Natural Language Processing

# Theory

* NLP serves a lot of use cases when working with text data.
* It can be used to compile documents, featurize them and compare their features.
* To featurize can be done through creating a vector count of all the possible words in the documents and simply count how many times the words of interest occur in the document(s).
* A document representing the vector of word counts is called a bag of words.
* You can then use cosine similarity to determine the similarities.
* We can improve on Bag of words by adjusting word count based on their frequency in the corpus (group of all the documents)
* Improving the bag of words can be done using TF-IDF (term Frequency-Inverse Document Frequency).
* TF is the importance of the term within the document while IDF is the importance of the term in the corpus.
* TF(d, t) is number of occurrence of term (t) in document (d) and IDF(t) = log D/t where D is total number of documents and t is number of documents with the term.
* TF-IDF = TF X IDF
* The library used for NLP in python is “nltk”

# NLP with Python - 1

* Import nltk
* Run nltk.download\_shell() and download the desired data set (in this case, stopwords). Then quit the shell with q.
* Import the data with [line.rstrip() for line in open (\*\*\*)] where \*\*\* is the directory
* Explore the data with enumerate
* Read the data into a data frame with pandas and set sep = \t, names = [‘a’, ‘b’] where a and b are column heads respectively.
* Explore the data frame with describe, info etc.
* Explore the data grouped by label and described e.g., df.groupby(‘label’).describe()
* Check out the length of the messages with df [‘length’] = df [‘message’].apply(len)

# NLP with Python - 2

* The simplest approach to converting a corpus to a vector of numbers is the Bag of Word approach where each unique word is assigned a number.
* To do this, you start with tokenization (removing unnecessary words and punctuations), create a function that:
* Remove punctuations by importing string and removing the characters in string.punctuation e.g., [c for c in sentence if c not in string.punctuation]
* Remove stop words by importing stopwords from nltk.corpus, then put in the data from removing punctuations to remove stop words e.g., [word for word in no\_punc if word not in stopwords.words(‘english)]
* Returns the final set of words
* Apply the function to the data set
* Convert the tokens to a vector (a matrix of messages against words) by
* Import CountVectorizer from the sklearn.feature\_extraction.text
* Set the bag of words vector to instantiate the CountVectorizer and pass the function from the prior step as analyzer and fit with the data e.g., bow\_transform = CountVectorizer(analyzer = text\_processor).fit(df [‘a’])
* You can check the total number of vocabulary words with len(bow\_transform.vocabulary\_) or print (len(bow\_transform.vocabulary\_))
* You can apply bow\_transform to any message instance in the data set and get the vector e.g., bow3 = bow\_transform.transform ([a]) where a is the variable name of the test message being checked
* You can check the shape of the vector for the message’s bag of words with print(bow3.shape)
* You can confirm each word in the bag of words matrix of a text message by grabbing its vector and querying the bow\_transform.get\_feature\_names()[a] where a is the vector number of the word.

# NLP with Python – 3

* You can use the ".transform” on the bow object to transform the entire set of messages e.g. bow\_transform.transform ([a]) where a is the column holding the data to be transformed.
* You can print the shape of the bow for the whole data by setting the ‘.transform’ to an object and printing the shape of the object with a.shape where a is the bow object.
* You can check the number of non-zero objects in the bag of words with a.nnz
* You can get the tfidf by importing Tfidftransformer from sklearn.feature\_extraction.text, instantiate tfidf and fit any individual bag of word in the bag of words on the tfidf object.
* You can check the idf of any word with a.idf\_[b.vocabulary\_[c]] wher a is the tfidf transformer object, b is Count vectorizer object and c is the word which idf is wanted.
* You can convert the entire bag of words corpus into a tfidf corpus by instantiating it and passing the entire bag of words e.g., a = b.transform(c) where a is the object name, b is the tfidf transformer object and c is the bag of words corpus

# Building the classifier

* The naïve bayes classifier is the best option for this instance
* Import the MultiNomialNB from sklearn.naïve\_bayes
* Instantiate the MultinomialNB and fit to the tfidf corpus and pass in the label for classification e.g., model = MultinomialNB().fit(a, b) where a is the tfidf corpus and b is the classification labels
* Pass in the tfidf of any word from the corpus into the MulinomialNB object predict to predict the class it belongs to e.g., model.predict(c) where c is the variable name of the tfidf of a word from the corpus
* You can do the same for the whole corpus by passing the tfidf of the corpus instead of the tfidf of one e.g., model.predict(d) where d is the tfidf of the corpus
* To really measure the predictive ability of the model, it should be tested on a test data set.
* In reality, the pipeline (an artificial model that saves the workflow of a text process and can repeat it every time automatically) method of sklearn is used instead of repeating all the steps for every new data set. To do this:
* Import Pipeline from sklearn.pipeline
* Instantiate it and pass in a list of every step (a tuple each) of the name and the method in the order they should be carried out. E.g., p = pipeline ([(‘a’, b), (‘c’, d)]) where a and c are step names and b and d are the methods to apply at that step respectively.
* Call the instance and fit the training data sets
* Get the predictions using the test data set
* Evaluate with classification report
* Summary is, after typical data cleaning:
* Process texts by removing punctuations and stop words
* Split into test and training set
* Create a pipeline with the next three steps below:
* Create the bag of words for the corpus with CountVectorizer
* Transform with TfidfTransformer
* Classify using MultinomialNB or any suitable classifier
* Fit the test data to the training set
* Predict with the test set
* Compare with a classification report