7

March 23, 2020

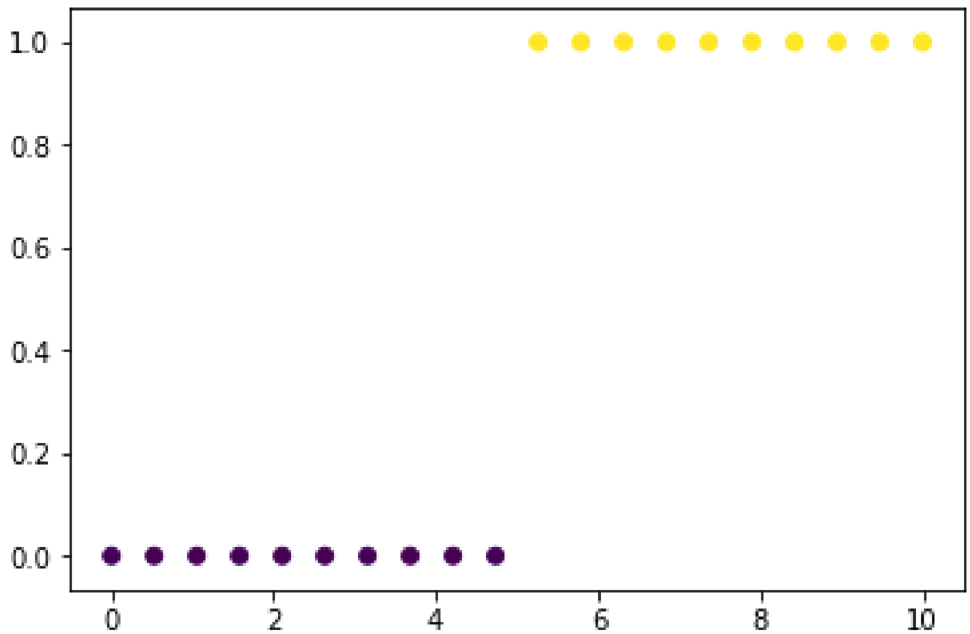
1 Assignment is at the bottom!

[17]:
```
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
```

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

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

[19]:
```
<matplotlib.collections.PathCollection at 0x1a1e9abc88>
```



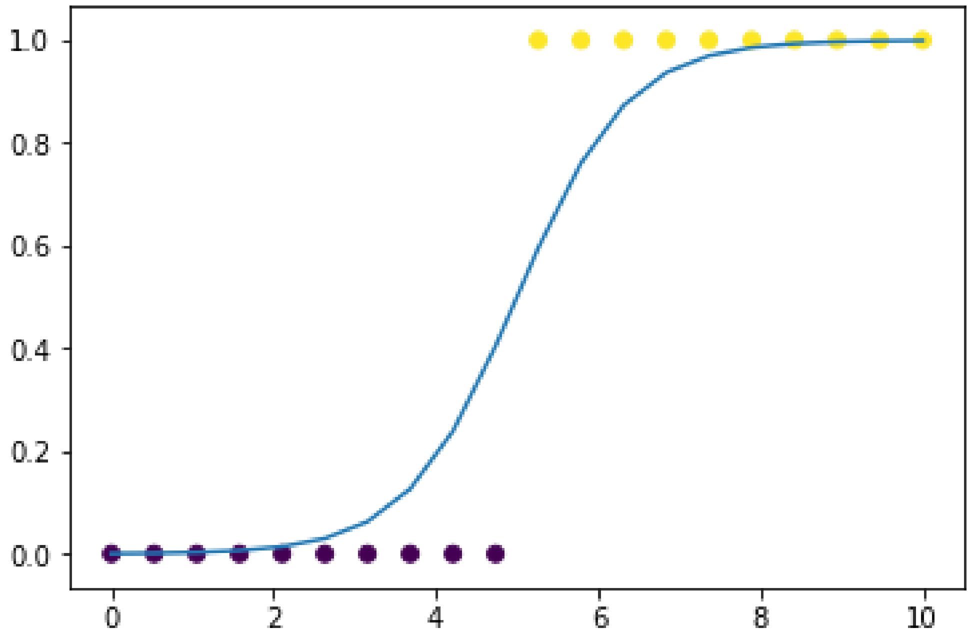
[20]:
```
model = LogisticRegression()
```

[21]:
```
model.fit(x.reshape(-1, 1),y)
```

[21]:
```
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)
```

[22]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1))[:,1])
```

[22]:
```
[<matplotlib.lines.Line2D at 0x1a1e9c01d0>]
```

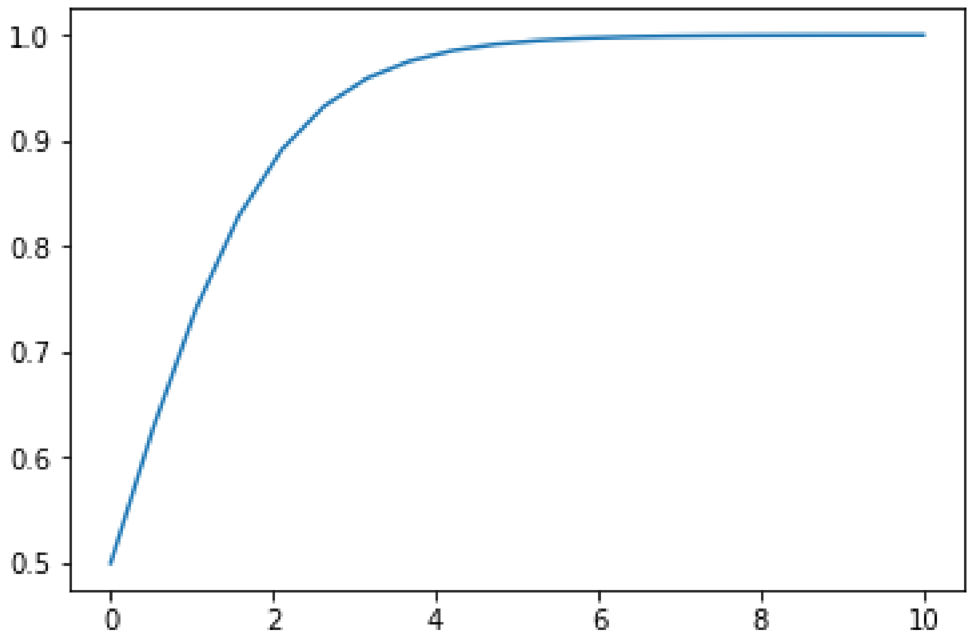


[23]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

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

[24]:
```
plt.plot(x, 1/(1+np.exp(-x)))
```

[24]: [<matplotlib.lines.Line2D at 0x1a1f432668>]

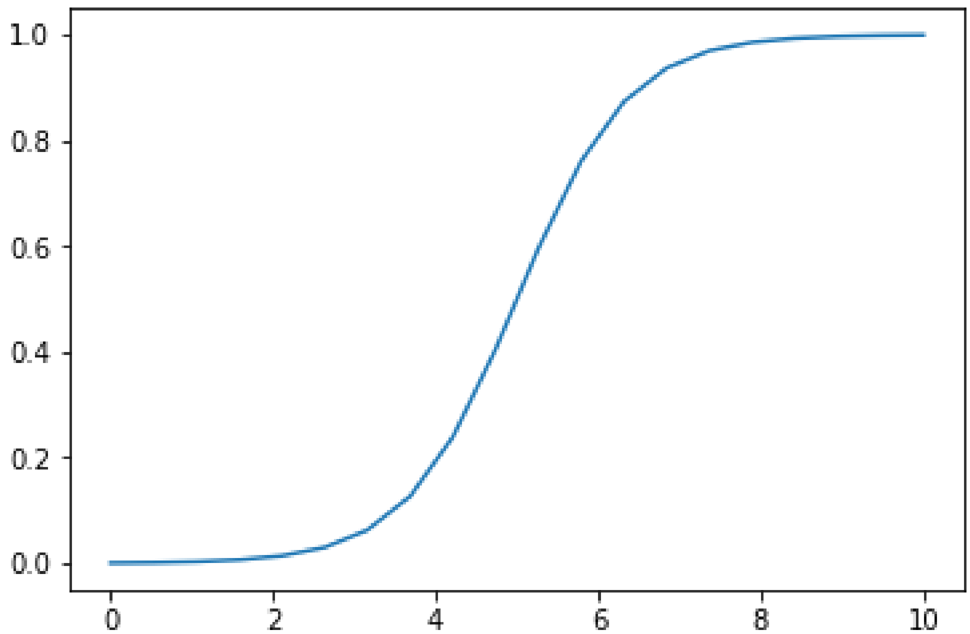


[25]:
```
b
```

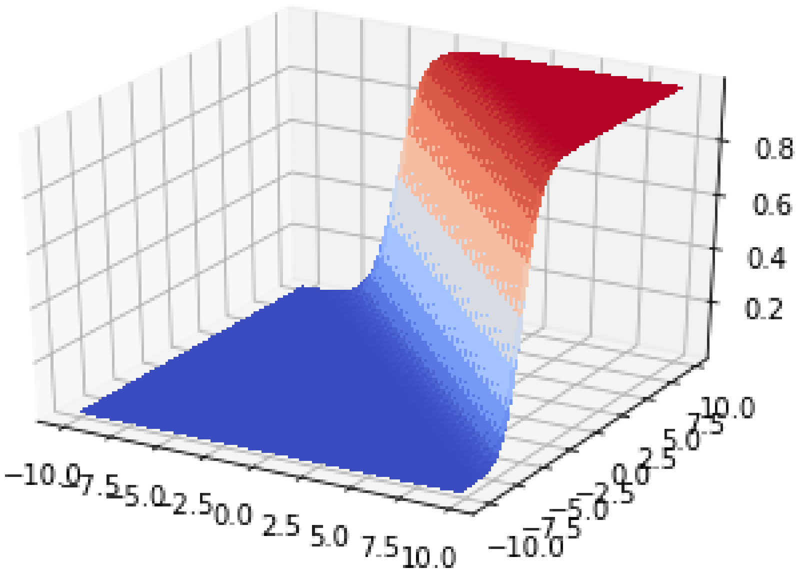
[25]: array([[1.46709085]])

[26]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```

[26]: [<matplotlib.lines.Line2D at 0x1a2012a4a8>]



[27]:
```
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)
```



[28]:
```
X
```

[28]:
```
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]])
```

[29]:
```
Y
```

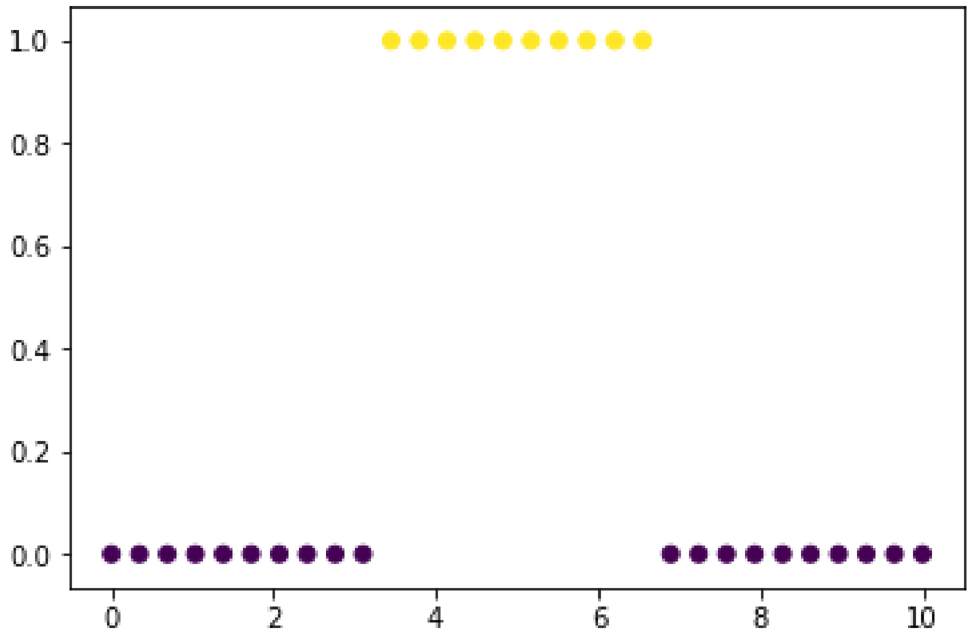
[29]:
```
array([[-10.  , -10.  , -10.  , …, -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, …,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , …,  -9.5 ,  -9.5 ,  -9.5 ],
       …,
       [  9.25,   9.25,   9.25, …,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , …,   9.5 ,   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?

[30]:
```
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

[31]:
```
plt.scatter(x,y, c=y)
```

[31]:
```
<matplotlib.collections.PathCollection at 0x1a20452048>
```

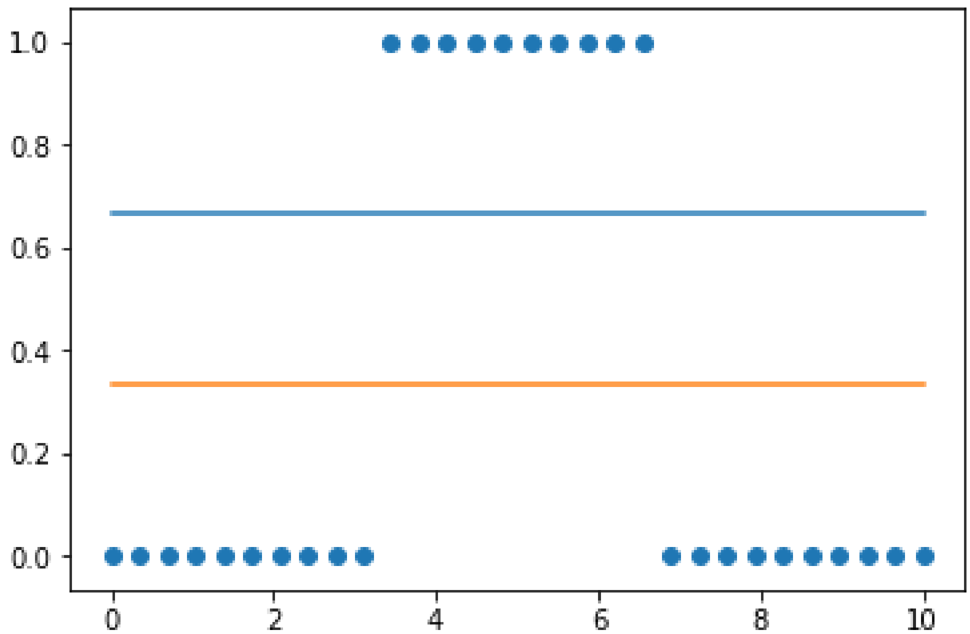


[32]:
```
model.fit(x.reshape(-1, 1),y)
```

[32]:
```
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)
```

[33]:
```
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

[33]:
```
[<matplotlib.lines.Line2D at 0x1a203d4940>,
 <matplotlib.lines.Line2D at 0x1a203627b8>]
```



[34]:
```
model1 = LogisticRegression()
model1.fit(x[:15].reshape(-1, 1),y[:15])
```

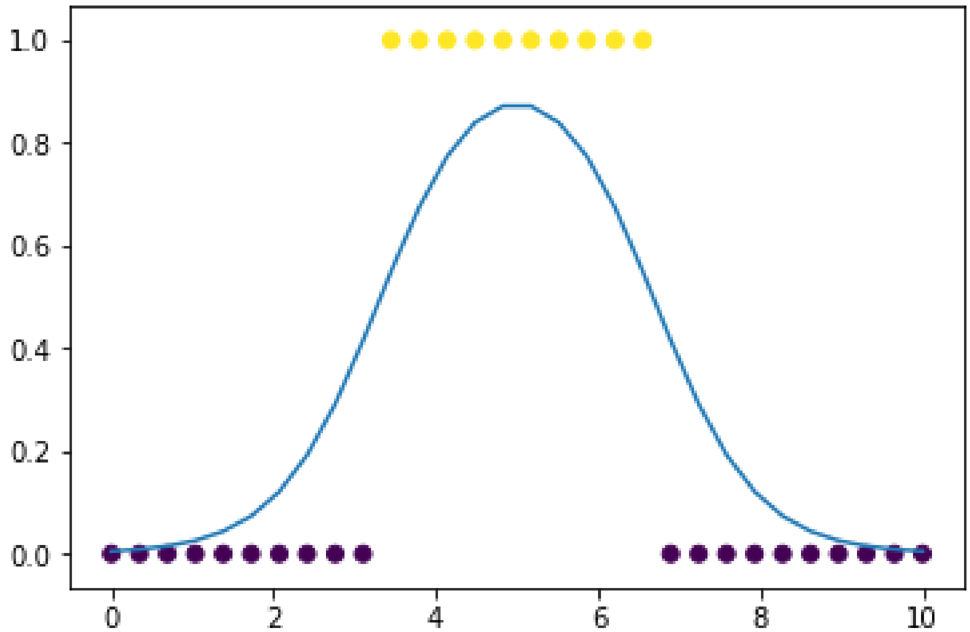
[34]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[35]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

[35]:
```
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)
```

[36]:
```
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])
```

[36]:
```
[<matplotlib.lines.Line2D at 0x1a202d7ba8>]
```



[37]:
```
df = pd.read_csv('../data/adult.data', index_col=False)
golden = pd.read_csv('../data/adult.test', index_col=False)
```

[38]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

[39]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

[40]:
```
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])
```

[41]:
```
df.salary.unique()
```

[41]:
```
array([' <=50K', ' >50K'], dtype=object)
```

[42]:
```python
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

[42]:
```
array([' <=50K', ' >50K'], dtype=object)
```

[43]:
```python
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

[43]:
```
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)
```

[44]:
```python
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

[45]:
```python
x.head()
```

[45]:
```
   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
```

[46]:
```python
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

[47]:
```python
accuracy_score(x.salary, pred)
```

[47]:
```
0.8250360861152913
```

[48]:
```
confusion_matrix(x.salary, pred)
```

[48]:
```
array([[23300,  1420],
       [ 4277,  3564]])
```

[49]:
```
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
```

[50]:
```
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 (create a train and a test set) and build 2 models: Logistic Regression and Decision Tree (shallow). Compare the test results.

2.2 2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, does the Logistic Regression have an improvement due to a lower variance?

[51]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

3 Workflow for Q1.

[52]:
```
student_data = pd.read_csv('/Users/Carancho/Documents/Johns Hopkins/ML:NN/
↪ml_dnn/student/student-por.csv',
                           delimiter = ';')
student_data.head(15)
```

[52]:
```
   school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
0      GP   F   18       U     GT3       A     4     4   at_home   teacher
1      GP   F   17       U     GT3       T     1     1   at_home     other
2      GP   F   15       U     LE3       T     1     1   at_home     other
3      GP   F   15       U     GT3       T     4     2    health  services
4      GP   F   16       U     GT3       T     3     3     other     other
5      GP   M   16       U     LE3       T     4     3  services     other
6      GP   M   16       U     LE3       T     2     2     other     other
7      GP   F   17       U     GT3       A     4     4     other   teacher
8      GP   M   15       U     LE3       A     3     2  services     other
9      GP   M   15       U     GT3       T     3     4     other     other
10     GP   F   15       U     GT3       T     4     4   teacher    health
11     GP   F   15       U     GT3       T     2     1  services     other
12     GP   M   15       U     LE3       T     4     4    health  services
13     GP   M   15       U     GT3       T     4     3   teacher     other
14     GP   M   15       U     GT3       A     2     2     other     other

    … famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0   …      4        3      4     1     1      3        4   0  11  11
1   …      5        3      3     1     1      3        2   9  11  11
2   …      4        3      2     2     3      3        6  12  13  12
3   …      3        2      2     1     1      5        0  14  14  14
4   …      4        3      2     1     2      5        0  11  13  13
5   …      5        4      2     1     2      5        6  12  12  13
6   …      4        4      4     1     1      3        0  13  12  13
7   …      4        1      4     1     1      1        2  10  13  13
8   …      4        2      2     1     1      1        0  15  16  17
9   …      5        5      1     1     1      5        0  12  12  13
10  …      3        3      3     1     2      2        2  14  14  14
11  …      5        2      2     1     1      4        0  10  12  13
12  …      4        3      3     1     3      5        0  12  13  12
13  …      5        4      3     1     2      3        0  12  12  13
14  …      4        5      2     1     1      3        0  14  14  15

[15 rows x 33 columns]
```

[53]:
```
# verify the data type of each column.
print(student_data.dtypes)
```

```
school        object
```

```
sex            object
age             int64
address        object
famsize        object
Pstatus        object
Medu            int64
Fedu            int64
Mjob           object
Fjob           object
reason         object
guardian       object
traveltime      int64
studytime       int64
failures        int64
schoolsup      object
famsup         object
paid           object
activities     object
nursery        object
higher         object
internet       object
romantic       object
famrel          int64
freetime        int64
goout           int64
Dalc            int64
Walc            int64
health          int64
absences        int64
G1              int64
G2              int64
G3              int64
dtype: object
```

[54]:
```
# extract features that are just int. Exapand categorical feats. later.
student_int_features = student_data.select_dtypes('int64').
 ↪drop(['G1','G2','G3'],axis=1)

# verify cols and check for missing data.
print(student_int_features.columns)
print(student_int_features.isna().sum())
```

```
Index(['age', 'Medu', 'Fedu', 'traveltime', 'studytime', 'failures', 'famrel',
       'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences'],
      dtype='object')
age           0
Medu          0
Fedu          0
```

```
traveltime    0
studytime     0
failures      0
famrel        0
freetime      0
goout         0
Dalc          0
Walc          0
health        0
absences      0
dtype: int64
```

[55]:
```
# extract the school period for model.
g1 = student_data.G1
g2 = student_data.G2
g3 = student_data.G3

# portugal grading scale is between 0-20. Passing is >9.5 rounded to anything␣
↪above 9.
# convert scores to pass or fail. (1:pass) (0:fail).
q1_grades = []
for val in g1:
    if val < 10:
        val = 0
        q1_grades.append(val)
    else:
        val = 1
        q1_grades.append(val)
```

[56]:
```
# split up testing and training sets.
x_train, x_test, y_train, y_test = train_test_split(student_int_features,
                                                    q1_grades,
                                                   test_size =.20)
```

[57]:
```
# build decision tree models utilizing the split data from cell above.
clf = DecisionTreeClassifier(max_depth = 1)
clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)
```

[58]:
```
# check accuracy for model at max_depth = 1
acc_score = accuracy_score(y_test, y_pred)
acc_score
```

[58]: 0.8692307692307693

[64]:
```
lgr_model = LogisticRegression()

# fit model based on scaled values.
lgr_model.fit(x_train, y_train)
lgr_pred = lgr_model.predict(x_test)
lgr_acc_score = accuracy_score(y_test, lgr_pred)
print(lgr_acc_score)
```

```
[1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1]
0.8615384615384616
```

```
/Users/Carancho/miniconda3/envs/ml_class_work/lib/python3.6/site-
packages/sklearn/linear_model/_logistic.py:940: 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

4 Q1 Answer

[60]:
```
print(f'Decision Tree model accuracy score of {acc_score:.2f} with only␣
↪numerical features.')
print(f'Logistic Regression accuracy score of  {lgr_acc_score:.2f} with only␣
↪numerical features')
print(classification_report(y_test, lgr_pred))
print(classification_report(y_test, y_pred))
```

```
Decision Tree model accuracy score of 0.87 with only numerical features.
Logistic Regression accuracy score of  0.86 with only numerical features
              precision    recall  f1-score   support

           0       0.88      0.47      0.61        30
           1       0.86      0.98      0.92       100

    accuracy                           0.86       130
   macro avg       0.87      0.72      0.76       130
weighted avg       0.86      0.86      0.84       130
```

```
              precision    recall  f1-score   support

           0       0.88      0.50      0.64        30
           1       0.87      0.98      0.92       100

    accuracy                           0.87       130
   macro avg       0.87      0.74      0.78       130
weighted avg       0.87      0.87      0.86       130
```

5 Q2 Answer

[61]:
```
clf2 = DecisionTreeClassifier(max_depth=10)
clf2.fit(x_train, y_train)

overfit_data = clf2.predict(x_test, y_test)
overfit_acc_score = accuracy_score(y_test, overfit_data)
```

[62]:
```
print(f'Allowing the model to be overfit will produce an accuracy score of␣
 ↪{overfit_acc_score:.2f}')
```

```
Allowing the model to be overfit will produce an accuracy score of 0.76
```

[63]:
```
print(classification_report(y_test, overfit_data))
```

```
              precision    recall  f1-score   support

           0       0.48      0.43      0.46        30
           1       0.83      0.86      0.85       100

    accuracy                           0.76       130
   macro avg       0.66      0.65      0.65       130
weighted avg       0.75      0.76      0.76       130
```

According to the models compiled, the results indicate that the logistic regression provide a better accuracy.

[ ]:
```
```