50.007 Machine Learning

Design Project

Report

Approaches & Results

https://github.com/Sean2309/50.007_ML_SentimentAnalysis_Project.git

Sean Phay Wei Xiang	1005969
Gizelle Lim Yin Xuan	1006141
Mubaraquali Muhammed Sufyanali	1006394

Introduction

In this assignment, our objective is to create a sequence labelling model for casual text using the principles of the Hidden Markov Model (HMM) that we've studied. The project aims to train this model using two provided training datasets and then utilise it to predict label sequences for new sentences.

There are a total of seven labels provided, which correspond to the hidden states within the Hidden Markov Model. These labels determine the observable states, which are the sequences of words present in the text.

These are the 7 unique labels:

O: Text outside of any recognized entity.
B-positive: Beginning of entities linked to positive sentiment.
B-negative: Beginning of entities related to negative sentiment.
B-neutral: Beginning of entities tied to neutral sentiment.
I-positive: Continuation of entities associated with positive sentiment.
I-negative: Continuation of entities associated with negative sentiment.
I-neutral: Continuation of entities connected to neutral sentiment.

Part 1

1.1

The function **estimate_emissions_params** estimates the emission parameters from the training dataset.

Inputs:

- *file*: List containing all of the lines present in the train dataset
    - Sample format: ['Estuvimos O\n', 'hace O\n', 'poco O\n', 'mi O\n', 'pareja O\n', 'y O\n', 'yo O\n', 'comiendo O\n', 'y O\n']

- *k*: Integer representing the number of new words not present in the training dataset

Output:

- *emissions_params*: Dictionary containing key-value pairs of each word mapped to each label within the training dataset
    - Sample format: 'Estuvimos': {'O': 0.00020664003306240529, 'B-positive': 0.0, 'B-neutral': 0.0, 'B-negative': 0.0, 'I-positive': 0.0, 'I-neutral': 0.0, 'I-negative': 0.0}, 'hace': {'O': 0.0008954401432704229, 'B-positive': 0.0…}

1.2

The function **estimate_emissions_params** estimates the emission parameters from the training dataset.
Inputs:

- *file*: List containing all of the lines present in the train dataset
    - Sample format: same as 1.1

- *k*: Integer representing the number of new words not present in the training dataset

Output:

- *emissions_params*: Dictionary containing key-value pairs of each word mapped to each label within the training dataset
    - Sample format: same as 1.1

1.3

The function **sentiment_analysis** predicts the labels using the emission parameters from the training dataset.

Inputs:

- *file*: List containing all of the lines present in the dev.in dataset
    - Sample format: ['Plato\n', 'degustación\n', ':\n', 'un\n', 'poco\n', 'abundante\n']

- *emissions_params*: Dictionary containing key-value pairs of each word mapped to each label
    - same as 1.1

Outputs:

- *predicted_output*: List containing each line of the output, each line containing the word and the predicted label
    - Sample format: ['Plato B-negative', 'degustación I-positive', ': O', 'un O', 'poco O', 'abundante O']

Results

Part 1 Result:	
ES: #Entity in gold data: 229 #Entity in prediction: 1164 #Correct Entity : 170 Entity precision: 0.1460 Entity recall: 0.7424 Entity F: 0.2441 #Correct Sentiment : 96 Sentiment precision: 0.0825 Sentiment recall: 0.4192 Sentiment F: 0.1378	**RU:** #Entity in gold data: 389 #Entity in prediction: 1378 #Correct Entity : 247 Entity precision: 0.1792 Entity recall: 0.6350 Entity F: 0.2796 #Correct Sentiment : 127 Sentiment precision: 0.0922 Sentiment recall: 0.3265 Sentiment F: 0.1437

Table 1: Precision, Recall and F scores for Part 1

Part 2

2.1

The function **estimate_transition_parameters** gives the transition probabilities by using the transition probability formula.

Inputs:

- *file*: List containing all of the lines present in the train dataset
    - Sample format: same as 1.1

Outputs:

- *transition_parameters*: Dictionary containing key-value pairs that maps each step's probability taken within the training dataset
    - Sample format: {('START', 'O'): 0.46445880452342486, ('O', 'O'): 0.9456845511712573, ('O', 'B-positive'): 0.038980620012503214, ('B-positive', 'O'): 0.8791304347826087, ('O', 'END'): 0.06773802081418012}

2.2

The function **viterbi_algorithm** gives the optimal labels in a specific order

Inputs:

- *test_data*: List containing all of the lines present in the dev.in dataset
    - Sample format: same as 1.3

- *emission_params*: Dictionary containing key-value pairs of each word mapped to each label within the training dataset
    - Sample format: same as 1.1

- *transition_params*: Dictionary containing key-value pairs that maps each step's probability taken within the training dataset
    - Sample format: same as 2.1

Outputs:

- *optimal_labels*: List containing sets of tuples of (word, predicted label) respectively
    - Sample format: [('Plato', 'O'), ('degustación', 'O'), (':', 'O'), ('un', 'O'), ('poco', 'O')]

Results

Part 2 Result:	
ES: #Entity in gold data: 229 #Entity in prediction: 117 #Correct Entity : 98 Entity precision: 0.8376 Entity recall: 0.4279 Entity F: 0.5665 #Correct Sentiment : 76 Sentiment precision: 0.6496 Sentiment recall: 0.3319 Sentiment F: 0.4393	**RU:** #Entity in gold data: 389 #Entity in prediction: 144 #Correct Entity : 110 Entity precision: 0.7639 Entity recall: 0.2828 Entity F: 0.4128 #Correct Sentiment : 82 Sentiment precision: 0.5694 Sentiment recall: 0.2108 Sentiment F: 0.3077

Table 2: Precision, Recall and F scores for Part 2

Part 3

The function **compute_kth_best_sequence** helps to compute the

Inputs:

- *words*: List containing all of the lines present in the dev.in dataset
    - Sample format: same as 1.3

- *emission_params*: Dictionary containing key-value pairs of each word mapped to each label within the training dataset
    - Sample format: same as 1.1

- *transition_params*: Dictionary containing key-value pairs that maps each step's probability taken within the training dataset
    - Sample format: same as 2.1

- *k*: Integer representing the number of new words not present in the training dataset

Outputs:

- *All_kth_best_tags:* A list of all the best tags of kth iteration

Results

Part 3 Result:	
ES, k = 2: #Entity in gold data: 229 #Entity in prediction: 224 #Correct Entity : 97 Entity precision: 0.4330 Entity recall: 0.4236 Entity F: 0.4283 #Correct Sentiment : 31 Sentiment precision: 0.1384 Sentiment recall: 0.1354 Sentiment F: 0.1369	**RU, k = 2:** #Entity in gold data: 389 #Entity in prediction: 243 #Correct Entity : 99 Entity precision: 0.4074 Entity recall: 0.2545 Entity F: 0.3133 #Correct Sentiment : 50 Sentiment precision: 0.2058 Sentiment recall: 0.1285 Sentiment F: 0.1582
ES, k = 8: #Entity in gold data: 229 #Entity in prediction: 177 #Correct Entity : 74 Entity precision: 0.4181 Entity recall: 0.3231 Entity F: 0.3645 #Correct Sentiment : 12 Sentiment precision: 0.0678 Sentiment recall: 0.0524 Sentiment F: 0.0591	**RU, k = 8:** #Entity in gold data: 389 #Entity in prediction: 215 #Correct Entity : 93 Entity precision: 0.4326 Entity recall: 0.2391 Entity F: 0.3079 #Correct Sentiment : 37 Sentiment precision: 0.1721 Sentiment recall: 0.0951 Sentiment F: 0.1225

Table 3: Precision, Recall and F scores for Part 3

Part 4:

For Part 4, we did extensive research on the possible algorithms to be used as alternatives to the Viterbi algorithm. We first tried predicting using Neural Networks. We created a class that will run for 20 Epochs under a learning rate of 0.1.

```python
# Define the neural network architecture
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.weights_input_hidden = np.random.randn(input_size, hidden_size)
        self.bias_hidden = np.zeros(hidden_size)
        self.weights_hidden_output = np.random.randn(hidden_size, output_size)
        self.bias_output = np.zeros(output_size)

    def forward(self, inputs):
        self.hidden_activation = np.dot(inputs, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = self.sigmoid(self.hidden_activation)
        self.output_activation = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.predicted_probs = self.softmax(self.output_activation)
        return self.predicted_probs

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def softmax(self, x):
```

Fig 1: Neural Network Algorithm Implementation

However, this algorithm was unable to give us satisfactory results and we shifted our focus to another algorithm: Beam Search Algorithm.

```python
def beam_search_algorithm(input_data, emission_probs, transition_probs, beam_width):
    flat_tagged_output = []
    current_words = []

    for token in input_data + ['\n']:
        if not token.strip():
            if current_words:
                best_path = beam_search_for_sentence(current_words, emission_probs, transition_probs, beam_width)
                flat_tagged_output.extend(best_path)
                current_words = []
        else:
            current_words.append(token)

    return flat_tagged_output
```

Fig 2: Beam Search Algorithm Implementation

We chose to implement the Beam Search algorithm due to its efficiency and potential to improve accuracy.

The Beam Search algorithm aims to predict the sentiment label sequence for tweets. It functions by considering multiple paths simultaneously, focusing on the most promising ones. This technique contrasts with the Viterbi algorithm's single-best-path approach.

We have 2 functions involved:

1. **beam_search_algorithm** function
    a. It processes the input data (text) by splitting it into sentences and passing each sentence to the beam_search_for_sentence function.

Inputs:

- *input_data*: List containing all of the lines present in the dev.in dataset
    - Sample format: same as 1.3

- *emission_probs*: Dictionary containing key-value pairs of each word mapped to each label within the training dataset
    - Sample format: same as 1.1

- *transition_probs*: Dictionary containing key-value pairs that maps each step's probability taken within the training dataset
    - Sample format: same as 2.1

- *beam_width*: Integer that states how many candidate paths or sequences are kept at each step of the search

Output:

- *flat_tagged_output:* List of labelled words
    - Sample format: [('Интерьер\n', 'O'), (',\n', 'O'), ('интерьер\n', 'O'), (',\n', 'O')]

2. **beam_search_for_sentence** function:
    a. This function performs Beam Search on a single sentence. It iterates through each word in the sentence and maintains active beams, considering various possible label sequences. It calculates the probability of each label sequence based on emission and transition probabilities, and it selects the top beam_width sequences at each step.

Input:

- Sentence: List containing the words present in a single sentence of the dataset
    - Sample format: ['restaurante\n', 'excelente\n', 'con\n', 'carne\n', 'de\n', 'alta\n', 'calidad\n', '.\n']['restaurante\n', 'excelente\n', 'con\n', 'carne\n', 'de\n', 'alta\n', 'calidad\n', '.\n']

- *mission_probs*: Dictionary containing key-value pairs of each word mapped to each label within the training dataset
    - Sample format: same as 1.1

- *transition_probs*: Dictionary containing key-value pairs that maps each step's probability taken within the training dataset

    - Sample format: same as 2.1

- *beam_width*: Integer that states how many candidate paths or sequences are kept at each step of the search

Output:

- *best_path*: List containing tuples of the word and the respective predicted labels for the optimal path
    - Sample format: [('restaurante\n', 'O'), ('excelente\n', 'O'), ('con\n', 'O'), ('carne\n', 'O'), ('de\n', 'O'), ('alta\n', 'O'), ('calidad\n', 'O'), ('.\n', 'O')]

After processing all words, it performs a termination step to calculate the probability of each label sequence reaching the "END" label. The sequence with the highest overall probability is chosen as the best path.

Overall, the code uses a forward-pass approach to explore multiple potential label sequences simultaneously, maintaining a limited number of active beams at each step to ensure efficiency. The algorithm considers both emission and transition probabilities to make informed decisions about the most likely label sequence for the input text.

This Beam Search implementation is designed to handle sequence labelling tasks, such as sentiment analysis, where each word is assigned a label from a predefined set of labels. The beam width parameter controls the number of label sequences the algorithm explores at each step, allowing a balance between computational efficiency and accuracy.

Conclusion

In conclusion, this project has been a comprehensive exploration of sentiment analysis for informal texts, employing the principles of Hidden Markov Models and advanced techniques such as the Viterbi algorithm and Beam Search. We designed, implemented, and refined our own sequence labelling models to accurately predict sentiment labels for words in various languages. Through this project, we gained a deeper understanding of probabilistic models, sequence labelling, and the challenges inherent in sentiment analysis. By evaluating our models on both training and development datasets, we demonstrated their effectiveness and fine-tuned their performance. This project not only showcased our technical skills but also highlighted the importance of adapting and innovating methodologies to address real-world language processing challenges.