In [13]:
```
from preprocess import *
from reprocess import *
from get_data import *
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import load_model
```

数据读取与数据预处理

In [14]:
```
seq_mi, affi_mi, name_mi = load_mi_rich('homo-miRNA.txt', 'original.xls')
affi_n, max_affi, min_affi = normalize(affi_mi)
```

In [16]:
```
seq_r, name_r, affi_r = choose(seq_mi, name_mi, affi_n)
seq_mi_oh = one_hot(seq_r)
```

In [17]:
```
seq_3d = shape2_3d(seq_mi_oh)
```

In [18]:
```
affi_n = affi_n.T
affi_r = affi_r.T
```

In [20]:
```
affi_c = classify(affi_r) # 三级分类
# affi_c = classify2(affi_r) # 二元分类
# affi_c = affi_r #直接求值
```

In [21]:
```
num_train = int(0.8*affi_c.shape[0])
affi_t=np.array(affi_c[:num_train])
seq_t=np.array(seq_3d[:num_train])
affi_tt=np.array(affi_c[num_train:])
seq_tt=np.array(seq_3d[num_train:])
```

检验数据分布

In [8]:
```
r = 0
mr = 0
m = 0
mw = 0
w = 0
for i in range(affi_n.shape[0]):
    if affi_n[i] > 0.75:
        w += 1
    elif affi_n[i] > 0.6:
        mw += 1
    elif affi_n[i] > 0.40:
        m += 1
    elif affi_n[i] > 0.25:
        mr += 1
    else:
```

```
        r += 1

print(w/affi_t.shape[0])
print(mw/affi_t.shape[0])
print(m/affi_t.shape[0])
print(mr/affi_t.shape[0])
print(r/affi_t.shape[0])
```

```
0.24079983841648153
0.22750959402140983
0.4411634013330640З
0.2553019592001616
0.08523530599878812
```

In [9]:

```
r = 0
mr = 0
m = 0
mw = 0
w = 0
for i in range(affi_t.shape[0]):
    if list(affi_t[i]) == [1,0,0]:
        w += 1
    elif list(affi_t[i]) == [0,1,0]:
        m += 1
    elif list(affi_t[i]) == [0,0,1]:
        r += 1
    else:
        mw += 1
print(w/affi_t.shape[0])
print(m/affi_t.shape[0])
print(r/affi_t.shape[0])
print(mw/affi_t.shape[0])
```

```
0.3765703898202383
0.3535043425570592
0.26992526762270247
0.0
```

训练模型

In [46]:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(32,5, strides=1, input_shape=(seq_t.shape[1], seq_t.s
    tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(32, input_shape=(seq_t.shape[1], seq_t.shape[2]), r
    ),
#     tf.keras.layers.SimpleRNN(10,return_sequences=True),
#     tf.keras.layers.SimpleRNN(10,return_sequences=True),
#     tf.keras.layers.SimpleRNN(10,return_sequences=True),
#     tf.keras.layers.SimpleRNN(10,return_sequences=True),
#     tf.keras.layers.Bidirectional(
#          tf.keras.layers.SimpleRNN(32, return_sequences=True)
#     ),
#     tf.keras.layers.Bidirectional(
#          tf.keras.layers.SimpleRNN(32, return_sequences=True)
#     ),
#     tf.keras.layers.Bidirectional(
#          tf.keras.layers.SimpleRNN(32)
#     ),
```

```
    tf.keras.layers.SimpleRNN(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(20),
#     tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(10),
#     tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(3),
#     tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('softmax')
])
```

In [ ]:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(32,5, strides=1, input_shape=(seq_t.shape[1], seq_t.s
    tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(32, input_shape=(seq_t.shape[1], seq_t.shape[2]), r
    ),
    tf.keras.layers.SimpleRNN(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(20),
#     tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(10),
#     tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(3),
#     tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('softmax')
])
```

In [55]:

```
lr = 1.5e-5
iteration = 2
```

In [57]:

```
opt = tf.keras.optimizers.SGD(lr=lr, momentum=0.9)
# model.compile(loss="categorical_crossentropy", optimizer=opt)
model.compile(loss="categorical_crossentropy", optimizer=opt)
# model.fit(x=seq_t, y=affi_t, epochs=iteration)
lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: lr * 10 **
model.fit(x=seq_t, y=affi_t, epochs=iteration, callbacks=[lr_schedule])
```

```
Epoch 1/2
774/774 [==============================] - 16s 18ms/step - loss: 0.3456
Epoch 2/2
774/774 [==============================] - 14s 18ms/step - loss: 0.3335
```

Out[57]:

```
<tensorflow.python.keras.callbacks.History at 0x3ef8e2be0>
```

读取已有模型或存储新模型

In [22]:

```
# model.save('./models/32conv1d5_biLSTM32_biRNN32_withBN_one_hot_enriched.h5')
model = load_model('./models/32conv1d5_biLSTM32_biRNN32_withBN_one_hot_enriched.
# model = load_model('./models/biLSTM32_biRNN32_withBN_mse_enriched.h5') # 直接求
# model = load_model('./models/biLSTM32_biRNN32_withBN_bin.h5') # 二元分类
```

In [20]:

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 19, 32)            672       
_________________________________________________________________
bidirectional_3 (Bidirection (None, 19, 64)            16640     
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 32)                3104      
_________________________________________________________________
batch_normalization_12 (Batc (None, 32)                128       
_________________________________________________________________
dense_18 (Dense)             (None, 32)                1056      
_________________________________________________________________
batch_normalization_13 (Batc (None, 32)                128       
_________________________________________________________________
dense_19 (Dense)             (None, 32)                1056      
_________________________________________________________________
batch_normalization_14 (Batc (None, 32)                128       
_________________________________________________________________
dense_20 (Dense)             (None, 32)                1056      
_________________________________________________________________
batch_normalization_15 (Batc (None, 32)                128       
_________________________________________________________________
dense_21 (Dense)             (None, 20)                660       
_________________________________________________________________
dense_22 (Dense)             (None, 10)                210       
_________________________________________________________________
dense_23 (Dense)             (None, 3)                 33        
_________________________________________________________________
activation_3 (Activation)    (None, 3)                 0         
=================================================================
Total params: 24,999
Trainable params: 24,743
Non-trainable params: 256
_________________________________________________________________
```

三级分类的错误率

In [24]:

```
# er,idx = one_hot_check(seq_tt, affi_tt, model, gate=1)
er, idx = one_hot_check(seq_tt, affi_tt, model, 0.3)
```

```
0.08062691872677331
```

二元分类的错误率

In [12]:

```
er, idx = bin_check(seq_tt, affi_tt, model)
```

```
0.2809823881079334
```

直接求值分类的错误率

In [23]:

```
er, idx = mse_check(seq_tt, affi_tt, model)
```

```
0.792373566004201
```

求预测矩阵

In [11]:

```
probas = model.predict(seq_tt)
```

ROC作图

In [ ]:

```
high_hat = list(probas[:, 0])
mid_hat = list(probas[:, 1])
low_hat = list(probas[:, 2])
high_real = list(affi_tt[:, 0])
mid_real = list(affi_tt[:, 1])
low_real = list(affi_tt[:, 2])
```

In [13]:

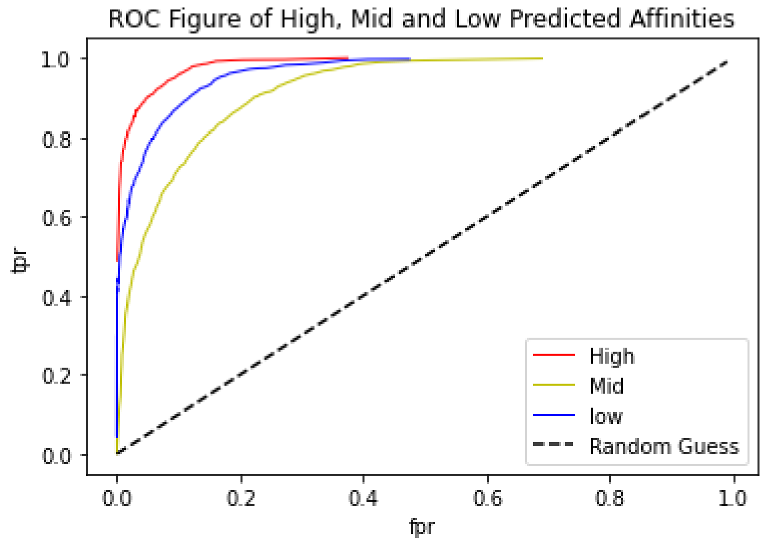
```
fpr_high, tpr_high = tf_judge(high_hat, high_real)
fpr_mid, tpr_mid = tf_judge(mid_hat, mid_real)
fpr_low, tpr_low = tf_judge(low_hat, low_real)
```

In [14]:

```
import matplotlib.pyplot as plt
```

In [15]:

```
plt.figure()
plt.plot(fpr_high, tpr_high, 'r', label="High",linewidth=1)
plt.plot(fpr_mid, tpr_mid, 'y', label="Mid",linewidth=1)
plt.plot(fpr_low, tpr_low, 'b', label="low",linewidth=1)
plt.plot(list(np.arange(0,1,0.01)), list(np.arange(0,1,0.01)), 'k--', label = 'R
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC Figure of High, Mid and Low Predicted Affinities")
plt.legend()
plt.savefig('./results/roc_hml.jpg')
```



读取pirna数据

In [3]:
```
data_pi = np.load('./data/pir_name_seq.npy', allow_pickle=True)
```

In [4]:
```
seq_pi = data_pi[0]['seq']
name_pi = data_pi[0]['name']
max_affi = data_pi[0]['max_affi']
min_affi = data_pi[0]['min_affi']
```

In [14]:
```
window_size = 23
```

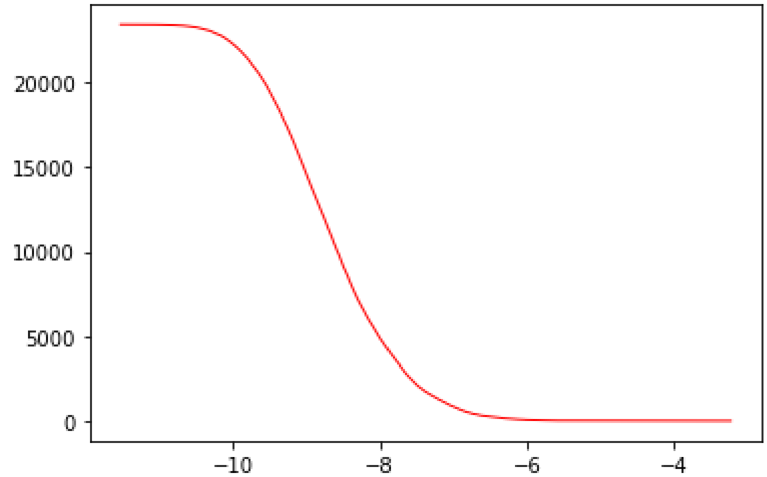
生成pirna结合度

In [ ]:
```
# high, mid, low, y = pi_check(seq_pi, name_pi, model, 23)
affi_hat, y = pi_check(seq_pi, name_pi, max_affi, min_affi, model, 23)
```

In [11]:
```
import matplotlib.pyplot as plt
```

In [17]:
```
plt.plot(sorted(affi_hat)[::-1],range(len(affi_hat)),'r',label="Predict",linewid
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x16079d2e0>]
```



In [37]:
```
with open('./results/results_affi_of_pi.json') as f:
    res = json.load(f)
affi_pi = [float(cache['affinity']) for cache in res]
```

In [56]:
```
gates = list(np.arange(float(format(min(affi_pi),'.1f')) - 0.1,float(format(max(
freq_affi = [sum([int((affi_pi[i] >= gates[j]) & (affi_pi[i] < gates[j+1])) for
```

In [57]:
```
gates = [gates[i] for i in range(len(freq_affi)) if freq_affi[i] != 0]
freq = [freq_affi[i]/sum(freq_affi) for i in range(len(freq_affi)) if freq_affi[
```

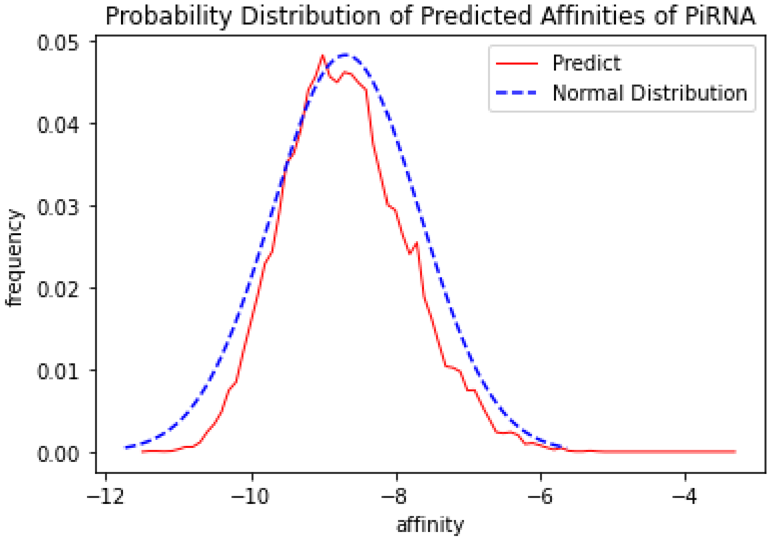
In [62]:
```
sum(freq) * sum(freq_affi)
```

Out[62]: 23437.999999999996

In [80]:
```
import math
u = sum(affi_pi)/len(affi_pi)  # 均值μ
sig = math.sqrt((max(affi_pi) - min(affi_pi))/8)  # 标准差δ
x = np.linspace(u - 3*sig, u + 3*sig, sum(freq_affi))   # 定义域
y = np.exp(-(x - u) ** 2 / (2 * sig ** 2)) / (math.sqrt(2*math.pi)*sig)
y = y / max(y) *max(freq)
```

In [ ]:
```
lost = (y)
```

In [82]:
```
plt.plot(gates,freq,'r',label="Predict",linewidth=1)
plt.plot(x,y,'b--', label='Normal Distribution')
plt.xlabel("affinity")
plt.ylabel("frequency")
plt.title("Probability Distribution of Predicted Affinities of PiRNA")
plt.legend()
plt.savefig('./results/Prob_Dis.jpg')
```


Probability Distribution of Predicted Affinities of PiRNA
Predict
Normal Distribution
frequency
affinity
0.05
0.04
0.03
0.02
0.01
0.00
−12
−10
−8
−6
−4

In [ ]: