Assignment7

July 10, 2024

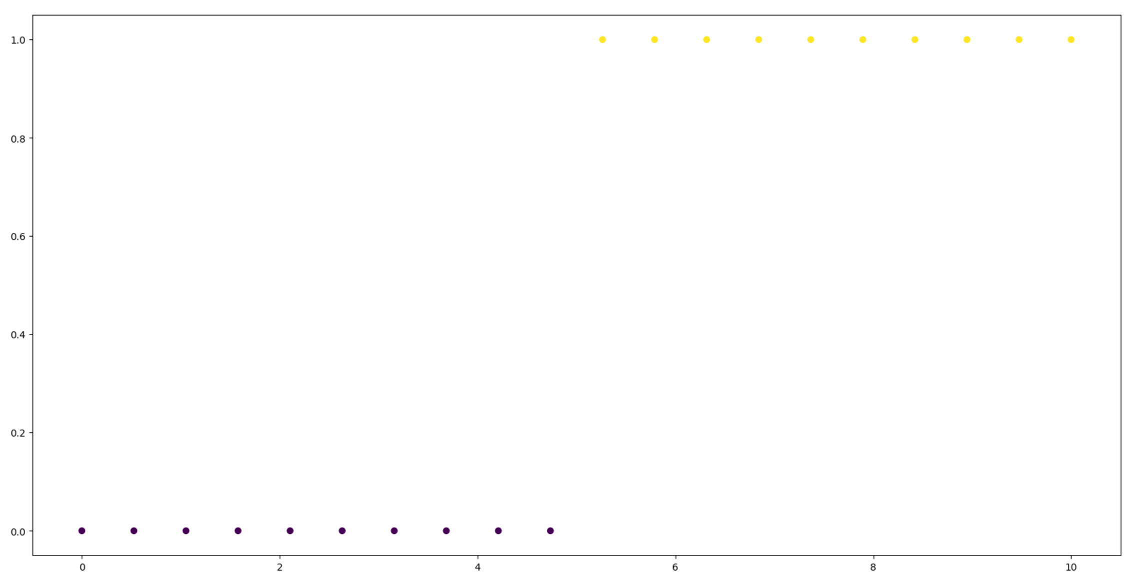
1 Assignment is at the bottom!

[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
```

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

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

[3]:
```
<matplotlib.collections.PathCollection at 0x7fe84e507b50>
```



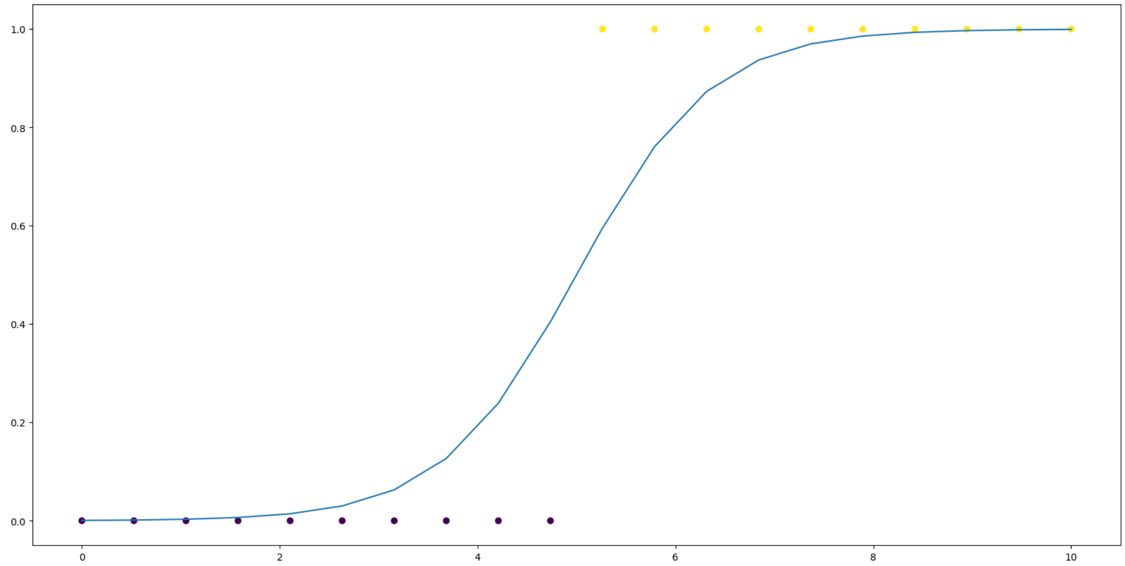
[4]:
```
model = LogisticRegression()
```

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

[5]: LogisticRegression()

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

[6]: [<matplotlib.lines.Line2D at 0x7fe844e1a250>]

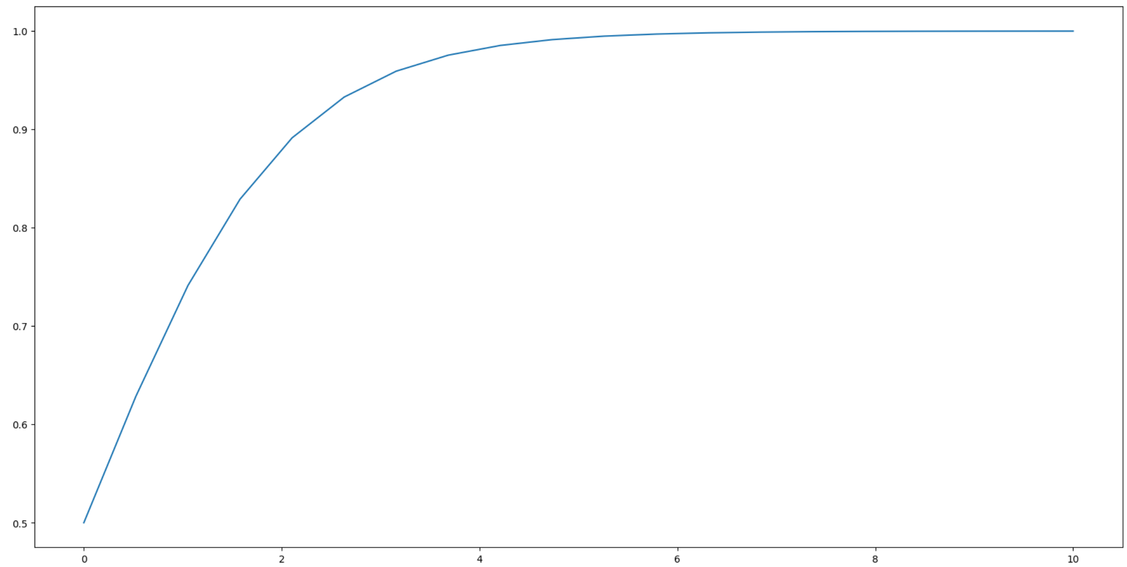


[7]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

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

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

[8]: [<matplotlib.lines.Line2D at 0x7fe844ec5410>]

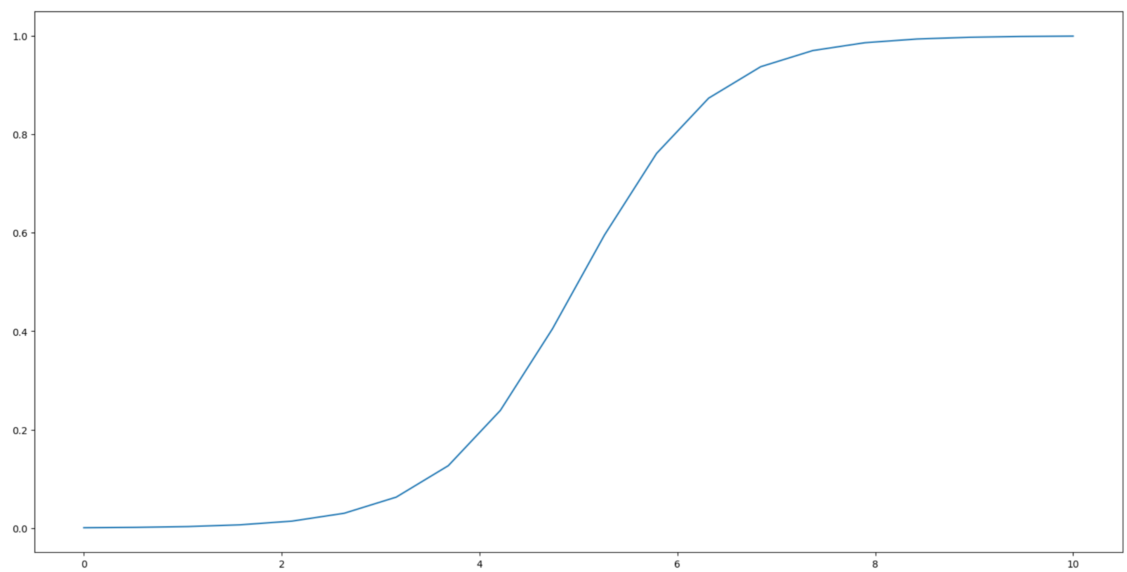


[9]:
```
b
```

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

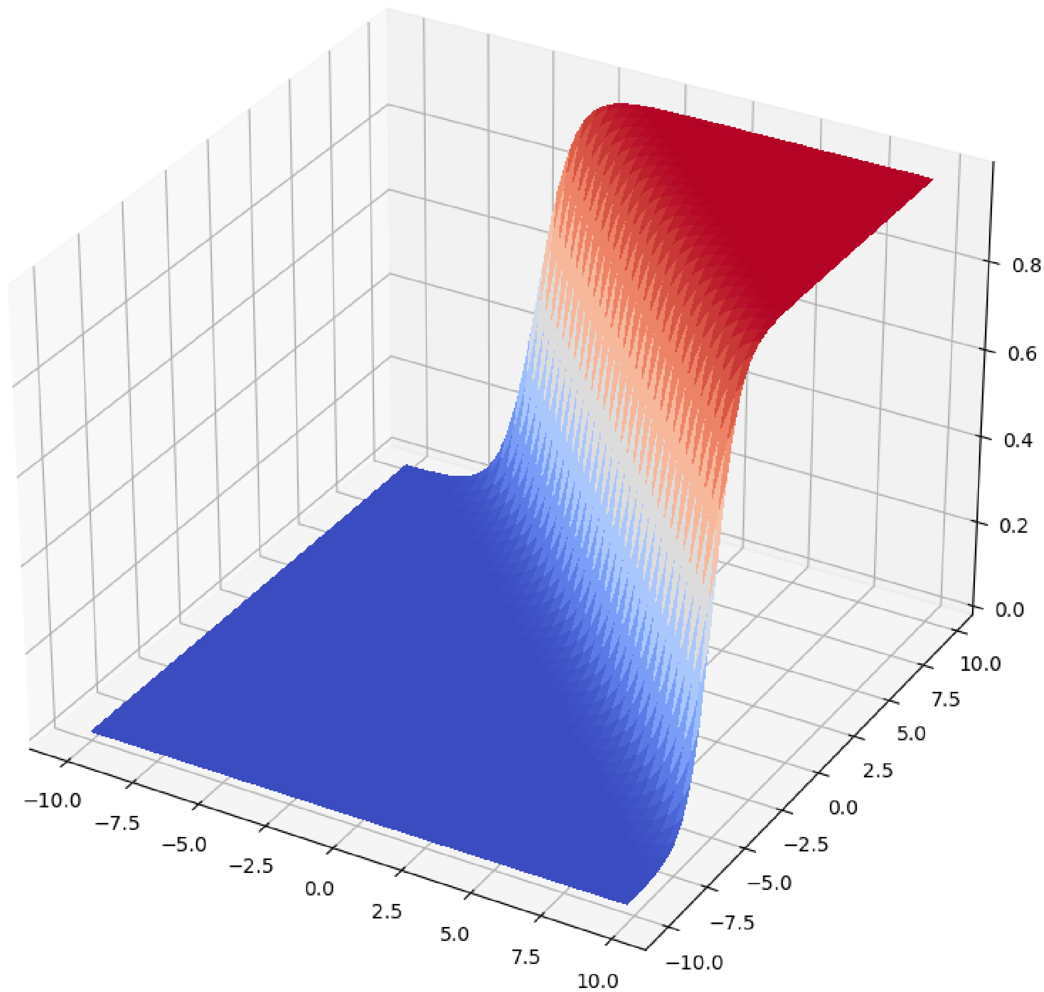
[10]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```

[10]: [<matplotlib.lines.Line2D at 0x7fe84dfc9090>]



[19]:

```
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
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)
```



[20]:
```
X
```

[20]:
```
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]])
```

[21]:
```
Y
```

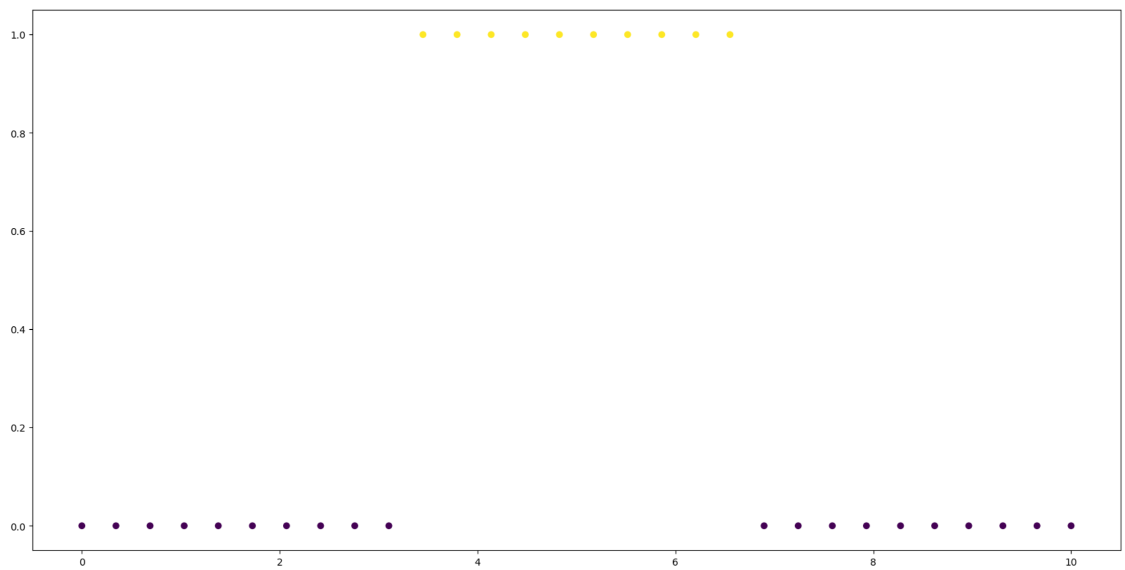
[21]:
```
array([[-10.  , -10.  , -10.  , …, -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, …,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , …,  -9.5 ,  -9.5 ,  -9.5 ],
       …,
       [  9.25,   9.25,   9.25, …,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , …,   9.5 ,   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?

[22]:
```python
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

[23]:
```python
plt.scatter(x,y, c=y)
```

[23]:
```
<matplotlib.collections.PathCollection at 0x7fe8449fc650>
```

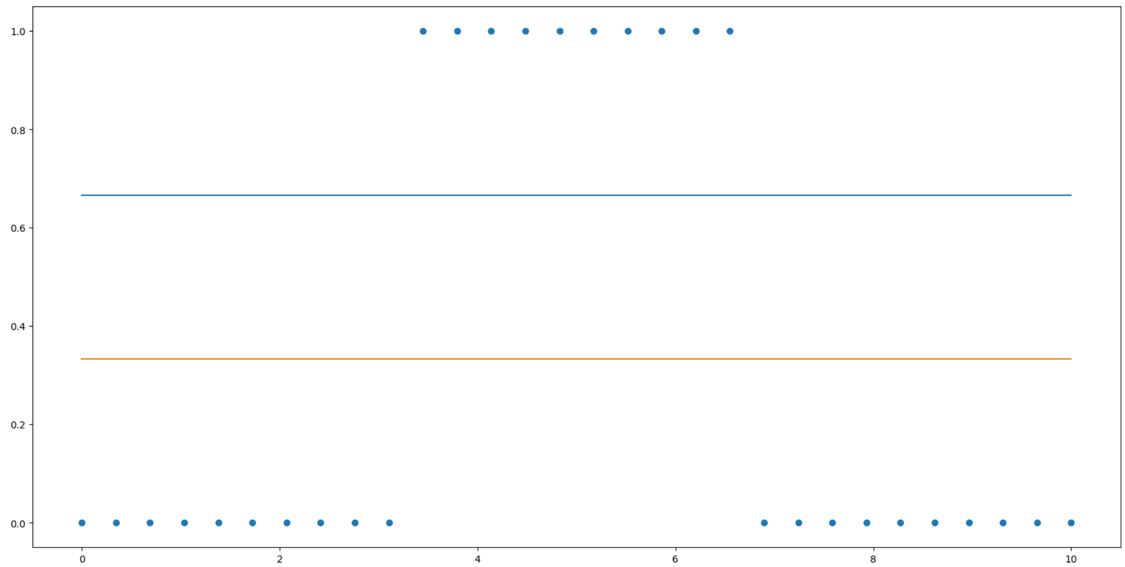


[24]:
```python
model.fit(x.reshape(-1, 1),y)
```

[24]:
```
LogisticRegression()
```

[25]:
```python
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

[25]:
```
[<matplotlib.lines.Line2D at 0x7fe844b113d0>,
 <matplotlib.lines.Line2D at 0x7fe842f1d1d0>]
```



[26]:
```
model1 = LogisticRegression()
model1.fit(x[:15].reshape(-1, 1),y[:15])
```

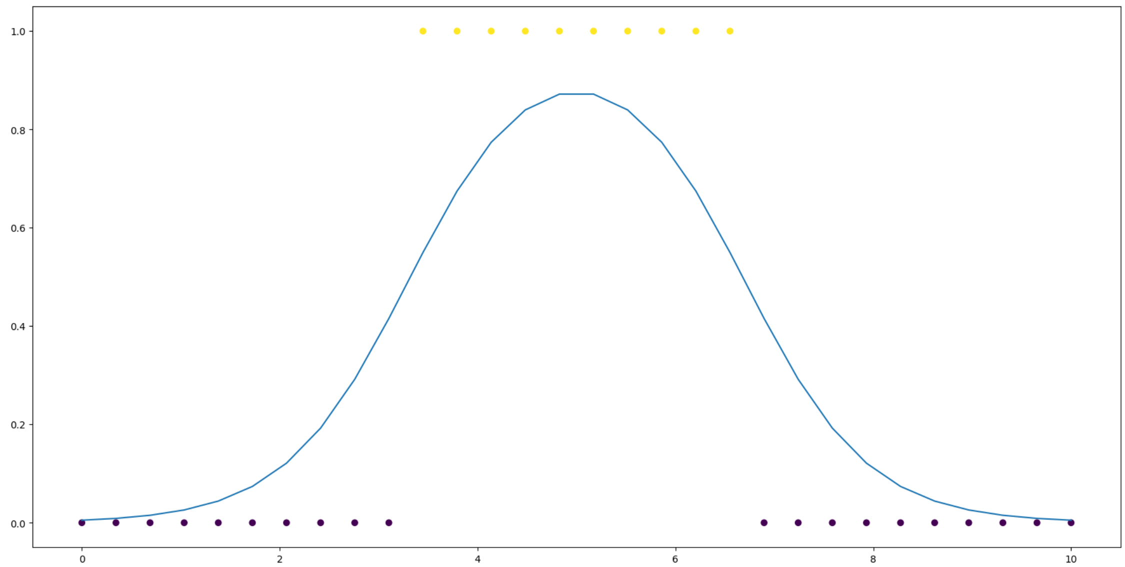
[26]: LogisticRegression()

[27]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

[27]: LogisticRegression()

[28]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model1.predict_proba(x.reshape(-1, 1))[:,1] * model2.
 ↪predict_proba(x.reshape(-1, 1))[:,1])
```

[28]: [<matplotlib.lines.Line2D at 0x7fe842f1e650>]



[30]:
```
df = pd.read_csv('adult.data', index_col=False)
golden = pd.read_csv('adult.test', index_col=False)
```

[31]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

[32]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

[33]:
```
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])
```

[34]:
```
df.salary.unique()
```

[34]: array([' <=50K', ' >50K'], dtype=object)

[35]:
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

[35]: array([' <=50K', ' >50K'], dtype=object)

[36]:
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

[36]:
```
LogisticRegression()
```

[37]:
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

[38]:
```
x.head()
```

[38]:
```
   age  workclass  fnlwgt  education  education-num  marital-status  \
0   39        7.0   77516        9.0             13             4.0
1   50        6.0   83311        9.0             13             2.0
2   38        4.0  215646       11.0              9             0.0
3   53        4.0  234721        1.0              7             2.0
4   28        4.0  338409        9.0             13             2.0

   occupation  relationship  race  sex  capital-gain  capital-loss  \
0         1.0           1.0   4.0  1.0          2174             0
1         4.0           0.0   4.0  1.0             0             0
2         6.0           1.0   4.0  1.0             0             0
3         6.0           0.0   2.0  1.0             0             0
4        10.0           5.0   2.0  0.0             0             0

   hours-per-week  native-country  salary
0              40            39.0     0.0
1              13            39.0     0.0
2              40            39.0     0.0
3              40            39.0     0.0
4              40             5.0     0.0
```

[39]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

[40]:
```
accuracy_score(x.salary, pred)
```

[40]:
```
0.8250360861152913
```

[41]:
```
confusion_matrix(x.salary, pred)
```

[41]:
```
array([[23300,  1420],
       [ 4277,  3564]])
```

[42]:
```
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
```

[43]:

```
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
```

2 Assignment

2.1 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

2.2 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

[47]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler

# Load the dataset
heart_data = pd.read_csv('heart.csv')

# Split the dataset into features and target variable
X = heart_data.drop('target', axis=1)
y = heart_data['target']

# Split the dataset into a training set and a test set
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Build a Logistic Regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Build a Decision Tree model
tree_model = DecisionTreeClassifier(max_depth=3)
tree_model.fit(X_train, y_train)

# Evaluate the Logistic Regression model
logistic_predictions = logistic_model.predict(X_test)
print("Logistic Regression Model:")
print(classification_report(y_test, logistic_predictions))
print(confusion_matrix(y_test, logistic_predictions))

# Evaluate the Decision Tree model
tree_predictions = tree_model.predict(X_test)
print("Decision Tree Model:")
print(classification_report(y_test, tree_predictions))
print(confusion_matrix(y_test, tree_predictions))
```

```
Logistic Regression Model:
              precision    recall  f1-score   support

           0       0.86      0.70      0.77       102
           1       0.75      0.88      0.81       103

    accuracy                           0.79       205
   macro avg       0.80      0.79      0.79       205
weighted avg       0.80      0.79      0.79       205

[[71 31]
 [12 91]]
Decision Tree Model:
              precision    recall  f1-score   support

           0       0.85      0.68      0.75       102
           1       0.73      0.88      0.80       103

    accuracy                           0.78       205
   macro avg       0.79      0.78      0.78       205
weighted avg       0.79      0.78      0.78       205

[[69 33]
 [12 91]]
```

```
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[50]:

```
# In this case, it appears that the Logistic Regression
# Model is the slightly superior model, because its f1-score
# and accuracy are higher among other metrics.
```

[49]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# Split the dataset into features and target variable
X = heart_data.drop('target', axis=1)
y = heart_data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train the original decision tree model
original_model = DecisionTreeClassifier(max_depth=3)  # Original depth
original_model.fit(X_train, y_train)

# Test the original decision tree model
original_y_pred = original_model.predict(X_test)

# Evaluate the performance of the original model
original_accuracy = accuracy_score(y_test, original_y_pred)
original_precision = precision_score(y_test, original_y_pred)
original_recall = recall_score(y_test, original_y_pred)
original_f1 = f1_score(y_test, original_y_pred)

# Train the overfit decision tree model
overfit_model = DecisionTreeClassifier(max_depth=10)  # Much deeper depth for␣
 ↪overfitting
overfit_model.fit(X_train, y_train)

# Test the overfit decision tree model
overfit_y_pred = overfit_model.predict(X_test)
```

```python
# Evaluate the performance of the overfit model
overfit_accuracy = accuracy_score(y_test, overfit_y_pred)
overfit_precision = precision_score(y_test, overfit_y_pred)
overfit_recall = recall_score(y_test, overfit_y_pred)
overfit_f1 = f1_score(y_test, overfit_y_pred)

# Compare the test results of the two models
print("Original Decision Tree Model:")
print("Accuracy:", original_accuracy)
print("Precision:", original_precision)
print("Recall:", original_recall)
print("F1 Score:", original_f1)

print("\nOverfit Decision Tree Model:")
print("Accuracy:", overfit_accuracy)
print("Precision:", overfit_precision)
print("Recall:", overfit_recall)
print("F1 Score:", overfit_f1)
```

```
Original Decision Tree Model:
Accuracy: 0.7804878048780488
Precision: 0.7338709677419355
Recall: 0.883495145631068
F1 Score: 0.8017621145374451

Overfit Decision Tree Model:
Accuracy: 0.9853658536585366
Precision: 1.0
Recall: 0.970873786407767
F1 Score: 0.9852216748768473
```

[51]:
```python
# In this case, the 'Overfit' Decision Tree Model
# outclasses the Logistic Regression Model as its accuracy
# is almost perfect and its precision actually is 1.0!
# Furthermore, the F1 Score is extremely high.
```