final-Copy1

June 1, 2021

1 CSM148 Final Project

The report includes all the important graphs, tables and results for the pipeline process, model parameters and performance.

This code part includes results for 2 different methods, of balancing dataset before/after splitting into training and testing set. Regarding to different order of split and balance, the model performance would be different. It is said that splitting before balancing would produce a more reliable result, which would be the main part of the code submission. I still attached the result of another ordering at the end, just for reference.

Thank you!

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import sklearn.metrics.cluster as smc
from sklearn.model_selection import KFold


from matplotlib import pyplot
import itertools

%matplotlib inline

import random

random.seed(42)
```

[2]:
```python
# Helper function allowing you to export a graph
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

[3]:
```python
# Helper function that allows you to draw nicely formatted confusion matrices
def draw_confusion_matrix(y, yhat, classes):
    '''
        Draws a confusion matrix for the given target and predictions
        Adapted from scikit-learn and discussion example.
    '''
    plt.cla()
    plt.clf()
    matrix = confusion_matrix(y, yhat)
    plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.colorbar()
    num_classes = len(classes)
    plt.xticks(np.arange(num_classes), classes, rotation=90)
    plt.yticks(np.arange(num_classes), classes)

    fmt = 'd'
    thresh = matrix.max() / 2.
    for i, j in itertools.product(range(matrix.shape[0]), range(matrix.
 ↪shape[1])):
        plt.text(j, i, format(matrix[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if matrix[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
```

1.1 Part 1. Basic Statistics

[4]:
```python
hd=pd.read_csv("healthcare-dataset-stroke-data.csv")
```

[5]:
```python
hd.head()
```

[5]:
```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
```

```
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2        Private          Rural             105.92  32.5     never smoked
3        Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```

[6]:
```
hd.describe()
```

[6]:
```
                 id          age  hypertension  heart_disease  \
count   5110.000000  5110.000000   5110.000000    5110.000000
mean   36517.829354    43.226614      0.097456       0.054012
std    21161.721625    22.612647      0.296607       0.226063
min       67.000000     0.080000      0.000000       0.000000
25%    17741.250000    25.000000      0.000000       0.000000
50%    36932.000000    45.000000      0.000000       0.000000
75%    54682.000000    61.000000      0.000000       0.000000
max    72940.000000    82.000000      1.000000       1.000000

       avg_glucose_level          bmi       stroke
count        5110.000000  4909.000000  5110.000000
mean          106.147677    28.893237     0.048728
std            45.283560     7.854067     0.215320
min            55.120000    10.300000     0.000000
25%            77.245000    23.500000     0.000000
50%            91.885000    28.100000     0.000000
75%           114.090000    33.100000     0.000000
max           271.740000    97.600000     1.000000
```

[7]:
```
hd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
```

```
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

1.1.1 null values

[8]:
```
hd[hd.isnull().any(axis=1)]
```

[8]:
```
         id  gender   age  hypertension  heart_disease ever_married  \
1     51676  Female  61.0             0              0          Yes
8     27419  Female  59.0             0              0          Yes
13     8213    Male  78.0             0              1          Yes
19    25226    Male  57.0             0              1           No
27    61843    Male  58.0             0              0          Yes
…       …     …   …          …             …           …
5039  42007    Male  41.0             0              0           No
5048  28788    Male  40.0             0              0          Yes
5093  32235  Female  45.0             1              0          Yes
5099   7293    Male  40.0             0              0          Yes
5105  18234  Female  80.0             1              0          Yes

          work_type Residence_type  avg_glucose_level  bmi   smoking_status  \
1     Self-employed          Rural             202.21  NaN     never smoked
8           Private          Rural              76.15  NaN          Unknown
13          Private          Urban             219.84  NaN          Unknown
19         Govt_job          Urban             217.08  NaN          Unknown
27          Private          Rural             189.84  NaN          Unknown
…             …             …                 …  …              …
5039        Private          Rural              70.15  NaN  formerly smoked
5048        Private          Urban             191.15  NaN           smokes
5093       Govt_job          Rural              95.02  NaN           smokes
5099        Private          Rural              83.94  NaN           smokes
5105        Private          Urban              83.75  NaN     never smoked

      stroke
1          1
8          1
```

```
13          1
19          1
27          1
…         …
5039        0
5048        0
5093        0
5099        0
5105        0

[201 rows x 12 columns]
```

[9]:
```
hd['smoking_status'].unique()
```

[9]:
```
array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

[10]:
```
len(hd.loc[hd['smoking_status']=="Unknown"])
```

[10]:
```
1544
```

[11]:
```
len(hd.loc[(hd['smoking_status']=="Unknown") & (hd['bmi'].isnull())])
```

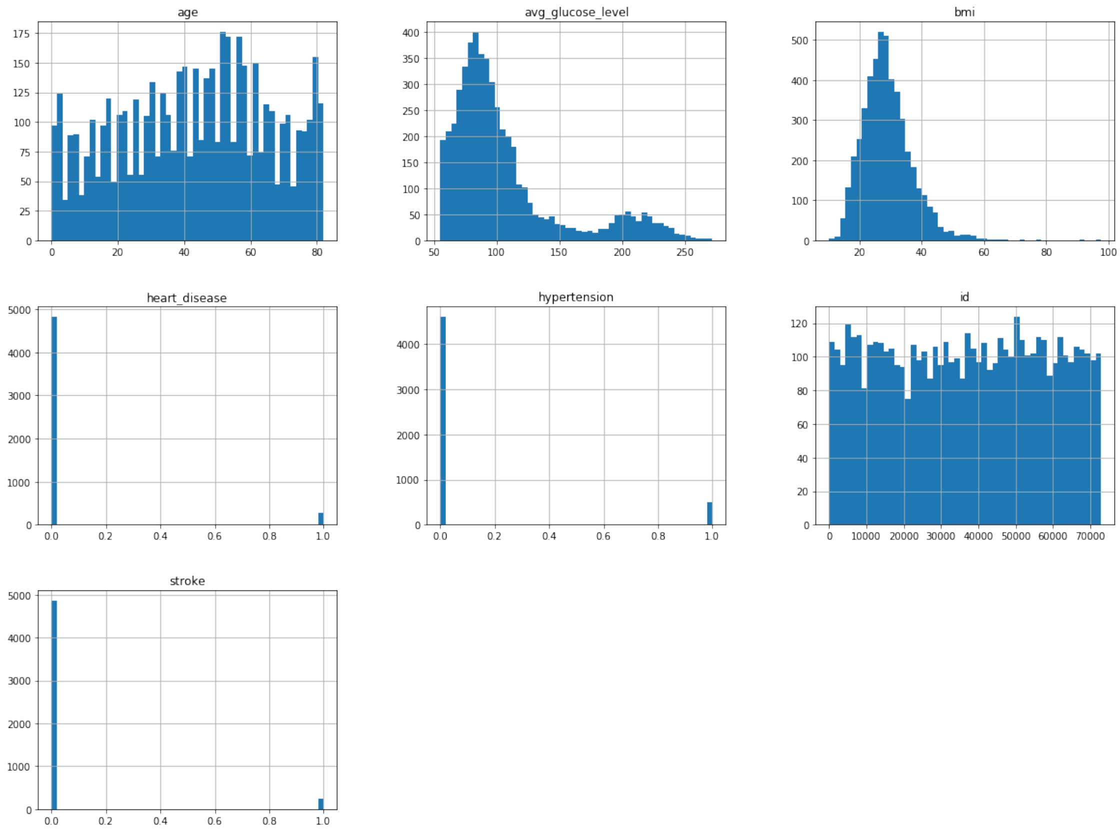
[11]:
```
61
```

[12]:
```
print(hd['ever_married'].unique())
print(hd['work_type'].unique())
print(hd['Residence_type'].unique())
```

```
['Yes' 'No']
['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
['Urban' 'Rural']
```

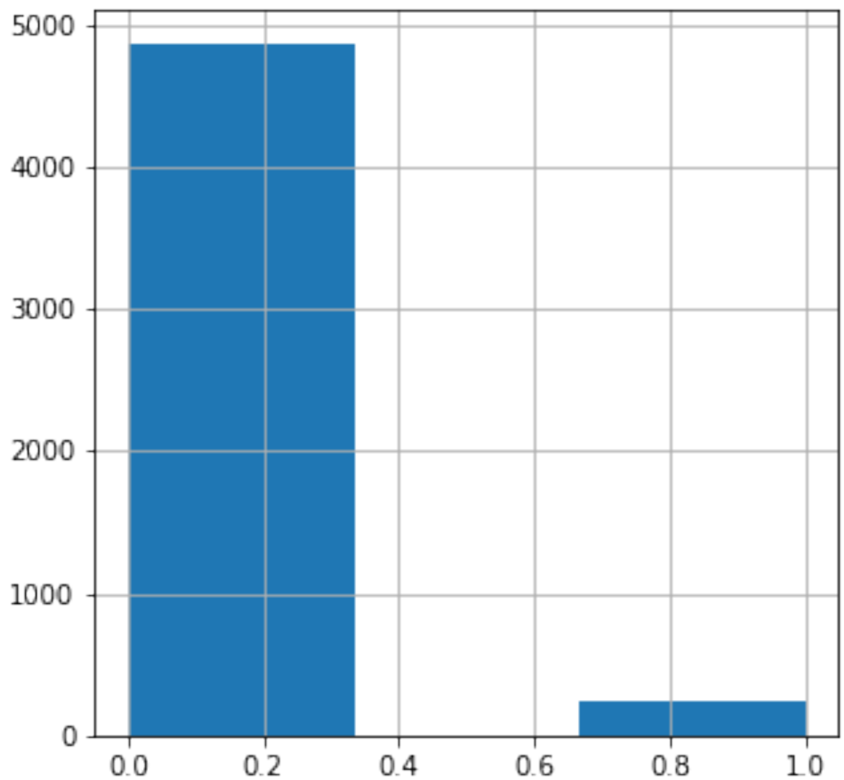
1.1.2 histogram

[13]:
```
hd.hist(bins=50, figsize=(20,15))
plt.show()
```



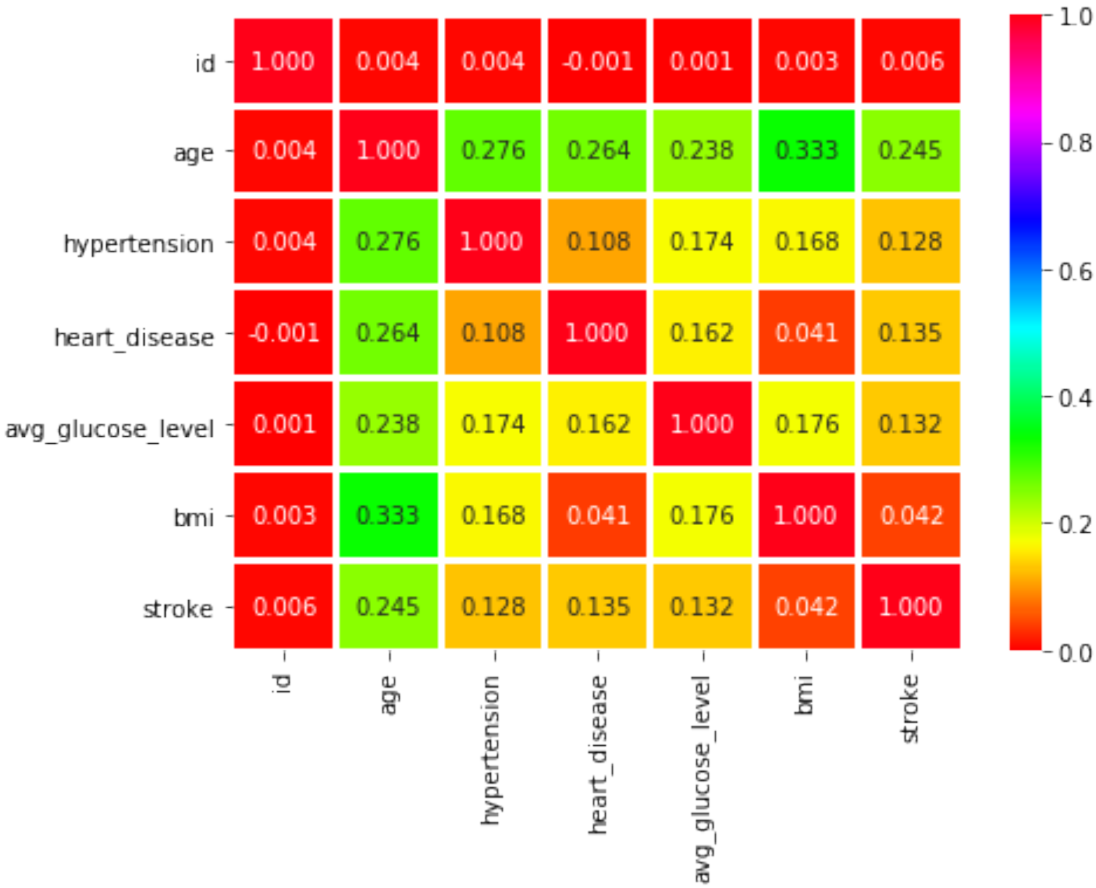
[14]:
```
hd['stroke'].hist(bins=3, figsize=(5,5))
hd['stroke'].value_counts()
```

[14]:
```
0    4861
1     249
Name: stroke, dtype: int64
```



1.1.3 Correlation

[15]:
```
plt.figure(figsize=(7,5))
sns.heatmap(hd.corr(),annot=True,cmap='hsv',fmt='.3f',linewidths=2)
plt.show()
```



1.2 Part 2. Prepare the Data

[16]:
```
hd=hd.drop(columns=['id'])
```

1.2.1 Imputation

[17]:
```
# bmi Null value --> average
#hd['bmi'].mean()
hd=hd.fillna(hd['bmi'].mean())
hd.head()
```

[17]:
```
   gender   age  hypertension  heart_disease ever_married      work_type  \
0    Male  67.0             0              1          Yes        Private
1  Female  61.0             0              0          Yes  Self-employed
2    Male  80.0             0              1          Yes        Private
3  Female  49.0             0              0          Yes        Private
4  Female  79.0             1              0          Yes  Self-employed
```

```
  Residence_type  avg_glucose_level        bmi   smoking_status  stroke
0          Urban             228.69  36.600000  formerly smoked       1
1          Rural             202.21  28.893237     never smoked       1
2          Rural             105.92  32.500000     never smoked       1
3          Urban             171.23  34.400000           smokes       1
4          Rural             174.12  24.000000     never smoked       1
```

1.2.2 Augmentation

[18]:
```python
# bmi * glucose
hd['fat_bsugar']=hd['bmi']*hd['avg_glucose_level']
hd.head()
```

[18]:
```
   gender   age  hypertension  heart_disease ever_married      work_type  \
0    Male  67.0             0              1          Yes        Private
1  Female  61.0             0              0          Yes  Self-employed
2    Male  80.0             0              1          Yes        Private
3  Female  49.0             0              0          Yes        Private
4  Female  79.0             1              0          Yes  Self-employed

  Residence_type  avg_glucose_level        bmi   smoking_status  stroke  \
0          Urban             228.69  36.600000  formerly smoked       1
1          Rural             202.21  28.893237     never smoked       1
2          Rural             105.92  32.500000     never smoked       1
3          Urban             171.23  34.400000           smokes       1
4          Rural             174.12  24.000000     never smoked       1

    fat_bsugar
0  8370.054000
1  5842.501436
2  3442.400000
3  5890.312000
4  4178.880000
```

1.2.3 Pipeline

[19]:
```python
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
smoke=[]
smoke=le.fit_transform(hd['smoking_status'])
hd['smoking_status']=smoke
```

[20]:
```python
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```
one_hot_features=['ever_married','gender','work_type','Residence_type']
numerical_features=['age','bmi','avg_glucose_level','hypertension','heart_disease','smoking_st

features=numerical_features+one_hot_features

hd_processing_pipeline=ColumnTransformer([
    ('numerical',StandardScaler(), numerical_features),
    ('one_hot',OneHotEncoder(categories='auto'), one_hot_features)
])

X=hd_processing_pipeline.fit_transform(hd[features])
```

[21]:
```
y=hd['stroke'].values
```

1.2.4 train-test split + balance data

[27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[38]:
```
unique, counts = np.unique(y_train, return_counts=True)
print(np.asarray((unique, counts)).T)
```

```
[[   0 3901]
 [   1  187]]
```

[55]:
```
new_X=[]
new_y=[]
tempx=[]
tempy=[]
for i in range(len(y_train)):
    if y_train[i]==1:
        new_X.append(X_train[i])
        new_y.append(y_train[i])
    if y_train[i]==0:
        tempx.append(X_train[i])
        tempy.append(y_train[i])

s=np.stack(random.sample(tempx,200))
s1=[0]*200
new_X=np.concatenate((new_X,s))
new_X=np.stack(new_X)
new_y=np.concatenate((new_y,s1))
new_y=np.stack(new_y)
```

1.3 Part 3. PCA

1.3.1 Interpret feature importance using regression

[40]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_regression,␣
↪f_regression

f_score, _ = f_regression(X,y)
discrete_features = np.zeros(X.shape[1])
discrete_features[0:len(hd_processing_pipeline.transformers[0][2])] = 1 #␣
↪Numerical features are at the start
mi = mutual_info_regression(X, y, discrete_features=discrete_features.
↪astype('bool_'))

print("Feature \t\t F-score \t\t MI")
for i,feature in enumerate(features):
  print(f"{feature} \t\t {f_score[i]} \t\t {mi[i]}")

X_best = SelectKBest(f_regression,k=10).fit_transform(X, y)
```

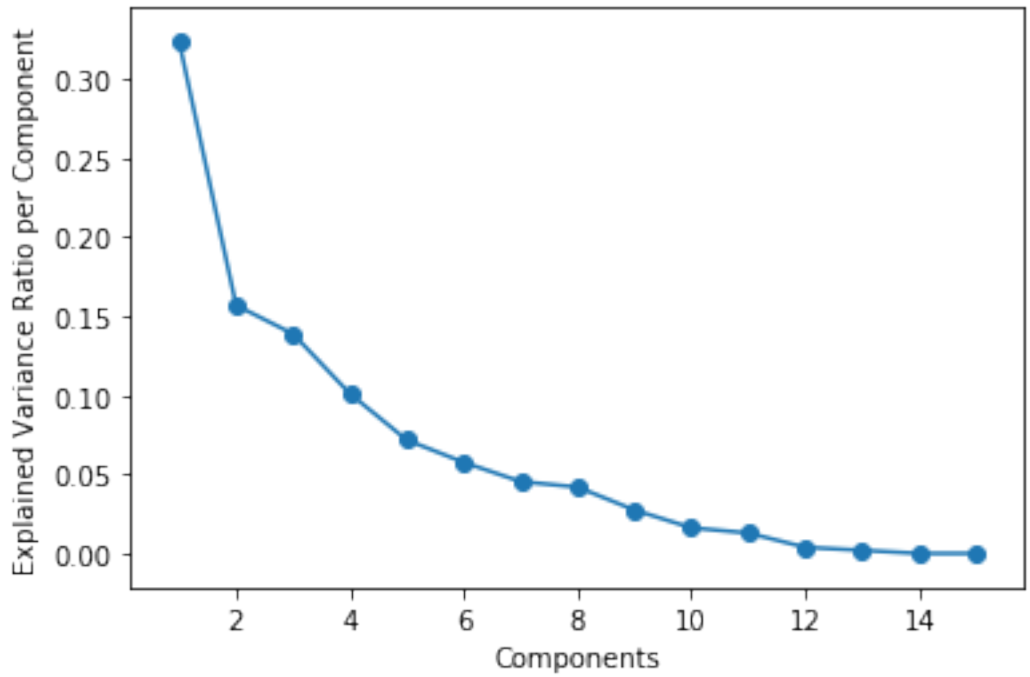
```
Feature                  F-score                 MI
age             326.9165678586869               0.050336115625047920
bmi             7.7597756541479805              0.007613028848614256
avg_glucose_level               90.50386961378669               0.0
hypertension            84.95354215997183               0.0
heart_disease           94.69840601634587               0.010021060016772942
smoking_status          4.043033245970726               0.0027097772057946834
fat_bsugar              81.09226477115658               0.3852996924907266
ever_married            60.66722965592213               0.01832169725456545
gender          60.66722965591385               0.0
work_type               0.4162314391342801              0.015461017511124275
Residence_type          0.4246250130154117              0.0
```

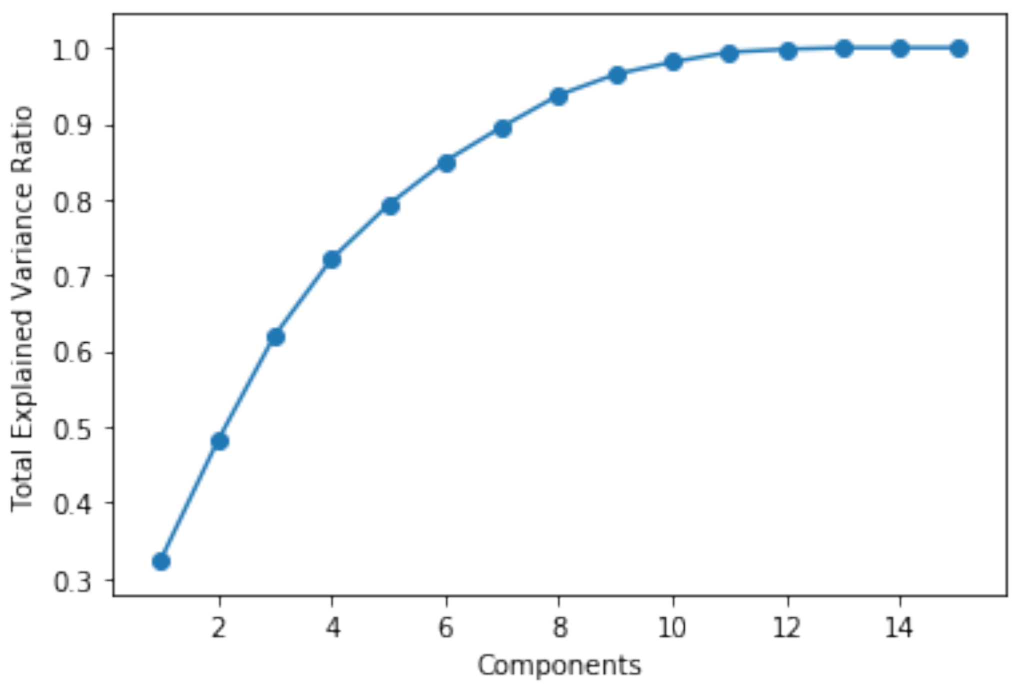
1.3.2 PCA dimensionality reduction

[60]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=15)
pc = pca.fit_transform(new_X)

plt.figure()
plt.plot(np.arange(15)+1,sorted(pca.explained_variance_ratio_,reverse=True))
plt.scatter(np.arange(15)+1,sorted(pca.explained_variance_ratio_,reverse=True),)
plt.xlabel("Components")
plt.ylabel("Explained Variance Ratio per Component")

plt.figure()
plt.plot(np.arange(15)+1,np.cumsum(pca.explained_variance_ratio_))
plt.scatter(np.arange(15)+1,np.cumsum(pca.explained_variance_ratio_))
```

```
plt.xlabel("Components")
plt.ylabel("Total Explained Variance Ratio")
```

[60]: Text(0, 0.5, 'Total Explained Variance Ratio')





[65]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=7)
pc_train = pca.fit_transform(new_X)
pc_test=pca.fit_transform(X_test)
```

[64]:
```
new_X.shape
```

[64]: (387, 19)

[66]:
```
pc_train.shape
```

[66]: (387, 7)

[67]:
```
pc_test.shape
```

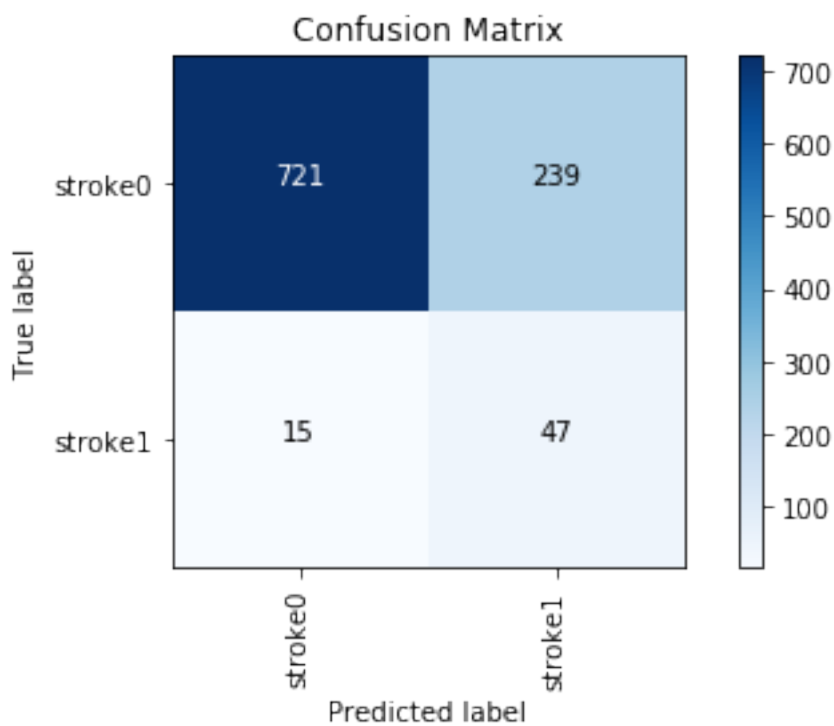
[67]: (1022, 7)

1.4 Part 4. Logistic Regression

[79]:
```
train_data_category = new_y
test_data_category = y_test

log_reg = LogisticRegression(penalty = 'l1', solver='liblinear')
log_reg.fit(new_X, new_y)
predicted = log_reg.predict(X_test)
score = log_reg.predict_proba(X_test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
↪predicted))
draw_confusion_matrix(y_test, predicted, ['stroke0', 'stroke1'])
fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
↪score)

print("Logistic Model Performance Results:\n")

pyplot.figure(1)
```

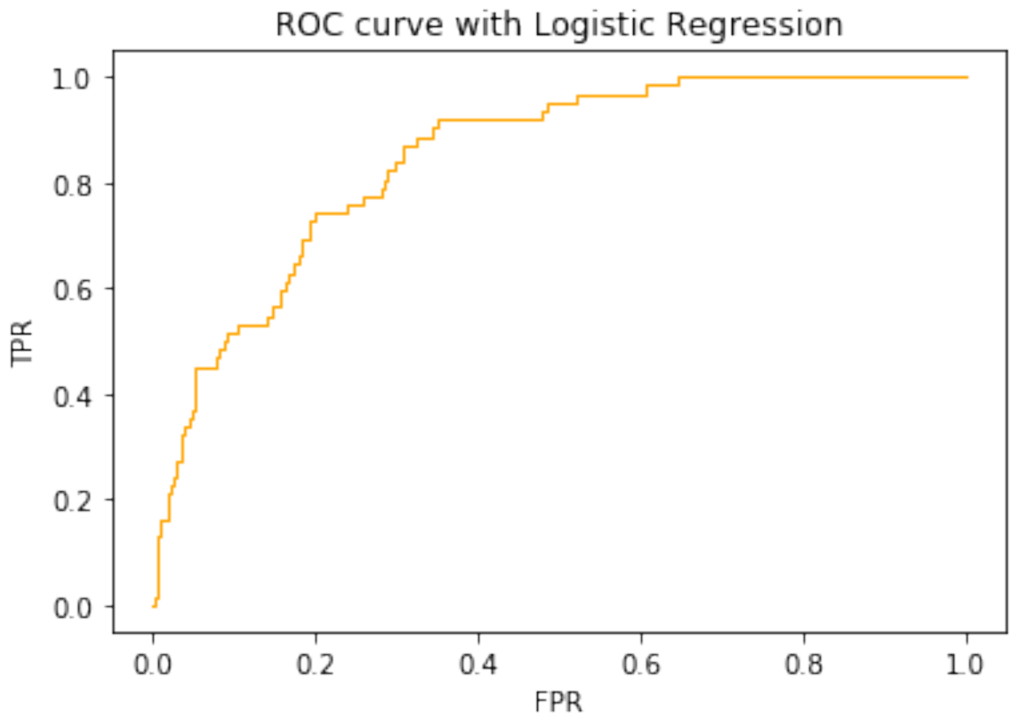
```
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.751468
Precision:   0.164336
Recall:      0.758065
F1 Score:    0.270115
Confusion Matrix:
 [[721 239]
 [ 15  47]]
```




```
Logistic Model Performance Results:
```

[79]: Text(0, 0.5, 'TPR')

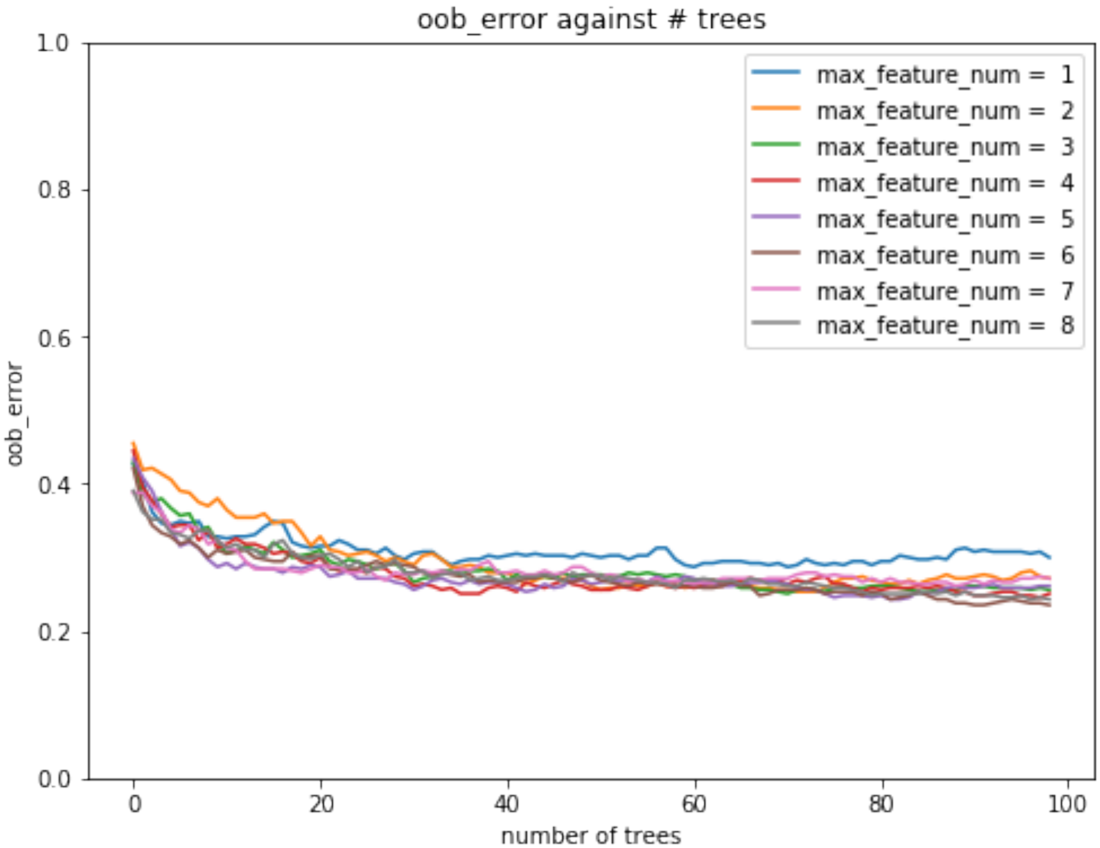


1.5 Part 5. Ensemble (Random Forest)

[82]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

tree_num = np.arange(1, 100) #best tree:15
max_feature_num = np.arange(1,9) #best feature num:7

rmse = [[], [], [], [], [],[],[],[]]
oob_error = [[], [], [], [], [],[],[],[]]


for i in tree_num:
    for j in max_feature_num:
        rfr = RandomForestClassifier(n_estimators = i, max_features = j,␣
↪max_depth = 20, bootstrap = True, oob_score = True, random_state = 42)
        rfr.fit(new_X, new_y)
      #  rfr_pred = cross_val_predict(rfr, X1, X_test, cv = 10)
        oob_error[j-1].append(1-rfr.oob_score_)
print (oob_error)
```

```python
f = plt.figure()
f.set_figwidth(8)
f.set_figheight(6)
for i in range(8):
    plt.plot(oob_error[i], label = 'max_feature_num = % i' % max_feature_num[i])
plt.title('oob_error against # trees')
plt.xlabel('number of trees')
plt.ylabel('oob_error')
plt.ylim([0,1])
plt.legend(loc = 'best')
plt.show()
```

```
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:523: UserWarning: Some inputs do not have
OOB scores. This probably means too few trees were used to compute any reliable
oob estimates.
  warn("Some inputs do not have OOB scores. "
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:528: RuntimeWarning: invalid value
encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:523: UserWarning: Some inputs do not have
OOB scores. This probably means too few trees were used to compute any reliable
oob estimates.
  warn("Some inputs do not have OOB scores. "
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:528: RuntimeWarning: invalid value
encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:523: UserWarning: Some inputs do not have
OOB scores. This probably means too few trees were used to compute any reliable
oob estimates.
  warn("Some inputs do not have OOB scores. "
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:528: RuntimeWarning: invalid value
encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:523: UserWarning: Some inputs do not have
OOB scores. This probably means too few trees were used to compute any reliable
oob estimates.
  warn("Some inputs do not have OOB scores. "
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/_forest.py:528: RuntimeWarning: invalid value
```

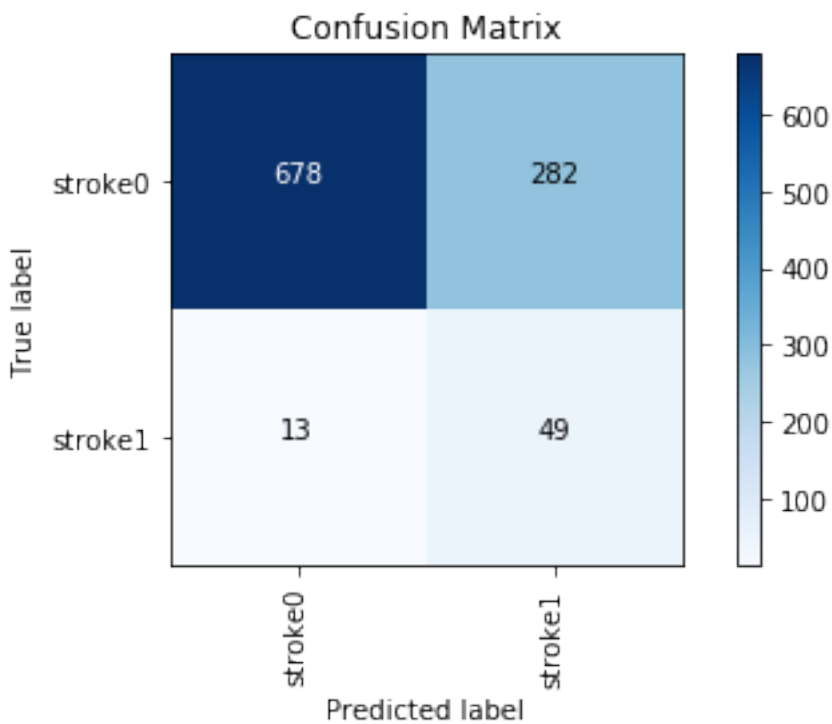


[85]:
```
max_depth_num=np.arange(1,21)

rmse = np.zeros(20)
oob_error = np.zeros(20)

for i in max_depth_num:
    rf_d = RandomForestClassifier(n_estimators = 15, max_features = 7,␣
↪max_depth = i, bootstrap = True, oob_score = True, random_state = 42)
    rf_d.fit(new_X, new_y)
    oob_error[i-1]=1-rf_d.oob_score_

f = plt.figure()
f.set_figwidth(5)
f.set_figheight(4)
plt.plot(max_depth_num,oob_error)
plt.title('oob_error against max depth')
plt.xlabel('depth')
plt.ylabel('oob_error')
plt.show()
```

[106]:
```
rf=RandomForestClassifier(n_estimators = 10, max_features = 5, max_depth = 3)

train_data_category = new_y
test_data_category = y_test

rf.fit(new_X, new_y)
predicted = rf.predict(X_test)
score = rf.predict_proba(X_test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(y_test, predicted, ['stroke0', 'stroke1'])
fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
 ↪score)
```

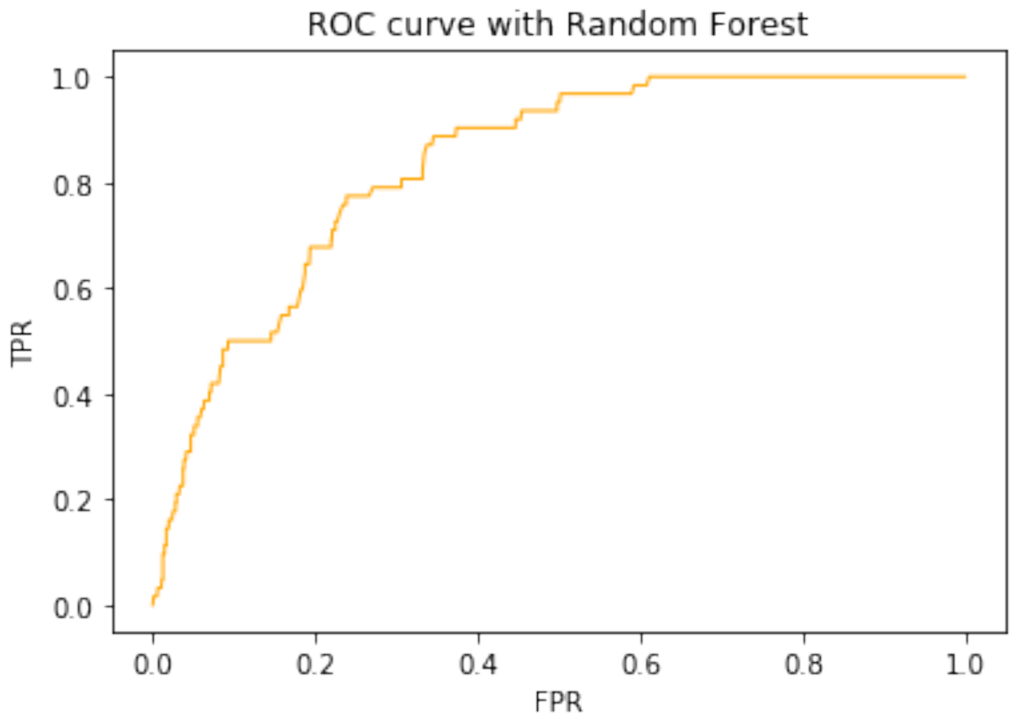
```
print("Random Forest Model Performance Results:\n")

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Random Forest")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.711350
Precision:   0.148036
Recall:      0.790323
F1 Score:    0.249364
Confusion Matrix:
 [[678 282]
 [ 13  49]]
```




```
Random Forest Model Performance Results:
```

[106]: Text(0, 0.5, 'TPR')



1.6 Part 6. Neural Net Classifier

[107]:
```
from sklearn.neural_network import MLPClassifier
parameters = {'solver': ['lbfgs'], 'alpha': 10.0 ** -np.arange(1, 7),␣
↪'hidden_layer_sizes':np.arange(1, 10)}
clf_grid = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1)
```

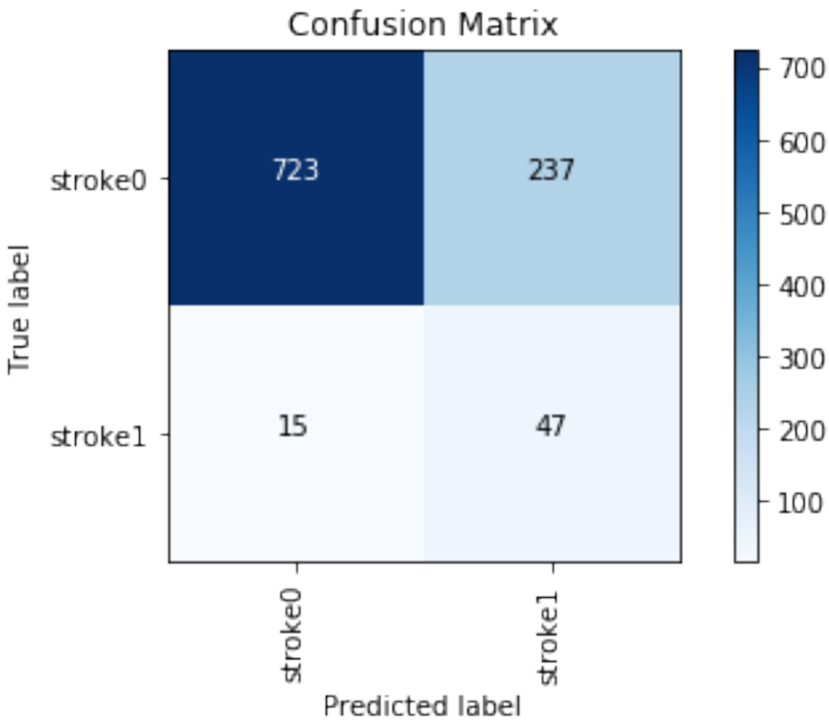
[108]:
```
clf_grid.fit(new_X,new_y)

print("Best score: %0.4f" % clf_grid.best_score_)
print("Using the following parameters:")
print(clf_grid.best_params_)
```

```
Best score: 0.7802
Using the following parameters:
{'alpha': 1e-06, 'hidden_layer_sizes': 1, 'solver': 'lbfgs'}
```

[111]:
```
nn=MLPClassifier(alpha= 1e-06, hidden_layer_sizes= 1, solver='lbfgs')

train_data_category = new_y
test_data_category = y_test

nn.fit(new_X, new_y)
```

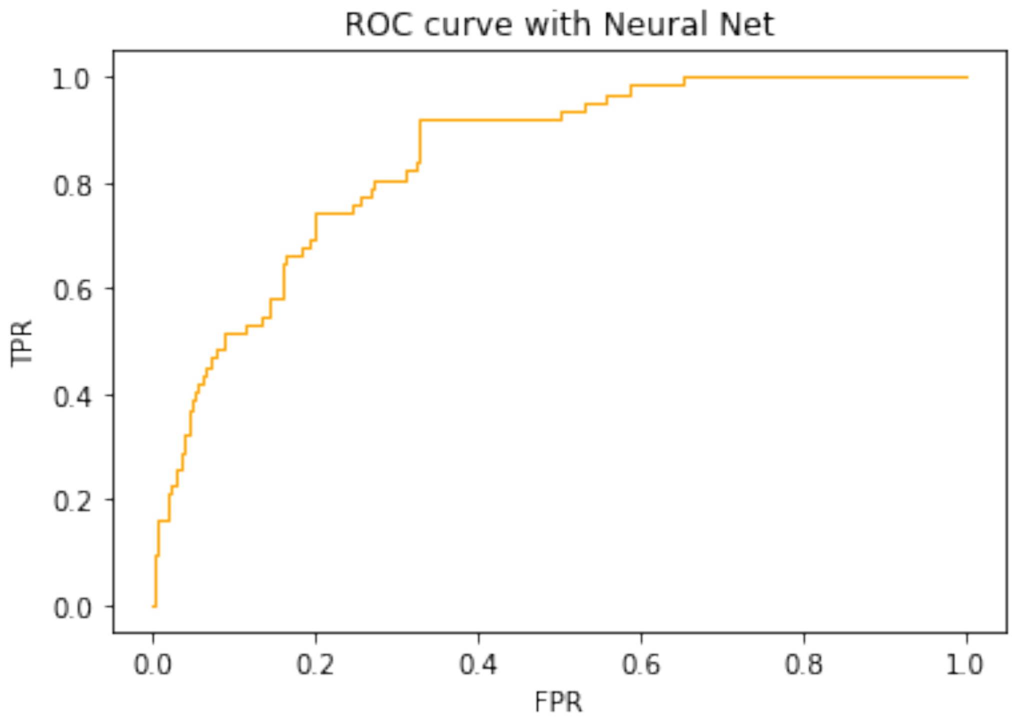
```
predicted = nn.predict(X_test)
score = nn.predict_proba(X_test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(y_test, predicted, ['stroke0', 'stroke1'])
fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
 ↪score)

print("Neural Net Model Performance Results:\n")

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Neural Net")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.753425
Precision:   0.165493
Recall:      0.758065
F1 Score:    0.271676
Confusion Matrix:
 [[723 237]
 [ 15  47]]
```




```
Neural Net Model Performance Results:
```

```
[111]: Text(0, 0.5, 'TPR')
```



1.7 Part 7. Cross Validation

[120]:
```
from sklearn.model_selection import KFold
from sklearn import model_selection

kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)

model_kfold = RandomForestClassifier(n_estimators = 10, max_features = 5,␣
↪max_depth = 3)

results_kfold = model_selection.cross_val_score(model_kfold, new_X, new_y,␣
↪cv=kfold)

print("For an Random Forest our mean accuracy across folds is: %.2f%%" %␣
↪(results_kfold.mean()*100.0))
```

```
For an Random Forest our mean accuracy across folds is: 75.96%
```

[122]:
```
kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)

model_kfold = MLPClassifier(alpha= 1e-06, hidden_layer_sizes= 1, solver='lbfgs')
```

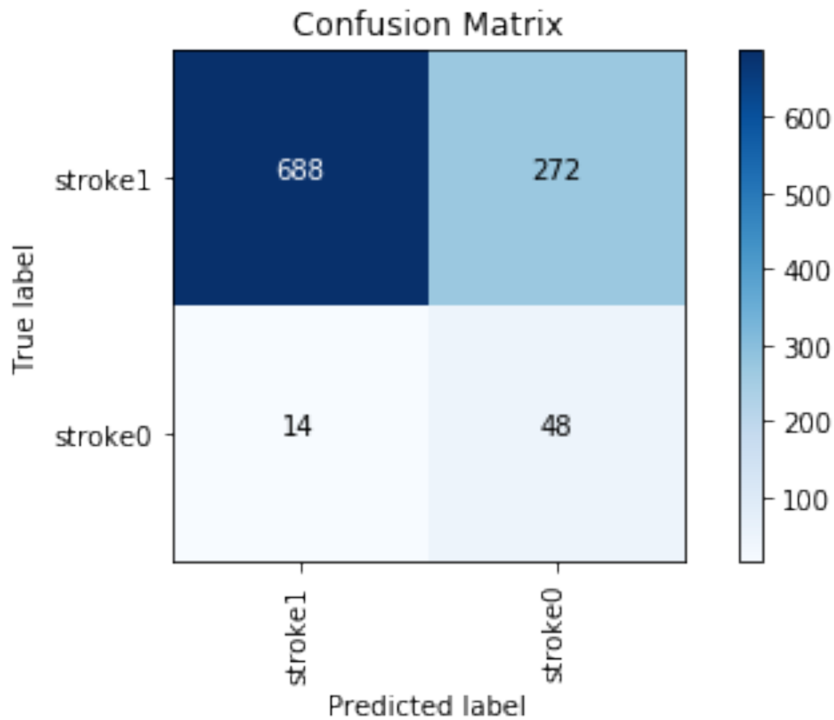
```
results_kfold = model_selection.cross_val_score(model_kfold, new_X, new_y,␣
↪cv=kfold)

print("For NN our mean accuracy across folds is: %.2f%%" % (results_kfold.
↪mean()*100.0))
```

```
For NN our mean accuracy across folds is: 75.20%
```

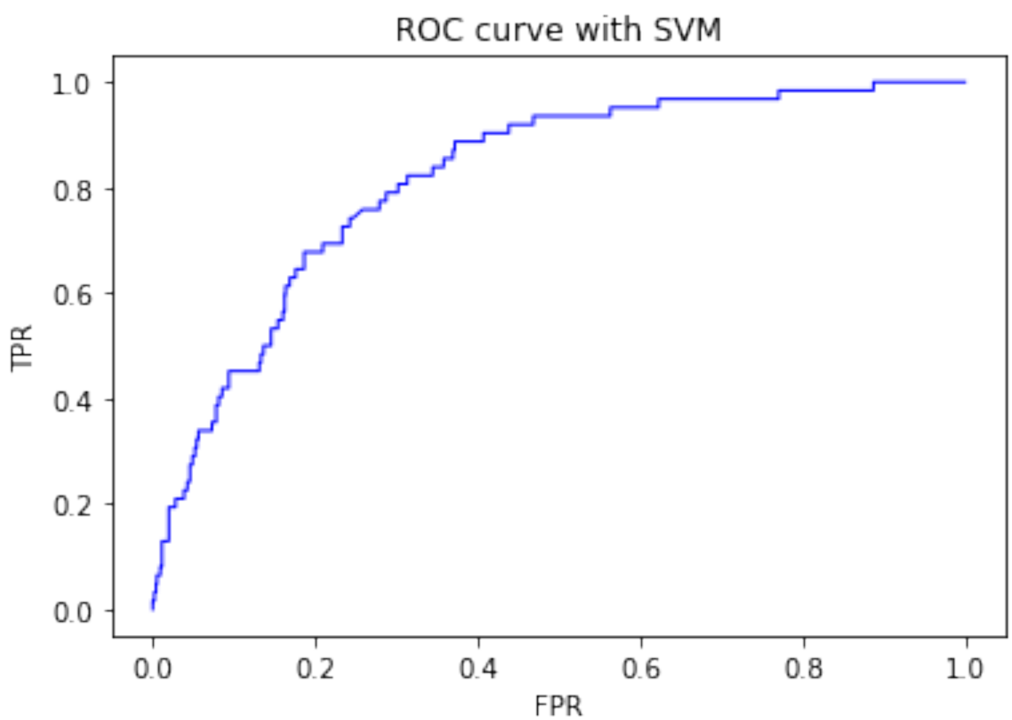
1.8 Part 8. Custom Model (SVM)

[116]:
```
svm = SVC(probability=True)
svm.fit(new_X, new_y)
testing_result = svm.predict(X_test)
predicted = svm.predict(X_test)
score = svm.predict_proba(X_test)

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
↪predicted))
draw_confusion_matrix(y_test, predicted, ['stroke1', 'stroke0'])

print("SVM Model Performance Results:\n")

fpr_svm, tpr_svm, thresholds = metrics.roc_curve(y_test, score[:, 1],␣
↪pos_label=1)

pyplot.figure(1)
pyplot.plot(fpr_svm, tpr_svm, color='blue', lw=1)
pyplot.title("ROC curve with SVM")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
pyplot.show()
```

```
Accuracy:    0.720157
Precision:   0.150000
Recall:      0.774194
F1 Score:    0.251309
Confusion Matrix:
 [[688 272]
 [ 14  48]]
```



SVM Model Performance Results:



1.9 Part 9. Attachment

1.9.1 Attachment of balancing data before splitting (no need to look at it if the above code is sufficient)

[126]:
```
#balance dataset
#data=hd.loc[hd['stroke'] == 0]

s0 = hd.stroke[hd.stroke.eq(0)].sample(250).index
s1 = hd.stroke[hd.stroke.eq(1)].sample(249).index

df = hd.loc[s0.union(s1)]
```

[127]:
```
one_hot_features=['ever_married','gender','work_type','Residence_type']
numerical_features=['age','bmi','avg_glucose_level','hypertension','heart_disease','smoking_st

features=numerical_features+one_hot_features

hd_processing_pipeline=ColumnTransformer([
    ('numerical',StandardScaler(), numerical_features),
    ('one_hot',OneHotEncoder(categories='auto'), one_hot_features)
])

X1=hd_processing_pipeline.fit_transform(df[features])
y1=df['stroke'].values
```

[128]:
```
pca = PCA(n_components=7)
pc = pca.fit_transform(X1)
```

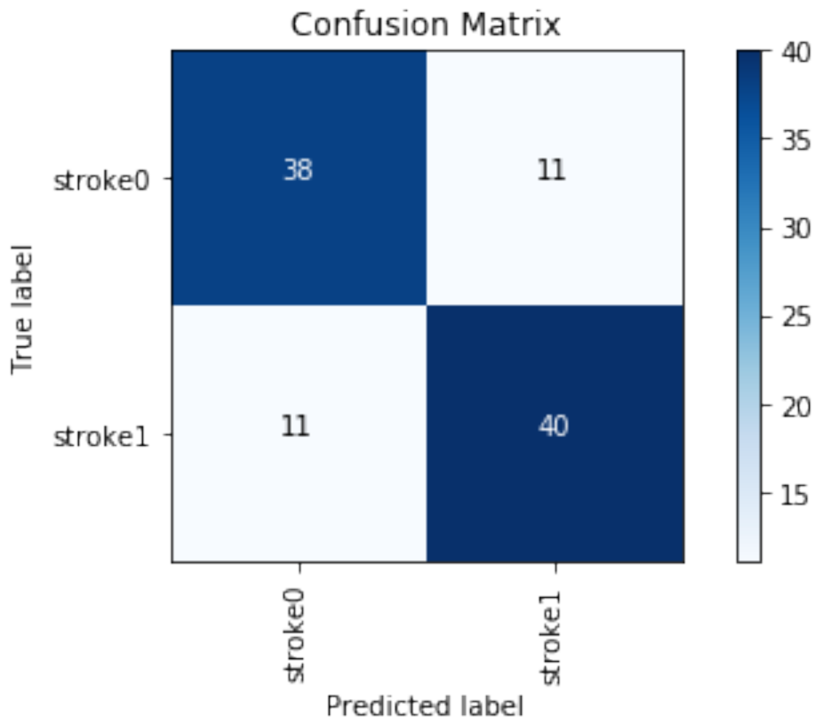
[129]:
```
train, test, target, target_test = train_test_split(X1, y1, test_size=0.2,␣
 ↪random_state=42)
pca_train, pca_test, pca_target, pca_target_test = train_test_split(pc, y1,␣
 ↪test_size=0.2, random_state=42)
```

1.9.2 Logistic Regression

[130]:
```
pca_train_data_category = pca_target
pca_test_data_category = pca_target_test

log_reg = LogisticRegression(penalty = 'l1', solver='liblinear')
log_reg.fit(pca_train, pca_target)
pca_predicted = log_reg.predict(pca_test)
pca_score = log_reg.predict_proba(pca_test)[:,1]
```

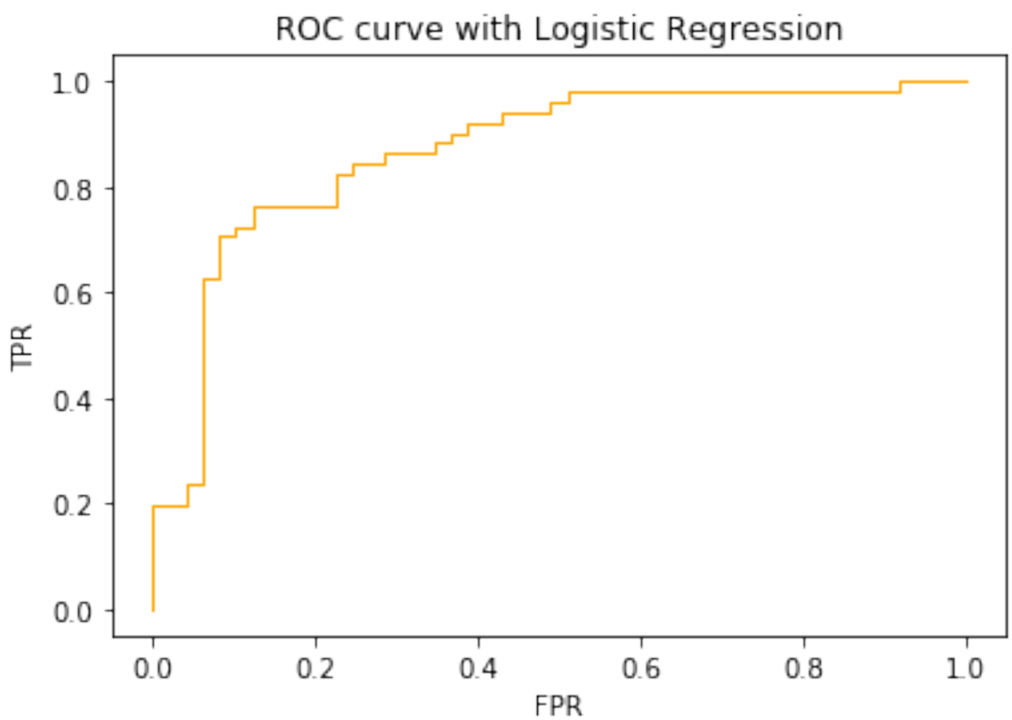
```
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(pca_test_data_category,␣
↪pca_predicted)))
print("%-12s %f" % ('Precision:', metrics.
↪precision_score(pca_test_data_category, pca_predicted, labels=None,␣
↪pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(pca_test_data_category,␣
↪pca_predicted, labels=None, pos_label=1, average='binary',␣
↪sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(pca_test_data_category,␣
↪pca_predicted, labels=None, pos_label=1, average='binary',␣
↪sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(pca_test_data_category,␣
↪pca_predicted))
draw_confusion_matrix(pca_target_test, pca_predicted, ['stroke0', 'stroke1'])
fpr_log_reg, tpr_log_reg, thresholds = metrics.
↪roc_curve(pca_test_data_category, pca_score)

print("Logistic Model Performance Results:\n")

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.780000
Precision:   0.784314
Recall:      0.784314
F1 Score:    0.784314
Confusion Matrix:
 [[38 11]
 [11 40]]
```




```
Logistic Model Performance Results:
```

```
[130]: Text(0, 0.5, 'TPR')
```

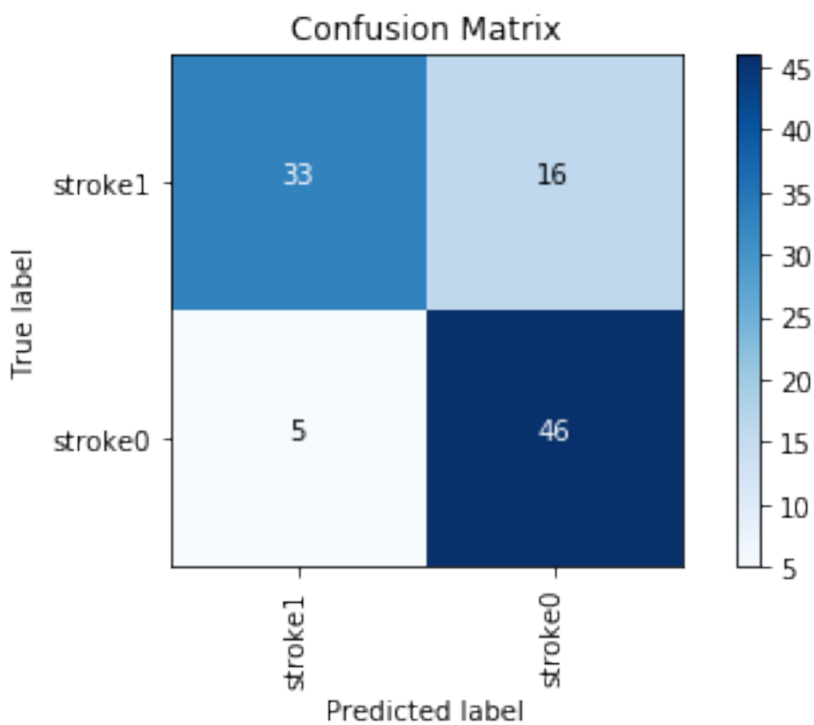


1.9.3 Random Forest

[131]:
```
rf=RandomForestClassifier(n_estimators = 15, max_features = 7, max_depth = 3)

train_data_category = target
test_data_category = target_test

rf.fit(train, target)
predicted = rf.predict(test)
score = rf.predict_proba(test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(target_test, predicted, ['stroke1', 'stroke0'])
```

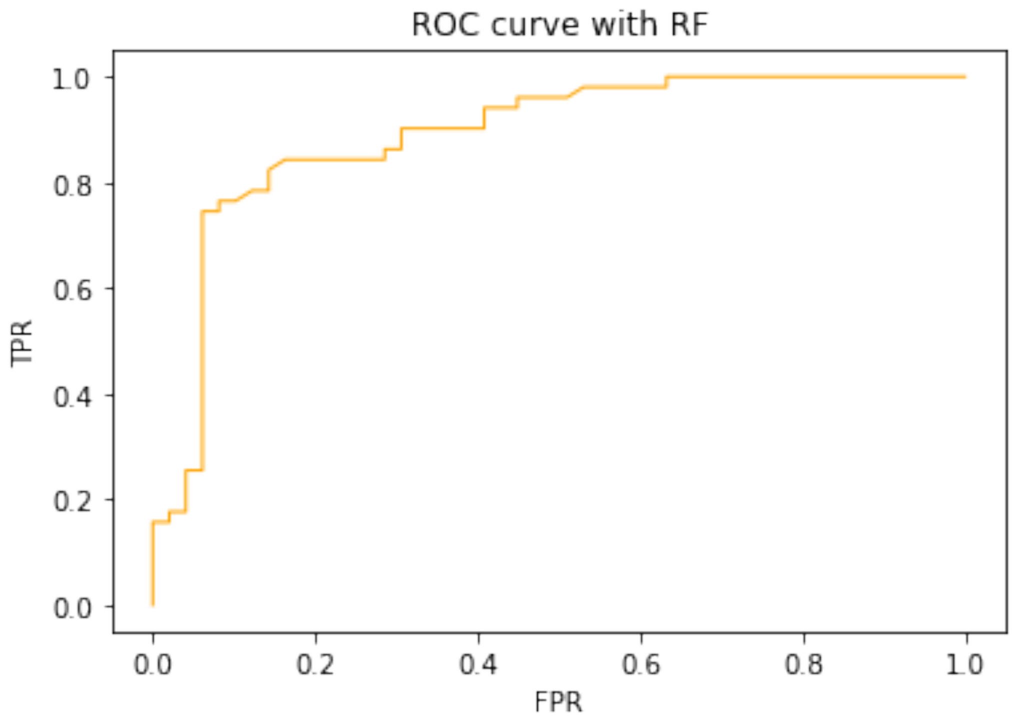
```
fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
↪score)

print("RF Model Performance Results:\n")

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with RF")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.790000
Precision:   0.741935
Recall:      0.901961
F1 Score:    0.814159
Confusion Matrix:
 [[33 16]
 [ 5 46]]
```




```
RF Model Performance Results:
```

[131]: `Text(0, 0.5, 'TPR')`

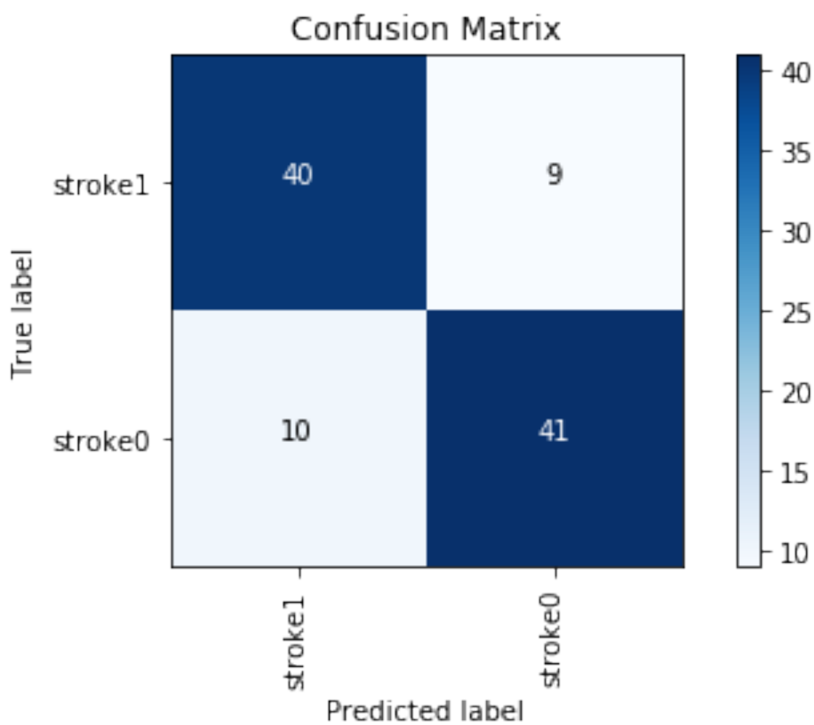


1.9.4 NN

[132]:
```
nn=MLPClassifier(alpha= 0.0001, hidden_layer_sizes= 1, solver='lbfgs')

train_data_category = target
test_data_category = target_test

nn.fit(train, target)
predicted = nn.predict(test)
score = nn.predict_proba(test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(target_test, predicted, ['stroke1', 'stroke0'])
```

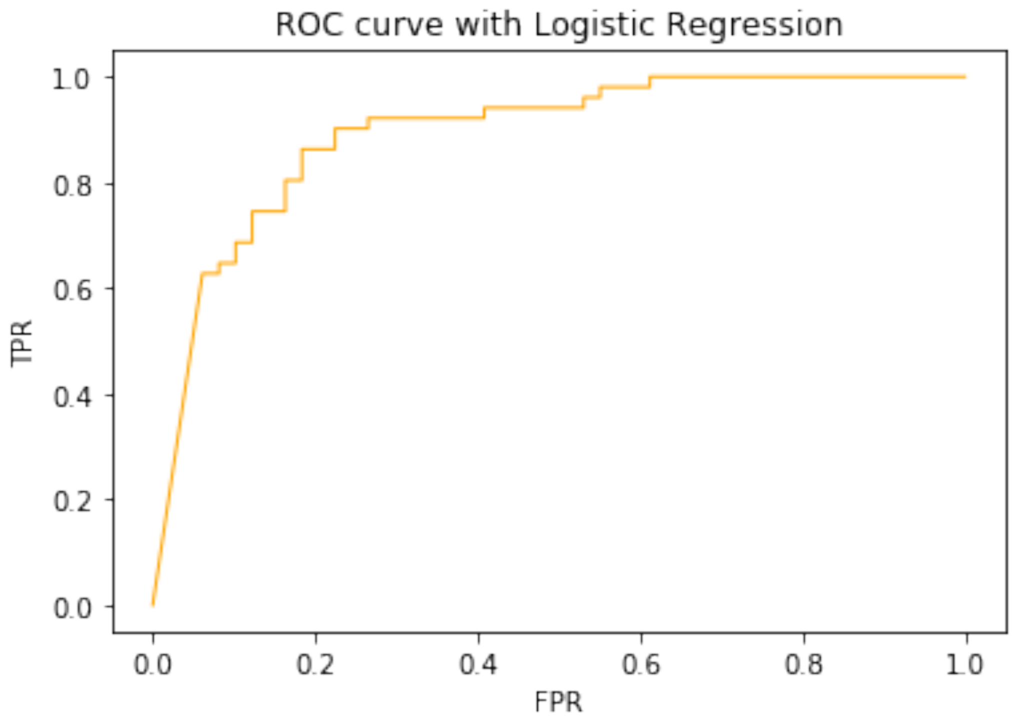
```
fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
 ↪score)

print("Logistic Model Performance Results:\n")

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.810000
Precision:   0.820000
Recall:      0.803922
F1 Score:    0.811881
Confusion Matrix:
 [[40  9]
 [10 41]]
```




```
Logistic Model Performance Results:
```

[132]: Text(0, 0.5, 'TPR')



1.9.5 Cross Validation

[136]:
```
kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)
model_kfold = rf=RandomForestClassifier(n_estimators = 15, max_features = 7,␣
↪max_depth = 3)
results_kfold = model_selection.cross_val_score(model_kfold, train, target,␣
↪cv=kfold)
print("For an Random Forest our mean accuracy across folds is: %.2f%%" %␣
↪(results_kfold.mean()*100.0))

kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle=True)
model_kfold = MLPClassifier(alpha= 0.0001, hidden_layer_sizes= 1,␣
↪solver='lbfgs')
results_kfold = model_selection.cross_val_score(model_kfold, train, target,␣
↪cv=kfold)
print("For an NN our mean accuracy across folds is: %.2f%%" % (results_kfold.
↪mean()*100.0))
```

```
For an Random Forest our mean accuracy across folds is: 70.66%
For an NN our mean accuracy across folds is: 73.68%
```

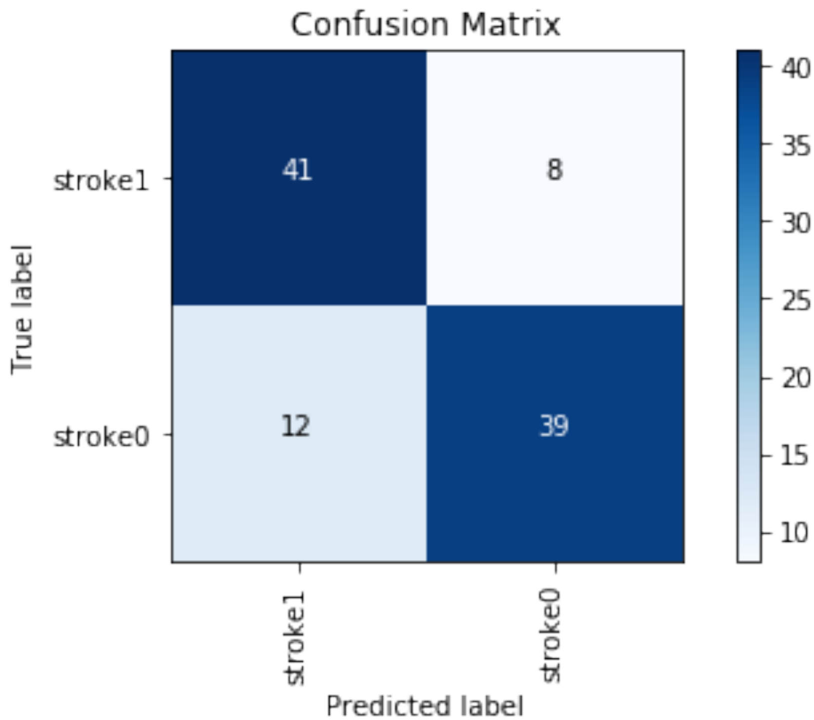
```
/Users/zhiyuanchen/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:470:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
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
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

1.9.6 SVM

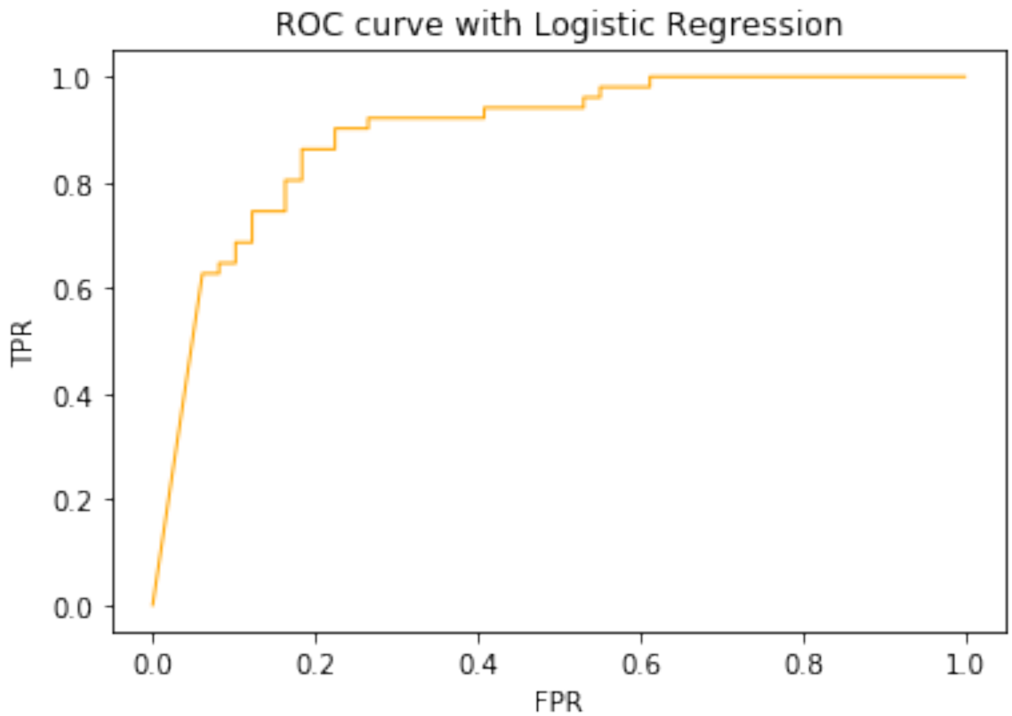
[138]:
```python
from sklearn.svm import LinearSVC
svm = LinearSVC(C=0.0001,random_state=42,max_iter=10000)

train_data_category = target
test_data_category = target_test

svm.fit(train, target)
predicted = svm.predict(test)
#score = svm.predict_proba(test)[:,1]

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(test_data_category,␣
 ↪predicted)))
print("%-12s %f" % ('Precision:', metrics.precision_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('Recall:', metrics.recall_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("%-12s %f" % ('F1 Score:', metrics.f1_score(test_data_category,␣
 ↪predicted, labels=None, pos_label=1, average='binary', sample_weight=None)))
print("Confusion Matrix: \n", metrics.confusion_matrix(test_data_category,␣
 ↪predicted))
draw_confusion_matrix(target_test, predicted, ['stroke1', 'stroke0'])
#fpr_log_reg, tpr_log_reg, thresholds = metrics.roc_curve(test_data_category,␣
 ↪score)

print("Logistic Model Performance Results:\n")

pyplot.figure(1)
pyplot.plot(fpr_log_reg, tpr_log_reg, color='orange', lw=1)
pyplot.title("ROC curve with Logistic Regression")
pyplot.xlabel('FPR')
pyplot.ylabel('TPR')
```

```
Accuracy:    0.800000
Precision:   0.829787
Recall:      0.764706
F1 Score:    0.795918
Confusion Matrix:
 [[41  8]
 [12 39]]
```



Logistic Model Performance Results:

```
[138]: Text(0, 0.5, 'TPR')
```


ROC curve with Logistic Regression
1.0
0.8
0.6
0.4
0.2
0.0
TPR
0.0
0.2
0.4
0.6
0.8
1.0
FPR

```
[ ]:
```

Executive Summary

The report summarizes the work done to analyze the stroke data set. It demonstrates the findings provided by data science techniques. Machine learning pipeline is implemented to predict whether a patient has stroke or not, based on the given set of features, including general information such as type of work, as well as medical indicators like average glucose level. Other than the technical part, domain knowledge is also researched to better interpret the results.

Main parts of the project and the details are described to give an overall view:

- Background: in this section, different features are analyzed on an industry level, to evaluate the possible correlations with stroke. The variable values are looked into and domain challenge are found based on the nature of the given data. In this part, the domain knowledge requirement of data science is demonstrated.
- Methodology: this section explains the whole data pipeline process, which includes, exploring basic statistics, data preparation before model application, finding best parameters of machine learning models, implementing different models on the given data, with performance evaluation using metrics and other techniques. Best model is found and results are shown.
- Results: this section would present the performance of different models in clean and professional visualization. The models are evaluated with different metrics and scores are shown in tables.
- Discussion: based on the found results, with the feature analysis, the domain knowledge can be better interpreted. In this section, the analytical results are evaluated, as in why are the models performing in the certain manners, what can be told from the parameter values, as well as why and how they are acting this way. The ways to better prepare for the data, and recommendations on what can be done for future analytical work are given for the UCLA hospital.
- Conclusion: a summary of the overall project

Key findings during the project and the main challenge encountered and solved are:

- Data set: the feature given are good indicators of stroke, but they are very general and lack of depth. This can post challenge to make accurate predictions. The models can't achieve high result based on the nature of limited information.
- Data science techniques: several challenges are encountered as the data is being prepared. 1) the given data is imbalanced, with the patient not having stroke significantly higher than the patient with stroke. This can make the data skewed towards a non stroke prediction, to solve this, the data can be manually balanced. However, models would have different performance based on when to balance the data, after splitting or before, and this is addressed in the report. 2) It is observed that the model would have a good recall performance, however the precision is low in general, which indicates a high false positive rate. Patient predicted to have stroke can be double checked. The reason of this type of model performance results from the way the data is balanced. As we arbitrarily balance the training data, which isn't a good representation of the real population, the model would be biased. 3) It is also observed that the hyperparameters of our model is of low values, which means that the given data isn't complex and doesn't have much to dig into. This would be reflected in the discussion as for suggestion given to the UCLA hospital.

Background

The provided features and their possible reasoning of correlation to the stroke likelihood are:

Feature	Meaning	Link
age	Age of patient	Stroke is more likely to happen with higher age
bmi	Body Mass Index	Physical health and lifestyle
avg glucose level	Blood Sugar Level	Physical health and eating habits
hypertension	Has or Not	Blood pressure, highly related to the cause of stroke
heart disease	Has or Not	Highly related to the cause of stroke
smoking status	Smoking history	Health condition
ever married	Yes or No	Lifestyle, life history, happiness and stress
gender	Male, Female, or Other	Contribution of biological difference
work type	Different type of work	Working hour, nature, indicates stress level
residence type	Rural or Urban	Indicates different lifestyle and stress level

Table 1: Feature Analysis

From the interpretation of the given features, both general information and more correlated features all can be good indicators of determining stroke. However, there are limitations with the data given. For example, the feature work type, is only of different values like government job, child, self employed, never worked... while in real life, there are more precise and comprehensive ways of categorization. The provided information on hypertention and heart disease is also shallow, if yes, how severe? When it started? And other more detailed information could be provided.

The nature of the data set also post domain challenges. For example, the values we are given are too general, the amount of data is not enough to make in depth analysis. We would process the data and apply different machine learning models to find out more.

Methodology

Basic Statistics

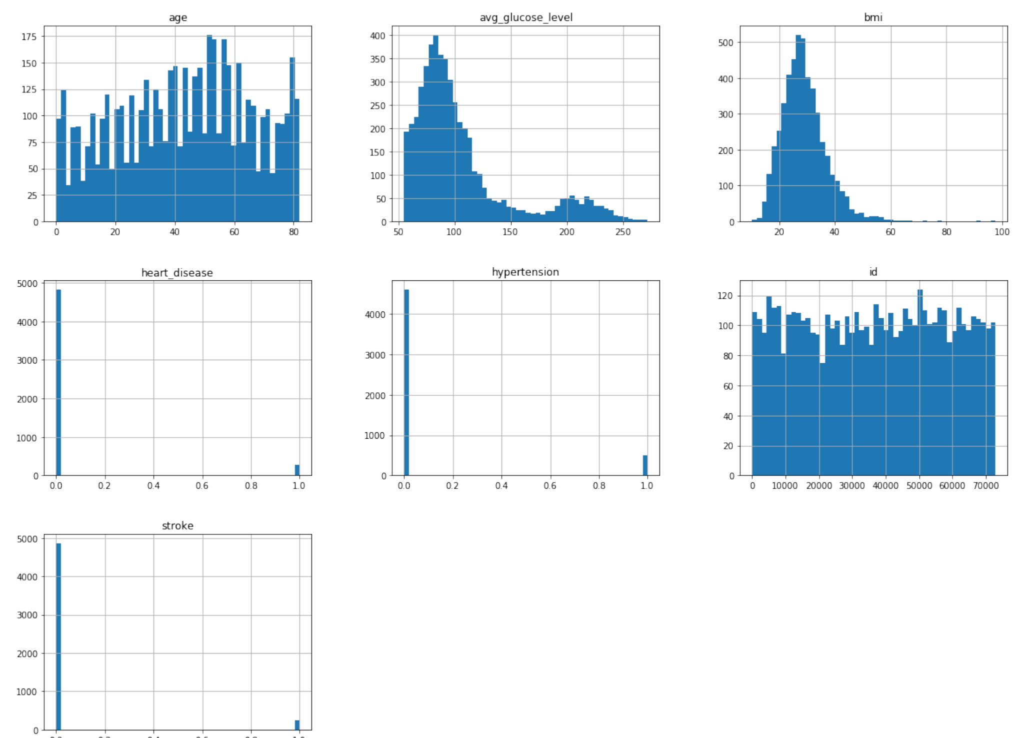


Figure 1: Histogram of Data

The dataset given consists of several features and the label corresponding to whether one has stroke or not. It has categorical features like: gender, ever married, work type, residence type, smoking status. One hot features such as hypertension and heart disease, and numerical features like bmi, average glucose level, age. There are 201 data of null values in bmi.
The histogram provides a good sense on the distribution of some feature values:

From here, it can be seen that the target variable stroke, is unbalanced, which would be processed in the later steps.
The correlation matrix offers insight into the correlations between features and the label:

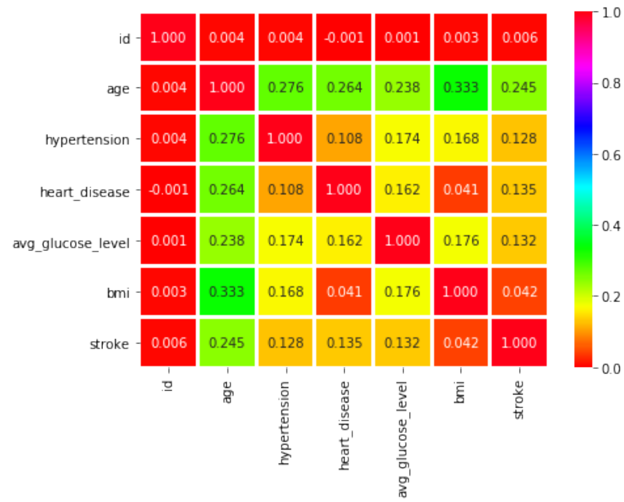


Figure 2: Correlation Matrix

From the graph, it can be seen that the feature id is not useful for our dataset, which satisfies the common sense. It can also be confirmed from the histogram above, where the distribution of this feature indicates there's no difference among all data.
Age, hypertension, heart disease, glucose level all are good indicators to determine stroke. The feature age also shares colinearities among other features.

Data Preparation

From the statistics earlier, the feature id would be dropped.
The imputation strategy for the lost bmi value is to fill with the mean bmi value. This is the best way than other methods such as dropping rows and deleting feature, as there's only a small portion of the bmi data loss, by filling the blanks we can retain other useful information. The average bmi value is safe and comparatively accurate. The data is normally distributed, which can be inferred from histogram as well as a domain knowledge of the general population. Thus imputing it with the mean value would be a useful solution.
The augmentation method is to feature cross over bmi and average glucose level. It is inferred from professional knowledge, as bmi means the body fatness and the glucose level relfects the blood sugar level. These two features are closely related, and thus can be augmented to enrich the data set information.

The data strategy of scaling and pipelining are as follows:

- smoking status: this feature includes formerly smoked, never smoked, smokes and unknown. The values have certain hierarchy in between, and thus label encoding is chosen.
- Other categorical features, ever married, work type, residence type, are all variables of low dimensionality. Thus one hot encoding would best represent them. OHE shows the independence between the values.
- A standard scaler is fed into the numerical values, to make them of unit mean and variance and thus prevent data skewing. One hot encoder is pipelined for the categorical features mentioned above.

Two different methods, balancing dataset before or after splitting into training and testing set, are performed for this project. Regarding to different order of split and balance, the model performance would be different. It is said that splitting before balancing would produce a more reliable result. The report would discuss model performance under both ways.

PCA

Principle Component Analysis is implemented to solve the high dimensionality of the dataframe. PCA is used to reduce the complexity of the dataframe. It would determined the most significant componenets by demonstrating how one component relates to the explanation of the variance. We can see this function from the graph:

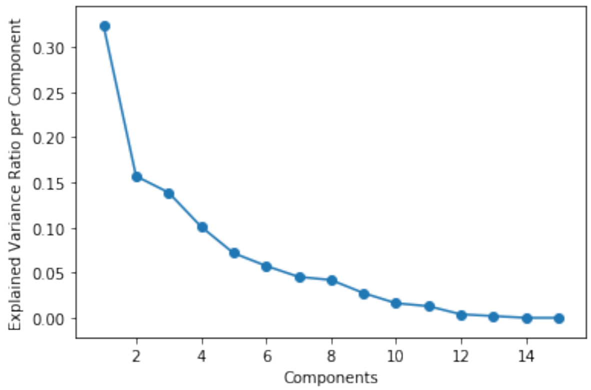


Figure 3: Explained Variance

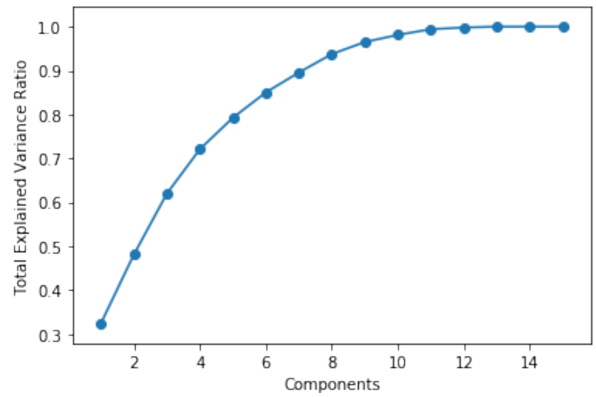


Figure 4: Total Explained Variance

It can be seen that the first five components can explain a large portion of the variance in the data. And as the number of components increased, total explained variance ratio tends to reach one hundred percent. After testing around different values, number of components of seven is chosen for the PCA, which can have a good grasp of explaining the variance.

Logistic Regression

Before applying the logistic regression model, we would first interpret feature importance using mutual information regression. The corresponding score is as follows:

Feature	F-score	MI
age	326.92	0.050
bmi	7.76	0.008
avg glucose level	90.50	0.0
hypertension	84.95	0.0
heart disease	94.70	0.010
smoking status	4.04	0.003
fat bsugar	81.09	0.038
ever married	60.67	0.018
gender	60.67	0.0
work type	0.42	0.015
residence type	0.42	0.0

Table 2: Feature scores for the dataset

We then apply the L1 regularization on the model. We received the resulting confusion matrix and ROC curve: (the resulting performance is included in the Result Section in the report)

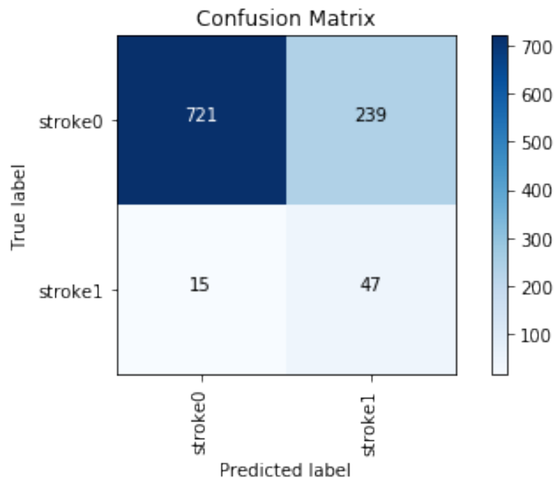


Figure 5: Confusion Matrix

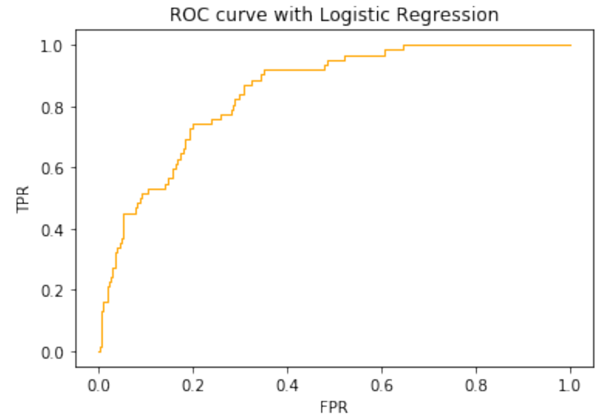


Figure 6: ROC Curve

Random Forest

To find the best parameters for the Random Forest Model, we would first sweep the values for the hyperparameter number of estimators and maximum number of features, which means the number of trees and features. The resulting OOB error across different parameter values are as follow:

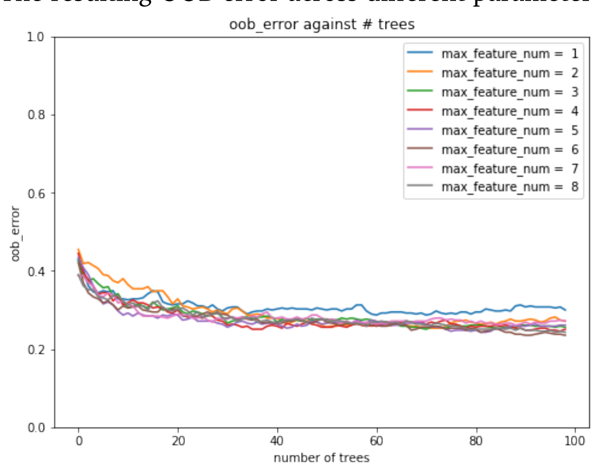


Figure 7: Find Estimator Num and Max Feature

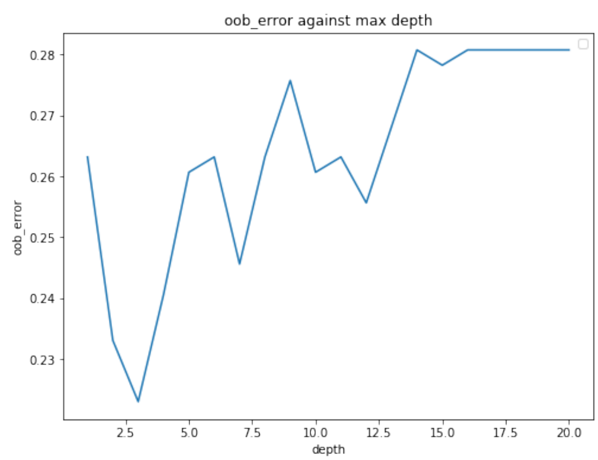


Figure 8: Find Max Depth

It can be seen that number of estimator of 15, with 7 max features achieve a lowest error. Based on this result, we would then find the value for another paramter maximum depth, which is the depth that the tree would dive into. From the resulting OOB error graph, we can see that a max depth of 3 is sufficient.
The resulting model performance are:

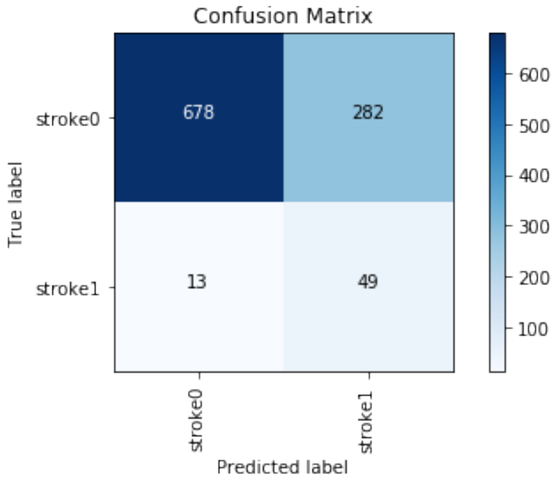


Figure 9: Confusion Matrix

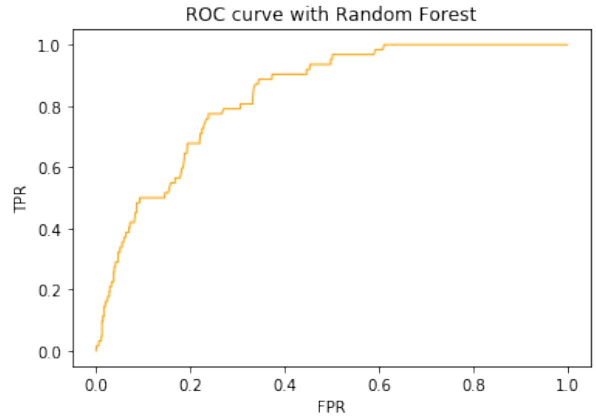


Figure 10: ROC Curve

Neural Net

Run a grid search on the parameter alpha and hidden layer size to find the best parameter for the neural net classifier. An alpha of 1e-6 and hidden layer size 1 has the best performance.

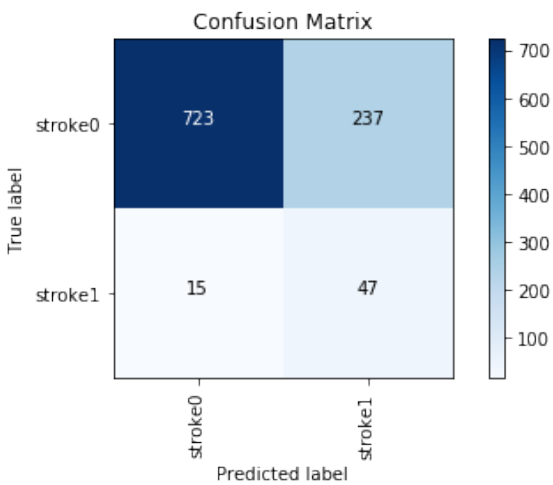


Figure 11: Confusion Matrix

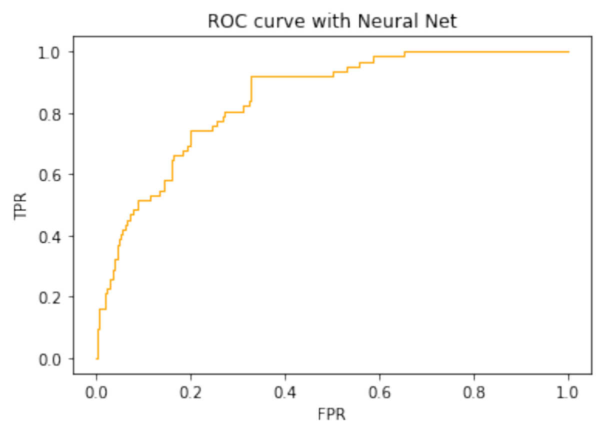


Figure 12: ROC Curve

Cross Validation

For an Random Forest our mean accuracy across folds is: 75.96%
For NN our mean accuracy across folds is: 75.20%

Custom Model

Applying SVM for the data set, we have the performance as below:

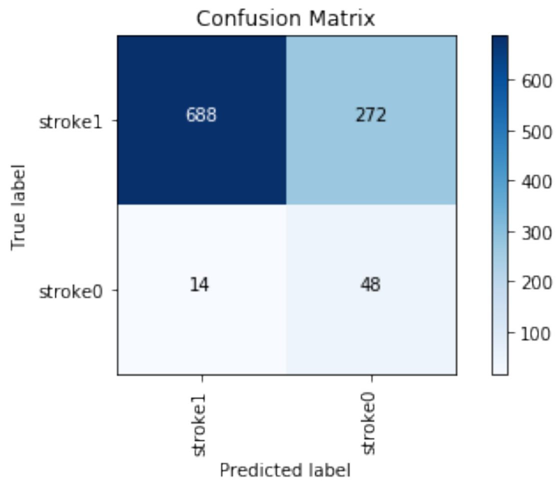


Figure 13: Confusion Matrix

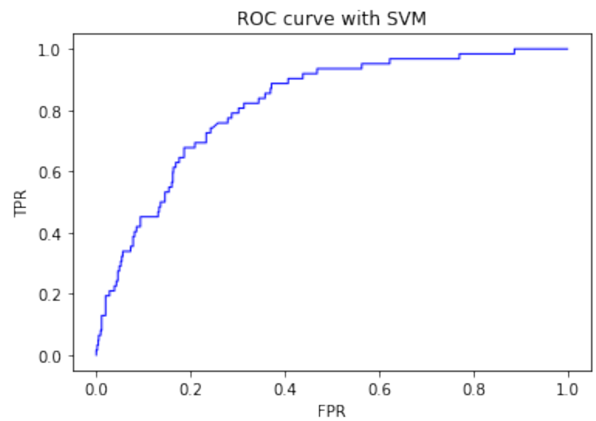


Figure 14: ROC Curve

Results

The results from the different models from the above section are:

	Logistic Regression	Random Forest	Neural Net	SVM
Accuracy	0.751468	0.711350	0.753425	0.720157
Precision	0.164336	0.148036	0.165493	0.150000
Recall	0.758065	0.790323	0.758065	0.774194
F1 Score	0.270115	0.249364	0.271676	0.251309
Parameters	L1, 'liblinear'	n estimators=10 max depth=3 max features=5	alpha=1e-6 hidden layer=1	

Table 3: Performance of Different Models

The best performing model is the logistic regression model, with an f1 score of 0.270115.
It's mentioned in the former section that two different methods of data preparation was implemented. If the data is balanced before splitting, the overall score would be higher. This method is less reliable as it doesn't provide a good insight to the real world data, but it would still be attached here for reference. The corresponding result from the differently preprocessed data with the same methodology is: The random forest

	Logistic Regression	Random Forest	Neural Net	SVM
Accuracy	0.810000	0.830000	0.790000	0.770000
Precision	0.807692	0.793103	0.758621	0.780000
Recall	0.823529	0.901961	0.862745	0.764706
F1 Score	0.815534	0.844037	0.807449	0.772277
Parameters	L1, 'liblinear'	n estimators=15 max depth=3 max features=7	alpha=0.0001 hidden layer=1	

Table 4: Performance of Different Models (processed data2)

model performs best under this condition.

Discussion

The data processed in the form of splitting after balancing achived good performance in general. However, it doesn't provide good indication to the real world data. The models applied with the processed data of splitting first then balancing, shows a high recall value in all, and low precision value. While precision means the correct predictions among all, and recall means the correct predictions retrieved from all suppose-to-be positive predictions, the true positive rate is high, and false positive rate is high. The models would very likely to predict patient with no stroke as having stroke. It makes sense as the way our models are trained reflects a problem caused by the arbitrarily balanced data, that the data is likely to be skewed.
Several recommendations can be made for UCLA hospital's future analytical work:

- From our models such as random forest, the best parameters are of low values. This indicates that the data set doesn't have in depth insight into the relationship between features given and the target value stroke. Thus it can be larger in quantity and more features of better professional value can be looked into, rather than a general basic information as provided.

- As the feature such as average glucose level is highly correlated with our target label, similar features regarding heart health status and blood index can be looked into. General information like the type of work that indicates stress level can also be served as qualitative measurements.

- If UCLA hospital were to incorporate our models, it is important to mention that a high false positive rate is an issue. Patients are suggested to be double checked based on an overall indicator.

Conclusion

In this project, we are given the dataframe of basic information of individuals with stroke or not. A machine learning pipeline is produced, several different models are incorporated to predict whether one has stroke or not based on the relevant features.
The steps are as follows: basic statistics are run first to gain insight into the provided data, then based on the findings, appropriate data processing techniques are chose, for example, useless features are dropped, missing values are imputed, and new feature are added using augmentation method. After encoding and scaling the data, into the form that can be best fed into the machine learning models, it is then split into training and testing set. Based on the imbalanced nature of our data, the training data is balanced to prevent data skewing. PCA is used to decrease data dimensionality, then several models, logistic regression, random forest, neural net, and SVM are assigned with its best parameters and their performance was compared.
In this project, we received industrial level real world data. We examined them, and used data science knowledge to solve for the domain challenge. Future work that can be done is also discussed to offer insights for the UCLA hospital.