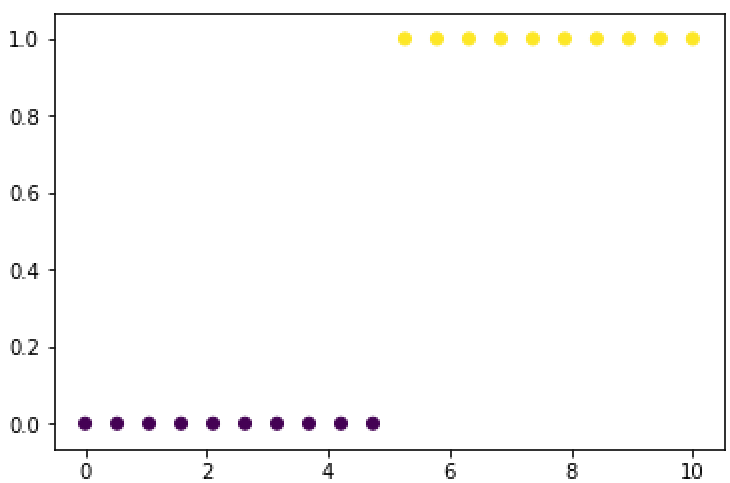
Assignment is at the bottom!

In [1]:
```
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 0x1143d20d0>
```



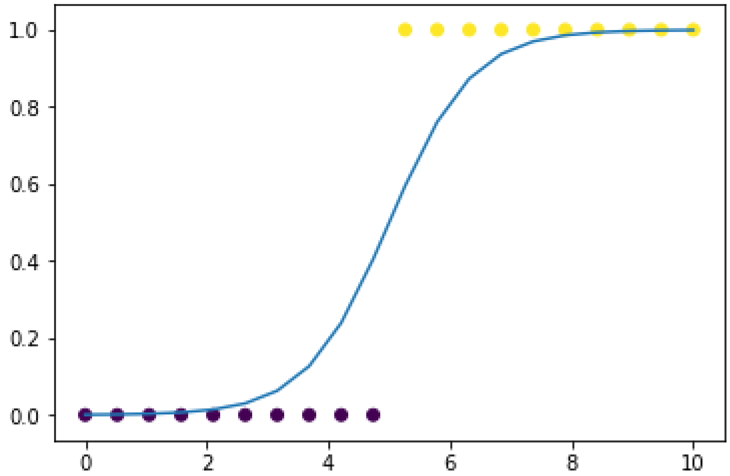
In [4]:
```
model = LogisticRegression()
```

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

Out[5]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

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 0x11653c390>]
```

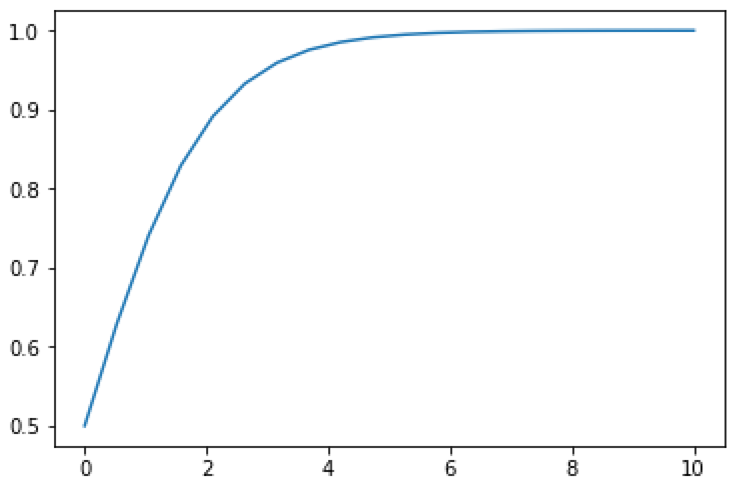


In [7]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

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

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

Out[8]:
```
[<matplotlib.lines.Line2D at 0x1166c60d0>]
```

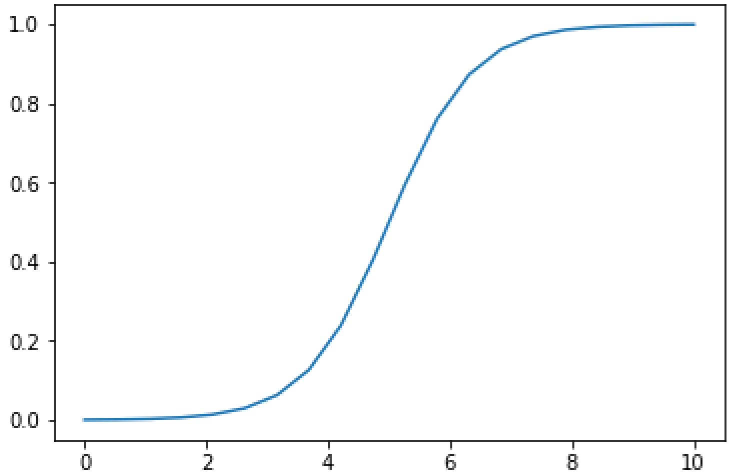


In [9]:
```
b
```

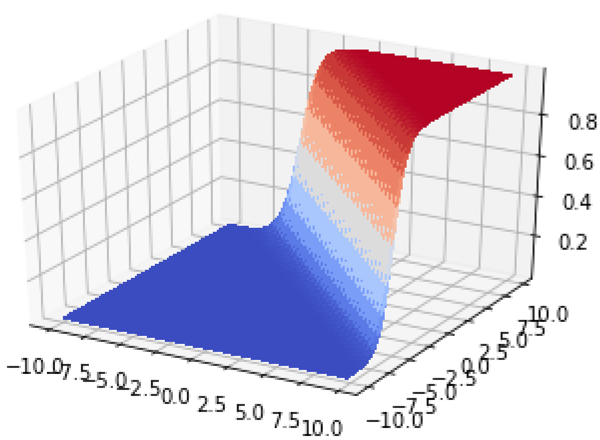
Out[9]:
```
array([[1.46709085]])
```

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

Out[10]:
```
[<matplotlib.lines.Line2D at 0x11679a890>]
```



In [11]:
```
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.gca(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 [12]:
```
X
```

Out[12]:
```
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 [13]:
```
Y
```

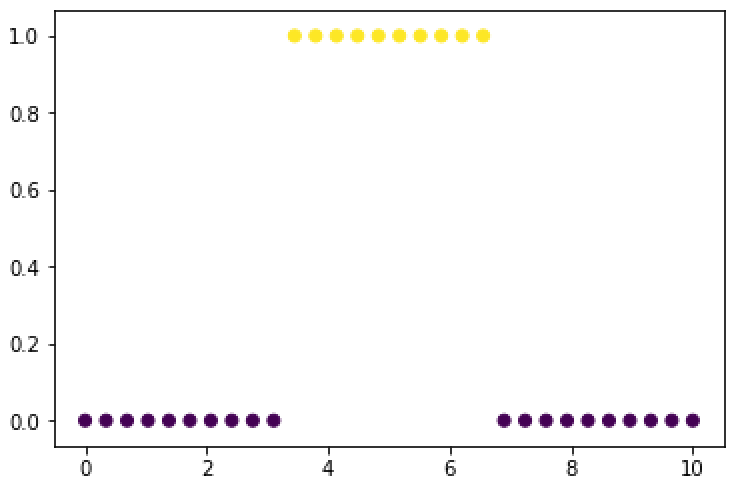
Out[13]:
```
array([[-10.  , -10.  , -10.  , ..., -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, ...,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , ...,  -9.5 ,  -9.5 ,  -9.5 ],
       ...,
       [  9.25,   9.25,   9.25, ...,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , ...,   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 [14]:
```
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

In [15]:
```
plt.scatter(x,y, c=y)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x116ba2810>
```

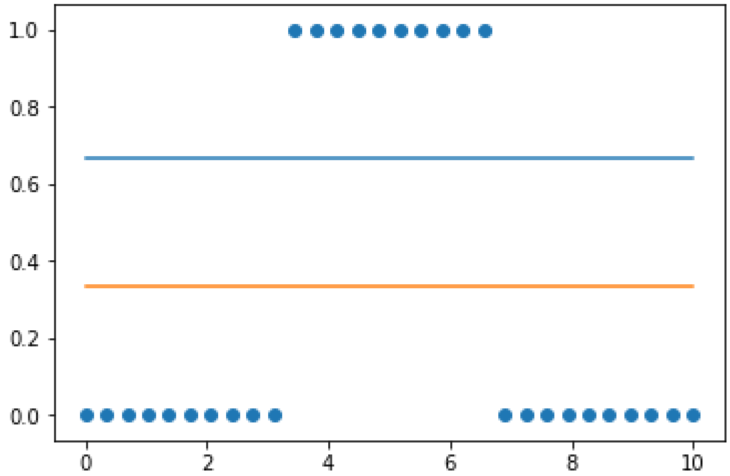


In [16]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[16]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [17]:
```
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x116d43450>,
 <matplotlib.lines.Line2D at 0x116d43690>]
```



In [18]:
```
model1 = LogisticRegression()
model1.fit(x[:15].reshape(-1, 1),y[:15])
```

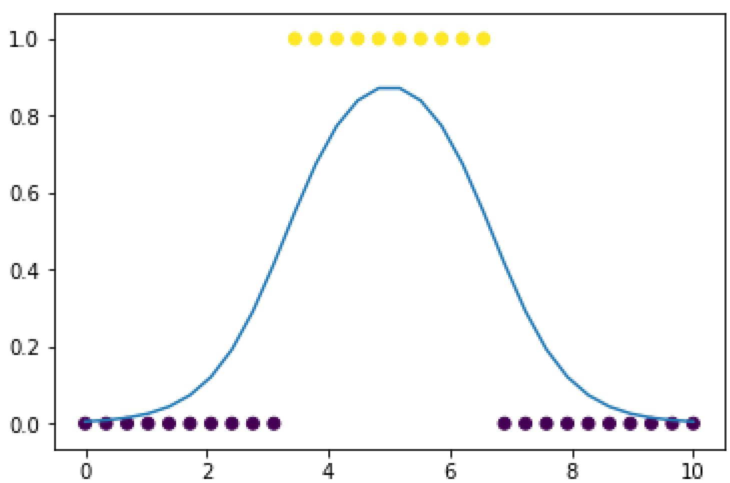
Out[18]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [19]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

Out[19]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [20]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model1.predict_proba(x.reshape(-1, 1))[:,1] * model2.predict_proba
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x116e26c50>]
```



In [21]:
```
df = pd.read_csv('../data/adult.data', index_col=False)
golden = pd.read_csv('../data/adult.test', index_col=False)
```

In [22]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

In [23]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

In [24]:
```
x = df.copy()

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

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

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

In [25]:
```
df.salary.unique()
```

Out[25]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [26]:
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

Out[26]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [27]:
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

Out[27]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [28]:
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

In [29]:
```
x.head()
```

Out[29]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex
0	39	7.0	77516	9.0	13	4.0	1.0	1.0	4.0	1.0
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 [30]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [31]:
```
accuracy_score(x.salary, pred)
```
Out[31]:
```
0.8250360861152913
```

In [32]:
```
confusion_matrix(x.salary, pred)
```
Out[32]:
```
array([[23300,  1420],
       [ 4277,  3564]])
```

In [33]:
```
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 [34]:
```
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

In [1]:
```
from sklearn.linear_model import LogisticRegression
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import numpy as np
from pylab import rcParams
rcParams['figure.figsize'] = 20, 10
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['font.size'] = 14
from sklearn import preprocessing
enc = preprocessing.OrdinalEncoder()
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [5]:
```
df = pd.read_csv('~/Desktop/Heart.csv', index_col=False)
```

In [30]:
```
df = df.dropna()
df.head()
df['AHD_yn'] = np.where(df['AHD'] == 'Yes', 1, 0)
df['AHD_yn'].isna().sum()
df['AHD_yn'].value_counts()

numeric_columns = df.select_dtypes(include=['number']).columns
df_numeric = df[numeric_columns]
```

In [31]:
```
X = df_numeric.drop(['AHD_yn'], axis=1)
y = df_numeric['AHD_yn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randor

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [32]:
```
lr_model = LogisticRegression()

lr_model.fit(X_train_scaled, y_train)

y_prd_logreg = lr_model.predict(X_test_scaled)

print("Accuracy Score:")
print(accuracy_score(y_test, y_prd_logreg))
print("Classification Report:")
print(classification_report(y_test, y_prd_logreg))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_prd_logreg))
```

```
Accuracy Score:
0.7833333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.70      0.76        30
           1       0.74      0.87      0.80        30

    accuracy                           0.78        60
   macro avg       0.79      0.78      0.78        60
weighted avg       0.79      0.78      0.78        60

Confusion Matrix:
[[21  9]
 [ 4 26]]
```

In [33]:

```
dtc_model = DecisionTreeClassifier(max_depth=2, random_state=51)

dtc_model.fit(X_train_scaled, y_train)

y_prd_dtc = dtc_model.predict(X_test_scaled)

accuracy_dtc = accuracy_score(y_test, y_prd_dtc)
print(f"Decision Tree Accuracy Score: {accuracy_dtc}")
print("\nDecision Tree:")
print(classification_report(y_test, y_prd_dtc))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_prd_dtc))
```

```
Decision Tree Accuracy Score: 0.7

Decision Tree:
              precision    recall  f1-score   support

           0       0.69      0.73      0.71        30
           1       0.71      0.67      0.69        30

    accuracy                           0.70        60
   macro avg       0.70      0.70      0.70        60
weighted avg       0.70      0.70      0.70        60


Confusion Matrix:
[[22  8]
 [10 20]]
```

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 [27]:

```
dtc_deep = DecisionTreeClassifier(max_depth=17, random_state=51)

dtc_deep.fit(X_train_scaled, y_train)

y_pred_dtcd = dtc_deep.predict(X_test_scaled)

accuracy_dtc = accuracy_score(y_test, y_pred_dtc_deep)
print(f"Decision Tree Accuracy Score: {accuracy_dtc}")
```

```
print("\nDecision Tree:")
print(classification_report(y_test, y_pred_dtcd))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_dtcd))
```

```
Decision Tree Accuracy Score: 0.7166666666666667

Decision Tree:
              precision    recall  f1-score   support

           0       0.71      0.73      0.72        30
           1       0.72      0.70      0.71        30

    accuracy                           0.72        60
   macro avg       0.72      0.72      0.72        60
weighted avg       0.72      0.72      0.72        60


Confusion Matrix:
[[22  8]
 [ 9 21]]
```

In [ ]: