ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

ΙΕΛ ILSP ATHENA Research & Innovation Information Technologies

M906 Programming

Final Project: ASBA

Aspect Based Sentiment Analysis

Kylafi Christina-Theano (Theatina) – LT1200012
Zafeiri Anthi - 7115182100007

Table of Contents

Introduction

Task Description

The task of the project at hand was to carry out a comparative research on multiclass text classification. Specifically, the goal was to perform experiments and build Machine Learning models that would classify an opinion **aspect** to a **polarity** label (3 labels – **negative**, **neutral**, **positive**).

Subtasks performed:

1. **split.py** program that parses an XML file and splits it to 10 parts (35 reviews per part) and stores every part in a separate xml file
2. **train.py** that trains a model, saves it to disk and takes as parameter an array that indicates the parts that will be used for training (default parts: all 1-10)
3. **test.py** that loads a saved model and uses it for predicting the polarities for the sentence aspects of a specific part and takes as parameter which part will be used (default part: random between 1-10)
4. **experiments.py** program that uses the functions of train.py and test.py. It does **10-fold cross validation**. At each iteration, 9 parts will be used for training and 1 for testing/evaluation. The accuracy for each of the 10 folds/iterations and an average of them should be calculated as well
5. **feature selection** technique and assess if all the features used are important. Report results with at least 2 different k values.

Structure

The .zip file attached, includes also the following directories:

1. Code : the .py files for the data processing, training, test and experiments
    a. Code/EmbeddingDicts : the dictionaries of sBERT & Word2Vec text embeddings
    b. Code/fun_lib : data processing functions
2. Data
    a. Data/Parts : the .xml parts split from the main .xml dataset
3. Models : the trained models on all parts for each text embeddings type & classification algorithm
4. Logfiles : .txt files were the evaluation scores of the 10 Fold Cross Validation experiments are stored
5. Results : ROC curves of the 10 Fold Cross Validation experiments

Data Pre Processing

Data Scaling

For data preprocessing, that refers to manipulation or dropping of data before it is used in order to ensure or enhance performance, we have used multiple packages and functions, such as the sklearn.preprocessing package which provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

More specifically, we have used the classes **MinMaxScaler** and **StandardScaler**.

MinMaxScaler transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

StandardScaler standardizes a feature by subtracting the mean and then scaling to unit variance. (Unit variance means dividing all the values by the standard deviation.)

Text Embeddings

Two main sentence embeddings were tested:

sBERT

(dimension: **768**)

For the purpose of this project we concluded that it is appropriate to use sentence embedding, instead of just word embedding, to convey the tone / polarity. Sentence embedding techniques represent entire sentences and their semantic information as vectors. We came to use the SBERT framework. SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings.

Sentence-BERT (or SBERT) uses a Siamese network like architecture to provide two sentences as an input. These two sentences are then passed to BERT models and a pooling layer to generate their embeddings. Then use the embeddings for the pair of sentences as inputs to calculate the cosine similarity.

(BERT is a transformer-based machine learning technique for natural language processing (NLP) pre-training. It is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context.)

Word2Vec

(dimension: **300**)

Word embeddings is a technique where individual words are transformed into a numerical representation of the word (a vector). Where each word is mapped to one vector, this vector is then learned in a way which resembles a neural network. The vectors try to capture various characteristics of that word with regard to the overall text. These characteristics can include the semantic relationship of the word, definitions, context, etc. With these numerical representations, you can do many things like identify similarity or dissimilarity between words.

To summarize, word embedding is a learned representation for text where words that have the same meaning have a similar representation. Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text. The technique to lean a word embedding from text data that we used in this project is Word2Vec. The Word2Vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence.

Dataset creation

The initial dataset and the split parts (created by running split.py) was in .xml format. In order to convert it to training and test samples, we merged useful information from each review and sentence (**text**) tag along with each of the sentences' opinions' information (**target**, aspect – **entity** & **attribute**).

Dataset Stats

Maximum Sentence Length:	**34**	tokens
Minimum Sentence Length:	**1**	token
Average Sentence Length:	**8**	tokens

Class **0** samples (**negative**) :	**748**
Class **1** samples (**neutral**) :	**101**
Class **2** samples (**positive**) :	**1657**

As it is derived from the dataset information above, the **average sentence length** is critically small, **8** tokens long. This information led as to decide to use pretrained **word / sentence vectors** as more useful than probably vectors trained on the current dataset.

Also, the dataset was unbalanced, as there was poor class 1 (neutral) support, resulting in worse scores for this class compared to the rest 2 (negative, positive), as it is also shown in the following sections (results / comparison) using scoring tables and performance figures.

Notes

1. Not all review sentences seem to have opinions
2. Some review sentences have more than 1 opinions
3. An opinion target had bug and was skipped (<Opinion target="&quot;salt encrusted shrimp&quot; appetizer" category="FOOD#STYLE_OPTIONS" polarity="negative" from="37" to="70"/>)

Implementation

Methods

Classification algorithms

1. **Logistic Regression** (**C**: 0.1, **solver**: lbfgs, **max_iter**: 1000, **StandardScaler**())

   From sklearn.linear_model module, that implements a variety of linear models, we have used the LogisticRegression class. Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable.

2. **Support Vector Machine** (**C**: 1.0, **kernel**: poly, **MinMaxScaler**())

   From sklearn.svm module, that includes Support Vector Machine algorithms, we have used the SVC class, for support vector classification.

3. **Random Forest** (**n_estimators**: 100, **StandardScaler**())

   From sklearn.ensemble module, that includes averaging algorithms based on randomized decision trees, we have used the RandomForestClassifier class.
   (Random forest is a supervised learning algorithm that can be used both for classification and regression. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.)

Experiments

Multiple experiments were carried out, both on text embeddings and the rest of the features.

Features

Initially, a variety of features was created such as n-grams (n=2,3,4), count vectorizer, tfidf vectorizer, POS tags, token lemmas/stems, text sentiment. However, after a few experiments, many were thought to be redundant and adding more noise than information, so the following combinations comprised the main experiments:

1. **AllFeats** (text embeddings, opinion target, opinion entity, opinion attribute)
2. **AllFeats** + **POS** (text embeddings, opinion target, opinion entity, opinion attribute, most common POS counter vector)
3. **AllFeats** + **FeatSelect** (feature selection applied on AllFeats for 2-3 different k values mentioned below)
4. **AllFeats** + **POS** + **FeatSelect** (same as 3. Applied on 2.)

Feature Selection

Feature selection is the process of identifying and selecting a subset of input variables that are most relevant to the target variable. For feature selection we have used a class from sklean.feature_selection and to be more precise the mutual_info_classif function.

Mutual information is the application of information gain to feature selection, is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable.

It is used to estimate mutual information for a discrete target variable.

Mutual information is straightforward when considering the distribution of two discrete (categorical or ordinal) variables, such as categorical input and categorical output data. Nevertheless, it can be adapted for use with numerical input and output data.

Here, we experiment and report the results below, for 3 different k values:

1. k = TrainingSize / 2
2. k = TrainingSize / 4
3. k = 4 * TrainingSize / 5

Results

Metrics

The models were evaluated using multiple metrics such as **Accuracy**, **Precision**, **Recall** and **F1** (harmonic Precision - Recall mean).

- **Precision** quantifies the number of positive class predictions that actually belong to the positive class.
- **Recall** quantifies the number of positive class predictions made out of all positive examples in the dataset.
- **F-score** provides a single score that balances both the concerns of precision and recall in one number.

Additionally, recall is the **proportion of actual positives was identified correctly (sensitivity)**. However, high sensitivity with extremely low specificity results in a useless classifier, labeling everything as positive.

A system **with high recall but low precision** returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

Here, **macro averaging** is mostly used, as perhaps the most straightforward among the numerous averaging methods.

ROC Curves

This kind of curve displays the model's detection/sensitivity & "false alarm" probability at various thresholds. The greater the true positive rate and the lower the false positive rate, the better the class diagnostic ability of the model.

10 Fold Cross Validation

Below, the scores of the **10 Fold Cross Validation** (experiments.py) are shown in the tables:

Logistic Regression Scores

	Accuracy (average)		F1 (highest)					
			sBERT			Word2Vec		
	sBERT	Word2Vec	Negative	Neutral	Positive	Negative	Neutral	Positive
AllFeats	.849	.773	.87	.40	.95	.75	.36	.90
AllFeats + POS	.585		.44	.00	.78			
AllFeats + FeatSelect (k=4/5)	.434	.555	.56	0.09	.81	.40	.03	.83
AllFeats + FeatSelect (k=1/2)	.493	.504	.72	.18	.89	.47	.14	.85
AllFeats + POS + FeatSelect (k=4/5)	.412		.57	.13	.85			
AllFeats + POS + FeatSelect (k=1/2)	.383		.59	.11	.84			

Support Vector Machine Scores

	Accuracy (average)		F1 (highest)					
			sBERT			Word2Vec		
	sBERT	Word2Vec	Negative	Neutral	Positive	Negative	Neutral	Positive
AllFeats	.815	.711	.83	.32	.93	.71	.20	.86
AllFeats + POS	.538		.41	.14	.72			
AllFeats + FeatSelect (k=4/5)	.578		.72	.07	.85			
AllFeats + FeatSelect (k=1/2)	.457		.58	.11	.87			

AllFeats + POS + FeatSelect (k=4/5)	.282		.71	.46	.61			
AllFeats + POS + FeatSelect (k=1/2)	.470		.61	.25	.81			

Random Forest Scores	**Accuracy** (average)		**F_1** (highest)					
			sBERT			**Word2Vec**		
	sBERT	Word2Vec	Negative	Neutral	Positive	Negative	Neutral	Positive
AllFeats	**.855**	.742	**.88**	.00	**.96**	.54	.00	.90
AllFeats + POS	.686	.681	.27	.00	.89	.05	.00	.88
AllFeats + FeatSelect (k=4/5)	.645		.27	.00	.85			
AllFeats + FeatSelect (k=1/2)	.653		.13	.00	.86			
AllFeats + POS + FeatSelect (k=4/5)	.673		.25	.00	.84			
AllFeats + POS + FeatSelect (k=1/2)	.677		.21	.00	.84			

As we can see from the logfiles and the tables above, some combinations of parts / features and feature selection provide the classifier a good sample set to train at, therefore that classifier can generalize better, resulting in higher scores.

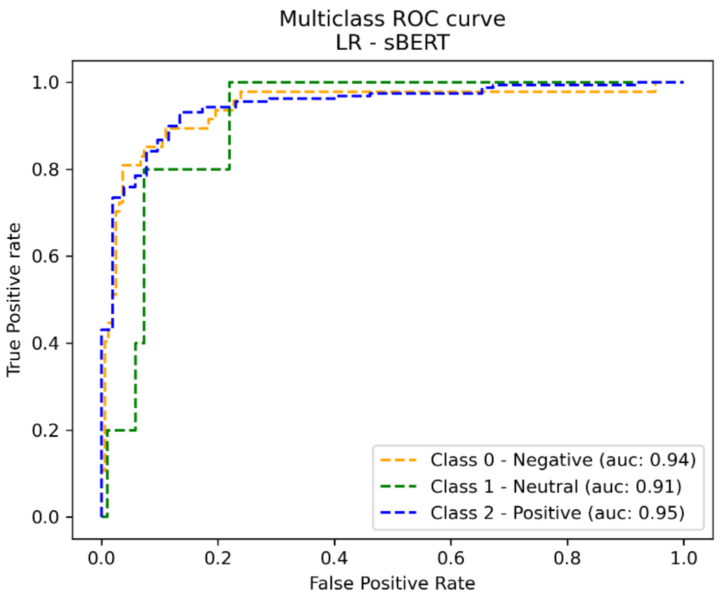
Consistently, **Random Forest** algorithm outputs higher accuracy scores and sBERT text embeddings prove to be more successful in text representation than Word2Vec in this particular task.

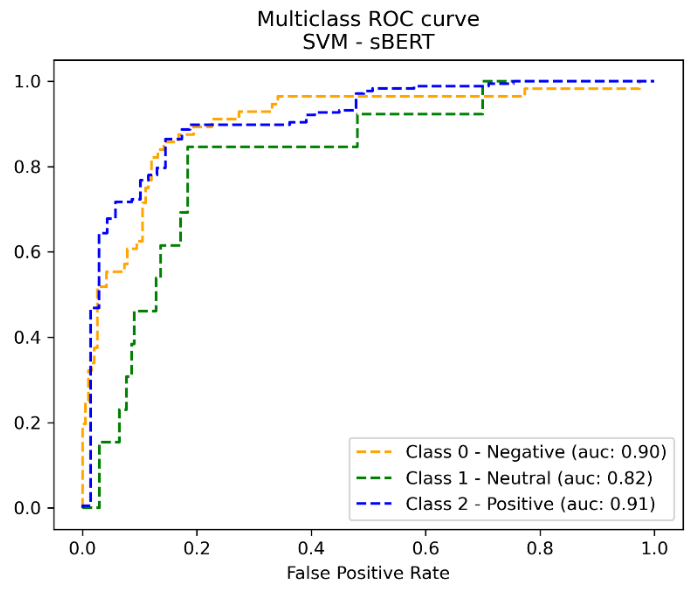
Comparison

Below, the **ROC Curve** of remarkable 10 Fold Cross Validation cases are shown for comparison as well as pattern or conclusion extraction.

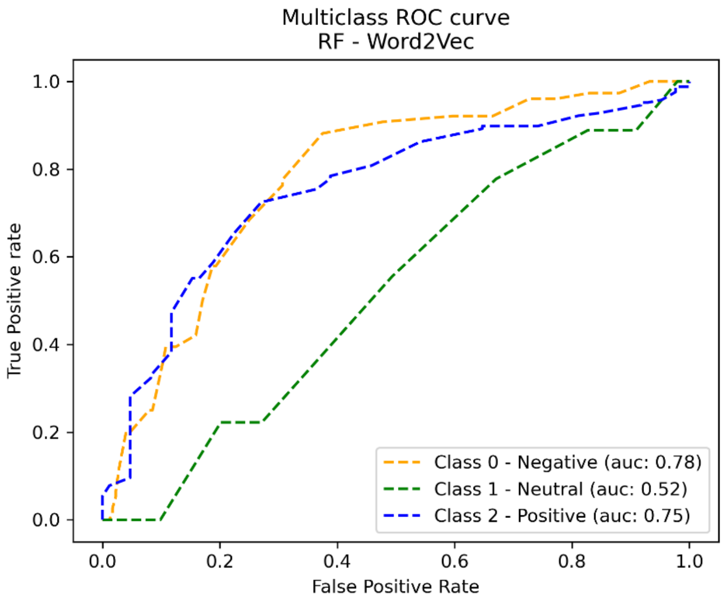
ROC Curves

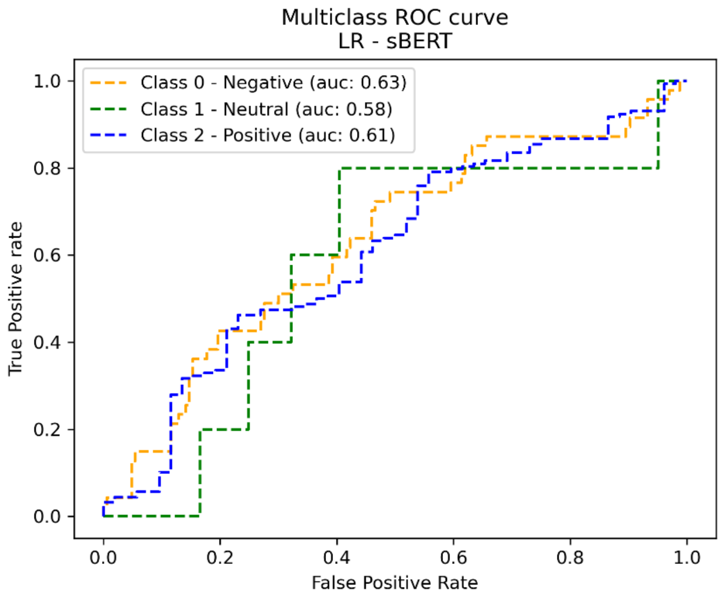
All Features (no **POS** tags)



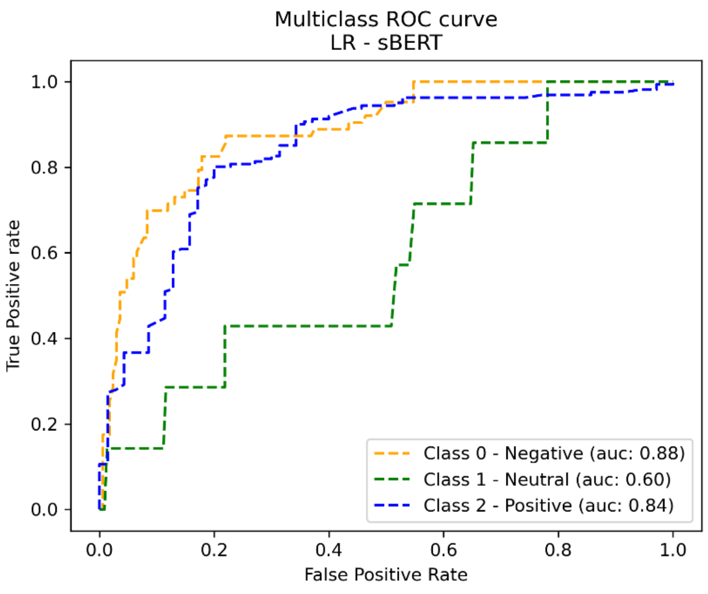


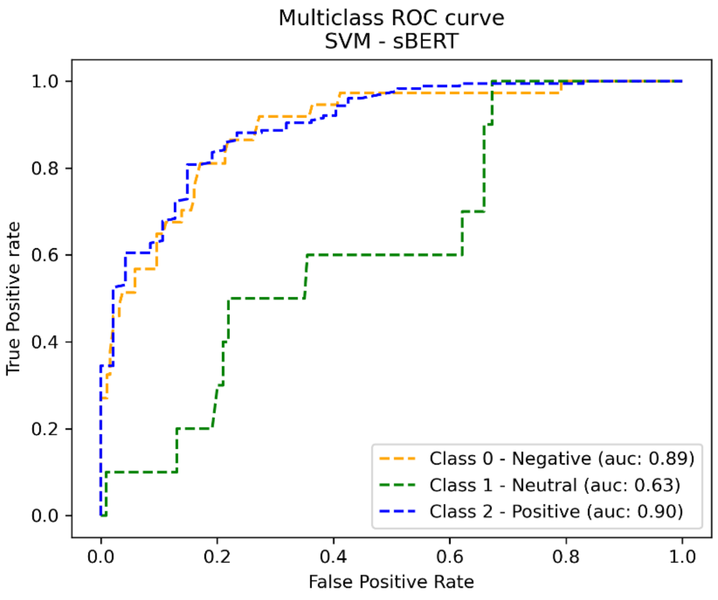
All Features + **POS** tags



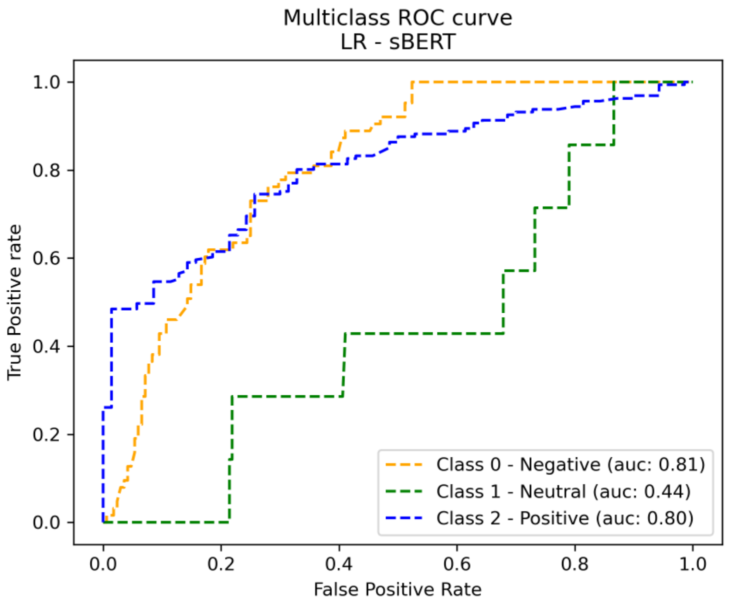


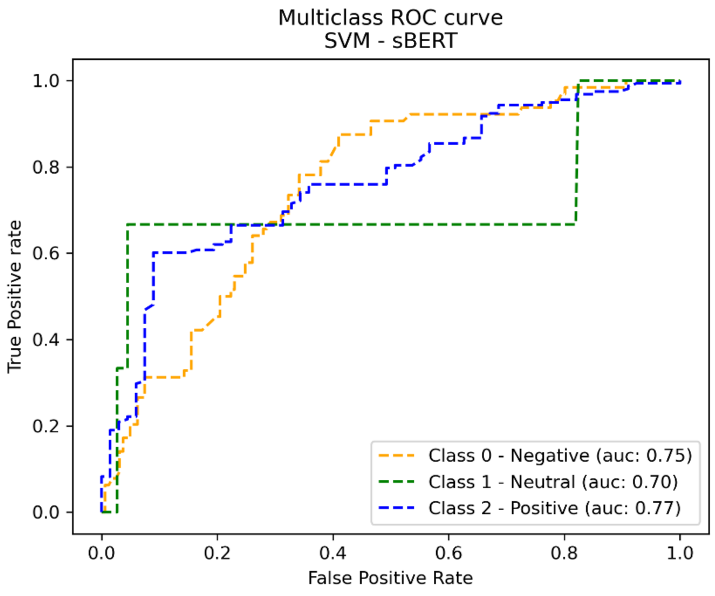
Feature Selection (no **POS** tags) – ½ training





Feature Selection + **POS** tags – ½ training





Considering the figures above, we can derive that although the accuracy of some models is not as high as that of the Random Forest's, their sensitivity (ability to distinguish 1 class VS the others) is remarkable in some parts (if not most of them).

Conclusions

Multiple conclusions could be drawn from the results, the figures, the performance scores and the log files of the models (stored in the .zip file of the project as mentioned above).

First of all, by experimenting with **feature selection** using multiple k values, it is obvious that **all** the **features** manually selected were **important**, judging by the **scores** and the **ROC** curves above.

Furthermore, the **data imbalance** creates a classifier **bias** in its performance (over the more represented/supported classes – negative & mostly positive), **sensitivity** and **accuracy**, casting it useful for certain tasks, unable to fully been characterized as successful.

sBERT text embeddings proved to be more successful as sentence representation.

POS tagging might be useful kind of information in other tasks, however in this one it seems that it offered no further information that could be used by the models during the training process for optimization.

Finally, between the 3 classification algorithms, **Random Forest** has shown a significantly better performance, result which could be used for further development and experimentation.

Future Work

The following key points could be the keywords/notes of a new, sequential project, as an optimization and further development of the present:

1. Insufficient data (especially for neutral class - 1) creates the need to further work for dataset **enrichment** with more useful features

2. More **Experiments** could be performed:
    a. Model types
    b. Model fine-tuning
    (grid search – time consuming, fine-tune on several feature combinations)
    c. Features
    d. Text embeddings (e.g. GloVe)
    e. Further data scaling

References

(Articles & Papers the project at hand was based on)

1. **Learning to Classify Text**
   https://www.nltk.org/book/ch06.html

2. **POS Tagging with NLTK**
   https://www.guru99.com/pos-tagging-chunking-nltk.html

3. **SemEval-2016 Task 5: Aspect Based Sentiment Analysis paper**
   http://galanisd.github.io/Papers/2016SemEvalABSATaskOverview.pdf

4. **How to append a new tag to an xml tree with BeaurifulSoup**
   https://stackoverflow.com/questions/56951979/how-to-append-a-new-tag-to-an-xml-tree-with-bs4

5. **Sentiment Analysis: Python's NLTK Library – pre-training, select features, classification**
   https://realpython.com/python-nltk-sentiment-analysis/

6. **EmotionsLIWClib | Extract the emotions of a given text (or sentence) using the LIWC dictionary**
   https://kandi.openweaver.com/python/mtrevi/EmotionsLIWClib#Summary

7. **Text Classification using scikit-learn**
   https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

8. **SVM for text classification**
   https://www.educative.io/answers/how-to-use-svms-for-text-classification

9. **Text Classification using SVM and Naive Bayes**
   https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34

10. **Yellowbrick's CVScores (Cross validation)**
    https://www.scikit-yb.org/en/latest/api/model_selection/cross_validation.html

11. **Solving issues (FileNotFoundError)**
    https://github.com/pytorch/pytorch/issues/35803

12. **LabelEncoder class documentation from sklearn**
    https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

13. **Label Encoding and Ordinal Encoding Guide**
    https://towardsdatascience.com/categorical-feature-encoding-547707acf4e5#8135

14. **Creating text features with different methods**
    http://uc-r.github.io/creating-text-features

15. **Leveraging N-grams to Extract Context From Text (case study example)**
    https://towardsdatascience.com/leveraging-n-grams-to-extract-context-from-text-bdc576b47049

16. **Text Analysis & Feature Engineering with NLP (overall guide and directions)**
https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d

17. **Preprocessing data package documentation from sklearn**
https://scikit-learn.org/stable/modules/preprocessing.html

18. **Feature selection module documentation from sklearn**
https://scikit-learn.org/stable/modules/feature_selection.html

19. **How to Perform Feature Selection for Regression Data**
https://machinelearningmastery.com/feature-selection-for-regression-data/

20. **Understanding Random Forests Classifiers in Python Tutorial**
https://www.datacamp.com/tutorial/random-forests-classifier-python

21. **Feature selection (summary and conclusion)**
https://medium.com/@hertan06/which-features-to-use-in-your-model-350630a1e31c

22. **RandomForestClassifier class documentation from sklearn**
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

23. **Sentence embedding techniques ideas**
https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/

24. **SentenceTransformers Documentation (SBERT documentation)**
https://www.sbert.net/

25. **Word Embedding and Word2Vec introduction**
https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

26. **Word2Vec implementation and guide**
https://towardsdatascience.com/word2vec-explained-49c52b4ccb71