kNN vs Linear Regression

Example I

Data Generation

Set the model parameters like dimension and standard error, and store the two centers (each is a two-dim vector) in m1 and m2.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier as knn
from sklearn import linear_model as lm

np.random.seed(100)

p = 2
s = 1      # sd for generating x
n = 100    # 2n obs (in total); n from each class
m1 = [1,0]
m0 = [0,1]
```

Generate n samples from each normal component. First, we generate a (2n)-by-p matrix with entries being iid samples form a normal dist with mean 0 and variance s-square. Then we form a (2n)-by-p matrix with the first n rows being the two-dimensional vector m1 (the center of the first normal component) and the next n rows being m2. We use command rep to generate repeated rows and use command rbind to stack two matrices vertically together.

In [7]:
```
n = 100  # generate the train data; n for each class
traindata = np.random.normal(size = (2 * n, p)) * s \
            + np.concatenate([np.array([m1] * n), np.array([m0] * n)])
Ytrain = np.concatenate(([1]*n, [0]*n))
```

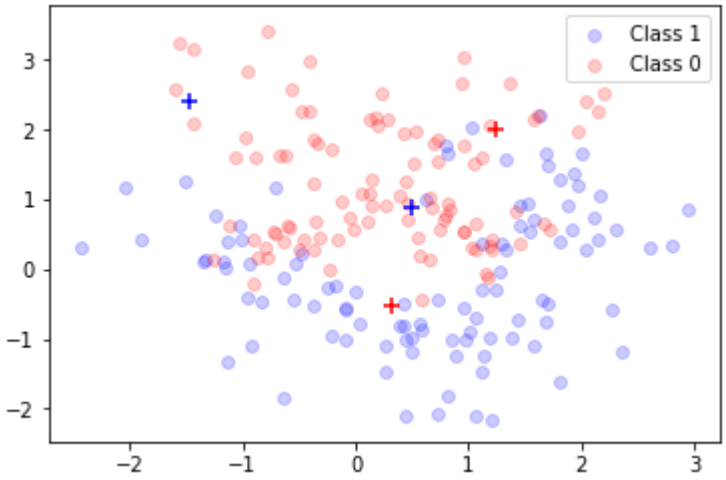
Generate 2N test samples similarly.

In [4]:
```
N = 5000
testdata = np.random.normal(size = (2 * N, p)) * s \
           + np.concatenate([np.array([m1] * N), np.array([m0] * N)])
Ytest = np.concatenate(([1]*N, [0]*N))
```

Visulization

Let's take a look of the data. In the plot generated by the code below, points from two groups are colored in red and blue, respectively; the two centers are plotted as +, and a legend is added to explain the association of each color.

In [85]:
```
plt.scatter(traindata[:n, 0], traindata[:n, 1], c = "blue", alpha=0.2, label=
'Class 1')
plt.scatter(traindata[n:, 0], traindata[n:, 1], c = "red", alpha=0.2, label='C
lass 0')

plt.scatter(m1[0], m1[1], marker = '+', s = 80, c = "blue")
plt.scatter(m0[0], m0[1], marker = '+', s = 80, c = "red")

plt.legend()
plt.show()
```



K-NN method

For choices of the neighhood size, I just use the values from the textbook.

In [8]:
```
myk = [151, 101, 69, 45, 31, 21, 11, 7, 5, 3, 1]

train_err_knn = []
test_err_knn = []

for j in myk:
    myknn = knn(n_neighbors = j)
    myknn.fit(traindata, Ytrain)
    train_err_knn.append(1 - myknn.score(traindata, Ytrain))
    test_err_knn.append(1- myknn.score(testdata, Ytest))
```

Least Sqaure Method

In [10]:
```
RegModel = lm.LinearRegression()
RegModel.fit(traindata, Ytrain)

Ytrain_pred_LS = RegModel.predict(traindata)
Ytest_pred_LS = RegModel.predict(testdata)

Ytrain_pred_LS = [1 if i >= 0.5 else 0 for i in Ytrain_pred_LS]
Ytest_pred_LS = [1 if i >= 0.5 else 0 for i in Ytest_pred_LS]

train_err_LS = sum(Ytrain != Ytrain_pred_LS) / float(2*n)
test_err_LS = sum(Ytest != Ytest_pred_LS) / float(2*N)
```

Bayes Error

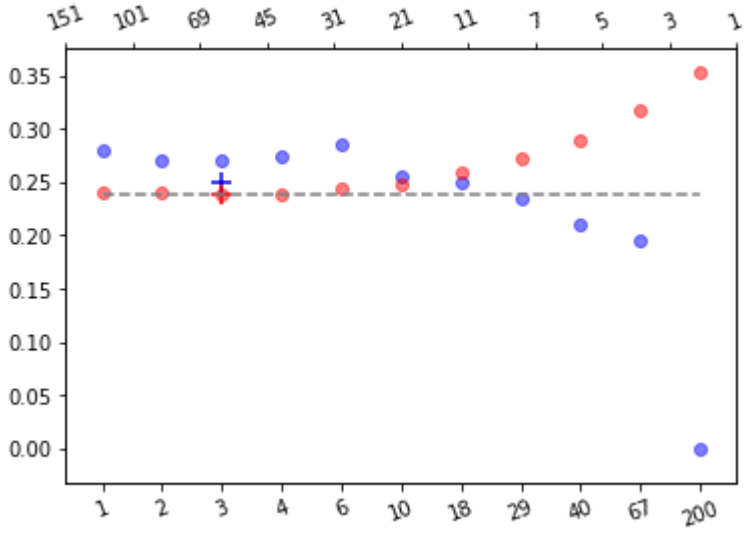
In [11]:
```
def myfun(x):
    return 2*np.dot(x, np.subtract(m1, m0)) - (np.dot(m1, m1) - np.dot(m0, m0
))

Ytest_pred_Bayes = [myfun(x) > 0 for x in testdata]
test_err_Bayes = sum(Ytest !=  Ytest_pred_Bayes) / float(2*N)
```

Plot the Performance

Test errors are in red and training errors are in blue. The upper x-coordinate indicates the K values, and the lower x-coordinate indicates the degree-of-freedom of the KNN procedures so the labels are reciprocally related to K.

The training and test errors for linear regression are plotted at df=3, since the linear model has 3 parameters, i.e., 3 dfs.

In [81]:
```
m = len(myk)
df = np.rint((2*n) / np.array(myk))
df = df.astype(int)

fig, ax1 = plt.subplots()

plt.scatter(range(m), train_err_knn, alpha = 0.5, c = "blue")
plt.scatter(range(m), test_err_knn, alpha = 0.5, c = "red")

plt.scatter(2, train_err_LS, marker = '+', s = 100, c = "blue")
plt.scatter(2, test_err_LS, marker = '+',  s = 100, c = "red")

plt.hlines(test_err_Bayes, 0, m-1, color = "gray", linestyle='--')

plt.xticks(range(m), df, rotation = 20)
ax2 = ax1.twiny()
ax2.set_xticks(range(m))
ax2.set_xticklabels(myk, rotation = 20)

plt.show()
```



Example II

Data Generation

Generate the 20 centers, 10 for each group.

In [82]:
```
csize = 10
p = 2
s = 1;      # sd for generating the centers within each class
m1 = np.random.normal(size = (csize, p)) * s \
     + np.concatenate([np.array([[1, 0]] * csize)])
m0 = np.random.normal(size = (csize, p)) * s \
     + np.concatenate([np.array([[0, 1]] * csize)])
```

Generate training data.

In [83]:
```
n=100;
# Randomly allocate the n samples for class 1  to the 10 clusters
id1 = np.random.randint(csize, size = n)
id0 = np.random.randint(csize, size = n)


# sd for generating x
s = np.sqrt(float(1)/5)

traindata = np.random.normal(size = (2 * n, p)) * s \
            + np.concatenate([m1[id1,:], m0[id0,:]])
ytrain = np.concatenate(([1]*n, [0]*n))
```

Visulization

In [84]:
```
plt.scatter(traindata[:n, 0], traindata[:n, 1], c = "blue", alpha=0.2, label=
'Class 1')
plt.scatter(traindata[n:, 0], traindata[n:, 1], c = "red", alpha=0.2, label='C
lass 0')

plt.scatter(m1[:,0], m1[:,1], marker = '+', s = 100, c = "blue")
plt.scatter(m0[:,0], m0[:,1], marker = '+', s = 100, c = "red")

plt.legend()
plt.show()
```