CoxAssignment06

Task: This assignment will help you understand and implement logistic regression trying to identify good system administrators.

Dataset: SystemAdministrators.csv

Data Imports

In [55]:
```
%matplotlib inline
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import matplotlib.pylab as plt
from sklearn.metrics import confusion_matrix
```

A management consultant is studying the roles played by experience and training in a system administrator's ability to complete a set of tasks in a specified amount of time. In particular, she is interested in discriminating between administrators who are able to complete given tasks within a specified time and those who are not. Data is collected on the performance of 75 randomly selected administrators.

Data Dictionary:

- Experience measures months of full-time system administrator experience | Numerical.
- Training measures the number of relevant training credits | Numerical.
- Completed is either Yes or No, according to whether or not the administrator completed the tasks | Categorical.

Load the data and print it.

In [56]:
```
SA = pd.read_csv('SystemAdministrators.csv')
SA.head(5)
```

Out[56]:

	Experience	Training	Completed task
0	10.9	4	Yes
1	9.9	4	Yes
2	10.4	6	Yes
3	13.7	6	Yes
4	9.4	8	Yes

Clean the data. If there is a space in a column header, add an underscore to the column name.

In [57]:
```
SA2 = SA.rename(columns={'Completed task': 'Completed_task'})
SA2
```

Out[57]:

	Experience	Training	Completed_task
0	10.9	4	Yes
1	9.9	4	Yes
2	10.4	6	Yes
3	13.7	6	Yes
4	9.4	8	Yes
...	...	...	...
70	5.6	4	No
71	5.9	8	No
72	6.4	6	No
73	3.8	4	No
74	5.3	4	No

75 rows × 3 columns

Change the categorical variable to numerical to include in your analysis. After this step, Completed_task should still be one column.

In [58]:
```
SA2['Completed_task'].replace(['No', 'Yes'],
                        [0, 1], inplace=True)
SA2
```

Out[58]:

	Experience	Training	Completed_task
0	10.9	4	1
1	9.9	4	1
2	10.4	6	1
3	13.7	6	1
4	9.4	8	1
...	...	...	...
70	5.6	4	0
71	5.9	8	0
72	6.4	6	0
73	3.8	4	0
74	5.3	4	0

75 rows × 3 columns

What are the predictors and what is the target variable?

```
- Predictors: Experience and Training
- Target Variable: Completed_task
```

Implement three logistic regression models. Tune the C and regularization parameter in your model.

In [100…

```
# model 1
predictors = ['Experience']
outcome = 'Completed_task'
```

In [101…

```
y = SA2[outcome]
X = SA2[predictors]
```

In [102…

```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [103…

```
logit_reg = LogisticRegression(penalty='none', C=1)
logit_reg.fit(train_X, train_y)
```

Out[103]:

```
LogisticRegression(C=1, penalty='none')
```

In [104…

```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -14.8706006744385
{'coefficient': array([1.76820318])}
```

In [105…

```
np.exp(1.76820318)
```

Out[105]:
```
5.860313966179437
```

In [107…
```
# model 2
predictors = ['Training']
outcome = 'Completed_task'
```

In [108…
```
y = SA2[outcome]
X = SA2[predictors]
```

In [109…
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [110…
```
logit_reg = LogisticRegression(penalty='l2',C=50)
logit_reg.fit(train_X, train_y)
```

Out[110]:
```
LogisticRegression(C=50)
```

In [111…
```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -2.4168081040812632
{'coefficient': array([0.27301591])}
```

In [112…
```
np.exp(1.7572771)
```

Out[112]:
```
5.796632236319038
```

In [114…
```
# model 3
predictors = ['Experience']
outcome = 'Completed_task'
```

In [115…
```
y = SA2[outcome]
X = SA2[predictors]
```

In [116…
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [117…
```
logit_reg = LogisticRegression(penalty='l2',C=100)
logit_reg.fit(train_X, train_y)
```

Out[117]:
```
LogisticRegression(C=100)
```

In [118…
```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -14.826944322182463
{'coefficient': array([1.76270219])}
```

In [119…
```
np.exp(1.76270219)
```

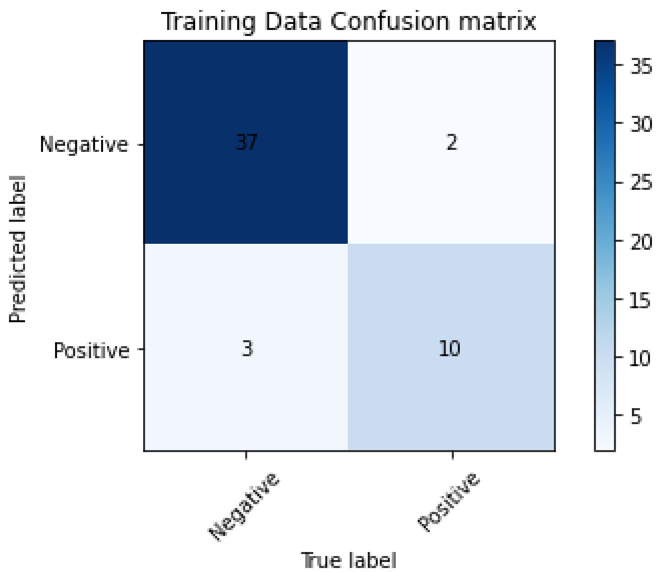
Out[119]:
```
5.828164944449608
```

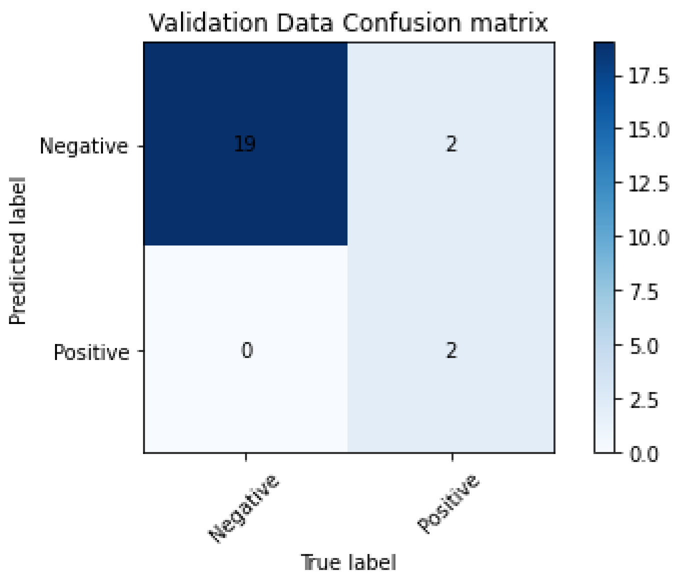
Evaluate your models. Please include a confusion matrix and accuracy score.

In [85]:
```
# model 1
print("Training set score: {:.3f}".format(logit_reg.score(train_X, train_y)))
print("Test set score: {:.3f}".format(logit_reg.score(valid_X, valid_y)))
```

```
Training set score: 0.904
Test set score: 0.913
```

In [106…
```
y_pred = logit_reg.predict(train_X)

cm = confusion_matrix(train_y, y_pred)

classes = ['Negative', 'Positive']

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Training Data Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.xlabel('True label')
plt.ylabel('Predicted label')
width, height = cm.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cm[x][y]), xy=(y, x),
                     horizontalalignment='center', verticalalignment='center')
plt.show()

y_pred = logit_reg.predict(valid_X)

cmtest = confusion_matrix(valid_y, y_pred)

classes = ['Negative', 'Positive']

plt.imshow(cmtest, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Validation Data Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.xlabel('True label')
plt.ylabel('Predicted label')
width, height = cmtest.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cmtest[x][y]), xy=(y, x),
                     horizontalalignment='center', verticalalignment='center')
plt.show()
```





In [92]:

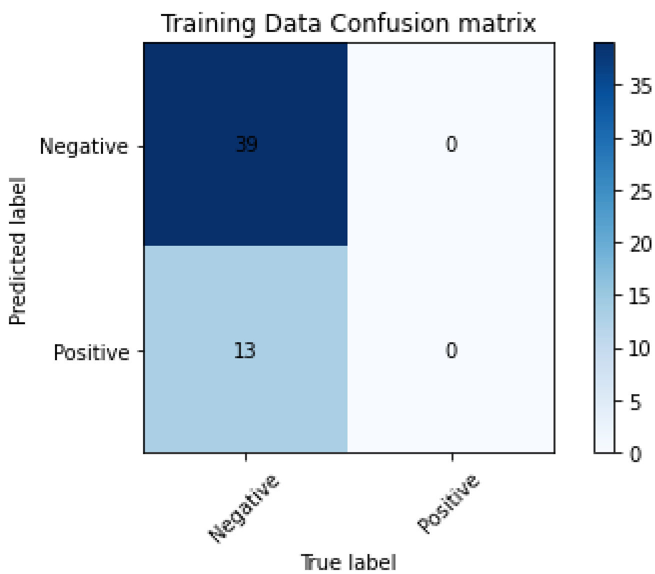
```
# model 2
print("Training set score: {:.3f}".format(logit_reg.score(train_X, train_y)))
print("Test set score: {:.3f}".format(logit_reg.score(valid_X, valid_y)))
```

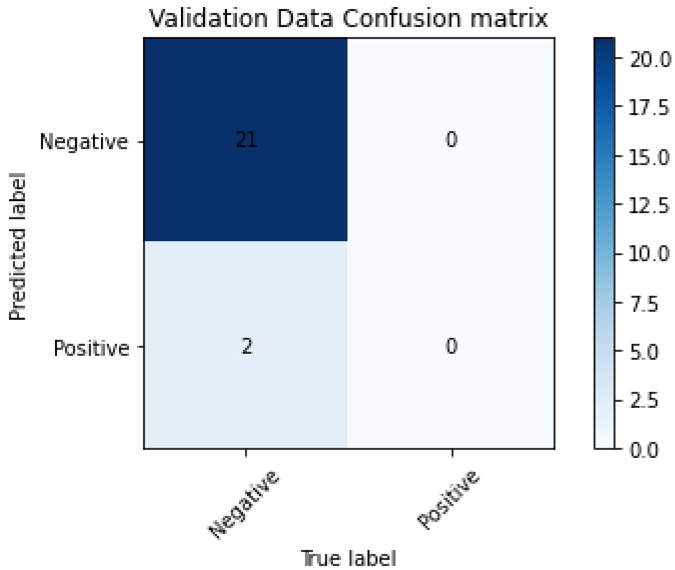
```
Training set score: 0.750
Test set score: 0.913
```

In [113…

```
y_pred = logit_reg.predict(train_X)

cm = confusion_matrix(train_y, y_pred)

classes = ['Negative', 'Positive']

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Training Data Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.xlabel('True label')
plt.ylabel('Predicted label')
```

```
width, height = cm.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cm[x][y]), xy=(y, x),
                     horizontalalignment='center', verticalalignment='center')
plt.show()

y_pred = logit_reg.predict(valid_X)

cmtest = confusion_matrix(valid_y, y_pred)

classes = ['Negative', 'Positive']

plt.imshow(cmtest, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Validation Data Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.xlabel('True label')
plt.ylabel('Predicted label')
width, height = cmtest.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cmtest[x][y]), xy=(y, x),
                     horizontalalignment='center', verticalalignment='center')
plt.show()
```





In [99]:

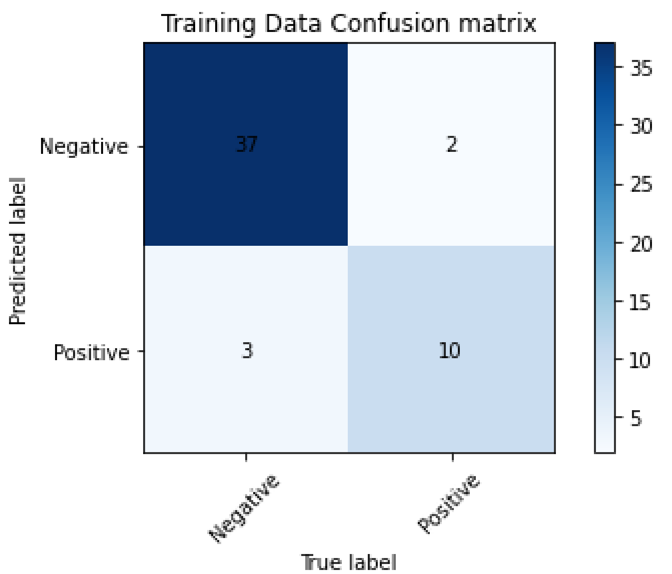
```
# model 3
print("Training set score: {:.3f}".format(logit_reg.score(train_X, train_y)))
print("Test set score: {:.3f}".format(logit_reg.score(valid_X, valid_y)))
```

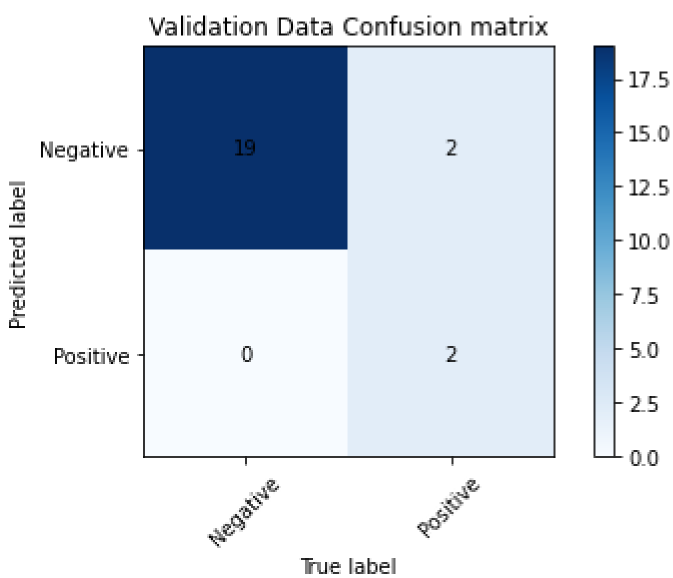
```
Training set score: 0.904
Test set score: 0.913
```

In [120…

```
y_pred = logit_reg.predict(train_X)

cm = confusion_matrix(train_y, y_pred)

classes = ['Negative', 'Positive']

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Training Data Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.tight_layout()
plt.xlabel('True label')
plt.ylabel('Predicted label')
width, height = cm.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cm[x][y]), xy=(y, x),
                     horizontalalignment='center', verticalalignment='center')
plt.show()

y_pred = logit_reg.predict(valid_X)

cmtest = confusion_matrix(valid_y, y_pred)

classes = ['Negative', 'Positive']

plt.imshow(cmtest, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Validation Data Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
```

```
plt.tight_layout()
plt.xlabel('True label')
plt.ylabel('Predicted label')
width, height = cmtest.shape
for x in range(width):
    for y in range(height):
        plt.annotate(str(cmtest[x][y]), xy=(y, x),
                     horizontalalignment='center', verticalalignment='center')
plt.show()
```





Using the confusion matrix and accuracy score, discuss how modifying C and the regularization parameter changed your model scores.

For model 1 and 3 I used the same predictor (Experience) but model 1 and a C=1 where model 3 had a C=100, this lead them having diffent coeffecients but only by 0.032146452. They also had similar training test scores and test set scores which were both very close to eachother which is ideal for the model. On the other hand, model 2, performed the worst with the predictor being Training. Model 2's training test score and test score were far from each other which leads me to believe with this data that Experience is the best indicator of whether a task will be completed or not.

Using your best model, write code to output the coefficients. Next, interpret the coefficients. Which ones are significant and which ones hold the greatest magnitude?

hint you must transform the coefficients using exponentiate to interpret them. There is also a rule you can use to interpret them once you get them into that form.

In [139…
```
# model 3
predictors = ['Experience']
outcome = 'Completed_task'
```

In [140…
```
y = SA2[outcome]
X = SA2[predictors]
```

In [141…
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [147…
```
logit_reg = LogisticRegression(penalty='l2',C=100)
logit_reg.fit(train_X, train_y)
```

Out[147]:
```
LogisticRegression(C=100)
```

In [148…
```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -14.826944322182463
{'coefficient': array([1.76270219])}
```

In [149…
```
np.exp(1.76270219)
```

Out[149]:
```
5.828164944449608
```

In [152…
```
logit_reg = LogisticRegression(penalty='l2',C=1000)
logit_reg.fit(train_X, train_y)
```

Out[152]:
```
LogisticRegression(C=1000)
```

In [153…
```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -14.86620736262152
{'coefficient': array([1.7676496])}
```

In [154…
```
np.exp(1.7676496)
```

Out[154]:
```
5.857070711357366
```

In [155…
```
logit_reg = LogisticRegression(penalty='l2',C=500)
logit_reg.fit(train_X, train_y)
```

Out[155]:
```
LogisticRegression(C=500)
```

In [156…

```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -14.861820245789318
{'coefficient': array([1.76709681])}
```

In [157…

```
np.exp(1.76709681)
```

Out[157]:

```
5.853833875966378
```

With the coeffecient from model 3 being greater than 1 for all three outcome shows that an increase in Experience feature increases the odds of completing a task (outcome variable).

In [ ]: