```
import pandas as pd
import numpy as np
import keras
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense,Activation,SimpleRNN
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Loading data

```
data=pd.read_csv("bbc.csv")

#Dropping the irrelevant first column
data=data.drop(data.columns[[0]],axis=1)
```

Splitting data to train, validation and test set

```
X, X_test, y, y_test =
train_test_split(data['Article'],data['Class'],test_size=0.2,train_siz
e=0.8,random_state=35)
X_train, X_val, y_train, y_val =
train_test_split(X,y,test_size=0.1,random_state=35)
```

Tokenizing the sentences and aligning the data with the max length Added padding

```
#padding the sentences to a particular length so that we can feed that
to our NN
#converting the target class to number
tokenizer = Tokenizer(num_words = 10000, oov_token="<OOV>")
tokenizer.fit_on_texts(np.array(X_train))
text_word_index = tokenizer.word_index
text_sequences = tokenizer.texts_to_sequences(np.array(X_train))
X_train_final = pad_sequences(text_sequences, padding='post',
maxlen=100)
class_tokenizer = Tokenizer()
class_tokenizer.fit_on_texts(np.array(y_train))
class_word_index = class_tokenizer.word_index
y_train_final =
np.array(class_tokenizer.texts_to_sequences(np.array(y_train)))
y_train_final.reshape(y_train_final.shape[0],)
```

```
#doing the tokenization and padding for the test and validation set
also
test_text_sequences = tokenizer.texts_to_sequences(np.array(X_test))
X_test_final = pad_sequences(test_text_sequences, padding='post',
maxlen=100)
y_test_final =
np.array(class_tokenizer.texts_to_sequences(np.array(y_test)))
y_test_final.reshape(y_test_final.shape[0],)
validation_text_sequences =
tokenizer.texts_to_sequences(np.array(X_val))
X_val_final = pad_sequences(validation_text_sequences, padding='post',
maxlen=100)
y_val_final =
np.array(class_tokenizer.texts_to_sequences(np.array(y_val)))
y_val_final.reshape(y_val_final.shape[0],)
```

```
array([2, 4, 1, 4, 1, 2, 1, 5, 2, 1, 4, 3, 2, 2, 3, 2, 2, 3, 4, 2, 2,
1,
       5, 2, 4, 3, 1, 3, 3, 1, 1, 2, 2, 5, 1, 3, 1, 5, 5, 4, 5, 2, 1,
1,
       3, 1, 1, 4, 3, 3, 2, 1, 4, 4, 1, 4, 2, 3, 2, 1, 2, 3, 4, 1, 2,
1,
       2, 3, 4, 4, 1, 4, 2, 3, 3, 3, 1, 3, 3, 4, 5, 4, 3, 1, 4, 1, 4,
2,
       4, 1, 3, 3, 1, 4, 1, 1, 1, 4, 2, 5, 4, 1, 2, 2, 1, 4, 1, 2, 4,
2,
       1, 4, 2, 1, 4, 1, 1, 2, 2, 4, 4, 2, 2, 2, 3, 4, 4, 3, 1, 5, 2,
3,
       4, 4, 2, 2, 1, 2, 3, 2, 1, 1, 3, 2, 4, 1, 3, 3, 3, 2, 4, 2, 1])
```

```
#creating these to store model and accuracies
models=list()
accuracies=list()
```

Creating the vanilla RNN model and training it

```
RNNmodel = tf.keras.Sequential([
tf.keras.layers.Embedding(10000, 200, input_length=100),
tf.keras.layers.SimpleRNN(100,input_shape=(100,6),return_sequences =
False),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(24, activation='relu'),
tf.keras.layers.Dense(6, activation='softmax'),
])
RNNmodel.compile(loss='sparse_categorical_crossentropy',optimizer='ada
m',metrics=['accuracy'])
history1=RNNmodel.fit(X_train_final,y_train_final,epochs=10,validation
_data=(X_val_final,y_val_final),verbose=1)
```

```
Epoch 1/10
43/43 [==============================] - 9s 138ms/step - loss: 1.6785
```

```
- accuracy: 0.2151 - val_loss: 1.5637 - val_accuracy: 0.2876
Epoch 2/10
43/43 [==============================] - 5s 108ms/step - loss: 1.5951
- accuracy: 0.2573 - val_loss: 1.5926 - val_accuracy: 0.2876
Epoch 3/10
43/43 [==============================] - 4s 90ms/step - loss: 1.5569 -
accuracy: 0.2907 - val_loss: 1.5158 - val_accuracy: 0.3595
Epoch 4/10
43/43 [==============================] - 4s 82ms/step - loss: 1.5115 -
accuracy: 0.3256 - val_loss: 1.5251 - val_accuracy: 0.2941
Epoch 5/10
43/43 [==============================] - 5s 119ms/step - loss: 1.1533
- accuracy: 0.6199 - val_loss: 1.4141 - val_accuracy: 0.3791
Epoch 6/10
43/43 [==============================] - 5s 111ms/step - loss: 0.2570
- accuracy: 0.9455 - val_loss: 1.8627 - val_accuracy: 0.3464
Epoch 7/10
43/43 [==============================] - 4s 105ms/step - loss: 0.0277
- accuracy: 0.9985 - val_loss: 1.9613 - val_accuracy: 0.3529
Epoch 8/10
43/43 [==============================] - 4s 87ms/step - loss: 0.0054 -
accuracy: 1.0000 - val_loss: 1.9596 - val_accuracy: 0.3137
Epoch 9/10
43/43 [==============================] - 4s 88ms/step - loss: 0.0029 -
accuracy: 1.0000 - val_loss: 2.0050 - val_accuracy: 0.3203
Epoch 10/10
43/43 [==============================] - 4s 85ms/step - loss: 0.0020 -
accuracy: 1.0000 - val_loss: 2.0375 - val_accuracy: 0.3333
```

Accuracy of vanilla RNN

```
models.append(RNNmodel)
y_pred_1=RNNmodel.predict(X_test_final)
real=[]
for i in range(len(y_test_final)):
    real.append(y_test_final[i][0])
predictions1=[]
for i in range(len(y_pred_1)):
    predictions1.append(np.argmax(y_pred_1[i]))
accuracy1=(accuracy_score(predictions1,real))*100
accuracies.append(history1.history['val_accuracy'][-1]*100)
print("Predictions:",predictions1)
print(classification_report(real, predictions1,
target_names=["1","2","3","4","5"]))
print("Accuracy of vanilla RNN on test data is : ",accuracy1)
```

```
Predictions: [2, 4, 4, 4, 5, 2, 2, 1, 4, 2, 1, 5, 4, 1, 4, 3, 2, 1, 4,
5, 1, 5, 1, 3, 3, 4, 3, 4, 2, 3, 5, 2, 3, 3, 4, 3, 5, 2, 2, 4, 4, 5,
4, 2, 3, 1, 5, 1, 1, 3, 4, 3, 3, 3, 1, 3, 5, 3, 5, 1, 1, 2, 5, 2, 4,
3, 2, 2, 4, 2, 4, 3, 1, 3, 3, 3, 3, 3, 2, 2, 4, 2, 1, 4, 5, 2, 1, 3,
4, 3, 4, 2, 3, 4, 3, 5, 4, 5, 3, 1, 2, 3, 1, 4, 2, 4, 3, 2, 1, 3, 3,
```

```
3, 1, 4, 3, 1, 1, 4, 3, 1, 4, 4, 1, 1, 2, 4, 1, 1, 3, 2, 3, 1, 2, 1,
3, 3, 1, 5, 2, 1, 3, 4, 5, 5, 3, 2, 1, 2, 3, 1, 2, 4, 3, 3, 4, 4, 4,
5, 4, 3, 2, 2, 1, 4, 2, 4, 4, 4, 1, 2, 2, 5, 2, 2, 3, 1, 1, 3, 4, 2,
3, 1, 2, 4, 2, 3, 5, 5, 2, 4, 4, 2, 4, 1, 2, 4, 5, 3, 3, 2, 1, 4, 2,
3, 4, 5, 4, 1, 1, 4, 1, 2, 4, 2, 1, 4, 5, 4, 4, 1, 5, 5, 4, 4, 4, 3,
4, 3, 2, 4, 3, 3, 4, 2, 4, 3, 3, 3, 3, 4, 2, 1, 4, 4, 2, 2, 4, 1, 4,
4, 1, 2, 4, 3, 4, 4, 2, 4, 4, 5, 4, 2, 2, 4, 4, 5, 5, 2, 4, 4, 2, 2,
5, 3, 3, 1, 1, 3, 3, 4, 3, 2, 1, 1, 1, 5, 5, 5, 1, 1, 2, 4, 4, 5, 4,
3, 4, 1, 4, 4, 5, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4, 1, 1, 3, 4, 4, 4, 4,
4, 1, 1, 3, 1, 1, 4, 2, 4, 1, 5, 1, 4, 3, 1, 2, 1, 5, 1, 5, 1, 4, 5,
1, 5, 3, 4, 1, 1, 5, 4, 5, 1, 5, 3, 2, 3, 2, 4, 4, 1, 4, 2, 3, 1, 1,
2, 1, 2, 4, 1, 1, 1, 1, 1, 2, 2, 3, 2, 5, 2, 5, 2, 3, 1]
              precision    recall  f1-score   support

           1       0.52      0.41      0.46       105
           2       0.29      0.28      0.29        74
           3       0.56      0.51      0.53        83
           4       0.32      0.41      0.36        82
           5       0.20      0.23      0.21        39

    accuracy                           0.39       383
   macro avg       0.38      0.37      0.37       383
weighted avg       0.41      0.39      0.39       383

Accuracy of vanilla RNN on test data is :  38.90339425587467
```

Creating the old FFN model from assignment 4 and training it

```
# tanh activation is used
oldFFNmodel = tf.keras.Sequential([
tf.keras.layers.Embedding(10000, 200, input_length=100),
tf.keras.layers.GlobalAveragePooling1D(),
tf.keras.layers.Dense(64, activation='tanh'),
tf.keras.layers.Dense(24, activation='tanh'),
tf.keras.layers.Dense(6, activation='softmax')
])
oldFFNmodel.compile(loss='sparse_categorical_crossentropy',optimizer='
adam',metrics=['accuracy'])
history2=oldFFNmodel.fit(X_train_final,y_train_final,epochs=10,validat
ion_data=(X_val_final,y_val_final),verbose=1)

Epoch 1/10
43/43 [==============================] - 2s 32ms/step - loss: 1.6174 -
accuracy: 0.2798 - val_loss: 1.3951 - val_accuracy: 0.4641
Epoch 2/10
43/43 [==============================] - 1s 30ms/step - loss: 1.1147 -
accuracy: 0.6577 - val_loss: 0.9121 - val_accuracy: 0.7516
Epoch 3/10
43/43 [==============================] - 1s 29ms/step - loss: 0.5702 -
accuracy: 0.8932 - val_loss: 0.6241 - val_accuracy: 0.7974
Epoch 4/10
```

```
43/43 [==============================] - 2s 39ms/step - loss: 0.2412 -
accuracy: 0.9724 - val_loss: 0.4706 - val_accuracy: 0.8301
Epoch 5/10
43/43 [==============================] - 2s 40ms/step - loss: 0.1006 -
accuracy: 0.9978 - val_loss: 0.3980 - val_accuracy: 0.8497
Epoch 6/10
43/43 [==============================] - 2s 43ms/step - loss: 0.0482 -
accuracy: 1.0000 - val_loss: 0.3697 - val_accuracy: 0.8562
Epoch 7/10
43/43 [==============================] - 2s 39ms/step - loss: 0.0288 -
accuracy: 1.0000 - val_loss: 0.3519 - val_accuracy: 0.8627
Epoch 8/10
43/43 [==============================] - 2s 39ms/step - loss: 0.0197 -
accuracy: 1.0000 - val_loss: 0.3445 - val_accuracy: 0.8627
Epoch 9/10
43/43 [==============================] - 2s 36ms/step - loss: 0.0145 -
accuracy: 1.0000 - val_loss: 0.3389 - val_accuracy: 0.8758
Epoch 10/10
43/43 [==============================] - 1s 31ms/step - loss: 0.0113 -
accuracy: 1.0000 - val_loss: 0.3363 - val_accuracy: 0.8889
```

Accuracy of old FFN model from assignment 4

```
models.append(oldFFNmodel)
y_pred_2=oldFFNmodel.predict(X_test_final)
real=[]
for i in range(len(y_test_final)):
    real.append(y_test_final[i][0])
predictions2=[]
for i in range(len(y_pred_2)):
    predictions2.append(np.argmax(y_pred_2[i]))
accuracy2=(accuracy_score(predictions2,real))*100
accuracies.append(history2.history['val_accuracy'][-1]*100)
print("Predictions:",predictions2)
print(classification_report(real, predictions2,
target_names=["1","2","3","4","5"]))
print("Accuracy of old FFN on test data is : ",accuracy2)
```

```
Predictions: [1, 1, 3, 3, 4, 4, 2, 1, 2, 4, 5, 5, 1, 5, 4, 3, 4, 3, 4,
1, 2, 3, 3, 3, 3, 4, 3, 3, 3, 3, 1, 3, 3, 1, 3, 2, 2, 1, 3, 4, 1, 1,
1, 2, 1, 1, 2, 5, 3, 3, 2, 3, 4, 3, 5, 4, 4, 1, 1, 5, 1, 1, 2, 5, 5,
2, 1, 5, 2, 3, 4, 4, 1, 1, 3, 3, 3, 3, 3, 2, 1, 1, 1, 4, 1, 2, 1, 3,
1, 5, 3, 3, 1, 4, 1, 1, 3, 4, 3, 1, 4, 3, 1, 4, 3, 1, 3, 4, 1, 3, 1,
1, 2, 1, 3, 1, 1, 2, 3, 1, 4, 1, 1, 1, 1, 4, 1, 4, 5, 2, 1, 4, 3, 1,
5, 1, 1, 4, 2, 1, 4, 1, 5, 5, 3, 1, 1, 1, 3, 4, 2, 4, 5, 3, 4, 4, 4,
4, 1, 3, 2, 2, 4, 4, 5, 4, 4, 2, 2, 3, 2, 2, 1, 3, 4, 1, 1, 4, 1, 4,
3, 3, 3, 5, 3, 1, 4, 4, 3, 4, 4, 4, 1, 1, 3, 1, 1, 3, 3, 1, 4, 1, 3,
3, 4, 5, 4, 1, 3, 3, 1, 2, 3, 5, 1, 1, 1, 1, 2, 2, 2, 4, 1, 2, 4, 3,
4, 3, 1, 2, 2, 3, 5, 2, 4, 4, 3, 3, 1, 4, 4, 3, 4, 1, 1, 2, 4, 1, 5,
4, 4, 2, 4, 3, 5, 1, 3, 5, 3, 2, 4, 2, 1, 4, 4, 1, 3, 4, 2, 2, 3, 3,
5, 3, 3, 1, 2, 2, 1, 5, 3, 1, 3, 2, 1, 2, 2, 5, 1, 4, 4, 2, 1, 3, 4,
```

```
4, 3, 2, 2, 2, 4, 5, 4, 1, 2, 5, 4, 1, 2, 4, 2, 5, 1, 2, 1, 3, 4, 2,
3, 3, 3, 2, 1, 3, 3, 2, 4, 5, 3, 1, 3, 4, 3, 4, 1, 1, 1, 1, 1, 2, 5,
2, 5, 3, 4, 1, 4, 4, 5, 3, 2, 1, 3, 2, 3, 4, 2, 1, 3, 4, 4, 3, 1, 2,
1, 1, 1, 3, 4, 1, 4, 2, 3, 3, 2, 5, 1, 3, 5, 5, 4, 1, 1]
              precision    recall  f1-score   support

           1       0.90      0.92      0.91       105
           2       0.90      0.76      0.82        74
           3       0.82      0.93      0.87        83
           4       0.86      0.87      0.86        82
           5       0.97      0.90      0.93        39

    accuracy                           0.88       383
   macro avg       0.89      0.87      0.88       383
weighted avg       0.88      0.88      0.88       383

Accuracy of old FFN on test data is :  87.72845953002611
```

Creating the new FFN model

```
newFFNmodel = tf.keras.Sequential([
tf.keras.layers.Embedding(10000, 200, input_length=100),
tf.keras.layers.GlobalAveragePooling1D(),
tf.keras.layers.Dense(64, activation='tanh'),
tf.keras.layers.Dense(24, activation='tanh'),
tf.keras.layers.Dense(6, activation='softmax')
])
newFFNmodel.compile(loss='sparse_categorical_crossentropy',optimizer='
adam',metrics=['accuracy'])
```

Initializing the weights of the new FFN model with the near-optimal weights of the old FFN model from assignment 4 (the near-optimal weights are taken after training the old FFN model)

```
for l_tg,l_sr in zip(oldFFNmodel.layers,newFFNmodel.layers):
    if l_tg!='global_average_pooling1d_2' and l_sr!
='global_average_pooling1d_2':
        wk0=l_sr.get_weights()
        l_tg.set_weights(wk0)
```

Training the new FFN model

```
history3=newFFNmodel.fit(X_train_final,y_train_final,epochs=10,validat
ion_data=(X_val_final,y_val_final),verbose=1)
```

```
Epoch 1/10
43/43 [==============================] - 2s 31ms/step - loss: 1.6266 -
accuracy: 0.2827 - val_loss: 1.3952 - val_accuracy: 0.5425
Epoch 2/10
43/43 [==============================] - 1s 28ms/step - loss: 1.1042 -
accuracy: 0.7420 - val_loss: 0.8599 - val_accuracy: 0.8105
Epoch 3/10
```

```
43/43 [==============================] - 1s 28ms/step - loss: 0.4895 -
accuracy: 0.9324 - val_loss: 0.4918 - val_accuracy: 0.8889
Epoch 4/10
43/43 [==============================] - 1s 29ms/step - loss: 0.1670 -
accuracy: 0.9826 - val_loss: 0.3632 - val_accuracy: 0.9020
Epoch 5/10
43/43 [==============================] - 1s 30ms/step - loss: 0.0634 -
accuracy: 0.9978 - val_loss: 0.3144 - val_accuracy: 0.9020
Epoch 6/10
43/43 [==============================] - 1s 29ms/step - loss: 0.0309 -
accuracy: 0.9993 - val_loss: 0.2938 - val_accuracy: 0.9216
Epoch 7/10
43/43 [==============================] - 1s 29ms/step - loss: 0.0190 -
accuracy: 1.0000 - val_loss: 0.2901 - val_accuracy: 0.9216
Epoch 8/10
43/43 [==============================] - 1s 32ms/step - loss: 0.0136 -
accuracy: 1.0000 - val_loss: 0.2877 - val_accuracy: 0.9216
Epoch 9/10
43/43 [==============================] - 1s 30ms/step - loss: 0.0104 -
accuracy: 1.0000 - val_loss: 0.2843 - val_accuracy: 0.9281
Epoch 10/10
43/43 [==============================] - 1s 31ms/step - loss: 0.0083 -
accuracy: 1.0000 - val_loss: 0.2841 - val_accuracy: 0.9281
```

Accuracy of new FFN model (initialized with near-optimal weights obtained from and after training old FFN model from assignment 4)

```
models.append(newFFNmodel)
y_pred_3=newFFNmodel.predict(X_test_final)
real=[]
for i in range(len(y_test_final)):
    real.append(y_test_final[i][0])
predictions3=[]
for i in range(len(y_pred_3)):
    predictions3.append(np.argmax(y_pred_3[i]))
accuracy3=(accuracy_score(predictions3,real))*100
accuracies.append(history3.history['val_accuracy'][-1]*100)
print("Predictions:",predictions3)
print(classification_report(real, predictions3,
target_names=["1","2","3","4","5"]))
print("Accuracy of old FFN on test data is : ",accuracy3)
```

```
Predictions: [1, 1, 3, 3, 4, 4, 2, 1, 4, 4, 5, 5, 1, 5, 4, 3, 4, 3, 5,
1, 2, 1, 3, 3, 3, 2, 3, 3, 3, 3, 1, 1, 3, 1, 3, 2, 2, 1, 3, 4, 1, 1,
1, 2, 1, 1, 2, 5, 3, 3, 2, 3, 4, 3, 5, 4, 4, 1, 1, 2, 1, 3, 2, 5, 4,
2, 1, 5, 2, 3, 4, 4, 1, 1, 3, 3, 3, 3, 3, 2, 1, 1, 1, 4, 1, 2, 1, 3,
1, 5, 1, 3, 3, 4, 1, 1, 3, 4, 1, 4, 4, 3, 1, 5, 3, 1, 3, 4, 1, 3, 1,
1, 2, 1, 3, 1, 1, 2, 3, 1, 4, 1, 1, 1, 4, 4, 1, 4, 5, 2, 1, 4, 3, 1,
5, 1, 1, 4, 2, 1, 4, 1, 5, 5, 3, 1, 1, 1, 3, 4, 2, 4, 5, 3, 4, 4, 4,
4, 1, 3, 2, 2, 4, 4, 5, 4, 4, 2, 2, 3, 2, 2, 1, 3, 4, 1, 1, 4, 1, 2,
3, 3, 1, 5, 1, 1, 2, 4, 3, 4, 4, 4, 1, 1, 1, 1, 1, 3, 3, 1, 4, 4, 3,
```

```
3, 4, 5, 4, 1, 3, 3, 1, 2, 3, 5, 1, 1, 1, 1, 2, 2, 2, 4, 2, 2, 4, 3,
4, 3, 1, 2, 2, 3, 5, 2, 5, 4, 3, 3, 1, 4, 4, 1, 4, 1, 1, 2, 4, 1, 5,
4, 4, 2, 4, 3, 5, 1, 3, 5, 3, 2, 4, 2, 1, 4, 4, 1, 3, 4, 2, 2, 3, 3,
5, 3, 3, 1, 2, 4, 1, 5, 3, 1, 1, 2, 1, 2, 2, 5, 1, 4, 4, 2, 1, 3, 4,
2, 3, 2, 2, 2, 4, 5, 4, 1, 2, 5, 4, 1, 2, 4, 2, 5, 1, 3, 1, 3, 4, 2,
3, 3, 3, 2, 1, 3, 3, 2, 3, 5, 3, 1, 3, 4, 3, 4, 1, 1, 1, 1, 1, 2, 5,
2, 5, 3, 4, 1, 4, 4, 5, 3, 2, 1, 3, 2, 3, 1, 2, 1, 1, 4, 4, 3, 1, 2,
1, 1, 1, 2, 4, 1, 4, 2, 1, 3, 5, 5, 1, 3, 5, 5, 4, 1, 1]
              precision    recall  f1-score   support

           1       0.85      0.92      0.89       105
           2       0.91      0.80      0.85        74
           3       0.88      0.92      0.90        83
           4       0.90      0.88      0.89        82
           5       0.95      0.92      0.94        39

    accuracy                           0.89       383
   macro avg       0.90      0.89      0.89       383
weighted avg       0.89      0.89      0.89       383

Accuracy of old FFN on test data is :  88.77284595300262
```

Creating the majority voting ensemble model

```
E_pred = []
for i in range(0,len(predictions1)):
    if predictions1[i] == predictions2[i]:
        E_pred.append( predictions1[i])
    elif predictions2[i] == predictions3[i]:
        E_pred.append(predictions2[i])
    elif predictions1[i] == predictions3[i]:
        E_pred.append(predictions1[i])
    else:
        E_pred.append(predictions2[i])
E_pred1 = E_pred
```

Accuracy of the majority voting ensemble model

```
accuracy4=(accuracy_score(E_pred1,real))*100
print(classification_report(real, E_pred1,
target_names=["1","2","3","4","5"]))
print("Majority voting ensemble accuracy on test data is :
",accuracy4)
```

```
              precision    recall  f1-score   support

           1       0.90      0.94      0.92       105
           2       0.93      0.77      0.84        74
           3       0.85      0.94      0.89        83
           4       0.85      0.88      0.86        82
           5       0.97      0.87      0.92        39
```

```
    accuracy                           0.89       383
   macro avg       0.90      0.88      0.89       383
weighted avg       0.89      0.89      0.89       383

Majority voting ensemble accuracy on test data is :  88.77284595300262
```

Creating the weighted voting ensemble model

```
weights = accuracies
E_pred = []
sum_acc = accuracies[0]+accuracies[1]+accuracies[2]
for i in range(0,len(predictions1)):
    E_pred.append(round((accuracies[0]*predictions1[i]
+accuracies[1]*predictions2[i]+accuracies[2]*predictions3[i])/
sum_acc))
```

Accuracy of the weighted voting ensemble model

```
accuracy5=(accuracy_score(E_pred,real))*100
print("Predictions:",E_pred)
print(classification_report(real, E_pred,
target_names=["1","2","3","4","5"]))
print("Weighted voting ensemble accuracy on test Data is :
",accuracy5)
```

```
Predictions: [1, 1, 3, 3, 4, 4, 2, 1, 3, 4, 4, 5, 1, 4, 4, 3, 4, 3, 4,
2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 3, 1, 3, 2, 2, 1, 3, 4, 1, 2,
1, 2, 1, 1, 2, 4, 3, 3, 2, 3, 4, 3, 4, 4, 4, 1, 2, 3, 1, 2, 2, 5, 4,
2, 1, 5, 2, 3, 4, 4, 1, 1, 3, 3, 3, 3, 3, 2, 1, 1, 1, 4, 2, 2, 1, 3,
1, 5, 2, 3, 2, 4, 1, 2, 3, 4, 2, 2, 4, 3, 1, 4, 3, 1, 3, 4, 1, 3, 1,
1, 2, 1, 3, 1, 1, 2, 3, 1, 4, 1, 1, 1, 2, 4, 1, 4, 5, 2, 1, 4, 3, 1,
5, 1, 1, 4, 2, 1, 4, 1, 5, 5, 3, 1, 1, 1, 3, 4, 2, 4, 5, 3, 4, 4, 4,
4, 1, 3, 2, 2, 4, 4, 5, 4, 4, 2, 2, 3, 2, 2, 1, 3, 4, 1, 1, 4, 1, 3,
3, 3, 2, 5, 2, 1, 3, 4, 3, 4, 4, 4, 1, 1, 2, 1, 2, 3, 3, 1, 4, 3, 3,
3, 4, 5, 4, 1, 3, 3, 1, 2, 3, 5, 1, 1, 2, 1, 2, 2, 2, 4, 2, 2, 4, 3,
4, 3, 1, 2, 2, 3, 5, 2, 4, 4, 3, 3, 1, 4, 4, 2, 4, 1, 1, 2, 4, 1, 5,
4, 4, 2, 4, 3, 5, 1, 3, 5, 3, 2, 4, 2, 1, 4, 4, 2, 3, 4, 2, 2, 3, 3,
5, 3, 3, 1, 2, 3, 1, 5, 3, 1, 2, 2, 1, 2, 2, 5, 1, 4, 4, 2, 1, 3, 4,
3, 3, 2, 2, 2, 4, 5, 4, 1, 2, 5, 4, 1, 2, 4, 2, 4, 1, 3, 1, 3, 4, 2,
3, 3, 3, 2, 1, 3, 3, 2, 4, 4, 3, 1, 3, 4, 3, 4, 1, 2, 1, 2, 1, 2, 5,
2, 5, 3, 4, 1, 4, 4, 5, 3, 2, 2, 3, 2, 3, 2, 2, 1, 2, 4, 4, 3, 1, 2,
1, 1, 1, 3, 4, 1, 4, 2, 2, 3, 3, 5, 1, 3, 5, 5, 4, 1, 1]
              precision    recall  f1-score   support

           1       0.91      0.78      0.84       105
           2       0.68      0.80      0.73        74
           3       0.82      0.92      0.86        83
           4       0.81      0.84      0.83        82
           5       1.00      0.72      0.84        39

    accuracy                           0.82       383
   macro avg       0.84      0.81      0.82       383
```

```
weighted avg       0.83      0.82      0.82       383

Weighted voting ensemble accuracy on test Data is :  81.98433420365535
```

Number of instances misclassified in vanilla RNN model but correctly classified in the ensemble model

```
count1 = 0
for i in range(0,len(predictions1)):
    if predictions1[i] != real[i] and E_pred1[i] == real[i]:
        count1 += 1
print("Number of instances misclassified in vanilla RNN model but
correctly classified in the ensemble model: ", count1)
```

```
Number of instances misclassified in vanilla RNN model but correctly
classified in the ensemble model:  198
```

Number of instances misclassified in old FFN model but correctly classified in the ensemble model

```
count2 = 0
for i in range(0,len(predictions2)):
    if predictions2[i] != real[i] and E_pred1[i] == real[i]:
        count2 += 1
print("Number of instances misclassified in old FFN model but
correctly classified in the ensemble model: ", count2)
```

```
Number of instances misclassified in old FFN model but correctly
classified in the ensemble model:  7
```

Number of instances misclassified in new FFN model but correctly classified in the ensemble model

```
count3 = 0
for i in range(0,len(predictions3)):
    if predictions3[i] != real[i] and E_pred1[i] == real[i]:
        count3 += 1
print("Number of instances misclassified in old FFN model but
correctly classified in the ensemble model: ", count3)
```

```
Number of instances misclassified in old FFN model but correctly
classified in the ensemble model:  7
```

Number of instances wrongly classified by all three models but correctly classified by the ensemble

```
count4 = 0
for i in range(0,len(predictions1)):
    if predictions1[i] != real[i] and predictions2[i] != real[i] and
predictions3[i] != real[i] and E_pred[i] == real[i]:
        count4 += 1
print("Number of instances wrongly classified by all three models but
correctly classified by the ensemble: ", count4)
```

```
Number of instances wrongly classified by all three models but
correctly classified by the ensemble:  1
```