In [5]:
```
import pandas as pd
import numpy as np
```

In [6]:
```
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

In [7]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

In [9]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout,Activation

from keras.layers.convolutional import Conv1D
from keras import backend as K
from keras.layers.core import Lambda
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding

import keras
from keras.utils import np_utils
```

In [10]:
```
data = pd.read_csv("IDS_Combined_Data_IoT.csv")
```

In [11]:
```
data.head(5)
```

Out[11]:

	ts	date	time	fridge_temperature	temp_condition	label	type	door_state	sphone_signal	latitude	...	FC2_Read_Discrete_Value	FC3_Read_H
0	1556245180	25-Apr-19	19:19:40	9.00	1	1	ddos	0	0	4.514077	...	32708	
1	1556245180	25-Apr-19	19:19:40	9.25	1	1	ddos	0	0	4.514077	...	32708	
2	1556245185	25-Apr-19	19:19:45	12.65	1	1	ddos	0	0	4.514077	...	32708	
3	1556245185	25-Apr-19	19:19:45	4.65	0	1	ddos	0	0	4.514077	...	32708	
4	1556245195	25-Apr-19	19:19:55	12.65	1	1	ddos	0	0	4.514077	...	32708	

5 rows × 22 columns

In [12]:
```
dataset = data.drop(['ts', 'date', 'time'], axis =1)
```

In [13]:
```
dataset.head(5)
```

Out[13]:

	fridge_temperature	temp_condition	label	type	door_state	sphone_signal	latitude	longitude	FC1_Read_Input_Register	FC2_Read_Discrete_Value	FC3_R
0	9.00	1	1	ddos	0	0	4.514077	14.421946	32450	32708	
1	9.25	1	1	ddos	0	0	4.514077	14.421946	32450	32708	
2	12.65	1	1	ddos	0	0	4.514077	14.421946	32450	32708	
3	4.65	0	1	ddos	0	0	4.514077	14.421946	32450	32708	
4	12.65	1	1	ddos	0	0	4.514077	14.421946	32450	32708	

In [14]:
```
dataset['label'].value_counts()
```

Out[14]:
```
0    245000
1    156119
Name: label, dtype: int64
```

In [15]:
```
dataset.isna().sum().sum()
```

Out[15]:
```
0
```

In [16]:
```
dataset.columns
```

Out[16]:
```
Index(['fridge_temperature', 'temp_condition', 'label', 'type', 'door_state',
       'sphone_signal', 'latitude', 'longitude', 'FC1_Read_Input_Register',
       'FC2_Read_Discrete_Value', 'FC3_Read_Holding_Register', 'FC4_Read_Coil',
       'motion_status', 'light_status', 'current_temperature',
       'thermostat_status', 'temperature', 'pressure', 'humidity'],
      dtype='object')
```

In [17]:
```
X = dataset.drop(['label','type'], axis = 1)
X.shape
```

Out[17]:
```
(401119, 17)
```

In [18]:
```
y = dataset['label']
y.shape
```

Out[18]:
```
(401119,)
```

In [19]:
```
sel = VarianceThreshold(threshold=(.8 * (1 - .8))) #don't change these values
X_T = sel.fit(X)
```

In [20]:
```
sel_cols = X_T.get_support(indices=True)
X_VT = X.iloc[:,sel_cols]
X_VT
```

Out[20]:

	fridge_temperature	latitude	longitude	FC1_Read_Input_Register	FC2_Read_Discrete_Value	FC3_Read_Holding_Register	FC4_Read_Coil	current_tempe
0	9.00	4.514077	14.421946	32450	32708	32035	32728	28.
1	9.25	4.514077	14.421946	32450	32708	32035	32728	28.
2	12.65	4.514077	14.421946	32450	32708	32035	32728	28.
3	4.65	4.514077	14.421946	32450	32708	32035	32728	28.
4	12.65	4.514077	14.421946	32450	32708	32035	32728	28.
...	...	...	...	...	...	...	...	
401114	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401115	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401116	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401117	6.70	4.514077	14.421946	32450	32708	32035	32728	28.
401118	6.70	4.514077	14.421946	32450	32708	32035	32728	28.

401119 rows × 11 columns

In [21]:
```
scaler = MinMaxScaler()
X_M = scaler.fit_transform(X_VT)
print(X_M)
```

```
[[0.61538462 0.00821665 0.00811167 ... 0.10772138 0.53355618 0.46251056]
 [0.63461538 0.00821665 0.00811167 ... 0.66504042 0.53355618 0.46251056]
 [0.89615385 0.00821665 0.00811167 ... 0.81188986 0.53355618 0.46251056]
 ...
 [0.43846154 0.00821665 0.00811167 ... 0.16348547 0.52695586 0.3756012 ]
 [0.43846154 0.00821665 0.00811167 ... 0.03263243 0.62043775 0.93746771]
 [0.43846154 0.00821665 0.00811167 ... 0.04525255 0.58120503 0.3689465 ]]
```

In [22]:
```
trainX, testX, trainY, testY = train_test_split(X_M, y, test_size = 0.20, random_state = 40)
```

In [23]:
```
print(X_M.shape)
print(y.shape)
print(trainX.shape)
print(trainY.shape)
print(testX.shape)
print(testY.shape)
```

```
(401119, 11)
(401119,)
(320895, 11)
(320895,)
(80224, 11)
(80224,)
```

In [24]:
```
#Test Train split
lr = LogisticRegression() #for binary
lr.fit(trainX, trainY)
y_predict = lr.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.8492346429996012
              precision    recall  f1-score   support

           0       0.83      0.95      0.88     48896
           1       0.90      0.69      0.78     31328

    accuracy                           0.85     80224
   macro avg       0.86      0.82      0.83     80224
weighted avg       0.86      0.85      0.84     80224
```

In [25]:
```
#Test Train split
lda = LinearDiscriminantAnalysis()
lda.fit(trainX, trainY)
y_predict = lda.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.8287544874351815
              precision    recall  f1-score   support

           0       0.80      0.96      0.87     48896
           1       0.91      0.63      0.74     31328

    accuracy                           0.83     80224
   macro avg       0.85      0.79      0.81     80224
weighted avg       0.84      0.83      0.82     80224
```

In [26]:
```
#Test Train split
#default parameters used in base paper (n_neighbors, default=5) (p, default=2 for Euclidean Distance)
knn = KNeighborsClassifier()
knn.fit(trainX, trainY)
y_predict = knn.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.8866050059832469
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     48896
           1       0.89      0.82      0.85     31328

    accuracy                           0.89     80224
   macro avg       0.89      0.87      0.88     80224
weighted avg       0.89      0.89      0.89     80224
```

In [27]:
```
#Test Train split
rf = RandomForestClassifier(n_estimators=10,criterion='gini')
rf.fit(trainX, trainY)
y_predict1 = rf.predict(testX)
print(accuracy_score(testY, y_predict1))
print(classification_report(testY, y_predict1))
```

```
0.8745138611886717
              precision    recall  f1-score   support

           0       0.88      0.92      0.90     48896
           1       0.86      0.81      0.83     31328

    accuracy                           0.87     80224
   macro avg       0.87      0.86      0.87     80224
weighted avg       0.87      0.87      0.87     80224
```

In [28]:
```
#Test Train Split
dt = DecisionTreeClassifier(criterion='gini')
dt.fit(trainX, trainY)
y_predict1 = dt.predict(testX)
print(accuracy_score(testY, y_predict1))
print(classification_report(testY, y_predict1))
```

```
0.8540337056242521
              precision    recall  f1-score   support

           0       0.88      0.88      0.88     48896
           1       0.81      0.81      0.81     31328

    accuracy                           0.85     80224
   macro avg       0.85      0.85      0.85     80224
weighted avg       0.85      0.85      0.85     80224
```

In [29]:
```
#Test Train Split
nb = GaussianNB()
nb.fit(trainX, trainY)
y_predict1 = nb.predict(testX)
print(accuracy_score(testY, y_predict1))
print(classification_report(testY, y_predict1))
```

```
0.78567510969286
              precision    recall  f1-score   support

           0       0.77      0.92      0.84     48896
           1       0.82      0.58      0.68     31328

    accuracy                           0.79     80224
   macro avg       0.80      0.75      0.76     80224
weighted avg       0.79      0.79      0.78     80224
```

In [30]:
```
#Test Train Split
svclassifier = SVC(kernel='rbf', gamma='auto')
svclassifier.fit(trainX, trainY)
y_predict = svclassifier.predict(testX)
print(accuracy_score(testY, y_predict))
print(classification_report(testY, y_predict))
```

```
0.8600543478260869
              precision    recall  f1-score   support

           0       0.84      0.95      0.89     48896
           1       0.90      0.72      0.80     31328

    accuracy                           0.86     80224
   macro avg       0.87      0.84      0.85     80224
weighted avg       0.86      0.86      0.86     80224
```

LSTM

In [31]:
```
X_M.shape
```

Out[31]: (401119, 11)

In [32]:
```
X1 = X_M.reshape((-1, 1, 11))
```

In [33]:
```
X1.shape
```

Out[33]: (401119, 1, 11)

In [34]:
```
(trainX, testX, trainY, testY) = train_test_split(X1, y, test_size = 0.20, random_state = 40)
```

In [35]:
```
# 1 LSTM Layer (input), 3 Dense Hidden Layers
model = Sequential()
model.add(LSTM(11, input_shape=(1, 11), activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=["accuracy"])
model.fit(trainX, trainY, epochs=35, batch_size=64, verbose=2, validation_data=(testX, testY))
```

```
Epoch 1/35
5014/5014 - 24s - loss: 0.3700 - accuracy: 0.8703 - val_loss: 0.3512 - val_accuracy: 0.8801 - 24s/epoch - 5ms/step
Epoch 2/35
5014/5014 - 21s - loss: 0.3521 - accuracy: 0.8799 - val_loss: 0.3442 - val_accuracy: 0.8841 - 21s/epoch - 4ms/step
Epoch 3/35
5014/5014 - 19s - loss: 0.3471 - accuracy: 0.8821 - val_loss: 0.3401 - val_accuracy: 0.8832 - 19s/epoch - 4ms/step
Epoch 4/35
5014/5014 - 20s - loss: 0.3433 - accuracy: 0.8839 - val_loss: 0.3406 - val_accuracy: 0.8827 - 20s/epoch - 4ms/step
Epoch 5/35
5014/5014 - 22s - loss: 0.3415 - accuracy: 0.8847 - val_loss: 0.3363 - val_accuracy: 0.8860 - 22s/epoch - 4ms/step
Epoch 6/35
5014/5014 - 20s - loss: 0.3409 - accuracy: 0.8849 - val_loss: 0.3383 - val_accuracy: 0.8848 - 20s/epoch - 4ms/step
Epoch 7/35
5014/5014 - 22s - loss: 0.3397 - accuracy: 0.8855 - val_loss: 0.3359 - val_accuracy: 0.8872 - 22s/epoch - 4ms/step
Epoch 8/35
5014/5014 - 21s - loss: 0.3391 - accuracy: 0.8861 - val_loss: 0.3347 - val_accuracy: 0.8879 - 21s/epoch - 4ms/step
Epoch 9/35
5014/5014 - 21s - loss: 0.3380 - accuracy: 0.8864 - val_loss: 0.3356 - val_accuracy: 0.8864 - 21s/epoch - 4ms/step
Epoch 10/35
```

In [36]:
```
loss, accuracy = model.evaluate(testX, testY)
print("Loss:" + str(loss))
print("Accuracy:" + str(accuracy))
```

```
2507/2507 [==============================] - 7s 2ms/step - loss: 0.3152 - accuracy: 0.8903
Loss:0.31520333886146545
Accuracy:0.8903445601463318
```

In [37]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 11)                1012

 dropout (Dropout)           (None, 11)                0

 dense (Dense)               (None, 128)               1536

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 100)               12900

 dropout_2 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 64)                6464

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 21,977
Trainable params: 21,977
Non-trainable params: 0
_________________________________________________________________
```

Conv-LSTM

In [38]:
```
X_M.shape
```

Out[38]:
```
(401119, 11)
```

In [39]:
```
X1 = X_M.reshape((-1, 1, 11))
```

In [40]:
```
X1.shape
```

Out[40]:
```
(401119, 1, 11)
```

In [41]:
```
(trainX, testX, trainY, testY) = train_test_split(X1, y, test_size = 0.20, random_state = 40)
```

In [42]:
```
nb_filter = 250
filter_length = 3

model = Sequential()
model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, padding='same', activation='tanh'))
model.add(Dropout(0.2))
model.add(MaxPooling1D(pool_size=1))
model.add(Dropout(0.2))
model.add(LSTM(17))
model.add(Dropout(0.2))
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=["accuracy"])
model.fit(trainX, trainY, epochs=35, batch_size=64, verbose=2, validation_data=(testX, testY))
```

```
Epoch 1/35
5014/5014 - 35s - loss: 0.3627 - accuracy: 0.8746 - val_loss: 0.3472 - val_accuracy: 0.8848 - 35s/epoch - 7ms/step
Epoch 2/35
5014/5014 - 30s - loss: 0.3483 - accuracy: 0.8815 - val_loss: 0.3394 - val_accuracy: 0.8844 - 30s/epoch - 6ms/step
Epoch 3/35
5014/5014 - 31s - loss: 0.3437 - accuracy: 0.8834 - val_loss: 0.3353 - val_accuracy: 0.8852 - 31s/epoch - 6ms/step
Epoch 4/35
5014/5014 - 30s - loss: 0.3401 - accuracy: 0.8848 - val_loss: 0.3408 - val_accuracy: 0.8832 - 30s/epoch - 6ms/step
Epoch 5/35
5014/5014 - 31s - loss: 0.3385 - accuracy: 0.8852 - val_loss: 0.3347 - val_accuracy: 0.8857 - 31s/epoch - 6ms/step
Epoch 6/35
5014/5014 - 29s - loss: 0.3381 - accuracy: 0.8857 - val_loss: 0.3345 - val_accuracy: 0.8857 - 29s/epoch - 6ms/step
Epoch 7/35
5014/5014 - 30s - loss: 0.3373 - accuracy: 0.8857 - val_loss: 0.3326 - val_accuracy: 0.8875 - 30s/epoch - 6ms/step
Epoch 8/35
5014/5014 - 31s - loss: 0.3369 - accuracy: 0.8861 - val_loss: 0.3319 - val_accuracy: 0.8875 - 31s/epoch - 6ms/step
Epoch 9/35
5014/5014 - 31s - loss: 0.3367 - accuracy: 0.8860 - val_loss: 0.3330 - val_accuracy: 0.8857 - 31s/epoch - 6ms/step
Epoch 10/35
5014/5014 - 29s - loss: 0.3356 - accuracy: 0.8864 - val_loss: 0.3299 - val_accuracy: 0.8868 - 29s/epoch - 6ms/step
Epoch 11/35
5014/5014 - 29s - loss: 0.3351 - accuracy: 0.8869 - val_loss: 0.3291 - val_accuracy: 0.8873 - 29s/epoch - 6ms/step
Epoch 12/35
5014/5014 - 30s - loss: 0.3339 - accuracy: 0.8872 - val_loss: 0.3272 - val_accuracy: 0.8889 - 30s/epoch - 6ms/step
Epoch 13/35
5014/5014 - 31s - loss: 0.3335 - accuracy: 0.8872 - val_loss: 0.3293 - val_accuracy: 0.8882 - 31s/epoch - 6ms/step
Epoch 14/35
5014/5014 - 31s - loss: 0.3327 - accuracy: 0.8871 - val_loss: 0.3272 - val_accuracy: 0.8868 - 31s/epoch - 6ms/step
Epoch 15/35
5014/5014 - 31s - loss: 0.3320 - accuracy: 0.8872 - val_loss: 0.3255 - val_accuracy: 0.8887 - 31s/epoch - 6ms/step
Epoch 16/35
5014/5014 - 31s - loss: 0.3315 - accuracy: 0.8877 - val_loss: 0.3266 - val_accuracy: 0.8871 - 31s/epoch - 6ms/step
Epoch 17/35
5014/5014 - 30s - loss: 0.3308 - accuracy: 0.8876 - val_loss: 0.3213 - val_accuracy: 0.8898 - 30s/epoch - 6ms/step
Epoch 18/35
5014/5014 - 30s - loss: 0.3307 - accuracy: 0.8875 - val_loss: 0.3233 - val_accuracy: 0.8891 - 30s/epoch - 6ms/step
Epoch 19/35
5014/5014 - 29s - loss: 0.3293 - accuracy: 0.8880 - val_loss: 0.3231 - val_accuracy: 0.8892 - 29s/epoch - 6ms/step
Epoch 20/35
5014/5014 - 30s - loss: 0.3291 - accuracy: 0.8881 - val_loss: 0.3220 - val_accuracy: 0.8879 - 30s/epoch - 6ms/step
Epoch 21/35
5014/5014 - 31s - loss: 0.3283 - accuracy: 0.8884 - val_loss: 0.3225 - val_accuracy: 0.8882 - 31s/epoch - 6ms/step
Epoch 22/35
5014/5014 - 32s - loss: 0.3276 - accuracy: 0.8886 - val_loss: 0.3279 - val_accuracy: 0.8864 - 32s/epoch - 6ms/step
```

```
Epoch 23/35
5014/5014 - 31s - loss: 0.3275 - accuracy: 0.8886 - val_loss: 0.3164 - val_accuracy: 0.8915 - 31s/epoch - 6ms/step
Epoch 24/35
5014/5014 - 29s - loss: 0.3272 - accuracy: 0.8885 - val_loss: 0.3287 - val_accuracy: 0.8821 - 29s/epoch - 6ms/step
Epoch 25/35
5014/5014 - 30s - loss: 0.3271 - accuracy: 0.8886 - val_loss: 0.3174 - val_accuracy: 0.8917 - 30s/epoch - 6ms/step
Epoch 26/35
5014/5014 - 31s - loss: 0.3263 - accuracy: 0.8889 - val_loss: 0.3196 - val_accuracy: 0.8902 - 31s/epoch - 6ms/step
Epoch 27/35
5014/5014 - 31s - loss: 0.3265 - accuracy: 0.8887 - val_loss: 0.3210 - val_accuracy: 0.8885 - 31s/epoch - 6ms/step
Epoch 28/35
5014/5014 - 32s - loss: 0.3263 - accuracy: 0.8884 - val_loss: 0.3166 - val_accuracy: 0.8907 - 32s/epoch - 6ms/step
Epoch 29/35
5014/5014 - 30s - loss: 0.3253 - accuracy: 0.8887 - val_loss: 0.3190 - val_accuracy: 0.8896 - 30s/epoch - 6ms/step
Epoch 30/35
5014/5014 - 29s - loss: 0.3258 - accuracy: 0.8887 - val_loss: 0.3191 - val_accuracy: 0.8895 - 29s/epoch - 6ms/step
Epoch 31/35
5014/5014 - 31s - loss: 0.3248 - accuracy: 0.8889 - val_loss: 0.3175 - val_accuracy: 0.8898 - 31s/epoch - 6ms/step
Epoch 32/35
5014/5014 - 31s - loss: 0.3249 - accuracy: 0.8886 - val_loss: 0.3158 - val_accuracy: 0.8916 - 31s/epoch - 6ms/step
Epoch 33/35
5014/5014 - 30s - loss: 0.3248 - accuracy: 0.8890 - val_loss: 0.3162 - val_accuracy: 0.8917 - 30s/epoch - 6ms/step
Epoch 34/35
5014/5014 - 30s - loss: 0.3247 - accuracy: 0.8890 - val_loss: 0.3196 - val_accuracy: 0.8880 - 30s/epoch - 6ms/step
Epoch 35/35
5014/5014 - 31s - loss: 0.3246 - accuracy: 0.8890 - val_loss: 0.3191 - val_accuracy: 0.8897 - 31s/epoch - 6ms/step
```

Out[42]:
```
<keras.callbacks.History at 0x1a697644a30>
```

In [43]:
```
loss, accuracy = model.evaluate(testX, testY)
print("Loss:" + str(loss))
print("Accuracy:" + str(accuracy))
```

```
2507/2507 [==============================] - 7s 3ms/step - loss: 0.3191 - accuracy: 0.8897
Loss:0.3190644681453705
Accuracy:0.8897212743759155
```

In [44]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 1, 250)            8500

 dropout_4 (Dropout)         (None, 1, 250)            0

 max_pooling1d (MaxPooling1D  (None, 1, 250)           0
 )

 dropout_5 (Dropout)         (None, 1, 250)            0

 lstm_1 (LSTM)               (None, 17)                18224

 dropout_6 (Dropout)         (None, 17)                0

 dense_4 (Dense)             (None, 128)               2304

 dropout_7 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 100)               12900

 dropout_8 (Dropout)         (None, 100)               0

 dense_6 (Dense)             (None, 64)                6464

 dropout_9 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 1)                 65

=================================================================
Total params: 48,457
Trainable params: 48,457
Non-trainable params: 0
_________________________________________________________________
```

IF Conv-LSTM doesn't achieve max accuracy then run this, other

In [45]:
```
nb_filter = 250
filter_length = 3

model = Sequential()
model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, padding='same', activation='tanh'))
model.add(MaxPooling1D(pool_size=1))
model.add(Conv1D(filters=nb_filter, kernel_size=filter_length, padding='same', activation='tanh'))
model.add(MaxPooling1D(pool_size=1))
model.add(Activation('softmax'))
model.add(LSTM(17, return_sequences=True))
model.add(LSTM(17, return_sequences=True))
model.add(Dense(128, activation='tanh'))
model.add(Dense(100, activation='tanh'))
model.add(Dense(64, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=["accuracy"])
model.fit(trainX, trainY, epochs=500, batch_size=100, verbose=2, validation_data=(testX, testY))
```

```
Epoch 1/500
3209/3209 - 34s - loss: 0.3715 - accuracy: 0.8673 - val_loss: 0.3515 - val_accuracy: 0.8797 - 34s/epoch - 11ms/step
Epoch 2/500
3209/3209 - 28s - loss: 0.3453 - accuracy: 0.8822 - val_loss: 0.3397 - val_accuracy: 0.8841 - 28s/epoch - 9ms/step
Epoch 3/500
3209/3209 - 27s - loss: 0.3366 - accuracy: 0.8857 - val_loss: 0.3417 - val_accuracy: 0.8799 - 27s/epoch - 8ms/step
Epoch 4/500
3209/3209 - 28s - loss: 0.3324 - accuracy: 0.8864 - val_loss: 0.3322 - val_accuracy: 0.8887 - 28s/epoch - 9ms/step
Epoch 5/500
3209/3209 - 28s - loss: 0.3289 - accuracy: 0.8870 - val_loss: 0.3273 - val_accuracy: 0.8868 - 28s/epoch - 9ms/step
Epoch 6/500
3209/3209 - 27s - loss: 0.3246 - accuracy: 0.8884 - val_loss: 0.3233 - val_accuracy: 0.8870 - 27s/epoch - 8ms/step
Epoch 7/500
3209/3209 - 28s - loss: 0.3222 - accuracy: 0.8892 - val_loss: 0.3226 - val_accuracy: 0.8878 - 28s/epoch - 9ms/step
Epoch 8/500
3209/3209 - 28s - loss: 0.3208 - accuracy: 0.8896 - val_loss: 0.3223 - val_accuracy: 0.8877 - 28s/epoch - 9ms/step
Epoch 9/500
3209/3209 - 27s - loss: 0.3196 - accuracy: 0.8899 - val_loss: 0.3178 - val_accuracy: 0.8896 - 27s/epoch - 9ms/step
Epoch 10/500
```

In [46]:
```
loss, accuracy = model.evaluate(testX, testY)
```

```
print("Loss:" + str(loss))
print("Accuracy:" + str(accuracy))
```

```
2507/2507 [==============================] - 11s 4ms/step - loss: 0.2994 - accuracy: 0.8950
Loss:0.29940277338027954
Accuracy:0.8950189352035522
```

In [47]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_1 (Conv1D)           (None, 1, 250)            8500

 max_pooling1d_1 (MaxPooling  (None, 1, 250)           0
 1D)

 conv1d_2 (Conv1D)           (None, 1, 250)            187750

 max_pooling1d_2 (MaxPooling  (None, 1, 250)           0
 1D)

 activation (Activation)     (None, 1, 250)            0

 lstm_2 (LSTM)               (None, 1, 17)             18224

 lstm_3 (LSTM)               (None, 1, 17)             2380

 dense_8 (Dense)             (None, 1, 128)            2304

 dense_9 (Dense)             (None, 1, 100)            12900

 dense_10 (Dense)            (None, 1, 64)             6464

 dense_11 (Dense)            (None, 1, 1)              65

=================================================================
Total params: 238,587
Trainable params: 238,587
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
```