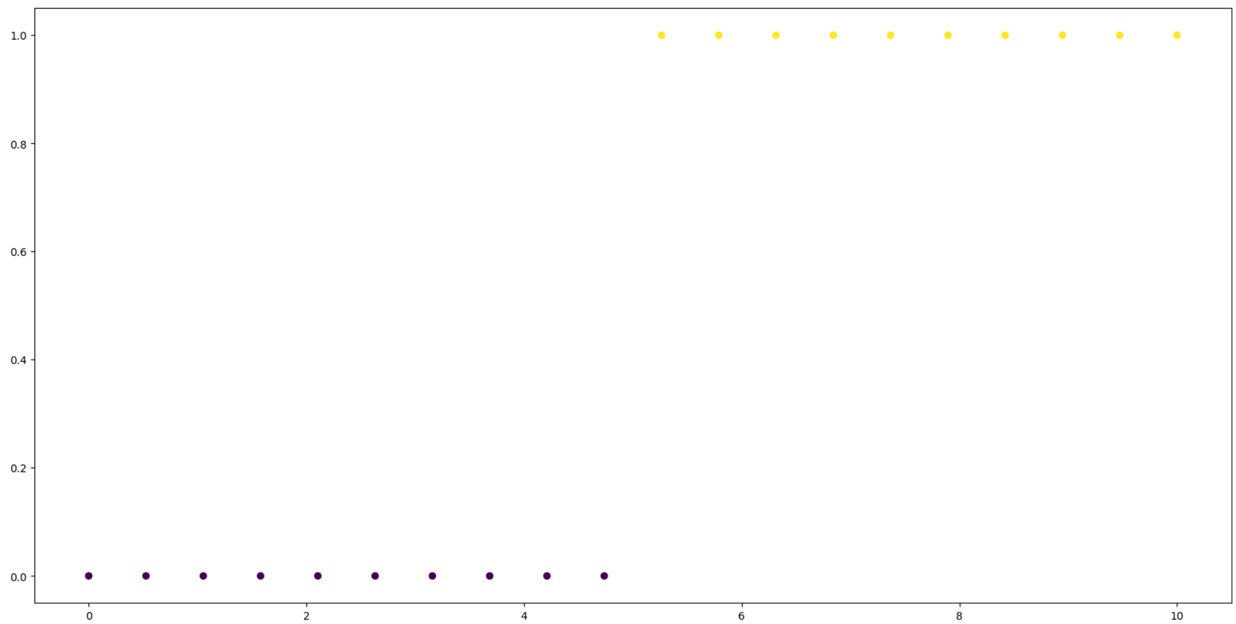
Assignment is at the bottom!

In [12]:
```
from sklearn.linear_model import LogisticRegression
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

from pylab import rcParams
rcParams['figure.figsize'] = 20, 10


from sklearn.linear_model import LogisticRegression as Model
```

In [2]:
```
y = np.concatenate([np.zeros(10), np.ones(10)])
x = np.linspace(0, 10, len(y))
```

In [3]:
```
plt.scatter(x, y, c=y)
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x262e4aa0130>
```



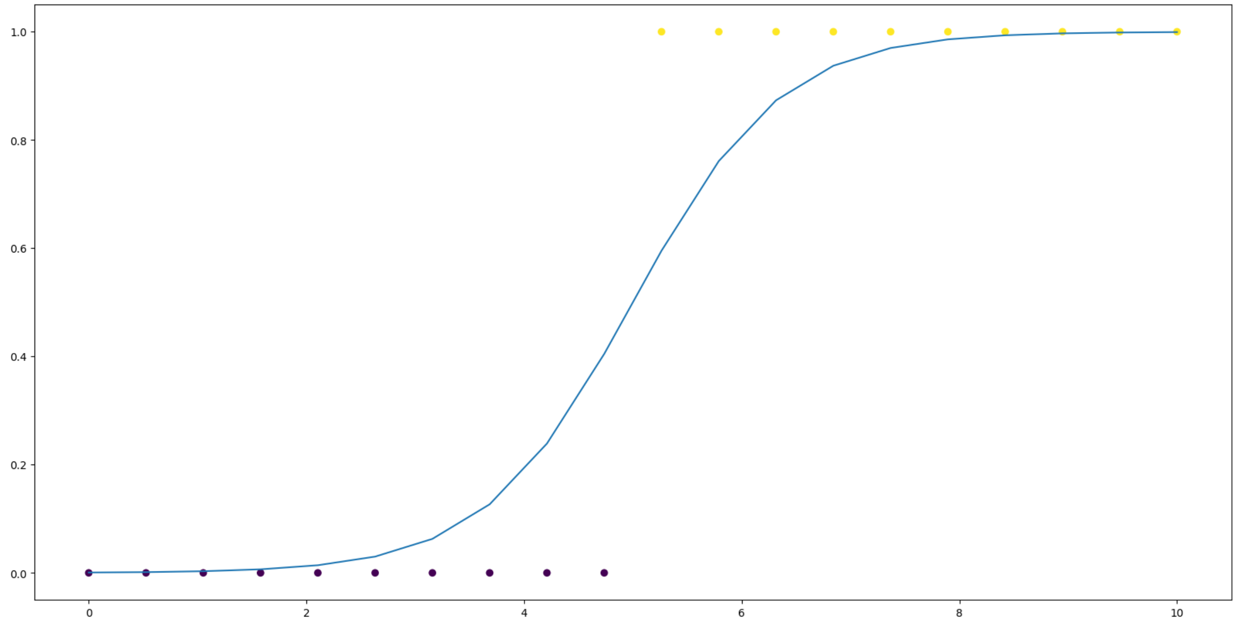
In [4]:
```
model = LogisticRegression()
```

In [5]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[5]:
```
▾ LogisticRegression
LogisticRegression()
```

In [6]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1))[:,1])
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x262e5f155d0>]
```

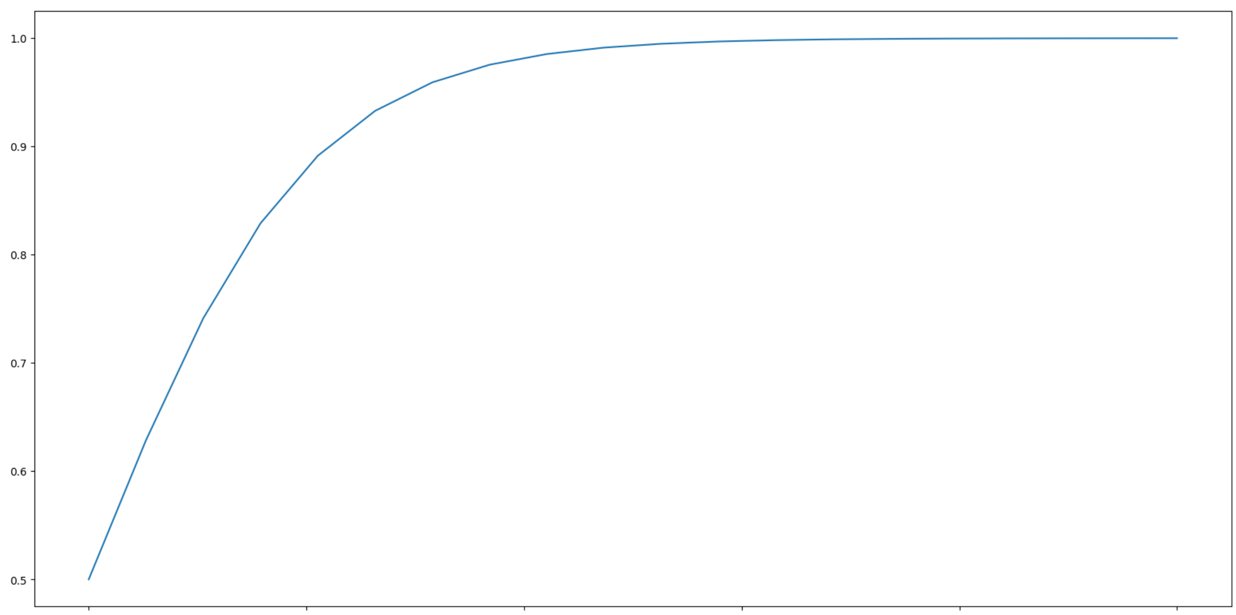


In [7]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

Out[7]:
```
(array([[1.46709085]]), array([-7.33542562]))
```

In [8]:
```
plt.plot(x, 1/(1+np.exp(-x)))
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x262e4b32650>]
```

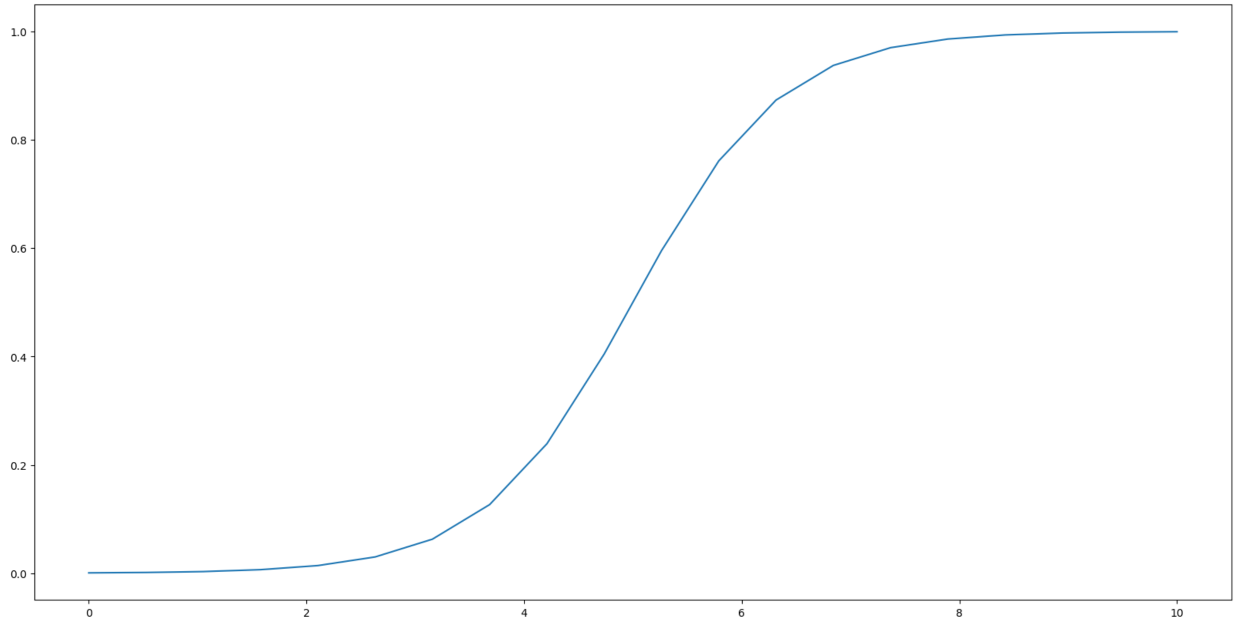


In [9]:
```
b
```

Out[9]:
```
array([[1.46709085]])
```

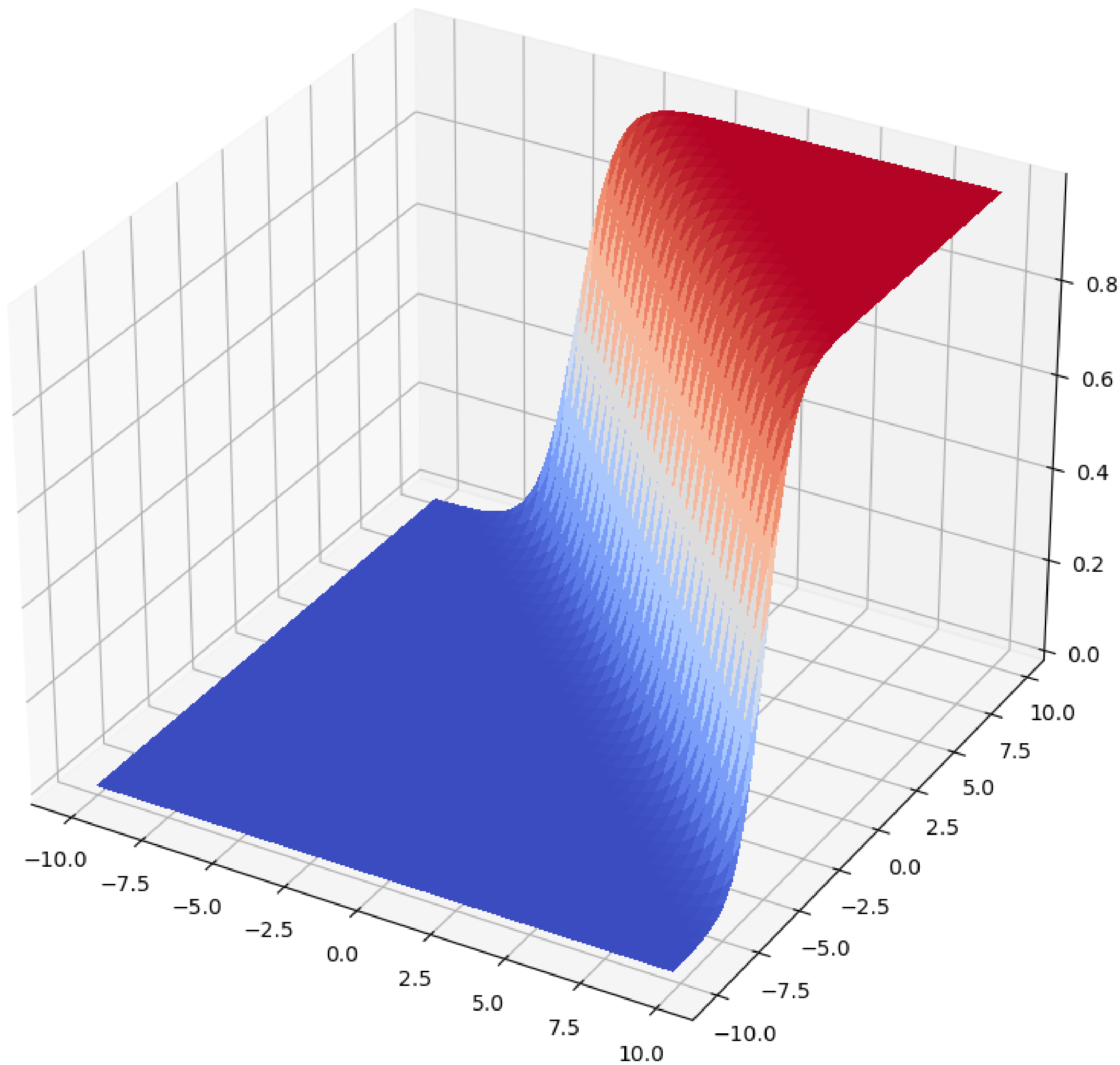
In [10]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x262e4b93f40>]
```



In [15]:

```
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import

import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np


fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Make data.
X = np.arange(-10, 10, 0.25)
Y = np.arange(-10, 10, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = 1/(1+np.exp(-(b[0]*X +b[0]*Y +b0)))
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
```



In [16]:
```
X
```

Out[16]:
```
array([[-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       ...,
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75]])
```

In [17]:
```
Y
```

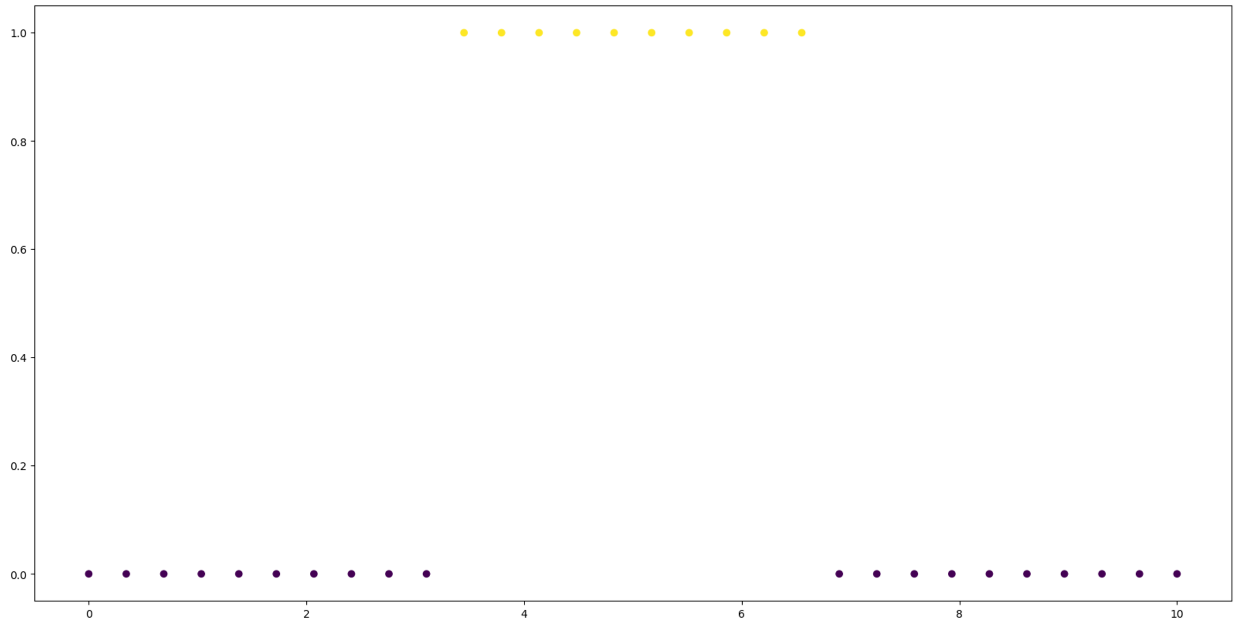
Out[17]:
```
array([[-10.  , -10.  , -10.  , ..., -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, ...,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , ...,  -9.5 ,  -9.5 ,  -9.5 ],
       ...,
       [  9.25,   9.25,   9.25, ...,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , ...,   9.5 ,   9.5 ,   9.5 ],
       [  9.75,   9.75,   9.75, ...,   9.75,   9.75,   9.75]])
```

What if the data doesn't really fit this pattern?

In [18]:
```
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

In [19]:
```
plt.scatter(x,y, c=y)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x262ea609330>
```

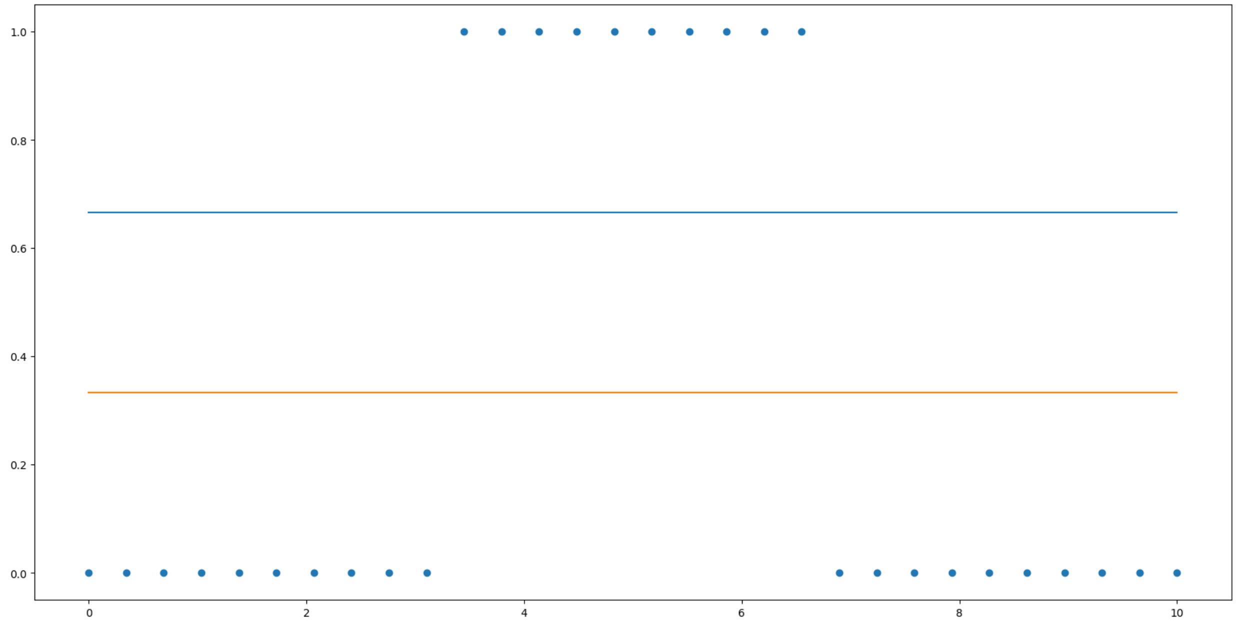


In [20]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[20]:
```
▾ LogisticRegression
LogisticRegression()
```

In [21]:
```
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x262ea5664a0>,
 <matplotlib.lines.Line2D at 0x262ea582860>]
```



In [22]:
```
model1 = LogisticRegression()
model1.fit(x[:15].reshape(-1, 1),y[:15])
```

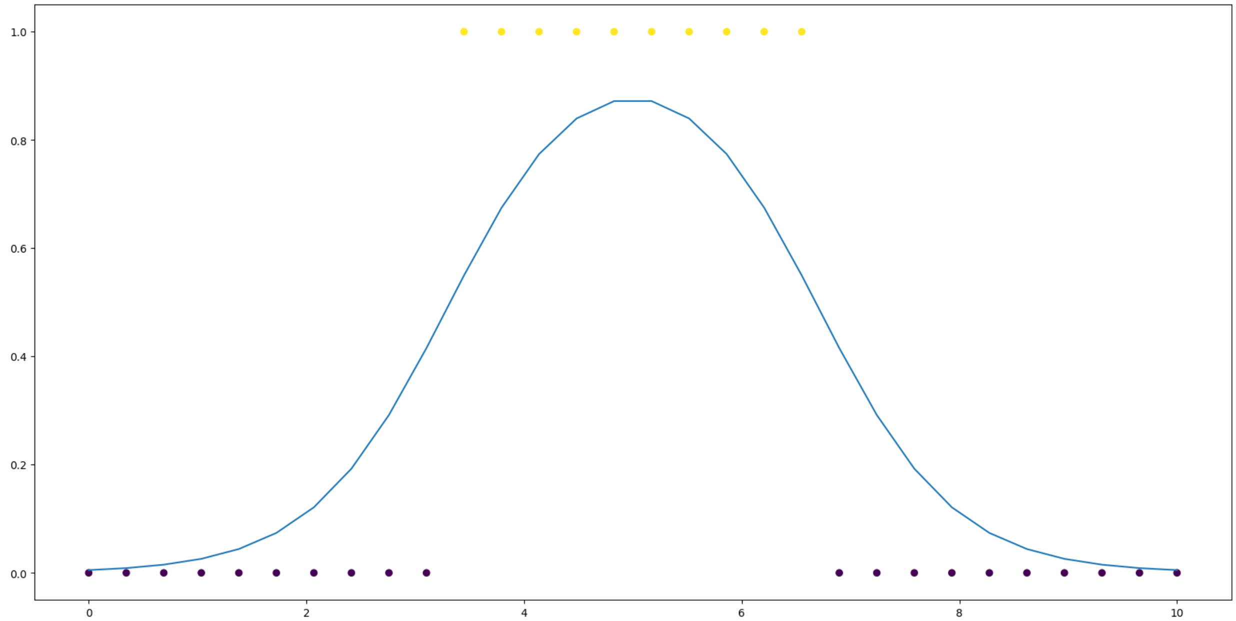
Out[22]:
```
▾ LogisticRegression
LogisticRegression()
```

In [23]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

Out[23]:
```
▾ LogisticRegression
LogisticRegression()
```

In [24]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model1.predict_proba(x.reshape(-1, 1))[:,1] * model2.predict_proba(x.resha
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x262eaea4af0>]
```



In [26]:
```
df = pd.read_csv('adult.data', index_col=False)
golden = pd.read_csv('adult.test', index_col=False)
```

In [27]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

In [28]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

In [29]:
```
x = df.copy()

x[transform_columns] = enc.fit_transform(df[transform_columns])

golden['salary'] = golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K
xt = golden.copy()

xt[transform_columns] = enc.transform(golden[transform_columns])
```

In [30]:
```
df.salary.unique()
```

Out[30]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [31]:
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

Out[31]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [32]:
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

Out[32]:
```
▾ LogisticRegression
LogisticRegression()
```

In [33]:
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

In [34]:
```
x.head()
```

Out[34]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capit g
0	39	7.0	77516	9.0	13	4.0	1.0	1.0	4.0	1.0	2
1	50	6.0	83311	9.0	13	2.0	4.0	0.0	4.0	1.0	
2	38	4.0	215646	11.0	9	0.0	6.0	1.0	4.0	1.0	
3	53	4.0	234721	1.0	7	2.0	6.0	0.0	2.0	1.0	
4	28	4.0	338409	9.0	13	2.0	10.0	5.0	2.0	0.0	

In [35]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [36]:
```
accuracy_score(x.salary, pred)
```

Out[36]:
```
0.8250360861152913
```

In [37]:
```
confusion_matrix(x.salary, pred)
```

Out[37]:
```
array([[23300,  1420],
       [ 4277,  3564]], dtype=int64)
```

In [38]:
```
print(classification_report(x.salary, pred))
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.94      0.89     24720
         1.0       0.72      0.45      0.56      7841

    accuracy                           0.83     32561
   macro avg       0.78      0.70      0.72     32561
weighted avg       0.81      0.83      0.81     32561
```

In [39]:
```
print(classification_report(xt.salary, pred_test))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.94      0.89     12435
         1.0       0.70      0.45      0.55      3846

    accuracy                           0.82     16281
   macro avg       0.77      0.69      0.72     16281
weighted avg       0.81      0.82      0.81     16281
```

Assignment

1. Use your own dataset (`Heart.csv` is acceptable), create a train and a test set, and build 2 models: Logistic Regression and Decision Tree (shallow). Compare the test results using `classification_report` and `confusion_matrix`. Explain which algorithm is optimal

2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, and explain which is optimal

In [58]:
```
#Question1
df = pd.read_csv('Heart.csv', index_col = False)
golden = pd.read_csv('Heart_test.csv', index_col = False)
df.dropna(inplace = True)
golden.dropna(inplace = True)
from sklearn.tree import DecisionTreeClassifier
```

In [59]:
```
transform_columns = ['ChestPain', 'Thal', 'AHD']
```

In [60]:
```
x = df.copy()
xt = golden.copy()

x[transform_columns] = enc.fit_transform(df[transform_columns])

xt[transform_columns] = enc.transform(golden[transform_columns])
```

In [61]:
```
model.fit(preprocessing.scale(x.drop('AHD', axis=1)), x.AHD)
```

Out[61]:
```
▾ LogisticRegression
LogisticRegression()
```

In [62]:
```
pred = model.predict(preprocessing.scale(x.drop('AHD', axis = 1)))
pred_test = model.predict(preprocessing.scale(xt.drop('AHD', axis = 1)))
```

In [63]:
```
accuracy_score(xt.AHD, pred_test)
```

Out[63]:
```
0.817258883248731
```

In [64]:
```
confusion_matrix(xt.AHD, pred_test)
```

Out[64]:
```
array([[91, 12],
       [24, 70]], dtype=int64)
```

In [65]:
```
print(classification_report(xt.AHD, pred_test))
```

```
              precision    recall  f1-score   support

         0.0       0.79      0.88      0.83       103
         1.0       0.85      0.74      0.80        94

    accuracy                           0.82       197
   macro avg       0.82      0.81      0.82       197
weighted avg       0.82      0.82      0.82       197
```

In [66]:
```
modela = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2)
```

In [67]:
```
modela.fit(x.drop(['AHD'], axis = 1), x.AHD)
```

Out[67]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In [68]:
```
predictionsa = modela.predict(xt.drop(['AHD'], axis = 1))
predictionsax = modela.predict(x.drop(['AHD'], axis = 1))
```

In [70]:
```
confusion_matrix(xt.AHD, predictionsa)
```

Out[70]:
```
array([[83, 20],
       [37, 57]], dtype=int64)
```

In [71]:
```
print(classification_report(xt.AHD, predictionsa))
```

```
              precision    recall  f1-score   support

         0.0       0.69      0.81      0.74       103
         1.0       0.74      0.61      0.67        94

    accuracy                           0.71       197
   macro avg       0.72      0.71      0.71       197
weighted avg       0.71      0.71      0.71       197
```

In [87]:
```
#The more optimal model is the Logistic Regression because the precision and recall ar
#This shows greater accuracy in predicting using the test set above. All other indicat
#Regression as being a more optimal model.
```

In [82]:
```
#Question2
modelb = DecisionTreeClassifier(criterion = 'entropy', max_depth = None)
```

In [83]:
```
modelb.fit(x.drop(['AHD'], axis = 1), x.AHD)
```

Out[83]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [84]:
```
predictionsb = modelb.predict(xt.drop(['AHD'], axis = 1))
predictionsbx = modelb.predict(x.drop(['AHD'], axis = 1))
```

In [85]:
```
confusion_matrix(xt.AHD, predictionsb)
```

Out[85]:
```
array([[88, 15],
       [27, 67]], dtype=int64)
```

In [86]:
```
print(classification_report(xt.AHD, predictionsb))
```

```
              precision    recall  f1-score   support

         0.0       0.77      0.85      0.81       103
         1.0       0.82      0.71      0.76        94

    accuracy                           0.79       197
   macro avg       0.79      0.78      0.78       197
weighted avg       0.79      0.79      0.79       197
```

In [ ]:
```
#Although the model without a max_depth (overfitted) was a much better model in terms
#Was not as optimal as the Logistic Regression model was.We can see the precision and
#the level that the Logistic was.
```