CS 445 Natural Language Processing Project 3: Text Classification Report

Project Goal and Dataset Explanation

The goal is to develop a text classification system for Turkish news articles using Naïve Bayes, Logistic Regression and CNN classification approaches. While doing so we are expected to try different hyper-parameters and find the ones that give the best accuracy without overfitting.

We were given a dataset that consists of news articles from Cumhuriyet newspaper. The documents were labeled based on their categories, there are five categories: türkiye, dünya, spor, video and yazarlar. We were provided with a train and test split, we created an additional validation split for hyper-parameter tuning in some cases.

1.1) Naïve Bayes Implementation

```
pipeline = Pipeline(steps=[
    ('tfidf', TfidfVectorizer()),
    ('multinomialnb', MultinomialNB()),
])

parameters = {
    'tfidf__ngram_range': ((1,1), (1,2)),
    'tfidf__stop_words': (None, stop_words),
    'tfidf__use_idf': (True, False),
    'multinomialnb__alpha': (0.5, 1.0),
    'multinomialnb__fit_prior': (True, False),
}

grid_search = GridSearchCV(pipeline, parameters, scoring='accuracy', verbose=1)

grid_search.fit(X_train, y_train)


print("Best score: %0.3f" % grid_search.best_score_)
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()

for param_name in sorted(parameters.keys()):
  print("\t%s: %r" % (param_name, best_parameters[param_name]))

print(classification_report(grid_search.predict(X_test), y_test))
print ('\n')
print(metrics.confusion_matrix(y_test, grid_search.predict(X_test)))
```

For the Naïve Bayes implementation I used Grid Search to find the best set of parameters that give the best output. I'm also doing the TFIDF vectorization in this part and deciding on what parameters would give the most accurate term weights. As you can see from the parameters I'm comparing the use of ngrams in TFIDF, whether to use unigrams only or using unigrams and bigrams together, I'm also deciding on whether or not to remove Turkish stop words and finally whether or not to use inverse document frequency weighting which gives more importance to terms that don't exist in all of the documents frequently, meaning the words that are unique in specific documents. For the actual Naïve Bayes parameters, I'm comparing the alpha values which are the additive smoothing parameters, alpha= 1.0 means more smoothing, meaning more is taken from probabilities that exist and given to 0 probabilities. The fit prior decides whether to learn class prior probabilities or not, if it's false a uniform prior is used.

1.2) Results of Naïve Bayes Classification with Grid Search

```
Best score: 0.746
Best parameters set:
        multinomialnb__alpha: 0.5
        multinomialnb__fit_prior: False
        tfidf__ngram_range: (1, 1)
        tfidf__stop_words: {'defa', 'gibi', 'nasıl', 'niçin', 'tüm', 'mu',
        tfidf__use_idf: True
              precision    recall  f1-score   support

       dunya       0.81      0.81      0.81       394
        spor       0.95      0.90      0.92       405
     turkiye       0.70      0.60      0.65       490
       video       0.26      0.92      0.41       117
    yazarlar       0.97      0.64      0.77       594

    accuracy                           0.73      2000
   macro avg       0.74      0.78      0.71      2000
weighted avg       0.83      0.73      0.76      2000



[[320   4  24   4  43]
 [  4 364   6   0  10]
 [ 26   3 296   5  91]
 [ 42  28 162 108  68]
 [  2   6   2   0 382]]
```

From the result we got the conclusion that the best parameters to use were alpha = 0.5 and fit_prior = False for Multinomial Naïve Bayes. This result makes sense since alpha = 1 corresponds to Laplace Smoothing and in Laplace Smoothing a lot is being taken from the higher probabilities which is inconvenient thus we resort to other Smoothing options such as add-k smoothing which 0.5 provides to us by taking lesser values from high probabilities while still not eliminating zero probabilities. Also the fit_prior = False option gave the best result which means uniform priors were used. As for the TFIDF vectorizer we see that using only ngrams gave a better score and removing stop words and using inverse document frequency also increased the score. All of these were expected outputs since the stop word removal is good for giving importance to more descriptive words as well as the inverse document frequency which is also useful to get more unique words for each document. .The accuracy turned out to be 0.73 which is pretty high although not perfect since Naïve Bayes assumes conditional independence for each case. We can also observe the f-1 scores for each category and the category that was most accurately classified was ‘spor’ while the least accurate is ‘video’ with a very low precision meaning there are many false positives.

2.1) Logistic Regression Implementation

```
pipeline = Pipeline(steps=[
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression(max_iter= 500)),
])

parameters = {
    'tfidf__stop_words': (None, stop_words),
    'clf__penalty' : ['l2'],
    'clf__C' : (1.0, 10),
    'clf__solver' : ['lbfgs', 'liblinear']
}

grid_search2 = GridSearchCV(pipeline, parameters, scoring='accuracy', verbose=1)

grid_search2.fit(X_train, y_train)


print("Best score: %0.3f" % grid_search2.best_score_)
print("Best parameters set:")
best_parameters = grid_search2.best_estimator_.get_params()

for param_name in sorted(parameters.keys()):
  print("\t%s: %r" % (param_name, best_parameters[param_name]))

print(classification_report(grid_search2.predict(X_test), y_test))
print ('\n')
print(confusion_matrix(y_test, grid_search2.predict(X_test)))
```

This time with Logistic Regression, the TFIDF vectorization operations are not investigated that much, only stop word use is investigated, this was done to save time. The TFIDF Vectorization operations are still done inside the pipeline. Here, in my parameters you can see a penalty which is used in specifying the norm used in the penalization, not all solvers support all penalties so this time for the sake of comparing solvers I kept the penalty same. C means Inverse of regularization strength and smaller values specify a stronger regularization, in this case that value is 1. Finally, the solver refers to the Algorithm to use in the optimization problem and I chose ‘liblinear’ and ‘lbfgs’ as they support both ‘l2’ penalty and ‘liblinear’ is said to perform better on smaller datasets so I wanted to test that.

In this Logistic Regression Grid Search implementation we are comparing the penalties and keeping the solver stable to examine which penalty would give the best result. We are also doing the TFIDF vectorization inside the pipeline.

```
pipeline = Pipeline(steps=[
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression(max_iter= 500)),
])

parameters = {
    'tfidf__stop_words': (None, stop_words),
    'clf__penalty' : ['l1','l2'],
    'clf__C' : (0.1, 1),
    'clf__solver' : ['liblinear']
}

grid_search2 = GridSearchCV(pipeline, parameters, scoring='accuracy', verbose=1)

grid_search2.fit(X_train, y_train)


print("Best score: %0.3f" % grid_search2.best_score_)
print("Best parameters set:")
best_parameters = grid_search2.best_estimator_.get_params()

for param_name in sorted(parameters.keys()):
  print("\t%s: %r" % (param_name, best_parameters[param_name]))

print(classification_report(grid_search2.predict(X_test), y_test))
print ('\n')
print(confusion_matrix(y_test, grid_search2.predict(X_test)))
```

2.2) Result of Logistic Regression Implementation with Grid Search

```
Best score: 0.843
Best parameters set:
        clf__C: 10
        clf__penalty: 'l2'
        clf__solver: 'liblinear'
        tfidf__stop_words: None
              precision    recall  f1-score   support

       dunya       0.87      0.82      0.84       422
        spor       0.95      0.94      0.95       388
     turkiye       0.71      0.79      0.75       382
       video       0.76      0.79      0.78       394
    yazarlar       0.95      0.90      0.93       414

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000



[[344   3  17  21  10]
 [  6 366   4   3   5]
 [ 37   2 300  57  25]
 [ 31  11  54 311   1]
 [  4   6   7   2 373]]
```

The results of both Logistic Regression Grid Search trials were the same, meaning the best score was returned when C value was 10, penalty was ‘l2’ and the solver was ‘liblinear’. Here, not removing the stop words in TFIDF vectorizer returned a better result, this may be because Logistic Regression looks at correlated features and stop words may in some cases provide that correlation. The C value is high meaning there wasn’t a need for strong regularization. And ‘l2’ penalty returned a better result than ‘l1’, while ‘liblinear’ provided a better result. ‘liblinear’ supposedly returns a better score in smaller datasets and our dataset was not too big so this makes sense. As for the accuracy we see that the accuracy improved compared to Naïve Bayes and it’s now 0.85. This was an expected result since Logistic Regression doesn’t assume conditional independence like Naïve Bayes and looks at the correlations between words. Here we see that the f-1 score of ‘spor’ is the highest which means it was again the most accurately classified one this may be because ‘spor’ data has some very specific words inside it like team names etc. We also found out that the f-1 score of ‘video’ category improved a lot compared to Naïve Bayes so we can come to the conclusion that assuming conditional independence wasn’t helpful for this data.

3.1) CNN Model with Randomized Word Embeddings (Static)

```
Max document length: 6994
Vocabulary size: 180206
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 6994)]            0
_________________________________________________________________
embedding_3 (Embedding)      (None, 6994, 100)         18020600
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 6991, 32)          12832
_________________________________________________________________
dropout_3 (Dropout)          (None, 6991, 32)          0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 3495, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 111840)            0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1118410
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 55
=================================================================
Total params: 19,151,897
Trainable params: 1,131,297
Non-trainable params: 18,020,600
_________________________________________________________________
None
Epoch 1/5
200/200 [==============================] - 7s 32ms/step - loss: 1.6019 - accuracy: 0.2248 - val_loss: 1.4666 - val_accuracy: 0.3519
Epoch 2/5
200/200 [==============================] - 6s 30ms/step - loss: 1.4274 - accuracy: 0.3339 - val_loss: 1.3885 - val_accuracy: 0.4406
Epoch 3/5
200/200 [==============================] - 6s 31ms/step - loss: 1.3650 - accuracy: 0.4368 - val_loss: 1.3405 - val_accuracy: 0.4444
Epoch 4/5
200/200 [==============================] - 6s 31ms/step - loss: 1.3146 - accuracy: 0.4474 - val_loss: 1.3030 - val_accuracy: 0.4594
Epoch 5/5
200/200 [==============================] - 6s 31ms/step - loss: 1.2841 - accuracy: 0.4545 - val_loss: 1.2629 - val_accuracy: 0.4775
Classification Report:
              precision    recall  f1-score   support

           0       0.34      0.42      0.37       322
           1       0.28      0.20      0.23       306
           2       1.00      0.00      0.01       337
           3       0.52      0.88      0.65       310
           4       0.65      0.90      0.75       325

    accuracy                           0.48      1600
   macro avg       0.56      0.48      0.40      1600
weighted avg       0.56      0.48      0.40      1600

confusion matrix:
[[134  50   0  98  40]
 [102  61   0  87  56]
 [126  76   1  70  64]
 [ 34   2   0 274   0]
 [  4  27   0   0 294]]
```

In this static CNN Model that uses Randomized Word Embeddings we are setting the trainable=False in Embedding layer to prevent the weights from being updated during training so that it's kept static. Also words are randomly initialized and then modified during training since this is a randomized model and we don't give any pretrained weights at the Embeddings layer. We can directly assume that this will return the worst score since there are no pretrained weights and there isn't an updated learning mechanism during training that provides a good training. Here the accuracy turned out to be 0.48 which is pretty low but what's more interesting is that in the 'turkiye' category the f-1 score is 0.01 with which means almost none of the 'turkiye' data was labeled correctly.

3.2) CNN Model with Randomized Word Embeddings (Non Static)

```
Max document length: 4107
Vocabulary size: 181399
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4107)]            0
_________________________________________________________________
embedding (Embedding)        (None, 4107, 100)         18139900
_________________________________________________________________
conv1d (Conv1D)              (None, 4104, 32)          12832
_________________________________________________________________
dropout (Dropout)            (None, 4104, 32)          0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 2052, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 65664)             0
_________________________________________________________________
dense (Dense)                (None, 10)                656650
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 55
=================================================================
Total params: 18,809,437
Trainable params: 18,809,437
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
200/200 [==============================] - 42s 199ms/step - loss: 1.3388 - accuracy: 0.4362 - val_loss: 0.8985 - val_accuracy: 0.6712
Epoch 2/5
200/200 [==============================] - 39s 196ms/step - loss: 0.5832 - accuracy: 0.7859 - val_loss: 0.5931 - val_accuracy: 0.7987
Epoch 3/5
200/200 [==============================] - 39s 197ms/step - loss: 0.1434 - accuracy: 0.9575 - val_loss: 0.6458 - val_accuracy: 0.7738
Epoch 4/5
200/200 [==============================] - 40s 198ms/step - loss: 0.0333 - accuracy: 0.9936 - val_loss: 0.7533 - val_accuracy: 0.7581
Epoch 5/5
200/200 [==============================] - 40s 198ms/step - loss: 0.0132 - accuracy: 0.9973 - val_loss: 0.7376 - val_accuracy: 0.7781
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.74      0.72       344
           1       0.93      0.89      0.91       307
           2       0.74      0.62      0.67       328
           3       0.72      0.76      0.74       316
           4       0.82      0.90      0.86       305

    accuracy                           0.78      1600
   macro avg       0.78      0.78      0.78      1600
weighted avg       0.78      0.78      0.78      1600

confusion matrix:
[[255   2  21  42  24]
 [  1 272  17  16   1]
 [ 46   7 203  37  35]
 [ 39   9  29 239   0]
 [ 23   2   4   0 276]]
```

Above are the results of CNN Model that uses Randomized Word Embeddings but is not kept static so the weights are being updated during training for a better result. For this we set trainable = True in Embeddings layer. The words are still randomly initialized which probably will give a worse result than that of CNNs with pretrained models. However this model is much better than the Static model and has the improved accuracy of 0.78 which is much higher. Since we allow the model to update the weights according to the results during the training we are providing a better learning process. The f-1 scores are all improved and especially ‘turkiye’ data is much more accurately classified now. The result in Static was probably because there weren’t much ‘turkiye’ data in the training data that was kept static.

3.3) CNN Model with Pretrained Word Embeddings from Project02

```
Max document length: 6994
Vocabulary size: 180656
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 6994)]            0
_________________________________________________________________
embedding (Embedding)        (None, 6994, 500)         90328000
_________________________________________________________________
conv1d (Conv1D)              (None, 6991, 32)          64032
_________________________________________________________________
dropout (Dropout)            (None, 6991, 32)          0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 3495, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 111840)            0
_________________________________________________________________
dense (Dense)                (None, 10)                1118410
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 55
=================================================================
Total params: 91,510,497
Trainable params: 91,510,497
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
200/200 [==============================] - 208s 1s/step - loss: 1.2787 - accuracy: 0.4676 - val_loss: 0.8752 - val_accuracy: 0.6556
Epoch 2/5
200/200 [==============================] - 203s 1s/step - loss: 0.5695 - accuracy: 0.7934 - val_loss: 0.6698 - val_accuracy: 0.7575
Epoch 3/5
200/200 [==============================] - 204s 1s/step - loss: 0.2277 - accuracy: 0.9248 - val_loss: 0.6548 - val_accuracy: 0.7600
Epoch 4/5
200/200 [==============================] - 204s 1s/step - loss: 0.0773 - accuracy: 0.9782 - val_loss: 0.7258 - val_accuracy: 0.7713
Epoch 5/5
200/200 [==============================] - 203s 1s/step - loss: 0.0296 - accuracy: 0.9929 - val_loss: 0.6968 - val_accuracy: 0.7931
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.82      0.75       332
           1       0.93      0.88      0.90       319
           2       0.75      0.54      0.63       325
           3       0.74      0.79      0.76       305
           4       0.88      0.94      0.91       319

    accuracy                           0.79      1600
   macro avg       0.80      0.79      0.79      1600
weighted avg       0.80      0.79      0.79      1600

confusion matrix:
[[273   3  12  31  13]
 [  9 280  12  11   7]
 [ 75  11 176  42  21]
 [ 34   5  26 240   0]
 [  6   2  10   1 300]]
```

Here we can see the results of the CNN Model that used pretrained Word Embeddings from our Project 2 in the Embedding layer. The accuracy improved just a little bit more compared to the previous model. The improvement in accuracy was expected because we are using weights now in the Embedding layer so that we have more of an idea about the relations between the words since the position of a word in the vector space is learned from text and is related to it's surrounding words that appear the most. Our data covered most words but we can still improve the word embeddings which we do in the next step. The f-1 scores are almost the same with the previous model.

3.4) CNN Model with Pretrained Word Embeddings from Turkish-Word2Vec (Static)

```
Max document length: 4107
Vocabulary size: 180232
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4107)]            0
_________________________________________________________________
embedding (Embedding)        (None, 4107, 400)         72092800
_________________________________________________________________
conv1d (Conv1D)              (None, 4104, 32)          51232
_________________________________________________________________
dropout (Dropout)            (None, 4104, 32)          0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 2052, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 65664)             0
_________________________________________________________________
dense (Dense)                (None, 10)                656650
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 55
=================================================================
Total params: 72,800,737
Trainable params: 707,937
Non-trainable params: 72,092,800
_________________________________________________________________
None
Epoch 1/5
200/200 [==============================] - 17s 74ms/step - loss: 1.2658 - accuracy: 0.4823 - val_loss: 0.8999 - val_accuracy: 0.6162
Epoch 2/5
200/200 [==============================] - 15s 73ms/step - loss: 0.8243 - accuracy: 0.6623 - val_loss: 0.7933 - val_accuracy: 0.7075
Epoch 3/5
200/200 [==============================] - 14s 72ms/step - loss: 0.6176 - accuracy: 0.7358 - val_loss: 0.7852 - val_accuracy: 0.6844
Epoch 4/5
200/200 [==============================] - 14s 73ms/step - loss: 0.4640 - accuracy: 0.8036 - val_loss: 0.6734 - val_accuracy: 0.7750
Epoch 5/5
200/200 [==============================] - 14s 72ms/step - loss: 0.3791 - accuracy: 0.8635 - val_loss: 0.6129 - val_accuracy: 0.7837
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.64      0.72       320
           1       0.89      0.93      0.91       311
           2       0.56      0.67      0.61       321
           3       0.79      0.80      0.79       316
           4       0.91      0.87      0.89       332

    accuracy                           0.78      1600
   macro avg       0.79      0.78      0.79      1600
weighted avg       0.79      0.78      0.79      1600

confusion matrix:
[[206   4  87  16   7]
 [  0 290  10   8   3]
 [ 35  10 216  42  18]
 [  5  12  47 252   0]
 [  8  11  23   0 290]]

N with Pretrained Word Embeddings from Turkish Word2Vec Model (Non Static)
```

These are the results of CNN Model with Pretrained Word Embeddings from Turkish-Word2Vec which is kept static. The results are almost the same with the results of the model with pretrained Word Embeddings from Project02 but normally if this wasn't kept static we would expect to see a better outcome which we see in the next model but still for a static model this outcome is pretty good when we compare it to the model that used randomized Word Embeddings and was kept static. The accuracy is 0.79 here and it was 0.48 in that model. The f-1 scores are again pretty similar to previous good models.

3.5) CNN Model with Pretrained Word Embeddings from Turkish-Word2Vec (Non Static)

```
Max document length: 6994
Vocabulary size: 180888
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 6994)]            0
_________________________________________________________________
embedding_1 (Embedding)      (None, 6994, 400)         72355200
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 6991, 32)          51232
_________________________________________________________________
dropout_1 (Dropout)          (None, 6991, 32)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 3495, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 111840)            0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1118410
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 55
=================================================================
Total params: 73,524,897
Trainable params: 73,524,897
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
200/200 [==============================] - 162s 806ms/step - loss: 1.3023 - accuracy: 0.4739 - val_loss: 0.9167 - val_accuracy: 0.6637
Epoch 2/5
200/200 [==============================] - 161s 808ms/step - loss: 0.6025 - accuracy: 0.7958 - val_loss: 0.6258 - val_accuracy: 0.7956
Epoch 3/5
200/200 [==============================] - 163s 814ms/step - loss: 0.3567 - accuracy: 0.8772 - val_loss: 0.6339 - val_accuracy: 0.7881
Epoch 4/5
200/200 [==============================] - 165s 825ms/step - loss: 0.2145 - accuracy: 0.9374 - val_loss: 0.6612 - val_accuracy: 0.7925
Epoch 5/5
200/200 [==============================] - 164s 821ms/step - loss: 0.1330 - accuracy: 0.9603 - val_loss: 0.7591 - val_accuracy: 0.7994
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.84      0.74       304
           1       0.90      0.94      0.92       332
           2       0.74      0.59      0.66       350
           3       0.84      0.71      0.77       306
           4       0.88      0.94      0.91       308

    accuracy                           0.80      1600
   macro avg       0.80      0.80      0.80      1600
weighted avg       0.80      0.80      0.80      1600

confusion matrix:
[[254   5  18   8  19]
 [ 11 312   2   4   3]
 [ 86  10 207  28  19]
 [ 21  17  52 216   0]
 [ 13   4   1   0 290]]
```

Here we are seeing the results of CNN Model with Pretrained Word Embeddings from Turkish-Word2Vec which is not kept static. This model returned the best accuracy which is 0.80 and has pretty good f1-scores for all of the categories. The lowest f1-score is on ‘turkiye’ category which was problematic in most of the cases. The reason why this model worked slightly better from our Project02 model could be because it was trained with more data so the weights were more accurate.

At the end we can come to the conclusion that keeping models Non Static and using pretrained Word Embeddings in Embedding Layers of our CNNs return a better score overall.

Explanations and Reasonings of some Functions Used in CNN

```
def encode_labels(y_train, y_test):
  le = LabelEncoder()
  le.fit(y_train)
  y_train_enc = le.transform(y_train)
  y_test_enc = le.transform(y_test)
  return y_train_enc, y_test_enc

y_train_enc, y_test_enc = encode_labels(y_train, y_test)
y_train_encoded_labels = np_utils.to_categorical(y_train_enc)

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train_encoded_labels, test_size=0.2)
```

This piece of code is necessary for assigning categories numerical values otherwise the CNN doesn't work properly. Also at the end we are splitting the training data to training and validation which we used in CNN as we needed to have a validation data to improve the learning. This is ran for all of the CNN models

```
def define_model(length, vocab_size):

  inputs1 = Input(shape=(length,))
  embedding1 = Embedding(vocab_size, 500, weights= [embeddingmatrix])(inputs1)
  conv1 = Conv1D(filters=32, kernel_size=4, activation='relu')(embedding1)

  drop1 = Dropout(0.5)(conv1)
  pool1 = MaxPooling1D(pool_size=2)(drop1)
  flat1 = Flatten()(pool1)

  dense1 = Dense(10, activation='relu')(flat1)
  outputs = Dense(5, activation='softmax')(dense1)
  model = Model(inputs=[inputs1], outputs=outputs)

  print(model.summary())
  plot_model(model, show_shapes=True)
  return model
```

This is the basic structure of the models I used, I played with the embedding layer's parameters for comparing different models. There is a convolution layer with ReLU, a dropout layer and a pooling layer and finally the output layer with softmax.

```
X_train_tokens = tokenizer.fit_on_texts(X_train)
encoded = tokenizer.texts_to_sequences(X_train)
encoded2 = tokenizer.texts_to_sequences(X_val)

def get_weight_matrix(embedding, vocab, embedding_size):
  vocab_size = len(vocab) + 1
  weight_matrix = np.zeros((vocab_size, embedding_size))
  for word, i in vocab.items():
    try:
      weight_matrix[i] = embedding.wv.word_vec(word)
    except KeyError:
      weight_matrix[i]=np.random.normal(0,np.sqrt(0.25),embedding_size)

  return weight_matrix

embeddingmatrix =get_weight_matrix(pretrained_project2,tokenizer.word_index, 500) #embedding matrix for project2 word2vec

embeddingmatrix2 =get_weight_matrix(trmodel,tokenizer.word_index, 400) #embedding matrix for trmodel word2vec
```

This is the code snippet that turns the Word2Vec models into weight matrixes so that we can use them in our Embedding Layers as parameters. The two models accept different output sizes hence why the embedding size is different. This part is ran before the pretrained models are used.

REFERENCES

- Chollet, F. (n.d.). The Keras Blog. Retrieved January 06, 2021, from https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
- Brownlee, J. (2020, September 02). How to Develop a Multichannel CNN Model for Text Classification. Retrieved January 06, 2021, from https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/
- Rajmehra03. (2019, January 06). A Detailed Explanation of Keras Embedding Layer. Retrieved January 06, 2021, from https://www.kaggle.com/rajmehra03/a-detailed-explanation-of-keras-embedding-layer
- Brownlee, J. (2020, September 02). How to Use Word Embedding Layers for Deep Learning with Keras. Retrieved January 06, 2021, from https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
- Qiao, F. (2019, January 08). Logistic Regression Model Tuning with scikit-learn - Part 1. Retrieved January 06, 2021, from https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5
- Brownlee, J. (2020, September 02). Deep Convolutional Neural Network for Sentiment Analysis (Text Classification). Retrieved January 06, 2021, from https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/
- Lecture Slides