0
for index,row in train.iterrows():
    words2 = row[0].split()
    pos_seq  = row[1].split(',')
    pos_new = []
    for i in range(len(pos_seq)):
        pos_new.append(pos_label[pos_seq[i]])
    for j in range(len(words2)):
        if j==0:
            temp=np.zeros(3)
            temp[1:]=np.array(pos_new[:2])
            feature_vector[k] = temp

        elif j==len(words2)-1:
            temp=np.zeros(3)
            temp[:2]=np.array(pos_new[-2:])
            feature_vector[k]  = temp

        else:
            temp=np.array(pos_new[j-1:j+2])
            feature_vector[k] = temp
        k+=1

feature=np.array(feature_vector)
y_train = np.array(y_train)
print(feature)
```

```
[[0. 1. 2.]
 [1. 2. 1.]
 [2. 1. 3.]
 ...
 [5. 1. 7.]
 [1. 7. 5.]
 [7. 5. 0.]]
```

Figure 7: Feature generation for Naive Bayes

1.4 Results

As everyday language has way more non-metaphors than metaphors, it doesn't suffice to rely on accuracy as the sole evaluation metric.

We use F1 score.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

We use the provided *eval.py* file to score our predictions on the validation test for all approaches.

For HMM, we achieved an F1 score of 50.37 with k smoothing set to 0.00000001 and lambda =0.8. Taking a high k value or lower lambda value would reduce the score.

For Naive Bayes, we experimented with the value of 'n', that is the number of words before and after the target word. Intuitively, more words should capture more context, however, we obtained best results with n=1. With n=2, F1 score dropped drastically. Also, there were errors where the model couldn't learn much from very small sentences.

For MaxEnt, we don't assume the corpus to be conditionally independent, unlike Naive Bayes. This boosts are accuracy. We obtained best results by training our model on a tuple of the word and its part of speech tag. Similar to Naive Bayes, results were best when the part of speech tag of only one context word is considered.

From the above results, we concluded that **MaxEnt performed better than HMM and Naive Bayes** for detecting metaphors on validation data.

To improve, we could train our model on a large dataset, so we have a larger vocabulary for metaphorical words. Currently, our model would fail to perform accurately if it encounters metaphors it hasn't seen before.

We observe that Naive Bayes falls short of HMM model in Precision, but more than makes up for it in recall. Also, we can observe that accuracy isn't a reliable metric in this case.

Most importantly, it is evident that MaxEnt performs the best across all metrics.

Model Name	Precision	Recall	F1-score	Accuracy
HMM	51.20	49.57	50.37	88.65
Naïve Bayes	46.26	59.33	51.99	87.27
Maximum Entropy	53.88	70.63	61.12	89.56

Figure 8: Precision, Recall, F score and Accuracy of each model on validation set

1.5 Comparing HMMs and Selected Model-2 (Maximum Entropy)

Maximum Entropy outperformed HMM model.
We pick one of the sentences where both HMM and MaxEnt outputs correct and wrong labels of the word.

For Example:

```
"I wanted to say , you [SEE] , that I know you thought Frannie should n't have [GONE]
, and that it 's ruined your holiday [PLANS] , and , on behalf of us all , I 'm sorry."
```

For the above sentence the words of interest are "see", "gone" , and "plans".
We have the following table :

Words	Label assigned by HMM	Label assigned by MaxEnt	Correct Label
see	Non-Metaphor	Metaphor	Metaphor
gone	Non-Metaphor	Metaphor	Non-Metaphor
plans	Metaphor	Metaphor	Metaphor

Figure 9: Label assigned by MaxEnt and HMM with Correct label

1.5.1 Error Analysis

From the above example from validation set, we have selected only the metaphor labels as predicted by our HMM and MaxEnt models. All the other labels have been correctly identified by both the models.
The MaxEnt model performs better on detecting new metaphors and our HMM model is more biased towards the metaphors occurring in the training set. This can be attributed to the fact that MaxEnt is considering the POS tags while generating the transition probabilities and HMM model does not consider them and gives more weight to non metaphors. Take the example of 'see' from Fig.8

Consider the word 'gone' which was incorrectly classified as metaphor by MaxEnt. This is due to the fact that the probability of 'gone' given it's POS tag is high as we considered POS features of the word and did not consider higher n-grams for MaxEnt model whereas HMM gives correct prediction as it is more biased towards non metaphors as mentioned above.

For the word 'plans' , both our models predicted it correctly as metaphors as 'plans' has occurred as metaphor with given POS tag in our training set

Error pattern Analysis: From these observations, we can conclude that our MaxEnt model performed better at detecting most metaphors but it fails in case when the words and POS tag sequence are are similar to other metaphors in training set and give high weights to metaphors . This can be improved by using

higher n-grams features consisting of the (word1:pos1, word2:pos2, word3:pos3) to get better context and improving the accuracy of the model.

Also, the emission probabilities are same for HMM and MaxEnt model but the transition probabilities of MaxEnt are not biased for non metaphors(even if they dominate in numbers) as opposed to HMM.

1.6 Competition Score

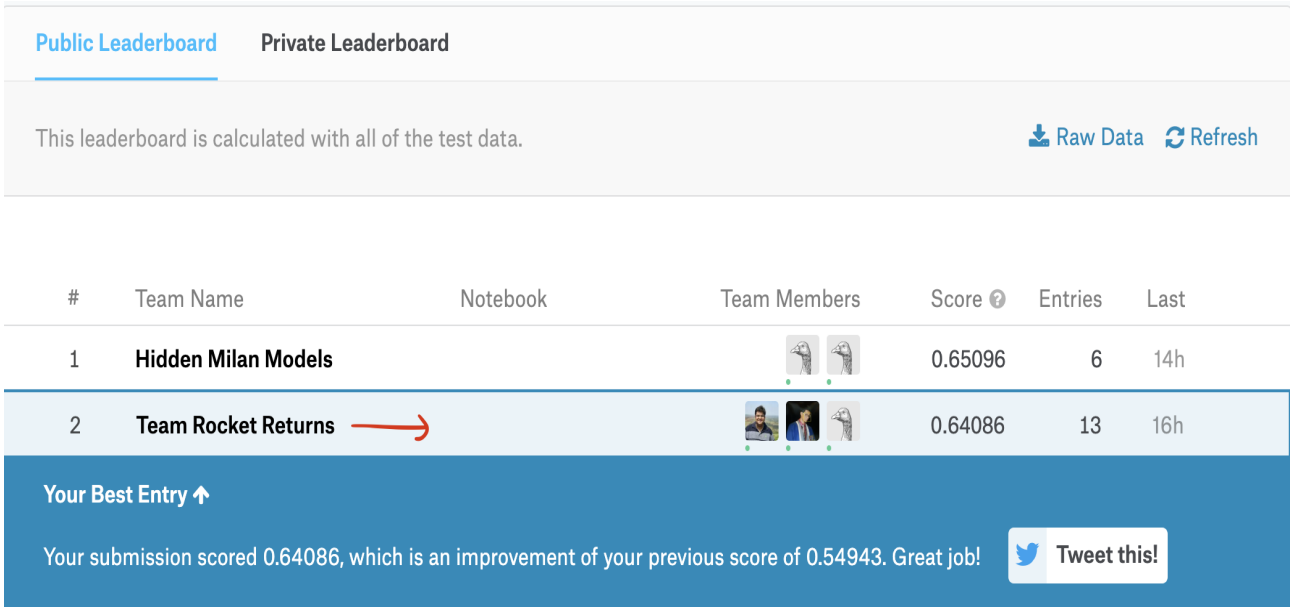

Figure 10: **Kaggle Rankings**: Best Score was achieved by **MaxEnt Model**

2 Code Pieces for Implementations

Code pieces/snippets are included in the "Implementation Section" itself. Please refer the codes for each model (HMM, Naive Bayes, MaxEnt) to see specific implementations.

3 Individual Member Contribution

Soham's Contribution: Feature Generation for Naive Bayes, Model implementation for MaxEnt, Viterbi Algorithm
Saurabh's Contribution: Model implementation for Naive Bayes, Feature generation for Hidden Markov Models, Viterbi Algorithm
Sudhanshu's Contribution: Feature Generation for MaxEnt model, Hyperparameter tuning and error analysis, Viterbi Algorithm.

4 Feedback

This project for detecting metaphors in a sentence clarified how we can tackle features, while processing, that are not conditional independent of each other. Also, we learned how we can bring the computational time complexity for achieving such optimum probable sequence of tags, down. This project deepened our knowledge of classifiers and how we can use them to label Literal/Metaphor tags in a sentence.