M906 - Programming II for NLP

Task: Aspect Based Sentiment Analysis - Text Processing final project

TEAM:

Ονοματεπώνυμο: Μαρία Χαμηλάκη

A.M.: 7115182300029

email: mariachamilaki@di.uoa.gr

Ονοματεπώνυμο: Ελένη Τσιλιγιάννη

A.M.: 7115182300027

email: eltsilig@di.uoa.gr

The Task:

The task of the project was to do a comparative research on a multi-class classification problem. We had to experiment and try several machine learning models in order to classify an opinion aspect to a polarity that has 3 labels - **negative**, **positive**, **neutral**.

The subtasks we applied and the corresponding files are:

1. `split.py`: this file reads the initial xml file, splits it into 10 parts each one having 35 sentences and stores these 10 parts into separate xml files.
2. `train.py`: this file has several functions, but basically trains a model, that the user can pick, and also takes an array parameter, declaring which parts of the above 10 we will use for training. The default value contains all 10 parts (1-10). The model is being saved in the disk. If the user want we can run it to prepare the whole dataset with all the features created and saved in 2 .csv files (**0_starting_data.csv**, **1_combined_features_data.csv**).
3. `test.py`: in this program we can load a model that has already been saved and we can use it for predicting the polarities of a specific part, that we can

give as an input parameter.

4. `experiments.py`: this function uses some of the functionality of traiin.py and test.py and performs a 10-fold cross validation, where at each iteration 9 parts are used for training and 1 for testing in a rotating basis. Our measurements (accuracy, precision, recall, F1) are averaged to obtain the final values.
5. `feature selection` functionality is being added in order to see which features are important. We have tested 2 different k-best values.

The Deliverable .zip:

- models/ : under this folder we have all the models saved. There are 3 types: these that mention the parts, that come from running the **train.py**, these that are under models/CV/ where they come from running 10-fold cross validation and they mention the testing part (coming from running **experiments.py,** without feature selection so the have **allFeatures** in their name) and these that come from running 10-fold cross validation, coming from running **experiments.py,** with feature selection so the have **kbest_<SOMETHING>** in their name.
- xml_parts/ : under this folder we have the splitted parts
- train_figures/ : under this folder ww have some ROC/AUC curves after running an example of **train.py**.
- **0_starting_data.csv** & **1_combined_features_data.csv files**: as mentioned we have the dataset and the features ready to read and load.
- examine_dataset.ipynb: a notebook that helped us get familiar and visualize our dataset.
- Report/ : the current report in .pdf format.

Additional saved models and the source code of the project can be found here: https://github.com/cayenne03/Programming2_for_NLP

Creating (new) features:

1. We used VADER from nltk `SentimentIntensityAnalyzer` by giving as input the processedText (this one is the raw text from the dataset by removing punctuation) so we got a value named `sentimentScore`. VADER provides four sentiment metrics:

- **Positive**: The proportion of text that is positive.
- **Negative**: The proportion of text that is negative.
- **Neutral**: The proportion of text that is neutral.
- **Compound**: The overall sentiment score, which ranges from -1 (most negative) to +1 (most positive).

We used the `compound` score, which is a normalized, weighted composite score that indicates the overall sentiment. When the `compound` score is 0, it indicates that the text is neither positive nor negative, but neutral.

Some values in `sentimentScore` are 0 because there are:

- **Balanced Sentiments**: The text contains an equal amount of positive and negative sentiments that cancel each other out.
- **Lack of Emotional Content**: The text does not contain any words that VADER recognizes as carrying sentiment (positive or negative).
- **Short/Neutral Statements**: The text might be very short or contain factual information without any emotional content.

2. After taking a look on our dataset statistics below (**examine_dataset.ipynb**) we decided to use **Word2Vec** embeddings of dimension=100 for `category` and `target` and when we have multiple words in each one of these we used the average value of the 2. Furthermore we used **Sentence-BERT** with dimension=384 in order to represent the `processedText` field which is basically the raw sentences but clean from punctuation.

```
Mean word length in 'category': 2.0674112485041882
Mean word length in 'target': 1.0981252493019544
Mean word length in 'processedText': 14.432788193059434
```

Word embeddings are a method where individual words are converted into numerical representations, known as vectors. Each word is mapped to a unique vector, which is learned in a way similar to a neural network. These vectors aim

to capture various features of a word in relation to the overall text, such as its semantic relationships, definitions, and context. With these numerical representations, you can identify similarities or differences between words.

In summary, word embeddings provide a way to represent text where words with similar meanings have similar vector representations. These methods learn real-valued vector representations for a fixed vocabulary from a text corpus. For this project, we used the Word2Vec technique to learn word embeddings. The Word2Vec algorithm employs a neural network model to learn word associations from a large text corpus. Once trained, this model can detect synonymous words or suggest additional words to complete a partial sentence.

For this project, we determined that using sentence embeddings, rather than just word embeddings, is more effective for capturing tone and polarity. Sentence embedding techniques represent entire sentences and their semantic information as vectors. We chose the SBERT framework for this purpose. SentenceTransformers is a Python framework that provides state-of-the-art embeddings for sentences, text, and images.

Sentence-BERT (SBERT) utilizes a Siamese network architecture, where two sentences are inputted into BERT models, followed by a pooling layer to generate their embeddings. These embeddings are then used to calculate the cosine similarity between the sentence pairs.

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

NOTES:

1) Not all review sentences have opinions.

2) Some sentences have more than one opinion, so we got each opinion as a new row in our dataset/dataframe.

3) We throw away rows/records from our dataset that do not have a polarity value.

4) Our dataset is very imbalanced, as we can see visually from our notebook **examine_dataset.ipynb**. Specifically:

Class samples **negative** : 748
Class samples
neutral : 101
Class samples
positive : 1657

Classification Algorithms:

1. **Logistic Regression:**

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. We used the one-vs-rest mode of the algorithm.

2. **Support Vector Machine:**

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

3. **Random Forest:**

We utilized the RandomForestClassifier class from the sklearn.ensemble module, which includes ensemble methods based on randomized decision trees.

Random Forest is a supervised learning algorithm suitable for both classification and regression tasks. It operates by creating multiple decision trees from randomly selected data samples. Each tree makes a prediction, and the algorithm determines the final prediction through a voting process. Additionally, Random Forest provides valuable insights into the importance of different features.

Experiments:

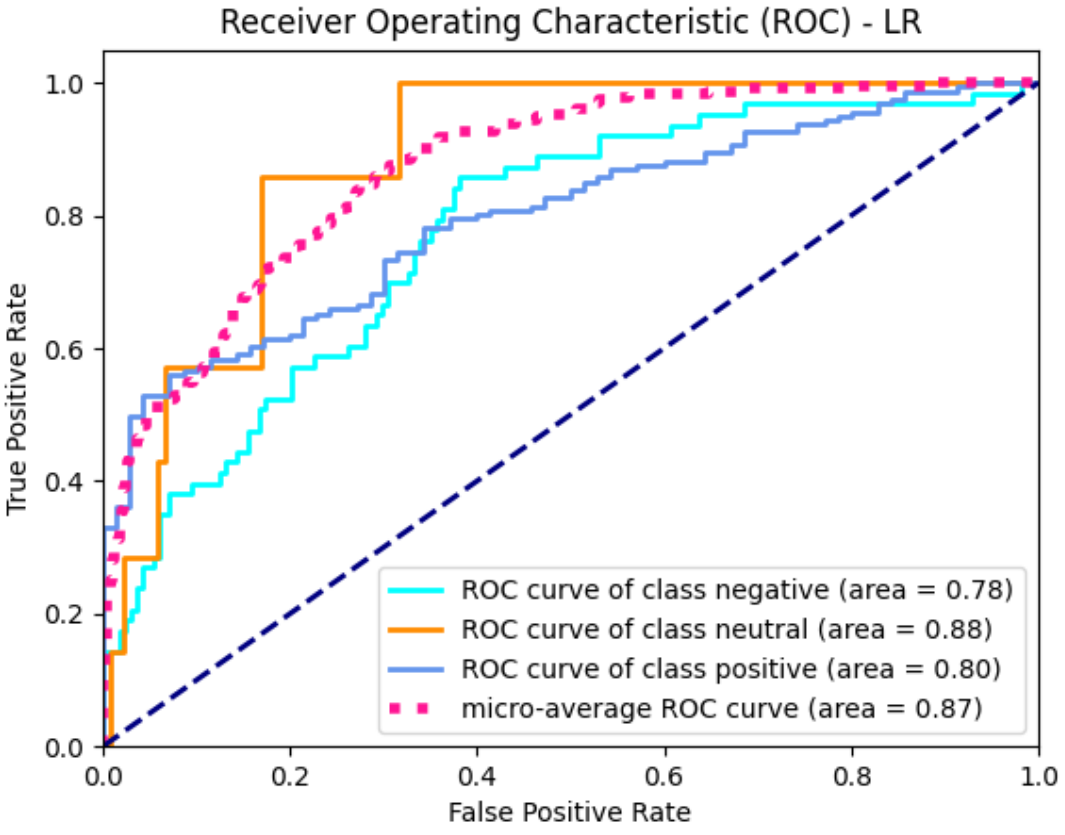
1. training on parts 2-10 and evaluation on part 1:

A) **Logistic Regression**:

```
python train.py --parts 2,3,4,5,6,7,8,9,10 --algorithm LR
.....................................................
Accuracy: 0.6623376623376623
Recall: 0.6623376623376623
Precision: 0.7121884449225444
F1 Score: 0.6816363844494286
              precision    recall  f1-score   support

    negative       0.47      0.51      0.49        63
     neutral       0.18      0.57      0.28         7
    positive       0.83      0.73      0.77       161

    accuracy                           0.66       231
   macro avg       0.49      0.60      0.51       231
weighted avg       0.71      0.66      0.68       231
```

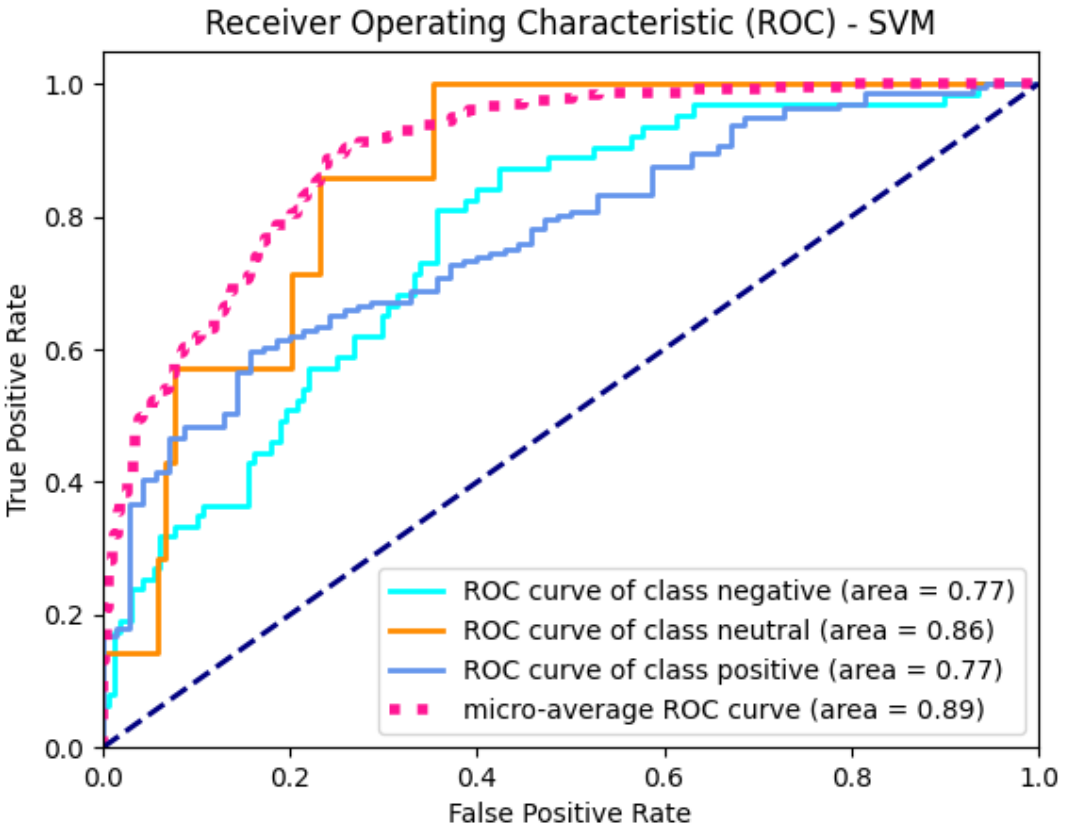


B) **SVM**:

```
python train.py --parts 2,3,4,5,6,7,8,9,10 --algorithm SVM
.....................................................
Accuracy: 0.6796536796536796
Recall: 0.6796536796536796
Precision: 0.7070707070707071
F1 Score: 0.6907661716274156
              precision    recall  f1-score   support

    negative       0.50      0.56      0.53        63
     neutral       0.21      0.43      0.29         7
    positive       0.81      0.74      0.77       161

    accuracy                           0.68       231
   macro avg       0.51      0.57      0.53       231
weighted avg       0.71      0.68      0.69       231
```

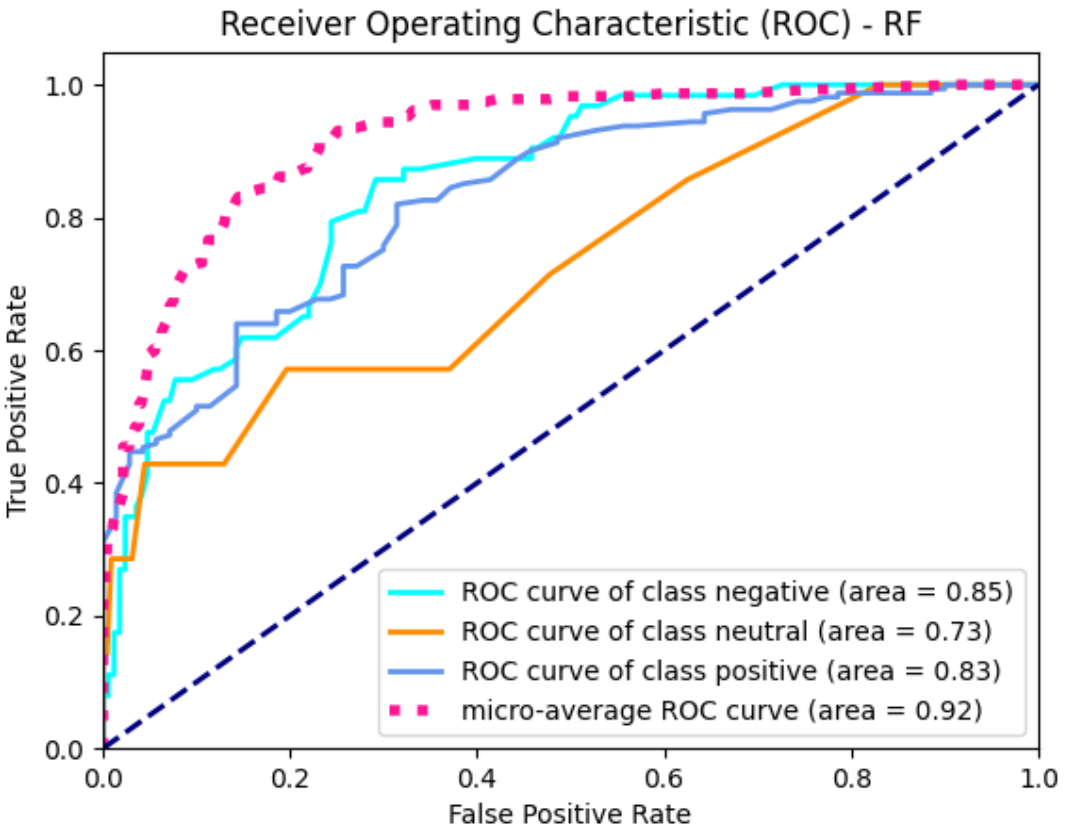


C) **Random Forest**:

```
$ python train.py --parts 2,3,4,5,6,7,8,9,10 --algorithm RF
..................................................
Accuracy: 0.7619047619047619
Recall: 0.7619047619047619
Precision: 0.7513745312658355
F1 Score: 0.7135927243698089
              precision    recall  f1-score   support

    negative       0.83      0.30      0.44        63
     neutral       0.00      0.00      0.00         7
    positive       0.75      0.98      0.85       161

    accuracy                           0.76       231
   macro avg       0.53      0.43      0.43       231
weighted avg       0.75      0.76      0.71       231
```



2. 10-fold cross validation with allFeatures included:

A) **Logistic Regression**:

```
$ python experiments.py --algorithm LR
.....................................................
================================= FINAL RESULTS ==============
Avg Accuracy: 0.7360618249361597
Avg Precision: 0.7641869232543843
Avg Recall: 0.7360618249361597
Avg F1: 0.7466053494965216
```

B) **SVM**:

```
$ python experiments.py --algorithm SVM
.....................................................
```

```
================================ FINAL RESULTS ==============
Avg Accuracy: 0.7395316082851603
Avg Precision: 0.7692395948821293
Avg Recall: 0.7395316082851603
Avg F1: 0.7495108044924185
```

C) **Random Forest**:

```
$ python experiments.py --algorithm RF
..........................................................
================================ FINAL RESULTS ==============
Avg Accuracy: 0.7598004608675434
Avg Precision: 0.757059336539829
Avg Recall: 0.7598004608675434
Avg F1: 0.7157833641729622
```

3. 10-fold cross validation k_best=100:

A) **Logistic Regression**:

```
$ python experiments.py --algorithm LR --k_best 100
..........................................................
================================ FINAL RESULTS ==============
Avg Accuracy: 0.7103388900529439
Avg Precision: 0.8148605160368119
Avg Recall: 0.7103388900529439
Avg F1: 0.7504790402315095
```

B) **SVM**:

```
$ python experiments.py --algorithm SVM --k_best 100
..........................................................
================================ FINAL RESULTS ==============
Avg Accuracy: 0.6743843913365498
Avg Precision: 0.8009666893760551
```

```
Avg Recall: 0.6743843913365498
Avg F1: 0.7221221725001458
```

C) **Random Forest**:

```
$ python experiments.py --algorithm RF --k_best 100
.....................................................
================================ FINAL RESULTS ==============
Avg Accuracy: 0.7889459987656192
Avg Precision: 0.7691796912496693
Avg Recall: 0.7889459987656192
Avg F1: 0.7661914498234269
```

4) 10-fold cross validation k_best=300:

A) **Logistic Regression**:

```
$ python experiments.py --algorithm RF --k_best 300
.....................................................
================================ FINAL RESULTS ==============
Avg Accuracy: 0.765172601609876
Avg Precision: 0.7531440336510151
Avg Recall: 0.765172601609876
Avg F1: 0.7299913956520864
```

B) **Random Forest**:

```
$ python experiments.py --algorithm RF --k_best 300
.....................................................
================================ FINAL RESULTS ==============
Avg Accuracy: 0.7672771960015514
Avg Precision: 0.7597380588075974
Avg Recall: 0.7672771960015514
Avg F1: 0.7310574773055155
```

Metrics:

Due to **imbalance of the target label “polarity”** the models were assessed using several metrics including Accuracy, Precision, Recall, and F1 Score (the harmonic mean of Precision and Recall).

- **Precision** measures the proportion of positive predictions that are actually correct.
- **Recall** measures the proportion of actual positive instances that were correctly identified by the model.
- **F1 Score** provides a single metric that balances the trade-off between Precision and Recall.

Additionally, Recall indicates the sensitivity, which is the proportion of actual positives correctly identified. However, high sensitivity with very low specificity results in a poor classifier that labels almost everything as positive.

A model with high Recall but low Precision returns many results, but most of its predictions are incorrect. Conversely, a model with high Precision but low Recall returns fewer results, but most of its predictions are correct. The ideal scenario is to have both high Precision and high Recall, resulting in many correct predictions.

In this evaluation, macro averaging was primarily used, as it is one of the most straightforward averaging methods.

ROC Curves:
ROC (Receiver Operating Characteristic) curves illustrate the model's sensitivity (true positive rate) and "false alarm" rate (false positive rate) at various thresholds. A model with a higher true positive rate and a lower false positive rate indicates better diagnostic performance.

Results & Conclusions:

- Among the three classification algorithms tested, Random Forest demonstrated significantly better performance, making it a strong candidate for further development and experimentation.

- The sentiment score feature greatly enhances the model's effectiveness.
- In both k=100 and k=300 feature selections (by printing them order by significance), features derived from SBERT embeddings consistently appeared, indicating that SBERT text embeddings are more effective for text representation than Word2Vec embeddings.
- The 'target' feature can be discarded. Instead, the 'category' feature, which includes only 12 unique values, should be converted into labels and One-Hot Encoded.
- While POS tagging can be useful in other tasks, it did not provide any additional valuable information for the models in this particular task.

What else to try further:

- Insufficient data, particularly for the neutral class, highlights the need for further dataset enrichment with additional useful features.
- Future work should include more experiments involving different model types, fine-tuning, exploring various features, and testing other embedding techniques.