Brenden Hein

CSE 802

Dr. Ross

<u>Homework 4</u>

1. Let x be a d-dimensional binary (0 or 1) vector with a multivariate Bernoulli distribution…
   a. $P(x|\theta) = \prod_{i=1}^{d} \theta^{xi}(1-\theta_i)^{1-xi}$

$$\ln(P(x_k|\theta)) = \ln(\theta^{x_k}(1-\theta)^{1-x_k})$$

$$\ln(P(x_k|\theta)) = x_k \ln(\theta) + (1-x_k)\ln(1-\theta)$$

$$\frac{d}{d\theta} = \frac{x_k}{\theta} - \frac{1-x_k}{1-\theta} = \sum_{k=1}^{n} \frac{x_k}{\theta} - \frac{1-x_k}{1-\theta} = 0$$

$$\sum_{k=1}^{n} \frac{x_k}{\theta} - \sum_{k=1}^{n} \frac{1-x_k}{1-\theta} = 0$$

$$\frac{1}{\theta}\sum_{k=1}^{n} x_k + \frac{1}{1-\theta}\sum_{k=1}^{n} x_k - \frac{n}{1-\theta} = 0$$

$$\left(\frac{1}{\theta} + \frac{1}{1-\theta}\right)\sum_{k=1}^{n} x_k = \frac{n}{1-\theta}$$

$$\left(\frac{(1-\theta)}{\theta(1-\theta)} + \frac{(\theta)}{\theta(1-\theta)}\right)\sum_{k=1}^{n} x_k = \frac{n}{1-\theta}$$

$$\frac{1-\theta+\theta}{\theta(1-\theta)}\sum_{k=1}^{n} x_k = \frac{n}{1-\theta}$$

$$\sum_{k=1}^{n} x_k = \frac{n}{1-\theta} * \frac{\theta(1-\theta)}{1}$$

$$\sum_{k=1}^{n} x_k = \frac{n}{1} * \frac{\theta}{1} \;\; => \;\; \boldsymbol{\theta} = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{k=1}}^{\boldsymbol{n}} \boldsymbol{x_k}$$

2. From the IMOX dataset….
   a. Using PCA, we reduce the 8-dimensional features to 2 features using the top 2 eigenvectors

```
[[-0.21534345  0.01447815]
 [-0.40181854 -0.08829337]
 [-0.27405584 -0.16398584]
 [-0.00509032 -0.05035686]
 [-0.37641963 -0.34025534]
 [-0.50488797 -0.47580712]
 [ 0.39970198 -0.55720747]
 [ 0.4008382  -0.55670568]]
```

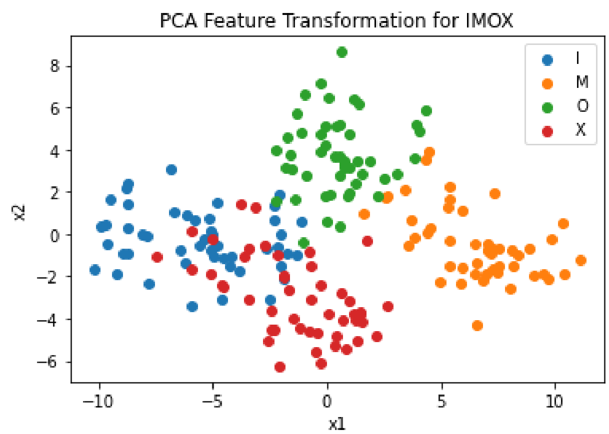

Fig. 1. – IMOX dataset reduced to d=2 using PCA

b. Using MDA, we reduce the 8-dimensional features to 2 features using the top 2 eigenvectors

```
[[-0.13698936  0.0164636 ]
 [-0.27737427 -0.06858576]
 [-0.18905829 -0.0546345 ]
 [-0.00137855 -0.00256901]
 [-0.5329115   0.11993169]
 [-0.75636344  0.05007692]
 [-0.02105779  0.58741996]
 [ 0.10967563  0.79377572]]
```

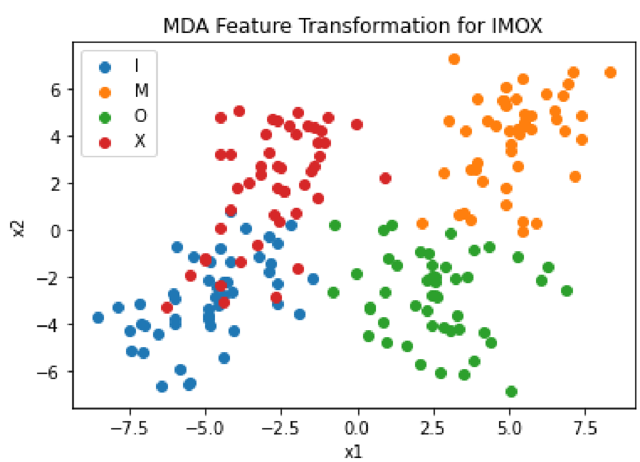

Fig. 2. – IMOX dataset reduced to d=2 using MDA

c. MDA appears to have better separation for the 4 classes. The reason for this is because it takes into account the differences between classes in the form of the scatter matrix, SB. So, unlike PCA where points from different classes are grouped together and can be pushed onto each other/overlap, MDA's taking into account the difference in classes yields better results.

3. For 2 feature selection algorithms…

   a. ***SBS***

      Input: $y = \{y_1, y_2, y_3, \dots y_d\}$
      Output: $x_k = \{x_1, x_2, x_3, \dots x_k\}$
      $x_j \in y;\ k = 0, 1, \dots d$

      <u>Initialization</u>
      $x_o = \{y_1, y_2, y_3, \dots y_d\}$
      $k = d$

      <u>Step 1</u>
      $x^- = argmax_{x \in x_k} J(x_k - x)$
      $x_{k-1} = x_k - x^-$
      $k = k - 1$
      $GO\ TO\ step\ 1$

      <u>Termination</u>
      $k = number\ of\ features\ to\ acquire$

   b. ***SBFS***

      Input: : $y = \{y_1, y_2, y_3, \dots y_d\}$
      Output: $x_k = \{x_1, x_2, x_3, \dots x_k\}$
      $x_j \in y;\ k = 0, 1, \dots d$

      <u>Initialization</u>
      $x_o = \{y_1, y_2, y_3, \dots y_d\}$
      $k = d$

      <u>Step 1 (exclusion)</u>
      $x^- = argmax_{x \in x_k} J(x_k - x)$
      $x_{k-1} = x_k - x^-$
      $k = k - 1$

      <u>Step 2 (conditional inclusion)</u>
      $x^+ = argmax_{x \in y - x_k} J(x_k + x)$
      $if\ (J(x_k + \{x^+\}) > \ J(x_{k+1}))$:
          $x_{k+1} = x_k + x^+$
          $k = k + 1$
          $GO\ TO\ step\ 2$
      $else$:
          $GO\ TO\ step\ 1$

      <u>Termination</u>
      $k = number\ of\ features\ to\ acquire$

4. Given a classifier using 15 features, trying to reduce the dimensionality to 5 or less
    a. SFS where xi is an arbitrary feature and not related to ordering
    $\{x_1\}$: 15 steps
    $\{x_1, x_2\}$: 14 steps
    $\{x_1, x_2, x_3\}$: 13 steps
    $\{x_1, x_2, x_3, x_4\}$: 12 steps
    $\{x_1, x_2, x_3, x_4, x_5\}$: 11 steps
    65 Steps
    b. (+l, -r) with (l, r) = (5, 3) where xi is an arbitrary feature and not related to ordering
    $\{x_1\}$: 15 steps
    $\{x_1, x_2\}$: 14 steps
    $\{x_1, x_2, x_3\}$: 13 steps
    $\{x_1, x_2, x_3, x_4\}$: 12 steps
    $\{x_1, x_2, x_3, x_4, x_5\}$: 11 steps
    $\{x_1, x_2, x_3, x_4\}$: 5 steps
    $\{x_1, x_2, x_3\}$: 4 steps
    $\{x_1, x_2\}$: 3 steps
    $\{x_1, x_2, x_3\}$: 13 steps
    $\{x_1, x_2, x_3, x_4\}$: 12 steps
    $\{x_1, x_2, x_3, x_4, x_5\}$: 11 steps
    $\{x_1, x_2, x_3, x_4, x_5, x_6\}$: 10 steps
    $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$: 9 steps
    $\{x_1, x_2, x_3, x_4, x_5, x_6\}$: 7 steps
    $\{x_1, x_2, x_3, x_4, x_5\}$: 6 steps
    $\{x_1, x_2, x_3, x_4\}$: 5 steps
    $\{x_1, x_2, x_3, x_4, x_5\}$: 11 steps
    161 Steps
    c. SBS where xi is an arbitrary feature and not related to ordering
    $\{x_1, x_2, \dots x_{14}\}$: 15 steps
    $\{x_1, x_2, \dots x_{13}\}$: 14 steps
    $\{x_1, x_2, \dots x_{12}\}$: 13 steps
    $\{x_1, x_2, \dots x_{11}\}$: 12 steps
    $\{x_1, x_2, \dots x_{10}\}$: 11 steps
    $\{x_1, x_2, \dots x_9\}$: 10 steps
    $\{x_1, x_2, \dots x_8\}$: 9 steps
    $\{x_1, x_2, \dots x_7\}$: 8 steps
    $\{x_1, x_2, \dots x_6\}$: 7 steps
    $\{x_1, x_2, \dots x_5\}$: 6 steps // Keep going because we're looking for 5 or less features
    $\{x_1, x_2, \dots x_4\}$: 5 steps
    $\{x_1, x_2, x_3\}$: 4 steps
    $\{x_1, x_2\}$: 3 steps
    $\{x_1\}$: 2 steps
    119 Steps

d. Exhaustive Search

$$C_5^{15} + C_4^{15} + C_3^{15} + C_2^{15} + C_1^{15} = \frac{15!}{5!(15-5)!} + \frac{15!}{4!(15-4)!} + \frac{15!}{3!(15-3)!} + \frac{15!}{2!(15-2)!} + \frac{15!}{1!(15-1)!} +$$

$$C = \frac{15*14*13*12*11*10!}{5*4*3*2*10!} + \frac{15*14*13*12*11!}{4*3*2*11!} + \frac{15*14*13*12!}{3*2*12!} + \frac{15*14*13!}{2*13!} + \frac{15*14!}{14!}$$

$$C = 3003 + 1365 + 455 + 105 + 15$$

4943 Steps

5. Using a Gaussian kernel function with N(20, 5) and N(35,5)

   a. Using 200 randomly generated points from 2 distributions

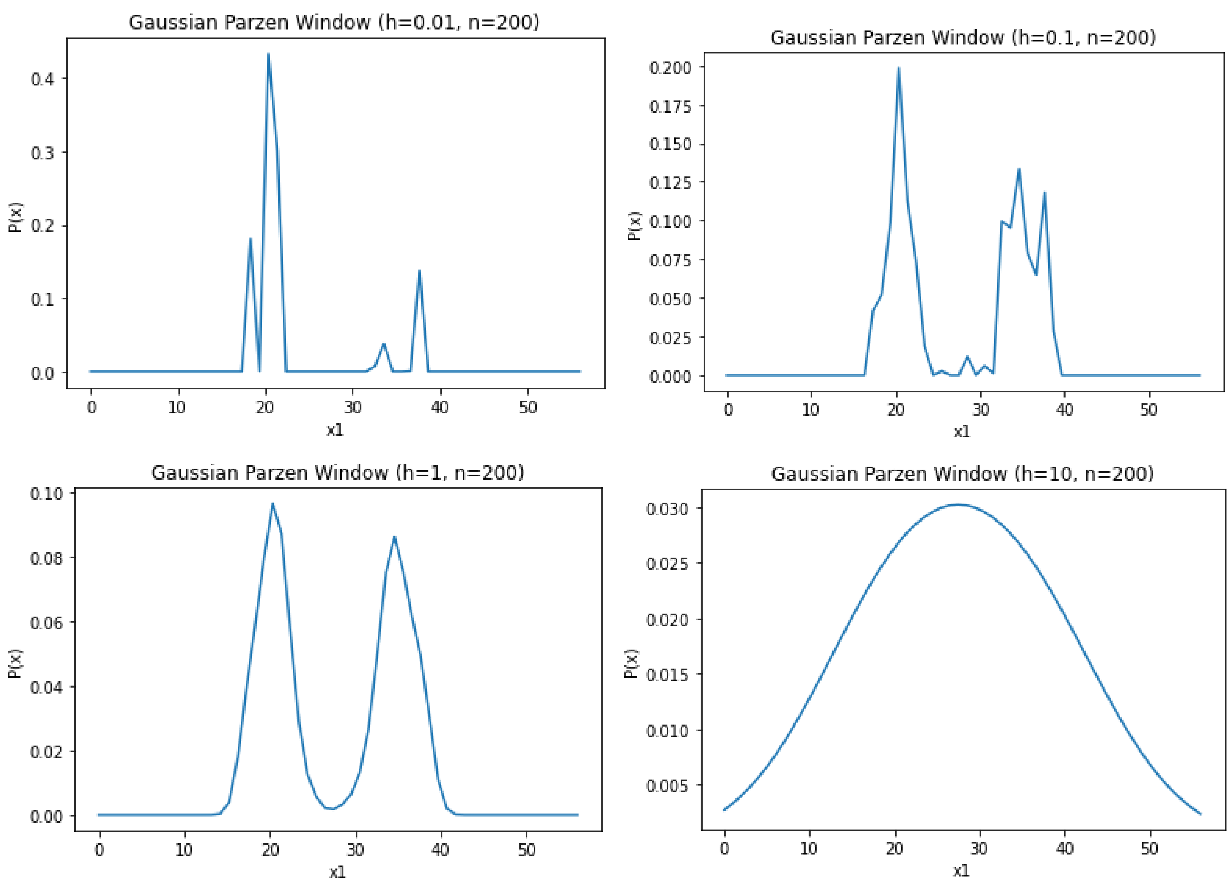

Fig. 3. The estimated distribution by using a gaussian kernel function with n=200 and window sizes [.01,.1,1,10]

b. Using 1000 randomly generated points from 2 distributions

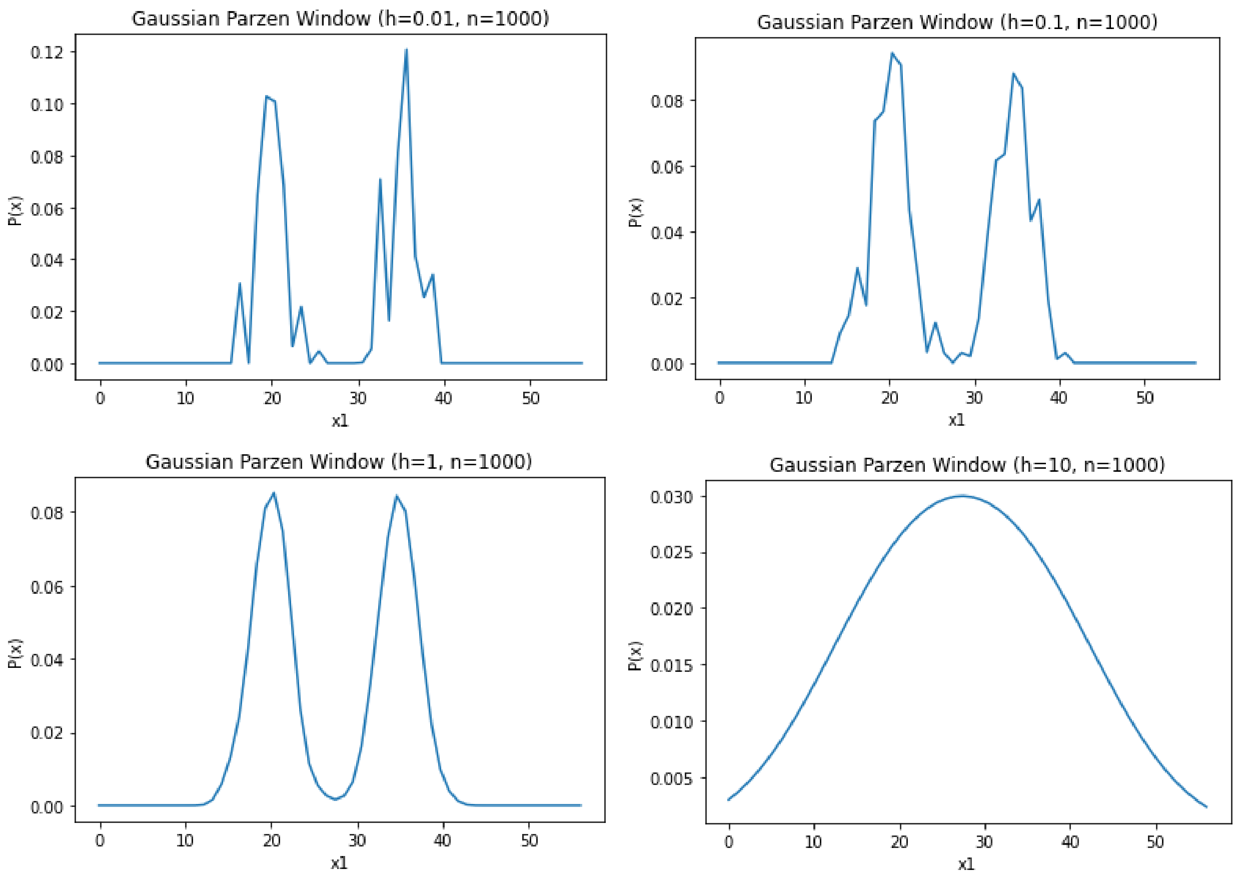

Fig. 4. The estimated distribution by using a gaussian kernel function with n=1000 and window sizes [.01,.1,1,10]

Using 2000 randomly generated points from 2 distributions

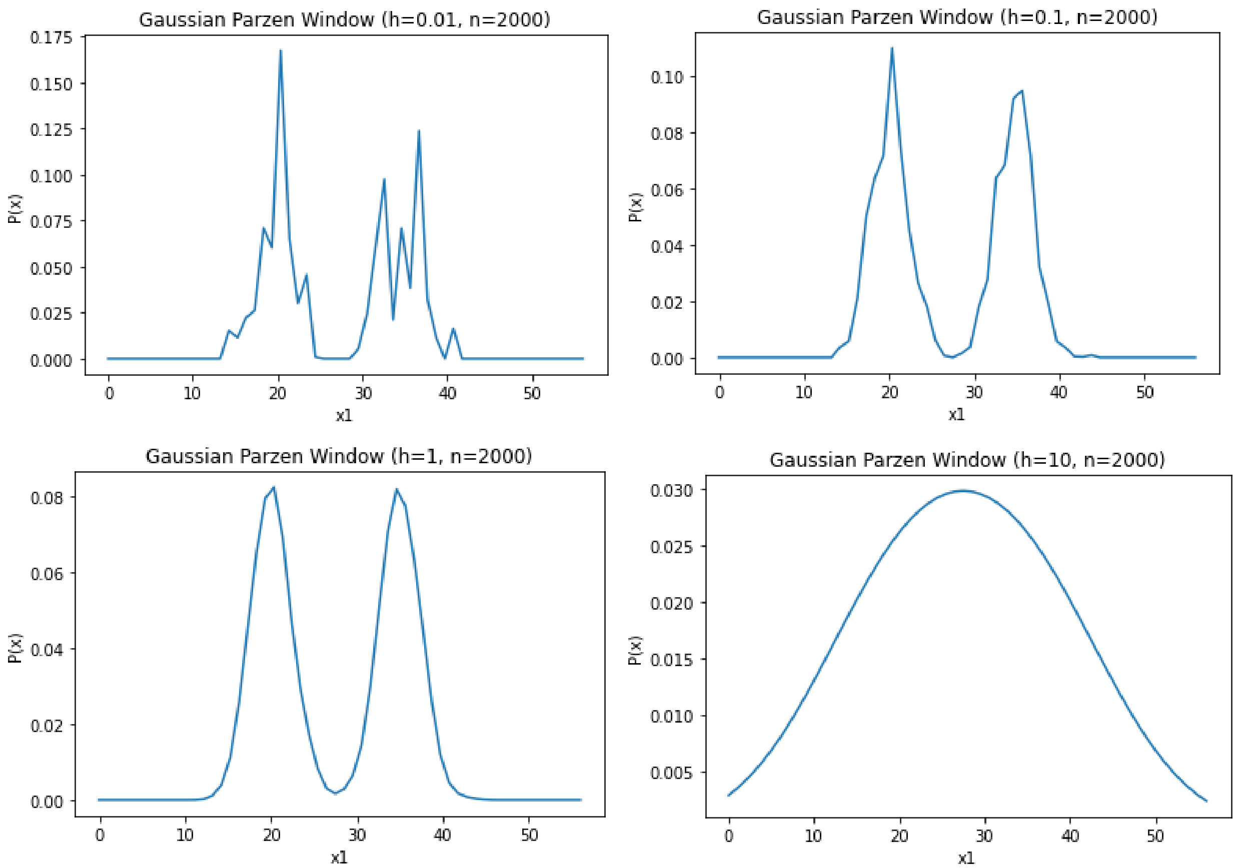

Fig. 5. The estimated distribution by using a gaussian kernel function with n=2000 and window sizes [.01,.1,1,10]

c. There are 2 things to note with these graphs. First, there is a clear cutoff point for a window that is too big. When the window width is large, the 2 distinct distributions start to blend, as the windows start to capture the other distribution as well as their own. The other thing is that as n increases, the distributions tend to look more normal and smoother, as with a higher n, the density is more accurately estimated.
$\lim_{n\to\infty} P_n(x) = P(x)$

6. Given a training set of 3 classes and 500 points for each class…
   a. The confusion matrix using the bivariate gaussian models

```
[[230   4  16]
 [  1 235  14]
 [  9  11 230]]
```

And an error rate of **.0733**

b. The confusion matrix using MLE estimates

```
[[231   4  15]
 [  1 238  11]
 [ 12  13 225]]
```

And an error rate of **.0747**

c. The confusion matrix using

```
[[228   5  17]
 [  1 238  11]
 [ 10  13 227]]
```

And an error rate of **.076**

d. When starting with known distributions and parameters, the error rate was at its lowest. However, when introducing more unknowns into the problem, the error rate increased. Despite this, the increase in error rate was minimal as we went from a to b to c. This is a good thing, as in many real-life pattern classification problems, the true parameters and the form of the distributions are unknown, so being able to estimate to such accuracy is important.

7. Using a K-NN classifier to classify datapoints for the iris dataset…
   a. For K=1,5,9,13,17,21 we get these confusion matrices

```
K=1
 [[25  0  0]
 [ 0 24  1]
 [ 0  3 22]]
```

```
K=5
 [[25  0  0]
 [ 0 23  2]
 [ 0  4 21]]
```

```
K=9
 [[25  0  0]
 [ 0 24  1]
 [ 0  3 22]]
```

```
K=13
 [[25  0  0]
 [ 0 24  1]
 [ 0  3 22]]
```

```
K=17
 [[25  0  0]
 [ 0 25  0]
 [ 0  3 22]]
```

```
K=21
 [[25  0  0]
 [ 0 25  0]
 [ 0  4 21]]
```

   b. The classification accuracy for the K values seemed to hover around 94-95% accuracy, with a jump down when k=5 and a smaller jump up when k=17. One thing I noticed was how setosa flowers (class 1) were classified perfectly for all k-values, which points to their having more unique feature sets than the other 2 classes (versicolor and virginica). Also, virginica appeared to be the most inaccurately classified, however, it was minor.

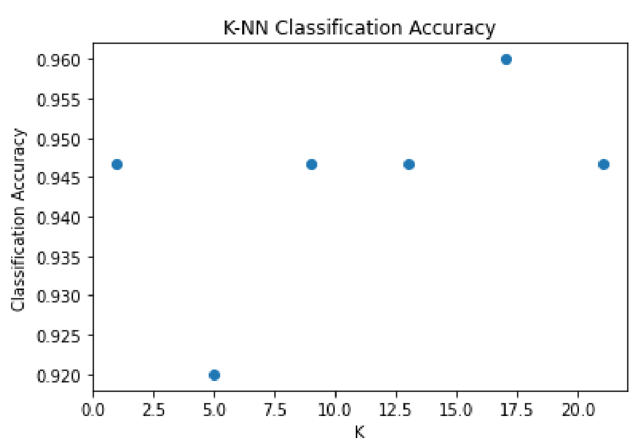

Fig. 6. The classification accuracy using K-NN for K sizes = 1, 5, 9, … 21

Appendix

- Q2

```
import matplotlib.pyplot as plt
import numpy as np
import itertools

IMOX = ["I", "M", "O", "X"]


def plot(imox, E, mu, name):
    """Plots the best 2 features using PCA feature transformation"""
    fig1, ax1 = plt.subplots()
    for w, features in enumerate(imox):
        x1, x2 = [], []
        for f in features:
            y = (E.T).dot(f - mu)
            x1.append(y[0])
            x2.append(y[1])
        ax1.scatter(x1, x2, label=IMOX[w])

    ax1.set_xlabel("x1")
    ax1.set_ylabel("x2")
    ax1.set_title(name + " Feature Transformation for IMOX")
    ax1.legend()


def main():
    # Gets the data from the IMOX file and put it into a dict
    fp, imox = open("imox_data.txt"), [[], [], [], []]
    for line in fp:
        line = line.split()
        w, features = int(line[-1]), [float(el) for el in line[:-1]]
        imox[w - 1].append(features)

    # Finds the scatter of the dataset
    imox_all = list(itertools.chain.from_iterable(imox))
    mu_o = np.mean(imox_all, axis=0)
    S = sum([np.outer((el - mu_o), (el - mu_o)) for el in imox_all])

    # Uses PCA for feature tranformation
    S_eigval, S_eigvec = np.linalg.eigh(S)
    indices = (-S_eigval).argsort()[:2]
    E = S_eigvec[:, [indices[0], indices[1]]]
    print(E, end="\n\n")
    plot(imox, E, mu_o, "PCA")

    # Uses MDA for feature transformation
    SW, mus = [], []
    for w, features in enumerate(imox):
        mu = np.mean(features, axis=0)
        SW.append(sum([np.outer((el - mu), (el - mu)) for el in features]))
        mus.append(mu)
    SW = sum(SW)
    SB = sum([len(imox[i]) * (np.outer((mus[i] - mu_o), (mus[i] - mu_o))) \
              for i in range(len(imox))])

    # Picks the top eigen vectors besed on their eigen values
    S = np.dot(np.linalg.inv(SW), SB)
    S_eigval, S_eigvec = np.linalg.eigh(S)
    indices = (-S_eigval).argsort()[:2]
    E = S_eigvec[:, [indices[0], indices[1]]]
    print(E, end="\n\n")
    plot(imox, E, mu_o, "MDA")


if __name__ == "__main__":
    main()
```

- Q5 – Generate

```
import numpy as np
n = int(input("How many samples:"))
w1 = np.random.normal(20, 5**.5, n)
w2 = np.random.normal(35, 5**.5, n)
fp1 = open("gen1_" + str(n) + ".txt", "w")
fp2 = open("gen2_" + str(n) + ".txt", "w")
for x in w1:
    fp1.write(str(x)+"\n")
for x in w2:
    fp2.write(str(x)+"\n")
fp1.close()
fp2.close()
```

- Q5

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as sp


def hn(gen, h, x):
    """For a given set of points (from a normal distribution), the resulting
    estimatiated denisity for a given x using a gaussian kernel function"""
    px, dist = 0, sp.norm(0, 1)
    for v in gen:  # estimate the density of x using generated points
        px += dist.pdf((x - v) / h)
    return px / (len(gen) * h)


def plot_density(gen, h, i):
    """For 56 points(0-55), each points density is estimated using a guassian
    kernel funciton, and the reuslting 56 points are plotted"""
    x_s = np.linspace(0, 56, 56)
    parzen = [hn(gen, h, x) for x in x_s]  # estimate the density of these points

    fig, ax = plt.subplots()
    ax.set_xlabel("x1")
    ax.set_ylabel("P(x)")
    ax.set_title("Gaussian Parzen Window (h={}, n={})".format(h, int(i) * 2))
    ax.plot(x_s, parzen)


def main():
    for i in ["100", "500", "1000"]:
        gen = [float(l) for l in open("gen1_{}.txt".format(i))] + \
              [float(l) for l in open("gen2_{}.txt".format(i))]
        plot_density(gen, .01, i)
        plot_density(gen, .1, i)
        plot_density(gen, 1, i)
        plot_density(gen, 10, i)


if __name__ == "__main__":
    main()
```

- Q6

```
from operator import itemgetter
from sklearn.metrics import confusion_matrix
import numpy as np
import scipy.stats as sp


def hn(train, h, x):
    """For a given set of points (from a normal distribution), the resulting
    estimatiated denisity for a given x using a gaussian kernel function"""
    dist, densities = sp.multivariate_normal([0, 0], [[1, 0], [0, 1]]), []
    for w, patterns in enumerate(train):  # estimate the density of x using generated
points
        px = 0
        for v in patterns:
            px += dist.pdf((np.subtract(x, v)) / h)
        densities.append((px / (len(train) * h * h), w + 1))
    return densities


def error(predicted, actual):
    """Gets the confusion matrix and the error rate for the test set"""
    cm = confusion_matrix(actual, predicted)
    w = sum([cm[i][j] for i in range(len(cm)) for j in range(len(cm[i])) if i != j])
    print(cm, "\n")
    print("Error:", w / len(actual), "\n\n")


def compare_results(test, gaus1, gaus2, gaus3):
    """finds the predicted class given the different gaussian distributions"""
    predicted, actual = [], []
    for w, patterns in enumerate(test):
        for pat in patterns:
            best = max([(gaus1.pdf(pat), 1), (gaus2.pdf(pat), 2),
                        (gaus3.pdf(pat), 3)], key=itemgetter(0))
            predicted.append(best[1])
            actual.append(w + 1)
    error(predicted, actual)


def get_confuse(test, mus, covs):
    """Does the set up of the data to get the actual and predicted values of
    the classifier using the test set"""
    mu1, mu2, mu3 = mus[0], mus[1], mus[2]
    cov1, cov2, cov3 = covs[0], covs[1], covs[2]
    gaus1 = sp.multivariate_normal(mu1, cov1)
    gaus2 = sp.multivariate_normal(mu2, cov2)
    gaus3 = sp.multivariate_normal(mu3, cov3)
    compare_results(test, gaus1, gaus2, gaus3)


def part_b(train, test):
    mus, covs = [], []
    for w, patterns in enumerate(train):
        mus.append(np.mean(patterns, axis=0))
        covs.append(np.cov(patterns, rowvar=False))
    get_confuse(test, mus, covs)


def part_c(train, test):
    predicted, actual = [], []
    for w, patterns in enumerate(test):
        for pat in patterns:
            d_est = hn(train, 1, pat)
            best = max(d_est, key=itemgetter(0))[1]
            predicted.append(best)
            actual.append(w + 1)
    error(predicted, actual)
```

```python
def main():
    fp = open("hw04_data.txt")
    train, test = [[], [], []], [[], [], []]

    for pat_s in fp:
        pat = [float(f) for f in pat_s.split()[:-1]]
        w = int(pat_s[-2]) - 1
        if len(train[w]) > 249:
            test[w].append(pat)
        else:
            train[w].append(pat)

    get_confuse(test, [[0, 0], [10, 0], [5, 5]],
                [[[4, 0], [0, 4]], [[4, 0], [0, 4]], [[5, 0], [0, 5]]])
    part_b(train, test)
    part_c(train, test)


if __name__ == "__main__":
    main()
```

- Q7

```python
from operator import itemgetter
from sklearn.metrics import confusion_matrix
import numpy as np
import pprint as pp
import scipy.spatial.distance as sp
import matplotlib.pyplot as plt


def plot_class(ks, ws):
    """Plots the classification effectiveness as a function of k"""
    fig, ax = plt.subplots()
    ax.set_xlabel("K")
    ax.set_ylabel("Classification Accuracy")
    ax.set_title("K-NN Classification Accuracy")
    ax.scatter(ks, ws)


def classify(k, distances):
    """Using K-NN, classifies the testing points USING the training points"""
    distances_k = sorted(distances, key=itemgetter(0))[:k]
    counts = [0, 0, 0]
    for d in distances_k:
        counts[d[1]] += 1
    return max(enumerate(counts), key=itemgetter(1))[0] + 1


def knn(k, train, test):
    """Uses K-NN to classify each of the test points"""
    predicted, actual = [], []
    for test_w, test_patterns in enumerate(test):
        for test_x in test_patterns:
            distances = []
            for train_w, train_patterns in enumerate(train):
                for train_x in train_patterns:
                    distances.append((sp.euclidean(test_x, train_x), train_w))
            predicted.append(classify(k, distances))
            actual.append(test_w + 1)

    cm = confusion_matrix(actual, predicted)
    w = sum([cm[i][j] for i in range(len(cm)) for j in range(len(cm[i])) \
             if i == j]) / len(actual)
    print("K={}, error={}\n".format(k, round(w, 3)), cm, end="\n\n")
    return w


def main():
    print()
    fp, train, test = open("iris_data.txt"), [[], [], []], [[], [], []]
    for line in fp:
        line = line.strip().split()
        if line:
            if len(train[int(line[-1]) - 1]) < 25:
                train[int(line[-1]) - 1].append([float(l) for l in line[:-1]])
            else:
                test[int(line[-1]) - 1].append([float(l) for l in line[:-1]])

    ws, ks = [], [i for i in range(1, 25, 4)]
    for k in ks:
        ws.append(knn(k, train, test))
    plot_class(ks, ws)


if __name__ == "__main__":
    main()
```