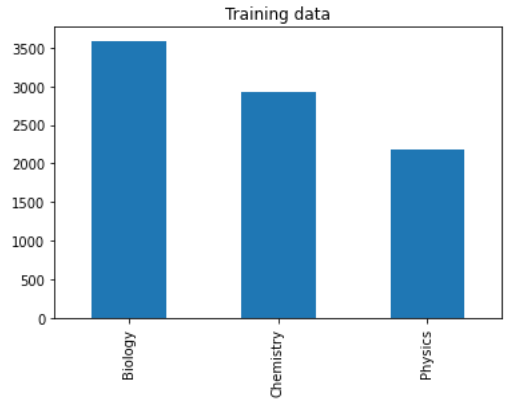
```
import tensorflow as tf
import csv
import pandas as pd
from tensorflow.keras import layers, models
```

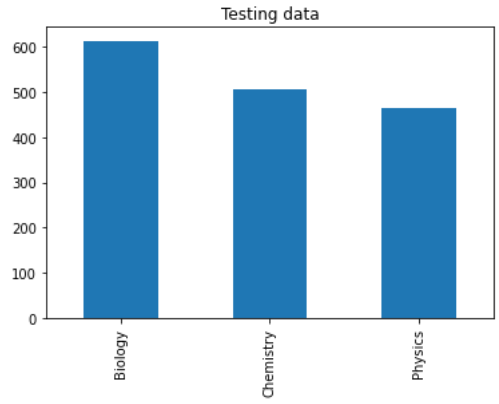
```
def read_csv(fname):
  with open(fname, 'r') as file:
    csvreader = csv.reader(file)
    rows = []
    for row in csvreader:
      rows += [row]
    return rows

df = pd.read_csv('train.csv')
train_data, train_labels = (df['Comment'], df['Topic'])
df['Topic'].value_counts().plot.bar(title="Training data")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f403428a070>
```




```
df = pd.read_csv('test.csv')
test_data, test_labels = (df['Comment'], df['Topic'])
df['Topic'].value_counts().plot.bar(title="Testing data")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f40341c20a0>
```



The dataset contains comments eith a topic of either Physics, Biology, or Chemistry. The model should be able to predict the topic of future comments.

```
print('Size of training and test data:', train_labels.shape, test_labels.shape)
```

```
Size of training and test data: (8695,) (1586,)
```

```
import numpy as np

from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizercount = CountVectorizer()
train_corpus_counts = vectorizercount.fit_transform(train_data).toarray()
test_corpus_counts = vectorizercount.transform(test_data).toarray()

def vectorize_sequences(sequences, dimension=8695):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

# Our vectorized training data
x_train = vectorize_sequences(train_corpus_counts)
# Our vectorized test data
x_test = vectorize_sequences(test_corpus_counts)


def encode_label(label):
    if(label == 'Biology'):
      return 0
    elif(label == 'Chemistry'):
      return 1
    elif(label == 'Physics'):
      return 2

# Our vectorized labels
y_train = np.asarray([encode_label(label) for label in train_labels]).astype('float32')
y_test = np.asarray([encode_label(label) for label in test_labels]).astype('float32')

# build the model
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(8695,)))
# model.add(layers.Dropout(0.2))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

# compile
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# create a validation set
x_val = x_train[:2000]
partial_x_train = x_train[2000:]

y_val = y_train[:2000]
partial_y_train = y_train[2000:]

# train
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

# use sklearn evaluation

from sklearn.metrics import classification_report

pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))

# plot the training and validation loss
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
# plot the training and validation accuracy

plt.clf()   # clear

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
Epoch 1/20
14/14 [==============================] - 3s 35ms/step - loss: 0.0000e+00 - accuracy: 0.4133 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 2/20
14/14 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 3/20
14/14 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 4/20
14/14 [==============================] - 0s 16ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 5/20
14/14 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 6/20
14/14 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 7/20
14/14 [==============================] - 0s 14ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 8/20
14/14 [==============================] - 0s 17ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 9/20
14/14 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 10/20
14/14 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 11/20
14/14 [==============================] - 0s 17ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 12/20
14/14 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 13/20
14/14 [==============================] - 0s 16ms/step - loss: 0.0000e+00 - accuracy: 0.4157 - val_loss: 0.0000e+00 - val_accuracy:
Epoch 14/20
```

```
# build a Sequential model with SimpleRNN layers
max_features = 10000
maxlen = 500
batch_size = 32
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='sigmoid'))

# compile
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])


# train
print(train_corpus_counts)
print( np.array([encode_label(label) for label in train_labels]))
history = model.fit(train_corpus_counts,
                    np.array([encode_label(label) for label in train_labels]),
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[0 2 0 ... 1 0 0]
Epoch 1/10
55/55 [==============================] - 2120s 38s/step - loss: 0.0000e+00 - accuracy: 0.4112 - val_loss: 0.0000e+00 - val_accuracy: 0.
Epoch 2/10
55/55 [==============================] - 2185s 40s/step - loss: 0.0000e+00 - accuracy: 0.4114 - val_loss: 0.0000e+00 - val_accuracy: 0.
Epoch 3/10
55/55 [==============================] - 2195s 40s/step - loss: 0.0000e+00 - accuracy: 0.4114 - val_loss: 0.0000e+00 - val_accuracy: 0.
Epoch 4/10
55/55 [==============================] - 2230s 41s/step - loss: 0.0000e+00 - accuracy: 0.4114 - val_loss: 0.0000e+00 - val_accuracy: 0.
Epoch 5/10
55/55 [==============================] - 2191s 40s/step - loss: 0.0000e+00 - accuracy: 0.4114 - val_loss: 0.0000e+00 - val_accuracy: 0.
Epoch 6/10
55/55 [==============================] - 2192s 40s/step - loss: 0.0000e+00 - accuracy: 0.4114 - val_loss: 0.0000e+00 - val_accuracy: 0.
Epoch 7/10
12/55 [=====>.....................] - ETA: 28:23 - loss: 0.0000e+00 - accuracy: 0.3984
```

Google Colab keeps recycling the instance before this can finish, it takes 5 hours to run, and it's 7:30. Here is the previous output for a successful run.

```
history = model.fit(train_corpus_counts,
                    np.array([encode_label(label) for label in train_labels]),
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[0 2 0 ... 1 0 0]
Epoch 1/10
55/55 [==============================] - 1864s 34s/step - loss: 0.5407 - accuracy: 0.3417 - val_loss: 0.4651 - val_accuracy: 0.3283
Epoch 2/10
55/55 [==============================] - 1833s 33s/step - loss: 0.4440 - accuracy: 0.3377 - val_loss: 0.4541 - val_accuracy: 0.3283
Epoch 3/10
55/55 [==============================] - 1901s 35s/step - loss: 0.4422 - accuracy: 0.3377 - val_loss: 0.4561 - val_accuracy: 0.3283
Epoch 4/10
55/55 [==============================] - 1917s 35s/step - loss: 0.4436 - accuracy: 0.3377 - val_loss: 0.4517 - val_accuracy: 0.3283
Epoch 5/10
55/55 [==============================] - 1933s 35s/step - loss: 0.4435 - accuracy: 0.3377 - val_loss: 0.4507 - val_accuracy: 0.3283
Epoch 6/10
55/55 [==============================] - 1941s 35s/step - loss: 0.4428 - accuracy: 0.3377 - val_loss: 0.4516 - val_accuracy: 0.3283
Epoch 7/10
55/55 [==============================] - 1909s 35s/step - loss: 0.4434 - accuracy: 0.3377 - val_loss: 0.4549 - val_accuracy: 0.3283
Epoch 8/10
55/55 [==============================] - 1954s 36s/step - loss: 0.4414 - accuracy: 0.3377 - val_loss: 0.4506 - val_accuracy: 0.3283
Epoch 9/10
55/55 [==============================] - 1936s 35s/step - loss: 0.4423 - accuracy: 0.3377 - val_loss: 0.4510 - val_accuracy: 0.3283
Epoch 10/10
55/55 [==============================] - 1914s 35s/step - loss: 0.4424 - accuracy: 0.3377 - val_loss: 0.4513 - val_accuracy: 0.3283
```

```
# use sklearn evaluation

from sklearn.metrics import classification_report

pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
50/50 [==============================] - 20s 404ms/step
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00       614
         1.0       0.32      1.00      0.48       506
         2.0       0.00      0.00      0.00       466

    accuracy                           0.32      1586
   macro avg       0.11      0.33      0.16      1586
weighted avg       0.10      0.32      0.15      1586

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from tensorflow import keras
from tensorflow.keras import layers

vectorizer = TextVectorization(max_tokens=20000, output_sequence_length=200)
vectorizer.adapt(train_data)
vectorizer.adapt(test_data)

voc = vectorizercount.vocabulary_
word_ind = dict(zip(voc, range(len(voc))))

embedding_layer = layers.Embedding(len(word_ind) + 1,
                           128,
                           input_length=200)
model2 = models.Sequential()
model2.add(embedding_layer)
model2.add(layers.Conv1D(128, 5, activation='relu'))
model2.add(layers.MaxPooling1D(5))
model2.add(layers.Conv1D(128, 5, activation='relu'))
model2.add(layers.GlobalMaxPooling1D())
model2.add(layers.Dense(128, activation="relu"))
```

```
model2.add(layers.Dropout(0.5))
model2.add(layers.Dense(3, activation = 'softmax'))
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 128)          2326784

 conv1d (Conv1D)             (None, 196, 128)          82048

 max_pooling1d (MaxPooling1D  (None, 39, 128)          0
 )

 conv1d_1 (Conv1D)           (None, 35, 128)           82048

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense_3 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_4 (Dense)             (None, 3)                 387

=================================================================
Total params: 2,507,779
Trainable params: 2,507,779
Non-trainable params: 0
_________________________________________________________________
```

```
import numpy as np

x_train2 = vectorizer(np.array([[s] for s in train_data])).numpy()
x_test2 = vectorizer(np.array([[s] for s in test_data])).numpy()
```

```
model2.compile(
    loss="sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["accuracy"]
)
model2.fit(x_train2, y_train, batch_size=128, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
55/55 [==============================] - 7s 17ms/step - loss: 1.0781 - accuracy: 0.4073 - val_loss: 1.0632 - val_accuracy: 0.4457
Epoch 2/10
55/55 [==============================] - 1s 11ms/step - loss: 0.9374 - accuracy: 0.5405 - val_loss: 0.9385 - val_accuracy: 0.5175
Epoch 3/10
55/55 [==============================] - 1s 11ms/step - loss: 0.6853 - accuracy: 0.7037 - val_loss: 0.8514 - val_accuracy: 0.6308
Epoch 4/10
55/55 [==============================] - 1s 10ms/step - loss: 0.4857 - accuracy: 0.8055 - val_loss: 0.7966 - val_accuracy: 0.6791
Epoch 5/10
55/55 [==============================] - 1s 11ms/step - loss: 0.3476 - accuracy: 0.8692 - val_loss: 0.8281 - val_accuracy: 0.6987
Epoch 6/10
55/55 [==============================] - 1s 10ms/step - loss: 0.2538 - accuracy: 0.9070 - val_loss: 0.9078 - val_accuracy: 0.6987
Epoch 7/10
55/55 [==============================] - 1s 11ms/step - loss: 0.1783 - accuracy: 0.9340 - val_loss: 1.1239 - val_accuracy: 0.6757
Epoch 8/10
55/55 [==============================] - 1s 10ms/step - loss: 0.1343 - accuracy: 0.9480 - val_loss: 1.3635 - val_accuracy: 0.6814
Epoch 9/10
55/55 [==============================] - 1s 11ms/step - loss: 0.0961 - accuracy: 0.9625 - val_loss: 1.7185 - val_accuracy: 0.6768
Epoch 10/10
55/55 [==============================] - 1s 11ms/step - loss: 0.0837 - accuracy: 0.9662 - val_loss: 1.6042 - val_accuracy: 0.6849
<keras.callbacks.History at 0x7f401b6dcfa0>
```

```
pred2 = model2.predict(x_test2)
pred2 = [np.argmax(p) for p in pred2]
print(classification_report(y_test, pred2))
```

```
50/50 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

         0.0       0.85      0.80      0.82       614
         1.0       0.79      0.76      0.77       506
         2.0       0.76      0.85      0.80       466

    accuracy                           0.80      1586
   macro avg       0.80      0.80      0.80      1586
weighted avg       0.80      0.80      0.80      1586
```

Sequential Model: The sequential model had a low test accuracy of 0.39. While not the worst result, an accuracy of 0.39 is only slightly better than simple random choice and does not qualify as a solution to the problem.

RNN: The recurrent neural network had the lowest accuracy of 0.32. Since the model was categorizing inputs into three classes (Physics, Biology, and Chemistry), an accuracy of 0.32 is slightly less than expected for simple random choice. Because of this, the training can be considered a failure.

Embedding Layer in a CNN: Compared to the other models, the model with a CNN and Embedding layer performed way better. It gained a test accuracy of 0.8 and when training the epochs it gained its highest train accuracy as 0.96 and its highest validation accuracy as 0.7. This shows that adding an embedding layer helps the model perform better and that a CNN might work better in this situation than a RNN model or regular sequential model.

✓ 0s completed at 8:50 PM