[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as
plt import seaborn as sns
import sklearn import pandas
as pd import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import
PolynomialFeatures import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.metrics import confusion_matrix, roc_curve, auc,
classificatio from sklearn import preprocessing from patsy import dmatrix
import pandas as pd import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns #visualization library
from sklearn.linear_model import LogisticRegression #problem will be
solved from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis #
Q from sklearn.neighbors import KNeighborsClassifier #(KNN)
from sklearn.metrics import confusion_matrix, classification_report,
precis import statsmodels.api as sm #to compute p-values from patsy import
dmatrices import sklearn.linear_model as skl_lm from sklearn.metrics
import mean_squared_error
from sklearn.model_selection import train_test_split, LeaveOneOut, KFold,
c from sklearn.preprocessing import PolynomialFeatures from sklearn import
tree
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier,
exp from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegress from sklearn.metrics import confusion_matrix,
mean_squared_error from BorutaShap import BorutaShap
from sklearn.model_selection import train_test_split,
GridSearchCV from sklearn.svm import SVC, LinearSVC import
matplotlib.pyplot as plt
from matplotlib.font_manager import
FontProperties from matplotlib.patches import
Ellipse, Polygon import matplotlib.gridspec as
gridspec import matplotlib.colors from pylab
import rcParams
from matplotlib.font_manager import FontProperties from
mpl_toolkits.axes_grid1.inset_locator import inset_axes
plt.style.use('seaborn-whitegrid') import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (17, 6)
mpl.rcParams['axes.labelsize'] = 14
mpl.rcParams['xtick.labelsize'] = 12
mpl.rcParams['ytick.labelsize'] = 12
mpl.rcParams['text.color'] = 'k'
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

In

```
df=pd.read_csv('/content/gdrive/Shareddrives/ECON412/finalproject/smoking.c
df.head()
```

```
Mounted at /content/gdrive
```

Out[4]:

	ID	gender	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	he
0	0	F	40	155	60	81.3	1.2	1.0	1.0	
1	1	F	40	160	60	81.0	0.8	0.6	1.0	
2	2	M	55	170	60	80.0	0.8	0.8	1.0	
3	3	M	40	165	70	88.0	1.5	1.5	1.0	
4	4	F	40	155	60	86.0	1.0	1.0	1.0	

5 rows × 27 columns

We will now rename and map some of our predictors.

In [5]:

```
df1 = df.drop(columns='oral',axis=1)
df1['gender'] = df1['gender'].map({'F':0, 'M':1})
df1['tartar'] = df1['tartar'].map({'N':0, 'Y':1})
df1.rename(columns = {'fasting blood sugar':'fasting_blood_sugar'},
inplace df1.rename(columns = {'Urine protein':'Urine_protein'},
inplace=True ) df1.rename(columns = {'serum
creatinine':'serum_creatinine'}, inplace=True df1.rename(columns =
{'dental caries':'dental_caries'}, inplace=True ) df1.rename(columns =
{'height(cm)':'height'}, inplace=True ) df1.rename(columns =
{'weight(kg)':'weight'}, inplace=True ) df1.rename(columns =
{'waist(cm)':'waist'}, inplace=True )
df1.rename(columns = {'eyesight(left)':'eyesight_left'}, inplace=True )
df1.rename(columns = {'eyesight(right)':'eyesight_right'}, inplace=True )
df1.rename(columns = {'hearing(left)':'hearing_left'}, inplace=True )
df1.rename(columns = {'hearing(right)':'hearing_right'}, inplace=True )
```

In [ ]:

```
df1
```

Out[18]:

	ID	gender	age	height	weight	waist	eyesight_left	eyesight_right	hearing_left	hearing
0	0	0	40	155	60	81.3	1.2	1.0	1.0	
1	1	0	40	160	60	81.0	0.8	0.6	1.0	
2	2	1	55	170	60	80.0	0.8	0.8	1.0	
3	3	1	40	165	70	88.0	1.5	1.5	1.0	
4	4	0	40	155	60	86.0	1.0	1.0	1.0	
...	...	...	...	...	...	...	...	...	...	
55687	55676	0	40	170	65	75.0	0.9	0.9	1.0	
55688	55681	0	45	160	50	70.0	1.2	1.2	1.0	
55689	55683	0	55	160	50	68.5	1.0	1.2	1.0	
55690	55684	1	60	165	60	78.0	0.8	1.0	1.0	
55691	55691	1	55	160	65	85.0	0.9	0.7	1.0	

55692 rows × 26 columns

In [ ]:

```
df1.corr()
```

Out[19]:

ID age e

gender height weight waist eyesight_left

ID	1.000000	0.008657	-0.000825	0.006306	0.004814	0.005384	0.009616
gender	0.008657	1.000000	-0.290095	0.741556	0.574956	0.419568	0.127424
age	-0.000825	-0.290095	1.000000	-0.479528	-0.324706	-0.026297	-0.195472
height	0.006306	0.741556	-0.479528	1.000000	0.675656	0.378902	0.151133
weight	0.004814	0.574956	-0.324706	0.675656	1.000000	0.822842	0.108433
waist	0.005384	0.419568	-0.026297	0.378902	0.822842	1.000000	0.027458
eyesight_left	0.009616	0.127424	-0.195472	0.151133	0.108433	0.027458	1.000000
eyesight_right	0.003088	0.125680	-0.192723	0.155665	0.113155	0.037996	0.354574
hearing_left	0.002676	-0.009407	0.203993	-0.078663	-0.050094	0.023790	-0.046571
hearing_right	-0.004959	-0.011579	0.208722	-0.078323	-0.052836	0.019286	-0.048788
systolic	0.002489	0.167289	0.134023	0.080585	0.266131	0.316922	-0.019330
relaxation	0.004649	0.177891	0.050745	0.113193	0.271634	0.292627	0.005199
fasting_blood_sugar	0.001493	0.098117	0.182351	0.019619	0.136237	0.211132	-0.041851
Cholesterol	-0.001092	-0.085270	0.055557	-0.082161	0.026403	0.065467	-0.004985
triglyceride	0.002314	0.241520	0.015102	0.156693	0.324429	0.361922	0.019717
HDL	-0.005464	-0.306728	0.007047	-0.213284	-0.358868	-0.376203	-0.015296
LDL	0.001429	-0.042525	0.043007	-0.048419	0.040560	0.072817	-0.007257
hemoglobin	0.006464	0.702214	-0.263078	0.539367	0.492970	0.387066	0.095234
Urine_protein	0.000382	0.015907	0.029625	0.005128	0.032566	0.045492	-0.002752
serum_creatinine	0.003830	0.507249	-0.106118	0.383883	0.324808	0.235024	0.071410
AST	-0.001865	0.095718	0.032576	0.041737	0.120130	0.142690	-0.007966
ALT	-0.002803	0.167903	-0.063937	0.126511	0.250634	0.252478	0.019326
Gtp	0.000823	0.237270	0.013031	0.139720	0.209625	0.243141	0.003850
dental_caries	0.000641	0.084408	-0.114984	0.079331	0.073536	0.044203	0.003684
tartar	0.002474	0.055473	-0.081796	0.055513	0.059921	0.046197	0.012532
smoking	0.011476	0.510340	-0.162557	0.396675	0.302780	0.226259	0.061204

26 rows × 26 columns

```
, X = dmatrices('smoking ~ age + gender + height + weight + waist + eyesight
ogit = sm.Logit(Y, X) esults_logit = logit.fit()
rint(results_logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.472327
         Iterations 7
                           Logit Regression Results
```

In [ ]:

```
==============================================================================
=====
Dep. Variable:                smoking   No. Observations:                  
55692
Model:                          Logit   Df Residuals:                      
55667
Method:                           MLE   Df Model:                          
24
Date:                Sun, 29 May 2022   Pseudo R-squ.:                    
0.2816
Time:                        14:29:47   Log-Likelihood:                   -2
6305.
converged:                       True   LL-Null:                          -3
6617.
Covariance Type:            nonrobust   LLR p-value:                      
0.000
==============================================================================
==============
                          coef    std err          z      P>|z|      [0.0
25      0.975]
--------------------------------------------------------------------------------
-----------
Intercept              -6.7684      0.430    -15.735      0.000      -7.6
11      -5.925
age                    -0.0002      0.001     -0.158      0.875      -0.0
02       0.002
gender                  2.9168      0.051     56.774      0.000       2.8
16       3.017
height                  0.0225      0.002     10.117      0.000       0.0
18       0.027
weight                 -0.0103      0.002     -4.776      0.000      -0.0
15      -0.006
waist                  -0.0012      0.003     -0.465      0.642      -0.0
06       0.004
eyesight_left          -0.0196      0.023     -0.870      0.384      -0.0
64       0.025
eyesight_right         -0.0103      0.023     -0.444      0.657      -0.0
56       0.035
hearing_left           -0.2188      0.081     -2.688      0.007      -0.3
78      -0.059
hearing_right           0.0213      0.080      0.264      0.792      -0.1
36       0.179
systolic               -0.0144      0.001    -11.401      0.000      -0.0
17      -0.012
relaxation              0.0097      0.002      5.604      0.000       0.0
06       0.013
fasting_blood_sugar     0.0035      0.001      6.367      0.000       0.0
02       0.005
```

In

```
Cholesterol             -0.0024      0.001     -4.619      0.000      -0.0
03      -0.001
triglyceride             0.0047      0.000     23.358      0.000       0.0
04       0.005
HDL                      0.0021      0.001      2.105      0.035       0.0
00       0.004
LDL                     -0.0001      0.000     -0.350      0.726      -0.0
01       0.001
hemoglobin               0.1390      0.011     12.745      0.000       0.1
18       0.160
Urine_protein            0.0117      0.027      0.439      0.660      -0.0
41       0.064
serum_creatinine        -0.8690      0.067    -12.949      0.000      -1.0
00      -0.737
AST                     -0.0011      0.001     -1.050      0.294      -0.0
03       0.001
ALT                     -0.0054      0.001     -6.835      0.000      -0.0
07      -0.004
Gtp                      0.0073      0.000     22.203      0.000       0.0
07       0.008
dental_caries            0.3163      0.026     12.285      0.000       0.2
66       0.367
tartar                   0.3376      0.022     15.472      0.000       0.2
95       0.380
=========================================================================
==============
```

Based on the P-values, the following variables are significant: gender, height, weight, relaxation, systolic, fasting_blood_sugar, Cholesterol, triglyceride, HDL, hemoglobin, serum_creatinine, ALT, Gtp, dental_caries & tartar. Now we will run another logit using only the signficant predictors.

[ ]:

```
Y, X = dmatrices('smoking ~ gender + height + weight + systolic + relaxatio
logit1 = sm.Logit(Y, X) results_logit1 = logit1.fit()
print(results_logit1.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.472436
         Iterations 7
                           Logit Regression Results
=========================================================================
=====
Dep. Variable:                smoking   No. Observations:
55692
Model:                          Logit   Df Residuals:
55676
Method:                           MLE   Df Model:
15
Date:                Sun, 29 May 2022   Pseudo R-squ.:
0.2815
Time:                        14:29:48   Log-Likelihood:                   -2
```

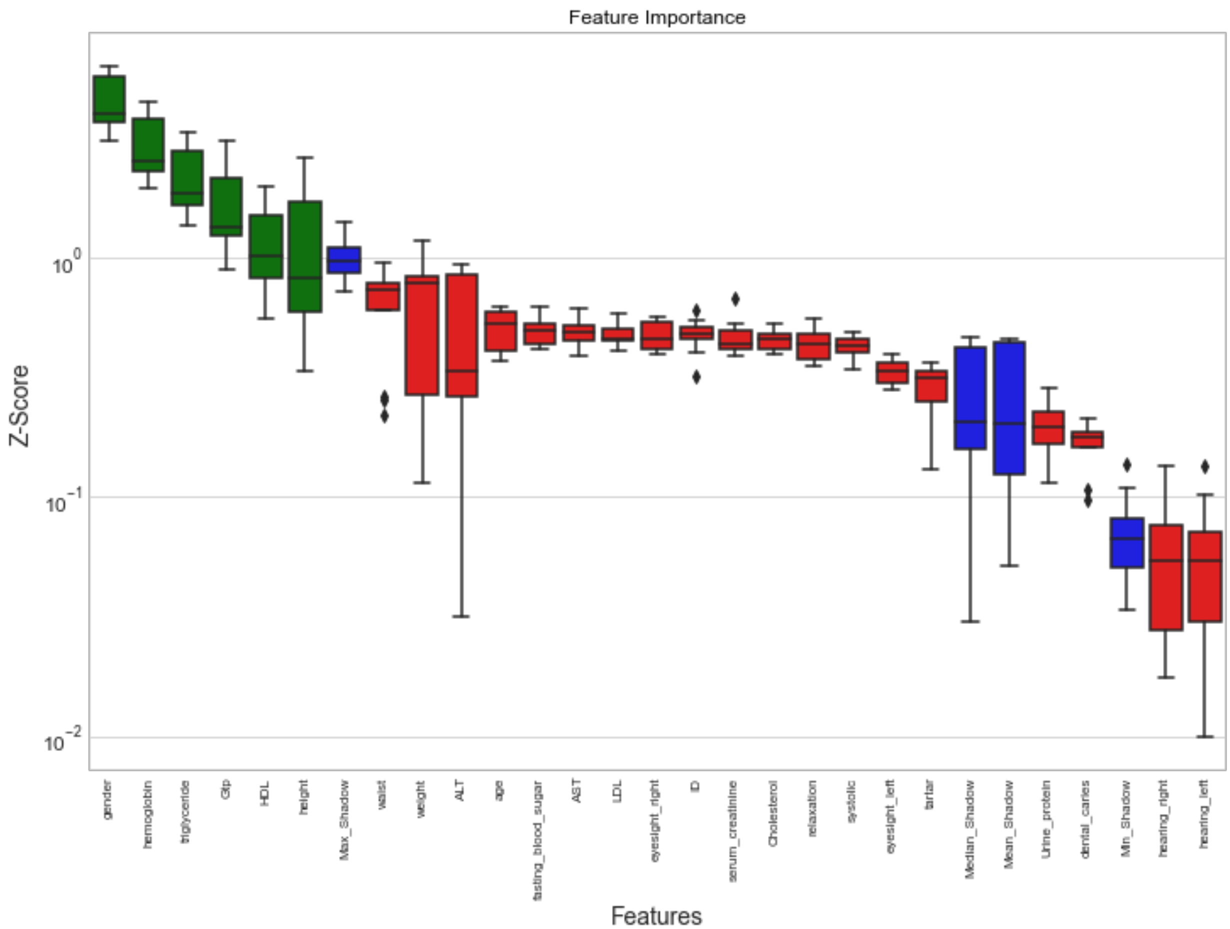
```
6311.
converged:                           True   LL-Null:                        -3
6617.
Covariance Type:              nonrobust   LLR p-value:
0.000
==========================================================================
==============
                          coef    std err          z      P>|z|      [0.0
25      0.975]
---------------------------------------------------------------------------
------------
Intercept               -7.1971      0.348    -20.671      0.000      -7.8
79      -6.515
gender                   2.9039      0.051     57.059      0.000       2.8
04       3.004
height                   0.0232      0.002     11.450      0.000       0.0
19       0.027
weight                  -0.0109      0.001     -8.584      0.000      -0.0
13      -0.008
systolic                -0.0146      0.001    -11.700      0.000      -0.0
17      -0.012
relaxation               0.0098      0.002      5.702      0.000       0.0
06       0.013
fasting_blood_sugar      0.0035      0.001      6.467      0.000       0.0
02       0.005
Cholesterol             -0.0026      0.000     -7.799      0.000      -0.0
03      -0.002
triglyceride             0.0048      0.000     25.609      0.000       0.0
04       0.005
HDL                      0.0023      0.001      2.507      0.012       0.0
01       0.004
hemoglobin               0.1408      0.011     13.130      0.000       0.1
20       0.162
serum_creatinine        -0.8733      0.067    -13.094      0.000      -1.0
04      -0.743
ALT                     -0.0060      0.001    -10.572      0.000      -0.0
07      -0.005
Gtp                      0.0073      0.000     22.287      0.000       0.0
07       0.008
dental_caries            0.3184      0.026     12.430      0.000       0.2
68       0.369
tartar                   0.3398      0.022     15.600      0.000       0.2
97       0.382
==========================================================================
==============
```

In [ ]:
```
X = df1.drop(['smoking'], axis = 1)
Y = df1['smoking']
X_train1, X_test1, Y_train1, Y_test1 = train_test_split(X, Y,
test_size=0.0


Feature_Selector = BorutaShap(importance_measure='shap',
classification=Tru
```

In

```
Feature_Selector.fit(X=X_test1, y=Y_test1, n_trials=20, random_state=0)
Feature_Selector.plot(which_features='all')
```

```
  0%|          | 0/20 [00:00<?, ?it/s]

6 attributes confirmed important: ['gender', 'height', 'triglyceride', 'H
DL', 'Gtp', 'hemoglobin']
19 attributes confirmed unimportant: ['eyesight_right', 'ALT', 'weight',
'relaxation', 'hearing_right', 'ID', 'tartar', 'hearing_left', 'systoli
c', 'Cholesterol', 'dental_caries', 'Urine_protein', 'eyesight_left',
'se rum_creatinine', 'age', 'fasting_blood_sugar', 'LDL', 'AST',
'waist'] 0 tentative attributes remains: []
```



All of our chosen variables are statistically signficant, now we will proceed with our analysis.

In [ ]:
```
X1 = df1[['gender', 'height', 'weight', 'relaxation', 'systolic', 'fasting_
Y1 = df1.smoking
```

In [ ]:
```
X1_train,X1_test,Y1_train,Y1_test = sklearn.model_selection.train_test_spli
```

In [ ]:
```
lr = LogisticRegression()
```

In [ ]:
```
logit2 = lr.fit(X1_train,Y1_train)
```

In [ ]:
```
conf_mat = confusion_matrix(Y1_test, lr.predict(X1_test))
print(conf_mat)

lr.score(X1_test, Y1_test)
print('Test Accuracy =', lr.score(X1_test, Y1_test))
```

```
[[7051 1814]
 [2072 2986]]
Test Accuracy = 0.720893485599368
```

LDA

In [ ]:
```
lda = LinearDiscriminantAnalysis()
lda.fit(X1_train,Y1_train)
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
solver='svd', store_covariance=False, tol=0.0001)
```

Out[27]:
```
LinearDiscriminantAnalysis()
```

In [ ]:
```
conf_mat = confusion_matrix(Y1_test, lda.predict(X1_test))
print(conf_mat) lda.score(X1_test, Y1_test) print('Test
Accuracy =', lda.score(X1_test, Y1_test))
```

```
[[6515 2350]
 [1203 3855]]
Test Accuracy = 0.7448107448107448
```

LDA performed better than the Logit Model.

QDA

In [ ]:
```
qda = QuadraticDiscriminantAnalysis()
qda.fit(X1_train,Y1_train)
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
store_covariance=False, tol=0.0001)
```

Out[29]:
```
QuadraticDiscriminantAnalysis()
```

In [ ]:
```
conf_mat = confusion_matrix(Y1_test, qda.predict(X1_test))
print(conf_mat) qda.score(X1_test, Y1_test) print('Test
Accuracy =', qda.score(X1_test, Y1_test))
```

```
[[6528 2337]
 [1374 3684]]
Test Accuracy = 0.7334626158155569
```

LDA performed better than QDA, implying that our classes might not require a non-linear classifier.

Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X1_train, Y1_train)
```

Out[31]:
```
GaussianNB()
```

In [ ]:
```
Y1_pred = classifier.predict(X1_test)
```

In [ ]:
```
conf_mat = confusion_matrix(Y1_test, Y1_pred)
print(conf_mat)
accuracy_score(Y1_test, Y1_pred)
print('Test Accuracy =', accuracy_score(Y1_test, Y1_pred))
```

```
[[5582 3283]
 [ 804 4254]]
Test Accuracy = 0.7064569417510594
```

Naive Bayes performs poorly when compared to LDA, QDA and Logit.

KNN

In [ ]:
```
nbrs1 = KNeighborsClassifier(n_neighbors=1)
nbrs1.fit(X1_train,Y1_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=1, p=2,
weights='uniform')
```

Out[7]:
```
KNeighborsClassifier(n_jobs=1, n_neighbors=1)
```

In [ ]:
```
conf_mat = confusion_matrix(Y1_test, nbrs1.predict(X1_test))
print(conf_mat)
nbrs1.score(X1_test, Y1_test) print('Test Accuracy =',
nbrs1.score(X1_test, Y1_test))
```

```
[[7192 1673]
 [1725 3333]]
```

In [ ]:

```
Test Accuracy = 0.7559434030022265
```

```
nbrs2 = KNeighborsClassifier(n_neighbors=2)
nbrs2.fit(X1_train,Y1_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=2, p=2,
weights='uniform')
```

Out[36]: KNeighborsClassifier(n_jobs=1, n_neighbors=2)

In [ ]:

```
conf_mat = confusion_matrix(Y1_test, nbrs2.predict(X1_test))
print(conf_mat)
nbrs2.score(X1_test, Y1_test) print('Test Accuracy =',
nbrs2.score(X1_test, Y1_test))
```

```
[[7909  956]
 [3043 2015]]
Test Accuracy = 0.7127774186597716
```

In [ ]:

```
nbrs3 = KNeighborsClassifier(n_neighbors=3)
nbrs3.fit(X1_train,Y1_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=3, p=2,
weights='uniform')
```

Out[38]: KNeighborsClassifier(n_jobs=1, n_neighbors=3)

In [ ]:

```
conf_mat = confusion_matrix(Y1_test, nbrs3.predict(X1_test))
print(conf_mat)
nbrs3.score(X1_test, Y1_test) print('Test Accuracy =',
nbrs3.score(X1_test, Y1_test))
```

```
[[6902 1963]
 [2105 2953]]
Test Accuracy = 0.7078215901745314
```

In [ ]:

```
nbrs4 = KNeighborsClassifier(n_neighbors=4)
nbrs4.fit(X1_train,Y1_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=4, p=2,
weights='uniform')
```

Out[40]: KNeighborsClassifier(n_jobs=1, n_neighbors=4)

In [ ]:

```
conf_mat = confusion_matrix(Y1_test, nbrs4.predict(X1_test))
print(conf_mat)
nbrs4.score(X1_test, Y1_test) print('Test Accuracy =',
nbrs4.score(X1_test, Y1_test))
```

```
[[7645 1220]
 [2841 2217]]
Test Accuracy = 0.708324355383179
```

In [ ]:

```
nbrs5 = KNeighborsClassifier(n_neighbors=5)
nbrs5.fit(X1_train,Y1_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=5, p=2,
weights='uniform')
```

Out[42]:

```
KNeighborsClassifier(n_jobs=1)
```

In [ ]:

```
conf_mat = confusion_matrix(Y1_test, nbrs5.predict(X1_test))
print(conf_mat)
nbrs5.score(X1_test, Y1_test) print('Test Accuracy =',
nbrs5.score(X1_test, Y1_test))
```

```
[[6933 1932]
 [2134 2924]]
Test Accuracy = 0.7079652373770021
```

KNN with 1 neighbor performed better when compared to other KNNs and all the models ran so far.

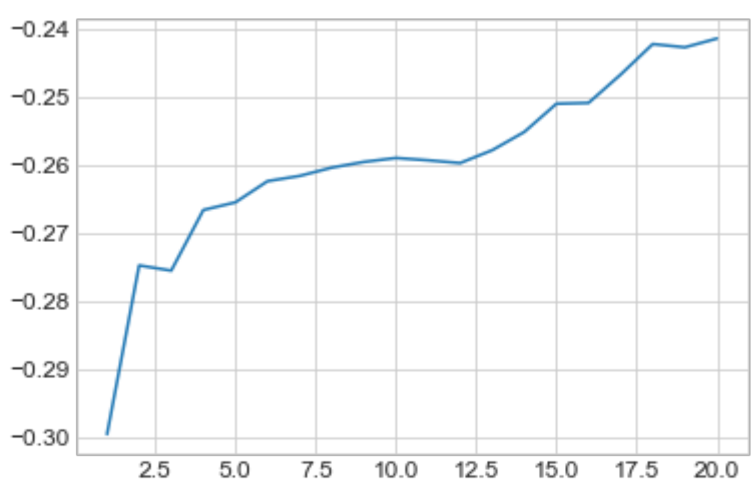
Decision Tree Classifier

In [ ]:

```
tree_depth = []

for i in range(1,21):
    cv_tree = DecisionTreeClassifier(max_depth=i)
    scores  = cross_val_score(estimator=cv_tree, X=X1_train, y=Y1_train, cv
    tree_depth.append(scores.mean())


plt.plot(range(1,21), tree_depth)
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x7fb2709eb280>]
```



According to the cross validation plot, the tree depth that produces the lowest training MSE is around 20.

In [ ]:
```
clf = DecisionTreeClassifier(max_depth=20)
clf.fit(X1_train, Y1_train) print("Training Accuracy =",
clf.score(X1_train, Y1_train))
```

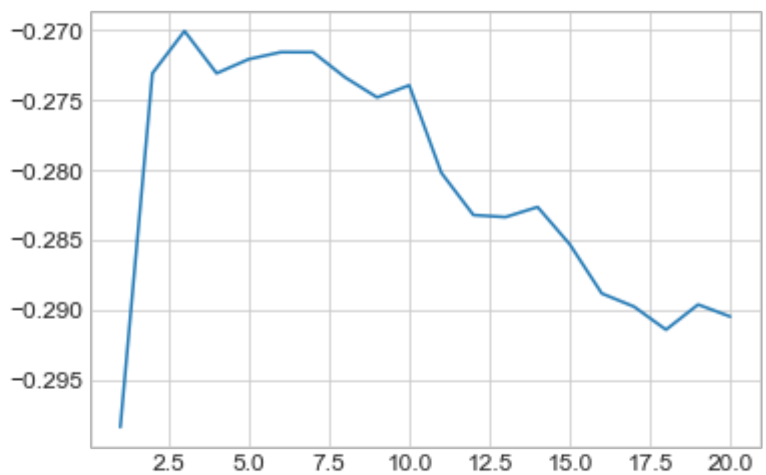
```
Training Accuracy = 0.9700256170844406
```

In [ ]:
```
pred1 = clf.predict(X1_test) cm = pd.DataFrame(confusion_matrix(Y1_test,
pred1).T, index=['No', 'Yes'], print(cm) print('Test Accuracy
=',(10707/13923))
```

```
       No   Yes
No   7203  1555
Yes  1662  3503
Test Accuracy = 0.7690152984270632
```

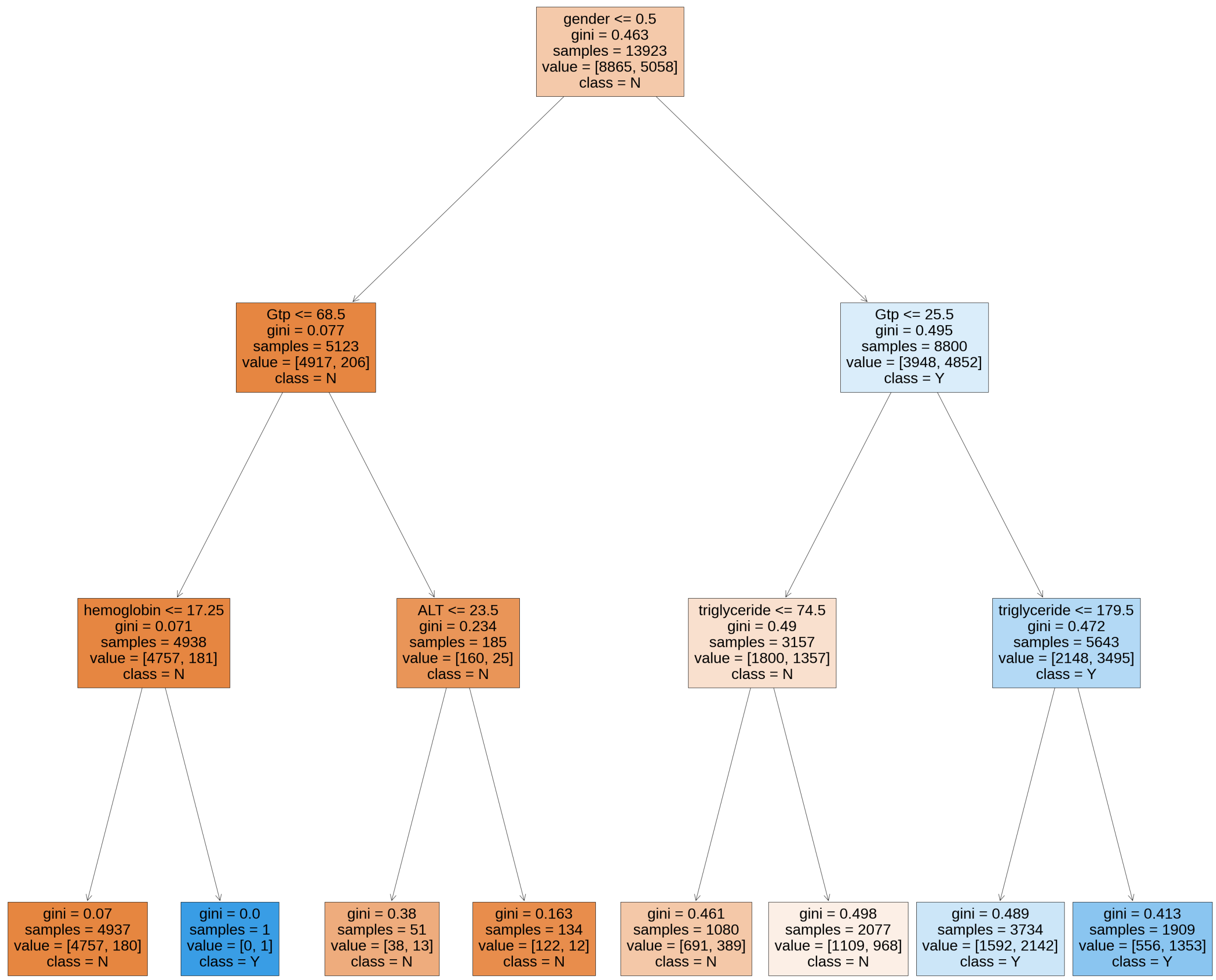
In [ ]:
```
tree_depth = []

for i in range(1,21):
    cv_tree = DecisionTreeClassifier(max_depth=i)
    scores  = cross_val_score(estimator=cv_tree, X=X1_test, y=Y1_test, cv=1
    tree_depth.append(scores.mean())


plt.plot(range(1,21), tree_depth)
```

Out[12]: [<matplotlib.lines.Line2D at 0x7fb26ce850a0>]



In [ ]:

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (60,60))
tree.plot_tree(clf1,feature_names = X1_test.columns,filled = True, class_na
```



So far, Decision Tree Calssifier performed the best with a test accuracy of approximately 77%

SVC

In [ ]:
```
svc1 = SVC(C = 0.01)
Fit1 = svc1.fit(X1_train, Y1_train)
```

In [ ]:
```
pred2 = svc1.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred2).T, index=['No', 'Yes'],
print(cm)
print('Test Accuracy =',(9606/13923))
```

```
       No   Yes
No   8047  3499
Yes   818  1559
Test Accuracy = 0.6899375134669252
```

In [ ]:
```
svc2 = SVC(C = 0.1)
Fit2 = svc2.fit(X1_train, Y1_train)
```

In [ ]:
```
pred3 = svc2.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred3).T, index=['No', 'Yes'],
print(cm)
print('Test Accuracy =',(10005/13923))
```

```
       No   Yes
No   7687  2740
Yes  1178  2318
Test Accuracy = 0.7185951303598362
```

In [ ]:
```
svc3 = SVC(C = 1)
Fit3 = svc3.fit(X1_train, Y1_train)
```

In [ ]:
```
pred4 = svc3.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred4).T, index=['No', 'Yes'],
print(cm)
print('Test Accuracy =',(10301/13923))
```

```
       No   Yes
No   7454  2211
Yes  1411  2847
Test Accuracy = 0.7398549163255046
```

In [ ]:
```
svc4 = SVC(C = 5)
Fit4 = svc4.fit(X1_train, Y1_train)
```

In [ ]:
```
pred5 = svc4.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred5).T, index=['No', 'Yes'],
print(cm)
print('Test Accuracy =',(10389/13923))
```

```
       No   Yes
No   7322  1991
Yes  1543  3067
Test Accuracy = 0.7461753932342168
```

In [ ]:
```
svc5 = SVC(C = 10)
Fit5 = svc5.fit(X1_train, Y1_train)
```

In [ ]:
```
pred6 = svc5.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred6).T, index=['No', 'Yes'],
print(cm)
print('Test Accuracy =',(10408/13923))
```

```
       No   Yes
No   7237  1887
Yes  1628  3171
Test Accuracy = 0.7475400416576887
```

SVC with a linear kernel and C=10 performed better than SVCs with lower values of C, however, it still could not beat the perfromance of the Decision Tree Classifier.

SVM

In [ ]:
```
svm1 = SVC(C=1, kernel='rbf')
Fit6 = svm1.fit(X1_train, Y1_train)
```

In [ ]:
```
pred7 = svm1.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred7).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7454  2211
Yes  1411  2847
```

In [ ]:
```
print('Test Accuracy =',((7454+2847)/13923))
```

```
Test Accuracy = 0.7398549163255046
```

In [ ]:
```
svm2 = SVC(C=3, kernel='rbf')
Fit7 = svm2.fit(X1_train, Y1_train)
```

In [ ]:
```
pred8 = svm2.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred8).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7365  2063
Yes  1500  2995
```

In [ ]:
```
print('Test Accuracy =',((7365+2995)/13923))
```

```
Test Accuracy = 0.7440925087983912
```

In [ ]:

```
svm3 = SVC(C=5, kernel='rbf')
Fit8 = svm3.fit(X1_train, Y1_train)
```

In [ ]:

```
pred9 = svm3.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred9).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7322  1991
Yes  1543  3067
```

In [ ]:

```
print('Test Accuracy =',((7322+3067)/13923))
```

```
Test Accuracy = 0.7461753932342168
```

In [ ]:

```
svm4 = SVC(C=8, kernel='rbf')
Fit9 = svm4.fit(X1_train, Y1_train)
```

In [ ]:

```
pred10 = svm4.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred10).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7268  1927
Yes  1597  3131
```

In [ ]:

```
print('Test Accuracy =',((7268+3131)/13923))
```

```
Test Accuracy = 0.7468936292465704
```

In [ ]:

```
svm5 = SVC(C=10, kernel='rbf')
Fit10 = svm5.fit(X1_train, Y1_train)
```

In [ ]:

```
pred11 = svm5.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred11).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7237  1887
Yes  1628  3171
```

In [ ]:

```
print('Test Accuracy =',((7237+3171)/13923))
```

```
Test Accuracy = 0.7475400416576887
```

In [ ]:

```
svm6 = SVC(C=1, kernel='poly')
Fit11 = svm6.fit(X1_train, Y1_train)
```

```
pred12 = svm6.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred12).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7361  2093
```

In [ ]:
```
Yes   1504  2965
```

In [ ]:
```python
print('Test Accuracy =',((7361+2965)/13923))
```
```
Test Accuracy = 0.7416505063563887
```

In [ ]:
```python
svm10 = SVC(C=3, kernel='poly')
Fit14 = svm10.fit(X1_train, Y1_train)
```

In [ ]:
```python
pred16 = svm10.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred16).T, index=['No', 'Yes'],
print(cm)
```
```
       No   Yes
No   7264  1945
Yes  1601  3113
```

In [ ]:
```python
print('Test Accuracy =',((7264+3113)/13923))
```
```
Test Accuracy = 0.7453135100193924
```

In [ ]:
```python
svm7 = SVC(C=5, kernel='poly')
Fit12 = svm7.fit(X1_train, Y1_train)
```

In [ ]:
```python
pred13 = svm7.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred13).T, index=['No', 'Yes'],
print(cm)
```
```
       No   Yes
No   7189  1826
Yes  1676  3232
```

In [ ]:
```python
print('Test Accuracy =',((7189+3232)/13923))
```
```
Test Accuracy = 0.7484737484737485
```

In [ ]:
```python
svm8 = SVC(C=8, kernel='poly')
Fit13 = svm8.fit(X1_train, Y1_train)
```

In [ ]:
```python
pred14 = svm8.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred14).T, index=['No', 'Yes'],
print(cm)
```
```
       No   Yes
No   7136  1747
Yes  1729  3311
```

```python
print('Test Accuracy =',((7136+3311)/13923))
```

In [ ]:

```
Test Accuracy = 0.750341162105868
```

In [ ]:

```
svm9 = SVC(C=10, kernel='poly')
Fit13 = svm9.fit(X1_train, Y1_train)
```

In [ ]:

```
pred15 = svm9.predict(X1_test)
cm = pd.DataFrame(confusion_matrix(Y1_test, pred15).T, index=['No', 'Yes'],
print(cm)
```

```
       No   Yes
No   7091  1703
Yes  1774  3355
```

In [ ]:

```
print('Test Accuracy =',((7091+3355)/13923))
```

```
Test Accuracy = 0.7502693385046326
```

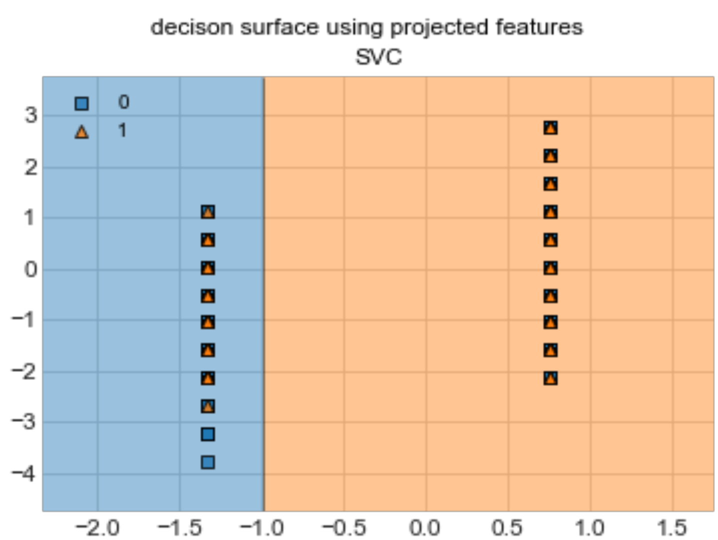
SVM with C=8 and a polynomial kernel performed best when compared to other SVMs and SVCs, however, it still couldnt beat the performance of the Decision Tree Classifier.

In [ ]:

```
x = preprocessing.scale(X1)
y = np.ravel(Y1)
```

In [ ]:

```
fig  = plt.figure()

fig.suptitle('decison surface using projected features')
labels = ['SVC', 'SVM poly', 'SVM radial' ]
gs = gridspec.GridSpec(1, 1)
for svm8, lab, grd in zip([Fit13], labels, ([0,0], [1,0],[2,0])):
    svm8.fit(np.stack((x[:,0], x[:,1]), axis=-1), y)
    ax  = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=np.stack((x[:,0], x[:,1]), axis=-1), y=y,
    plt.title(lab)
plt.show()
```



In [ ]:

```
plt.style.use('seaborn-whitegrid')
fig, ax = plt.subplots(1, figsize=(15,6))
# false positive rates and true positive rates
fpr, tpr, _ = roc_curve(Y1_test, clf.predict(X1_test))
fpr1, tpr1, _ = roc_curve(Y1_test,  nbrs1.predict(X1_test))
_ = ax.plot(fpr, tpr, lw=2, label='Classification Tree. ROC curve (Area = %
_ = ax.plot(fpr1, tpr1, lw=2, label='KNN. ROC curve (Area = %0.2f)' % auc(f
_ = ax.set_title('Classification Tree and KNN n=1')
_ = ax.plot([0, 1], 'k--', lw=2)
_ = ax.set_xlim([-0.05, 1.0])
_ = ax.set_ylim([0.0, 1.05])
_ = ax.set_xlabel('False Positive Rate (FPR)')
_ = ax.set_ylabel('True Positive Rate (TPR)')
_ = ax.legend(loc='center left', bbox_to_anchor=(.05, -0.3), fontsize = 12)
_ = ax.set_aspect(1)
```