rpate375-KNN

April 15, 2020

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from sklearn import preprocessing
from sklearn.model_selection import train_test_split as tts
import scipy.spatial.distance as dist_measure
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap
from sklearn.metrics import confusion_matrix
```

[2]:
```
raw_data= pd.read_csv("iris.
 ↪csv",names=['sepal_length','sepal_width','petal_length','petal_width','labels'])
raw_data.tail()
```

[2]:

	sepal_length	sepal_width	petal_length	petal_width	labels
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

[3]:
```
#encoding the the class labels
labelencoder= preprocessing.LabelEncoder()
raw_data['labels']= labelencoder.fit_transform(raw_data['labels'])
raw_data.tail()
```

[3]:

	sepal_length	sepal_width	petal_length	petal_width	labels
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

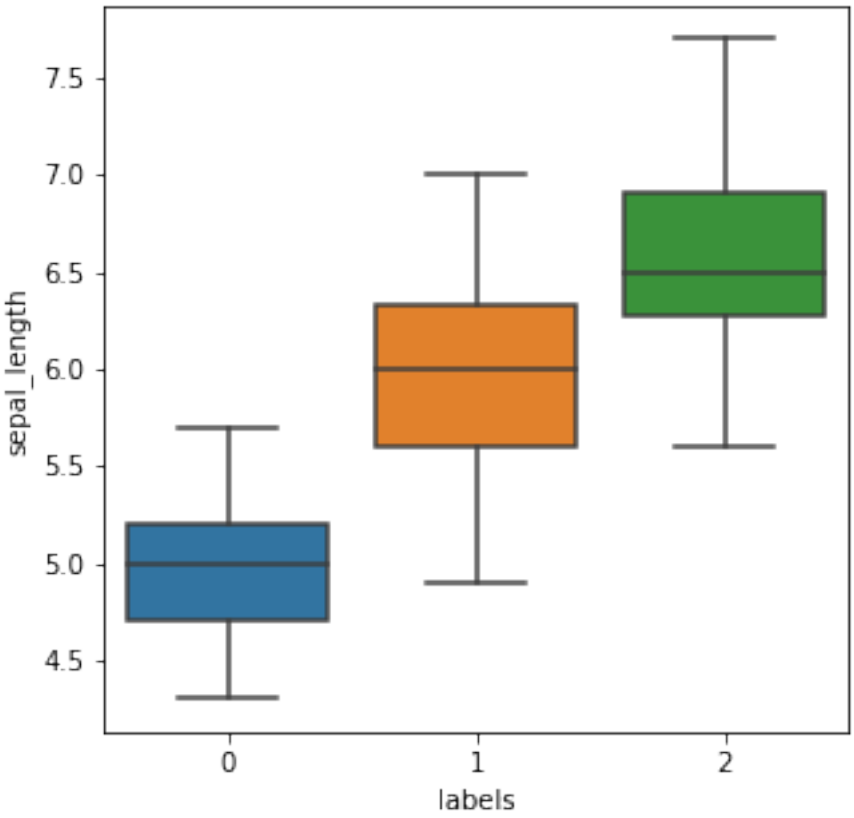
[4]:
```
#splitting the data into training set and test set using stratify sampling
X_train, X_test, y_train, y_test = tts(raw_data, raw_data['labels'],␣
 ↪test_size=0.2,stratify=raw_data['labels'])
print (X_train.shape, y_train.shape)
```

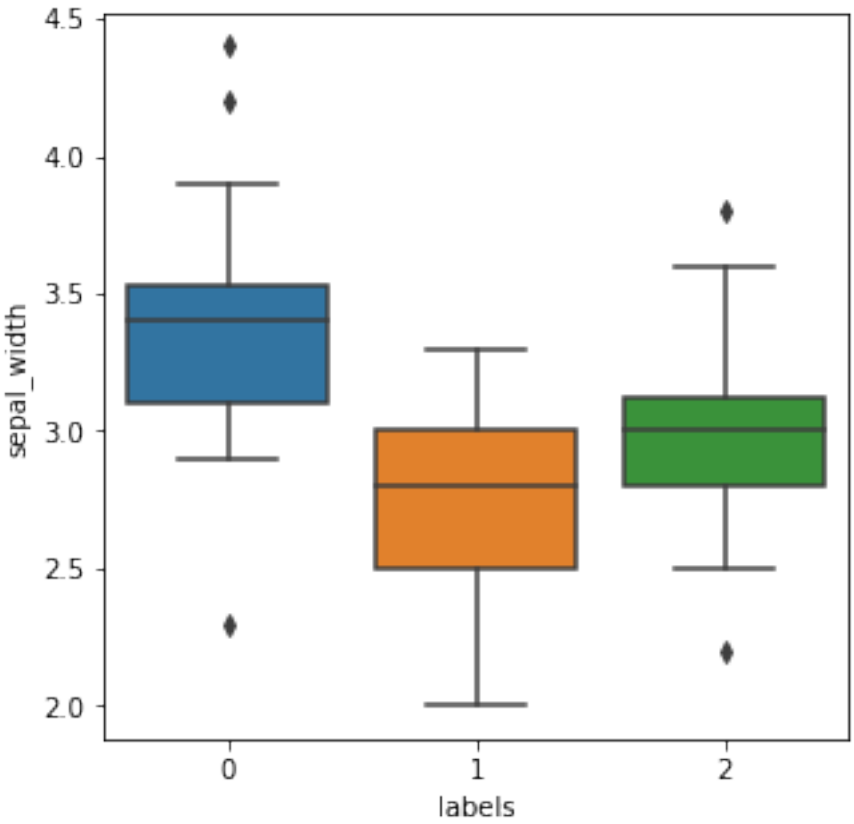
```
(120, 5) (120,)
```

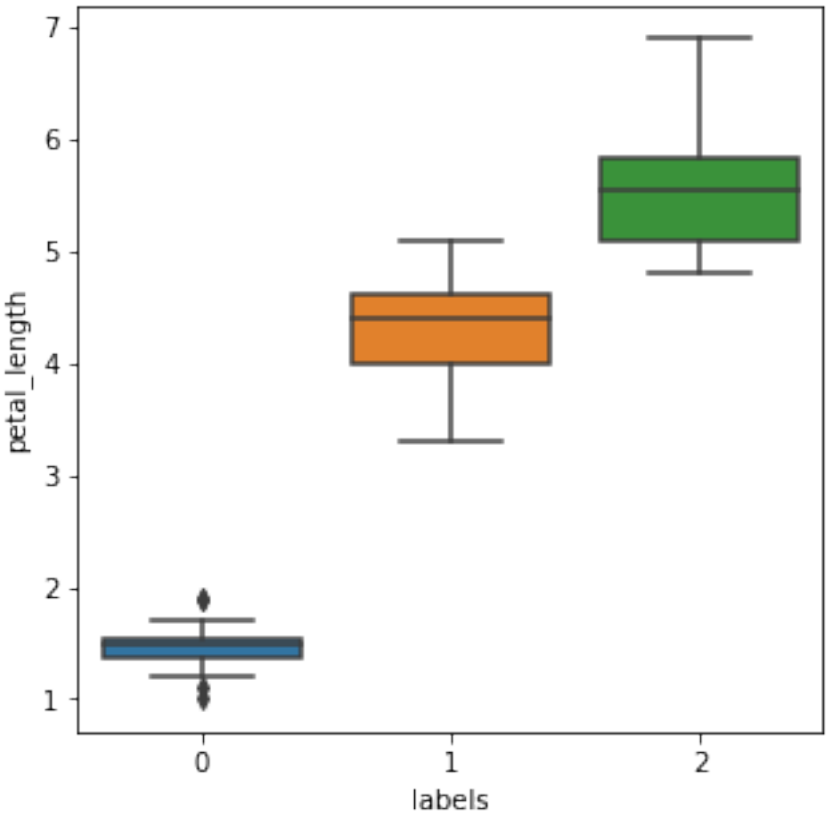
[5]:
```
# the function which returns the result of the predictions
#parameters: train set, test set, type of distance used for measurement, k␣
 ↪which the number of the nearest neighbors


def k_nearest_neighbor(train_set,target_set,test_set,dist_type,k,**kwargs):
    p = kwargs.get('p', 0)
    random_voting=kwargs.get('random_voting',False)
    #calculate the distance between the points for each instance in test to the␣
 ↪points in the training set
    Y=dist_measure.cdist(train_set, test_set,dist_type,p)
    #converting to dataframe with columns as the test data index and indices as␣
 ↪train set index
    #each entry in the dataframe represents the distance between [train_set␣
 ↪instance][test_set_instance].
    Y=pd.DataFrame(Y)
    #creating the dictionary with the predicted value.
    answer={}
    #looping through each of the column(test set index) and getting the k␣
 ↪nearest indices of the training set
    # and gettting the class label from the train set and using mode to predict␣
 ↪the encoded class label.
    for test_point in Y.columns:
        #getting the k nearest points
        indices=Y[test_point].nsmallest(k).index.to_list()
        predicted_label=[target_set.iloc[x] for x in indices ]
        answer[test_point]=predicted_label
    output=pd.DataFrame(answer)
    if(random_voting==True):
        output=output.sample(n=1,random_state=1).dropna().T
        output =pd.DataFrame(output[:].values.ravel())
    else:
        output=output.mode().dropna().T
    output=output.rename(columns={0:"predicted_label_value"}).
 ↪astype({"predicted_label_value": 'int32'})
    return output


def test_prediction_results(predicted_value, actual_value,k,**kwargs):
    print_accuracy=kwargs.get('print_accuracy',True)
```

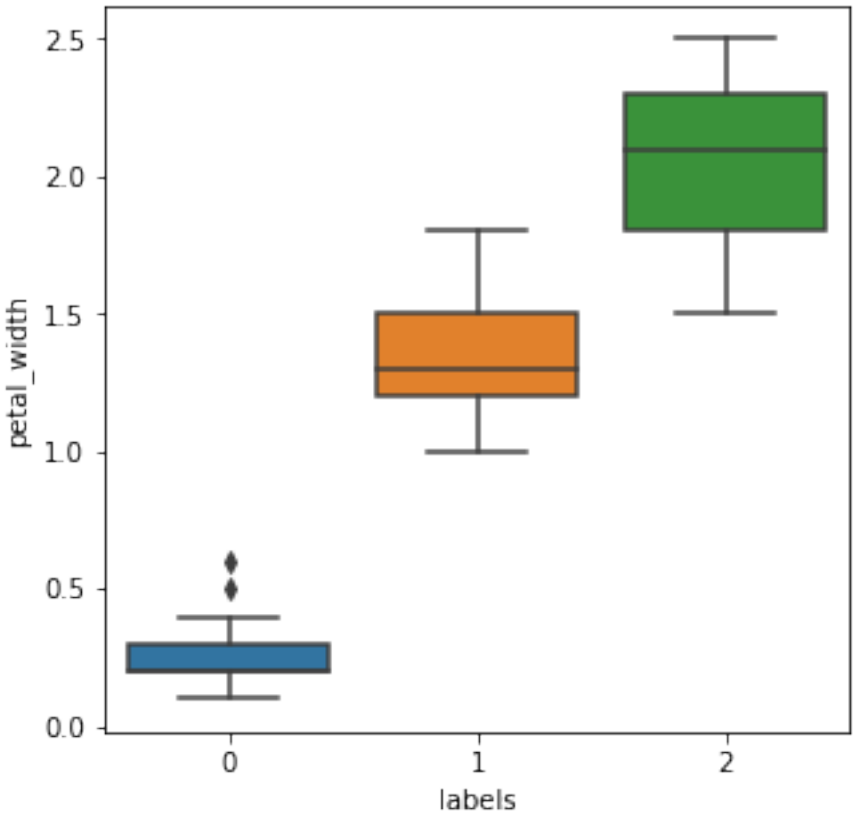
```
    predicted_value['Actual_label_value']=list(actual_value)
    ␣
 ↪accuracy=accuracy_score(predicted_value['predicted_label_value'],predicted_value['Actual_la
 ↪)
    if(print_accuracy):
        print("Accuracy of this KNN classifer for k=%d is %f"%(k,accuracy))
    return (accuracy,k,predicted_value)

def sklearn_knnClassifer(train,test,target,test_target,k):
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(train,target)
    print(neigh.predict(test))
    print('model avg mean accuracy: ',neigh.score(test,test_target))

# sklearn_knnClassifer(X_train,X_test,y_train,y_test,15)
# # sklearn_knnClassifer(X_train,X_test,y_train,20)
X_train=X_train.iloc[:,:4]
X_test=X_test.iloc[:,:4]

# predicted_labels=k_nearest_neighbor(X_train,y_train,X_test,'minkowski',6,p=3)
# testresult=test_prediction_results(predicted_labels,y_test,6)
```

[6]:
```
#Comparing the feature distribution using following function
def compare_features_by_class(X_train,y_train):
    for column in X_train.columns:
        ax,fig=plt.subplots(figsize=(5,5))
        sns.boxplot(x=y_train,y=X_train[column])
compare_features_by_class(X_train,y_train)
```


7.5
7.0
6.5
6.0
5.5
5.0
4.5
sepal_length
0
1
2
labels


4.5
4.0
3.5
3.0
2.5
2.0
sepal_width
0
1
2
labels


7
6
5
4
3
2
1
petal_length
0
1
2
labels



0.1 Understanding the box plot

1. label 1 has the minimum outliers compared to the other labels.
2. sepal length has the lowest outliers among the features and sepal width has the most.

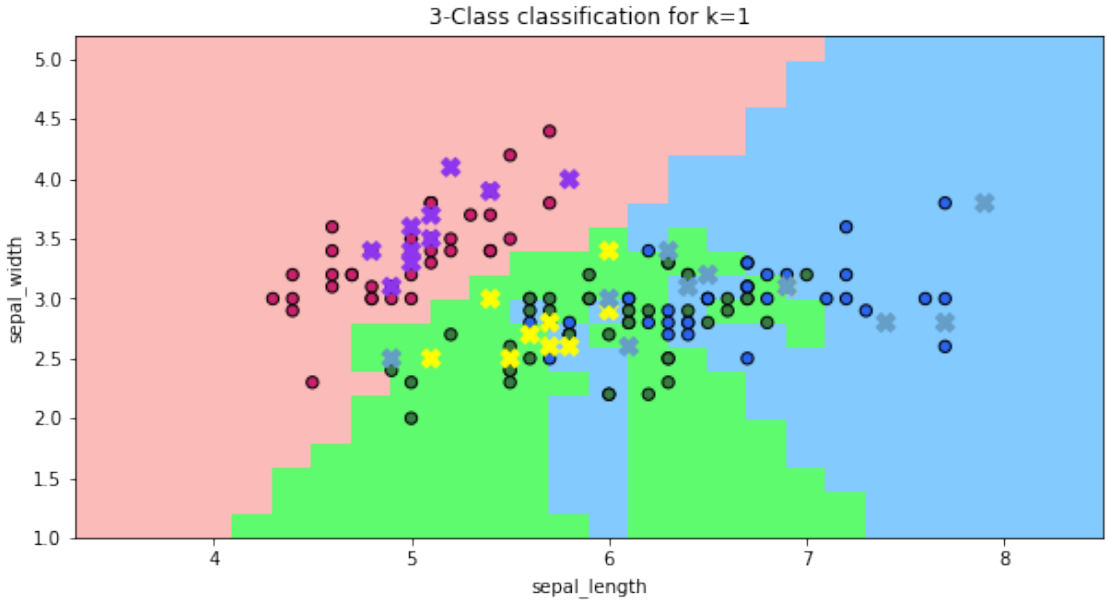
[7]:
```
def plotly_compare_features_by_class(X_train,y_train):
    fig=go.Figure()
    for column in X_train.columns:
            fig.add_trace(go.Box(x=y_train,y=X_train[column],name=column))
    fig.update_layout(boxmode='group')
    fig.show()
plotly_compare_features_by_class(X_train,y_train)
```

[8]:
```
#reference: professor's provided example
cmap_light = ListedColormap(['#FBBBB9', '#5EFB6E', '#82CAFF'])
cmap_bold = ListedColormap(['#CA226B', '#387C44', '#2B65EC'])
cmap_test = ListedColormap(['#8E35EF', '#FFFF00', '#659EC7'])
def decision_boundary_plot(train,target,test,test_target,k,p):
    title="3-Class classification for k={}".format(k)
    train=train.iloc[:,:2]
    model = KNeighborsClassifier(n_neighbors=k,p=p)
```

```
    model.fit(train,target)
    #meshstep size parameter
    h=0.2
    x_min,x_max=train.iloc[:,0].min()-1,train.iloc[:,0].max()+1
    y_min,y_max=train.iloc[:,1].min()-1,train.iloc[:,1].max()+1

    xx,yy=np.meshgrid(np.arange(x_min,x_max,h),np.arange(y_min,y_max,h))

    Z=model.predict(np.c_[xx.ravel(),yy.ravel()])
    Z=Z.reshape(xx.shape)
    plt.figure(figsize=(10,5))
    plt.title(title);
    plt.xlabel(train.columns[0])
    plt.ylabel(train.columns[1])
    plt.pcolormesh(xx,yy, Z, cmap=cmap_light)
    plt.scatter(train.iloc[:,0],train.iloc[:
↪,1],c=target,cmap=cmap_bold,edgecolor='k',s=40)
    plt.scatter(test.iloc[:,0],test.iloc[:
↪,1],c=test_target,cmap=cmap_test,linewidth=1, marker='X',s=100)
    plt.show()
```

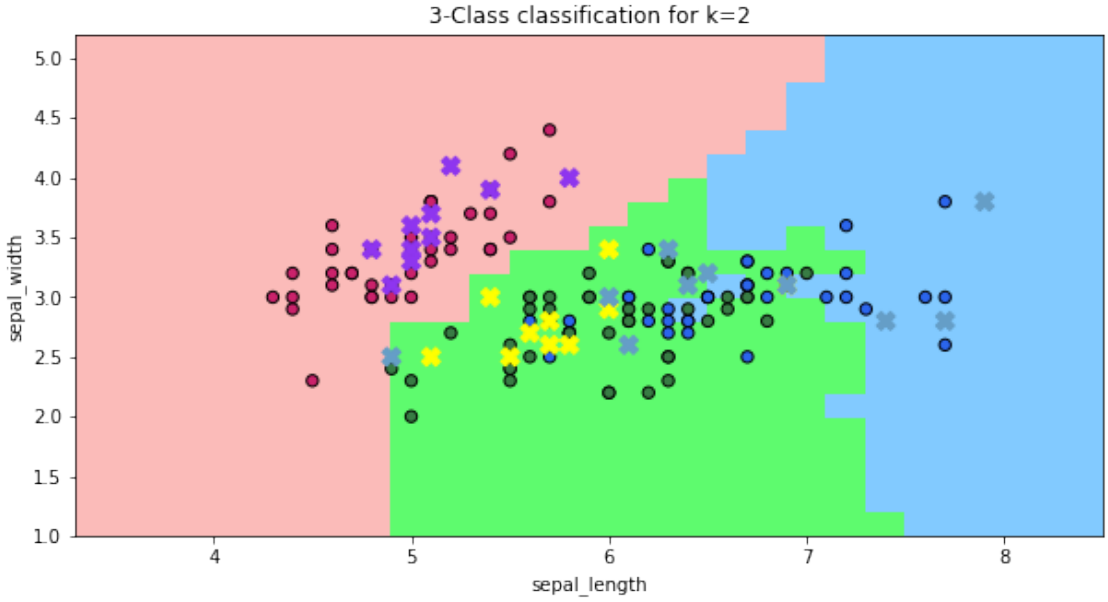
[9]:
```
#b with k=1 plotting the decision boundaries
#defined all in one function.
def results(X_train,y_train,X_test,y_test,distance_measure,k,**kwargs):
    p=kwargs.get('p',0)
    ␣
↪predicted_labels=k_nearest_neighbor(X_train,y_train,X_test,distance_measure,k,p=p)
    results=test_prediction_results(predicted_labels,y_test,k)
    decision_boundary_plot(X_train,y_train,X_test,y_test,k,2)
    return results

result=results(X_train,y_train,X_test,y_test,'euclidean',1)
```

```
Accuracy of this KNN classifer for k=1 is 0.966667
```

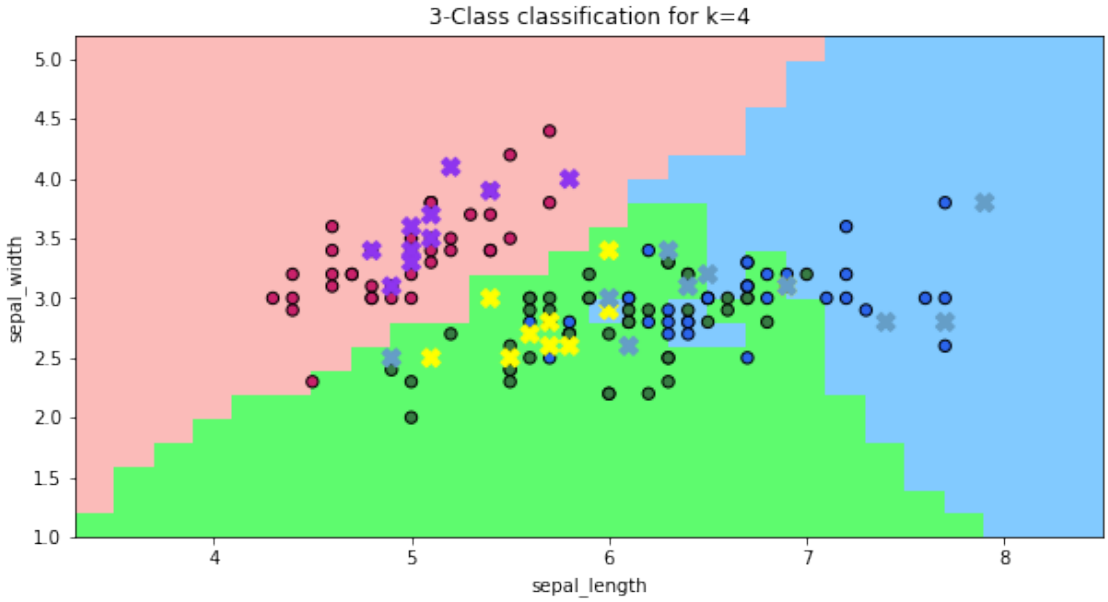


[10]:
```
# for different value of k plotting the boundary and predicting the class label.
list_of_k=[2,4,6,10,15]
for k in list_of_k:
    result=result=results(X_train,y_train,X_test,y_test,'euclidean',k)
```

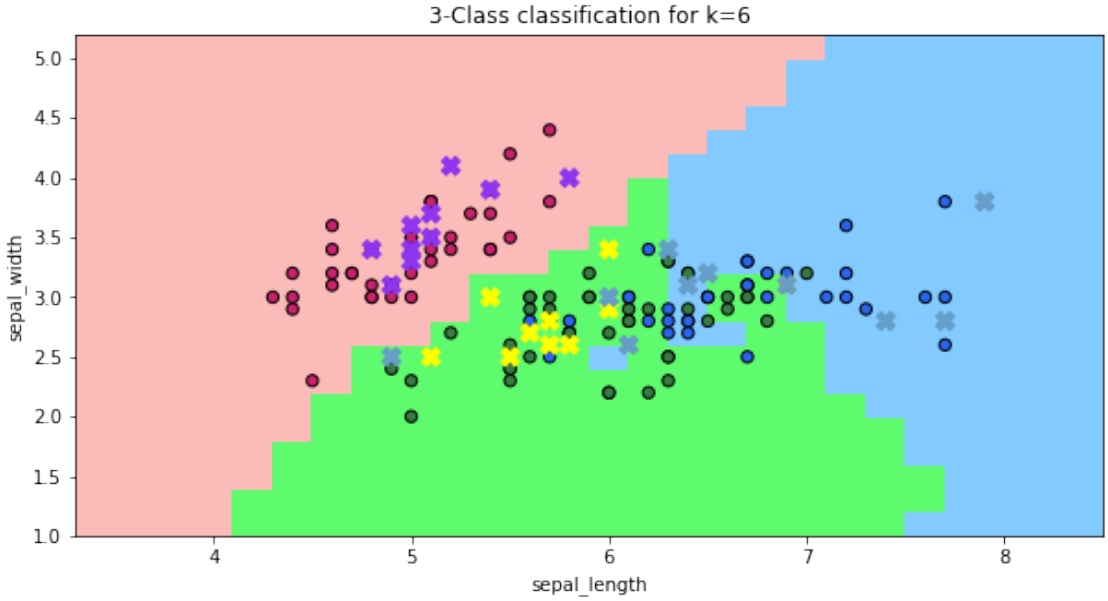
```
Accuracy of this KNN classifer for k=2 is 0.900000
```



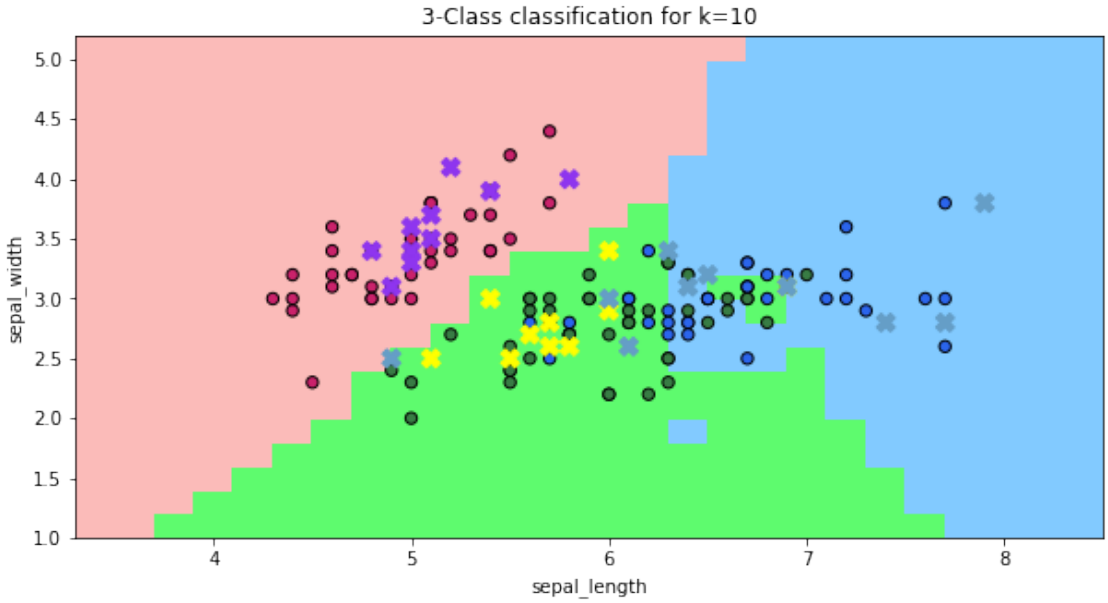
```
Accuracy of this KNN classifer for k=4 is 0.966667
```



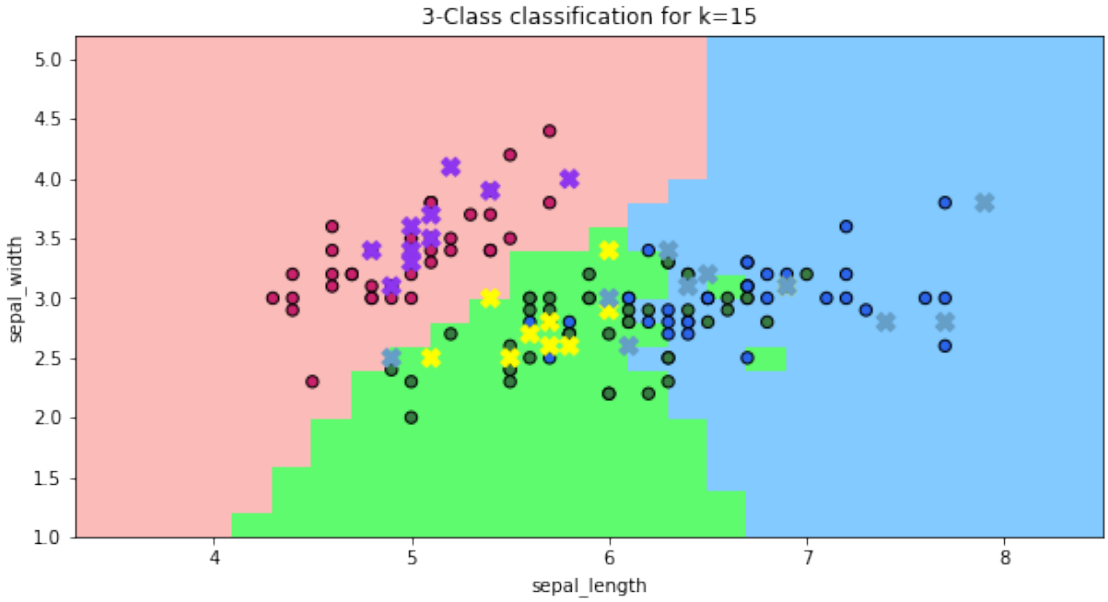

```
Accuracy of this KNN classifer for k=6 is 0.966667
```




```
Accuracy of this KNN classifer for k=10 is 0.933333
```




```
Accuracy of this KNN classifer for k=15 is 0.966667
```

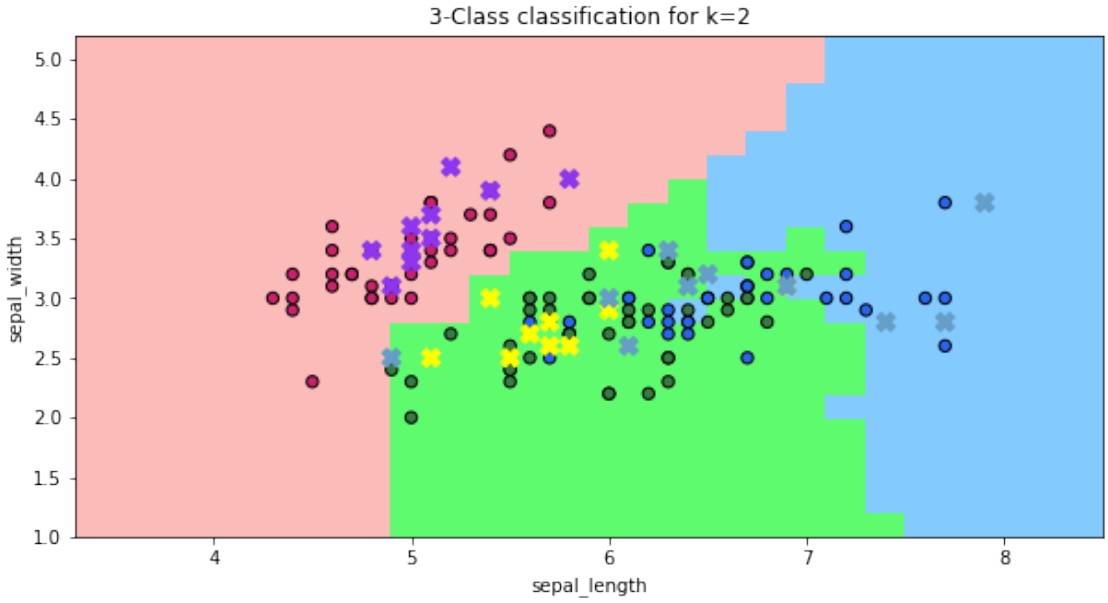


0.2 Insights from the decision boundary plot with different values of K

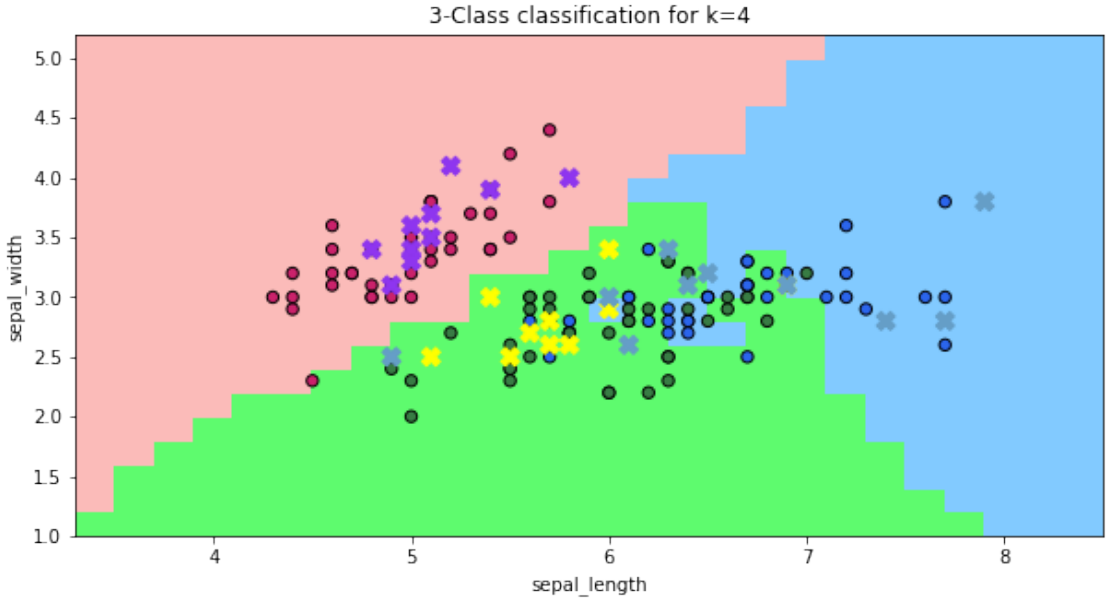
1. when the k value is small the boundries are not smooth and the accuracy is less compared to a the a higher value of K.
2. and when value is very large then the boundries does get smooth but also opens up pockets.

[11]:
```
list_of_k=[2,4,6,10,15]
for k in list_of_k:
    result=result=results(X_train,y_train,X_test,y_test,'minkowski',k,p=3)
```

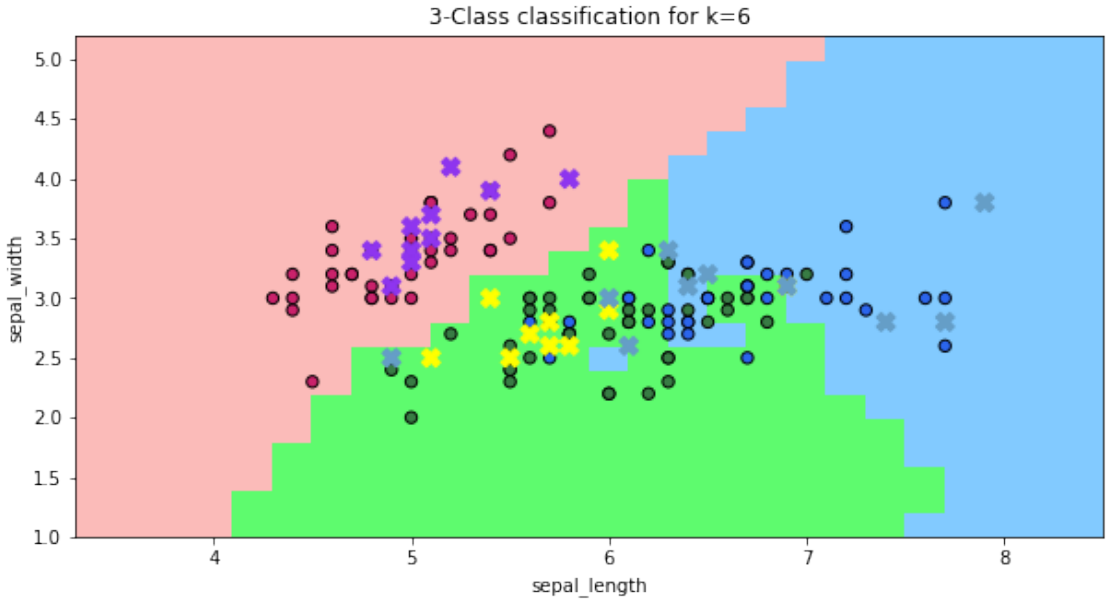
```
Accuracy of this KNN classifer for k=2 is 0.900000
```



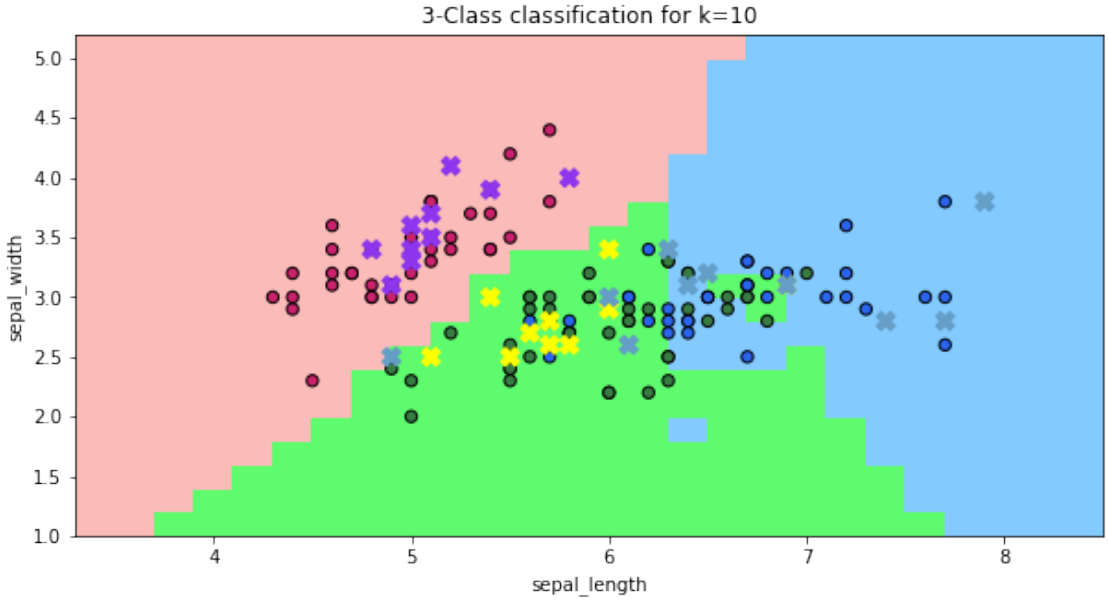

```
Accuracy of this KNN classifer for k=4 is 0.966667
```



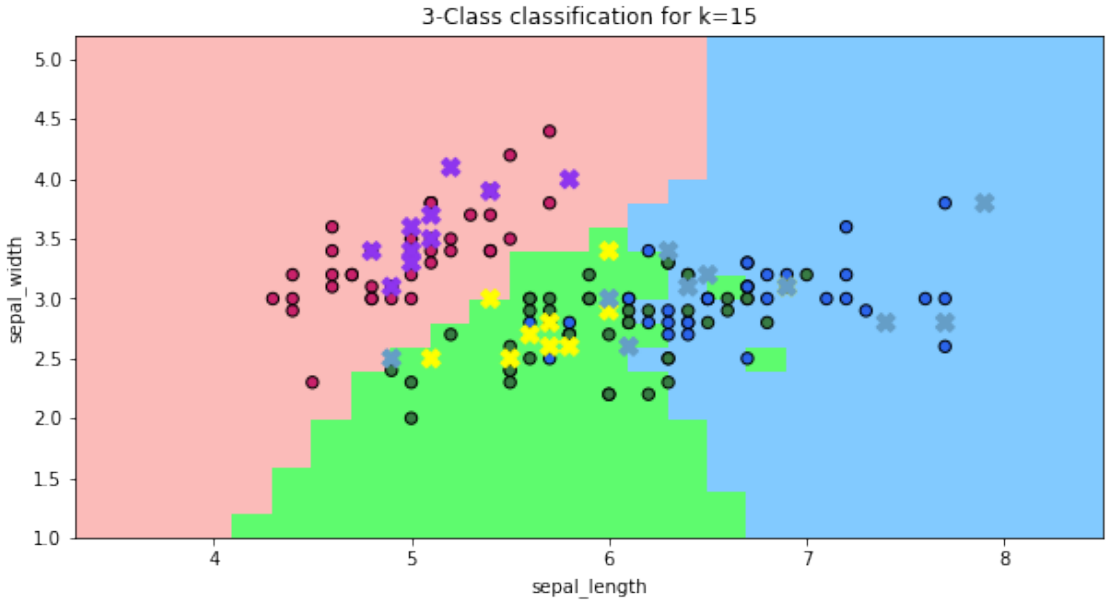
```
Accuracy of this KNN classifer for k=6 is 0.966667
```




```
Accuracy of this KNN classifer for k=10 is 0.966667
```




```
Accuracy of this KNN classifer for k=15 is 0.966667
```



0.3 Euclidean distance vs Minkowski distance

-I have used the stratified sampling and so the results are different everytime.
1. There is change in which the grouping is done in both the classfication but the accuracy is almost the same. 2. Increase in the K value from 2 to 15 in both the metrics had a different result of accuracy for some test set. 3. The average accuracy for k=2,4,6,10,15 was about .98 for minkowski distance almost the same for euclidean distance.

1 Part 2

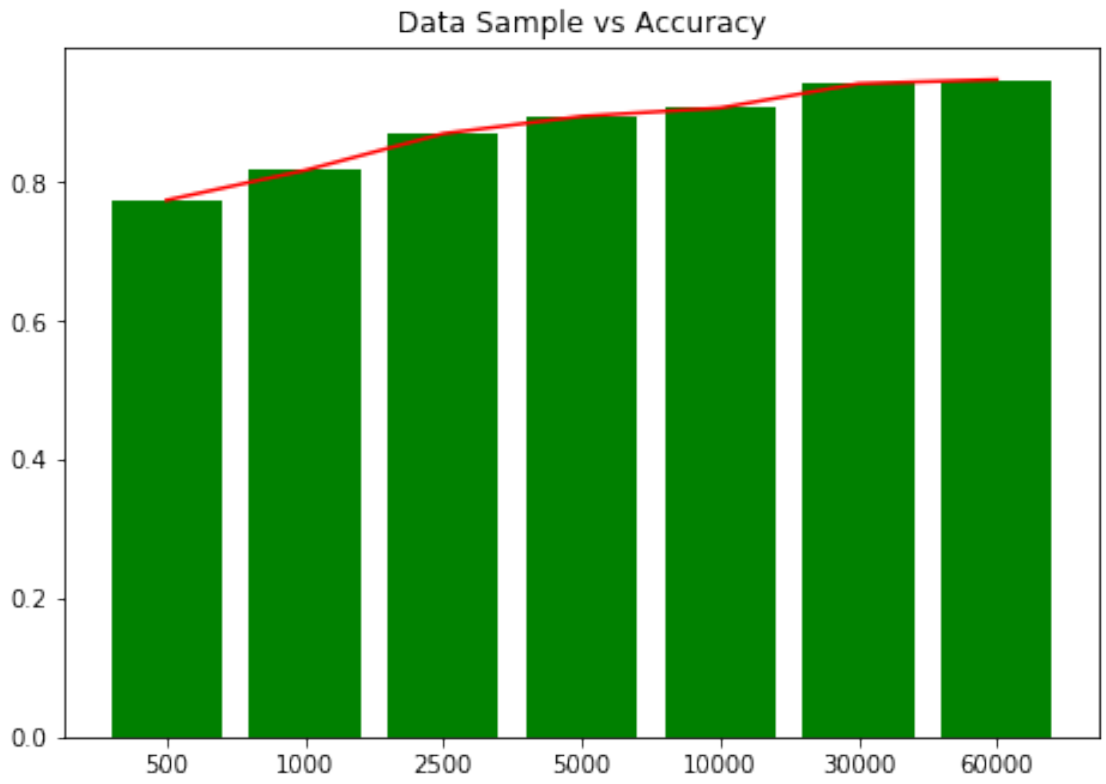
[12]:
```
#loading the data sets
train_data= pd.read_csv('mnist_train.csv',header=None)
test_data=pd.read_csv('mnist_test.csv',header=None)
```

[13]:
```
#function which computes the prediction for the datasets and returns a tuple␣
↪of predicted label,k and accuracy
def mnist_prediction(train_data,test_data,num_train,num_test,k,**kwargs):
    #if the we choose to do random voiting instead of majority voting
    random_voting=kwargs.get('random_voting',False)
    #passing the train and test set to the classifier
    predicted_labels=k_nearest_neighbor(train_data.iloc[:num_train,1:
↪],train_data.iloc[:num_train,0],test_data.iloc[:num_test,1:
↪],'euclidean',2,random_voting=random_voting)
    #predicted labels is a dataframe which has two features  actual label and␣
↪predicted label
```

```
    return test_prediction_results(predicted_labels,test_data.iloc[:
↪num_test,0],k,print_accuracy=False)
```

[14]:
```
%%time
#list of input sizes
list_of_inputs=[500,1000,2500,5000,10000,30000,60000]

#getting the error for each of the sizes
error_vs_training=[]
for train_data_points in list_of_inputs:
    ␣
↪accuracy,k,predicted_value=mnist_prediction(train_data,test_data,train_data_points,1000,2)
    error_vs_training.append([train_data_points,accuracy])

error_vs_training=pd.DataFrame(error_vs_training,columns=['Data␣
↪Sample','Accuracy'])
print(error_vs_training)
#     print('for %d training set and %d test set with k=%d the accuracy is:␣
↪%f'%(train_data_points,1000,k,accuracy))
#plotting the figure
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.set_title("Data Sample vs Accuracy")
ax.plot(error_vs_training['Data Sample'].apply(lambda x:
↪str(x)),error_vs_training['Accuracy'],color='red')
ax.bar(error_vs_training['Data Sample'].apply(lambda x:
↪str(x)),error_vs_training['Accuracy'],color='green')
plt.show()
```

```
   Data Sample  Accuracy
0          500     0.774
1         1000     0.817
2         2500     0.870
3         5000     0.895
4        10000     0.907
5        30000     0.942
6        60000     0.948
```

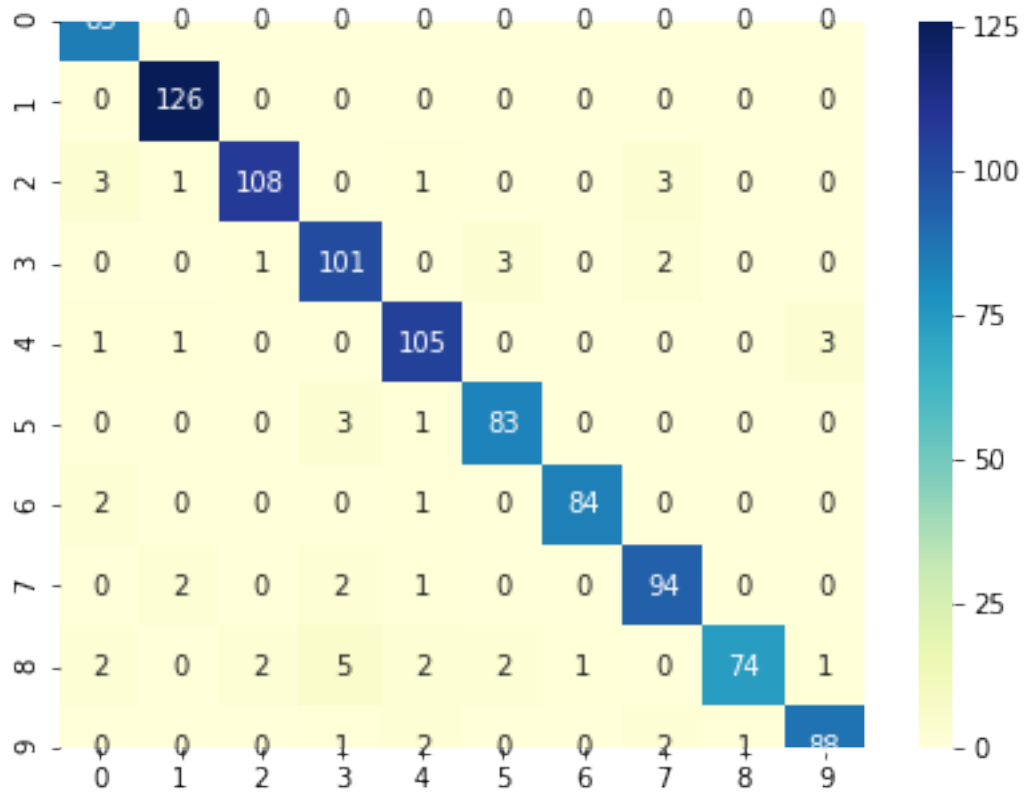



```
CPU times: user 1min 34s, sys: 749 ms, total: 1min 35s
Wall time: 1min 35s
```

[15]:
```
#the best case was with 60,000 observations.
accuracy,k,predicted_value=mnist_prediction(train_data,test_data,60000,1000,2)

c_matrix=confusion_matrix(predicted_value['Actual_label_value'],␣
 ↪predicted_value['predicted_label_value'])
c_maxtrix=pd.DataFrame(c_matrix,columns=[i for i in range(0,10)])
print(c_maxtrix)
f, ax = plt.subplots(figsize=(7, 5))
ax=sns.heatmap(c_matrix,annot=True,square=True,cmap="YlGnBu", fmt="d")
```

```
    0    1    2    3    4    5    6    7    8    9
0  85    0    0    0    0    0    0    0    0    0
1   0  126    0    0    0    0    0    0    0    0
2   3    1  108    0    1    0    0    3    0    0
3   0    0    1  101    0    3    0    2    0    0
4   1    1    0    0  105    0    0    0    0    3
5   0    0    0    3    1   83    0    0    0    0
6   2    0    0    0    1    0   84    0    0    0
7   0    2    0    2    1    0    0   94    0    0
```

```
8   2    0    2    5    2    2    1    0   74    1
9   0    0    0    1    2    0    0    2    1   88
```



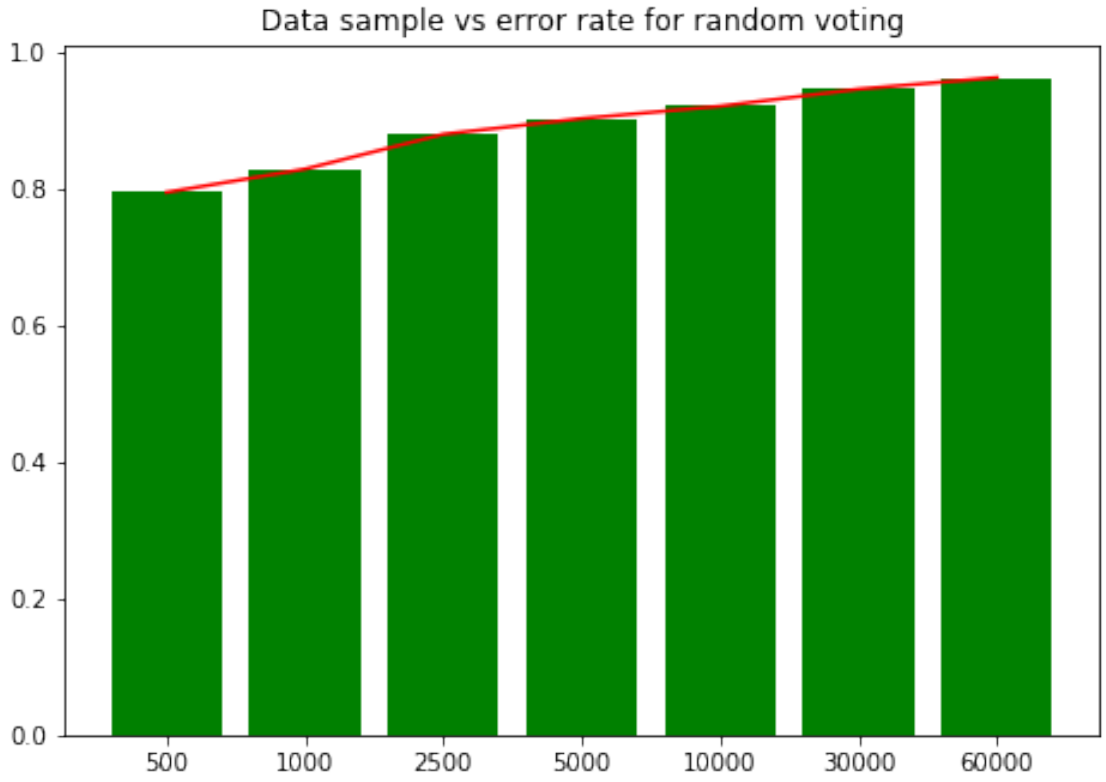
1.1 Understanding the confusion matrix

1. The columns represents the actual labels and the rows represents the predicted labels.
2. The diagonal in the confusion matrix shows the True Positives.
3. taking an instance in the matrix, for example row 9th and 3rd column value 5 shows that the classifer predicted label 3 as 8 for 5 times thats false negative.

[16]:
```
rand_error_vs_training=[]
for train_data_points in list_of_inputs:
    ␣
 ↪accuracy,k,predicted_value=mnist_prediction(train_data,test_data,train_data_points,1000,2,r
    rand_error_vs_training.append([train_data_points,accuracy])
rand_error_vs_training=pd.DataFrame(rand_error_vs_training,columns=['Data␣
 ↪Sample','Accuracy'])
print(rand_error_vs_training)
#     print('for %d training set and %d test set with k=%d the accuracy is:␣
 ↪%f'%(train_data_points,1000,k,accuracy))
fig = plt.figure()
```

```
ax = fig.add_axes([0,0,1,1])
ax.set_title("Data sample vs error rate for random voting")
ax.plot(rand_error_vs_training['Data Sample'].apply(lambda x:
 ↪str(x)),rand_error_vs_training['Accuracy'],color='red')
ax.bar(rand_error_vs_training['Data Sample'].apply(lambda x:
 ↪str(x)),rand_error_vs_training['Accuracy'],color='green')
plt.show()
```

	Data Sample	Accuracy
0	500	0.794
1	1000	0.828
2	2500	0.879
3	5000	0.902
4	10000	0.920
5	30000	0.945
6	60000	0.962



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: