# Text Classification 2

The data set contained an assortment of tweets associated with an MBTI personality type. After running the dense sequential model a few times through the initial data set, I realized that the data set as it was would not work well with the way the sequential model works. So, the data set was edited so that the tweets were associated with introvert/extrovert tags based off the initial MBTI personality types. Then, the dense sequential model was run again, with far better accuracy as a result. The model should be able to predict the introversion/extroversion of a person based off of the training data of tweets given.

## Sequential Model

In [3]:

```
# some necessary packages
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models

from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd

# set seed for reproducibility
np.random.seed(1234)

df = pd.read_csv('twitter_MBTI_ edited2.csv', header=0, usecols=[0,1], encodin
print('rows and columns:', df.shape)
print(df.head())
```

```
rows and columns: (7811, 2)
                                               tweet mbti
0  @Pericles216 @HierBeforeTheAC @Sachinettiyil T...    i
1  @Hispanthicckk Being you makes you look cute||...    i
2  @Alshymi Les balles sont rÃ©elles et sont tirÃ...    i
3  I'm like entp but idiotic|||Hey boy, do you wa...    i
4  @kaeshurr1 Give it to @ZargarShanif ... He has...    i
```

In [4]:

```
# split df into train and test
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (6238, 2)
test data size:  (1573, 2)
```

In [5]:
```
# set up X and Y
num_labels = 2
vocab_size = 25000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.tweet)

x_train = tokenizer.texts_to_matrix(train.tweet, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.tweet, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.mbti)
y_train = encoder.transform(train.mbti)
y_test = encoder.transform(test.mbti)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (6238, 25000) (6238,)
test shapes: (1573, 25000) (1573,)
test first five labels: [1 1 1 1 1]
```

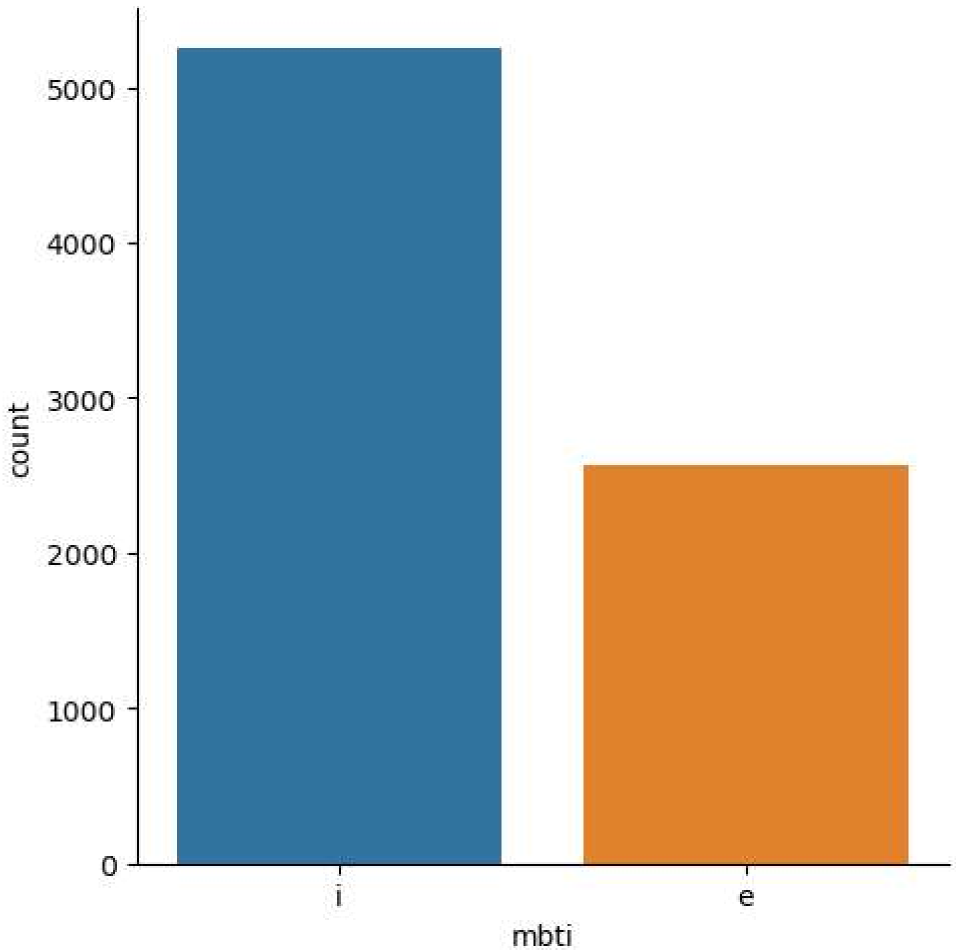
In [14]:
```
import seaborn as sb

# Load data
twitter = df = pd.read_csv('twitter_MBTI_ edited2.csv')
X = twitter.tweet
y = twitter.mbti

# convert to data frames
df = pd.DataFrame(X, columns=twitter.tweet)
df_y = pd.DataFrame(y, columns=['mbti'])

sb.catplot(x="mbti", kind='count', data=df_y)
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x20e5223e730>



In [6]:
```
# fit model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal',
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/30
57/57 [==============================] - 6s 44ms/step - loss: 0.6495 - accura
cy: 0.6575 - val_loss: 0.6137 - val_accuracy: 0.7099
Epoch 2/30
57/57 [==============================] - 2s 40ms/step - loss: 0.3764 - accura
cy: 0.8497 - val_loss: 0.6682 - val_accuracy: 0.6667
Epoch 3/30
57/57 [==============================] - 1s 21ms/step - loss: 0.1046 - accura
cy: 0.9806 - val_loss: 0.7887 - val_accuracy: 0.6426
Epoch 4/30
57/57 [==============================] - 1s 21ms/step - loss: 0.0243 - accura
cy: 0.9998 - val_loss: 0.9599 - val_accuracy: 0.6394
Epoch 5/30
57/57 [==============================] - 1s 22ms/step - loss: 0.0090 - accura
cy: 1.0000 - val_loss: 1.0223 - val_accuracy: 0.6346
Epoch 6/30
57/57 [==============================] - 2s 27ms/step - loss: 0.0049 - accura
cy: 1.0000 - val_loss: 1.0837 - val_accuracy: 0.6426
Epoch 7/30
57/57 [==============================] - 2s 27ms/step - loss: 0.0033 - accura
cy: 1.0000 - val_loss: 1.1242 - val_accuracy: 0.6410
Epoch 8/30
57/57 [==============================] - 1s 24ms/step - loss: 0.0024 - accura
cy: 1.0000 - val_loss: 1.1512 - val_accuracy: 0.6442
Epoch 9/30
57/57 [==============================] - 1s 22ms/step - loss: 0.0018 - accura
cy: 1.0000 - val_loss: 1.1783 - val_accuracy: 0.6410
Epoch 10/30
57/57 [==============================] - 1s 22ms/step - loss: 0.0014 - accura
cy: 1.0000 - val_loss: 1.2003 - val_accuracy: 0.6410
Epoch 11/30
57/57 [==============================] - 1s 22ms/step - loss: 0.0011 - accura
cy: 1.0000 - val_loss: 1.2261 - val_accuracy: 0.6394
Epoch 12/30
57/57 [==============================] - 1s 23ms/step - loss: 9.4531e-04 - ac
curacy: 1.0000 - val_loss: 1.2494 - val_accuracy: 0.6362
Epoch 13/30
57/57 [==============================] - 1s 23ms/step - loss: 7.9741e-04 - ac
curacy: 1.0000 - val_loss: 1.2696 - val_accuracy: 0.6314
Epoch 14/30
57/57 [==============================] - 2s 28ms/step - loss: 6.8304e-04 - ac
curacy: 1.0000 - val_loss: 1.2856 - val_accuracy: 0.6378
Epoch 15/30
57/57 [==============================] - 1s 24ms/step - loss: 5.9211e-04 - ac
curacy: 1.0000 - val_loss: 1.2995 - val_accuracy: 0.6346
Epoch 16/30
57/57 [==============================] - 1s 23ms/step - loss: 5.1754e-04 - ac
curacy: 1.0000 - val_loss: 1.3168 - val_accuracy: 0.6394
Epoch 17/30
57/57 [==============================] - 1s 24ms/step - loss: 4.5776e-04 - ac
curacy: 1.0000 - val_loss: 1.3310 - val_accuracy: 0.6394
Epoch 18/30
57/57 [==============================] - 1s 23ms/step - loss: 4.0778e-04 - ac
curacy: 1.0000 - val_loss: 1.3438 - val_accuracy: 0.6362
Epoch 19/30
57/57 [==============================] - 1s 24ms/step - loss: 3.6634e-04 - ac
curacy: 1.0000 - val_loss: 1.3598 - val_accuracy: 0.6410
```

```
Epoch 20/30
57/57 [==============================] - 1s 23ms/step - loss: 3.2929e-04 - ac
curacy: 1.0000 - val_loss: 1.3715 - val_accuracy: 0.6378
Epoch 21/30
57/57 [==============================] - 1s 22ms/step - loss: 2.9865e-04 - ac
curacy: 1.0000 - val_loss: 1.3826 - val_accuracy: 0.6394
Epoch 22/30
57/57 [==============================] - 1s 23ms/step - loss: 2.7228e-04 - ac
curacy: 1.0000 - val_loss: 1.3953 - val_accuracy: 0.6394
Epoch 23/30
57/57 [==============================] - 1s 23ms/step - loss: 2.4958e-04 - ac
curacy: 1.0000 - val_loss: 1.4092 - val_accuracy: 0.6362
Epoch 24/30
57/57 [==============================] - 1s 23ms/step - loss: 2.2932e-04 - ac
curacy: 1.0000 - val_loss: 1.4208 - val_accuracy: 0.6378
Epoch 25/30
57/57 [==============================] - 1s 23ms/step - loss: 2.1156e-04 - ac
curacy: 1.0000 - val_loss: 1.4310 - val_accuracy: 0.6394
Epoch 26/30
57/57 [==============================] - 1s 23ms/step - loss: 1.9628e-04 - ac
curacy: 1.0000 - val_loss: 1.4414 - val_accuracy: 0.6394
Epoch 27/30
57/57 [==============================] - 1s 23ms/step - loss: 1.8192e-04 - ac
curacy: 1.0000 - val_loss: 1.4518 - val_accuracy: 0.6378
Epoch 28/30
57/57 [==============================] - 1s 24ms/step - loss: 1.6945e-04 - ac
curacy: 1.0000 - val_loss: 1.4638 - val_accuracy: 0.6410
Epoch 29/30
57/57 [==============================] - 1s 24ms/step - loss: 1.5817e-04 - ac
curacy: 1.0000 - val_loss: 1.4731 - val_accuracy: 0.6378
Epoch 30/30
57/57 [==============================] - 1s 23ms/step - loss: 1.4796e-04 - ac
curacy: 1.0000 - val_loss: 1.4820 - val_accuracy: 0.6378
```

In [7]:

```
# evaluate
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
16/16 [==============================] - 1s 3ms/step - loss: 1.1928 - accurac
y: 0.7076
Accuracy:  0.707565188407898
```

In [8]:

```
print(score)
```

```
[1.192842960357666, 0.707565188407898]
```

In [9]:
```
# get predictions so we can calculate more metrics
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
pred[:10]
```

```
50/50 [==============================] - 2s 3ms/step
```

Out[9]:
```
array([[0.82517004],
       [0.9127619 ],
       [0.2753336 ],
       [0.9999835 ],
       [0.977856  ],
       [0.07948193],
       [0.7949867 ],
       [0.9999486 ],
       [0.99988973],
       [0.99999535]], dtype=float32)
```

In [10]:
```
pred_labels[:10]
```

Out[10]:
```
[1, 1, 0, 1, 1, 0, 1, 1, 1, 1]
```

In [11]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
print('accuracy score: ', accuracy_score(y_test, pred_labels))
print('precision score: ', precision_score(y_test, pred_labels))
print('recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
accuracy score:  0.7075651621106167
precision score:  0.7398042414355628
recall score:  0.8654580152671756
f1 score:  0.7977132805628848
```

# RNN Model

In [19]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing

max_features = 10000
maxlen = 500
batch_size = 32

# pad the data to maxlen
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

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='binary_crossentropy',
              metrics=['accuracy'])


history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
39/39 [==============================] - 9s 163ms/step - loss: 0.6374 - accur
acy: 0.6637 - val_loss: 0.6236 - val_accuracy: 0.6843
Epoch 2/10
39/39 [==============================] - 6s 163ms/step - loss: 0.6345 - accur
acy: 0.6705 - val_loss: 0.6237 - val_accuracy: 0.6843
Epoch 3/10
39/39 [==============================] - 5s 138ms/step - loss: 0.6346 - accur
acy: 0.6705 - val_loss: 0.6249 - val_accuracy: 0.6843
Epoch 4/10
39/39 [==============================] - 5s 132ms/step - loss: 0.6349 - accur
acy: 0.6705 - val_loss: 0.6239 - val_accuracy: 0.6843
Epoch 5/10
39/39 [==============================] - 5s 133ms/step - loss: 0.6344 - accur
acy: 0.6705 - val_loss: 0.6240 - val_accuracy: 0.6843
Epoch 6/10
39/39 [==============================] - 6s 143ms/step - loss: 0.6343 - accur
acy: 0.6705 - val_loss: 0.6235 - val_accuracy: 0.6843
Epoch 7/10
39/39 [==============================] - 5s 138ms/step - loss: 0.6343 - accur
acy: 0.6705 - val_loss: 0.6232 - val_accuracy: 0.6843
Epoch 8/10
39/39 [==============================] - 5s 139ms/step - loss: 0.6373 - accur
acy: 0.6707 - val_loss: 0.6239 - val_accuracy: 0.6843
Epoch 9/10
39/39 [==============================] - 6s 151ms/step - loss: 0.6335 - accur
acy: 0.6705 - val_loss: 0.6246 - val_accuracy: 0.6843
Epoch 10/10
39/39 [==============================] - 6s 148ms/step - loss: 0.6342 - accur
acy: 0.6705 - val_loss: 0.6230 - val_accuracy: 0.6843
```

In [30]:

```
from sklearn.metrics import classification_report

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

```
50/50 [==============================] - 2s 1ms/step
              precision    recall  f1-score   support

           0       0.61      0.04      0.07       525
           1       0.67      0.99      0.80      1048

    accuracy                           0.67      1573
   macro avg       0.64      0.51      0.43      1573
weighted avg       0.65      0.67      0.56      1573
```

## LTSM RNN

In [33]:
```
# Attempt LTSM RNN
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing

max_features = 10000
maxlen = 500
batch_size = 32

# pad the data to maxlen
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

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='binary_crossentropy',
              metrics=['accuracy'])


history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
39/39 [==============================] - 10s 166ms/step - loss: 0.6367 - accu
racy: 0.6705 - val_loss: 0.6235 - val_accuracy: 0.6843
Epoch 2/10
39/39 [==============================] - 6s 162ms/step - loss: 0.6345 - accur
acy: 0.6705 - val_loss: 0.6246 - val_accuracy: 0.6843
Epoch 3/10
39/39 [==============================] - 5s 135ms/step - loss: 0.6346 - accur
acy: 0.6705 - val_loss: 0.6235 - val_accuracy: 0.6843
Epoch 4/10
39/39 [==============================] - 5s 139ms/step - loss: 0.6346 - accur
acy: 0.6705 - val_loss: 0.6289 - val_accuracy: 0.6843
Epoch 5/10
39/39 [==============================] - 5s 134ms/step - loss: 0.6345 - accur
acy: 0.6705 - val_loss: 0.6246 - val_accuracy: 0.6843
Epoch 6/10
39/39 [==============================] - 6s 146ms/step - loss: 0.6345 - accur
acy: 0.6705 - val_loss: 0.6240 - val_accuracy: 0.6843
Epoch 7/10
39/39 [==============================] - 5s 137ms/step - loss: 0.6354 - accur
acy: 0.6707 - val_loss: 0.6239 - val_accuracy: 0.6843
Epoch 8/10
39/39 [==============================] - 5s 137ms/step - loss: 0.6336 - accur
acy: 0.6705 - val_loss: 0.6278 - val_accuracy: 0.6843
Epoch 9/10
39/39 [==============================] - 5s 138ms/step - loss: 0.6340 - accur
acy: 0.6705 - val_loss: 0.6236 - val_accuracy: 0.6843
Epoch 10/10
39/39 [==============================] - 6s 149ms/step - loss: 0.6339 - accur
acy: 0.6705 - val_loss: 0.6233 - val_accuracy: 0.6843
```

In [36]:
```
pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
50/50 [==============================] - 2s 22ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       525
           1       0.67      1.00      0.80      1048

    accuracy                           0.67      1573
   macro avg       0.33      0.50      0.40      1573
weighted avg       0.44      0.67      0.53      1573

C:\Users\paopu\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\paopu\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\paopu\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

## GRU RNN

In [37]:
```
# Attempt GRU RNN
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing

max_features = 10000
maxlen = 500
batch_size = 32

# pad the data to maxlen
train_data = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.GRU(32))
model.add(layers.Dense(1, activation='sigmoid'))

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


history = model.fit(train_data,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
39/39 [==============================] - 18s 341ms/step - loss: 0.6431 - accu
racy: 0.6705 - val_loss: 0.6240 - val_accuracy: 0.6843
Epoch 2/10
39/39 [==============================] - 13s 325ms/step - loss: 0.6343 - accu
racy: 0.6705 - val_loss: 0.6241 - val_accuracy: 0.6843
Epoch 3/10
39/39 [==============================] - 13s 340ms/step - loss: 0.6343 - accu
racy: 0.6705 - val_loss: 0.6245 - val_accuracy: 0.6843
Epoch 4/10
39/39 [==============================] - 14s 365ms/step - loss: 0.6345 - accu
racy: 0.6705 - val_loss: 0.6253 - val_accuracy: 0.6843
Epoch 5/10
39/39 [==============================] - 14s 351ms/step - loss: 0.6343 - accu
racy: 0.6705 - val_loss: 0.6251 - val_accuracy: 0.6843
Epoch 6/10
39/39 [==============================] - 15s 390ms/step - loss: 0.6347 - accu
racy: 0.6705 - val_loss: 0.6239 - val_accuracy: 0.6843
Epoch 7/10
39/39 [==============================] - 15s 374ms/step - loss: 0.6342 - accu
racy: 0.6705 - val_loss: 0.6238 - val_accuracy: 0.6843
Epoch 8/10
39/39 [==============================] - 15s 374ms/step - loss: 0.6344 - accu
racy: 0.6705 - val_loss: 0.6237 - val_accuracy: 0.6843
Epoch 9/10
39/39 [==============================] - 14s 362ms/step - loss: 0.6342 - accu
racy: 0.6705 - val_loss: 0.6254 - val_accuracy: 0.6843
Epoch 10/10
39/39 [==============================] - 14s 361ms/step - loss: 0.6339 - accu
racy: 0.6705 - val_loss: 0.6237 - val_accuracy: 0.6843
```

In [38]:
```
pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
50/50 [==============================] - 5s 47ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       525
           1       0.67      1.00      0.80      1048

    accuracy                           0.67      1573
   macro avg       0.33      0.50      0.40      1573
weighted avg       0.44      0.67      0.53      1573


C:\Users\paopu\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\paopu\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\paopu\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

# Embeddings

In [32]:
```
# set up the Embedding layer in a Sequential model

model = models.Sequential()
model.add(layers.Embedding(max_features, 8, input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc']
model.summary()

history = model.fit(train_data, y_train, epochs=10, batch_size=32, validation_
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 500, 8)            80000

 flatten_2 (Flatten)         (None, 4000)              0

 dense_8 (Dense)             (None, 16)                64016

 dense_9 (Dense)             (None, 1)                 17

=================================================================
Total params: 144,033
Trainable params: 144,033
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
156/156 [==============================] - 7s 25ms/step - loss: 0.6385 - acc:
0.6705 - val_loss: 0.6323 - val_acc: 0.6843
Epoch 2/10
156/156 [==============================] - 3s 18ms/step - loss: 0.6376 - acc:
0.6705 - val_loss: 0.6236 - val_acc: 0.6843
Epoch 3/10
156/156 [==============================] - 1s 9ms/step - loss: 0.6352 - acc:
0.6705 - val_loss: 0.6296 - val_acc: 0.6843
Epoch 4/10
156/156 [==============================] - 1s 9ms/step - loss: 0.6349 - acc:
0.6705 - val_loss: 0.6241 - val_acc: 0.6843
Epoch 5/10
156/156 [==============================] - 1s 9ms/step - loss: 0.6317 - acc:
0.6705 - val_loss: 0.6251 - val_acc: 0.6843
Epoch 6/10
156/156 [==============================] - 1s 9ms/step - loss: 0.6299 - acc:
0.6711 - val_loss: 0.6258 - val_acc: 0.6843
Epoch 7/10
156/156 [==============================] - 2s 11ms/step - loss: 0.6261 - acc:
0.6707 - val_loss: 0.6231 - val_acc: 0.6843
Epoch 8/10
156/156 [==============================] - 2s 10ms/step - loss: 0.6216 - acc:
0.6745 - val_loss: 0.6284 - val_acc: 0.6811
Epoch 9/10
156/156 [==============================] - 2s 13ms/step - loss: 0.6181 - acc:
0.6774 - val_loss: 0.6254 - val_acc: 0.6811
Epoch 10/10
156/156 [==============================] - 2s 11ms/step - loss: 0.6155 - acc:
0.6806 - val_loss: 0.6308 - val_acc: 0.6779
```

# Analysis

Initially, going into this assignment, I thought working with a non-binary data set would be sufficient for the model. I was proven wrong when I tried and tested it out, and came up with extremely low accuracy results. I thought that this was probably an indication of me doing it wrong, and I was right. After editing the dataset and sorting by introvert vs extroverts, I was able to get much better results with the model accuracy results. After this, I messed around here and there in small ways to improve the results of all the models, leading to the final results seen.

Overall, the dense sequential model clearly did the best, with an accuracy score of 70%. This is significantly better than the 1% I had been getting previously, and editing the dataset in general did wonders. Then, afterwards I tried the simple RNN approach. This wasn't as good as the dense sequential model, but it also didn't lag significantly behind, coming up at 67% accuracy-- only 3% behind the dense sequential model. After testing out other RMS techniques, they all ended up being fairly similar to eachother, just being a little bit worse than the simple RNN model. They differered primarily in their precision and recall scores, as the accuracy remained at 67% regardless of the RNN technique used. Then, embeddings were attempted. This resulted in accuracy coming up a percentage higher to 68%-- the only one closest to the dense sequential model by far. Embeddings seem to provide a slight boost to the results overall, but the dense sequential model still reigns supreme in the end.