Computational Argumentation Assignment - 3

How to run main.py?

- One package named "***spacytextblob***" needs to be installed to run main.py. We have provided requirements.txt for it.

- Kindly run main.py on the new arm64 image. Command for the same is:

    - docker run --mount type=bind,src="$(pwd)",dst=/mnt --workdir="/mnt" -it registry.webis.de/code-lib/public-images/upb-ca22:1.0-arm64 sh -c 'pip install -r requirements.txt && python main.py'

- Command for evaluation.py:

    - docker run --mount type=bind,src="$(pwd)",dst=/mnt --workdir="/mnt" -it registry.webis.de/code-lib/public-images/upb-ca22:1.0-arm64 sh -c 'python evaluation.py -c ./data/essay-corpus.json -p ./prediction.json -s ./data/train-test-split.csv'

Features Extraction

We extracted following features from the text:

- **Topic** - Topic of the essay
- **All Major claims** - Major claims are the main arguments of the essay.
- **All Premises** - Premises provide the support for the conclusion in an argument. If we extract the topic, claims, premises and conclusion from the essay, and then examine it together, it can help in understanding the presence/absence of confirmation bias in the essay.
- **Conclusion** - Conclusion of the essay
- **Sentiment** - We calculate overall sentiment of the essay. If the essay includes confirmation bias, the overall sentiment of the essay might tend towards negative. Therefore, we added a sentiment feature.

- **Embeddings** - Using word2vec model, we converted the above textual features into vectors.
    - Word2vec is a technique for generating word embeddings based on neural networks. Vectors close to each other have similar meaning and therefore, it helps in contextual understanding.
    - First we build the word2vec model on our corpus (essays). Then, we just extract vectors from it.

These features will help in contextual understanding of the essay. Instead of passing the whole essay, we segregate the essay as above shown features (same as how normally humans would study the essay and classify presence/absence of confirmation bias).

Model Selection and Hyper-parameter Tuning

We trained our dataset on 4 different models. We performed 10 fold cross validation and hyperparameter tuning using GridSearchCV.

Model	F1 Score
SVM	0.77
Guassian Naive Bayes	0.67
RandomForest	0.73
LSTM Neural network	0.63

SVM: We selected support vector machines for this binary classification task. SVM finds the best hyperplane segregating two classes. This internal working of svm suits very well for our data and yields an F1 score of 0.77. We performed hyper-parameter tuning for three major parameters: Regularization(C), gamma and kernel. Out of which ‘C=0.1’, ‘gamma=1’ and ‘kernel=rbf’ turned out to be the best.

Gaussian Naive Bayes - It is a probabilistic classifier algorithm based on naive bayes which treats each feature independently. It classifies the label by calculating probability assuming each feature to be independent and because of that f1 score is low.

Random Forest - It is a decision tree based algorithm. It forms decision nodes based on features one after the other. It performed well but svm performed even better.

LSTM Neural network - Here we used three variants for Neural network by using different layers which we got test accuracies of 0.5489, 0.5714 and 0.6316 respectively.

1. Flatten layer, Flattening converts input features into a 1-dimensional array for inputting it to the next layer. Output is flattened to create a single long feature vector.

2. GlobalmaxPooling1D which downsamples the input representation by taking the maximum value over the time dimension.

3. a 1D convolution layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.