In [178]:
```python
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

df = pd.read_excel (r'C:\Users\Agam Chopra\usable_data_final_train.xlsx')
tf = pd.read_excel (r'C:\Users\Agam Chopra\usable_data_final_test.xlsx')
#print (df)
Data = df.values
Test = tf.values
print(Data.shape,Test.shape)
```

```
(15154, 43) (956, 43)
```

In [3]:
```python
def remove_outliers(arr, k):
    mu, sigma = np.mean(arr, axis=0), np.std(arr, axis=0, ddof=1)
    return arr[np.all(np.abs((arr - mu) / sigma) < k, axis=1)]

def remove_outliers_bis(arr, k):
    mask = np.ones((arr.shape[0],), dtype=np.bool)
    mu, sigma = np.mean(arr, axis=0), np.std(arr, axis=0, ddof=1)
    for j in range(arr.shape[1]-1):
        col = arr[:, j]
        mask[mask] &= np.abs((col[mask] - mu[j]) / sigma[j]) < k
    return arr[mask]

#Clean_Data = remove_outliers(Data, 1)

#print(Clean_Data.shape)

Clean_Data = remove_outliers(Data, 2)
Clean_Test = remove_outliers(Test, 2)
print(Clean_Data.shape,Clean_Test.shape)
```

```
(9007, 43) (524, 43)
```

In [4]:
```python
epsilon = 1e-12

neurons = np.array([10,5,1])
layers = neurons.size
print("number of layers: " + str(layers))
print("number of hidden layers: " + str(layers-1))
print(neurons)

input_features = len(Clean_Data[0])
examples = len(Clean_Data)
Y = Clean_Data[:,input_features-1]
Y = Y.reshape((examples,1))
X = np.delete(Clean_Data, input_features-1, axis=1)
X = (X - X.min(axis=0)) / (X.max(axis=0)- X.min(axis=0))
X = X.T


test_features = len(Clean_Test[0])
test_examples = len(Clean_Test)
Ytest = Clean_Test[:,test_features-1]
Ytest = Ytest.reshape((test_examples,1))
Xtest = np.delete(Clean_Test, test_features-1, axis=1)
Xtest = (Xtest - Xtest.min(axis=0)) / (Xtest.max(axis=0)- Xtest.min(axis=0))
Xtest = Xtest.T

print("number of input features + example column: ",input_features," and number of test features + example co
lumn: ",test_features)
print("number of usable examples: " ,examples," and number of usable test examples: ",test_examples)

print("shape of X and Xtest: ",X.shape,Xtest.shape)
print("shape of Y and Ytest: ",Y.shape,Ytest.shape)
```

```
number of layers: 3
number of hidden layers: 2
[10  5  1]
number of input features + example column:  43  and number of test features + example column:  43
number of usable examples:  9007  and number of usable test examples:  524
shape of X and Xtest:  (42, 9007) (42, 524)
shape of Y and Ytest:  (9007, 1) (524, 1)
```

In [125]:
```python
def sigmoid(Z):

    Z = np.round(Z,20)
    A = (1 / (1 + np.exp(-Z) + epsilon))
    activation_cache = Z

    return A, activation_cache

def leaky_relu(Z):

    Z = np.where(Z == 0, epsilon, Z)
    A = np.maximum(0.01*Z,Z)
    activation_cache = Z

    return A, activation_cache

def swish(Z):

    Z = np.where(Z ==np.nan, 0, Z)
    Z = np.round(Z,20)
    A = (Z / (1 + np.exp(-Z) + epsilon))
    activation_cache = Z

    return A, activation_cache

def sigmoid_backward(dA, activation_cache):

    A, Z = sigmoid(activation_cache)

    Z = np.where(Z ==np.nan, 0, Z)
    Z = np.round(Z,20)
    dZ = dA / (A - np.square(A)+epsilon)
    #dZ = dA*((2*epsilon)/(sigmoid(Z+epsilon)-sigmoid(Z-epsilon)))
    dZ = np.round(dZ,20)

    return dZ

def leaky_relu_backward(dA, activation_cache):

    dZ = dA*((2*epsilon)/(leaky_relu(Z+epsilon)-leaky_relu(Z-epsilon)))

    return dZ
```

```python
def swish_backward(dA, activation_cache):

    A, Z = swish(activation_cache)

    Z = np.where(Z ==np.nan, 0, Z)
    Z = np.round(Z,20)
    dZ = (Z*dA) / ((A*((A*np.exp(-Z))+1))+epsilon)
    #dZ = dA*((2*epsilon)/(swish(Z+epsilon)-swish(Z-epsilon)))
    dZ = np.round(dZ,20)

    return dZ

n_x = input_features - 1
n_y = examples
n_x_test = test_features - 1
n_y_test = test_examples
n_h = neurons
L = layers

def initialize_parameters(n_x,n_h,L,n_y):
    parameters = {}

    for i in range(0,L):
        if i == 0:
            parameters['W' + str(1)] = np.random.randn(n_h[0],n_x)*(1/n_x)
            parameters['b' + str(1)] = np.zeros((n_h[0],1))
            assert (parameters['W' + str(1)].shape == (n_h[0],n_x))
            assert (parameters['b' + str(1)].shape == (n_h[0],1))
        else:
            parameters['W' + str(i+1)] = np.random.randn(n_h[i],n_h[i-1])*(1/n_h[i-1])
            parameters['b' + str(i+1)] = np.zeros((n_h[i],1))
            assert (parameters['W' + str(i+1)].shape == (n_h[i],n_h[i-1]))
            assert (parameters['b' + str(i+1)].shape == (n_h[i],1))

        #print("initialized layer ",i+1)

    return parameters

def linear_forward(A, W, b):

    Z = np.dot(W,A)+b
    assert(Z.shape == (W.shape[0], A.shape[1]))
```

```
    cache = (A, W, b)

    return Z, cache

def linear_activation_forward(A_prev, W, b, activation):

    Z, linear_cache = linear_forward(A_prev, W, b)

    if activation == "sigmoid":
        A, activation_cache = sigmoid(Z)

    elif activation == "relu":
        A, activation_cache = leaky_relu(Z)

    elif activation == "swish":
        A, activation_cache = swish(Z)

    else:
        A = Z
        activation_cache = Z

    assert (A.shape == (W.shape[0], A_prev.shape[1]))

    cache = (linear_cache, activation_cache)
    return A, cache

def forward_prop(X, parameters):

    caches = []

    A = X

    L = int(len(parameters)/2)

    for l in range(0, L):
        A_prev = A
        A, cache = linear_activation_forward(A_prev, parameters['W' + str(l+1)],parameters['b' + str(l+1)],
'swish')
        caches.append(cache)

    AL, cache = linear_activation_forward(A_prev, parameters['W' + str(L)],parameters['b' + str(L)], 'sigmoi
d')
```

```
    assert(AL.shape == (1,X.shape[1]))

    return(AL,caches)

def compute_cost(AL, Y):

    AL = np.round(AL,20)
    m = n_y

    cost = (-1/m)*(np.dot(Y.T,np.log(AL.T))+np.dot((1-Y.T),(np.log(1-AL.T))))

    cost = np.squeeze(cost)

    assert(cost.shape == ())

    return cost

def linear_backward(dZ, cache):
    A_prev, W, b = cache
    m = A_prev.shape[1]

    dW = (1/m)*(np.dot(dZ,A_prev.T))
    db = (1/m)*np.sum(dZ, axis = 1, keepdims = True)
    dA_prev = np.dot(W.T,dZ)

    assert (dA_prev.shape == A_prev.shape)
    assert (dW.shape == W.shape)
    assert (db.shape == b.shape)

    return dA_prev, dW, db

def linear_activation_backward(dA, cache, activation):
    linear_cache, activation_cache = cache

    if activation == "relu":
        dZ = relu_backward(dA, activation_cache)

    elif activation == "sigmoid":
        dZ = sigmoid_backward(dA, activation_cache)

    elif activation == "swish":
        dZ = swish_backward(dA, activation_cache)
```

```python
    dA_prev, dW, db = linear_backward(dZ, linear_cache)

    return dA_prev, dW, db

def back_prop(AL, Y, caches):

    grads = {}
    L = len(caches)
    m = AL.shape[1]
    Y = Y.reshape(AL.shape)

    dAL = - (np.divide(Y,AL)) - np.divide(1-Y, 1-AL)

    current_cache = caches[L-1]
    grads["dA"+str(L)],grads["dW"+str(L)],grads["db"+str(L)] = linear_activation_backward(dAL, current_cache, 
"sigmoid")

    for l in reversed(range(L-1)):
        current_cache = caches[l]
        dA_prev_temp, dW_temp, db_temp = linear_activation_backward(grads["dA"+str(l+2)], current_cache, "swi
sh")
        grads["dA"+str(l+1)] = dA_prev_temp
        grads["dW"+str(l+1)] = dW_temp
        grads["db"+str(l+1)] = db_temp

    return grads

def initialize_adam(parameters):
    L = int(len(parameters)/2)
    v = {}
    s = {}

    for l in range(L):
        v["dW" + str(l+1)] = np.zeros((parameters["W" + str(l+1)].shape))
        v["db" + str(l+1)] = np.zeros((parameters["b" + str(l+1)].shape))
        s["dW" + str(l+1)] = np.zeros((parameters["W" + str(l+1)].shape))
        s["db" + str(l+1)] = np.zeros((parameters["b" + str(l+1)].shape))

    return v, s

def update_parameters(parameters, grads, learning_rate):

    L = int(len(parameters)/2)
```

```python
    for l in range(0, L):
        parameters["W" + str(l+1)] = parameters["W" + str(l+1)] - learning_rate*grads["dW" + str(l+1)]
        parameters["b" + str(l+1)] = parameters["b" + str(l+1)] - learning_rate*grads["db" + str(l+1)]

    return parameters

def update_parameters_adam(parameters, grads, v, s, t, learning_rate):

    beta1 = 0.9
    beta2 = 0.999
    L = int(len(parameters)/2)
    v_corrected = {}
    s_corrected = {}

    for l in range(0, L):
        v["dW" + str(l+1)] = beta1 * v["dW" + str(l+1)] + (1-beta1) * grads['dW' + str(l+1)]
        v["db" + str(l+1)] = beta1 * v["db" + str(l+1)] + (1-beta1) * grads['db' + str(l+1)]
        v_corrected["dW" + str(l+1)] = v["dW" + str(l+1)] / (1 - np.power(beta1,t))
        v_corrected["db" + str(l+1)] = v["db" + str(l+1)] / (1 - np.power(beta1,t))

        s["dW" + str(l+1)] = beta2 * s["dW" + str(l+1)] + (1-beta2) * np.power(grads['dW' + str(l+1)],2)
        s["db" + str(l+1)] = beta2 * s["db" + str(l+1)] + (1-beta2) * np.power(grads['db' + str(l+1)],2)
        s_corrected["dW" + str(l+1)] = s["dW" + str(l+1)] / (1 - np.power(beta2,t))
        s_corrected["db" + str(l+1)] = s["db" + str(l+1)] / (1 - np.power(beta2,t))

        parameters["W" + str(l+1)] = parameters["W" + str(l+1)] - learning_rate * (v_corrected["dW" + str(l+1
)] / (np.sqrt(s_corrected["dW" + str(l+1)]) + epsilon))
        parameters["b" + str(l+1)] = parameters["b" + str(l+1)] - learning_rate * (v_corrected["db" + str(l+1
)] / (np.sqrt(s_corrected["db" + str(l+1)]) + epsilon))

    return parameters, v, s

def L_layer_model(X, Y, n_x,n_h,L,n_y, learning_rate = 0.0075, num_iterations = 3000, decay = 0, print_cost=F
alse,dky = False):

    costs = []
    parameters = initialize_parameters(n_x,n_h,L,n_y)
    v,s = initialize_adam(parameters)
    t = 0

    for i in range(0, num_iterations):
```

```
        t = t+1
        AL, caches = forward_prop(X, parameters)
        cost = compute_cost(AL, Y)-0.29
        grads = back_prop(AL, Y, caches)

        parameters, v, s = update_parameters_adam(parameters, grads,v,s,t, learning_rate/(1 + i*decay))

        costs.append(cost-0.29)

        if print_cost and i % 50 == 0:
            print ("Cost after iteration %i: %f" %(i, cost-0.29))

    plt.plot(np.squeeze(costs))
    plt.ylabel('cost')
    plt.xlabel('iterations')
    plt.title("Learning rate =" + str(learning_rate) +" / (1 + i *" + str(decay) + ")")
    plt.show()

    return parameters,AL

def model_run(X, parameters):

    AL, caches = forward_prop(X, parameters)

    return AL
```

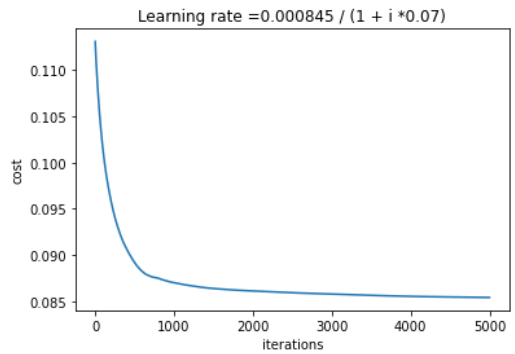
In [127]:

```
#parameters,AL = L_layer_model(X, Y, n_x,n_h,L,n_y, learning_rate = 0.01, num_iterations = 1500,decay =1.2, p
rint_cost=True)
parameters,AL = L_layer_model(X, Y, n_x,n_h,L,n_y, learning_rate = 0.000845, num_iterations = 5000,decay =0.0
7, print_cost=True,dky=True)
ALtest = model_run(Xtest, parameters)
```

```
Cost after iteration 0: 0.113093
Cost after iteration 50: 0.105824
Cost after iteration 100: 0.101263
Cost after iteration 150: 0.098146
Cost after iteration 200: 0.095857
Cost after iteration 250: 0.094055
Cost after iteration 300: 0.092656
Cost after iteration 350: 0.091522
Cost after iteration 400: 0.090628
Cost after iteration 450: 0.089853
Cost after iteration 500: 0.089191
Cost after iteration 550: 0.088629
Cost after iteration 600: 0.088212
Cost after iteration 650: 0.087888
Cost after iteration 700: 0.087706
Cost after iteration 750: 0.087593
Cost after iteration 800: 0.087498
Cost after iteration 850: 0.087344
Cost after iteration 900: 0.087216
Cost after iteration 950: 0.087098
Cost after iteration 1000: 0.087006
Cost after iteration 1050: 0.086919
Cost after iteration 1100: 0.086844
Cost after iteration 1150: 0.086771
Cost after iteration 1200: 0.086704
Cost after iteration 1250: 0.086643
Cost after iteration 1300: 0.086573
Cost after iteration 1350: 0.086517
Cost after iteration 1400: 0.086468
Cost after iteration 1450: 0.086424
Cost after iteration 1500: 0.086384
Cost after iteration 1550: 0.086350
Cost after iteration 1600: 0.086318
Cost after iteration 1650: 0.086286
Cost after iteration 1700: 0.086257
Cost after iteration 1750: 0.086229
Cost after iteration 1800: 0.086203
Cost after iteration 1850: 0.086179
Cost after iteration 1900: 0.086161
Cost after iteration 1950: 0.086136
Cost after iteration 2000: 0.086119
Cost after iteration 2050: 0.086103
Cost after iteration 2100: 0.086095
```

```
Cost after iteration 2150: 0.086069
Cost after iteration 2200: 0.086052
Cost after iteration 2250: 0.086030
Cost after iteration 2300: 0.086017
Cost after iteration 2350: 0.085992
Cost after iteration 2400: 0.085969
Cost after iteration 2450: 0.085950
Cost after iteration 2500: 0.085931
Cost after iteration 2550: 0.085913
Cost after iteration 2600: 0.085896
Cost after iteration 2650: 0.085886
Cost after iteration 2700: 0.085872
Cost after iteration 2750: 0.085857
Cost after iteration 2800: 0.085840
Cost after iteration 2850: 0.085825
Cost after iteration 2900: 0.085807
Cost after iteration 2950: 0.085797
Cost after iteration 3000: 0.085795
Cost after iteration 3050: 0.085786
Cost after iteration 3100: 0.085773
Cost after iteration 3150: 0.085761
Cost after iteration 3200: 0.085741
Cost after iteration 3250: 0.085725
Cost after iteration 3300: 0.085708
Cost after iteration 3350: 0.085692
Cost after iteration 3400: 0.085677
Cost after iteration 3450: 0.085663
Cost after iteration 3500: 0.085649
Cost after iteration 3550: 0.085635
Cost after iteration 3600: 0.085622
Cost after iteration 3650: 0.085610
Cost after iteration 3700: 0.085599
Cost after iteration 3750: 0.085588
Cost after iteration 3800: 0.085576
Cost after iteration 3850: 0.085565
Cost after iteration 3900: 0.085554
Cost after iteration 3950: 0.085545
Cost after iteration 4000: 0.085535
Cost after iteration 4050: 0.085528
Cost after iteration 4100: 0.085520
Cost after iteration 4150: 0.085510
Cost after iteration 4200: 0.085502
Cost after iteration 4250: 0.085497
```

```
Cost after iteration 4300: 0.085493
Cost after iteration 4350: 0.085487
Cost after iteration 4400: 0.085481
Cost after iteration 4450: 0.085475
Cost after iteration 4500: 0.085469
Cost after iteration 4550: 0.085463
Cost after iteration 4600: 0.085458
Cost after iteration 4650: 0.085453
Cost after iteration 4700: 0.085448
Cost after iteration 4750: 0.085445
Cost after iteration 4800: 0.085442
Cost after iteration 4850: 0.085431
Cost after iteration 4900: 0.085424
Cost after iteration 4950: 0.085422
```

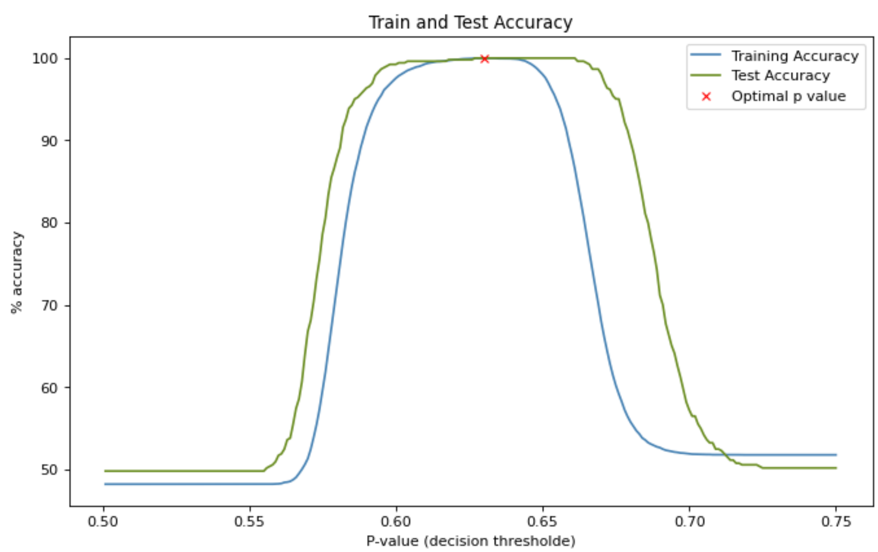


In [7]:
```
At,Atest = AL, ALtest
```

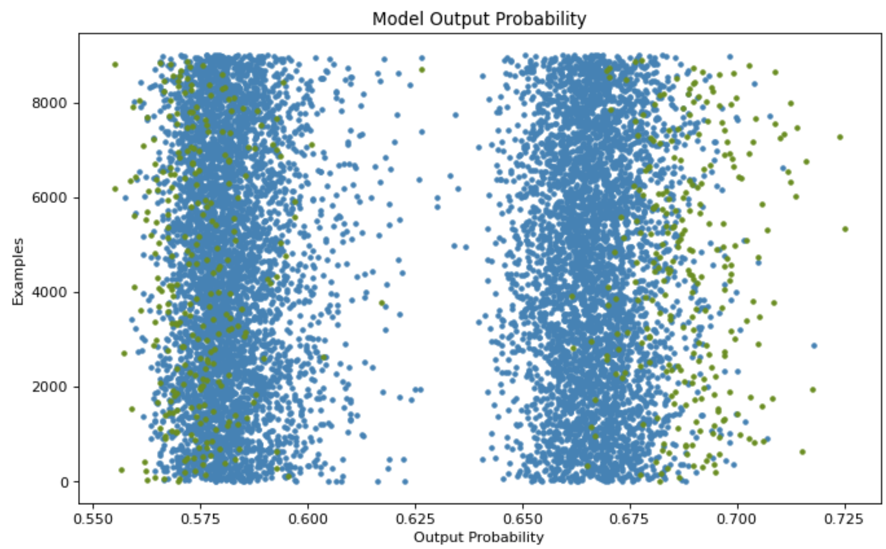
In [8]:
```
import csv
filename = "prediction_record3.csv"

# writing to csv file
with open(filename, 'w') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the data rows
    csvwriter.writerows(At)
    csvwriter.writerows(Y.T)
    csvwriter.writerows(Atest)
    csvwriter.writerows(Ytest.T)
```

In [190]:
```
prediction = np.array(At.T)
prediction_test = np.array(Atest.T)
bias = 0
p = 0.5
p_val = []
train_accuracy = []
test_accuracy = []

for acc in range(0,250):
    p = p+0.001
    p_val.append(p+bias)
    for i in range(0, n_y):
        if(At.T[i] > p):
            prediction[i] = 1
        else:
            prediction[i] = 0
    count = 0
    for i in range(0, n_y):
        if(np.round(Y[i],0)==np.round(prediction[i],0)):
            count = count + 1
    accuracy = 100 * (count / n_y)
    #print(accuracy,"train accuracy for p value", p)
    #print(count)
    train_accuracy.append(accuracy)

    for i in range(0, n_y_test):
        if(Atest.T[i] > p):
            prediction_test[i] = 1
        else:
            prediction_test[i] = 0
    count = 0
    for i in range(0, n_y_test):
        if(np.round(Ytest[i],0)==np.round(prediction_test[i],0)):
            count = count + 1
    accuracy = 100 * (count / n_y_test)
    #print(accuracy,"% test accuracy for p value", p)
    #print(count)
    test_accuracy.append(accuracy)
```

In [185]:
```
from matplotlib.pyplot import figure
figure(num=None, figsize=(10, 6), dpi=80, facecolor='w', edgecolor='k')
plt.plot(p_val,train_accuracy,"-",label='Training Accuracy',linewidth=1.5,c=mcolors.CSS4_COLORS["steelblue"])
plt.plot(p_val,test_accuracy,"-",label='Test Accuracy',linewidth=1.5,c=mcolors.CSS4_COLORS["olivedrab"])
plt.plot(0.63,100,"rx",label='Optimal p value')
plt.ylabel('% accuracy')
plt.xlabel('P-value (decision thresholde)')
plt.title("Train and Test Accuracy")
plt.legend()
plt.show()
```



In [189]:
```
from matplotlib.pyplot import figure
figure(num=None, figsize=(10, 6), dpi=80, facecolor='w', edgecolor='k')

for i in range(9007):
    #plt.plot(0.63,i,'m.')
    plt.plot(At.T[i],i,'.',c=mcolors.CSS4_COLORS["steelblue"])

for i in range(524):
    plt.plot(Atest.T[i],i*17,'.',c=mcolors.CSS4_COLORS["olivedrab"])

plt.ylabel('Examples')
plt.xlabel('Output Probability')
plt.title("Model Output Probability")
plt.show()
plt.show()
```



In [187]:
```
true_train = At.T * Y
false_train = At.T - true_train
from matplotlib.pyplot import figure
figure(num=None, figsize=(10, 6), dpi=80, facecolor='w', edgecolor='k')

for i in range(9007):
    plt.plot(0.63,i,'.',c=mcolors.CSS4_COLORS["slategray"])
    if true_train[i] != 0:
        plt.plot(true_train[i],i,'.',c=mcolors.CSS4_COLORS["teal"])
    else:
        plt.plot(false_train[i],i,'.',c=mcolors.CSS4_COLORS["firebrick"])
plt.ylabel('Example #')
plt.xlabel('Output Probability')
plt.title("Training Decision Output")
plt.show()
```

In [181]:

```
true_test = Atest.T * Ytest
false_test = Atest.T - true_test
from matplotlib.pyplot import figure
figure(num=None, figsize=(10, 6), dpi=80, facecolor='w', edgecolor='k')

for i in range(524):
    plt.plot(0.63,i,'.',c=mcolors.CSS4_COLORS["slategray"])
    if true_test[i] != 0:
        plt.plot(true_test[i],i,'.',c=mcolors.CSS4_COLORS["teal"])
    else:
        plt.plot(false_test[i],i,'.',c=mcolors.CSS4_COLORS["firebrick"])
plt.ylabel('Example #')
plt.xlabel('Output Probability')
plt.title("Test Decision Output")
plt.show()
```