In [3]:

```
import pandas as pd
import numpy as np
df = pd.read_csv("Saikumar Pagidipalli../input/musk-dataset/musk_csv.csv")
df.head()
```

Out[3]:

| | ID | molecule_name | conformation_name | f1 | f2 | f3 | f4 | f5 | f6 | f7 | ... | f158 | f159 | f160 | f161 | f162 | f163 | f164 | f165 | f1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | MUSK-211 | 211_1+1 | 46 | -108 | -60 | -69 | -117 | 49 | 38 | ... | -308 | 52 | -7 | 39 | 126 | 156 | -50 | -112 | |
| 1 | 2 | MUSK-211 | 211_1+10 | 41 | -188 | -145 | 22 | -117 | -6 | 57 | ... | -59 | -2 | 52 | 103 | 136 | 169 | -61 | -136 | |
| 2 | 3 | MUSK-211 | 211_1+11 | 46 | -194 | -145 | 28 | -117 | 73 | 57 | ... | -134 | -154 | 57 | 143 | 142 | 165 | -67 | -145 | |
| 3 | 4 | MUSK-211 | 211_1+12 | 41 | -188 | -145 | 22 | -117 | -7 | 57 | ... | -60 | -4 | 52 | 104 | 136 | 168 | -60 | -135 | |
| 4 | 5 | MUSK-211 | 211_1+13 | 41 | -188 | -145 | 22 | -117 | -7 | 57 | ... | -60 | -4 | 52 | 104 | 137 | 168 | -60 | -135 | |

**5 rows × 170 columns**

In [4]:

```
df.describe()
```

Out[4]:

| | ID | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 |
|---|---|---|---|---|---|---|---|---|---|
| count | 6598.00000 | 6598.000000 | 6598.000000 | 6598.000000 | 6598.000000 | 6598.000000 | 6598.000000 | 6598.000000 | 6598.000000 |
| mean | 3299.50000 | 58.945135 | -119.128524 | -73.146560 | -0.628372 | -103.533495 | 18.359806 | -14.108821 | -1.858290 |
| std | 1904.82287 | 53.249007 | 90.813375 | 67.956235 | 80.444617 | 64.387559 | 80.593655 | 115.315673 | 90.372537 |
| min | 1.00000 | -31.000000 | -199.000000 | -167.000000 | -114.000000 | -118.000000 | -183.000000 | -171.000000 | -225.000000 |
| 25% | 1650.25000 | 37.000000 | -193.000000 | -137.000000 | -70.000000 | -117.000000 | -28.000000 | -159.000000 | -85.000000 |
| 50% | 3299.50000 | 44.000000 | -149.000000 | -99.000000 | -25.000000 | -117.000000 | 33.000000 | 27.000000 | 19.000000 |
| 75% | 4948.75000 | 53.000000 | -95.000000 | -19.000000 | 42.000000 | -116.000000 | 74.000000 | 57.000000 | 61.000000 |
| max | 6598.00000 | 292.000000 | 95.000000 | 81.000000 | 161.000000 | 325.000000 | 200.000000 | 220.000000 | 320.000000 |

**8 rows × 168 columns**

In [5]:

```
df.isnull().any()
```

Out[5]:

```
ID                   False
molecule_name        False
conformation_name    False
f1                   False
f2                   False
                     ...
f163                 False
f164                 False
f165                 False
f166                 False
class                False
```

```
Length: 170, dtype: bool
```

In [6]:

```python
df.duplicated()
```

Out[6]:

```
0       False
1       False
2       False
3       False
4       False
        ...
6593    False
6594    False
6595    False
6596    False
6597    False
Length: 6598, dtype: bool
```

In [7]:

```python
corln=df.corr().abs()
corln
```

Out[7]:

| | ID | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | ... | f158 | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ID** | 1.000000 | 0.197844 | 0.119750 | 0.179274 | 0.248703 | 0.106119 | 0.165094 | 0.140705 | 0.430635 | 0.179556 | ... | 0.057309 | 0.046 |
| **f1** | 0.197844 | 1.000000 | 0.142464 | 0.164292 | 0.291054 | 0.001037 | 0.090659 | 0.071879 | 0.416191 | 0.090701 | ... | 0.010485 | 0.146 |
| **f2** | 0.119750 | 0.142464 | 1.000000 | 0.611675 | 0.244145 | 0.134689 | 0.125947 | 0.449526 | 0.187318 | 0.484187 | ... | 0.172481 | 0.309 |
| **f3** | 0.179274 | 0.164292 | 0.611675 | 1.000000 | 0.378516 | 0.080964 | 0.210432 | 0.754798 | 0.426254 | 0.760313 | ... | 0.261200 | 0.502 |
| **f4** | 0.248703 | 0.291054 | 0.244145 | 0.378516 | 1.000000 | 0.044896 | 0.222191 | 0.453786 | 0.185135 | 0.420407 | ... | 0.308906 | 0.607 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **f163** | 0.291805 | 0.061317 | 0.028524 | 0.131828 | 0.182826 | 0.027131 | 0.046881 | 0.131520 | 0.090044 | 0.127985 | ... | 0.067344 | 0.109 |
| **f164** | 0.039315 | 0.142004 | 0.069193 | 0.111005 | 0.189530 | 0.046188 | 0.024317 | 0.049882 | 0.266026 | 0.058630 | ... | 0.010607 | 0.081 |
| **f165** | 0.196997 | 0.443060 | 0.133091 | 0.086150 | 0.364233 | 0.090038 | 0.002557 | 0.000166 | 0.298356 | 0.016455 | ... | 0.056730 | 0.177 |
| **f166** | 0.043655 | 0.057199 | 0.046361 | 0.020434 | 0.072985 | 0.081910 | 0.050493 | 0.004980 | 0.138932 | 0.002590 | ... | 0.036913 | 0.040 |
| **class** | 0.625410 | 0.120883 | 0.099896 | 0.089760 | 0.098592 | 0.045040 | 0.089248 | 0.113093 | 0.201554 | 0.147509 | ... | 0.003181 | 0.021 |

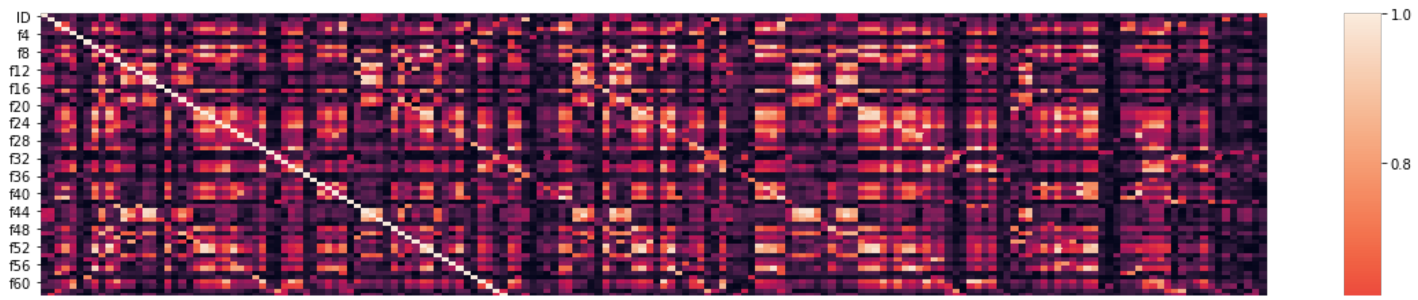
**168 rows × 168 columns**

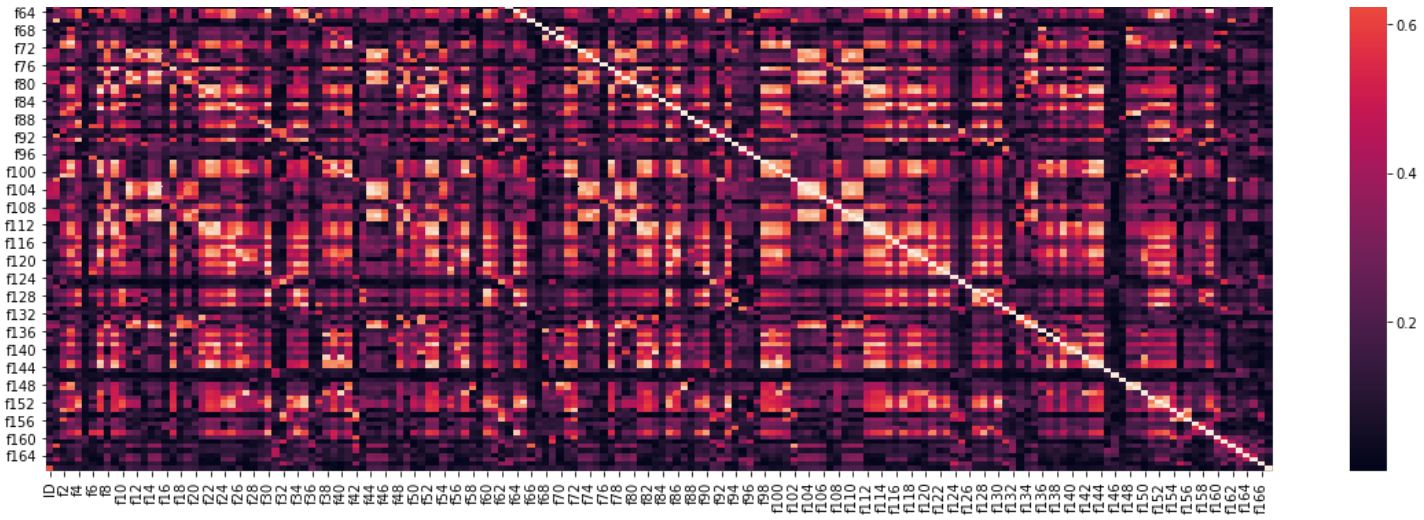
In [8]:

```python
#lets viualize in Heatmap
import seaborn as sns
import matplotlib.pyplot as plt
plt.subplots(figsize=(20,10))
sns.heatmap(corln)
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc481260588>
```






In [ ]:

```
# lets Do the features extraction usong absolute Co-relation matrix,[Greater the value gr
eater the linear relationship]
```

In [9]:

```
# lets reduce the columns which were doesn't require for us using Correlation
columns = np.full((corln.shape[0],), True, dtype=bool)
for i in range(corln.shape[0]):
    for j in range(i+1, corln.shape[0]):
        if corln.iloc[i,j] >= 0.9:
            if columns[j]:
                columns[j] = False
selected_columns = corln.columns[columns]
data = df[selected_columns]
data
```

Out[9]:

| | ID | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | ... | f157 | f158 | f160 | f161 | f162 | f163 | f164 | f165 | f166 | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 46 | -108 | -60 | -69 | -117 | 49 | 38 | -161 | -8 | ... | -244 | -308 | -7 | 39 | 126 | 156 | -50 | -112 | 96 | 1 |
| **1** | 2 | 41 | -188 | -145 | 22 | -117 | -6 | 57 | -171 | -39 | ... | -235 | -59 | 52 | 103 | 136 | 169 | -61 | -136 | 79 | 1 |
| **2** | 3 | 46 | -194 | -145 | 28 | -117 | 73 | 57 | -168 | -39 | ... | -238 | -134 | 57 | 143 | 142 | 165 | -67 | -145 | 39 | 1 |
| **3** | 4 | 41 | -188 | -145 | 22 | -117 | -7 | 57 | -170 | -39 | ... | -236 | -60 | 52 | 104 | 136 | 168 | -60 | -135 | 80 | 1 |
| **4** | 5 | 41 | -188 | -145 | 22 | -117 | -7 | 57 | -170 | -39 | ... | -236 | -60 | 52 | 104 | 137 | 168 | -60 | -135 | 80 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **6593** | 6594 | 51 | -123 | -23 | -108 | -117 | 134 | -160 | 82 | -230 | ... | 62 | -66 | -14 | -29 | 107 | 171 | -44 | -115 | 118 | 0 |
| **6594** | 6595 | 44 | -104 | -19 | -105 | -117 | 142 | -165 | 68 | -225 | ... | 60 | -51 | -9 | 150 | 129 | 158 | -66 | -144 | -5 | 0 |
| **6595** | 6596 | 44 | -102 | -19 | -104 | -117 | 72 | -165 | 65 | -219 | ... | -226 | 90 | -8 | 150 | 130 | 159 | -66 | -144 | -6 | 0 |
| **6596** | 6597 | 51 | -121 | -23 | -106 | -117 | 63 | -161 | 79 | -224 | ... | -238 | 86 | -14 | -31 | 106 | 171 | -44 | -116 | 117 | 0 |
| **6597** | 6598 | 51 | -122 | -23 | -106 | -117 | 190 | -161 | 80 | -227 | ... | 95 | 40 | -14 | -30 | 107 | 171 | -44 | -115 | 118 | 0 |

**6598 rows × 109 columns**

In [10]:

```
#  (We do normalization to reduce and even eliminate data redundancy and Data normalizati
on transforms multiscaled data to the same scale. After normalization
# all variables have a similar influence on the model) using MinMaxScalar
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
x= min_max_scaler.fit_transform(data.iloc[:,0:108])
y = min_max_scaler.fit_transform(data.iloc[:,108:])
```

In [11]:

```
# Now Preparing Data For Algorithm
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(5278, 108) (1320, 108) (5278, 1) (1320, 1)
```

In [12]:

```
# Prepare the Data CNN Deep Learning technique
X_train = np.reshape(x_train,(x_train.shape[0],18,6,1))
X_test= np.reshape(x_test,(x_test.shape[0],18,6,1))
```

In [38]:

```
import keras
y_train = keras.utils.to_categorical(y_train,num_classes = 2)
y_test = keras.utils.to_categorical(y_test,num_classes = 2)
print(y_train.shape,y_test.shape)
```

```
(5278, 2, 2) (1320, 2, 2)
```

In [39]:

```
from sklearn.model_selection import train_test_split
from keras.models import Sequential
import seaborn as sns
from keras.layers import Dense, Dropout, Flatten,Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
```

In [21]:

```
model=Sequential()
model.add(Conv2D(64,kernel_size=(2,2),activation='relu',input_shape=(18,6,1)))
model.add(Conv2D(64,(2,2),activation='relu'))
model.add(Conv2D(64,(2,2),activation='relu'))
model.add(Conv2D(32,(2,2),activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.20))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2,activation='sigmoid'))
```

In [22]:

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

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 17, 5, 64)         320
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 4, 64)         16448
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 15, 3, 64)         16448
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 2, 32)         8224
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 1, 32)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 1, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 224)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               28800
```

```
dropout_4 (Dropout)          (None, 128)              0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                258
=================================================================
Total params: 70,498
Trainable params: 70,498
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
history = model.fit(X_train,y_train,batch_size=128,epochs=20,validation_data=(X_test,y_t
est))
```

```
Train on 5278 samples, validate on 1320 samples
Epoch 1/20
5278/5278 [==============================] - 2s 424us/step - loss: 0.4941 - accuracy: 0.8
115 - val_loss: 0.4119 - val_accuracy: 0.8447
Epoch 2/20
5278/5278 [==============================] - 1s 268us/step - loss: 0.4103 - accuracy: 0.8
462 - val_loss: 0.3806 - val_accuracy: 0.8447
Epoch 3/20
5278/5278 [==============================] - 1s 270us/step - loss: 0.3499 - accuracy: 0.8
461 - val_loss: 0.3090 - val_accuracy: 0.8644
Epoch 4/20
5278/5278 [==============================] - 2s 304us/step - loss: 0.2889 - accuracy: 0.8
773 - val_loss: 0.2506 - val_accuracy: 0.9008
Epoch 5/20
5278/5278 [==============================] - 2s 301us/step - loss: 0.2440 - accuracy: 0.9
002 - val_loss: 0.2133 - val_accuracy: 0.9170
Epoch 6/20
5278/5278 [==============================] - 2s 285us/step - loss: 0.2084 - accuracy: 0.9
146 - val_loss: 0.1781 - val_accuracy: 0.9280
Epoch 7/20
5278/5278 [==============================] - 2s 308us/step - loss: 0.1835 - accuracy: 0.9
254 - val_loss: 0.1579 - val_accuracy: 0.9375
Epoch 8/20
5278/5278 [==============================] - 2s 285us/step - loss: 0.1737 - accuracy: 0.9
334 - val_loss: 0.1592 - val_accuracy: 0.9367
Epoch 9/20
5278/5278 [==============================] - 2s 311us/step - loss: 0.1548 - accuracy: 0.9
429 - val_loss: 0.1263 - val_accuracy: 0.9515
Epoch 10/20
5278/5278 [==============================] - 2s 332us/step - loss: 0.1252 - accuracy: 0.9
540 - val_loss: 0.1604 - val_accuracy: 0.9364
Epoch 11/20
5278/5278 [==============================] - 1s 267us/step - loss: 0.1171 - accuracy: 0.9
552 - val_loss: 0.1015 - val_accuracy: 0.9640
Epoch 12/20
5278/5278 [==============================] - 2s 296us/step - loss: 0.0892 - accuracy: 0.9
635 - val_loss: 0.0762 - val_accuracy: 0.9746
Epoch 13/20
5278/5278 [==============================] - 1s 269us/step - loss: 0.0728 - accuracy: 0.9
742 - val_loss: 0.0506 - val_accuracy: 0.9833
Epoch 14/20
5278/5278 [==============================] - 1s 266us/step - loss: 0.0579 - accuracy: 0.9
793 - val_loss: 0.0474 - val_accuracy: 0.9856
Epoch 15/20
5278/5278 [==============================] - 1s 261us/step - loss: 0.0451 - accuracy: 0.9
828 - val_loss: 0.0298 - val_accuracy: 0.9905
Epoch 16/20
5278/5278 [==============================] - 1s 268us/step - loss: 0.0297 - accuracy: 0.9
896 - val_loss: 0.0302 - val_accuracy: 0.9890
Epoch 17/20
5278/5278 [==============================] - 1s 265us/step - loss: 0.0229 - accuracy: 0.9
930 - val_loss: 0.0172 - val_accuracy: 0.9924
Epoch 18/20
5278/5278 [==============================] - 1s 264us/step - loss: 0.0145 - accuracy: 0.9
964 - val_loss: 0.0104 - val_accuracy: 0.9977
Epoch 19/20
5278/5278 [==============================] - 1s 274us/step - loss: 0.0157 - accuracy: 0.9
950 - val_loss: 0.0087 - val_accuracy: 0.9966
```

```
Epoch 20/20
5278/5278 [==============================] - 1s 264us/step - loss: 0.0097 - accuracy: 0.9
971 - val_loss: 0.0061 - val_accuracy: 0.9992
```

In [24]:

```
score=model.evaluate(X_test,y_test,verbose=0)
print(score)
```

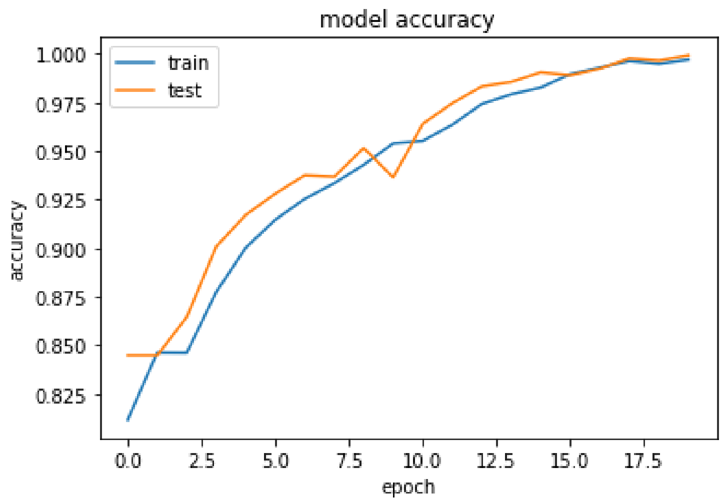
```
[0.006135636661554489, 0.9992424249649048]
```

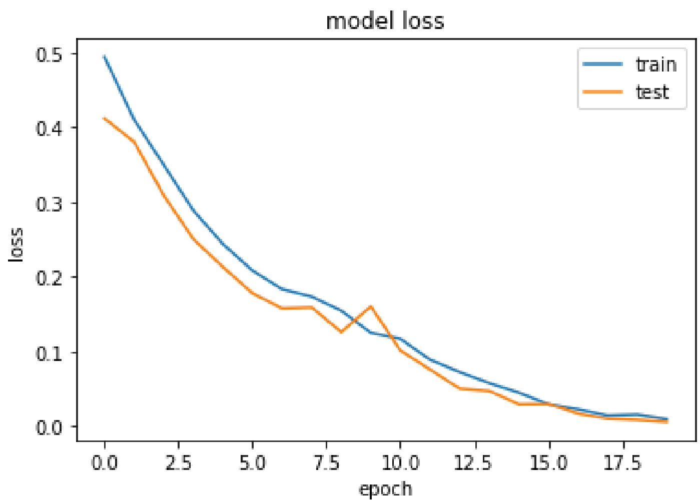
In [25]:

```
%matplotlib inline

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.savefig('accuracy.png',dpi = 100)
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.savefig('loss.png',dpi = 100)
plt.show()
```




In [26]:

```
from sklearn.metrics import f1_score, precision_score, recall_score
```

In [29]:

```
model.predict(X_train[0].reshape(1,18,6,1))
```

Out[29]:

```
array([[9.999981e-01, 1.597790e-06]], dtype=float32)
```

In [ ]:

```
# Model defines 99.9% is Musk and 0.0001% is Non-Musk
```

In [32]:

```
p = model.predict(X_test)
p
```

Out[32]:

```
array([[9.9999321e-01, 6.2685485e-06],
       [9.8603082e-01, 1.4621062e-02],
       [9.9779886e-01, 2.3648716e-03],
       ...,
       [9.9338585e-01, 5.6921756e-03],
       [9.9968612e-01, 3.2398241e-04],
       [9.9997365e-01, 3.0990857e-05]], dtype=float32)
```

In [33]:

```
def preprocess(m):
    a = np.zeros((1320))
    for i in range(1320):
        if m[i][0]>m[i][1]:
            a[i] = 1
        else:
            a[i] = 0
    return a
y_predict = preprocess(yp)
```

In [34]:

```
# our y_test is in 2d shape
test = preprocess(y_test)
print(test.shape,y_predict.shape)
```

```
(1320,) (1320,)
```

In [35]:

```
print("f1_score:",f1_score(test,y_predict))
print("recall:",recall_score(test,y_predict))

print("Validation Loss:",score[0])
print("Validation Accuracy:",score[1])
```

```
f1_score: 0.9995513683266039
recall: 0.9991031390134529
Validation Loss: 0.006135636661554489
Validation Accuracy: 0.9992424249649048
```