Assignment 1 - Machine Learning Project - Sanne Weering - s4127579

Task 1

For this task the data needed to be featurized. I did this by checking for each tweet for all of the words in the whole corpus if that word occured in the tweet, if so, the word count would be +1. The result is the function get_features, which returns a 2D NumPy array that represents word counts per tweet. I didn't add extra features.

Task 2

For this task I implemented the evaluate function, which calculates the accuracy, precision, recall and f1-measure per class and also their macro average score.
The results are the following:

```
***** Evaluation *****
  Accuracy: 0.8124560168895144
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.655897   |0.727599   |0.689890   |
 |joy        |0.677632   |0.575419   |0.622356   |
 |optimism   |0.322314   |0.317073   |0.319672   |
 |sadness    |0.628647   |0.620419   |0.624506   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.571122   |0.560127   |0.564106   |
 |-----------|-----------|-----------|-----------|
```

Task 3

For this task I implemented two different kernels, the 'poly' one and the 'rbf' one. The default one that was used in this assignment was 'linear'. For implementation I changed kernel='linear' to kernel='poly' and for the 'rbf' one to kernel='rbf'. These were the results when using the different kernels.

```
***** Evaluation *****
  Accuracy: 0.7009148486981
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.399706   |0.974910   |0.566962   |
 |joy        |0.500000   |0.019553   |0.037634   |
 |optimism   |0.500000   |0.040650   |0.075188   |
 |sadness    |0.416667   |0.039267   |0.071770   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.454093   |0.268595   |0.187889   |
 |-----------|-----------|-----------|-----------|
```

'poly' kernel

```
***** Evaluation *****
  Accuracy: 0.7885292047853624
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.517068   |0.922939   |0.662806   |
 |joy        |0.767742   |0.332402   |0.463938   |
 |optimism   |1.000000   |0.024390   |0.047619   |
 |sadness    |0.685393   |0.479058   |0.563945   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.742551   |0.439697   |0.434577   |
 |-----------|-----------|-----------|-----------|
```

'rbf' kernel

As you can see, the overall results when using the 'poly' kernel were much lower than the default 'linear' kernel. One interesting thing is that the recall of anger is really high, which means that almost all instances of the anger label were retrieved. The 'rbf' kernel also gives overall lower results than when using the linear kernel. Except for the precision scores, these are higher. The recall for anger is also really high when using this kernel. Another interesting

thing is that the precision of optimism is one, this means that all tweets in the test data that got the label optimism were in all cases the true label.

Task 4

For this part of the assignment I reset the kernel to 'linear' and I implemented a Naive Bayes classifier. The fit function is for training the classifier, it takes train data features and their labels as inputs. First it calculates the priors per class (number of sentences per class/number of all of the sentences) and then the word likelihoods per class (number of words per class/number of total word count per class). The predict function predicts labels for the test data, it takes test data features as input and returns an array of test data labels. The function first loops over each test tweet and calculates the tweet likelihood for each class, then it chooses the label with the highest probability. The performance of the Naive Bayes classifier with the features specified in task 1 are seen below:

```
***** Evaluation *****
  Accuracy: 0.7318789584799437
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.425287   |0.994624   |0.595813   |
 |joy        |0.944444   |0.094972   |0.172589   |
 |optimism   |1.000000   |0.016260   |0.032000   |
 |sadness    |0.871795   |0.178010   |0.295652   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.810382   |0.320967   |0.274014   |
 |-----------|-----------|-----------|-----------|
```

Task 5

For this task I implemented a KNN classifier. The fit function stores the features and labels of the training data. The predict function takes test data features as input, calculates for each test feature the distance between test feature and all of the training features. Then it picks the K training tweets with the lowest distances (so the most similar), gets labels of the K most similar training tweets and chooses the label with the most votes. I experimented with both cosine distance and euclidean distance in combination with different values for k, the results are seen below:

```
***** Evaluation *****
  Accuracy: 0.723082336382829
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.467933   |0.706093   |0.562857   |
 |joy        |0.423664   |0.310056   |0.358065   |
 |optimism   |0.245283   |0.105691   |0.147727   |
 |sadness    |0.439394   |0.303665   |0.359133   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.394069   |0.356376   |0.356946   |
 |-----------|-----------|-----------|-----------|
```

cosine distance, k=3

```
***** Evaluation *****
  Accuracy: 0.7181562280084448
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.453693   |0.693548   |0.548547   |
 |joy        |0.392670   |0.209497   |0.273224   |
 |optimism   |0.500000   |0.056911   |0.102190   |
 |sadness    |0.415978   |0.395288   |0.405369   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.440585   |0.338811   |0.332333   |
 |-----------|-----------|-----------|-----------|
```

euclidean distance, k=3

```
***** Evaluation *****
  Accuracy: 0.7216748768472906
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.472989   |0.706093   |0.566499   |
 |joy        |0.396078   |0.282123   |0.329527   |
 |optimism   |0.222222   |0.146341   |0.176471   |
 |sadness    |0.464286   |0.306283   |0.369085   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.388894   |0.360210   |0.360395   |
 |-----------|-----------|-----------|-----------|
```

cosine distance, k=5

```
***** Evaluation *****
  Accuracy: 0.7023223082336383
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.424107   |0.681004   |0.522696   |
 |joy        |0.349206   |0.184358   |0.241316   |
 |optimism   |0.269231   |0.056911   |0.093960   |
 |sadness    |0.393548   |0.319372   |0.352601   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.359023   |0.310411   |0.302643   |
 |-----------|-----------|-----------|-----------|
```

euclidean distance, k=5

```
***** Evaluation *****
  Accuracy: 0.7308233638282899
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.472808   |0.763441   |0.583962   |
 |joy        |0.454082   |0.248603   |0.321300   |
 |optimism   |0.225000   |0.073171   |0.110429   |
 |sadness    |0.464789   |0.345550   |0.396396   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.404170   |0.357691   |0.353022   |
 |-----------|-----------|-----------|-----------|
```

cosine distance, k=10

```
***** Evaluation *****
  Accuracy: 0.7023223082336383
 |-----------|-----------|-----------|-----------|
 |Class      |Precision  |Recall     |F1-measure |
 |-----------|-----------|-----------|-----------|
 |anger      |0.413424   |0.761649   |0.535939   |
 |joy        |0.355769   |0.103352   |0.160173   |
 |optimism   |0.571429   |0.032520   |0.061538   |
 |sadness    |0.386525   |0.285340   |0.328313   |
 |-----------|-----------|-----------|-----------|
 |MACRO-AVG  |0.431787   |0.295715   |0.271491   |
 |-----------|-----------|-----------|-----------|
```

euclidean distance, k=10

Task 6

For this task I implemented an N-fold cross-validation function named cross_validate and called this function in the main. (instead of calling train_test() in the main) This function reads in training data and extracts features and labels from it. Then the training data will be splitted in K consecutive folds and each fold is then used once as a test while the k-1 remaining folds form the training set. For each fold used as the test set the f1-score is calculated and then the average f1-score is calculated. The results are seen below:
(I used the 'svm' classifier)

```
***** Reading the dataset *****
Number of samples: 3257
Average F1-measure for 10-fold: 0.6073171750877546
```

```
***** Reading the dataset *****
Number of samples: 3257
Average F1-measure for 5-fold: 0.5992033332190994
```

Task 7

For this part of the assignment I implemented a preprocess function for preprocessing the data. (I changed the function call in the main back to train_test())
First I only did word tokenization for each sentence, this gives an overall accuracy of 0.73, which is lower than without the preprocessing. After that I removed punctuation and the accuracy went from 0.7319 to 0.7322, so a very small improvement. Then I removed the user tags, this gave an accuracy of 0.7379. The precision is in all the steps higher than without preprocessing, the recall and f1-measure is lower in all steps.