Exercise 9.3

Brandon Sams

12May2020

Part 1: Neural Network Classifier with Scikit

Using the multi-label classifier dataset from earlier exercises (categorized-comments.jsonl in the reddit folder), fit a neural network classifier using scikit-learn. Use the code found in chapter 12 of the Applied Text Analysis with Python book as a guideline. Report the accuracy, precision, recall, F1-score, and confusion matrix.

In [1]:
```
import pandas as pd, numpy as np, json, re, pickle
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, auc, precision_recall_fscore_support
from sklearn.metrics import classification_report
from sklearn.neural_network import MLPClassifier
```

In [2]:
```
def read_data(file):
    """
    Take a json file location and
    read the file into a pandas data frame
    Args: full path to file
    Returns: pandas dataframe with data from file
    """

    data = []

    with open(file) as f:
        for line in f:
            data.append(json.loads(line))

    # convert to data frame

    return pd.DataFrame(data)
```

In [3]:
```
# read controversy data

con_df = read_data('categorized-comments.jsonl')

# check size, structure and categories

print('Size: ', len(con_df), '\n',
      'Shape: ', con_df.info(), '\n',
      'Categories: ', con_df.cat.unique())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606476 entries, 0 to 606475
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   cat     606476 non-null  object
 1   txt     606476 non-null  object
dtypes: object(2)
memory usage: 9.3+ MB
Size:  606476
 Shape:  None
 Categories:  ['sports' 'science_and_technology' 'video_games']
```

In [4]:
```
def clean_text(text):
    """
    Remove punctuations and special characters, makes lower case
    Args: text
    Output: text
    """

    text=text.lower()
    text=re.sub('</?.*?>',' <>', text)
    text=re.sub('\\d|\\W+|_',' ',text)
    text=re.sub('[^a-zA-Z]'," ", text)

    return text
```

In [5]:
```
# Create stop words list
stop_words = stopwords.words('english')

def tokenize_remove_stopwords(txt):
    token_txt = word_tokenize(txt)
    no_stop_txt = [word for word in token_txt if word not in stop_words]
    no_stop_string = ' '.join(no_stop_txt)
    return(no_stop_string)
```

In [6]:
```
# since the size is humongus, I will take sample of the 2 categories.
# by trial, sample of 50000 from each category can be easily handled by my machine

size = 50000    # sample size
replace = True  # with replacement
fn = lambda obj: obj.loc[np.random.choice(obj.index, size, replace),:]

categories = con_df.groupby('cat', as_index=False).apply(fn)

# free up memory

del con_df

categories['txt'] = categories['txt'].apply(lambda x:clean_text(x))
#categories['txt'] = categories['txt'].apply(tokenize_remove_stopwords)
categories.reset_index(drop=True, inplace=True)

categories.cat = pd.Categorical(categories.cat)
categories['cat_codes'] = categories.cat.cat.codes

categories.head()
```

Out[6]:

	cat	txt	cat_codes
0	science_and_technology	which is good for privacy they store it for up...	0
1	science_and_technology	agreed it s not unheard of but it is still rar...	0
2	science_and_technology	the huawei honor is pretty good it has bands...	0
3	science_and_technology	gt we don t know this yet they struggle https...	0
4	science_and_technology	omfg	0

In [7]:
```
pd.concat([categories, pd.get_dummies(categories.cat)],axis = 1)
```

Out[7]:

	cat	txt	cat_codes	science_and_technology	sports	video_games
0	science_and_technology	which is good for privacy they store it for up...	0	1	0	0
1	science_and_technology	agreed it s not unheard of but it is still rar...	0	1	0	0
2	science_and_technology	the huawei honor is pretty good it has bands...	0	1	0	0
3	science_and_technology	gt we don t know this yet they struggle https...	0	1	0	0
4	science_and_technology	omfg	0	1	0	0
...	...	...	...	...	...	...
149995	video_games	good to know i ll keep that in mind for the ne...	2	0	0	1
149996	video_games	ds remakes	2	0	0	1
149997	video_games	invoker or slark	2	0	0	1
149998	video_games	it s mostly the hype generated by terry crews	2	0	0	1
149999	video_games	you have problems kid i love how you re acting...	2	0	0	1

150000 rows × 6 columns

In [8]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidfconverter = TfidfVectorizer(max_features=1500, stop_words=stop_words)
X = tfidfconverter.fit_transform(categories.txt).toarray()
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, categories.cat_codes, test_size=0.2, random_state=0)
```

In [10]:
```
classifier = MLPClassifier(hidden_layer_sizes=(150,30,15,15),random_state=1, max_iter=20, verbose=True, activation = '
relu', solver='adam')
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

```
Iteration 1, loss = 0.71525282
Iteration 2, loss = 0.60867110
Iteration 3, loss = 0.56798250
Iteration 4, loss = 0.51434957
Iteration 5, loss = 0.46800649
Iteration 6, loss = 0.42675279
Iteration 7, loss = 0.39082716
Iteration 8, loss = 0.35899659
Iteration 9, loss = 0.33014425
Iteration 10, loss = 0.30797898
Iteration 11, loss = 0.28739397
Iteration 12, loss = 0.27218620
Iteration 13, loss = 0.26108569
Iteration 14, loss = 0.25401585
Iteration 15, loss = 0.24630976
Iteration 16, loss = 0.24024005
Iteration 17, loss = 0.23679391
Iteration 18, loss = 0.23302999
Iteration 19, loss = 0.23129372
Iteration 20, loss = 0.22802770
```

In [11]:
```
print(classification_report(y_test, y_pred))
```

```
precision    recall  f1-score   support

           0       0.84      0.86      0.85      9887
           1       0.68      0.77      0.72      9987
           2       0.74      0.62      0.68     10126

    accuracy                           0.75     30000
   macro avg       0.75      0.75      0.75     30000
weighted avg       0.75      0.75      0.75     30000
```

In [12]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[8540  787  560]
 [ 620 7686 1681]
 [ 953 2859 6314]]
```

In [13]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.7513333333333333
```

Part 2: Neural Network Classifier with Keras

Using the multi-label classifier dataset from earlier exercises (categorized-comments.jsonl in the reddit folder), fit a neural network classifier using Keras. Use the code found in chapter 12 of the Applied Text Analysis with Python book as a guideline. Report the accuracy, precision, recall, F1-score, and confusion matrix.

In [14]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
import keras
import tensorflow as tf
```

```
Using TensorFlow backend.
```

In [15]:
```
input_dim = X_train.shape[1]  # Number of features

model = Sequential()
model.add(keras.layers.Dense(150, input_dim=input_dim, activation='relu'))
model.add(keras.layers.Dense(30, activation='sigmoid'))
model.add(keras.layers.Dense(15, activation='sigmoid'))
model.add(keras.layers.Dense(15, activation='sigmoid'))
model.add(keras.layers.Dense(3, activation='sigmoid'))
```

In [16]:
```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 150)               225150
_________________________________________________________________
dense_2 (Dense)              (None, 30)                4530
_________________________________________________________________
dense_3 (Dense)              (None, 15)                465
_________________________________________________________________
dense_4 (Dense)              (None, 15)                240
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 48
=================================================================
Total params: 230,433
Trainable params: 230,433
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
y_train_keras = keras.utils.to_categorical(y_train)
y_test_keras = keras.utils.to_categorical(y_test)
```

In [18]:
```
history = model.fit(X_train, y_train_keras,
                    epochs=20,
                    verbose=True,
                    validation_data=(X_test, y_test_keras),
                    batch_size=10)
```

```
Train on 120000 samples, validate on 30000 samples
Epoch 1/20
120000/120000 [==============================] - 523s 4ms/step - loss: 0.4182 - accuracy: 0.7983 - val_loss: 0.3763 -
val_accuracy: 0.8158
Epoch 2/20
120000/120000 [==============================] - 391s 3ms/step - loss: 0.3631 - accuracy: 0.8213 - val_loss: 0.3647 -
val_accuracy: 0.8196
Epoch 3/20
120000/120000 [==============================] - 438s 4ms/step - loss: 0.3393 - accuracy: 0.8326 - val_loss: 0.3538 -
val_accuracy: 0.8260
Epoch 4/20
120000/120000 [==============================] - 402s 3ms/step - loss: 0.3088 - accuracy: 0.8484 - val_loss: 0.3477 -
val_accuracy: 0.8324
Epoch 5/20
120000/120000 [==============================] - 504s 4ms/step - loss: 0.2773 - accuracy: 0.8647 - val_loss: 0.3525 -
val_accuracy: 0.8344
Epoch 6/20
120000/120000 [==============================] - 576s 5ms/step - loss: 0.2481 - accuracy: 0.8793 - val_loss: 0.3553 -
val_accuracy: 0.8393
Epoch 7/20
120000/120000 [==============================] - 516s 4ms/step - loss: 0.2244 - accuracy: 0.8911 - val_loss: 0.3764 -
val_accuracy: 0.8404
Epoch 8/20
120000/120000 [==============================] - 505s 4ms/step - loss: 0.2043 - accuracy: 0.9003 - val_loss: 0.3899 -
val_accuracy: 0.8408
Epoch 9/20
120000/120000 [==============================] - 500s 4ms/step - loss: 0.1882 - accuracy: 0.9078 - val_loss: 0.4191 -
val_accuracy: 0.8429
Epoch 10/20
120000/120000 [==============================] - 472s 4ms/step - loss: 0.1763 - accuracy: 0.9138 - val_loss: 0.4369 -
val_accuracy: 0.8430
Epoch 11/20
120000/120000 [==============================] - 538s 4ms/step - loss: 0.1667 - accuracy: 0.9179 - val_loss: 0.4674 -
val_accuracy: 0.8423
Epoch 12/20
120000/120000 [==============================] - 688s 6ms/step - loss: 0.1596 - accuracy: 0.9209 - val_loss: 0.4768 -
val_accuracy: 0.8438
Epoch 13/20
120000/120000 [==============================] - 519s 4ms/step - loss: 0.1538 - accuracy: 0.9238 - val_loss: 0.5165 -
val_accuracy: 0.8416
Epoch 14/20
120000/120000 [==============================] - 483s 4ms/step - loss: 0.1491 - accuracy: 0.9255 - val_loss: 0.5160 -
val_accuracy: 0.8424
Epoch 15/20
120000/120000 [==============================] - 385s 3ms/step - loss: 0.1446 - accuracy: 0.9275 - val_loss: 0.5419 -
val_accuracy: 0.8408
Epoch 16/20
120000/120000 [==============================] - 382s 3ms/step - loss: 0.1413 - accuracy: 0.9292 - val_loss: 0.5538 -
val_accuracy: 0.8431
Epoch 17/20
120000/120000 [==============================] - 379s 3ms/step - loss: 0.1392 - accuracy: 0.9301 - val_loss: 0.5546 -
val_accuracy: 0.8406
Epoch 18/20
120000/120000 [==============================] - 379s 3ms/step - loss: 0.1366 - accuracy: 0.9313 - val_loss: 0.5761 -
val_accuracy: 0.8411
Epoch 19/20
120000/120000 [==============================] - 414s 3ms/step - loss: 0.1346 - accuracy: 0.9321 - val_loss: 0.5876 -
val_accuracy: 0.8420
Epoch 20/20
120000/120000 [==============================] - 419s 3ms/step - loss: 0.1327 - accuracy: 0.9329 - val_loss: 0.5822 -
val_accuracy: 0.8429
```

In [19]:
```
loss, accuracy = model.evaluate(X_train, y_train_keras, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test_keras, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9361
Testing Accuracy:  0.8429
```

In [20]:
```
y_test_keras_results = tf.argmax(y_test_keras,axis = 1)
```

In [21]:
```
y_pred_keras_results = tf.argmax(model.predict(X_test, verbose = 1),axis = 1)
```

```
30000/30000 [==============================] - 4s 122us/step
```

In [22]:
```
print(classification_report(y_test_keras_results, y_pred_keras_results))
```

```
precision    recall  f1-score   support

           0       0.84      0.87      0.85      9887
           1       0.68      0.76      0.72      9987
           2       0.74      0.63      0.68     10126

    accuracy                           0.75     30000
   macro avg       0.75      0.75      0.75     30000
weighted avg       0.75      0.75      0.75     30000
```

In [23]:
```
print(confusion_matrix(y_test_keras_results, y_pred_keras_results))
```

```
[[8565  748  574]
 [ 658 7638 1691]
 [ 979 2795 6352]]
```

In [24]:
```
print(accuracy_score(y_test_keras_results, y_pred_keras_results))
```

```
0.7518333333333334
```

Part 3: Classifying Images

In chapter 20 of the Machine Learning with Python Cookbook, implement the code found in section 20.15 classify MSINT images using a convolutional neural network. Report the accuracy of your results.

In [25]:
```
import numpy as numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
```

In [26]:
```
# set that the color channel value will be first
K.set_image_data_format("channels_first")
```

In [27]:
```
# Set seed
np.random.seed(0)
```

In [28]:
```
# Set image information
channels = 1
height = 28
width = 28
```

In [29]:
```
# Load data and target from MNIST data
(data_train, target_train), (data_test, target_test) = mnist.load_data()
```

In [30]:
```
# Reshape training image data into features
data_train = data_train.reshape(data_train.shape[0],channels, height, width)
```

In [31]:
```
# Reshape test image data into features
data_test = data_test.reshape(data_test.shape[0], channels, height, width)
```

In [32]:
```
# Rescale pixel intensity to between 0 and 1
features_train = data_train / 255
features_test = data_test / 255
```

In [33]:
```
# One-hot encode target
target_train = np_utils.to_categorical(target_train)
target_test = np_utils.to_categorical(target_test)
number_of_classes = target_test.shape[1]
```

In [34]:
```
# Start neural network
network = Sequential()
```

In [35]:
```
# Add convolutional layer with 64 filters, a 5x5 window, and ReLU activation function
network.add(Conv2D(filters=64,
                   kernel_size=(5, 5),
                   input_shape=(channels, width, height),
                   activation='relu'))
```

In [36]:
```
# Add max pooling layer with a 2x2 window
network.add(MaxPooling2D(pool_size=(2, 2)))
```

In [37]:
```
# Add dropout layer
network.add(Dropout(0.5))
```

In [38]:
```
# Add layer to flatten input
network.add(Flatten())
```

In [39]:
```
# Add fully connected layer of 128 units with a ReLU activation function
network.add(Dense(128, activation="relu"))
```

In [40]:
```
# Add dropout layer
network.add(Dropout(0.5))
```

In [41]:
```
# Add fully connected layer with a softmax activation function
network.add(Dense(number_of_classes, activation="softmax"))
```

In [42]:
```
# Compile neural network
network.compile(loss="categorical_crossentropy", # Cross-entropy
                optimizer="rmsprop", # Root Mean Square Propagation
                metrics=["accuracy"]) # Accuracy performance metric
```

In [43]:
```
# Train neural network
network.fit(features_train, # Features
            target_train, # Target
            epochs=20, # Number of epochs
            verbose=1, # Print description after each epoch
            batch_size=1000, # Number of observations per batch
            validation_data=(features_test, target_test)) # Data for evaluation
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 293s 5ms/step - loss: 0.5912 - accuracy: 0.8175 - val_loss: 0.1895 - v
al_accuracy: 0.9445
Epoch 2/20
60000/60000 [==============================] - 280s 5ms/step - loss: 0.1926 - accuracy: 0.9431 - val_loss: 0.0917 - v
al_accuracy: 0.9722
Epoch 3/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.1254 - accuracy: 0.9632 - val_loss: 0.0627 - v
al_accuracy: 0.9800
Epoch 4/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0983 - accuracy: 0.9707 - val_loss: 0.0500 - v
al_accuracy: 0.9843
Epoch 5/20
60000/60000 [==============================] - 278s 5ms/step - loss: 0.0812 - accuracy: 0.9758 - val_loss: 0.0419 - v
al_accuracy: 0.9858
Epoch 6/20
60000/60000 [==============================] - 283s 5ms/step - loss: 0.0698 - accuracy: 0.9797 - val_loss: 0.0383 - v
al_accuracy: 0.9874
Epoch 7/20
60000/60000 [==============================] - 278s 5ms/step - loss: 0.0617 - accuracy: 0.9811 - val_loss: 0.0364 - v
al_accuracy: 0.9882
Epoch 8/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0578 - accuracy: 0.9829 - val_loss: 0.0333 - v
al_accuracy: 0.9886
Epoch 9/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0537 - accuracy: 0.9834 - val_loss: 0.0333 - v
al_accuracy: 0.9886
Epoch 10/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0498 - accuracy: 0.9844 - val_loss: 0.0311 - v
al_accuracy: 0.9894
Epoch 11/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0462 - accuracy: 0.9858 - val_loss: 0.0280 - v
al_accuracy: 0.9904
Epoch 12/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0437 - accuracy: 0.9864 - val_loss: 0.0290 - v
al_accuracy: 0.9903
Epoch 13/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0392 - accuracy: 0.9880 - val_loss: 0.0282 - v
al_accuracy: 0.9896
Epoch 14/20
60000/60000 [==============================] - 287s 5ms/step - loss: 0.0393 - accuracy: 0.9875 - val_loss: 0.0271 - v
al_accuracy: 0.9908
Epoch 15/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0379 - accuracy: 0.9884 - val_loss: 0.0278 - v
al_accuracy: 0.9908
Epoch 16/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0341 - accuracy: 0.9893 - val_loss: 0.0269 - v
al_accuracy: 0.9910
Epoch 17/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0337 - accuracy: 0.9893 - val_loss: 0.0281 - v
al_accuracy: 0.9908
Epoch 18/20
60000/60000 [==============================] - 278s 5ms/step - loss: 0.0327 - accuracy: 0.9898 - val_loss: 0.0252 - v
al_accuracy: 0.9917
Epoch 19/20
60000/60000 [==============================] - 277s 5ms/step - loss: 0.0313 - accuracy: 0.9904 - val_loss: 0.0273 - v
al_accuracy: 0.9910
Epoch 20/20
60000/60000 [==============================] - 276s 5ms/step - loss: 0.0300 - accuracy: 0.9907 - val_loss: 0.0266 - v
al_accuracy: 0.9906
```

Out[43]:
```
<keras.callbacks.callbacks.History at 0x1a28647c90>
```

In [44]:
```
val_loss, val_accuracy = network.evaluate(features_test, target_test)

print("The accuracy of the model, when compared against the testing data, is {:.4f}%".format(val_accuracy * 100))
```

```
10000/10000 [==============================] - 25s 3ms/step
The accuracy of the model, when compared against the testing data, is 99.0600%
```

In [ ]: