­­­­­

M913 Dialogue Systems

A blue and white logo

Description automatically generated with low confidenceGraphical user interface, text

Description automatically generated

Kylafi Christina-Theano (Theatina) – LT1200012

Final Project: UniPal

**Uni**versity **Pal** – Academic Assistant

Table of Contents

[Introduction 2](#_Toc120462083)

[Task Description 2](#_Toc120462084)

[Structure 2](#_Toc120462085)

[Text Embeddings 2](#_Toc120462086)

[sBERT 2](#_Toc120462087)

[Word2Vec 2](#_Toc120462088)

[Dataset creation 3](#_Toc120462089)

[Dataset Stats 3](#_Toc120462090)

[Notes 3](#_Toc120462091)

[Implementation 3](#_Toc120462092)

[Methods 4](#_Toc120462093)

[Classification algorithms 4](#_Toc120462094)

[Experiments 4](#_Toc120462095)

[Features 4](#_Toc120462096)

[Feature Selection 5](#_Toc120462097)

[Results 6](#_Toc120462098)

[Comparison 9](#_Toc120462099)

[ROC Curves 9](#_Toc120462100)

[Conclusions 11](#_Toc120462101)

[Future Work 12](#_Toc120462102)

[References 13](#_Toc120462103)

# 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**).

## Structure

## 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.

1. **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.

1. **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

# Conclusions

# Future Work

**Notes**

1. Action uni class schedule: working well (due to different .xls format): 19-20, 20-21, 21-22

Fix: add more if’s to include them

1. Google calendar add reminders for exam/class

# References

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

1. **Learning to Classify Text**

<https://www.nltk.org/book/ch06.html>

1. **POS Tagging with NLTK**

<https://www.guru99.com/pos-tagging-chunking-nltk.html>

1. **SemEval-2016 Task 5: Aspect Based Sentiment Analysis paper**

<http://galanisd.github.io/Papers/2016SemEvalABSATaskOverview.pdf>

1. **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>
2. **Sentiment Analysis: Python's NLTK Library – pre-training, select features, classification**<https://realpython.com/python-nltk-sentiment-analysis/>
3. **EmotionsLIWClib | Extract the emotions of a given text (or sentence) using the LIWC dictionary**<https://kandi.openweaver.com/python/mtrevi/EmotionsLIWClib#Summary>
4. **Text Classification using scikit-learn**   
   <https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a>
5. **SVM for text classification**  
   https://www.educative.io/answers/how-to-use-svms-for-text-classification
6. **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
7. **Yellowbrick’s CVScores (Cross validation)**  
   https://www.scikit-yb.org/en/latest/api/model\_selection/cross\_validation.html
8. **Solving issues (FileNotFoundError)**  
   https://github.com/pytorch/pytorch/issues/35803
9. **LabelEncoder class documentation from sklearn**  
   https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
10. **Label Encoding and Ordinal Encoding Guide**  
    https://towardsdatascience.com/categorical-feature-encoding-547707acf4e5#8135
11. **Creating text features with different methods**  
    http://uc-r.github.io/creating-text-features
12. **Leveraging N-grams to Extract Context From Text (case study example)**  
    https://towardsdatascience.com/leveraging-n-grams-to-extract-context-from-text-bdc576b47049
13. **Text Analysis & Feature Engineering with NLP (overall guide and directions)**  
    https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d
14. **Preprocessing data package documentation from sklearn**  
    https://scikit-learn.org/stable/modules/preprocessing.html
15. **Feature selection module documentation from sklearn**

https://scikit-learn.org/stable/modules/feature\_selection.html

1. **How to Perform Feature Selection for Regression Data**  
   <https://machinelearningmastery.com/feature-selection-for-regression-data/>
2. **Understanding Random Forests Classifiers in Python Tutorial**  
   https://www.datacamp.com/tutorial/random-forests-classifier-python
3. **Feature selection (summary and conclusion)**https://medium.com/@hertan06/which-features-to-use-in-your-model-350630a1e31c
4. **RandomForestClassifier class documentation from sklearn**

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

1. **Sentence embedding techniques ideas**

<https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/>

1. **SentenceTransformers Documentation (SBERT documentation)**

<https://www.sbert.net/>

1. **Word Embedding and Word2Vec introduction**  
   <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>
2. **Word2Vec implementation and guide**

https://towardsdatascience.com/word2vec-explained-49c52b4ccb71