Data Mining Lab2 Report

- Preprocessing
  - Refine the raw string data: There are usually some special tokens like “<LH>” and redundant white space exist in the tweet content; therefore, we use the following code to remove them.
  - Oversampling the minority classes: There is some data imbalance exists in this emotion dataset, and overfitting can happen if we don’t alleviate it. Thus, we use SMOTE to oversampling the minority classes and keep the majority classes intact.
- Feature Engineering
  - Convert the label from string to integer: We build a label encoder mapping the labels to unique integer, and thus the numerical label can be further used by Bert model to do classification task
  - Convert text to embeddings:
    - For Bert architecture, we first use **BertTokenizer()** to tokenize the text and then convert the tokens into embeddings
    - For Bag of Word, we use **CounterVectorizer()** from scikit-learn to count the appearance of the words for each document, and extract the top 300 features as the final embeddings.
    - For TF-IDF, we use **TfidfVectorizer()**, from scikit-learn to compute the tf-idf values for each document, and extract the top 1000 features as the final embeddings.
    - For Sentence Transformers, we directly convert the whole document into a sentence embedding with some pretrained models. (We use "**all-MiniLM-L6-v2**" as our pretrained weights)
- Models
  - Ensemble methods:
    - **Multinomial Naive Bayes model**: We use **MultinomialNB()** from scikit-learn as the model to train with the embedding we obtained from BOW and TF-IDF respectively.
    - **K-Nearest Neighbors**: We use **KNeighborsClassifier()** from scikit-learn to fit the embeddings from sentence transformers, and set **k = 10**, which is the tradeoff between the performance and training time.

    After we get the prediction from three models, we then use hard voting to determine the final prediction. If there is a tie, we randomly pick an answer from the three predictions.
  - BertForSequenceClassification: We finetune two different pretrained weights
    - **bert-base-uncased**: The weights are the commonly used one released by google.
    - **BERTweet**: The weights released by VinAI are trained based on the RoBERTa procedure with 850M English Tweets as the training corpus.

The embeddings of the tokens are fed into the model to predict the probabilities of the true label for each class, and choose the greatest one as our final prediction.

- Performance

Methods	Public Score
Ensemble Method	0.38649
Bert (bert-based-uncased)	**0.53452**
Bert (Bertweet)	0.46971

- Observation & Analyzation

With the results, we can know that Bert based methods outperform the ensemble method we build significantly, and here are some analyzation and possible work for improvement:

  - **Ensemble Method**

    Here we use MultinomialNB and KNN as our estimators, and the training f1-scores of the 3 models are all around 0.44~0.47, which means 3 estimators are somehow balanced. However, the performance of the voting results decline greatly, and possible causes may be:

    - Too few estimators for this classification task: There are 8 emotion classes in this dataset, but we only use 3 different estimators for the hard major voting, which can often encounter ties and be forced to randomly select a prediction as the final answer. We think it may be better if we use more estimators, but the tradeoff must be considered between the performance and the total training and inference time.
    - The type of estimators: We use MultinomialNB since it is fast and easy to train, but we replace the model for sentence transformers with KNN due to the negative float point in the embeddings. It is possible that other machine learning models can do better prediction, but we do not develop experiments for that since it will be time-consuming to find the best one, and we want to focus on the diversity of the methods but not the best solution of each method.

  - **BertForSequenceClassification**

    We want to know whether pre-training with related datasets can get better performance or not. During the training process, we observe that the training loss of BERTweet is always higher than another one, even if we finetune it for 3 epochs (bert-base-uncased tune only 1 epoch), the final training loss is still much higher. We first consider that the situation is some effect of regularization to avoid overfitting, however, the public score tells us the truth. The possible adjustment can be hyperparameters tuning to elicit the potential capability of BERTweet (we use almost the same hyperparameters for the two weights, but BERTweet take much less time training🤔)