Lab 5 - Classification : k-NN and Naive Bayes (using sklearn libraries)

k-NN

Use **"Pima Indians Diabetes Dataset from UCI Machine Learning Repository"** for this question.It is a binary class dataset. Split the dataset into train(80%), validation(10%) and test sets(10%).

Run k-Nearest neighbours for different k values. Choose your own subset of k (atleast 10) and choose the best value of k from this subset. In solving real-world problems, the values of k are chosen based on experience and hence it is a tunable hyperparameter. Select the k, using validation set, which returns the best accuracy score. Report accuracy score by performing k-NN on the test dataset using the chosen k value.

In [1]:

```
#Loading libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
```

In [2]:

```
#Loading the data
diab_data = pd.read_csv('diabetes.csv')
diab_data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

In [3]:

```
#Splitting data into training, validation and testing sets
X_train, X_val, y_train, y_val = train_test_split(diab_data.loc[:, diab_data.columns != 'Outcome'], diab_data['Outcome'], test_size=0.2, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.5, random_state=4)

print(X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape)
```

```
(614, 8) (77, 8) (77, 8) (614,) (77,) (77,)
```

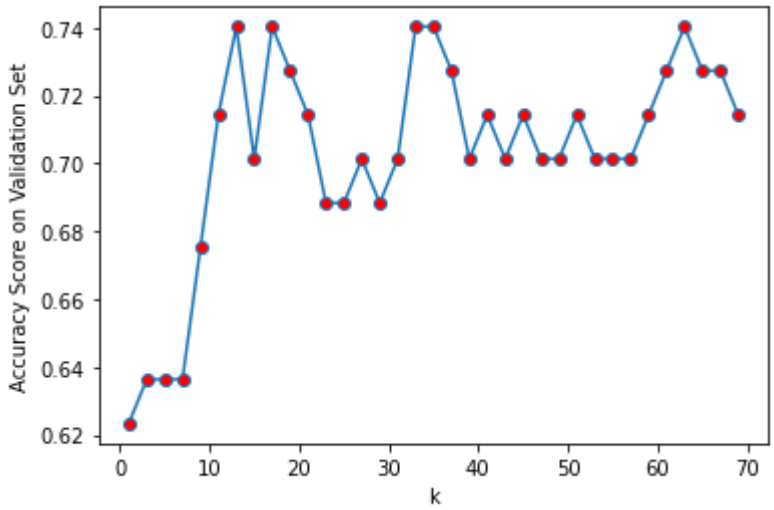
In [4]:

```
#Applying K-NN
from sklearn.neighbors import KNeighborsClassifier

#k values: (1,3,7,15,31,63)
#k_vals = [1,3,7,9,11,13,15,31,45,55,63,71,91,101,111,131,151]
k_vals = range(1,70,2)
scores=[]
for k in k_vals:
    Model = KNeighborsClassifier(n_neighbors = k)
    Model.fit(X_train,y_train)
    score = Model.score(X_val,y_val)
    scores.append(score)
```

In [5]:

```
#Plotting for different k:
plt.plot(k_vals,scores, linestyle='-', marker='o', markerfacecolor ='r');
plt.xlabel('k')
plt.ylabel('Accuracy Score on Validation Set')

bestK = k_vals[scores.index(max(scores))]
print('Value of k which performs best on validation set is',bestK)
```

```
Value of k which performs best on validation set is 13
```



In [6]:

```
#Test accuracy for k=13
Model = KNeighborsClassifier(n_neighbors = bestK)
Model.fit(X_train,y_train)
test_score = Model.score(X_test,y_test)
print("Test accuracy for k=%d is %.3f"%(bestK,test_score))
```

```
Test accuracy for k=13 is 0.805
```

Summarize your findings and results here

Naive Bayes

Use **"Optical recognition of handwritten digits dataset"** for this question. **Download dataset from sklearn**. The dataset has 10 classes and 64 attributes (8x8 images). Visualise images from the dataset. Perform a train test split in the ratio 4:1.

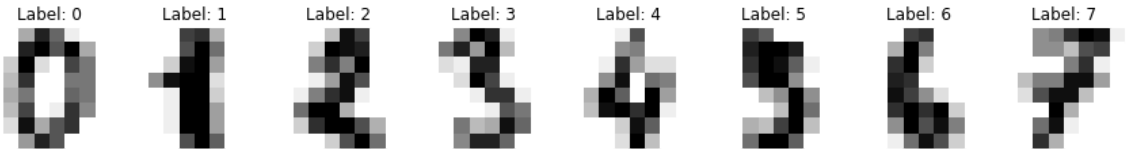
Naive Bayes - perform multiclass classification to classify the dataset into one of the ten classes. Experiment with the priors (Gaussian and Bernoulli) and report the best prior. Report the accuracies in terms of F1 scores and the confusion matrix (use sklearn functions for this too).

In [7]:

```
#Loading the datset
from sklearn.datasets import load_digits
digits = load_digits()
```

In [8]:

```
#Visualising the images
_, axes = plt.subplots(1, 8,figsize=(15,4))
images_and_labels = list(zip(digits.images, digits.target))
for ax, (image, label) in zip(axes[:], images_and_labels[:8]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Label: %i' % label)
```



In [9]:

```
n_samples = len(digits.images)
xdata = digits.images.reshape((n_samples, -1))
```

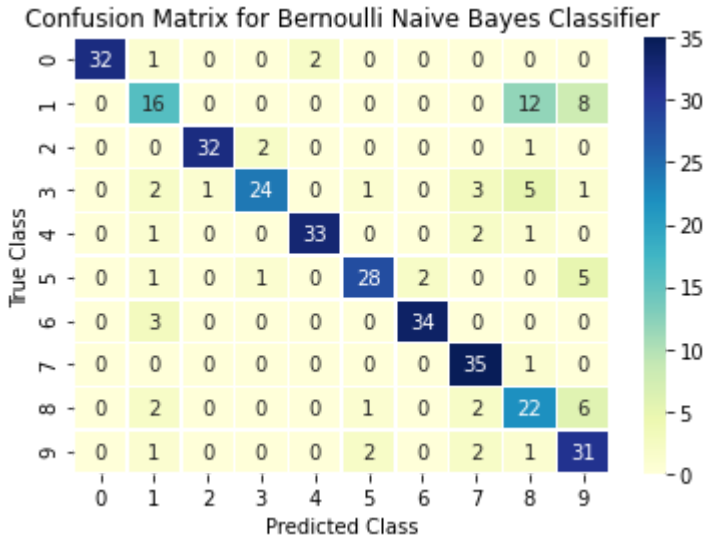
In [10]:

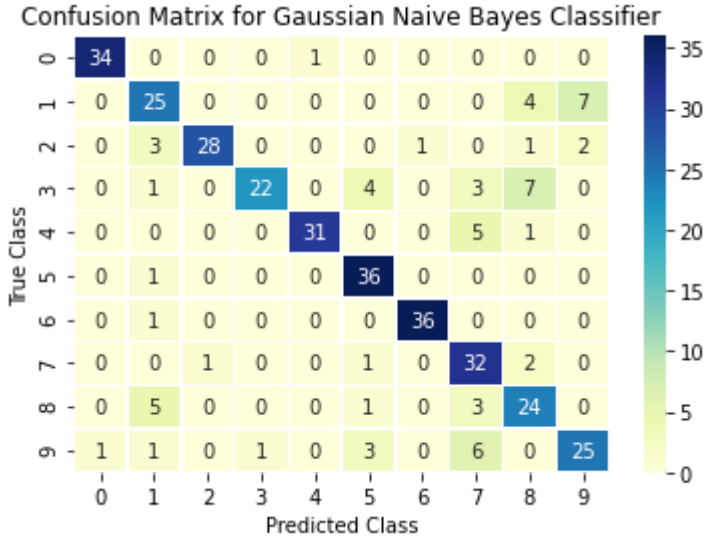
```
# Split data into train and test subsets
X_train, X_test, y_train, y_test = train_test_split(
    xdata, digits.target, test_size=0.2, shuffle=False)
```

In [11]:

```
#Naive bayes classifier
from sklearn.naive_bayes import BernoulliNB, GaussianNB
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

clf = BernoulliNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
BernoulliF1Score = f1_score(y_test,y_pred,average='micro')
BernoulliCM = confusion_matrix(y_test,y_pred)

clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
GaussianF1Score = f1_score(y_test,y_pred,average='micro')
GaussianCM = confusion_matrix(y_test,y_pred)


print("F1 score with Bernoulli Naive Bayes: %.3f" %BernoulliF1Score)
print("F1 score with Gaussian Naive Bayes: %.3f" %GaussianF1Score)

#Plotting Confusion Matrices
ax=sns.heatmap(BernoulliCM, annot=True, linewidths=.5,cmap="YlGnBu")
ax.set_xlabel('Predicted Class')
ax.set_ylabel('True Class')
ax.set_title('Confusion Matrix for Bernoulli Naive Bayes Classifier')
plt.show();

ax=sns.heatmap(GaussianCM, annot=True, linewidths=.5,cmap="YlGnBu")
ax.set_xlabel('Predicted Class')
ax.set_ylabel('True Class')
ax.set_title('Confusion Matrix for Gaussian Naive Bayes Classifier')
plt.show();
```

```
F1 score with Bernoulli Naive Bayes: 0.797
F1 score with Gaussian Naive Bayes: 0.814
```





Summarize your findings and results here

In [12]:

```
print("F1 score with Bernoulli Naive Bayes: %.3f" %BernoulliF1Score)
print("F1 score with Gaussian Naive Bayes: %.3f" %GaussianF1Score)
```

```
F1 score with Bernoulli Naive Bayes: 0.797
F1 score with Gaussian Naive Bayes: 0.814
```

Gaussian Naive Bayes performs better (better F1 score) than Bernoulli Naive Bayes on the given data.