Cox Deliverable 4

CIS 368

Data Dictionary:

```
- gender: Male or Female response from participant
- age: Age in years of the individual
- avg_glucose: Average glucose levels of the individual
- bmi: Body mass index of the individual
- smoking_status: is the individual a former smoker, never smoked, or
currently smokes
- stroke_poss: is the individual likely to have a stroke
```

Problem Statement:

How likely are males and females likely to have a stroke based on their smoking status, average glucose levels, and their body mass index.

Predictor Variables:

```
- gender
- age
- avg_glucose
- bmi
- smoking_status
```

Target Variable:

```
- stroke_poss
```

In [73]:

```
%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
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn import tree
import graphviz

from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.datasets import make_moons
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
```

Preparing the Data

In [33]:

```
stroke = pd.read_csv('stroke.csv')
stroke.head()
```

Out[33]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gl
0	9046	Male	67.0	0	1	Yes	Private	Urban	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	
2	31112	Male	80.0	0	1	Yes	Private	Rural	
3	60182	Female	49.0	0	0	Yes	Private	Urban	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	

In [34]:

```
stroke2 = stroke.drop(['id', 'hypertension', 'heart_disease', 'ever_married', 'work_ty
stroke2
```

Out[34]:

	gender	age	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	228.69	36.6	formerly smoked	1
1	Female	61.0	202.21	NaN	never smoked	1
2	Male	80.0	105.92	32.5	never smoked	1
3	Female	49.0	171.23	34.4	smokes	1
4	Female	79.0	174.12	24.0	never smoked	1
...	...	...	...	...	...	...
5105	Female	80.0	83.75	NaN	never smoked	0
5106	Female	81.0	125.20	40.0	never smoked	0
5107	Female	35.0	82.99	30.6	never smoked	0
5108	Male	51.0	166.29	25.6	formerly smoked	0
5109	Female	44.0	85.28	26.2	Unknown	0

5110 rows × 6 columns

In [35]:

```
stroke3 = stroke2.rename(columns={'stroke': 'stroke_possibility'})
stroke3
```

Out[35]:

	gender	age	avg_glucose_level	bmi	smoking_status	stroke_possibility
0	Male	67.0	228.69	36.6	formerly smoked	1
1	Female	61.0	202.21	NaN	never smoked	1
2	Male	80.0	105.92	32.5	never smoked	1
3	Female	49.0	171.23	34.4	smokes	1
4	Female	79.0	174.12	24.0	never smoked	1
...	...	...	...	...	...	...
5105	Female	80.0	83.75	NaN	never smoked	0
5106	Female	81.0	125.20	40.0	never smoked	0
5107	Female	35.0	82.99	30.6	never smoked	0
5108	Male	51.0	166.29	25.6	formerly smoked	0
5109	Female	44.0	85.28	26.2	Unknown	0

5110 rows × 6 columns

In [36]:

```
stroke3['gender'].replace(['Male', 'Female', 'Other'],
                        [0, 1, 2], inplace=True)
stroke3
```

Out[36]:

	gender	age	avg_glucose_level	bmi	smoking_status	stroke_possibility
0	0	67.0	228.69	36.6	formerly smoked	1
1	1	61.0	202.21	NaN	never smoked	1
2	0	80.0	105.92	32.5	never smoked	1
3	1	49.0	171.23	34.4	smokes	1
4	1	79.0	174.12	24.0	never smoked	1
...	...	...	...	...	...	...
5105	1	80.0	83.75	NaN	never smoked	0
5106	1	81.0	125.20	40.0	never smoked	0
5107	1	35.0	82.99	30.6	never smoked	0
5108	0	51.0	166.29	25.6	formerly smoked	0
5109	1	44.0	85.28	26.2	Unknown	0

5110 rows × 6 columns

In [37]:

```
stroke3['smoking_status'].replace(['formerly smoked', 'never smoked', 'smokes', 'Unknc
                                   [0, 1, 2, 3], inplace=True)
stroke3
```

Out[37]:

	gender	age	avg_glucose_level	bmi	smoking_status	stroke_possibility
0	0	67.0	228.69	36.6	0	1
1	1	61.0	202.21	NaN	1	1
2	0	80.0	105.92	32.5	1	1
3	1	49.0	171.23	34.4	2	1
4	1	79.0	174.12	24.0	1	1
...	...	...	...	...	...	...
5105	1	80.0	83.75	NaN	1	0
5106	1	81.0	125.20	40.0	1	0
5107	1	35.0	82.99	30.6	1	0
5108	0	51.0	166.29	25.6	0	0
5109	1	44.0	85.28	26.2	3	0

5110 rows × 6 columns

In [38]:

```
stroke4 = stroke3.dropna()
```

In [39]:

```
stroke4
```

Out[39]:

	gender	age	avg_glucose_level	bmi	smoking_status	stroke_possibility
0	0	67.0	228.69	36.6	0	1
2	0	80.0	105.92	32.5	1	1
3	1	49.0	171.23	34.4	2	1
4	1	79.0	174.12	24.0	1	1
5	0	81.0	186.21	29.0	0	1
...	...	...	...	...	...	...
5104	1	13.0	103.08	18.6	3	0
5106	1	81.0	125.20	40.0	1	0
5107	1	35.0	82.99	30.6	1	0
5108	0	51.0	166.29	25.6	0	0
5109	1	44.0	85.28	26.2	3	0

4909 rows × 6 columns

Logistic Regrssion

In [40]:
```
predictors = ['smoking_status', 'bmi', 'avg_glucose_level', 'age', 'gender']
outcome = 'stroke_possibility'
```

In [41]:
```
y = stroke4[outcome]
X = stroke4[predictors]
```

In [42]:
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [43]:
```
logit_reg = LogisticRegression(penalty='l2', C=50)
logit_reg.fit(train_X, train_y)
```

Out[43]:
```
LogisticRegression(C=50)
```

In [44]:
```
print('intercept', logit_reg.intercept_[0])
print({'coefficient': logit_reg.coef_[0]})
```

```
intercept -7.94889492232567
{'coefficient': array([-0.06478192,  0.00576886,  0.00572133,  0.07126623, -0.0011293
9])}
```

In [45]:
```
np.exp(-0.06478192)
```

Out[45]:
```
0.9372718413343792
```

In [46]:
```
np.exp(0.00576886)
```

Out[46]:
```
1.0057855319167497
```

In [47]:
```
np.exp(0.00572133)
```

Out[47]:
```
1.0057377280664852
```

In [48]:
```
np.exp(0.07126623)
```

Out[48]:
```
1.0738670834483637
```

In [49]:
```
np.exp(-0.00112939)
```

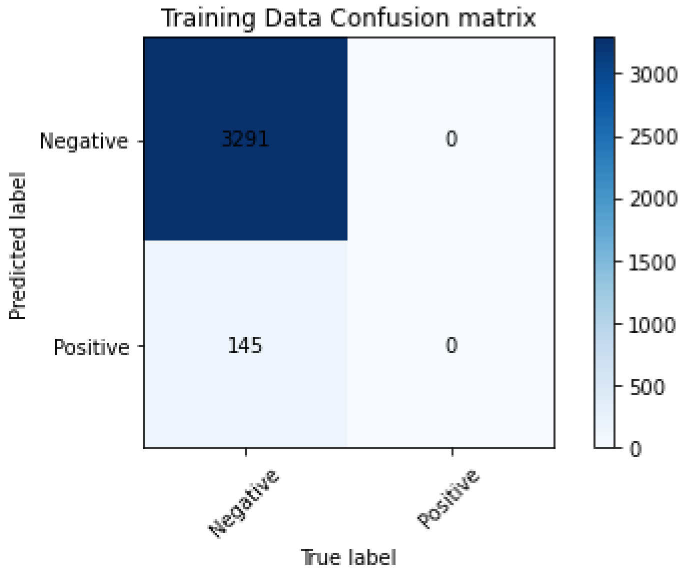
Out[49]:
```
0.9988712475208602
```

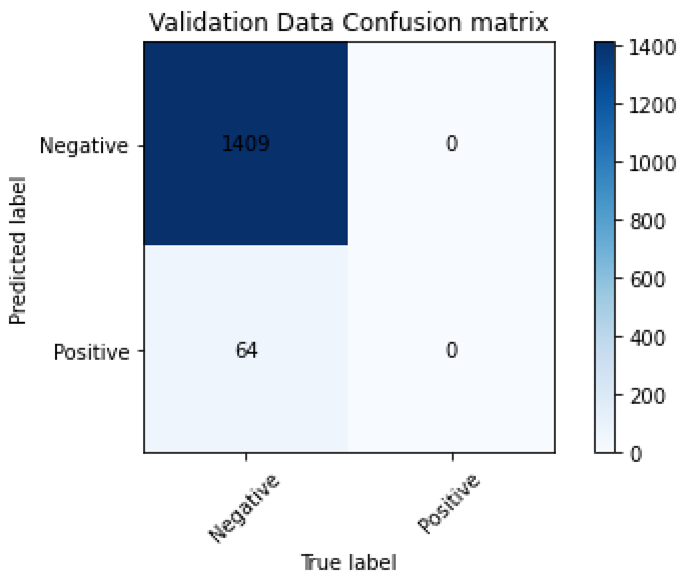
In [50]:
```
print("Training set score: {:.3f}".format(logit_reg.score(train_X, train_y)))
print("Test set score: {:.3f}".format(logit_reg.score(valid_X, valid_y)))
```

```
Training set score: 0.958
Test set score: 0.957
```

In [51]:
```
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()
```





Decision Tree

In [52]:
```
X = stroke4.drop(columns=['stroke_possibility'])
y = stroke4['stroke_possibility']
```

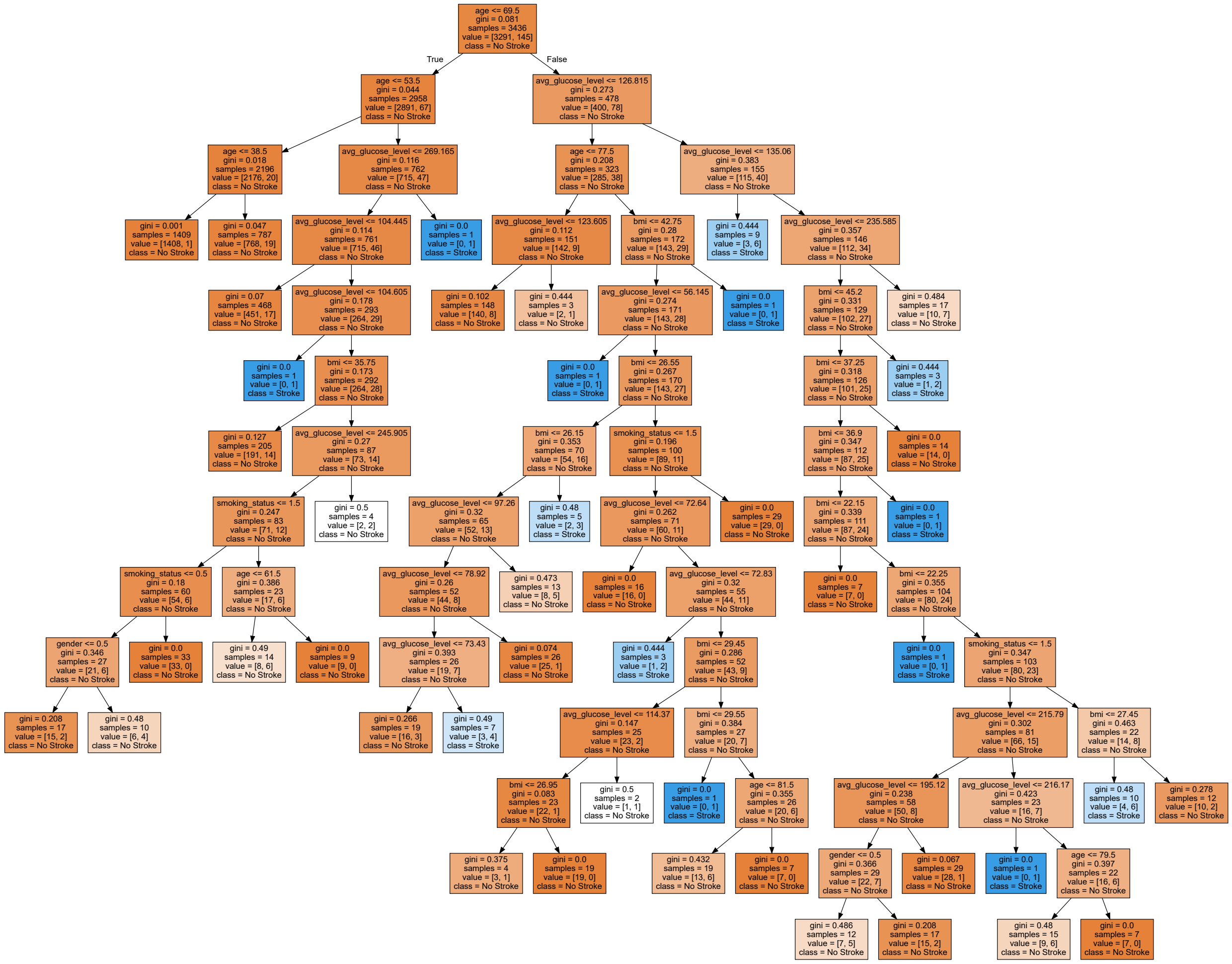
In [53]:
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_stat
```

In [54]:
```
strokeTree = DecisionTreeClassifier(max_depth=30, min_samples_split=20, min_impurity_d
strokeTree.fit(train_X, train_y)
```

Out[54]:
```
DecisionTreeClassifier(max_depth=30, min_impurity_decrease=0.0001,
                       min_samples_split=20)
```

In [55]:
```
dot_data = tree.export_graphviz(strokeTree, out_file=None,
                                feature_names=train_X.columns,
                                filled=True, class_names=['No Stroke', 'Stroke'])
graph = graphviz.Source(dot_data, format="png")
graph
```

Out[55]:



In [56]:
```
y_train_pred = strokeTree.predict(train_X)
y_train_pred
```

Out[56]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [57]:
```
# create confusion matrix comparing between validation data and prediced
cm = confusion_matrix(train_y, y_train_pred)

# calculate the accuracy score
accuracy = accuracy_score(train_y, y_train_pred)

print("Confusion Matrix:")
print(cm)
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
[[3277   14]
 [ 114   31]]
Accuracy: 0.9627473806752037
```

In [58]:
```
y_pred = strokeTree.predict(valid_X)
y_pred
```

Out[58]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [59]:
```
# create confusion matrix comparing between validation data and prediced
cm = confusion_matrix(valid_y, y_pred)
```

```
# calculate the accuracy score
accuracy = accuracy_score(valid_y, y_pred)

print("Confusion Matrix:")
print(cm)
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
[[1395   14]
 [  61    3]]
Accuracy: 0.9490835030549898
```

Support Vector Machine

In [130…

```
X = stroke4.drop(columns=['stroke_possibility'])
y = stroke4['stroke_possibility']
```

In [131…

```
rbf_kernel_svm_clf = Pipeline([
("scaler", StandardScaler()),
("svm_clf", SVC(kernel="rbf", gamma=50, C=20))
])
rbf_kernel_svm_clf.fit(X, y)
```

Out[131]:

```
Pipeline(steps=[('scaler', StandardScaler()), ('svm_clf', SVC(C=20, gamma=50))])
```

In [132…

```
y_train_pred = rbf_kernel_svm_clf.predict(X)
y_train_pred
```

Out[132]:

```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

In [133…

```
cm = confusion_matrix(y, y_train_pred)
# calculate the accuracy score
accuracy = accuracy_score(y, y_train_pred)
print("Confusion Matrix:")
print(cm)
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
[[4700    0]
 [   1  208]]
Accuracy: 0.9997962925239356
```

In [ ]:

```
```