CS563 Natural Language Processing
Assignment: 2

Mukuntha N S (1601CS27)
Shikhar Jaiswal (1601CS44)

March 18, 2020

1 Assignment Description

Designing and implementing a **Decision Tree** based model to identify the masked name from the sentence and applying them for **Coreference Resolution** on the **WikiCREM Dataset**.

2 Procedure

Installation

Install the following dependencies either using pip or through conda in a Python 3.5+ environment:
• jupyter
• sklearn

```
python3 -m pip install jupyter sklearn
```

or alternatively,

```
conda install -c anaconda jupyter sklearn
```

Running The Notebook

To run the program, head to the directory *Assignment*2/*Q*1. Please note that the **dataset should be extracted in the same folder as above**. Use the following command to run the notebook:

```
jupyter notebook
```

3 Discussion

The primary objective of the assignment is to implement a Decision Tree based model applying it for Coreference Resolution. The following sub-sections contain the explanation for individual code snippets. For a detailed look at the output, please refer the notebook and the individual files provided.

Notebook Code - Coreference Resolution System

Pre-processing

Import the Python dependencies, and check the data samples and their values and pre-process the corpus.

```
# Import the libraries for Decision Tree Classifier.
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_recall_fscore_support, accuracy_score
```

```
# Parse the dataset for training examples.
train_examples = []

with open('WikiCREM/WikiCREM_train.txt') as fp:
    i = 0
    example = {}

    for line in fp:
        i += 1

        if i == 1:
            example['tokens'] = line.split()
        elif i == 2:
            for j, token in enumerate(example['tokens']):
                if token == '[MASK]':
                    example['mask_idx'] = j
        elif i == 3:
            example['candidates'] = line.rstrip().split(',')
        elif i == 4:
            example['true_label'] = line.rstrip()
        else:
            if 'mask_idx' in example.keys():
                train_examples.append(example)

            example = {}
            i = 0
            continue
```

```
# Parse the dataset for testing examples.
test_examples = []
```

```
with open('WikiCREM/WikiCREM_dev.txt') as fp:
    i = 0
    example = {}

    for line in fp:
        i += 1

        if i == 1:
            example['tokens'] = line.split()
        elif i == 2:
            for j, token in enumerate(example['tokens']):
                if token == '[MASK]':
                    example['mask_idx'] = j
        elif i == 3:
            example['candidates'] = line.rstrip().split(',')
        elif i == 4:
            example['true_label'] = line.rstrip()
        else:
            if 'mask_idx' in example.keys():
                test_examples.append(example)

            example = {}
            i = 0
            continue
```

Feature Generation Code

We generate the following sets of features:
- Minimum Distance to Matching Noun Phrase (Integer: 0, 1, ...)
- Minimum Distance to Non-Matching Noun Phrase (Integer: 0, 1, ...)
- Minimum Sentence Distance to Matching Noun Phrase (Integer: 0, 1, ...)
- Minimum Sentence Distance to Non-Matching Noun Phrase (Integer: 0, 1, ...)
- Number of Matching Repetitions Within The Passage (Integer: 0, 1, ...)
- Number of Non-Matching Repetitions Within The Passage (Integer: 0, 1, ...)
- Presence of Matching Anaphor (Integer: 0, 1)
- Presence of Non-Matching Anaphor (Integer: 0, 1)
- Presence of Matching Cataphor (Integer: 0, 1)
- Presence of Matching Cataphor (Integer: 0, 1)

```
# Function for finding the minimum distance between two matching NPs.
def generate_matching_distance_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        distance = 9999
```

```
        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                distance = min(distance, abs(i - example['mask_idx']))

        if candidate == example['true_label']:
            p = distance
        else:
            n = distance

    return p, n
```

```
# Function for finding the minimum distance between two non-matching NPs.
def generate_non_matching_distance_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        distance = 9999

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                distance = min(distance, abs(i - example['mask_idx']))

        if candidate == example['true_label']:
            n = distance
        else:
            p = distance

    return p, n
```

```
# Function for finding the minimum sentence distance between two matching NPs.
def generate_matching_sentence_distance_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        distance = 9999
        idx = -1
        sentence_distance = 0

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                if abs(i - example['mask_idx']) < distance:
                    distance = abs(i - example['mask_idx'])
                    idx = i

        for i in range(min(idx, example['mask_idx']), max(idx, example['mask_idx'])):
            if example['tokens'][i] == '.':
```

```
                sentence_distance += 1

        if candidate == example['true_label']:
            p = sentence_distance
        else:
            n = sentence_distance

    return p, n
```

```
# Function for finding the minimum sentence distance between two non-matching NPs.
def generate_non_matching_sentence_distance_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        distance = 9999
        idx = -1
        sentence_distance = 0

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                if abs(i - example['mask_idx']) < distance:
                    distance = abs(i - example['mask_idx'])
                    idx = i

        for i in range(min(idx, example['mask_idx']), max(idx, example['mask_idx'])):
            if example['tokens'][i] == '.':
                sentence_distance += 1

        if candidate == example['true_label']:
            n = sentence_distance
        else:
            p = sentence_distance

    return p, n
```

```
# Function for finding the repetition count of matching NPs.
def generate_matching_repetition_count_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        count = 0

        for c in candidate.split():
            count = max(example['tokens'].count(c), count)

        if candidate == example['true_label']:
```

```
            p = count
        else:
            n = count

    return p, n
```

```
# Function for finding the repetition count of non-matching NPs.
def generate_non_matching_repetition_count_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        count = 0

        for c in candidate.split():
            count = max(example['tokens'].count(c), count)

        if candidate == example['true_label']:
            n = count
        else:
            p = count

    return p, n
```

```
# Function for finding the existence of matching anaphoric NPs.
def generate_matching_anaphor_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        flag = 0

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                if i - example['mask_idx'] < 0:
                    flag = 1

        if candidate == example['true_label']:
            p = flag
        else:
            n = flag

    return p, n
```

```
# Function for finding the existence of non-matching anaphoric NPs.
def generate_non_matching_anaphor_feature(example):
    p = 0
```

```
    n = 0

    for candidate in example['candidates']:
        flag = 0

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                if i - example['mask_idx'] < 0:
                    flag = 1

        if candidate == example['true_label']:
            n = flag
        else:
            p = flag

    return p, n
```

```
# Function for finding the existence of matching cataphoric NPs.
def generate_matching_cataphor_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        flag = 0

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                if i - example['mask_idx'] > 0:
                    flag = 1

        if candidate == example['true_label']:
            p = flag
        else:
            n = flag

    return p, n
```

```
# Function for finding the existence of non-matching cataphoric NPs.
def generate_non_matching_cataphor_feature(example):
    p = 0
    n = 0

    for candidate in example['candidates']:
        flag = 0

        for i, token in enumerate(example['tokens']):
            if token in candidate.split():
                if i - example['mask_idx'] > 0:
```

```
                    flag = 1

        if candidate == example['true_label']:
            n = flag
        else:
            p = flag

    return p, n
```

```
# Function for generating the test and training sets.
def feature_set_generation(examples):
    X = []
    Y = []

    for example in examples:
        p1, n1 = generate_matching_distance_feature(example)
        p2, n2 = generate_non_matching_distance_feature(example)
        p3, n3 = generate_matching_sentence_distance_feature(example)
        p4, n4 = generate_non_matching_sentence_distance_feature(example)
        p5, n5 = generate_matching_repetition_count_feature(example)
        p6, n6 = generate_non_matching_repetition_count_feature(example)
        p7, n7 = generate_matching_anaphor_feature(example)
        p8, n8 = generate_non_matching_anaphor_feature(example)
        p9, n9 = generate_matching_cataphor_feature(example)
        p10, n10 = generate_non_matching_cataphor_feature(example)
        X.append([p1, p2, p3, p4, p5, p6, p7, p8, p9, p10])
        Y.append(1)
        X.append([n1, n2, n3, n4, n5, n6, n7, n8, n9, n10])
        Y.append(0)

    return X, Y
```

Testing and Analysis

We implement a standard Entropy Gain-based Decision Tree. We split the overall training dataset into 10 fold cross validation sets. We obtain the accuracy on the individual folds, and report the average value of accuracy for cross validation. Finally, we obtain the prediction results on the test dataset, and report the accuracy, precision, recall and f-score metrics.

```
X_Train, Y_Train = feature_set_generation(train_examples)
X_Test, Y_Test = feature_set_generation(test_examples)

scores = cross_val_score(DecisionTreeClassifier(criterion = 'entropy', random_state = 42),
                         X_Train, Y_Train, cv = 10)

sum_scores = 0
for score in scores:
    sum_scores += score
```

```
sum_scores /= len(scores)

print('10-Fold Cross Validation Score: ', sum_scores)
```

Cross Validation on Training Set

```
10-Fold Cross Validation Score:  0.6800051005658361
```

```
clf = DecisionTreeClassifier(criterion = 'entropy', random_state = 42).fit(X_Train, Y_Train)
Y_Pred = clf.predict(X_Test)
prec, rec, fscore, _ = precision_recall_fscore_support(Y_Test, Y_Pred, average = 'macro')
print('Accuracy: ', accuracy_score(Y_Test, Y_Pred), ' Precision: ', prec, ' Recall: ',
      rec, ' F-Score: ', fscore)
```

Testing Set Scores

```
Accuracy:  0.6836  Precision:  0.68367643  Recall:  0.6836  F-Score:  0.68356707
```

Additionally for the notebook codes, please refer *Q1/Q1 - Decision Tree Coreference Resolution.ipynb*.