Can we Predict Volatility in Stock Prices Using SEC filings?

Team Members : Apurva, Harsh, Jaskaran, Suchita, Vidhi

Machine Learning

1. Load Libraries

In [ ]:
```
# mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
# import required libraries
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MultiLabelBinarizer

# Deep Learning Libraries
import tensorflow as tf
import keras
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense, Dropout, GRU, Input, Embedding, Bidirectional
from tensorflow.keras.layers import Flatten, Conv1D, MaxPooling1D, GlobalMaxPooling1D, Bat
chNormalization
from tensorflow.keras.layers import concatenate
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import backend as K
from tensorflow.keras.metrics import AUC
from tensorflow.keras.utils import multi_gpu_model

config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True

#!pip install keras==2.3.0
#from tensorflow.python.keras.layers import CuDNNGRU

#sess_config.gpu_options.allow_growth = True
#from keras.backend.tensorflow_backend import set_session
#import tensorflow.compat.v1 as tf
#tf.disable_v2_behavior()
```

In [ ]:
```
# check the cpu and gpu being used
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 11758833143096406305
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 9718225536536755501
physical_device_desc: "device: XLA_CPU device"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 4972091611459687244
physical_device_desc: "device: XLA_GPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 15473775744
locality {
  bus_id: 1
  links {
  }
}
incarnation: 10441743454703665301
physical_device_desc: "device: 0, name: Tesla V100-SXM2-16GB, pci bus id: 0000:00:04.0, compute capability: 7.0"
]
```

2. Preprocess dataframe

In [ ]:
```
# read data
df = pd.read_csv("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/lemmatized_text.csv")
```

In [ ]:
```
df.release_date.value_counts()
```

```
2017-07-27 07:37:05    2
2017-10-02 07:38:03    1
2016-06-15 16:05:32    1
2016-08-04 06:24:56    1
2018-02-28 10:34:51    1
                      ..
2016-08-03 16:16:09    1
2015-04-20 21:44:20    1
2014-11-20 16:01:31    1
2016-01-13 11:28:06    1
2016-04-26 07:05:39    1
Name: release_date, Length: 1650, dtype: int64
```

In [ ]:
```
mlb = MultiLabelBinarizer()
df = df.join(pd.DataFrame(mlb.fit_transform(df.pop('items')), columns=mlb.classes_,), sort=False, how="left")
```

In [ ]:
```
df.isna().sum()
```

```
Unnamed: 0            0
Unnamed: 0.1          0
Unnamed: 0.1.1        0
ticker                0
cik                   0
doc_name              0
txt_link              0
GICS Sector           0
GICS Sub Industry     0
text                  0
release_date          0
price_change         42
vix                   1
rm_week              39
rm_month             34
rm_qtr               41
rm_year              56
processed_text        0
text_len              0
signal                0
                      0
'                     0
,                     0
.                     0
0                     0
1                     0
2                     0
3                     0
4                     0
5                     0
6                     0
7                     0
8                     0
9                     0
I                     0
[                     0
]                     0
e                     0
m                     0
t                     0
dtype: int64
```

In [ ]:
```
# get rid of rows that has missing values
df.dropna(subset=['vix', 'rm_week', 'rm_month', 'rm_qtr', 'rm_year'], inplace=True)
```

In [ ]:

```
#### Define number of words, and embedding dimensions
max_words = 34603
embed_dim = 100

def load_embeddings(vec_file):
    print("Loading Glove Model")
    f = open(vec_file, 'r')
    model = {}
    for line in f:
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print("Done. {} words loaded!".format(len(model)))
    return model

def tokenize_and_pad(docs, max_words=max_words):
    global t
    t = Tokenizer()
    t.fit_on_texts(docs)
    docs = pad_sequences(sequences = t.texts_to_sequences(docs), maxlen = max_words, paddi
ng = 'post')
    global vocab_size
    vocab_size = len(t.word_index) + 1

    return docs

def oversample(X, docs, y):
    # Get number of rows with imbalanced class
    target = y.sum().idxmax()
    n = y[target].sum()
    # identify imbalanced targets
    imbalanced = y.drop(target, axis=1)
    #For each target, create a dataframe of randomly sampled rows, append to list
    append_list =  [y.loc[y[col]==1].sample(n=n-y[col].sum(), replace=True, random_state=2
0) for col in imbalanced.columns]
    append_list.append(y)
    y = pd.concat(append_list, axis=0)
    # match y indexes on other inputs
    X = X.loc[y.index]
    docs = pd.DataFrame(docs_train, index=y_train.index).loc[y.index]
    assert (y.index.all() == X.index.all() == docs.index.all())
    return X, docs.values, y
```

In [ ]:
```
# Separate into X and Y
cols = ['GICS Sector', 'vix', 'rm_week', 'rm_month', 'rm_qtr', 'rm_year']
cols.extend(list(mlb.classes_))

X = df[cols]
docs = df['processed_text']
y = df['signal']

# Get Dummies
docs = tokenize_and_pad(docs)
X = pd.get_dummies(columns = ['GICS Sector'], prefix="sector", data=X)
y = pd.get_dummies(columns=['signal'], data=y)

aux_shape = len(X.columns)
```

In [ ]:
```
# Split into train, validation and test data
X_train, X_test, y_train, y_test, docs_train, docs_test = train_test_split(X, y, docs,
                                                    stratify=y,
                                                    test_size=0.3,
                                                    random_state=20)

X_val, X_test, y_val, y_test, docs_val, docs_test = train_test_split(X_test, y_test, docs_
test,
                                                    stratify=y_test,
                                                    test_size=0.5,
                                                    random_state=20)
```

In [ ]:
```
cont_features = ['vix', 'rm_week', 'rm_month', 'rm_qtr', 'rm_year']
aux_features = cont_features + [item for item in mlb.classes_]
ss = StandardScaler()
X_train[cont_features] = ss.fit_transform(X_train[cont_features])
X_val[cont_features] = ss.transform(X_val[cont_features])
X_test[cont_features] = ss.transform(X_test[cont_features])

X_train, docs_train, y_train = oversample(X_train, docs_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
-a-copy
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:1734: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
-a-copy
  isetter(loc, value[:, i].tolist())
```

3. Load Word Embeddings

In [ ]:
```
embeddings_index = load_embeddings("/content/drive/My Drive/NLP-Stock-Prediction-master/glove.6B.100d.txt")
```

```
Loading Glove Model
Done. 400000 words loaded!
```

In [ ]:
```
words_not_found = []

embedding_matrix = np.zeros((vocab_size, embed_dim))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
    else:
        words_not_found.append(word)
print('number of null word embeddings: %d' % np.sum(np.sum(embedding_matrix, axis=1) == 0))
```

```
number of null word embeddings: 1565176
```

In [ ]:
```
# Save data
np.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/docs_train.npy",docs_train)
np.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/docs_val.npy",docs_val)
np.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/docs_test.npy",docs_test)

X_train.to_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/X_train.pkl")
X_val.to_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/X_val.pkl")
X_test.to_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/X_test.pkl")

y_train.to_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/y_train.pkl")
y_val.to_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/y_val.pkl")
y_test.to_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/y_test.pkl")

np.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/embedding_matrix.npy",embedding_matrix)
```

4. Build & Train Models

In [ ]:
```
def build_model(output_classes, architecture, vaux_shape=aux_shape, vvocab_size=vocab_size, vembed_dim=embed_dim, embedding_matrix=embedding_matrix, max_seq_len=max_words):

    #with tf.device('/cpu:0'):
    main_input= Input(shape=(max_seq_len,), name='doc_input')
    main = Embedding(input_dim = vocab_size,
                         output_dim = embed_dim,
```

```
                        weights = [embedding_matrix],
                        input_length = max_seq_len,
                        trainable=False)(main_input)

    if architecture == 'mlp':
        # Densely Connected Neural Network (Multi-Layer Perceptron)
        main = Dense(32, activation='relu')(main)
        main = Dropout(0.2)(main)
        main = Flatten()(main)
    elif architecture == 'cnn':
        # 1-D Convolutional Neural Network
        main = Conv1D(64, 3, strides=1, padding='same', activation='relu')(main)
        #Cuts the size of the output in half, maxing over every 2 inputs
        main = MaxPooling1D(pool_size=3)(main)
        main = Dropout(0.2)(main)
        main = Conv1D(32, 3, strides=1, padding='same', activation='relu')(main)
        main = GlobalMaxPooling1D()(main)
    elif architecture == 'rnn':
        main = Bidirectional(layer=GRU(32, return_sequences=False), merge_mode='concat')(m
ain)
        main = BatchNormalization()(main)
    elif architecture =="rnn_cnn":
        main = Conv1D(64, 5, padding='same', activation='relu')(main)
        main = MaxPooling1D()(main)
        main = Dropout(0.2)(main)
        main = Bidirectional(layer=GRU(32,return_sequences=False),merge_mode='concat')(mai
n)
        main = BatchNormalization()(main)

    else:
        print('Error: Model type not found.')

    auxiliary_input = Input(shape=(aux_shape,), name='aux_input')
    x = concatenate([main, auxiliary_input])
    x = Dense(32, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='relu')(x)
    x = Dense(32, activation='relu')(x)
    main_output = Dense(output_classes, activation='softmax', name='main_output')(x)
    model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output], name=archit
ecture)

    #sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    #model = multi_gpu_model(model, gpus=2)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy',
'AUC'])

    return model


def gen():
    print('generator initiated')
    idx = 0
```

```
    while True:
        yield [docs_train[:32], X_train[:32]], y_train[:32]
        print('generator yielded a batch %d' % idx)
        idx += 1
```

In [ ]:
```
model_dict = dict()
```

In [ ]:
```
mlp = build_model(3, "mlp")

model_dict["mlp"] = mlp.fit([docs_train, X_train], y_train, batch_size=64, epochs=10, verb
ose=1, validation_data = ([docs_val, X_val], y_val))

mlp.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/mlp.hdf5")
with open('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/mlp.pkl
', 'wb') as file_pi:
  pickle.dump(model_dict["mlp"].history, file_pi)
```

```
Epoch 1/10
37/37 [==============================] - 6s 165ms/step - loss: 1.0867 - accuracy: 0.4230 - auc: 0.6125 - val_loss: 1.0529 - val_acc
uracy: 0.5672 - val_auc: 0.7156
Epoch 2/10
37/37 [==============================] - 6s 156ms/step - loss: 0.9119 - accuracy: 0.6486 - auc: 0.8250 - val_loss: 1.3063 - val_acc
uracy: 0.3739 - val_auc: 0.6504
Epoch 3/10
37/37 [==============================] - 6s 156ms/step - loss: 0.6840 - accuracy: 0.7559 - auc: 0.9106 - val_loss: 1.8054 - val_acc
uracy: 0.6345 - val_auc: 0.7616
Epoch 4/10
37/37 [==============================] - 6s 156ms/step - loss: 0.5278 - accuracy: 0.7795 - auc: 0.9302 - val_loss: 2.3256 - val_acc
uracy: 0.6050 - val_auc: 0.7302
Epoch 5/10
37/37 [==============================] - 6s 155ms/step - loss: 0.5004 - accuracy: 0.8027 - auc: 0.9452 - val_loss: 2.7536 - val_acc
uracy: 0.6261 - val_auc: 0.7447
Epoch 6/10
37/37 [==============================] - 6s 152ms/step - loss: 0.4457 - accuracy: 0.8121 - auc: 0.9491 - val_loss: 2.5481 - val_acc
uracy: 0.6261 - val_auc: 0.7511
Epoch 7/10
37/37 [==============================] - 6s 152ms/step - loss: 0.4053 - accuracy: 0.8202 - auc: 0.9558 - val_loss: 3.0857 - val_acc
uracy: 0.5966 - val_auc: 0.7407
Epoch 8/10
37/37 [==============================] - 6s 153ms/step - loss: 0.4359 - accuracy: 0.8198 - auc: 0.9558 - val_loss: 2.8573 - val_acc
uracy: 0.5882 - val_auc: 0.7264
Epoch 9/10
37/37 [==============================] - 6s 153ms/step - loss: 0.4059 - accuracy: 0.8190 - auc: 0.9564 - val_loss: 2.7048 - val_acc
uracy: 0.5882 - val_auc: 0.7278
Epoch 10/10
37/37 [==============================] - 6s 152ms/step - loss: 0.4068 - accuracy: 0.8228 - auc: 0.9587 - val_loss: 2.9846 - val_acc
uracy: 0.5630 - val_auc: 0.7109
```

In [ ]:
```
cnn = build_model(3, "cnn")

model_dict["cnn"] = cnn.fit([docs_train, X_train], y_train, batch_size=64, epochs=10, verb
ose=1, validation_data = ([docs_val, X_val], y_val))

cnn.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/cnn.hdf5")
with open('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/cnn.pkl
', 'wb') as file_pi:
    pickle.dump(model_dict["cnn"].history, file_pi)
```

```
Epoch 1/10
37/37 [==============================] - 7s 196ms/step - loss: 1.0957 - accuracy: 0.3625 - auc: 0.5328 - val_loss: 1.1090 - val_acc
uracy: 0.3109 - val_auc: 0.4924
Epoch 2/10
37/37 [==============================] - 7s 186ms/step - loss: 1.0802 - accuracy: 0.3874 - auc: 0.5787 - val_loss: 1.1044 - val_acc
uracy: 0.3235 - val_auc: 0.5070
Epoch 3/10
37/37 [==============================] - 7s 182ms/step - loss: 1.0570 - accuracy: 0.4530 - auc: 0.6346 - val_loss: 1.1052 - val_acc
uracy: 0.3319 - val_auc: 0.5141
Epoch 4/10
37/37 [==============================] - 7s 182ms/step - loss: 1.0464 - accuracy: 0.4758 - auc: 0.6515 - val_loss: 1.0575 - val_acc
uracy: 0.4790 - val_auc: 0.6417
Epoch 5/10
37/37 [==============================] - 7s 182ms/step - loss: 1.0223 - accuracy: 0.5054 - auc: 0.6787 - val_loss: 1.0091 - val_acc
uracy: 0.6008 - val_auc: 0.7157
Epoch 6/10
37/37 [==============================] - 7s 181ms/step - loss: 1.0004 - accuracy: 0.5079 - auc: 0.6937 - val_loss: 1.0742 - val_acc
uracy: 0.4370 - val_auc: 0.6023
Epoch 7/10
37/37 [==============================] - 7s 182ms/step - loss: 0.9810 - accuracy: 0.5199 - auc: 0.7109 - val_loss: 0.9717 - val_acc
uracy: 0.6050 - val_auc: 0.7359
Epoch 8/10
37/37 [==============================] - 7s 181ms/step - loss: 0.9496 - accuracy: 0.5491 - auc: 0.7339 - val_loss: 0.9841 - val_acc
uracy: 0.5420 - val_auc: 0.7125
Epoch 9/10
37/37 [==============================] - 7s 184ms/step - loss: 0.9313 - accuracy: 0.5568 - auc: 0.7481 - val_loss: 0.9548 - val_acc
uracy: 0.5672 - val_auc: 0.7326
Epoch 10/10
37/37 [==============================] - 7s 182ms/step - loss: 0.9102 - accuracy: 0.5671 - auc: 0.7616 - val_loss: 0.9935 - val_acc
uracy: 0.5504 - val_auc: 0.7000
```

In [ ]:
```
rnn = build_model(3, "rnn")

model_dict["rnn"] = rnn.fit([docs_train, X_train], y_train, batch_size=32, epochs=10, verb
ose=1, validation_data = ([docs_val, X_val], y_val))

rnn.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn.hdf5")
with open('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/rnn.pkl
', 'wb') as file_pi:
    pickle.dump(model_dict["rnn"].history, file_pi)
```

```
Epoch 1/10
73/73 [==============================] - 146s 2s/step - loss: 1.1023 - accuracy: 0.3754 - auc: 0.5433 - val_loss: 1.0933 - val_accu
racy: 0.3151 - val_auc: 0.5518
Epoch 2/10
73/73 [==============================] - 144s 2s/step - loss: 1.0665 - accuracy: 0.4269 - auc: 0.6090 - val_loss: 1.0611 - val_accu
racy: 0.4496 - val_auc: 0.6286
Epoch 3/10
73/73 [==============================] - 144s 2s/step - loss: 1.0328 - accuracy: 0.4715 - auc: 0.6580 - val_loss: 1.0143 - val_accu
racy: 0.5084 - val_auc: 0.6780
Epoch 4/10
73/73 [==============================] - 143s 2s/step - loss: 1.0079 - accuracy: 0.4865 - auc: 0.6802 - val_loss: 1.0246 - val_accu
racy: 0.4874 - val_auc: 0.6632
Epoch 5/10
73/73 [==============================] - 144s 2s/step - loss: 0.9829 - accuracy: 0.5049 - auc: 0.7013 - val_loss: 1.0059 - val_accu
racy: 0.5252 - val_auc: 0.6826
Epoch 6/10
73/73 [==============================] - 143s 2s/step - loss: 0.9657 - accuracy: 0.5199 - auc: 0.7187 - val_loss: 1.0113 - val_accu
racy: 0.5042 - val_auc: 0.6809
Epoch 7/10
73/73 [==============================] - 143s 2s/step - loss: 0.9577 - accuracy: 0.5380 - auc: 0.7259 - val_loss: 0.9668 - val_accu
racy: 0.5630 - val_auc: 0.7238
Epoch 8/10
73/73 [==============================] - 143s 2s/step - loss: 0.9365 - accuracy: 0.5581 - auc: 0.7403 - val_loss: 0.9593 - val_accu
racy: 0.5420 - val_auc: 0.7172
Epoch 9/10
73/73 [==============================] - 143s 2s/step - loss: 0.9138 - accuracy: 0.5731 - auc: 0.7553 - val_loss: 0.8972 - val_accu
racy: 0.5756 - val_auc: 0.7716
Epoch 10/10
73/73 [==============================] - 143s 2s/step - loss: 0.9094 - accuracy: 0.5744 - auc: 0.7593 - val_loss: 0.9649 - val_accu
racy: 0.5504 - val_auc: 0.7182
```

In [ ]:
```
rnn_cnn = build_model(3, "rnn_cnn")

model_dict["rnn_cnn"] = rnn_cnn.fit([docs_train, X_train], y_train, batch_size=32, epochs=
10, verbose=1, validation_data = ([docs_val, X_val], y_val))

rnn_cnn.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn_cnn.hd
f5")
with open('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/rnn_cnn
.pkl', 'wb') as file_pi:
    pickle.dump(model_dict["rnn_cnn"].history, file_pi)
```

```
Epoch 1/10
73/73 [==============================] - 77s 1s/step - loss: 1.0976 - accuracy: 0.3685 - auc: 0.5384 - val_loss: 1.0973 - val_accur
acy: 0.3782 - val_auc: 0.5213
Epoch 2/10
73/73 [==============================] - 76s 1s/step - loss: 1.0673 - accuracy: 0.4324 - auc: 0.6177 - val_loss: 1.0331 - val_accur
acy: 0.5462 - val_auc: 0.7045
Epoch 3/10
73/73 [==============================] - 76s 1s/step - loss: 1.0425 - accuracy: 0.4698 - auc: 0.6460 - val_loss: 1.0557 - val_accur
acy: 0.4874 - val_auc: 0.6291
Epoch 4/10
73/73 [==============================] - 76s 1s/step - loss: 1.0220 - accuracy: 0.4809 - auc: 0.6680 - val_loss: 1.0440 - val_accur
acy: 0.4748 - val_auc: 0.6358
Epoch 5/10
73/73 [==============================] - 76s 1s/step - loss: 1.0033 - accuracy: 0.4818 - auc: 0.6844 - val_loss: 0.9900 - val_accur
acy: 0.5294 - val_auc: 0.7029
Epoch 6/10
73/73 [==============================] - 76s 1s/step - loss: 0.9781 - accuracy: 0.5204 - auc: 0.7089 - val_loss: 0.9063 - val_accur
acy: 0.6050 - val_auc: 0.7788
Epoch 7/10
73/73 [==============================] - 76s 1s/step - loss: 0.9582 - accuracy: 0.5311 - auc: 0.7201 - val_loss: 0.9941 - val_accur
acy: 0.5252 - val_auc: 0.7040
Epoch 8/10
73/73 [==============================] - 76s 1s/step - loss: 0.9475 - accuracy: 0.5354 - auc: 0.7304 - val_loss: 1.0089 - val_accur
acy: 0.5168 - val_auc: 0.6837
Epoch 9/10
73/73 [==============================] - 76s 1s/step - loss: 0.9284 - accuracy: 0.5504 - auc: 0.7456 - val_loss: 1.0140 - val_accur
acy: 0.5084 - val_auc: 0.6811
Epoch 10/10
73/73 [==============================] - 75s 1s/step - loss: 0.9063 - accuracy: 0.5689 - auc: 0.7583 - val_loss: 0.9745 - val_accur
acy: 0.5336 - val_auc: 0.7159


---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-20-d4b2257dc3e6> in <module>()
      4
      5 rnn_cnn.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn_cnn.hdf5")
----> 6 with open('Data/trainHistory/rnn_cnn.pkl', 'wb') as file_pi:
      7     pickle.dump(model_dict["rnn_cnn"].history, file_pi)

FileNotFoundError: [Errno 2] No such file or directory: 'Data/trainHistory/rnn_cnn.pkl'
```

5. Model Evaluation

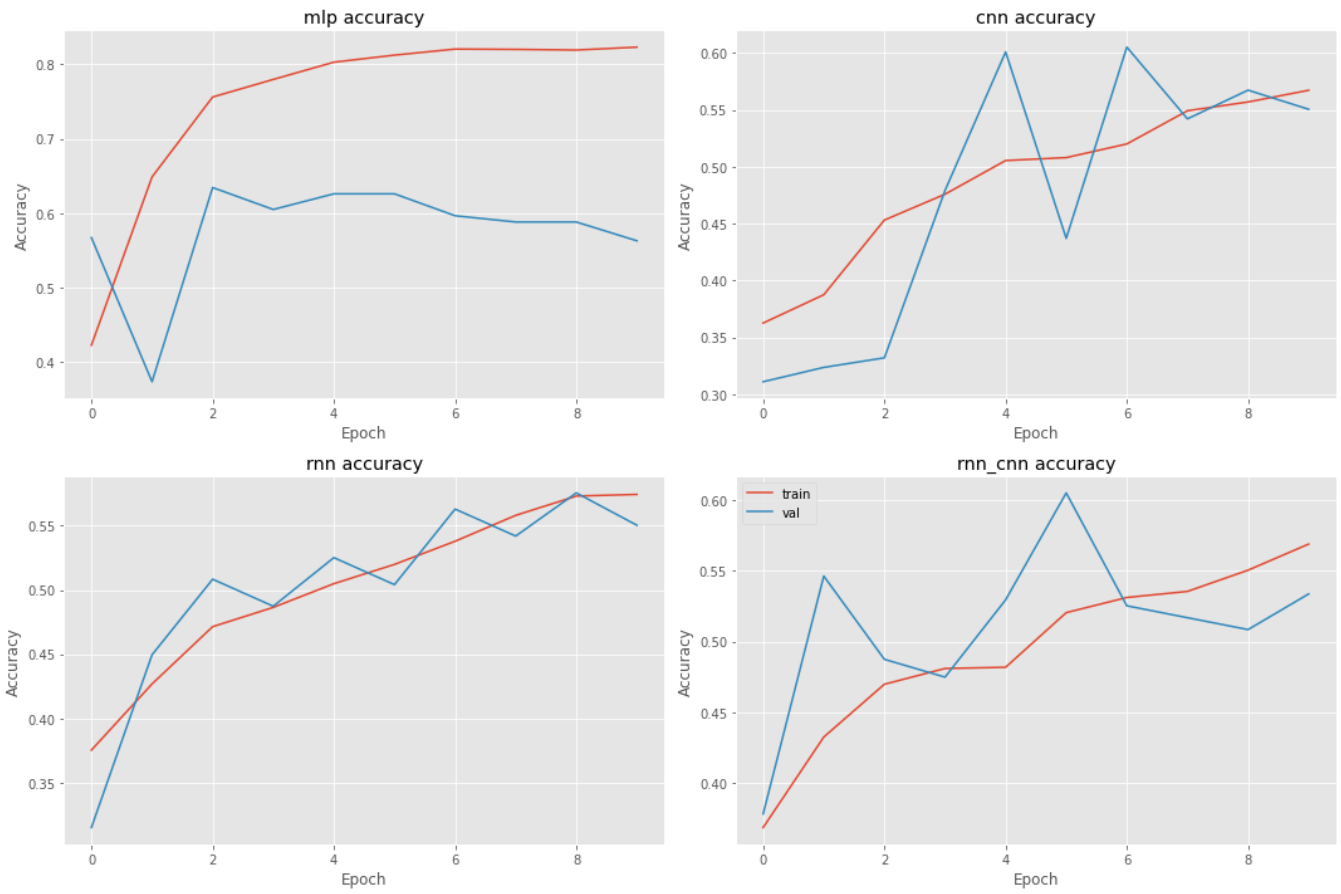
Loss, Accuracy, and AUC_ROC graphs for training and validation data

In [ ]:
```
X_test = pd.read_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/X_test.pkl")
y_test = pd.read_pickle("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/y_test.pkl")
docs_test = np.load("/content/drive/My Drive/NLP-Stock-Prediction-master/Pickles/docs_test.npy")

mlp_hist = pickle.load(open("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/mlp.pkl","rb"))
cnn_hist = pickle.load(open("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/cnn.pkl","rb"))
rnn_hist = pickle.load(open("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/rnn.pkl","rb"))
rnn_cnn_hist = pickle.load(open("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/rnn_cnn.pkl","rb"))
```

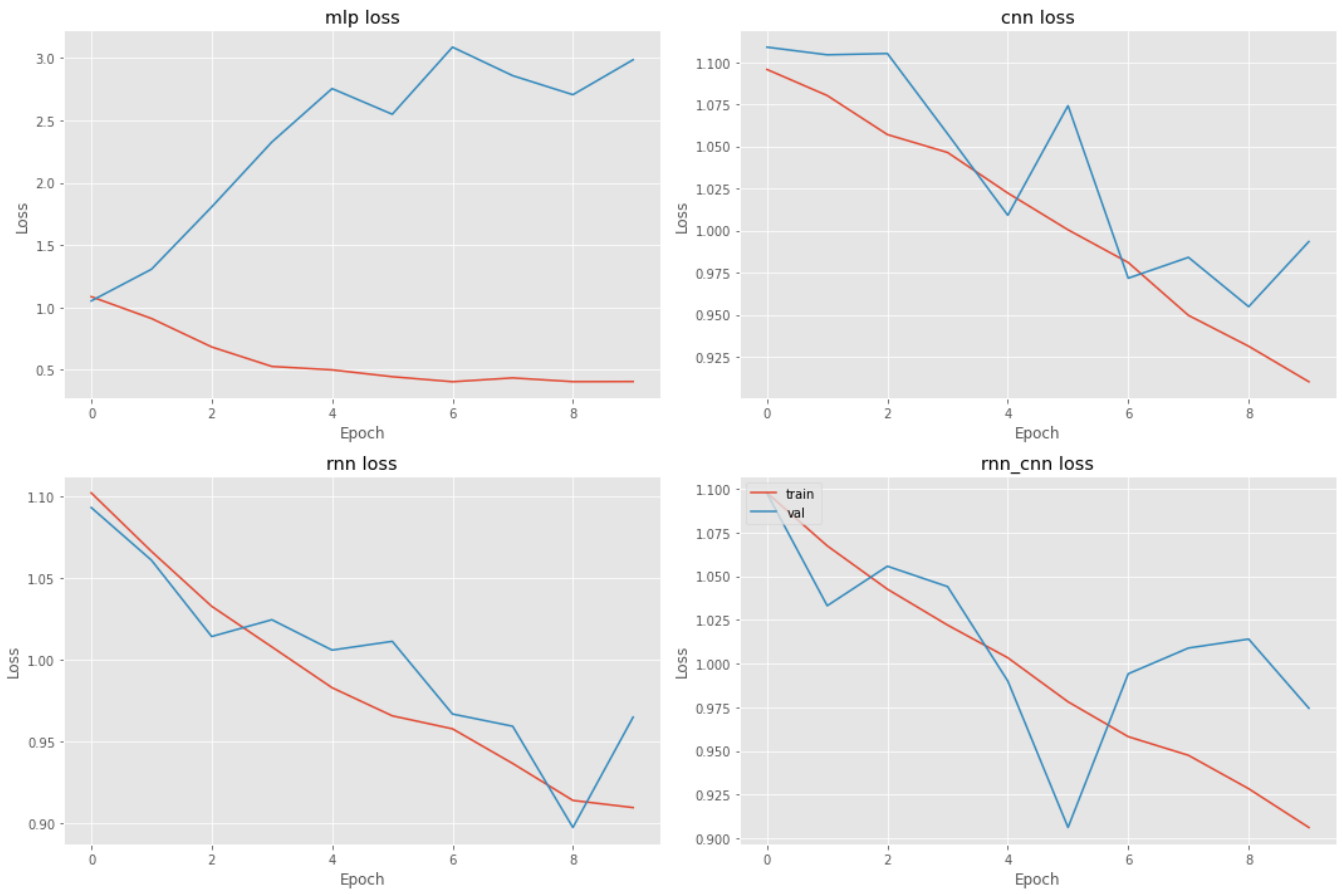
In [ ]:
```
from keras.models import load_model

mlp = load_model('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/mlp.hdf5')
cnn = load_model('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/cnn.hdf5')
rnn = load_model('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn.hdf5')
rnn_cnn = load_model('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn_cnn.hdf5')
```

In [ ]:
```
def plot_metrics(model_dict, metric, x_label, y_label):
    val_metric = 'val_'+metric
    plots = 1
    plt.figure(figsize=[15,10])
    for model, history in model_dict.items():
        plt.subplot(2, 2, plots)
        plt.plot(history[metric])
        plt.plot(history[val_metric])
        plt.title('{0} {1}'.format(model, metric))
        plt.ylabel(y_label)
        plt.xlabel(x_label)
        plots += 1
    plt.legend(['train', 'val'], loc='upper left')
    plt.tight_layout()
    #plt.savefig("Graphs/{}.png".format(metric),format="png") uncomment later
    plt.show()
plt.style.use("ggplot")
model_dict = {"mlp": mlp_hist,
              "cnn": cnn_hist,
              "rnn": rnn_hist,
              "rnn_cnn": rnn_cnn_hist}
```

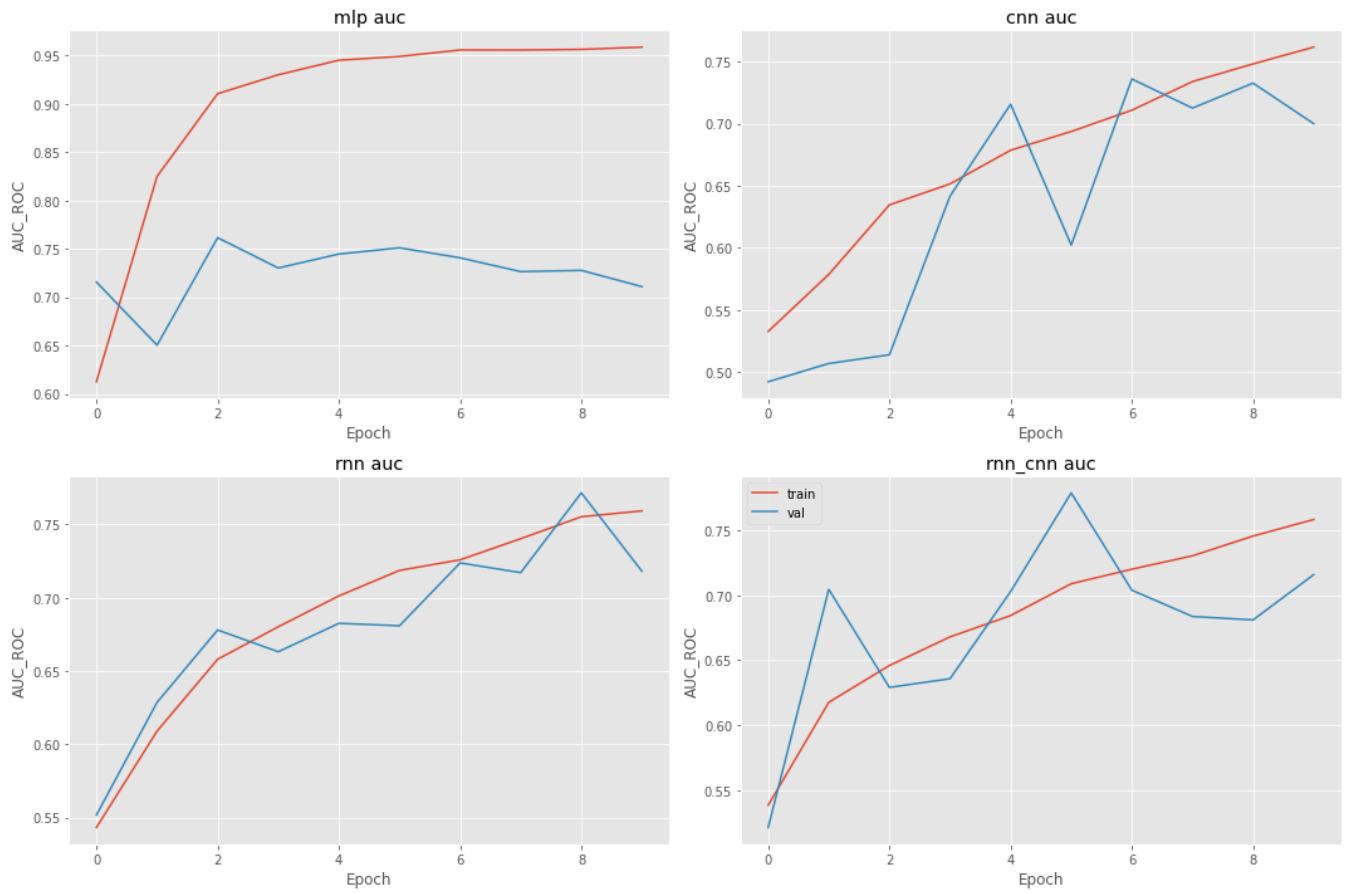
In [ ]:
```
plot_metrics(model_dict,"accuracy", "Epoch", "Accuracy") ## Accuracy
```



In [ ]:
```
plot_metrics(model_dict,"loss","Epoch","Loss") ## Loss
```



In [ ]:
```
plot_metrics(model_dict,'auc',"Epoch","AUC_ROC") ## AUC_ROC
```



After 10 epochs, the RNN and RNN_CNN models generalize the best. We can try extending training for more epochs to see how training improves. I focus on the CNN-RNN model as it has had comparable accuracy to the RNN model and was trained in half the time.

6. Extend Training for More Epochs

In [ ]:
```
history = rnn_cnn.fit(x = [docs_train, X_train],
                      y = y_train,
                      batch_size = 32,
                      epochs = 20,
                      verbose = 1,
                      validation_data = ([docs_val, X_val], y_val))
```

```
Epoch 1/20
73/73 [==============================] - 80s 1s/step - loss: 0.9091 - accuracy: 0.5753 - auc_3: 0.7580 - val_loss: 0.9607 - val_acc
uracy: 0.5378 - val_auc_3: 0.7278
Epoch 2/20
73/73 [==============================] - 78s 1s/step - loss: 0.8835 - accuracy: 0.5813 - auc_3: 0.7745 - val_loss: 0.9548 - val_acc
uracy: 0.5420 - val_auc_3: 0.7354
Epoch 3/20
73/73 [==============================] - 77s 1s/step - loss: 0.8616 - accuracy: 0.5997 - auc_3: 0.7860 - val_loss: 1.0498 - val_acc
uracy: 0.4874 - val_auc_3: 0.6788
Epoch 4/20
73/73 [==============================] - 76s 1s/step - loss: 0.8618 - accuracy: 0.5860 - auc_3: 0.7861 - val_loss: 1.0309 - val_acc
uracy: 0.5588 - val_auc_3: 0.7166
Epoch 5/20
73/73 [==============================] - 75s 1s/step - loss: 0.8371 - accuracy: 0.6040 - auc_3: 0.7999 - val_loss: 1.2365 - val_acc
uracy: 0.4412 - val_auc_3: 0.5879
Epoch 6/20
73/73 [==============================] - 75s 1s/step - loss: 0.8169 - accuracy: 0.6233 - auc_3: 0.8118 - val_loss: 1.0575 - val_acc
uracy: 0.5336 - val_auc_3: 0.7142
Epoch 7/20
73/73 [==============================] - 75s 1s/step - loss: 0.8217 - accuracy: 0.6212 - auc_3: 0.8096 - val_loss: 1.2524 - val_acc
uracy: 0.4328 - val_auc_3: 0.5812
Epoch 8/20
73/73 [==============================] - 75s 1s/step - loss: 0.8078 - accuracy: 0.6315 - auc_3: 0.8190 - val_loss: 1.0039 - val_acc
uracy: 0.5924 - val_auc_3: 0.7480
Epoch 9/20
73/73 [==============================] - 75s 1s/step - loss: 0.7934 - accuracy: 0.6371 - auc_3: 0.8223 - val_loss: 0.9936 - val_acc
uracy: 0.5504 - val_auc_3: 0.7305
Epoch 10/20
73/73 [==============================] - 74s 1s/step - loss: 0.7724 - accuracy: 0.6478 - auc_3: 0.8335 - val_loss: 1.1169 - val_acc
uracy: 0.4832 - val_auc_3: 0.6577
Epoch 11/20
73/73 [==============================] - 75s 1s/step - loss: 0.7660 - accuracy: 0.6521 - auc_3: 0.8374 - val_loss: 0.9062 - val_acc
uracy: 0.6092 - val_auc_3: 0.7841
Epoch 12/20
73/73 [==============================] - 75s 1s/step - loss: 0.7615 - accuracy: 0.6585 - auc_3: 0.8391 - val_loss: 1.0078 - val_acc
uracy: 0.5504 - val_auc_3: 0.7352
Epoch 13/20
73/73 [==============================] - 74s 1s/step - loss: 0.7372 - accuracy: 0.6692 - auc_3: 0.8492 - val_loss: 1.0354 - val_acc
uracy: 0.5672 - val_auc_3: 0.7237
Epoch 14/20
73/73 [==============================] - 74s 1s/step - loss: 0.7263 - accuracy: 0.6757 - auc_3: 0.8540 - val_loss: 1.0104 - val_acc
uracy: 0.5924 - val_auc_3: 0.7522
Epoch 15/20
73/73 [==============================] - 75s 1s/step - loss: 0.7303 - accuracy: 0.6752 - auc_3: 0.8539 - val_loss: 1.0405 - val_acc
uracy: 0.5378 - val_auc_3: 0.7261
Epoch 16/20
73/73 [==============================] - 75s 1s/step - loss: 0.7179 - accuracy: 0.6744 - auc_3: 0.8571 - val_loss: 1.0532 - val_acc
uracy: 0.5630 - val_auc_3: 0.7191
Epoch 17/20
73/73 [==============================] - 75s 1s/step - loss: 0.7065 - accuracy: 0.6877 - auc_3: 0.8622 - val_loss: 1.1069 - val_acc
uracy: 0.5000 - val_auc_3: 0.6974
Epoch 18/20
73/73 [==============================] - 75s 1s/step - loss: 0.7076 - accuracy: 0.6911 - auc_3: 0.8624 - val_loss: 1.0633 - val_acc
uracy: 0.5462 - val_auc_3: 0.7195
Epoch 19/20
73/73 [==============================] - 75s 1s/step - loss: 0.6818 - accuracy: 0.6928 - auc_3: 0.8729 - val_loss: 1.0602 - val_acc
uracy: 0.5546 - val_auc_3: 0.7316
Epoch 20/20
73/73 [==============================] - 75s 1s/step - loss: 0.6681 - accuracy: 0.7117 - auc_3: 0.8790 - val_loss: 1.0595 - val_acc
uracy: 0.5756 - val_auc_3: 0.7352
```

In [ ]:
```
rnn_cnn.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn_cnn_30
.hdf5")
with open('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/rnn_cnn
_30.pkl', 'wb') as file_pi:
    pickle.dump(history.history, file_pi)
```

Validation accuracy doesn't seem to be improving much, we can try training for another few epochs to be safe...

In [ ]:
```
history2 = rnn.fit(x = [docs_train, X_train],
                   y = y_train,
                   batch_size = 32,
                   epochs = 20,
                   verbose = 1,
                   validation_data = ([docs_val, X_val], y_val))
```

```
Epoch 1/20
73/73 [==============================] - 141s 2s/step - loss: 0.8902 - accuracy: 0.5890 - auc_2: 0.7716 - val_loss: 1.0983 - val_ac
curacy: 0.4580 - val_auc_2: 0.6252
Epoch 2/20
73/73 [==============================] - 140s 2s/step - loss: 0.8790 - accuracy: 0.5976 - auc_2: 0.7787 - val_loss: 1.1019 - val_ac
curacy: 0.4118 - val_auc_2: 0.6162
Epoch 3/20
73/73 [==============================] - 140s 2s/step - loss: 0.8543 - accuracy: 0.6152 - auc_2: 0.7946 - val_loss: 0.8735 - val_ac
curacy: 0.6176 - val_auc_2: 0.7939
Epoch 4/20
73/73 [==============================] - 140s 2s/step - loss: 0.8470 - accuracy: 0.6233 - auc_2: 0.7966 - val_loss: 0.9248 - val_ac
curacy: 0.5504 - val_auc_2: 0.7490
Epoch 5/20
73/73 [==============================] - 140s 2s/step - loss: 0.8345 - accuracy: 0.6139 - auc_2: 0.8035 - val_loss: 0.9487 - val_ac
curacy: 0.5420 - val_auc_2: 0.7363
Epoch 6/20
73/73 [==============================] - 140s 2s/step - loss: 0.8364 - accuracy: 0.6105 - auc_2: 0.8013 - val_loss: 1.0460 - val_ac
curacy: 0.4958 - val_auc_2: 0.6959
Epoch 7/20
73/73 [==============================] - 140s 2s/step - loss: 0.8080 - accuracy: 0.6435 - auc_2: 0.8177 - val_loss: 1.0040 - val_ac
curacy: 0.5672 - val_auc_2: 0.7187
Epoch 8/20
73/73 [==============================] - 139s 2s/step - loss: 0.7919 - accuracy: 0.6547 - auc_2: 0.8259 - val_loss: 1.1395 - val_ac
curacy: 0.4496 - val_auc_2: 0.6565
Epoch 9/20
73/73 [==============================] - 140s 2s/step - loss: 0.7784 - accuracy: 0.6534 - auc_2: 0.8312 - val_loss: 1.0340 - val_ac
curacy: 0.5882 - val_auc_2: 0.7554
Epoch 10/20
73/73 [==============================] - 141s 2s/step - loss: 0.7758 - accuracy: 0.6478 - auc_2: 0.8320 - val_loss: 1.0125 - val_ac
curacy: 0.5798 - val_auc_2: 0.7425
Epoch 11/20
73/73 [==============================] - 143s 2s/step - loss: 0.7515 - accuracy: 0.6658 - auc_2: 0.8425 - val_loss: 1.0021 - val_ac
curacy: 0.5798 - val_auc_2: 0.7556
Epoch 12/20
73/73 [==============================] - 146s 2s/step - loss: 0.7427 - accuracy: 0.6615 - auc_2: 0.8468 - val_loss: 0.9929 - val_ac
curacy: 0.6303 - val_auc_2: 0.7767
Epoch 13/20
73/73 [==============================] - 145s 2s/step - loss: 0.7174 - accuracy: 0.6834 - auc_2: 0.8581 - val_loss: 1.1007 - val_ac
curacy: 0.5546 - val_auc_2: 0.7121
Epoch 14/20
73/73 [==============================] - 146s 2s/step - loss: 0.7176 - accuracy: 0.6782 - auc_2: 0.8575 - val_loss: 1.1189 - val_ac
curacy: 0.5126 - val_auc_2: 0.7078
Epoch 15/20
73/73 [==============================] - 145s 2s/step - loss: 0.7008 - accuracy: 0.6933 - auc_2: 0.8646 - val_loss: 1.1064 - val_ac
curacy: 0.4832 - val_auc_2: 0.6613
Epoch 16/20
73/73 [==============================] - 144s 2s/step - loss: 0.7060 - accuracy: 0.6817 - auc_2: 0.8620 - val_loss: 1.1454 - val_ac
curacy: 0.4916 - val_auc_2: 0.6662
Epoch 17/20
73/73 [==============================] - 145s 2s/step - loss: 0.6931 - accuracy: 0.6881 - auc_2: 0.8678 - val_loss: 1.1195 - val_ac
curacy: 0.5294 - val_auc_2: 0.7161
Epoch 18/20
73/73 [==============================] - 145s 2s/step - loss: 0.6705 - accuracy: 0.7134 - auc_2: 0.8780 - val_loss: 1.1687 - val_ac
curacy: 0.5462 - val_auc_2: 0.7099
Epoch 19/20
73/73 [==============================] - 143s 2s/step - loss: 0.6780 - accuracy: 0.7057 - auc_2: 0.8738 - val_loss: 1.2671 - val_ac
curacy: 0.5000 - val_auc_2: 0.6939
Epoch 20/20
73/73 [==============================] - 144s 2s/step - loss: 0.6561 - accuracy: 0.7199 - auc_2: 0.8839 - val_loss: 1.0823 - val_ac
curacy: 0.6639 - val_auc_2: 0.8055
```

In [ ]:

```
rnn.save("/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn_30.hdf5")
with open('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/trainHistory/rnn_30.
pkl', 'wb') as file_pi:
    pickle.dump(history2.history, file_pi)
```

It doesn't look more epochs will improve the model any further

5. Evaluation on Test Data

The three metrics are listed as

[test_loss, test_accuracy, test_auc_roc]

In [ ]:
```
rnn_cnn_30 = load_model('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models
/rnn_cnn_30.hdf5')
rnn_cnn_30.evaluate([docs_test, X_test], y_test, batch_size=64)
```

```
4/4 [==============================] - 2s 471ms/step - loss: 1.0764 - accuracy: 0.5314 - auc_3: 0.7226

[1.07644522190094, 0.5313807725906372, 0.722558319568634]
```

In [ ]:
```
rnn_30 = load_model('/content/drive/My Drive/NLP-Stock-Prediction-master/Data 2/models/rnn
_30.hdf5')
rnn_30.evaluate([docs_test, X_test], y_test, batch_size=64)
```

```
4/4 [==============================] - 3s 871ms/step - loss: 0.9437 - accuracy: 0.6485 - auc_2: 0.8232

[0.94370037317276, 0.6485355496406555, 0.8232173919677734]
```

In [ ]:
```
predictions = rnn_30.predict([docs_test, X_test])
pred_class = predictions.argmax(axis=-1)

res = (X_test.reset_index()).merge(pd.DataFrame(pred_class.ravel()), left_index = True, ri
ght_index=True).rename(columns={0:'Prediction'})
res2 = df.merge(res, left_on='Unnamed: 0', right_on='index')
res2[['ticker','signal','Prediction']]
```

	ticker	signal	Prediction
0	ADSK	stay	1
1	ADSK	stay	1
2	ADSK	stay	1
3	ADSK	stay	1
4	ADSK	up	1
...	...	...	...
234	YUM	stay	1
235	YUM	stay	1
236	YUM	stay	0
237	ZBH	up	0
238	ZBH	stay	0

239 rows × 3 columns