Mini project 2

IEE 520

Submitted by:

Sanket Bhale 1214617222
Indraneel Phirke 1214462080

Question 1.

Read the material on a nearest neighbor classifier. Select a dataset of you own choice of sufficient size, at least 500 rows, (prefer over a thousand rows), and at least 10 predictor attributes (prefer at least 20), and a target attribute. The target should be categorical.

Build a K nearest neighbor classifier for your data in Python, and select K through the following steps:

- Hold back 30% of your data for testing.
- For the remaining data, use 5 fold cross validation to select K.
- In each fold, use the training data to predict the test data for each value of K. Consider odd values of K from 1 to at least 25.

(a) Provide your code.

(b) Plot the test error rate over each fold (5 points) versus K. Comment on the value of K selected and why you selected the value. Consider error rate and complexity.

(c) Must the best (minimum error rate) value for K be the same in each fold for any data set? Are the best values selected for K the same in each fold for your data the same?

(d) Extend your code, as needed, to estimate the generalization error rate for your choice of K.

a) Code is answered in this section, answers to question b,c and d are provided following this section.

About the Dataset:

We found our data from the Center for Machine Learning and Intelligent Systems (machine learning repository[1]). Our data set has been extracted from 800 images of the "Avila Bible", a 12th-century giant Latin copy of the Bible. The prediction task consists in associating each pattern to a copyist. Therefore, we are trying to classify the data into different letters (A,B,C,D,E,F,G,H,I,W,X,Y). The data set consisted of the training set and test set, in which we merged and combined so we have 20867 instances. **As there is too much data, we randomly pick 2000 out of them.** Also, we have 10 attributes and all of the attribute characteristics are real numbers. We are trying to train our model to be able to classify the data using the attributes given by implementing the Nearest Neighbour Classifier.

Attributes Information:

F1: intercolumnar distance
F2: upper margin
F3: lower margin
F4: exploitation
F5: row number

[1] https://archive.ics.uci.edu/ml/datasets

F6: modular ratio
F7: interlinear spacing
F8: weight
F9: peak number
F10: modular ratio/ interlinear spacing
Target Value Classes: A, B, C, D, E, F, G, H, I, W, X, Y

Summary of Data:

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	-0.004015	0.005610	-0.018357	0.011850	-0.006059	0.000200	-0.051972	0.004971	-0.021343	-0.010651
std	0.937483	0.934714	1.013025	1.010174	0.993026	1.002806	1.066574	0.988453	1.012788	1.035888
min	-3.498799	-2.426761	-3.210528	-5.440122	-4.922215	-7.450257	-11.935457	-4.011262	-4.426048	-6.719324
25%	-0.128929	-0.259834	0.050694	-0.542563	0.172340	-0.598658	-0.044076	-0.515698	-0.403638	-0.510161
50%	0.056229	-0.063555	0.207175	0.112964	0.261718	-0.038073	0.220177	0.104134	0.032902	-0.027021
75%	0.204355	0.211237	0.349432	0.651257	0.261718	0.564038	0.446679	0.639509	0.469443	0.539625
max	9.943651	19.470188	6.260173	3.987152	1.066121	4.508898	4.901228	4.510897	2.963961	3.744023

Class Label Frequency Count:

Before fitting the data to model and train, we compute the frequency count of each class by using the following code:

- `pd.Series(data.iloc[:, -1]).value_counts().sort_index()`

Note that the target variable of this dataset is categorical and they are also letters. So, we encoded the letter labels before fitting the data to the model.

```
A    842
C     24
D     57
E    191
F    384
G     93
H     82
I    164
W     10
X    105
Y     48
Name: Class: A, B, C, D, E, F, G, H, I, W, X, Y, dtype: int64
```

From the results above, we can know that the target values are highly imbalanced. Class W has only 10 and Class A 842 observations.

Model training:

As stated by the problem description, we first split our data into training and test sets which compose 70% and 30% of all the data, then we used 5-fold cross-validation to further split the training and validation sets.

Python code:

```
from google.colab import drive
drive.mount('/content/drive')
```

Library needed for k Nearest Neighbour

```
# For compatibility with Python 2
from __future__ import print_function

# To load datasets
from sklearn import datasets

# To import the classifier (K-Nearest Neighbors Classifier and Regressor)
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

# To measure accuracy
from sklearn import metrics

from sklearn.model_selection import KFold

# To support plots
from ipywidgets import interact
import ipywidgets as widgets
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

import numpy as np

# To display all the plots inline
%matplotlib inline

# To splite the data
from sklearn.model_selection import train_test_split, cross_val_score

seed = 2357

# To increase quality of figures
plt.rcParams["figure.figsize"] = (20, 10)
```

Loading Data

```python
#import data
import pandas as pd
path = "/content/drive/Shared drives/IEE520/Mini Project 2/Avila_2000_Random_rows.csv"
data = pd.read_csv(path)
```

Summary of Data

```python
data.describe()
```

	Intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ratio	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	-0.004015	0.005610	-0.018357	0.011850	-0.006059	0.000200	-0.051972	0.004971	-0.021343	-0.010651
std	0.937483	0.934714	1.013025	1.010174	0.993026	1.002806	1.066574	0.988453	1.012788	1.035888
min	-3.498799	-2.426761	-3.210528	-5.440122	-4.922215	-7.450257	-11.935457	-4.011262	-4.426048	-6.719324
25%	-0.128929	-0.259834	0.050694	-0.542563	0.172340	-0.598658	-0.044076	-0.515698	-0.403638	-0.510161
50%	0.056229	-0.063555	0.207175	0.112964	0.261718	-0.038073	0.220177	0.104134	0.032902	-0.027021
75%	0.204355	0.211237	0.349432	0.651257	0.261718	0.564038	0.446679	0.639509	0.469443	0.539625
max	9.943651	19.470188	6.260173	3.987152	1.066121	4.508898	4.901228	4.510897	2.963961	3.744023

```python
pd.Series(data.iloc[:, -1]).value_counts().sort_index()
```

```
A     842

C      24

D      57

E     191

F     384

G      93

H      82

I     164

W      10

X     105

Y      48

Name: Class: A, B, C, D, E, F, G, H, I, W, X, Y, dtype: int64
```

Preprocessing data

Encoding class labels to number

```python
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data = data.apply(le.fit_transform)
data = data.to_numpy()
```

Extract feature and class label

```python
X = data[:, 0:-1]
y = data[:, -1]
```

Standardizing Features

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

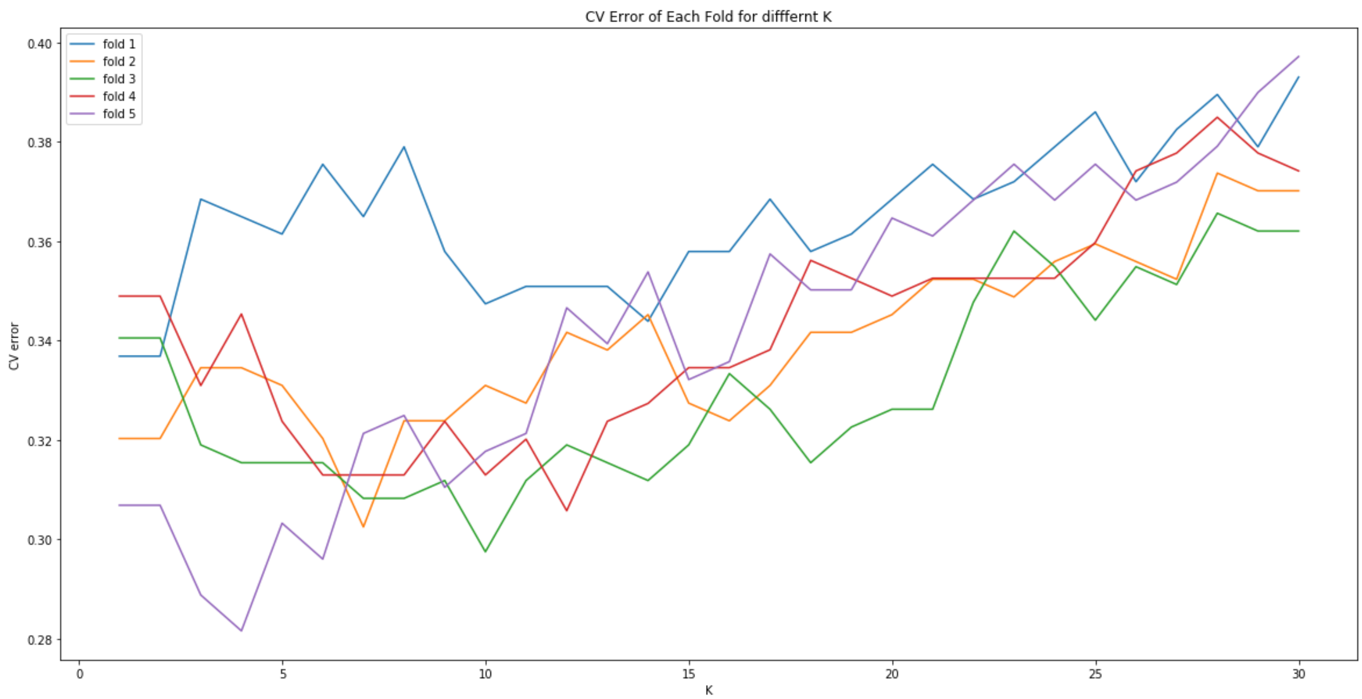
Split to train and test set

```python
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=seed)
```

Choosing K

Choose by analyzing the cross-validation error (1- accuracy score)

```python
cv_errors = []
n_fold=5
max_num_neighbour = 30
neighbors = list(np.arange(1, max_num_neighbour+1))

# perform 5-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    scores = cross_val_score(knn, X_train, y_train, cv=n_fold, scoring='accuracy')
    cv_errors.append(1-scores)

cv_errors = zip(*cv_errors)
plt.figure()

folds = np.arange(1, n_fold+1)
for i in folds:
  plt.plot(neighbors, cv_errors[i-1], label=('fold '+str(i)))

plt.title('CV Error of Each Fold for difffernt K')
plt.xlabel('K')
plt.ylabel('CV error')
plt.legend()

```



Select k that Attains Min CV Error in Each Fold

```
for i in np.arange(0, n_fold):
  min_error = min(cv_errors[i])
  min_k = [k+1 for k, v in enumerate(cv_errors[i]) if v == min_error]

  print("For fold", str(i+1), ', the min test error is', str(min_error), 'attaining at index/indicies', str(min_k))
```

For fold 1 , the min test error is 0.33684210526315794 attaining at index/indicies [1, 2]
For fold 2 , the min test error is 0.302491103202847 attaining at index/indicies [7]
For fold 3 , the min test error is 0.29749103942652333 attaining at index/indicies [10]
For fold 4 , the min test error is 0.3057553956834532 attaining at index/indicies [12]
For fold 5 , the min test error is 0.2815884476534296 attaining at index/indicies [4]

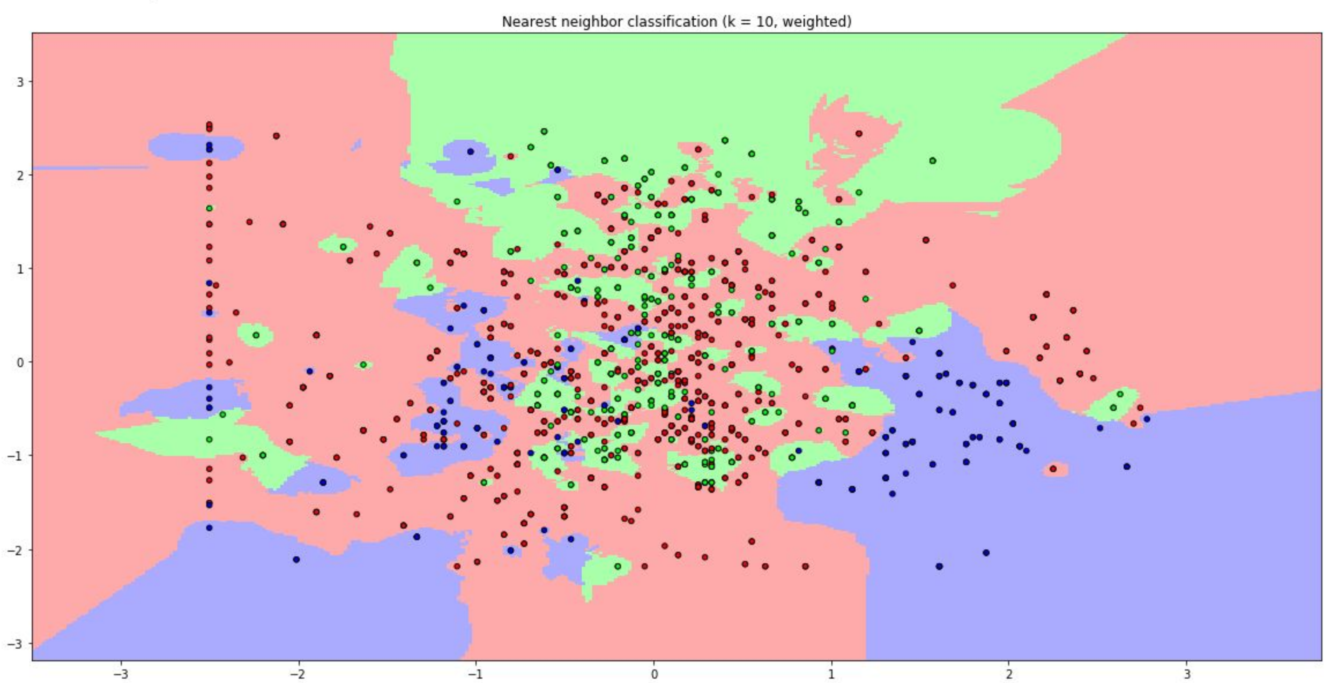
Visualization (for Model Complexity Analysis)

```
# Here we use closure to store the related variables
def create_plot_knn_classification(_X, _y):
    X, y = _X, _y
    def plot_knn(k=3, weighted=True):
        h = .02  # step size in the mesh
        cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
        cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
        if weighted:
```

```
            clf = KNeighborsClassifier(k, weights='distance')
        else:
            clf = KNeighborsClassifier(k, weights='uniform')
        clf.fit(X, y)
        x1_min = X[:, 0].min() - 1
        x1_max = X[:, 0].max() + 1
        x2_min = X[:, 1].min() - 1
        x2_max = X[:, 1].max() + 1
        xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, h),
                               np.arange(x2_min, x2_max, h))
        Z = clf.predict(np.c_[xx1.ravel(), xx2.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(xx1.shape)
        plt.figure()
        plt.pcolormesh(xx1, xx2, Z, cmap=cmap_light)

        # Plot also the training points
        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                    edgecolor='k', s=20)
        plt.xlim(xx1.min(), xx1.max())
        plt.ylim(xx2.min(), xx2.max())
        plt.title("Nearest neighbor classification (k = %i, %s)"
                  % (k, 'weighted' if weighted else 'unweighted'))
        plt.show()
    return plot_knn
interact(create_plot_knn_classification(X_train[:, :2], y_train), k=(1, max_num_neighbour, 1))
```

k 10

weighted



<function __main__.plot_knn>

Hypertuning model parameters using GridSearchCV

```
from sklearn.model_selection import GridSearchCV #create new a knn model

knn2 = KNeighborsClassifier(weights='distance')#create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': neighbors}#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn2, param_grid, cv=n_fold)#fit model to data
knn_gscv.fit(X_train, y_train)
```

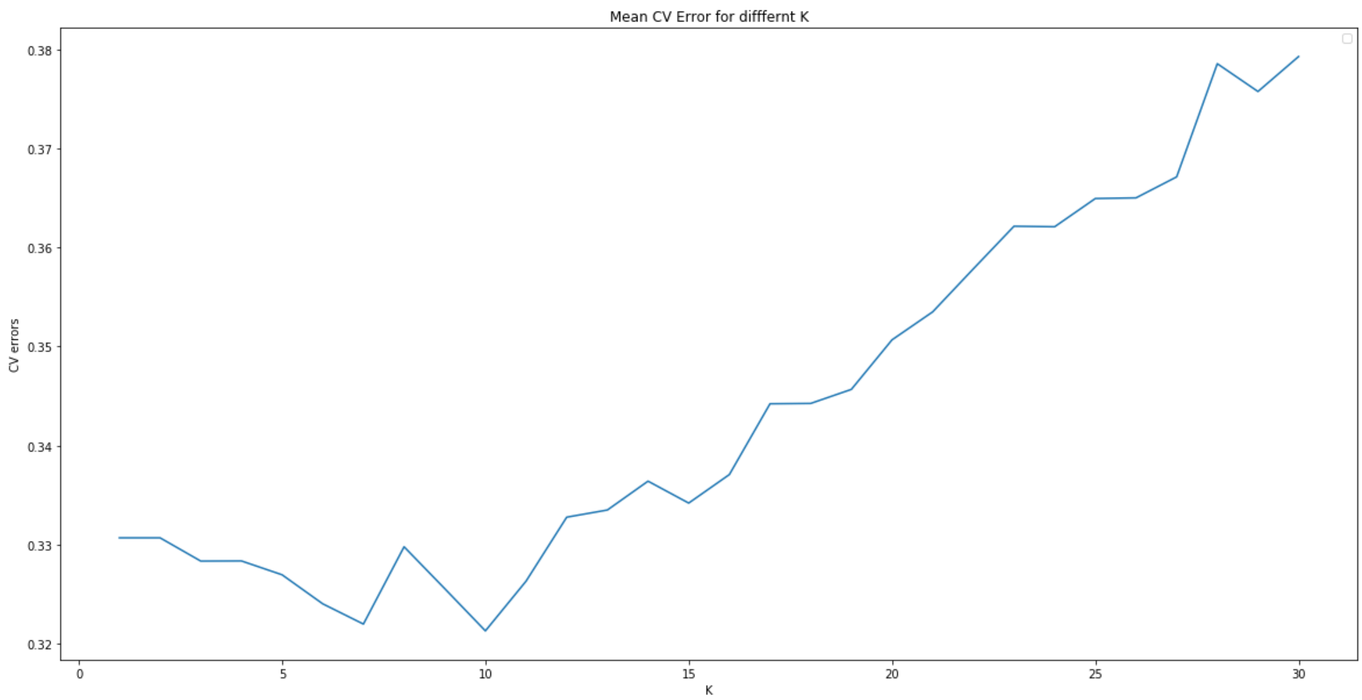
/usr/local/lib/python2.7/dist-packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
 DeprecationWarning)
GridSearchCV(cv=5, error_score='raise-deprecating',
 estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
 metric_params=None, n_jobs=None, n_neighbors=5, p=2,
 weights='distance'),
 fit_params=None, iid='warn', n_jobs=None,
 param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]},
 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
 scoring=None, verbose=0)

```
print("Using GridSearchCV, the optimum value of k is", str(knn_gscv.best_params_.values()))
```

```
Using GridSearchCV, the optimum value of k is [10]
```

Select K by Mean CV Error

```
mean_cv_errors =np.mean(cv_errors, axis = 0)

plt.plot(neighbors, mean_cv_errors)
plt.title('Mean CV Error for difffernt K')
plt.xlabel('K')
plt.ylabel('CV errors')
plt.legend()
```




```
k_mse = [k+1 for k, v in enumerate(mean_cv_errors) if v == min(mean_cv_errors)]

print('The minimum mean test error is', str(min(mean_cv_errors)), 'attaining at index/indicies', str(k_mse))
```

The minimum mean test error is 0.3212918971090744 attaining at index/indicies [10]

Model evaluation
Generalization Error

From the above results, we therefore pick K = 10

```
k=10
model = KNeighborsClassifier(k, weights='distance')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
test_error = 1- metrics.accuracy_score(yhat, y_test)
print(test_error)
```

0.3466666666666667

Confusion Matrix for Test Data

```
!pip install pandas_ml
from pandas_ml import ConfusionMatrix

cm = ConfusionMatrix(y_test, yhat)
print(cm)

cm.print_stats()
```

```
ax = cm.plot(backend='seaborn', annot=True, fmt='g')
ax.set_title('Test Confusion Matrix')
plt.show()

```

/usr/local/lib/python2.7/dist-packages/pandas/core/indexing.py:1494: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
 return self._getitem_tuple(key)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/stats.py:60: FutureWarning: supplying multiple axes to axis is deprecated and will be removed in a future version.
 num = df[df > 1].dropna(axis=[0, 1], thresh=1).applymap(lambda n: choose(n, 2)).sum().sum() - np.float64(nis2 * njs2) / n2
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:330: RuntimeWarning: divide by zero encountered in double_scalars
 return(np.float64(self.TPR) / self.FPR)

Predicted	0	1	2	3	4	5	6	7	8	9	10	__all__
Actual												
0	222	0	0	2	31	2	5	0	0	0	0	262
1	4	1	0	2	0	0	0	0	0	0	0	7
2	3	0	6	3	3	0	1	0	0	0	0	16
3	13	0	1	24	7	4	1	1	0	2	0	53
4	61	0	0	0	52	4	1	0	0	0	0	118
5	9	0	0	1	5	9	0	0	0	0	0	24
6	12	0	0	1	2	3	9	0	0	0	0	27
7	4	0	0	0	0	0	0	39	0	0	0	43
8	3	0	0	2	0	0	0	0	0	0	0	5
9	2	0	0	1	1	0	0	0	0	22	2	28
10	6	0	0	0	0	0	0	1	0	2	8	17
__all__	339	1	7	36	101	22	17	41	0	26	10	600

/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:236: RuntimeWarning: invalid value encountered in double_scalars
 return(np.float64(self.TP) / self.PositiveTest)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:267: RuntimeWarning: invalid value encountered in double_scalars
 return(np.float64(self.FP) / self.PositiveTest)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:302: RuntimeWarning: invalid value encountered in true_divide
 * (self.TN + self.FP) * (self.TN + self.FN)))
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:330: RuntimeWarning: invalid value encountered in double_scalars
 return(np.float64(self.TPR) / self.FPR)
Confusion Matrix:

Predicted	0	1	2	3	4	5	6	7	8	9	10	__all__
Actual												
0	222	0	0	2	31	2	5	0	0	0	0	262

1	4	1	0	2	0	0	0	0	0	0	0	7
2	3	0	6	3	3	0	1	0	0	0	0	16
3	13	0	1	24	7	4	1	1	0	2	0	53
4	61	0	0	0	52	4	1	0	0	0	0	118
5	9	0	0	1	5	9	0	0	0	0	0	24
6	12	0	0	1	2	3	9	0	0	0	0	27
7	4	0	0	0	0	0	0	39	0	0	0	43
8	3	0	0	2	0	0	0	0	0	0	0	5
9	2	0	0	1	1	0	0	0	0	22	2	28
10	6	0	0	0	0	0	0	1	0	2	8	17
__all__	339	1	7	36	101	22	17	41	0	26	10	600

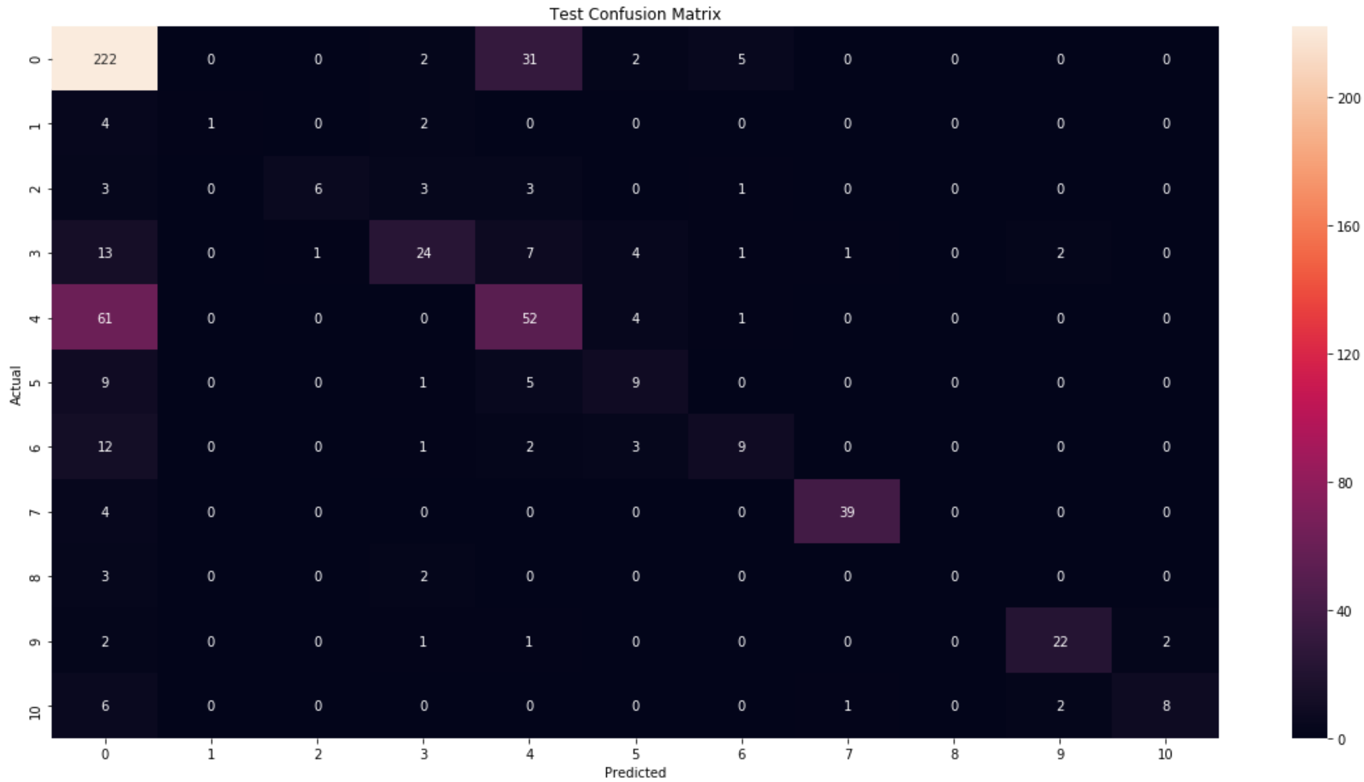
Overall Statistics:

Accuracy: 0.6533333333333333
95% CI: (0.6137375242022135, 0.6914110734506197)
No Information Rate: ToDo
P-Value [Acc > NIR]: 6.303856518830028e-06
Kappa: 0.5078650887853968
Mcnemar's Test P-Value: ToDo

Class Statistics:

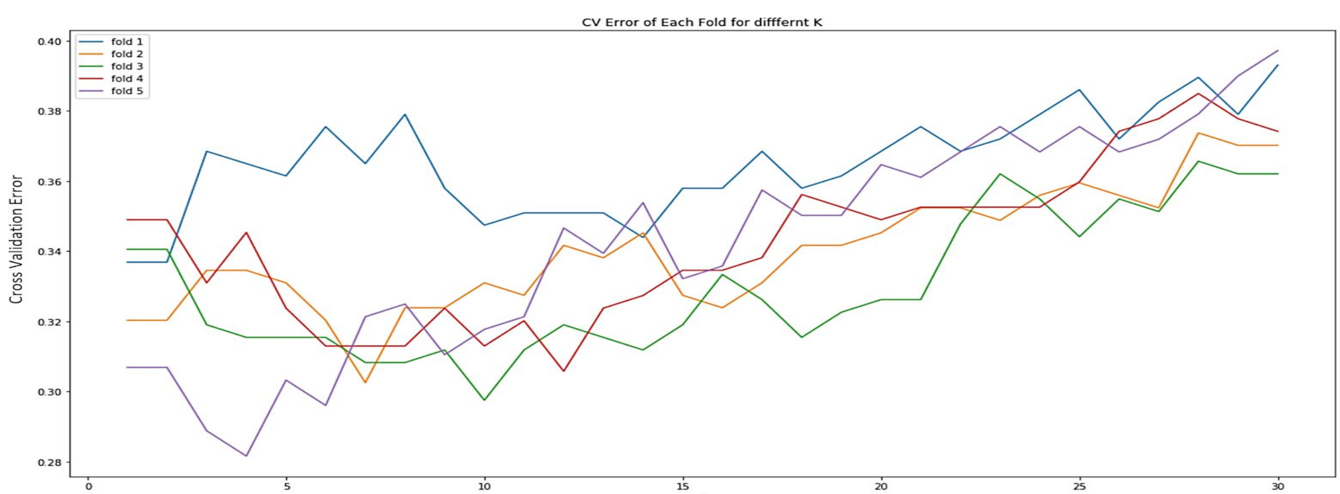
Classes	0	...	10
Population	600	...	600
P: Condition positive	262	...	17
N: Condition negative	338	...	583
Test outcome positive	339	...	10
Test outcome negative	261	...	590
TP: True Positive	222	...	8
TN: True Negative	221	...	581
FP: False Positive	117	...	2
FN: False Negative	40	...	9
TPR: (Sensitivity, hit rate, recall)	0.847328	...	0.470588
TNR=SPC: (Specificity)	0.653846	...	0.996569
PPV: Pos Pred Value (Precision)	0.654867	...	0.8
NPV: Neg Pred Value	0.846743	...	0.984746
FPR: False-out	0.346154	...	0.00343053
FDR: False Discovery Rate	0.345133	...	0.2
FNR: Miss Rate	0.152672	...	0.529412
ACC: Accuracy	0.738333	...	0.981667
F1 score	0.738769	...	0.592593
MCC: Matthews correlation coefficient	0.501392	...	0.605475
Informedness	0.501174	...	0.467158
Markedness	0.501611	...	0.784746
Prevalence	0.436667	...	0.0283333
LR+: Positive likelihood ratio	2.44784	...	137.176

LR-: Negative likelihood ratio	0.233498	...	0.531234
DOR: Diagnostic odds ratio	10.4833	...	258.222
FOR: False omission rate	0.153257	...	0.0152542



b and c)

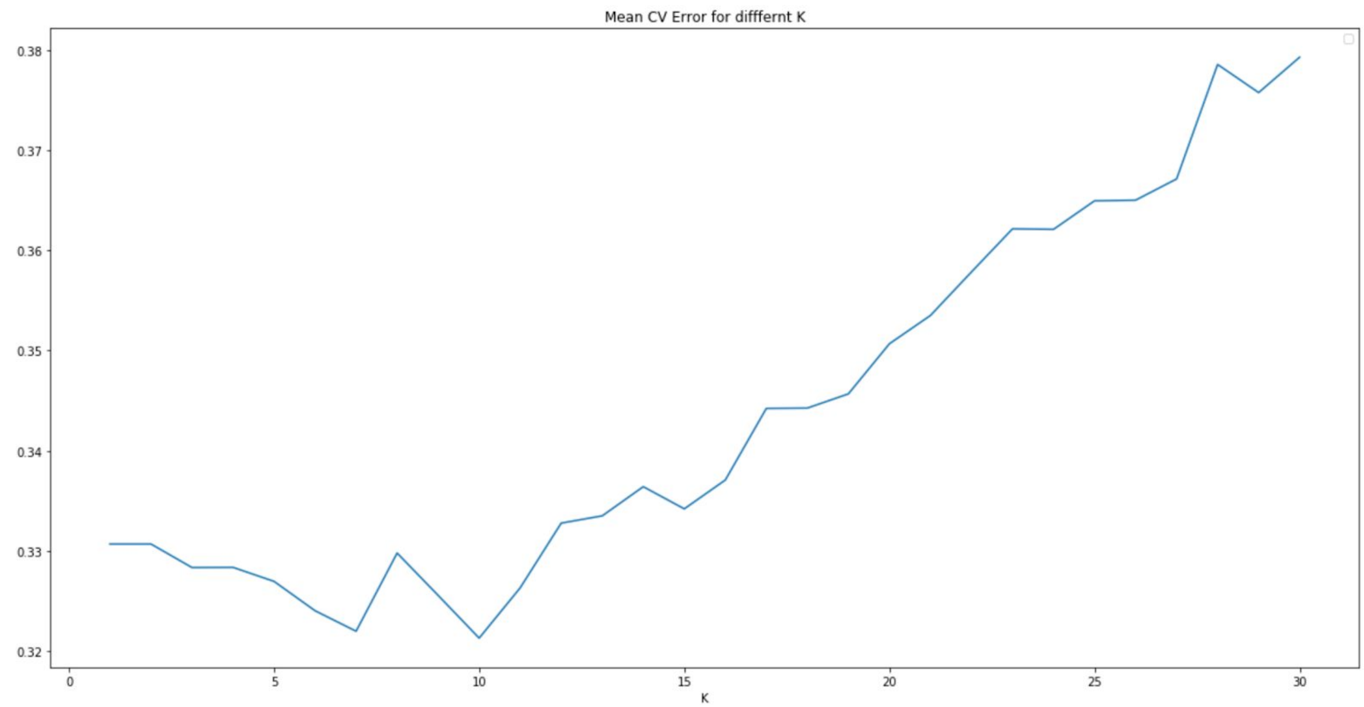
The test error rate over each fold versus K are plotted in this section. The cross-validation error in general for all folds increase with increase in number of K. We cant select directly the value of k as the minimum error differs for each combination of fold and value of k. But for the larger section after k=7, the test error for fold 3 is observed to be minimum among all folds. To choose the final values we make use of grid search.



The k values that Attain Minimum Cross Validation Error in Each Fold:

For fold 1 , the min test error is 0.33684 attaining at index/indicies [1, 2]
For fold 2 , the min test error is 0.30249 attaining at index/indicies [7]
For fold 3 , the min test error is 0.29749 attaining at index/indicies [10]
For fold 4 , the min test error is 0.30575 attaining at index/indicies [12]
For fold 5 , the min test error is 0.28158 attaining at index/indicies [4]

As we can see the best value for K is not the same in each fold for any data set. Using GridSearch Cross Validation, the optimum value of k is 10. Also the minimum mean test error is 0.3213 attaining index 10 according to the following chart. So the best value selected for K is not the same in each fold for our data.



Therefore, from the results above, we pick K = 10 as the number of neighbors for our nearest neighbor classifier.

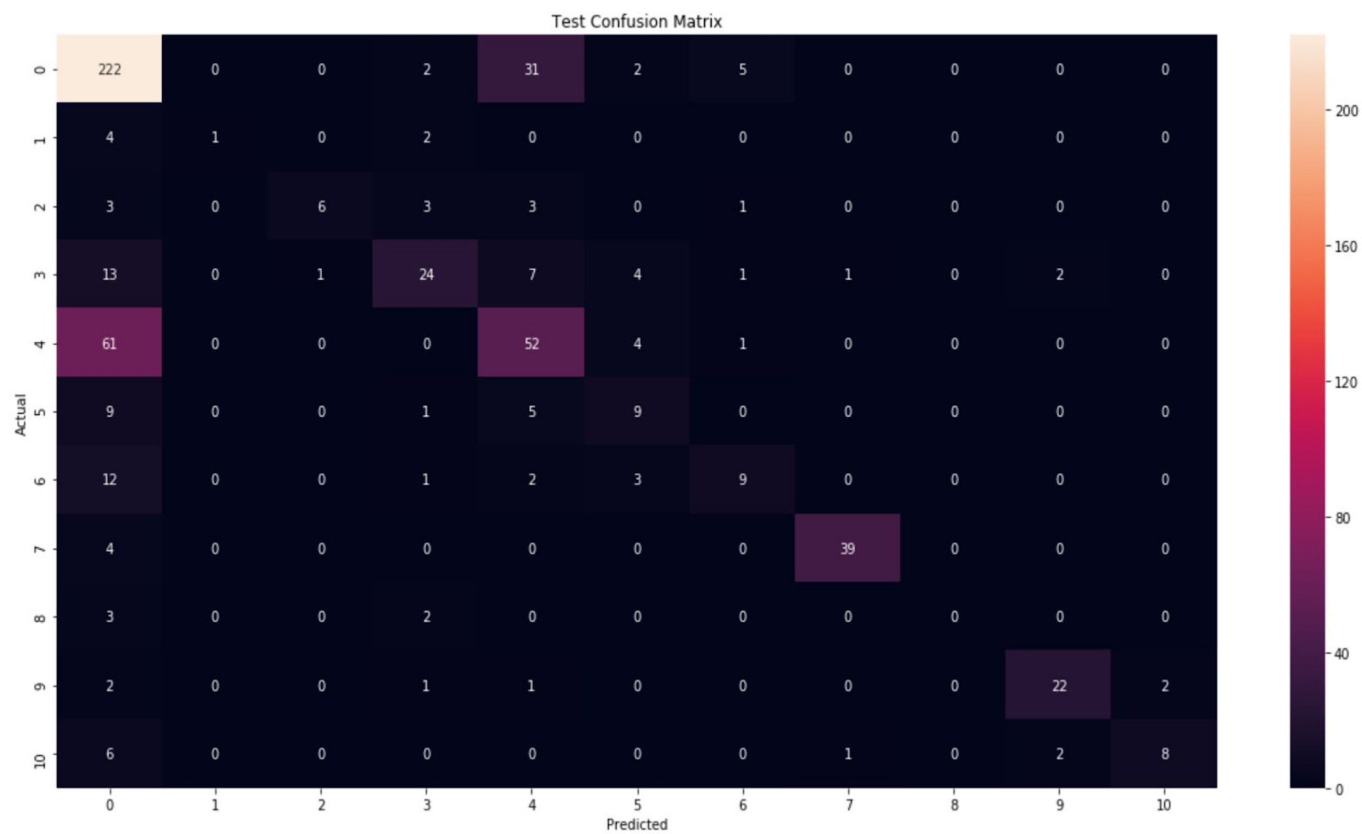
d)

Generalization Error

```
k=10
model = KNeighborsClassifier(k, weights='distance')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
test_error = 1- metrics.accuracy_score(yhat, y_test)
print(test_error)
```

The estimation of the generalization error rate for K=10 is 0.34667.

Confusion Matrix for Test Data



Question 2.

Suppose that a SVM with a Gaussian kernel

$$K(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right) \tag{1}$$

with $\sigma^2 = 2$, for a problem with two input variables x_1 and x_2 generates the following support vectors, class variable y and α values. Also assume that the estimate of $w_0 = 0.7$.

Support vector	x_1	x_2	α	y
1	-0.6	0.7	0.2	-1
2	0.4	-0.2	0.1	1
3	0.2	0.4	0.3	1

Table 1: Data for function estimation

(a) Write the expression for the estimated classifier $\hat{f}(x)$.

(b) What class is the point (-0.1, 0.4) assigned to?

(c) Explain why are polynomial variables are not directly constructed for a problem such as this. Be brief and clear.

a) $\hat{f}(X = (x, y)) = \hat{w}_0 + \sum_{i=1}^{3} \alpha_i y_i K(X_i, X) = \hat{w}_0 + \alpha_1 y_1 K(X_1, X) + \alpha_2 y_2 K(X_2, X) + \alpha_3 y_3 K(X_3, X) =$

$\hat{w}_0 + \alpha_1 y_1 exp(-\frac{(x_{11}-x)^2+(x_{21}-y)^2}{2*\sigma^2}) + \alpha_2 y_2 exp(-\frac{(x_{21}-x)^2+(x_{22}-y)^2}{2*\sigma^2}) + \alpha_3 y_3 exp(-\frac{(x_{31}-x)^2+(x_{32}-y)^2}{2*\sigma^2}) =$

$0.7 - 0.2 * exp(-\frac{(-0.6-x)^2+(0.7-y)^2}{4}) + 0.1 * exp(-\frac{(0.4-x)^2+(-0.2-y)^2}{4}) + 0.3 * exp(-\frac{(0.2-x)^2+(0.4-y)^2}{4})$

b) $\hat{f}(X = (-0.1, 0.4)) = 0.7 + 0.2 * (-1) * exp(-\frac{(-0.6+0.1)^2+(0.7-0.4)^2}{2*2}) +$

$0.1 * 1 * exp(-\frac{(0.4+0.1)^2+(-0.2-0.4)^2}{2*2}) + 0.3 * 1 * exp(-\frac{(0.2+0.1)^2+(0.4-0.4)^2}{2*2}) = 0.8955$

Since the value is close to 1 and far from -1, we assign point (-0.1, 0.4) to class 1.

c) It is because of the fact that Gaussian Kernel implies polynomial terms of all degrees which means an infinite set of predictors. The reason why polynomial variables are not directly used for this type of problem is that the kernel function makes it feasible to compute these models. For example with 50 inputs, degree-2 polynomial contains 1325 terms which are computationally expensive.

Question 3:

For the data you selected for the first question, use Python to construct a SVM classifier.

- Hold back the same 30% of test data as in the first question.
- Select parameters for a SVM based on the remaining data and attempt to build the best possible model.
  (a) When your SVM model is complete, compare the error rate on the 30% test data from the SVM and best nearest neighbor classifier you obtained in the first question. Compute confusion matrices for the test data from each model and comments on any differences in performance.
  (b) Which model do you prefer and why? Be brief and clear.

Code is answered in the following section. Answers to question a and b are answered in later section.

Model training:

As stated by the problem description, we first split our data into training and test sets which compose 70% and 30% of all the data, then we try to build the best possible model.

Python code:

Split to train and test set

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=seed)
```

Choosing Best SVM

```
# Here we use closure to store the related variables
def create_plot_svm_classification_kernels(_X, _y):
    X, y = _X, _y
    def plot_svc_kernel(C=1, kernel='linear', expand=3.1, intensity=0.5):
        if kernel.startswith('poly'):
            clf = SVC(kernel='poly', C=C, gamma='auto', degree=int(kernel[4:]))
        else:
            clf = SVC(kernel=kernel, C=C, gamma='auto')
        clf.fit(X, y)
        fig, ax = plt.subplots()
        ax.plot((np.min(X[:, 0])-expand, np.max(X[:, 0])+expand), (np.min(X[:,
1])-expand, np.max(X[:, 1])+expand), alpha=0.0)
        xlim = ax.get_xlim()
        ylim = ax.get_ylim()
        xx = np.linspace(xlim[0], xlim[1], 400)
        yy = np.linspace(ylim[0], ylim[1], 200)
        YY, XX = np.meshgrid(yy, xx)
        xy = np.vstack([XX.ravel(), YY.ravel()]).T
        Z = clf.decision_function(xy).reshape(XX.shape)
        v = max(-np.min(Z), np.max(Z))
```

```
        cf = ax.contourf(XX, YY, Z, 200, cmap='coolwarm', norm =
mpl.colors.Normalize(vmin=-v, vmax=v), alpha=intensity)
        ax.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.bwr)
        ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.8,
                   linestyles=['--', '-', '--'], linewidths=[2, 5, 2])
        ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
                   linewidth=1, facecolors='none', edgecolors='k')
        plt.xlabel('Sepal width')
        plt.ylabel('Petal length')
        plt.title('Support Vector Machines Classifier: C=%s, %s kernel.' % (str(C),
kernel))
        plt.show()
    return plot_svc_kernel
```

```
kernels = ['linear'] + ['poly'+str(x) for x in range(1, 9)] + ['rbf']
c_widget = widgets.FloatLogSlider(
    value=1,
    base=10,
    min=-4,
    max=4,
    step=0.5,
    continuous_update=False,
    description='C')
expand_widget = widgets.FloatSlider(
    value=3.1,
    min=0,
    max=10,
    step=0.1,
    continuous_update=False,
    description='expand:')
intensity_widget = widgets.FloatSlider(
    value=0.5,
    min=0.1,
    max=0.9,
    step=0.1,
    continuous_update=False,
    description='contour intensity:')


req_idx = [i for i, e in enumerate(y_train) if e == 3 or e ==4]
y_trim = y_train[req_idx]
X_trim = X_train[req_idx,:2]

interact(create_plot_svm_classification_kernels(X_trim, y_trim), C=c_widget,
kernel=kernels, expand=expand_widget, intensity=intensity_widget)
```

```
# Here we use closure to store the related variables
def create_plot_svm_classification_rbf(_X, _y):
    X, y = _X, _y
    def plot_svc_rbf(C=1, gamma=1, expand=0, intensity=0.1):
        clf = SVC(kernel='rbf', C=C, gamma=gamma)
        clf.fit(X, y)
```

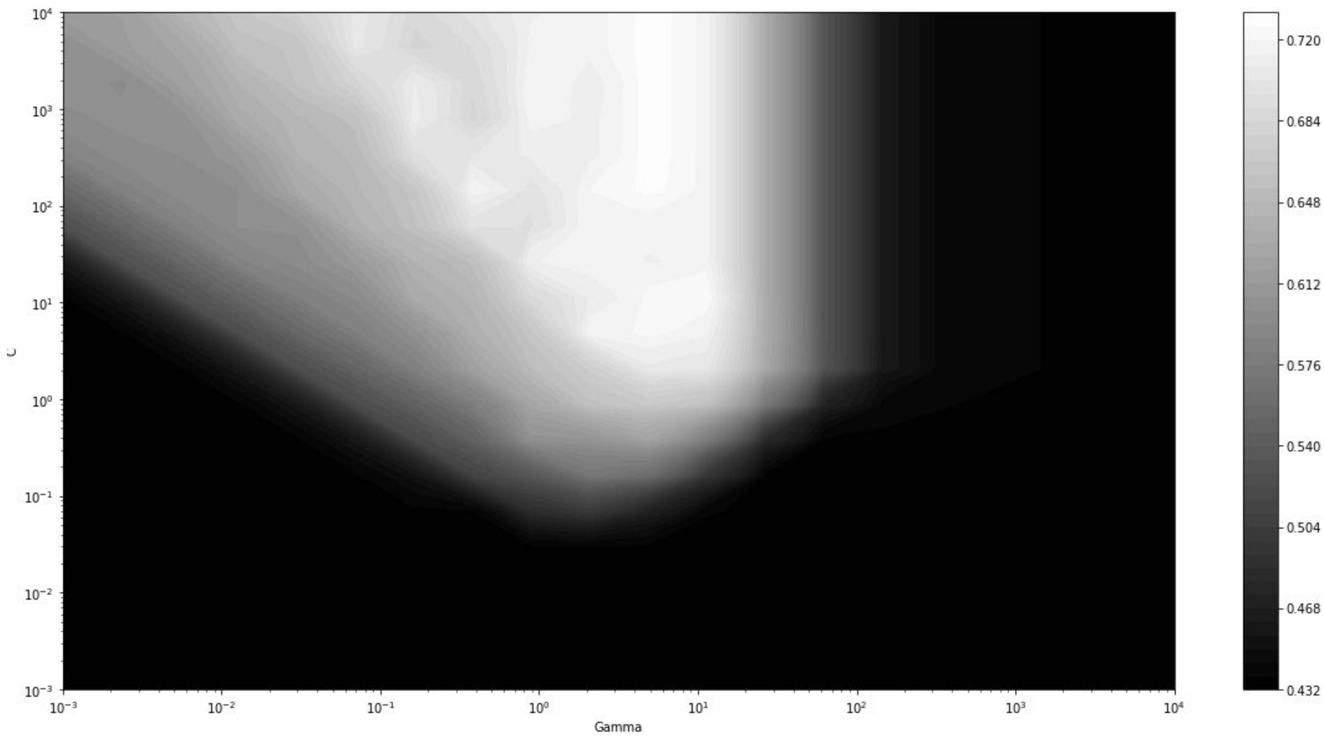
```
•         fig, ax = plt.subplots()
•         ax.plot((np.min(X[:, 0])-expand, np.max(X[:, 0])+expand), (np.min(X[:,
  1])-expand, np.max(X[:, 1])+expand), alpha=0.0)
•         xlim = ax.get_xlim()
•         ylim = ax.get_ylim()
•         xx = np.linspace(xlim[0], xlim[1], 400)
•         yy = np.linspace(ylim[0], ylim[1], 200)
•         YY, XX = np.meshgrid(yy, xx)
•         xy = np.vstack([XX.ravel(), YY.ravel()]).T
•         Z = clf.decision_function(xy).reshape(XX.shape)
•         v = max(-np.min(Z), np.max(Z))
•         cf = ax.contourf(XX, YY, Z, 100, cmap='coolwarm', norm =
  mpl.colors.Normalize(vmin=-v, vmax=v), alpha=intensity)
•         ax.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.bwr)
•         ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.8,
•                    linestyles=['--', '-', '--'], linewidths=[2, 5, 2])
•         ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
•                    linewidth=1, facecolors='none', edgecolors='k')
•         plt.xlabel('Sepal width')
•         plt.ylabel('Petal length')
•         plt.title('Support Vector Machines Classifier: C=%s, Gamma=%s.' % (str(C),
  str(gamma)))
•         plt.show()
•     return plot_svc_rbf
```

```
• C_widget = widgets.FloatLogSlider(
•     value=10,
•     base=10,
•     min=-4,
•     max=3,
•     step=0.5,
•     continuous_update=False,
•     description='C:')
• gamma_widget = widgets.FloatLogSlider(
•     value=10*math.sqrt(10),
•     base=10,
•     min=-4,
•     max=3,
•     step=0.5,
•     continuous_update=False,
•     description='gamma:')
• expand_widget = widgets.FloatSlider(
•     value=0.2,
•     min=0,
•     max=10,
•     step=0.1,
•     continuous_update=False,
•     description='expand:')
• intensity_widget = widgets.FloatSlider(
•     value=0.7,
•     min=0.1,
•     max=0.9,
```

```
        step=0.1,
        continuous_update=False,
        description='intensity:')
interact(create_plot_svm_classification_rbf(X_train[:,:2], y_train[:,]), C=C_widget,
gamma=gamma_widget, expand=expand_widget, intensity=intensity_widget)
```

Image

```
create_scaler = create_scaler_minmax
scaler = create_scaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

def plot_param_search_rbf(X_train, y_train, X_test, y_test, Cs, gammas):
    def compute_accuracy(C, gamma):
        clf = SVC(kernel='rbf', C=C, gamma=gamma)
        clf.fit(X_train, y_train)
        return clf.score(X_test, y_test)

    Cs = np.power(10, np.linspace(-3, 4, num=20, endpoint=True))
    gammas = np.power(10, np.linspace(-3, 4, num=20, endpoint=True))

    C_mesh, gamma_mesh = np.meshgrid(Cs, gammas)
    Z = np.zeros(C_mesh.shape)
    for i in range(len(gammas)):
        for j in range(len(Cs)):
            Z[i, j] = compute_accuracy(C_mesh[i, j], gamma_mesh[i, j])

    fig, ax = plt.subplots()
    plt.contourf(gamma_mesh, C_mesh, Z, 50, cmap='gray')
    plt.colorbar()
    ax.set_xscale('log')
    ax.set_yscale('log')
    ax.set_xlabel('Gamma')
    ax.set_ylabel('C')
    plt.show()

Cs = np.power(10, np.linspace(-3, 4, num=20, endpoint=True))
gammas = np.power(10, np.linspace(-3, 4, num=20, endpoint=True))
plot_param_search_rbf(X_train, y_train, X_test, y_test, Cs, gammas)
```



Model evaluation

- `!pip install pandas_ml`

Generalization Error

From the above results, we therefore Select gama around 5 and C greater than 150

```
gamma=5
C=200
model =  SVC(kernel='rbf', C=C, gamma=gamma)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
test_error = 1- metrics.accuracy_score(yhat, y_test)
print(test_error)
```

0.2716666666666666

Confusion Matrix for Test Data

```
from pandas_ml import ConfusionMatrix

cm = ConfusionMatrix(y_test, yhat)
print(cm)

cm.print_stats()
ax = cm.plot(backend='seaborn', annot=True, fmt='g')
ax.set_title('Test Confusion Matrix')
plt.show()
```

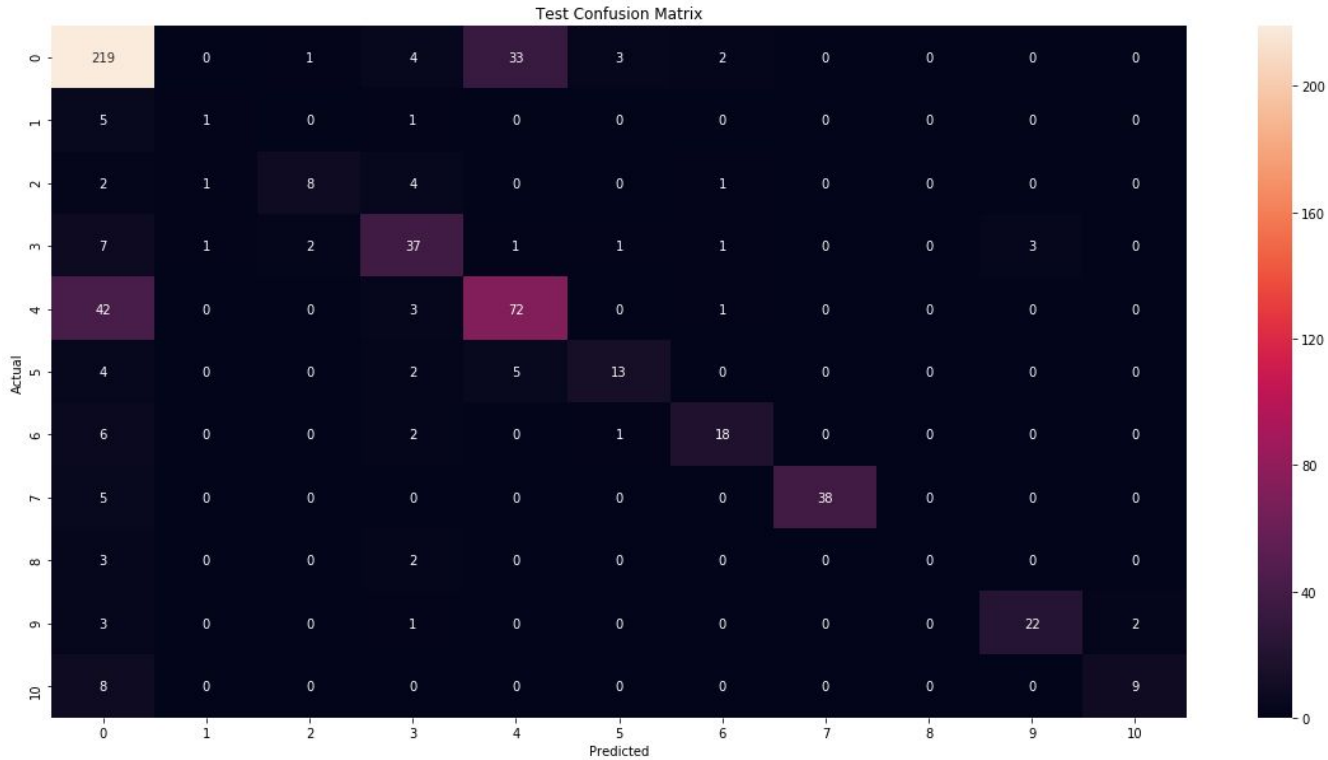
```
/usr/local/lib/python2.7/dist-packages/pandas/core/indexing.py:1494: FutureWarning:
```

```
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
  return self._getitem_tuple(key)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/stats.py:60: FutureWarning:
supplying multiple axes to axis is deprecated and will be removed in a future version.
  num = df[df > 1].dropna(axis=[0, 1], thresh=1).applymap(lambda n: choose(n, 2)).sum().sum()
- np.float64(nis2 * njs2) / n2

Predicted     0  1   2   3  4   5   6   7  8   9  10  __all__
Actual
0             219  0   1   4   33   3   2   0  0   0   0     262
1             5  1   0   1  0   0   0   0  0   0   0         7
2             2  1   8   4  0   0   1   0  0   0   0        16
3             7  1   2  37  1   1   1   0  0   3   0        53
4             42  0   0   3   72   0   1   0  0   0   0      118
5             4  0   0   2  5  13   0   0  0   0   0        24
6             6  0   0   2  0   1  18   0  0   0   0        27
7             5  0   0   0  0   0   0  38  0   0   0        43
8             3  0   0   2  0   0   0   0  0   0   0         5
9             3  0   0   1  0   0   0   0  0  22   2        28
10            8  0   0   0  0   0   0   0  0   0   9        17
__all__       304  3  11  56  111  18  23  38  0  25  11     600

/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:330: RuntimeWarning:
divide by zero encountered in double_scalars
  return(np.float64(self.TPR) / self.FPR)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:236: RuntimeWarning:
invalid value encountered in double_scalars
  return(np.float64(self.TP) / self.PositiveTest)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:267: RuntimeWarning:
invalid value encountered in double_scalars
  return(np.float64(self.FP) / self.PositiveTest)
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:302: RuntimeWarning:
invalid value encountered in true_divide
  * (self.TN + self.FP) * (self.TN + self.FN)))
/usr/local/lib/python2.7/dist-packages/pandas_ml/confusion_matrix/bcm.py:330: RuntimeWarning:
invalid value encountered in double_scalars
  return(np.float64(self.TPR) / self.FPR)

Confusion Matrix:

Predicted     0  1   2   3  4   5   6   7  8   9  10  __all__
Actual
0             219  0   1   4   33   3   2   0  0   0   0     262
1             5  1   0   1  0   0   0   0  0   0   0         7
2             2  1   8   4  0   0   1   0  0   0   0        16
3             7  1   2  37  1   1   1   0  0   3   0        53
4             42  0   0   3   72   0   1   0  0   0   0      118
5             4  0   0   2  5  13   0   0  0   0   0        24
```

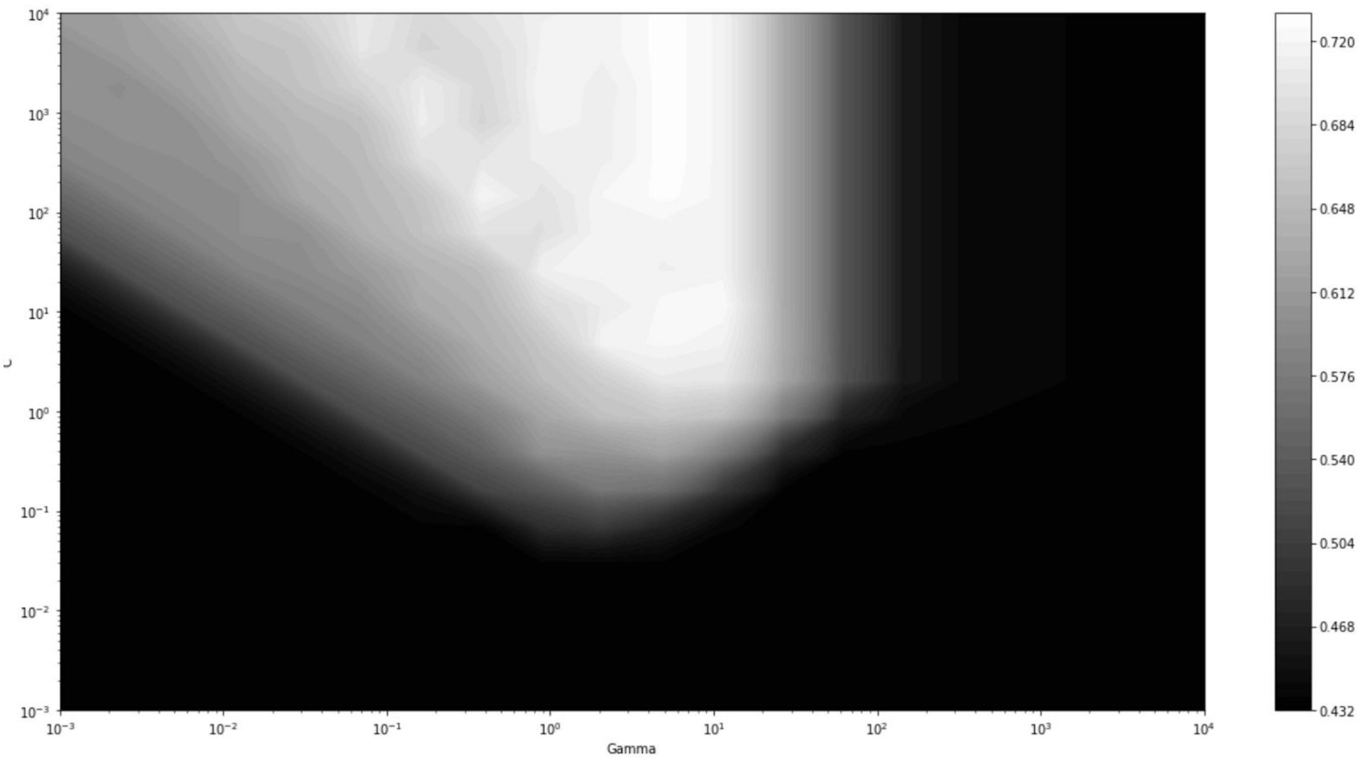
```
6             6  0   0   2  0   1  18   0  0   0   0         27
7             5  0   0   0  0   0   0  38  0   0   0         43
8             3  0   0   2  0   0   0   0  0   0   0         5
9             3  0   0   1  0   0   0   0  0  22   2         28
10            8  0   0   0  0   0   0   0  0   0   9         17
__all__       304  3  11  56  111  18  23  38  0  25  11     600


Overall Statistics:

Accuracy: 0.7283333333333334
95% CI: (0.6908408732251395, 0.7635608596930066)
No Information Rate: ToDo
P-Value [Acc > NIR]: 1.350615014115965e-28
Kappa: 0.6245897549085466
Mcnemar's Test P-Value: ToDo


Class Statistics:

Classes                                              0   ...      10
Population                                           600  ...     600
P: Condition positive                                262  ...           17
N: Condition negative                                338  ...     583
Test outcome positive                                304  ...           11
Test outcome negative                                296  ...     589
TP: True Positive                                    219  ...           9
TN: True Negative                                    253  ...     581
FP: False Positive                                   85  ...            2
FN: False Negative                                   43  ...            8
TPR: (Sensitivity, hit rate, recall)   0.835878  ... 0.529412
TNR=SPC: (Specificity)                  0.748521  ...0.996569
PPV: Pos Pred Value (Precision)         0.720395  ...0.818182
NPV: Neg Pred Value                     0.85473  ... 0.986418
FPR: False-out                          0.251479  ...  0.00343053
FDR: False Discovery Rate               0.279605  ...0.181818
FNR: Miss Rate                          0.164122  ...0.470588
ACC: Accuracy                           0.786667  ...0.983333
F1 score                                0.773852  ...0.642857
MCC: Matthews correlation coefficient  0.579743  ... 0.650541
Informedness                            0.584399  ...0.525981
Markedness                              0.575124  ...0.804599
Prevalence                              0.436667  ...   0.0283333
LR+: Positive likelihood ratio          3.32384  ... 154.324
LR-: Negative likelihood ratio          0.219262  ...0.472208
DOR: Diagnostic odds ratio              15.1592  ... 326.813
FOR: False omission rate                0.14527  ...   0.0135823

[26 rows x 11 columns]
```

Confusion matrix Image

```
[26 rows x 11 columns]
```



a) **The plot of accuracy for different combinations of c and gamma:**

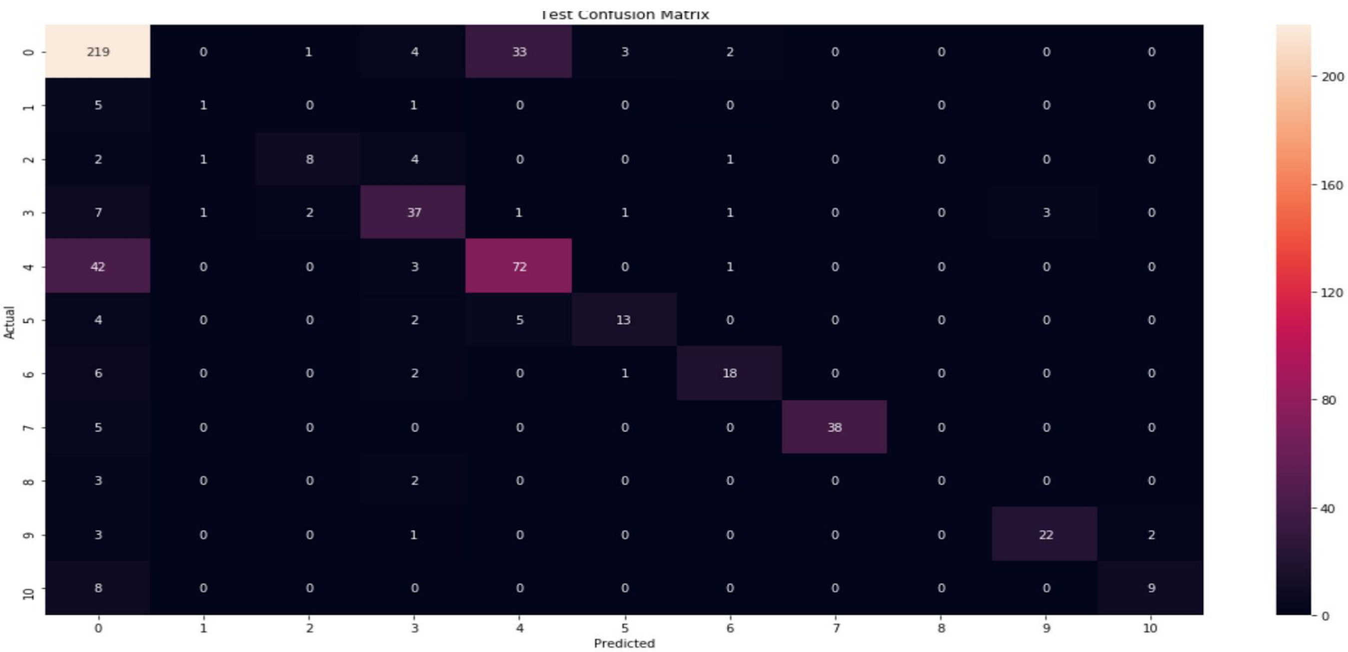


Generalization Error

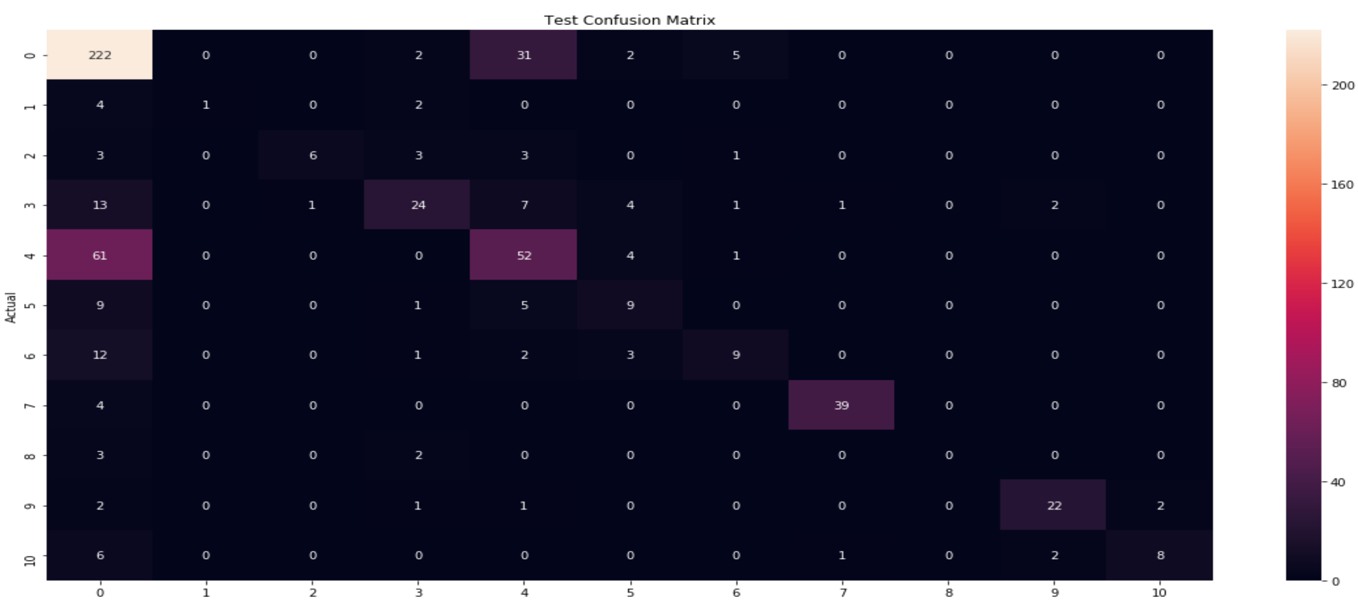
```
gamma=5
C=200
model =  SVC(kernel='rbf', C=C, gamma=gamma)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
test_error = 1- metrics.accuracy_score(yhat, y_test)
print(test_error)
0.2716666666666666
```

The white region in the above figure shows the area with high accuracy. The region for values of gamma=5 and c>=150 is expected to be with the highest accuracy in the given plot. After selecting the values of gamma=5 and c= 200, the generalisation error obtained is 0.2716 and accuracy=0.7283. These values are better than the KNN model (they are compared in the following section).

Confusion Matrix for Test Data on SVM (RBF)



Confusion Matrix for Test Data on best KNN model with k=10



	SVM model (gamma=5, c=100)	Best KNN model (k=10)
Generalisation error	0.2716	0.3466
Overall Accuracy	0.7283	0.6533

The generalisation error in the SVM model is lesser than the Best KNN model.

b) Which model do you prefer and why? Be brief and clear

As we can see from the comparisons in the previous questions, we can prefer SVM model over KNN. If we prioritize on the basis of generalization error and accuracy, the SVM performs better than the KNN for this set of data.