CA – Assignment 2: Argument Mining

- Group: `FakeNews`
- Group members:
    - Adnan Manzoor
    - Sajjad Pervaiz
    - Kevin Taylor
    - Christoph Schäfer

Structure

```
.
└── argument-mining-assignment
    ├── Documentation.pdf
    ├── README
    ├── requirements.txt
    └── code
        ├── data
        │     ├── essay_corpus.json
        │     ├── pred.txt
        │     ├── test_BIO.txt
        │     ├── train-test-split.csv
        │     ├── train_BIO.txt
        ├── convert_to_bio.py
        ├── convert_to_train_test_bio.py
        ├── evaluation.py
        ├── model.py
```

Scripts

`essay_corpus.json`: Data corpus created in Data Acquisition assignment.

`convert_to_bio.py`: provided along with the assignment to convert json corpus to `BIO` format.

`convert_to_train_test_bio.py`: Our implementation of converting `essay_corpus.json` to bio format.

- We call `convert_to_bio` function from `convert_to_bio.py` script in our implementation to create `train_BIO.txt` and `test_BIO.txt` based on `train-test-split.csv` scheme. The files are placed in `data/` folder and will be later used to train and test the `ML` model.

`model.py`: The ML model that we use for generating predictions. `evaluation.py`: Script to evaluate the `F1` score of the `ML` model.

How to run the scripts

- On a venv install the requirements specified in `requirements.txt`

- Make sure you have the same directory structure as above otherwise adjust the paths in the scripts accordingly.
- Run `convert_to_train_test_bio.py`, which will create `train_BIO.txt` and `test_BIO.txt` in `data/`.
- Run `model.py` to generate the predictions in `data/` directory with name `pred.txt`
- Run `evaluation` script with `preds.txt` as predictions and `test_BIO.txt` as ground truth.

Model Explanation

We choose `Naive Bayes (NB)` for its simplicity and used `Bag of words` as our feature representation to train the model. `NB` achieves `Macro F1-Score: 0.235` and `Weighted F1-Score: 0.501`.

Feature Selection

We used `n-grams` to capture the maximum context around each `token` in the training dataset. To implement the model we used `Pipeline` from `Sklearn` and passed it `CountVectorizer()` that calculates word embeddings (bow) for each token. Our `n-gram` logic is the following.

```
For each token:
    if there exists 2 preceeding and succeeding tokens in a sentence`:
        n-gram_token = [preceedingToken1, preceedingToken2, targeToken, 
succeedingToken1, succeedingToken2]
    else:
        add `empty` string for corresponding slot.
```

The motivation is to capture as much context as possible around a token.