

```
In [1]: """
ECGR 5105 - Intro to Machine Learning
Homework 4 - Part 1
Phillip Harmon
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [2]: #Load and Build the Dataset
from sklearn.datasets import load_breast_cancer

loaded = load_breast_cancer()

labels = np.reshape(loaded.target, (len(loaded.target),1))
inputs = pd.DataFrame(loaded.data)
names = np.append(loaded.feature_names, 'label')

dataset = pd.DataFrame(np.concatenate([inputs,labels],axis=1))
dataset.columns = names

dataset
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.180
...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.172
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.175
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.159
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.239
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.158

569 rows × 31 columns

```
In [3]: #Sort and clean the Dataset
        from sklearn.preprocessing import MinMaxScaler, StandardScaler

        x = dataset.iloc[:,0:-1].values
        y = dataset.iloc[:, -1].values

        scaler = MinMaxScaler()
        x = scaler.fit_transform(x)
```

```

In [4]: #Explore Default SVC with A range of PCA Component Counts
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, cla

frameLog = []
modelLog = []
accuracyLog = []
precisionLog = []
recallLog = []
cols = []
maxPC = len(x[0])+1

for k in range(1,maxPC):

    pca = PCA(n_components = k)
    pcs = pca.fit_transform(x)
    cols.append('PC'+str(k))
    pcFrame = pd.DataFrame(data=pcs,columns=cols)
    frameLog.append(pcFrame)

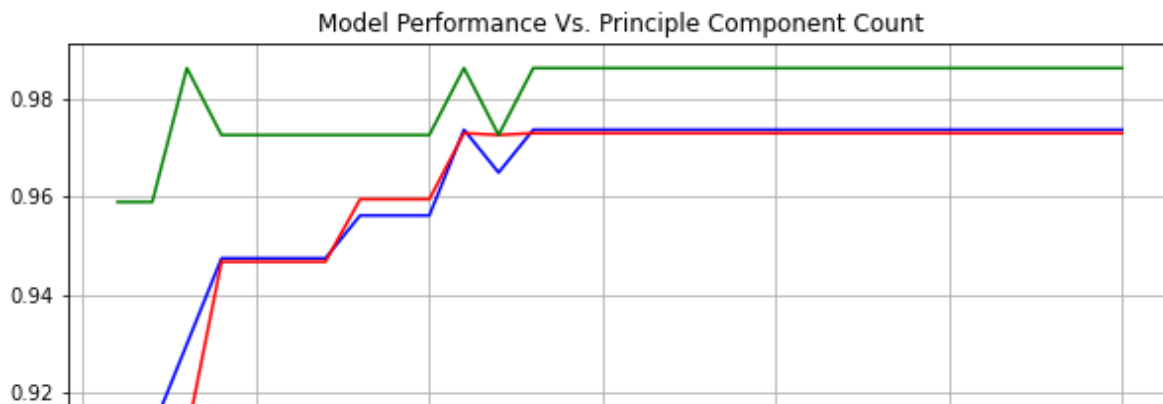
    xt, xv, yt, yv = train_test_split(pcFrame, y,
                                      train_size = 0.8, test_size = 0.2,
                                      random_state=1337)

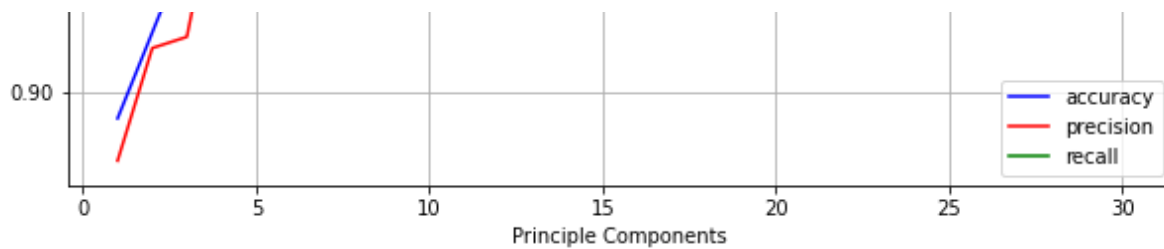
    model = SVC(random_state=1337)
    model.fit(xt,yt)
    modelLog.append(model)

    yp = model.predict(xv)
    accuracyLog.append(accuracy_score(yv,yp))
    precisionLog.append(precision_score(yv,yp))
    recallLog.append(recall_score(yv,yp))

plt.rcParams["figure.figsize"] = (10,5)
plt.grid()
plt.xlabel('Principle Components')
plt.title('Model Performance Vs. Principle Component Count')
plt.plot(range(1,maxPC),accuracyLog,color='blue',label='accuracy')
plt.plot(range(1,maxPC),precisionLog,color='red',label='precision')
plt.plot(range(1,maxPC),recallLog,color='green',label='recall')
plt.legend();

```





```

In [5]: #Print Best Results
K = accuracyLog.index(max(accuracyLog))
print("According to the plot above, the highest accuracy occurs at a lowest di
xt, xv, yt, yv = train_test_split(frameLog[K], y, train_size = 0.8, test_size =
yp = modelLog[K].predict(xv)
print("Classification Report for K={}".format(K+1))
print("-----")
print(classification_report(yv,yp))

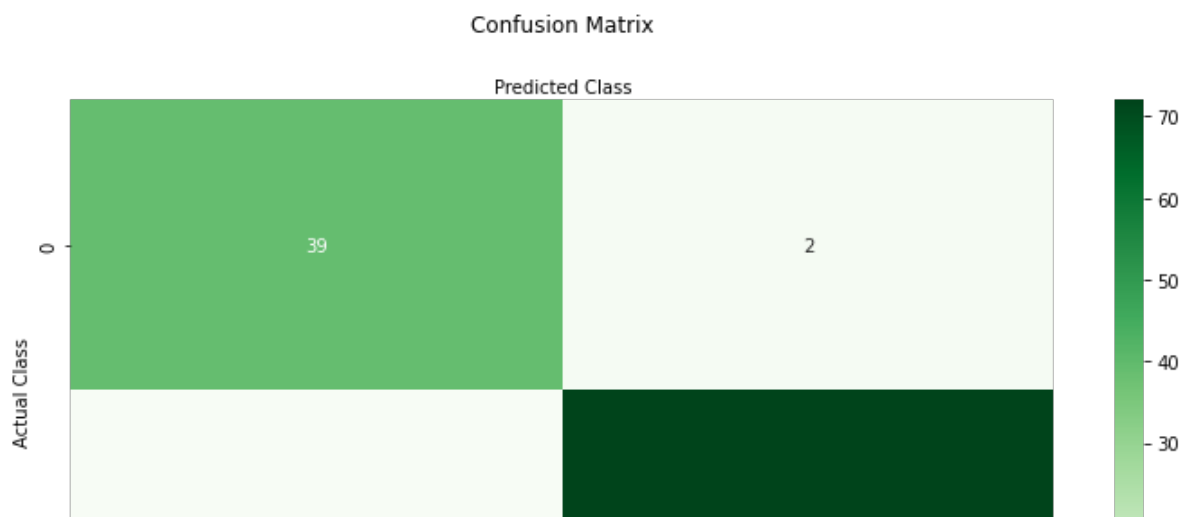
#Analyze using the Confusion Matrix
import seaborn as sns
classes = ['Benign', 'Malignant']
figure, axis = plt.subplots()
ticks = np.arange(len(classes))
plt.xticks(ticks, classes)
plt.yticks(ticks, classes)
sns.heatmap(pd.DataFrame(confusion_matrix(yv,yp)),
            annot=True, cmap="Greens", fmt='g')
axis.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1)
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()

```

According to the plot above, the highest accuracy occurs at a lowest dimensionality of K=11

Classification Report for K=11

	precision	recall	f1-score	support
0.0	0.97	0.95	0.96	41
1.0	0.97	0.99	0.98	73
accuracy			0.97	114
macro avg	0.97	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114



```
In [6]: #Refresh the Dataset
x = dataset.iloc[:,0:-1].values
y = dataset.iloc[:, -1].values

scaler = MinMaxScaler()
x = scaler.fit_transform(x)
```

```

In [7]: #Now to explore different Kernelizations and parameters
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV

kernels = ['rbf', 'linear', 'sigmoid', 'poly']

parameters = {
    'pca__n_components' : range(1, len(x[0])+1),
    'svc__C'            : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'svc__gamma'        : [0.000001, 0.00001, 0.0001, 0.001, 'auto', 'scale']
}

pca = PCA(random_state=1337)

modelLog = dict()
accuracyLog = dict()
precisionLog = dict()
recallLog = dict()

for colonel in kernels:
    svc = SVC(kernel=colonel, random_state=1337)
    pipeline = make_pipeline(pca, svc)

    grid = GridSearchCV(pipeline, parameters)

    print("\n=====
    print("Searching a variety of Parameters to find the best model with {}-ty
    %time grid.fit(xt, yt)
    print("\nBest model among all parameter options for {}-type kernelization:
    print(grid.best_params_)

    yp = grid.predict(xv)
    print("\nReport for Best Model Found for {}-type kernelization:".format(co
    print("-----")
    print(classification_report(yv, yp))
    print("Confusion Matrix:")
    print(confusion_matrix(yv, yp))

    modelLog[colonel] = grid
    accuracyLog[colonel] = accuracy_score(yv, yp)
    precisionLog[colonel] = precision_score(yv, yp)
    recallLog[colonel] = recall_score(yv, yp)

```

```

=====
=====

```

Searching a variety of Parameters to find the best model with rbf-type kernelization...

Wall time: 1min 36s

Best model among all parameter options for rbf-type kernelization:
{'pca__n_components': 7, 'svc__C': 7, 'svc__gamma': 'scale'}

Report for Best Model Found for rbf-type kernelization:

```

-----

```

	precision	recall	f1-score	support
0.0	0.90	0.90	0.90	41
1.0	0.95	0.95	0.95	73
accuracy			0.93	114
macro avg	0.92	0.92	0.92	114
weighted avg	0.93	0.93	0.93	114

Confusion Matrix:

```
[[37  4]
 [ 4 69]]
```

```
=====
=====
Searching a variety of Parameters to find the best model with linear-type kernelization...
```

Wall time: 42.8 s

Best model among all parameter options for linear-type kernelization:

```
{'pca_n_components': 8, 'svc_C': 3, 'svc_gamma': 1e-06}
```

Report for Best Model Found for linear-type kernelization:

```
-----
```

	precision	recall	f1-score	support
0.0	0.98	0.98	0.98	41
1.0	0.99	0.99	0.99	73
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

Confusion Matrix:

```
[[40  1]
 [ 1 72]]
```

```
=====
=====
Searching a variety of Parameters to find the best model with sigmoid-type kernelization...
```

Wall time: 1min 16s

Best model among all parameter options for sigmoid-type kernelization:

```
{'pca_n_components': 10, 'svc_C': 7, 'svc_gamma': 'auto'}
```

Report for Best Model Found for sigmoid-type kernelization:

```
-----
```

	precision	recall	f1-score	support
0.0	0.97	0.88	0.92	41
1.0	0.94	0.99	0.96	73
accuracy			0.95	114

macro avg	0.95	0.93	0.94	114
weighted avg	0.95	0.95	0.95	114

Confusion Matrix:

```
[[36  5]
 [ 1 72]]
```

```
=====
=====
Searching a variety of Parameters to find the best model with poly-type kernelization...
```

Wall time: 1min 8s

Best model among all parameter options for poly-type kernelization:
{'pca__n_components': 8, 'svc__C': 10, 'svc__gamma': 'scale'}

Report for Best Model Found for poly-type kernelization:

```
-----
              precision    recall  f1-score   support

         0.0         0.97         0.88         0.92         41
         1.0         0.94         0.99         0.96         73

 accuracy                   0.95         114
 macro avg         0.95         0.93         0.94         114
 weighted avg         0.95         0.95         0.95         114
```

Confusion Matrix:

```
[[36  5]
 [ 1 72]]
```

```

In [8]: #Plot the results
w = 0.15
fig = plt.subplots(figsize = (12,8))

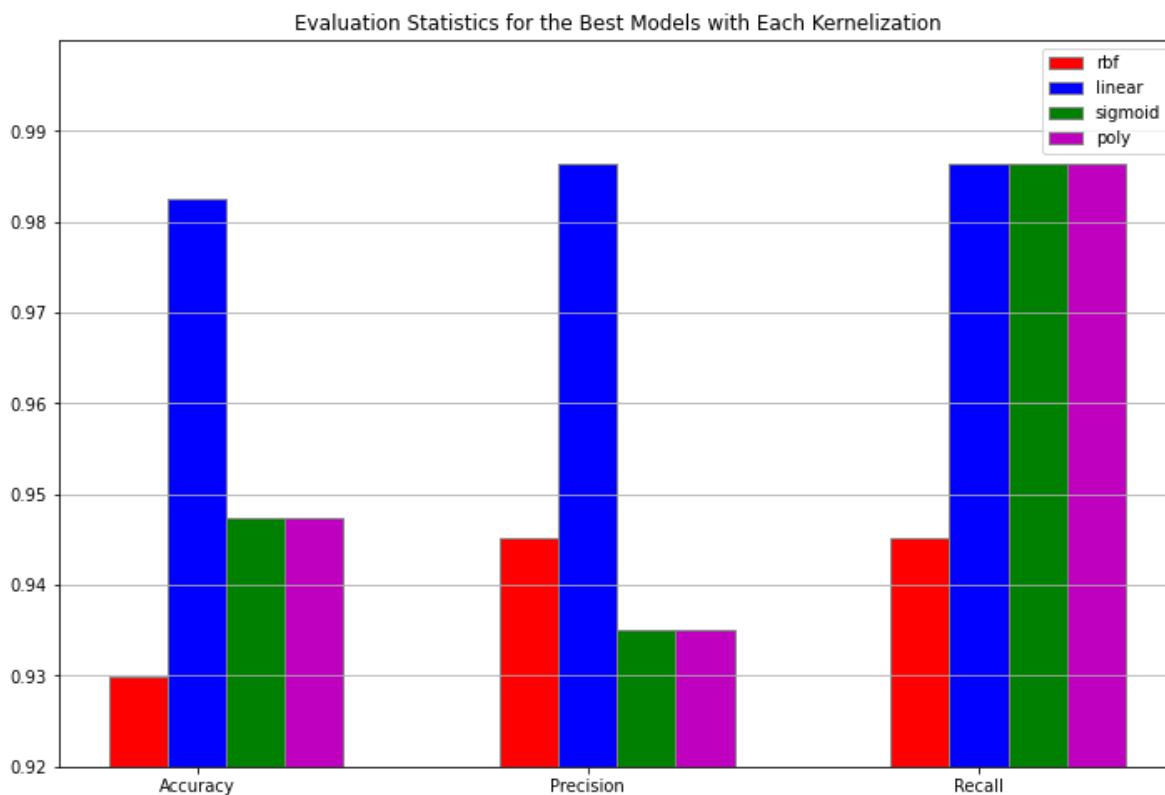
rbf = [accuracyLog['rbf'],precisionLog['rbf'],recallLog['rbf']]
linear = [accuracyLog['linear'],precisionLog['linear'],recallLog['linear']]
sigmoid = [accuracyLog['sigmoid'],precisionLog['sigmoid'],recallLog['sigmoid']]
poly = [accuracyLog['poly'],precisionLog['poly'],recallLog['poly']]

br1 = np.arange(len(rbf))
br2 = [wx + w for wx in br1]
br3 = [wx + w for wx in br2]
br4 = [wx + w for wx in br3]

plt.bar(br1, rbf, color='r', width=w, edgecolor='grey', label='rbf')
plt.bar(br2, linear, color='b', width=w, edgecolor='grey', label='linear')
plt.bar(br3, sigmoid, color='g', width=w, edgecolor='grey', label='sigmoid')
plt.bar(br4, poly, color='m', width=w, edgecolor='grey', label='poly')

plt.xticks([wx + w for wx in range(len(rbf))], ['Accuracy', 'Precision', 'Recall'])
yscale = 100
ylims = [int(yscale*min(accuracyLog.values())),yscale]
ytick = range(ylims[0],ylims[1])
plt.yticks([i/yscale for i in ytick])
plt.legend()
plt.title("Evaluation Statistics for the Best Models with Each Kernelization")
plt.ylim((ylims[0]/yscale,ylims[1]/yscale))
plt.grid(axis='y')
plt.show()

```



```

In [10]: #Best model Analysis
model = modelLog['linear']

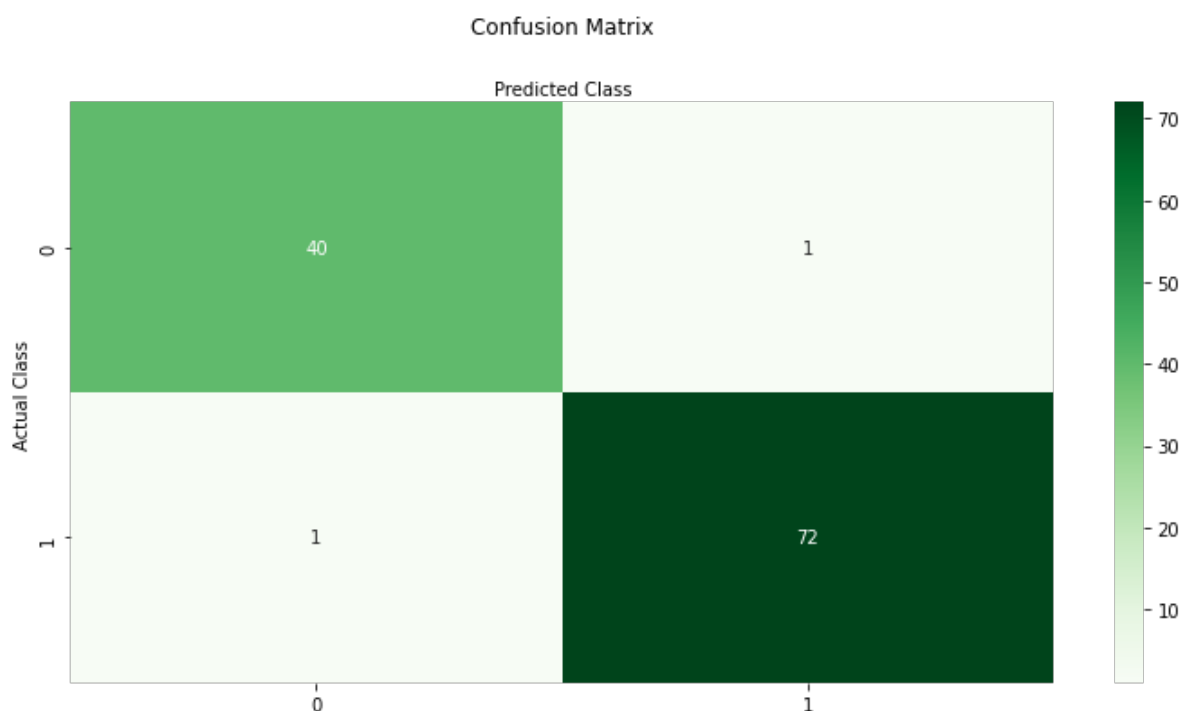
print("Classification Report")
print("-----")
print(classification_report(yv,model.predict(xv)))

#Analyze using the Confusion Matrix
classes = ['Benign','Malignant']
figure, axis = plt.subplots()
ticks = np.arange(len(classes))
plt.xticks(ticks, classes)
plt.yticks(ticks, classes)
sns.heatmap(pd.DataFrame(confusion_matrix(yv,model.predict(xv))),
            annot=True, cmap="Greens", fmt='g')
axis.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1)
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class');

```

Classification Report

	precision	recall	f1-score	support
0.0	0.98	0.98	0.98	41
1.0	0.99	0.99	0.99	73
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114



In []: