Student Name: Niketan Eshwar Moon
Student Id: a1790186
Course: Applied Natural Language Processing (Semester 1/ 2022)

ANLP Assignment 3
Building an aspect-based sentiment analysis algorithm based on Syntactic Parsing

Importing Libraries

Imported libraries such as numpy, pandas matplotlib to do the normal calculations and reading dataset and converting to a dataframe. Imported ElementTree from XML to parse the XML data provided. Used spacy for doing syntactic parsing using dependency parsing. Used other libraries such as nltk to apply tokenization in the text preprocessing phase. Imported CSV to write to the CSV file.

Importing the Dataset

Imported the dataset “Restaurants.xml” and pass it to ElementTree to get the root of the XML file in this case <sentences>.

Transforming XML Data to CSV file and loading the dataset

Created a function called transformXMLtoCSV() to create a CSV file from XML data. Used CSV writer to first write only the headings to the CSV file “restaurants.csv”. Later went through the XML file and grabbed the sentence id which we get as an attribute of the <sentence> tag. Similar way we grabbed **term** and **polarity** which are attributes of tag <aspectTerm>. While parsing through the tree it was observed that for some tags <sentence> there was no aspectTerms present, in such cases I did not write it to the CSV. The function needs to be called to convert the XML to the CSV file with a root as an argument to parse the XML file. Looping through the root will give all its children and then again looping through each child gives its children. This is how parsing is achieved in the code. Once the CSV file is generated, I read it through pandas.read_csv() method.

Data Exploration

Used head and info to explore data further. The dataset has 2021 entries and 4 columns. The description of the columns is as follows.
id - sentence id
term - Aspect term or the topic on which sentiment analysis will be carried on
polarity - Tells us the sentiment of the sentence.

Visualizing the polarities

Plotted value_counts for polarity to get different values for polarity predictor. To closely look at the types of polarities I plotted the data and found out that there are

four sentiments - positive, negative, neutral, and conflict. Conflict is not needed so we can drop all the rows that contain conflict as a sentiment.

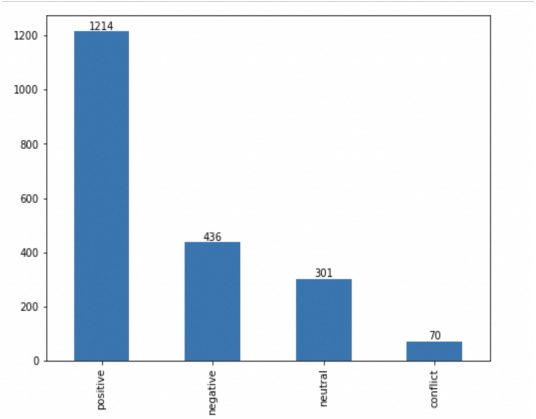

Fig 1. Before dropping the column

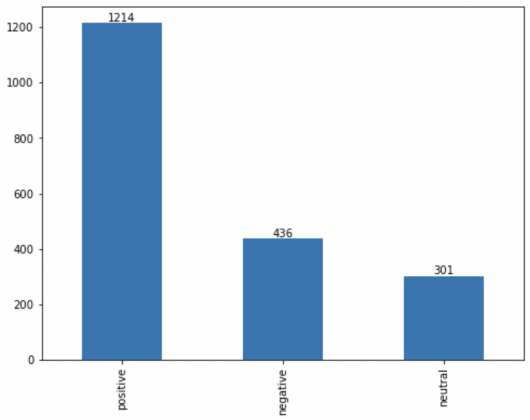

Fig 2. After dropping the column

Extracting positive and negative words

Used two files positive-words.txt and negative-words.txt to extract positive words and negative words from file and store it as a list.

Text Preprocessing

Before doing text preprocessing **removed all the words from the stopwords list which appear in the positive and negative words list.** This ensures that while applying the stopwords removal technique, no sentiment words are removed from the dataset.

Data Cleaning

Method Name: remove_punctuation(text), convert_to_lowercase(text)
Removing punctuation and converting all the data to lowercase for two predictors **text** and **term.** Used re package to match punctuation and used lower() to convert the text to lowercase.

Tokenization and stopwords removal

Method name: apply_tokenization_and_remove_stopwords(text)
Used tokenization to separate each word as a token and then passed these tokens to check if it exists in the stopwords list generated from spacy. If the token exists in stopwords, then that token is removed from the sentence and the sentence joined with the remaining tokens is returned. Applied method on the text and term predictors in the dataset.

Lemmatization

Method name: apply_lemmatization(sampleText)
Used WordNetLemmatizer to convert the words to their root form. For example: low, lower, and lowest are converted to low.

This is how the whole text preprocessing is done on text and term predictors.

Syntactic parsing

- ### Visualization of Rule 1

  A sample sentence "food outstanding little perk great" is taken and passed through the nlp() method. Spacy's displacy function is used to render the visualization of the sentence.

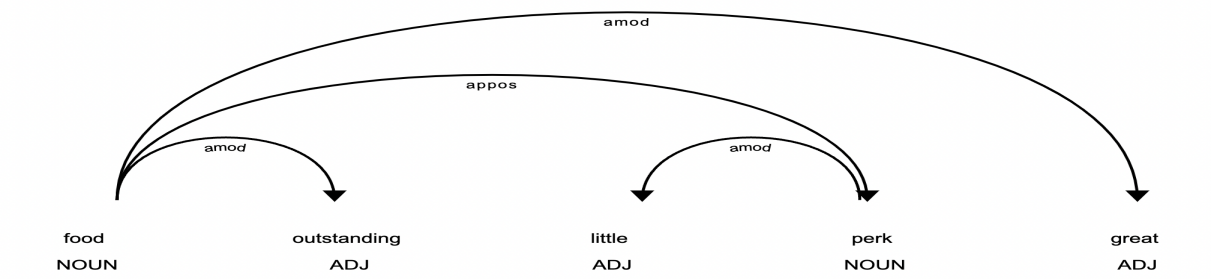

Fig 3. Visualization of Parts of Speech for a sentence.

As you can observe, for the topic of food, there are children like outstanding, perk, great which are Noun as well as adjectives. So, in the first rule, we will go through all the children of an aspect term.

- ### Custom Rule 1

  Created class CustomRule1. While initiating an instance of the object it takes the dataset, positive words list, and negative words list.
  The **__init__()** function i.e the constructor has some predefined variables as follows:
  self.pred_polarity_list - this is initialized to an empty list but afterward it will contain predicted polarities for each sentence according to the topic.
  self.sentences - list of all the **text** predictors from the dataset converted to a list.
  self.terms - Contains all the aspect terms for each text from the dataset converted to a list.
  self.positive_words - list of positive words
  self.negative_words - list of negative words
  Created methods for the class to predict the polarities for a sentence
  Method Name: **calculatePredictedPolarities(self)**
  - Loops through each sentence in the sentences list. Later pass that sentence to the nlp() method to convert it into a POS here it is called as doc.
  - Now go through each word (also called tokens) in the doc and see if it matches the term inside the self.terms listed in that index. For getting the index we used enumerate to loop through self.sentences.

- When the token matches the aspect term, we loop through all its children. For getting children, token.head.children are used.
- Later for each child of the token.children, check if the child is in the positive words list or negative words list.
- If it is in the positive words list then polarity is set to positive.
- Elif it is in the negative words list, polarity is set to negative.
- If the child is not in either of the lists then polarity is set to neutral.

Method Name: calculatePredictedPolaritiesByCount(self)

- Same as the previous method. But the slight difference is in the calculation of polarities for each sentence.
- Count of each sentiment positive, negative, or neutral is set to zero.
- Now whenever a child exists in a positive list, the positive count is increased by 1. Same for negative counts and neutral counts.
- After the whole sentence is traversed, polarity is assigned whichever has the maximum count among all three.

At end of any of these above two methods, a predicted polarity list is returned.

- **Getting the predicted list**
  - Initialized an instance of the class CustomRule1
  - Called function calculatePredictedPolarities to get a list of predicted polarities to each sentence.
  - After getting the list, a new feature/predictor is added to the dataset called pred_polarity.

Performance Evaluation

- **Class Name CustomPerformanceEvaluation**
  - While initializing the class, pass in the dataset
  - The constructor **__init__** method creates the following
  - **self.actualPolarity**: List of polarity predictor from the dataset
  - **self.predictedPolarity**: List of pred_polarity predictor from the dataset
  - **self.precision** - initialized to 0
  - **self.recall** - initialized to 0
  - There are different methods that I created to find true positive, true negative, false positive, false negative, precision and recall. The description for each method is given below.
  - **calculateTruePositive(self, label):** This method calculates the total count when the label (can be “positive”, “negative” or “neutral”) is equal to the actualPolarity element and it is also equal to the predictedPolarity element for the same idx. For getting the index we loop through the list of actualPolarity.
  - **calculateTrueNegative(self, label):** This method calculates the total count when the label is not equal to the actual polarity element and is also not equal to the predicted polarity element at a particular index.

- **calculateFalsePositive(self, label)**: calculates total count when label is not equal to actualPolarity[idx] but is equal to predictedPolarity[idx] (element).
- calculateFalseNegative(self, label): calculates total count when label is equal to actualPolarity[idx] but is not equal to predictedPolarity[idx].
- calculatePrecision(self, truePositive, false Positive): Calculated precision according to the formula

  $Precision = \frac{true\ positive}{true\ positive + false\ positive}$

  Later stored it into self.precision.
- calculateRecall(self, truePositive, falseNegative): Calculated recall according to the formula

  $Recall = \frac{true\ positive}{true\ positive + false\ negative}$

  Later stored it into self.recall.
- calcualteF1Score(self): Calculated f1 score according to the formula.

  $f1\ score = 2\frac{precision * recall}{precision + recall}$

- **Getting precision recall and f1 score for Each Sentiment separately**
  - **Positive Sentiment**
  - Created an instance of class CustomPerformanceEvaluation for positive sentiment
  - Then called function calculateTruePositive on “positive” label.
  - Similarly got true negative, false positive and false negative by calling respective methods with “positive” label.
  - Called calculatePrecision function and pass in true positive and false positive from above.
  - Similarly called calculateRecall function and pass in true positive and false negative from above.
  - Called f1 score for getting the f1 score.

  - **Negative Sentiment**
  - Followed the same process as done for positive sentiment.
  - This time “negative” label is passed while calculating all the above variables.

  - **Neutral Sentiment**
  - Similar process as done for positive sentiment
  - “neutral” label was passed instead of “positive” label.
- **Visualisation of rule 2**
  - Please refer to Fig. 1.3 for visualizing the POS(Parts Of Speech) of a sentence.
  - Same sample sentence that was taken in visualization rule 1 is visualized
  - In this case we take all children that are adjectives.

- In this case outstanding and great.
- For checking if the child is an adjective or not we used the pos_ attribute and check if it is “ADJ” or not.

- **Custom Rule 2**
  - Create class custom rule 2 where the name of each method is the same and works similar to custom rule 2 but with one difference.
  - While calculating predicted polarity we first go through all the tokens in a sentence doc. Then once we get token same to aspect term, we loop again through token children and check if that child.pos_ is equal to “ADJ” or not
  - If it is equal then we check if the child is in the positive list or negative list or else neutral which is the same logic as applied in custom rule 1.
  - The same changes are also made in the calculatePredictedPolarityByCount to check only the adjectives if they are in the list or not.
- **Getting the predicted polarities**
  - Before calculating predicted polarities we first drop the column from the dataset.
  - Later we initialize an instance with class custom rule 2 and call the calculatedPredictedPolarity function which returns the list of predicted polarities.
  - Once the predicted polarities list is received, a new feature/predictor “pred_polarity” is created with this data in the dataset.
- **Performance Evaluation**
  - Same class CustomPerformanceEvaluation is used.
  - First, an instance is created with custom rule 2 and all the process is similar to performance evaluation for custom rule 1.
- **Visualization of rule 3**
  - Please refer to Fig. 1.3 for visualizing the POS(Parts Of Speech) of a sentence.
  - This time all the nouns from the children of the aspect term is selected.
  - In this case, perk
- **Custom Rule 3**
  - A class called CustomRule3 is created. The constructor is the same as custom rule 1.
  - All methods are the same but the difference is while looping through the children of the token (aspect term) we check if the child.pos_ is equal to NOUN or not. If it is a noun then only we check if the child is in the positive list or negative list, or else it is assigned as neutral.

- **Getting predicted polarities**
  - Before getting polarities first drop the pred_polarity column from the dataset.
  - Next, create an instance of custom rule 3 and call calculatedPredictedPolarity same as for other Rules.
  - Later, create a column pred_polarity in the dataset with predicted polarities.
- **Performance Evaluation**
  - Same process as done for rule 1 and rule 2
- **Visualisation of rule 4**
  - Please refer to Fig. 1.3 for visualizing the POS(Parts Of Speech) of a sentence.
  - In this case, we first go through children of the topic. Then we first check if the child is NOUN or not.
  - If the child is NOUN then we go through its children and then check if the child is an adjective or not.
  - In this case, it is “little”
- **Custom Rule 4**
  - Created class CustomRule4 with all the methods same as above.
  - While calculating polarity, we first select if the token is equal to the aspect term and if it is equal then go through its children and check for all the nouns. Later go through children of all the NOUNs inside token and select only ADJ
  - Once we get adjectives, check that word if it is in the positive list or negative list and assign polarity accordingly. If it is not in both list then assign polarity as negative.
- **Getting predicted polarities**
  - Drop the pred_polarity column from the dataset.
  - Create the instance for custom rule 4 and then call caculatePredictedPolarity to get predicted polarities.
  - Once you get the predicted polarities, create a feature/predictor pred_polarity in the dataset column.
- **Performance Evaluation**
  - Same process as done for rule 1, rule 2, rule 3, and rule 4.
- **Visualisation of rule 5**
  - Check if it has children and calls the recursive function to get all the children. The function call sampleRecursion() will keep calling itself and go through each token and all its children.
  - In this case, it will parse the whole sentence from token → children → If one child has children → children → until there are no children for a given token. Here the child prints out as outstanding, perk, little, great.

- ## Custom Rule 5
    - Created a class called CustomRule5 with the same changes other than the above four class rules.
    - The class __init__ method is the same as the above four rules.
    - Used method **calculatePredictedPolarities(self)** to get the predicted polarities. Also used method **iteratingChild(self, token)** to go through children of each token. The description of the methods is as follows.
    - Method Name: **calculatePredictedPolarities(self)** will go through all the tokens in the doc. First, it will check if the token is in the positive or negative list and assign polarity accordingly. If it is not in either list then it will check if it has children or not.
    - If it does not have children then assign polarity neutral.
    - Else Call the function **iteratingChild(self, token).** This is a recursive function where it goes through each token inside the token's children and checks again if the token is positive or negative. If it is positive then assign positive. If it is a negative word list then assign negative else again call the recursive function.
    - If a token does not have children, then it will just return neutral and neutral polarity will be assigned.

- **Getting predicted polarities**
    - Drop the column pred_polarity and again make an instance of Custom Rule 5 class. Now get all the predicted polarities by calling **calculatePredictedPolarities()** function and later add a feature pred_polarity to the dataset with the received predicted polarities

- **Performance Evaluation**
    - Same process as done for rule 1, rule 2, rule 3 and rule 4.

- ## Analysis on performance evaluation of all rules

Label	Performance	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Positive	Precision	0.83679525222551 93	0.86977886977886 98	0.80555555555555 56	0.79841897233201 59	0.84179104477611 94
Positive	Recall	0.23228995057660 626	0.29159802306425 04	0.02388797364085 6673	0.16639209225700 163	0.23228995057660 626
Positive	F1 Score	0.36363636 36363636 365	0.43676742751388 037	0.0464	0.27539195637355 146	0.36410587475790 83

Negative	Precision	0.555555555555555556	0.6818181818181818	0.45	0.65	0.5316455696202531
Negative	Recall	0.08027522935779817	0.10321100917431193	0.02064220183486 2386	0.08944954128440367	0.09633027522935 78
Negative	F1 Score	0.14028056112224 45	0.17928286852589642	0.03947368421052632	0.15725806451612906	0.16310679611650486
Neutral	Precision	0.17730496453900 71	0.15925925925925927	0.16408876933423	0.17030567685589 52	0.18241042345276873
Neutral	Recall	0.91362126245847 18	0.14285714285714285	0.81063122923588 04	0.12956810631229235	0.93023255813953 49
Neutral	F1 Score	0.29697624190064 8	0.15061295971978983	0.27293064876957496	0.14716981132075468	0.30501089324618735

- **Failure Case**: As you can see that rule 3 performed badly in positive and negative labels. This is because we considered only children of aspect term that are nouns. Since there is less probability of nouns being in the positive and negative list most of the polarity is assigned to neutral and hence f1 score for that one is greater than the positive and negative label f1 score for rule 3. In sample visualization we can see that the token is “perk” which is used for comparison this will neither be present in positive nor negative because when we say “no perk” then we say it can be negative or if we say “lot of perks” then this can be positive. This can be said in the failure case while applying the rules.
- **Success Case**: Also, you can observe that Rule 2 performed best in positive and negative criteria because in rule 2 we considered all adjectives inside the aspect term children. Since there is a greater chance of adjectives in positive and negative lists we get the best f1 score for the positive and negative labels. Thus we can say that rule 2 is the success case and performed better than others.

● Other rules that can be made from similar insights

- Rule 5 can be now to go from aspect term → children that are adjectives → then its children and compare with a positive and negative list
- Rule 6 can be to go from aspect term → children that are noun → then to children that are adjectives → then again to their children. And later compared with the positive and negative lists.