Unique grams-

Input- 4 grams and Testing data.

Description-

1. Loading the label-

-Mentioning the path where the label will be loaded.

-Using dictionary reader to open the specified path

-Putting data in each of the output csv file.

1. Grams\_dictionary

-Basically, generates gram dictionary for a file

- Opening the path for the dictionary

-Adding one list of grams( strings) to the dictionary

-Further, using a tree and transferring the contents in dict to it

-Displaying the tree as the output.

3. Reduce\_dictionary

-This function is used to add up ngrams dictionary.

-Using result variable and initially transferring the contents of the dictionary to the assigned variable

-The dictionary consists of labels and the names of the labels, initially it prints the existing information.

- The ngarms are added up in the initialized result array.

- The result array is printed.

4. Heap top

* Initializing the heap
* Defining a counter and checking it with the root of the heap
* Further transferring the labels and ngrams to the heap.
* Printing the heap.

Output-

Trainlabelsnew.csv containing the loaded labels which in turns consists of 4 grams.

Join grams-

Input- Takes ngrams and testing data as the input, this program depends on unique grams.

Description

1. Join ngrams

-Merging all the ngrams.

-Initializing a heap.

-Heaps will consists of ngrams extracted.

-dict\_all will contain the joined ngrams.

-dict\_all will be displayed.

1. Num\_instances

-Printing the path where the dictionary consists of the instances

-Num\_instances are generated using the existing labels.

-Printing the generated instances.

1. Entropy, Info gain

-Calculates variable ratios of the given instances for entropy.

-Information gain is calculated using the entropy.

1. Heap gain

-Features of the num instances are generated.

-Initializing a heap and storing the generated features.

-Calculating heap gain by comparing the heap with the information gained earlier.

-printing the heap gain.

1. Gen\_df

-Generating all the features that includes the joined ngrams we calculated earlier.

-Printing the dictionary path and names.

-Building a new dictionary that will contain the ngram features.

-Comparing it with the existing dictionary by performing a look up approach and checking if it has the same ngrams.

-Generating the binary features.

-Further num\_instances, joined grams and the entire features are transferred to the 2 csv files as the output

Output-

Trainlabels750.csv and Testlabels750.csv, 2 files for train and test each.

dll-

Input- testing data and dll single file

Description-

1. Load labels-

-Loading the labels and mentioning the path

- A result array is created and the labels are loaded into it.

-Displaying the result array

1. Dll single file-

-Two variables are used- dict\_pattern that describes the pattern of dict and f\_lines that consists the list itself.

-Then it uses an external variable ‘p’ that does strip and split function.

-Further it is checked if lines in p are similar to f\_lines

-Finally, pattern dictionary are displayed.

1. Reduce dict

-Loading the labels into the trainlabelsnew.csv file

-create a dict\_all that will store the features generated

-The variable f\_name will have the .asm data and train labels

-Check if the features match the ones in dict\_all

-Print dict\_all

1. Num instances

* The path and labels are considered.
* Opening the dictionary reader
* Setting up counters p and n to check if the class in the label
* Displaying the variables p and n.

1. Entropy and Info gain

- Calculates variable ratios of the given instances for entropy.

-Information gain is calculated using the entropy.

1. Heap gain

* Initializes a heap and considers the root as the first element of the heap.
* Counts the items in the list and checks if the grams are in dict\_all.
* Then it determines the sum of class labels by adding previous to the next.
* The heap gain is found using the information gain and checked if it is higher than the root of the heap.
* Result variable stores the gain, it is displayed along with the heap itself.

1. gen\_df

-First, this function tests if the data is train or test.

-Further, it generates a list for binary features and dict for tmp\_pattern

-It checks if the features are in the tmp\_pattern and then adds the binary features to the list.

-The program adds num instances generated to the output csv files.

Output- testdll.csv and traindll.csv containing the dll features.

Semi Model-

Description-

Initialize Model-

gen\_data- # the 4k features! merge them into pandas corresponding id and labels train\_frequency.csv including more things instr coun daf features

#train\_daf = pd.read\_csv("train\_daf.csv")

#test\_daf = pd.read\_csv("test\_daf.csv")

#daf\_list = [0,165,91,60,108,84,42,93,152,100] #daf list for 500 grams.

# dll features

gen\_semi\_label(model)- read in data labels = np.array(X.Class - 1) # for the purpose of using multilogloss fun.

del X['Id']

del X['Class'] model3.csv'

cross\_validate(model\_list) # 9 classes prints log loss error

cross\_validate(model\_list)

semi\_learning(model\_list)

Out put – runs the model