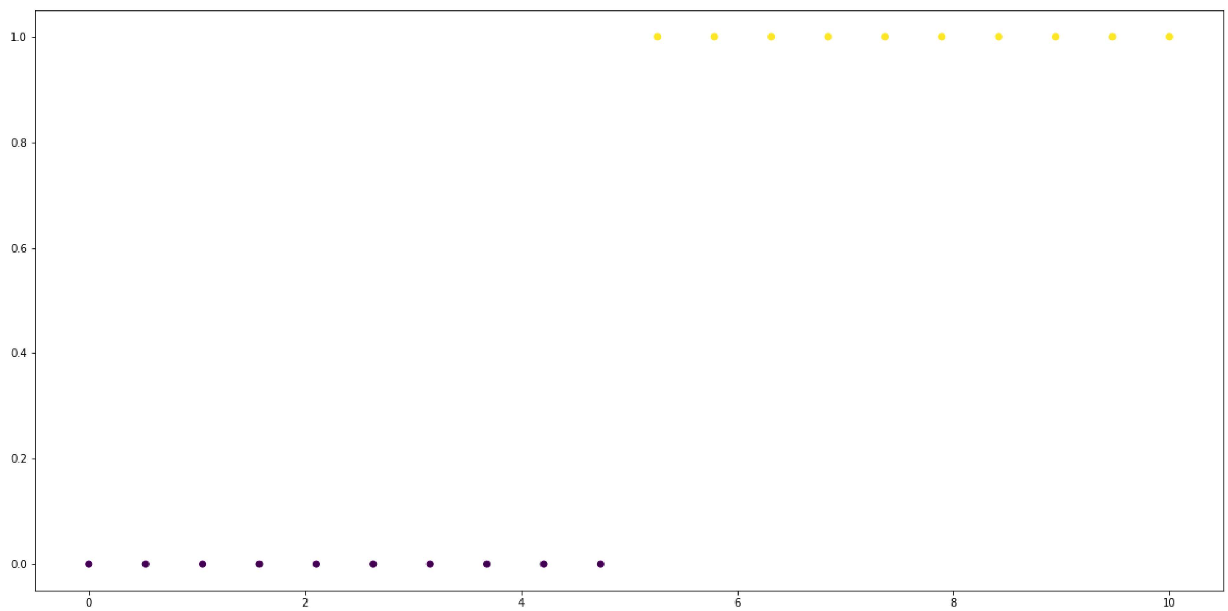
Assignment is at the bottom!

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 0x23c6d905a50>
```



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

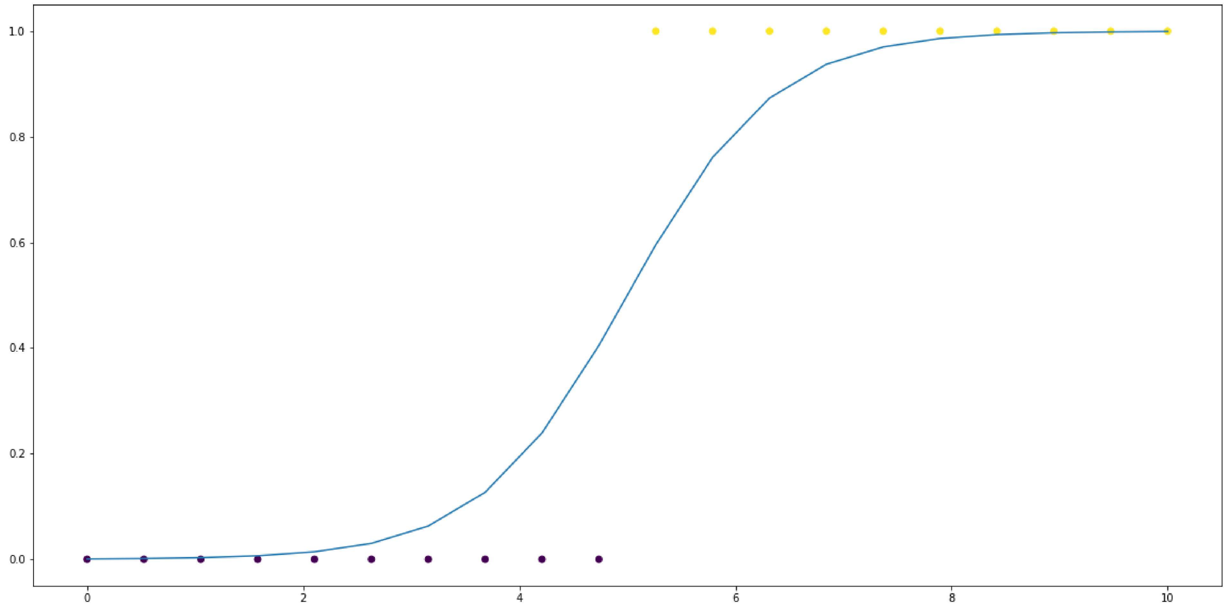
In [5]:
```
model.fit(x.reshape(-1, 1),y)
```

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

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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 0x23c7034ab00>]
```

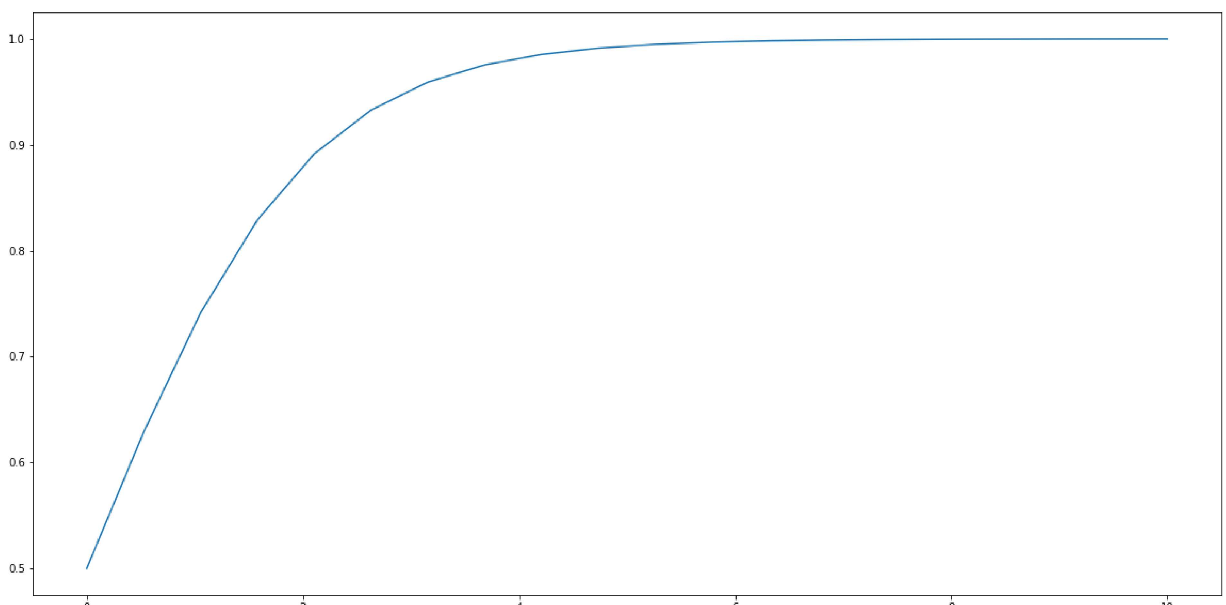


In [7]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

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

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

Out[8]:
```
[<matplotlib.lines.Line2D at 0x23c703bdf30>]
```

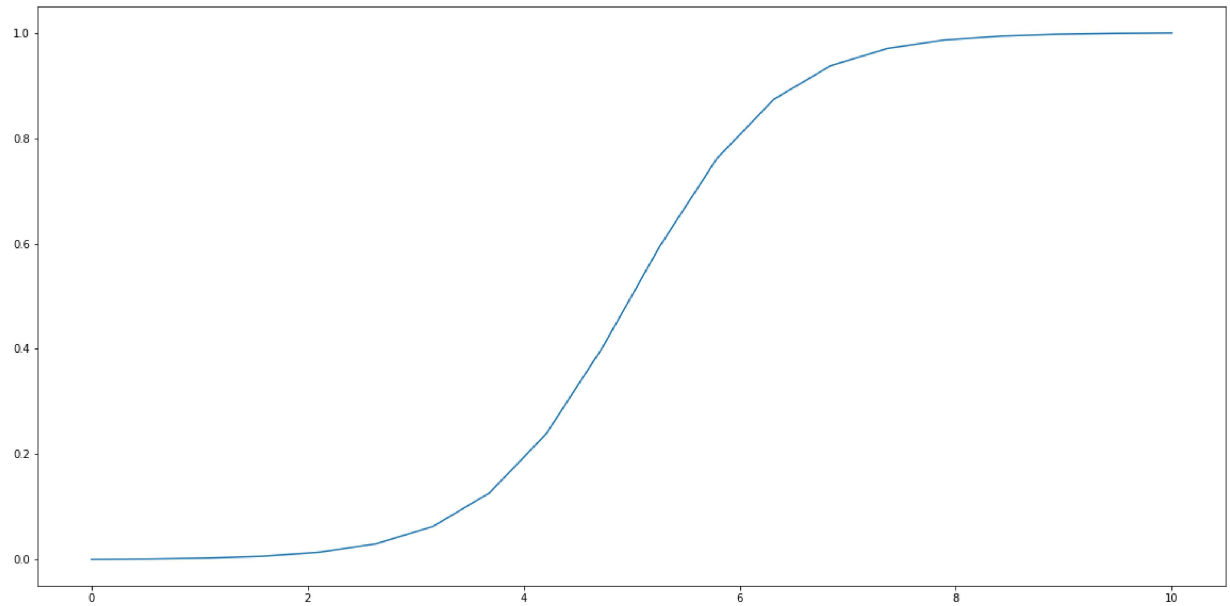


In [9]:
```
b
```

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

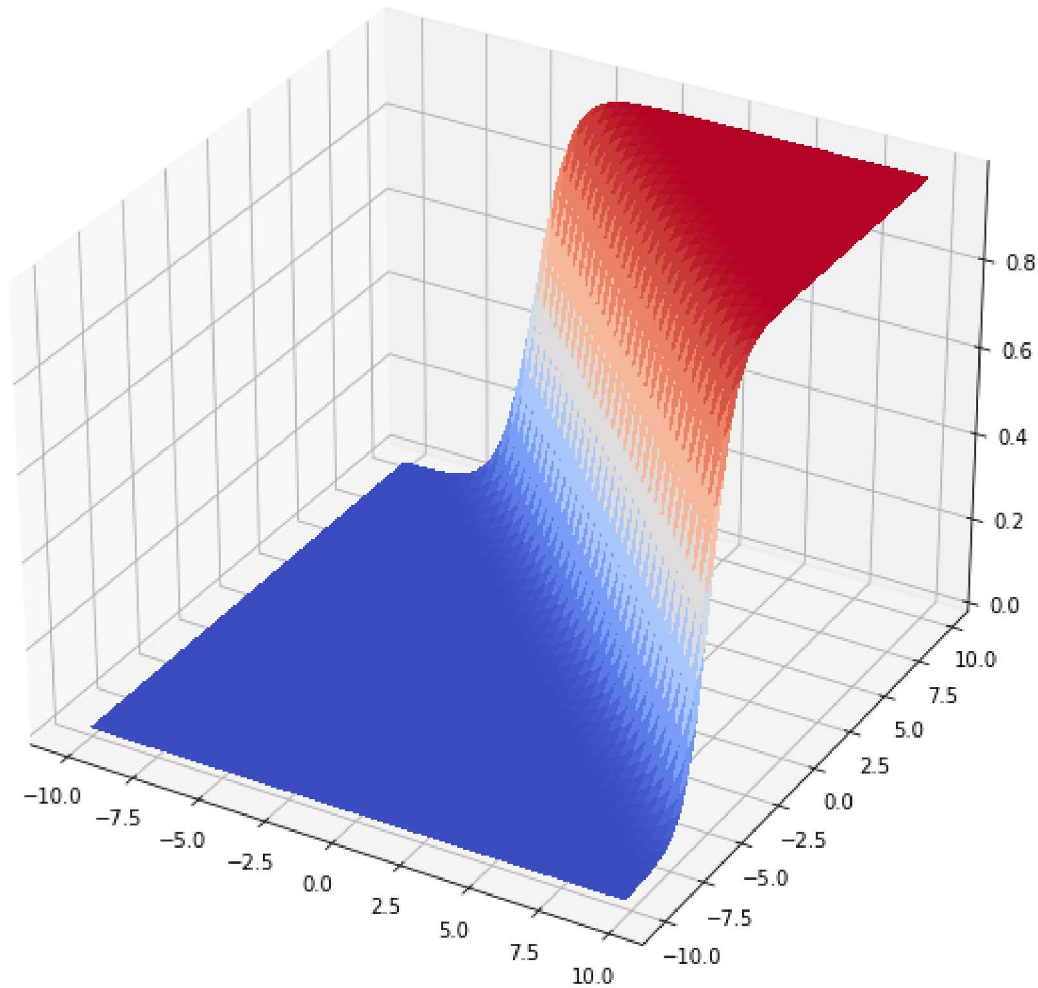
In [10]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x23c6fa00fa0>]
```



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)
```

```
C:\Users\gdlev\AppData\Local\Temp\ipykernel_14704\290434025.py:10: MatplotlibDe
precationWarning: Calling gca() with keyword arguments was deprecated in Matplo
tlib 3.4. Starting two minor releases later, gca() will take no keyword argumen
ts. The gca() function should only be used to get the current axes, or if no ax
es exist, create new axes with default keyword arguments. To create a new axes
with non-default arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



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
```

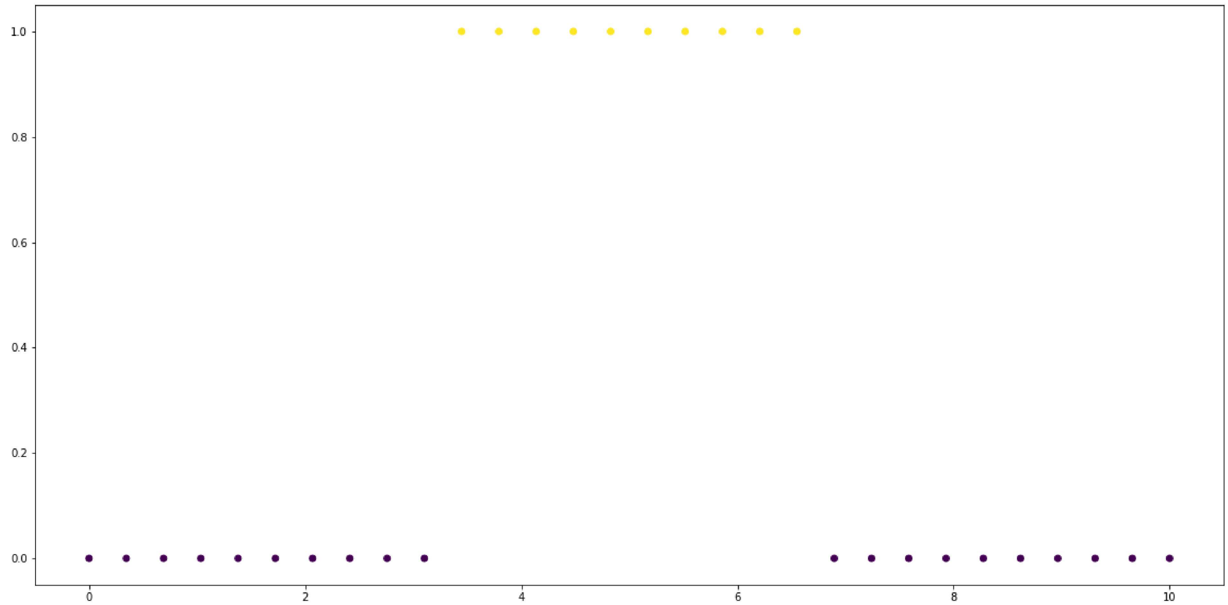
Out[13]:
```
array([[-10.  , -10.  , -10.  , ..., -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, ...,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , ...,  -9.5 ,  -9.5 ,  -9.5 ],
       ...,
       [  9.25,   9.25,   9.25, ...,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , ...,   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 0x23c70676230>



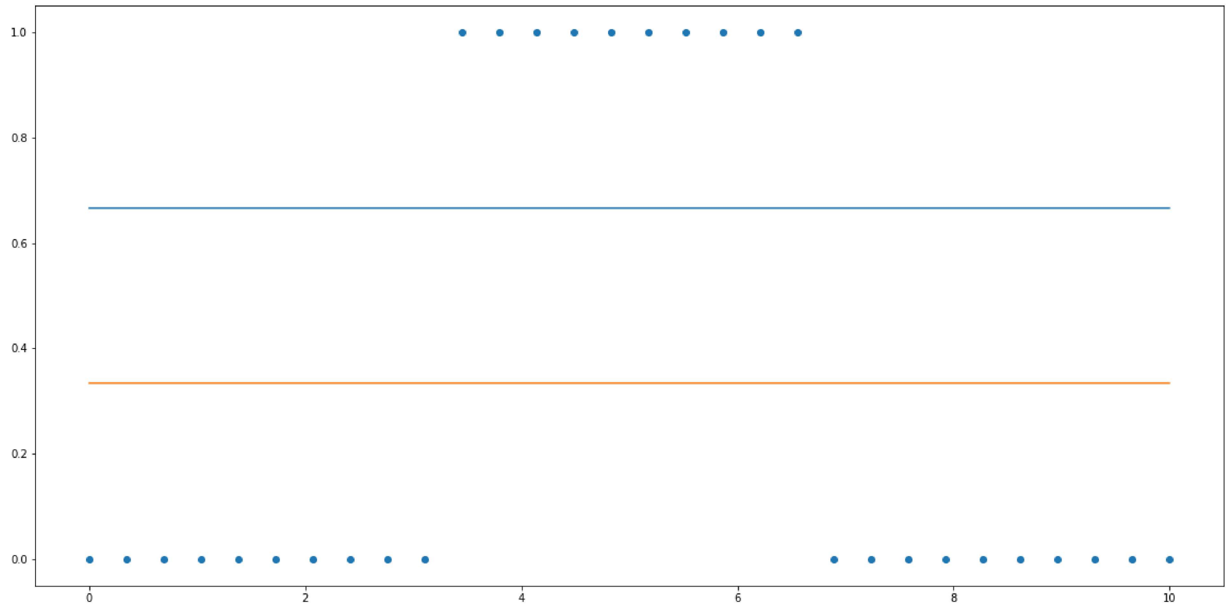
In [16]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[16]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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

Out[17]:
```
[<matplotlib.lines.Line2D at 0x23c706b1810>,
 <matplotlib.lines.Line2D at 0x23c706b1870>]
```



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

Out[18]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

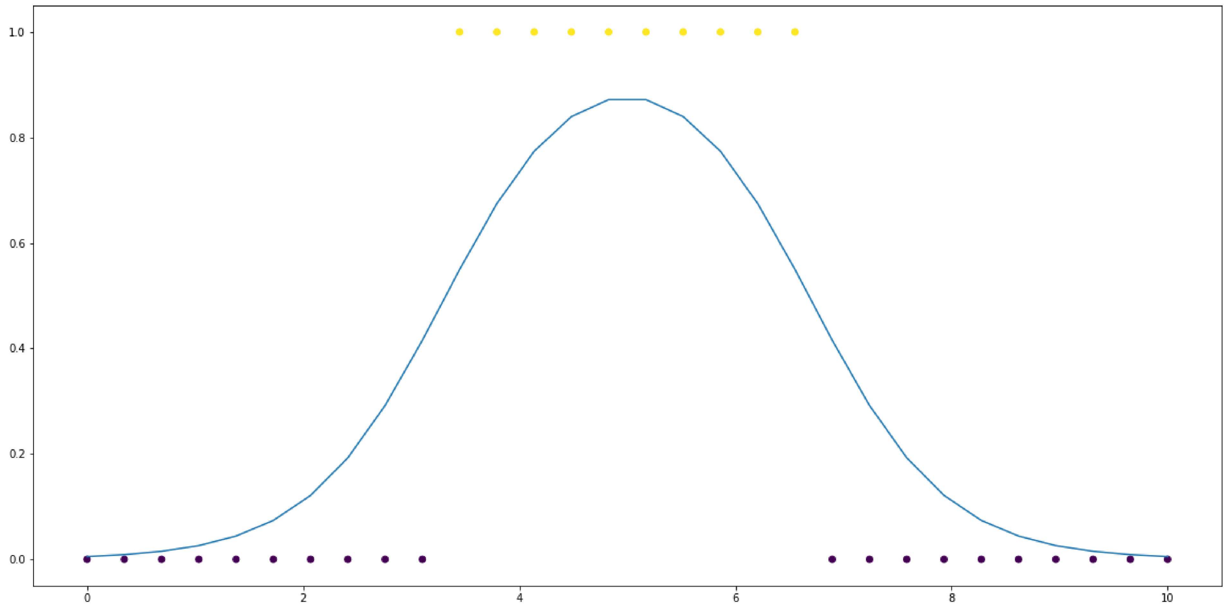
In [19]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

Out[19]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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

Out[20]:
```
[<matplotlib.lines.Line2D at 0x23c70720f40>]
```



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()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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	ca
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]], dtype=int64)
```

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 (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`. Which algorithm is superior?

2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, and explain why it's superior

In [1]:
```
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import math
import re
plt.rcParams['figure.figsize'] = [10, 5]
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    classification_report,
    confusion_matrix, auc, roc_curve,
    roc_auc_score
)
```

In [2]:
```
df_orig = pd.read_csv("NFL Play by Play 2009-2018 (v5).csv")
```

```
C:\Users\gdlev\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:333
1: DtypeWarning: Columns (42,166,167,168,169,174,175,178,179,182,183,188,189,19
0,191,194,195,203,204,205,218,219,220,231,232,233,238,240,241,249) have mixed t
ypes.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [3]:
```
df = df_orig.copy()
```

In [4]:
```
all_cols = df.columns
cols_include = [
    "posteam_type",
    "yardline_100",
    "quarter_seconds_remaining",
    "half_seconds_remaining",
    "game_seconds_remaining",
    "game_half",
    "drive",
    # "sp",
    "qtr",
    "down",
    "goal_to_go",
    "ydstogo",
    "play_type",
    "shotgun",
    "no_huddle",
    "posteam_timeouts_remaining",
    "defteam_timeouts_remaining",
    "score_differential",
    "fourth_down_converted"

]
df = df_orig[cols_include].copy()
```

In [5]:
```
# for i in df.columns:
#     print(f"Column: {i}")
#     print(df[i].value_counts())

# category columns
# posteam_type
# game_half
# play_type

### Predict fourth down conversions



df_4th = df[(df.down == 4) & df.play_type.isin(['pass','run'])].copy()
df_4th.loc[(df_4th.score_differential >= -3) & (df_4th.score_differential < 0), '
df_4th.loc[(df_4th.score_differential == 0), 'curr_result'] = "tied"
df_4th.loc[(df_4th.score_differential <= -3) & (df_4th.score_differential >= -7),
df_4th.loc[(df_4th.score_differential > 0), 'curr_result'] = "winning"
df_4th.loc[(df_4th.score_differential <= -7), 'curr_result'] = "losing_td"
# df_4th = df_4th[(df_4th.score_differential >= -7) & (df_4th.score_differential
# # df_4th = df_4th[df_4th.half_seconds_remaining >= 500]
# # df_4th = df_4th[df_4th.game_seconds_remaining > 500]
# df_4th = df_4th[df_4th.goal_to_go == 0]
# df_4th = df_4th[df_4th.yardline_100 <= 50]
df_4th = df_4th[df_4th.ydstogo <= 2]



print(df_4th.describe())
print(df_4th.fourth_down_converted.value_counts())

cat_cols = ['posteam_type', 'game_half', 'curr_result']
df_4th = df_4th.drop(['down', 'play_type'], axis = 1)
df_4th = pd.concat([df_4th.drop(cat_cols, axis = 1), pd.get_dummies(df_4th[cat_co
df_4th = df_4th.dropna()


# df_4th["curr_winning"] = np.where(df_4th.score_differential < 0, 0, 1)
```

```
       yardline_100  quarter_seconds_remaining  half_seconds_remaining  \
count   2321.000000                2321.000000             2321.000000   
mean      30.814735                 385.418354              708.037915   
std       21.016028                 260.931426              508.949574   
min        1.000000                   1.000000                1.000000   
25%       12.000000                 144.000000              231.000000   
50%       32.000000                 360.000000              649.000000   
75%       45.000000                 607.000000             1134.000000   
max       90.000000                 900.000000             1800.000000   

       game_seconds_remaining        drive          qtr    down  goal_to_go  \
count             2321.000000  2321.000000  2321.000000  2321.0  2321.000000   
mean              1442.462732    13.308488     2.832831     4.0     0.139164   
std               1074.512766     7.344336     1.147550     0.0     0.346192   
min                  1.000000     1.000000     1.000000     4.0     0.000000   
25%                418.000000     7.000000     2.000000     4.0     0.000000   
50%               1290.000000    14.000000     3.000000     4.0     0.000000   
```

```
75%              2384.000000   19.000000    4.000000    4.0    0.000000
max              3519.000000   34.000000    5.000000    4.0    1.000000

           ydstogo      shotgun    no_huddle  posteam_timeouts_remaining  \
count  2321.000000  2321.000000  2321.000000                 2321.000000
mean      1.233089     0.316674     0.059026                    2.290392
std       0.422890     0.465279     0.235725                    0.899779
min       1.000000     0.000000     0.000000                    0.000000
25%       1.000000     0.000000     0.000000                    2.000000
50%       1.000000     0.000000     0.000000                    3.000000
75%       1.000000     1.000000     0.000000                    3.000000
max       2.000000     1.000000     1.000000                    3.000000

       defteam_timeouts_remaining  score_differential  fourth_down_converted
count                 2321.000000         2321.000000            2321.000000
mean                     2.435157           -4.306333               0.620422
std                      0.836787           12.172666               0.485386
min                      0.000000          -59.000000               0.000000
25%                      2.000000          -12.000000               0.000000
50%                      3.000000           -4.000000               1.000000
75%                      3.000000            1.000000               1.000000
max                      3.000000           48.000000               1.000000
1.0    1440
0.0     881
Name: fourth_down_converted, dtype: int64
```

In [6]:

```
X_df = df_4th.drop("fourth_down_converted", axis = 1)
y_df = df_4th[["fourth_down_converted"]]
```

In [7]:

```
def get_accuracies(y_true, preds, model_name = "Model", verbose = True):
    acc = accuracy_score(y_true, preds)
    f1 = f1_score(y_true, preds)
    auc = roc_auc_score(y_true, preds)
    confus = confusion_matrix(y_true, preds)
    classif = classification_report(y_true, preds)

    acc_list = [acc,f1,auc, confus, classif]
    if verbose:
        print(f"\033[1m{model_name}\033[0m: \n\nAccuracy: {acc_list[0] * 100:.2f}

    return acc_list
```

In [128]:
```
def train_model(X_df, y_df, trainsize = 0.75, model_type = "Logistic Regression",

    X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, train_size =
    if model_type == "Logistic Regression":
        model = linear_model.LogisticRegression()
    else:
        model = DecisionTreeClassifier(max_depth = maxdepth)

    model.fit(X_train, y_train.values.ravel())
    preds = model.predict(X_test)

    get_accuracies(y_test, preds, model_name = model_type)

    return model
```

In [139]:
```
print(train_model(X_df, y_df, model_type = "Logistic Regression"))
print(train_model(X_df, y_df, model_type = "Decision Tree"))
print(train_model(X_df, y_df, model_type = "Decision Tree", maxdepth = 10))
```

```
Logistic Regression:

Accuracy: 62.13%

F1-Score: 74.71%

AUC Score: 53.74%

Confusion Matrix:
 [[ 36 190]
 [ 30 325]]

Classification Report:

               precision    recall  f1-score   support

         0.0       0.55      0.16      0.25       226
         1.0       0.63      0.92      0.75       355

    accuracy                           0.62       581
   macro avg       0.59      0.54      0.50       581
weighted avg       0.60      0.62      0.55       581



LogisticRegression()
Decision Tree:

Accuracy: 58.86%

F1-Score: 73.36%

AUC Score: 51.75%

Confusion Matrix:
 [[ 13 233]
 [  6 329]]

Classification Report:

               precision    recall  f1-score   support

         0.0       0.68      0.05      0.10       246
         1.0       0.59      0.98      0.73       335

    accuracy                           0.59       581
   macro avg       0.63      0.52      0.42       581
weighted avg       0.63      0.59      0.46       581



DecisionTreeClassifier(max_depth=3)
```

```
Decision Tree:

Accuracy: 60.24%

F1-Score: 70.87%

AUC Score: 54.53%

Confusion Matrix:
 [[ 69 149]
 [ 82 281]]

Classification Report:

               precision    recall  f1-score   support

         0.0       0.46      0.32      0.37       218
         1.0       0.65      0.77      0.71       363

    accuracy                           0.60       581
   macro avg       0.56      0.55      0.54       581
weighted avg       0.58      0.60      0.58       581



DecisionTreeClassifier(max_depth=10)
```

```
c:\Users\gdlev\AppData\Local\Programs\Python\Python310\lib\site-packages\sklear
n\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

In [118]:
```
importance_df = pd.DataFrame(list(zip(model.feature_names_in_, model.feature_impo
importance_df = importance_df.rename(columns = {importance_df.columns[0]: "featur
```

In [66]:
```
importance_df[importance_df.importance == 0].featurename
```

Out[66]:
```
5                          qtr
6                   goal_to_go
8                      shotgun
9                    no_huddle
10    posteam_timeouts_remaining
13           posteam_type_away
14           posteam_type_home
15              game_half_Half1
16              game_half_Half2
17           game_half_Overtime
18         curr_result_losing_fg
19      curr_result_losing_fg_td
20         curr_result_losing_td
21              curr_result_tied
22           curr_result_winning
Name: featurename, dtype: object
```

Compare the test results

According to this split, it appears that the logistic regression model slightly outperforms the shallow decision tree mode. This is known because according to the classification report, the logistic regression model catches 92% of the cases where the possession team correctly converts a fourth down. In addition, the model predicts 63% correctly whether a team will or will not convert the fourth down. This leads to an overall F1 Score of 75%, F1 is a useful metric in comparing models, because it shows how prevalent false positives and false negatives that are predicted by the model. The model makes 581 predictions, the logistic regression model predicts a team should go for a fourth down 355 times, and predicts a team should kick or punt on the fourth down 226 times. In nominal terms, out of the 355 predictions to go for the fourth down, 325/355 are correct. This can be compared to the shallow decision tree model which has an overall F1 Score of 72.45%. Something to note however is that a false positive is significantly worse than a false negative. A false positive would represent the model predicting a team should go for a fourth down, even though they end up not converting, a false negative would be advising the team to kick or punt on the fourth down when they likely could have converted. The reason why a false positive is worse is because in football if a fourth down play is not converted than the opposing team immediately gets the ball where the play ends, this could lead to a significant disadvantage for the possession team. Due to this, the best model in my opinion, would be the model with the lowest amount of false positives which is decision tree which records 6/581 = 1% which is less than the logistic regression model which has a false positive rate of 30/581 = 5%. ¶

Using the same logic as before, I would argue that the shallow decision tree model is even better than the deep decision tree model. Regardless of accuracy metrics, the most important asepct of the model is low false positive rate and the deep decision tree model actually has the highest false positive rate of any of the three models at 82/581 = 14%. However, I believe that if I created a more accurate model, that by increasing the depth of the decision tree would overfit the model and lead to much higher metrics, unfortunately I did not observe this phenomenon in my own data. Comparing all three models and using understanding of football strategy, I believe the model I would choose to use for this split would be the shallow decision tree model due to its low false positive rate.