ASSIGNMENT 1: [IFT6390]

LÉA RICARD & JOSEPH D. VIVIANO

1. Probabilities

A percentage of 1.5 of women in their 40s who take a routine test (mammogram) have breast cancer. Among women that have breast cancer, there is a 87% chance that the test is positive. In women that do not have breast cancer, there is a probability of 9.6 that the test is positive.A woman in her forties who has passed this routine test receives a positive test result. What is the probability that it is actually breast cancer?
$P(D)$ = the probability of having disease.
$P(T)$ = the probability of having a positive test result.

- 1.5% of women in their 40s have breast cancer, therefore $P(D) = 0.015$.
- 87% true positive rate, therefore $P(T|D) = 0.87$.
- 9.6% false positive rate, therefore $P(T|\neg D) = 0.096$.

We want to know:

$$P(D|T) = \frac{P(T|D)P(D)}{P(T)} \tag{1.1}$$

Which means we need to calculate $P(T)$ which is

$$P(T) = P(T|D)P(D)+P(T|\neg D)P(\neg D) = 0.87\times0.015+0.096\times(1-0.015) \approx 0.1076 \tag{1.2}$$

$$P(D|T) = \frac{0.87 \times 0.015}{0.1076} \approx 0.1213 \tag{1.3}$$

2. Curse of Dimensionality

2.1. *Consider a hypercube in dimension d with side length c. What is the volume V?*

In the 2-dimensional case, $area = c^2$. In the 3 dimensional case, $V = c^3$. In the n-dimensional case, $V = c^d$.

Date: Sept 2018.

2.2. *X is a random vector of dimension d ($x \in d$) distributed uniformly within the hypercube (the probability density $p(x) = 0$ for all x outside the cube). What is the probability density function $p(x)$ for x inside the cube? Indicate which property(ies) of probability densities functions allow you to calculate this result.*

For all probability distributions:

$$\int_{-\inf}^{\inf} p(x) = 1 \tag{2.1}$$

We know $p(x) = 0$ for all points outside of the hypercube. Therefore, the probability of being in a particular point inside the hypercube is

$$p(x) = \frac{1}{c^d} \tag{2.2}$$

2.3. *Consider the outer shell (border) of the hypercube of width 3% of c (covering the part of the hypercube extending from the faces of the cube and 0.03c inwards). For example, if c = 100cm, the border will be 3cm (left, right, top, etc ...) and will delimit this way a second (inner) hypercube of side 100 − 3 − 3 = 94cm. If we generate a point x according to the previously defined probability distribution (by sampling), what is the probability that it falls in the border area? What is the probability that it falls in the smaller hypercube?*

Let b be the amount to remove from the border on one side (i.e., left) of the outer hypercube. $p(x_{large}) = 1$. Therefore in the general case the probability we are in the smaller hypercube is:

$$p(x_{small}) = (c - 2b)^d / c^d \tag{2.3}$$

$$p(x_{border}) = 1 - p(x_{small}) \tag{2.4}$$

Therefore for the above example:

$$p(x_{small}) = (100 - 2 \times 3)^d / 100^d = 94^d / 100^d \tag{2.5}$$

And as before, the probability we are in the border is:

$$p(x_{border}) = 1 - p(x_{small}) \tag{2.6}$$

2.4. *Calculate the above for $d = 1, 2, 3, 5, 10, 100, 1000$.*

$$1 - 94^1/100^1 = 0.06 \tag{2.7}$$

$$94^2/100^2 = 0.1163 \tag{2.8}$$

$$94^3/100^3 = 0.1694 \tag{2.9}$$

$$94^5/100^5 = 0.2661 \tag{2.10}$$

$$94^{10}/100^{10} = 0.4614 \tag{2.11}$$

$$94^{100}/100^{100} = 0.9980 \tag{2.12}$$

$$94^{1000}/100^{1000} \approx 1 \tag{2.13}$$

2.5. *What do you conclude?*

When the dimension grows, the probability that x falls into the narrow border at the edge of the hypercube becomes more likely, which is contrary of our intuitions at lower dimensions.

3. Parametric Gaussian vs Parzen Window Density Estimation

3.1. **Isotropic Gaussian Distribution.** *Suppose we have trained the parameters of an isotropic Gaussian density function on D (by maximizing the likelihood) in order to estimate the probability density function.*

3.1.1. *Name these parameters and indicate their dimension.* The named parameters are $\mu \in \mathbb{R}^d$-long vector of means, and $\Sigma \in \mathbb{R}^{d \times d}$ covariance matrix, where n is the number of samples.

3.1.2. *If we learn these parameters using MLE, express the formula which will give us the value of the optimal parameters as a function of the data points in D..*

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{3.1}$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^{\mathrm{T}} \tag{3.2}$$

3.1.3. *What is the algorithmic complexity of this training method?* For the μ parameter, the algorithm complexity is in $\mathcal{O}(nd)$, since it is summing over the n vectors of d length. For the Σ parameter the algorithmic complexity is limited by the summation of n $d \times d$ matrices. Computing this final matrix requires n matrix multiplications of a row and column vectors, each of which is a d^2 operation, so the complexity is in $\mathcal{O}(nd^2)$.

3.1.4. *For a test point x, write the function that will give the probability density predicted at that point.*

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}\mathrm{e}^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{3.3}$$

3.1.5. *What is the algorithmic complexity for this prediction?* The cost of this operation is limited by the find the determinant of Σ and the inversion of Σ. Both of these operations take $\mathcal{O}(d^{2.373})$ assuming we use the Coppersmith–Winograd algorithm for the inversion and fast matrix multiplication for finding the determinant.

3.2. **Parzen windows with Isotropic Gaussian Kernels.**

3.2.1. *What does the "training/learning" phase of Parzen windows with a fixed sigma consist of?* If σ is fixed by the user, nothing is learned during training (the Gaussians are simply centered on each training data point and then summed to create a density). Basicially this algorithm learns to remember the data.

3.2.2. *Write the function that gives the probability density at point x.*

$$p(x) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{N}_{x,\sigma}(x) \tag{3.4}$$

Expanded becomes:

$$p(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{(2\pi)^{\frac{d}{2}}\sigma^d}\mathrm{e}^{-\frac{1}{2}\frac{d(x_{test},x_{train})^2}{\sigma^2}} \tag{3.5}$$

3.2.3. *What is the algorithmic complexity for calculating this prediction?* If we assume that calculating the distnace between two vectors of length d is $\mathcal{O}(d)$ (as it is for euclidean distance for example), then the total cost is $\mathcal{O}(dn)$, since we have to calculate the distance between x_{test} and all of the n data points x_{train}.

3.3. **Capacity/Expressivitiy.**

3.3.1. *Which one of these two approaches has the highest capacity?* The Parzen Gaussian is more expressive, because it can store information for every data point. The capacity of the algorithm grows as we give it more data points, this isn't true for the Gaussian distribution, which averages over all data points, so it has a fixed capacity for a given dimensionality, no matter how many training data the algorithm is shown.

3.3.2. *When are we likely to be overfitting?* Parzen windows with Isotropic Gaussian Kernels, in the case that we used a large number of training examples with a small σ would result in extreme memorization of the noise in the training data (i.e., overfitting).

3.3.3. *Why is sigma in Parzen windows is usually treated as a hyperparameter, whereas for parametric Gaussian density estimation it is usually treated as a parameter?* Because in parametric Gaussian density estimation, σ is learned from the data, while it is fixed for all data points when using Parzen windows.

3.4. **Empirical Risk.**

3.4.1. *Express the equation of a diagonal Gaussian density in R^d. Specify what are its parameters and their dimensions.*

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}\mathrm{e}^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}, \tag{3.6}$$

where the parameters are Σ (dimension $d \times d$) and μ (dimension d).

3.4.2. *Show that the components of a random vector following a diagonal Gaussian distribution are independent random variables.* Random variables are independent if the joint density is equal to the product of marginal densities, i.e., $p(x_1, x_2, ..., x_d) = p(x_1)p(x_2)...p(x_n)$. Let

$$\begin{aligned} X &= (X_1, X_2, ..., X_d)^T \\ \mu &= (\mu_1, \mu_2, ..., \mu_d)^T \\ \Sigma &= (\sigma_{ij})_{i=1,2,...,d; j=1,2,...,d} \end{aligned} \tag{3.7}$$

Since the components of the random vector follow a diagonal Gaussian distribution, Σ has the form:

$$\Sigma = \begin{bmatrix} \sigma_{11} & 0 & 0 & \dots & 0 \\ 0 & \sigma_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_{dd} \end{bmatrix}, where (\sigma_{ij} = 0)_{i \neq j} \tag{3.8}$$

and

$$\Sigma^{-1} = \begin{bmatrix} \sigma_{11}^{-1} & 0 & 0 & \dots & 0 \\ 0 & \sigma_{22}^{-1} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_{dd}^{-1} \end{bmatrix}, where (\sigma_{ij} = 0)_{i \neq j}. \tag{3.9}$$

Hence, we can change the exponent in the equation for a multidimensional gaussian:

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{|\Sigma|}} \mathrm{e}^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{3.10}$$

with

(3.11)

$$\begin{aligned}(X-\mu)^T \Sigma^{-1} (X-\mu) &= \left[(X_1-\mu_1)\sigma_{11}^{-1} + (X_2-\mu_2)\sigma_{22}^{-1} + ... + (X_d-\mu_d)\sigma_{dd}^{-1} \right] \begin{bmatrix} (X_1-\mu_1) \\ (X_2-\mu_2) \\ \vdots \\ (x_d-\mu_d) \end{bmatrix} \\ &= (X_1-\mu_1)^2\sigma_{11}^{-1} + (X_2-\mu_2)^2\sigma_{22}^{-1} + ... + (X_d-\mu_d)^2\sigma_{dd}^{-1} \\ &= \sum_{i=1}^{d} (X_i-\mu_i)^2 \sigma_{ii}^{-1}\end{aligned}$$

Since we know that the determinant of a diagonal matrix is equal to the product of all diagonal elements,

$$\begin{aligned} p(x) &= \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\prod_{i=1}^{d} \sigma_{ii}}} \mathrm{e}^{-\frac{1}{2}\sum_{n=1}^{N}(X_i-\mu_i)^2\sigma_{ii}^{-1}} \\ &= \prod_{i=1}^{d} \left(\left(\frac{1}{\sqrt{2\pi\sigma_{ii}}}\right) \mathrm{e}^{-\frac{1}{2}(X_i-\mu_i)^2\sigma_{ii}^{-1}} \right) \end{aligned} \tag{3.12}$$

which is the product of the marginal density of each variable (dimension d).

3.4.3. *Using $-$ log p(x) as the loss, write down the equation corresponding to the empirical risk minimization on the training set D (in order to learn the parameters).* The log-likelyhood function is:

$$log(D|\mu, \Sigma) = -\frac{N}{2} log|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}[(X_n - \mu)^T \Sigma^{-1}(X_n - \mu)] + c \tag{3.13}$$

Where c is a constant.

3.4.4. *Solve this equation analytically in order to obtain the optimal parameters.* Using the previous equation, we can derivate the log likelihood function in terms of μ and find the optimal parameters when this derivative is equal to zero:

$$\frac{\partial log(D|\mu, \Sigma)}{\partial \mu} = \sum_{i=1}^{N}(X_i - \mu)^T \Sigma^- 1 = 0 \tag{3.14}$$

$$\begin{gathered} 0 = N\mu - \sum_{i=1}^{N} X_i \\ \hat{\mu} = \frac{1/N}{\sum_{i=i}^{N} X_i} \end{gathered} \tag{3.15}$$

Where $\hat{\mu}$ is of dimension: $\mathbb{R}^d$.

Next we do the same to find :

$$\begin{aligned} \log(D|\mu, \Sigma) &\propto -\frac{N}{2}\log|\Sigma| - \frac{1}{2}\sum_{n=1}^{N} tr(\Sigma^{-1}(X_n - \mu)(N_n - \mu)^T) \\ &= -\frac{N}{2}\log|\Sigma| - \frac{1}{2} tr(\Sigma^{-1}\sum_{n=1}^{N}[(X_n - \mu)(N_n - \mu)^T]) \end{aligned} \tag{3.16}$$

Here we used the ‘trace trick’ to re-arrange the multiplied elements on the right side of the equation (if you have matrix multiplications that result in a scalar), one can use trace to rearrange the arguments), i.e., $UVU' = tr(VUU')$.

Also note that:

$$\begin{gathered} \frac{\partial}{\partial A} tr[AB] = B^T \\ \frac{\partial}{\partial A} log|A| = A^{-T} \end{gathered} \tag{3.17}$$

Which allows to isolate $[(X_n - \mu)(N_n - \mu)^T]$ when differentiating with respect to Σ^{-1}, and to transform $|\Sigma^{-1}|$ to Σ.

(3.18)

$$\begin{aligned}
\frac{\partial l(\mu, \Sigma|X_i)}{\partial \Sigma^{-1}} &= \frac{\partial}{\partial \Sigma^{-1}}\big(\frac{N}{2}log|\Sigma^{-1}|\big) + \frac{\partial}{\partial \Sigma^{-1}}\big(-\frac{1}{2}tr(\Sigma^{-1}\sum_{n=1}^{N}[(X_n-\mu)(X_n-\mu)^T])\big) = 0 \\
&\Leftrightarrow \frac{N}{2}\Sigma - \frac{1}{2}\sum_{i=1}^{N}(X_i-\mu)(X_i-\mu)^T = 0 \\
&\Leftrightarrow N\Sigma - \sum_{i=1}^{N}(X_i-\mu)(X_i-\mu)^T = 0 \\
&\Leftrightarrow \Sigma - \frac{1}{N}\sum_{i=1}^{N}(X_i-\mu)(X_i-\mu)^T = 0 \\
&\Leftrightarrow \hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(X_i-\hat{\mu})(X_i-\hat{\mu})^T
\end{aligned}$$

Where $\hat{\Sigma}$ is of dimension: $\mathbb{R}^{d\times d}$.

Université de Montréal
Email address: `leaa.ricard@gmail.com & joseph@viviano.ca`

Density Estimation

September 28, 2018

```
In [5]: import numpy as np
        import math
        %pylab inline
        import matplotlib.pyplot as plt

        class diagonal_gaussian_parametric:
            """
            1. Implement a diagonal Gaussian parametric density estimator. It
            will have to work for data of arbitrary dimension d. As seen in
            the labs, it should have a train() method to learn the parameters
            and a method predict() which calculates the log density.
            """
            def __init__(self):
                pass

            def train(self, train_data):

                # if only one test_data is passed, add dummy dimension
                if len(np.shape(train_data)) == 1:
                    train_data = np.expand_dims(train_data, axis=1)

                self.train_data = train_data

                try:
                    self.n, self.d = np.shape(self.train_data)
                except:
                    self.n = np.shape(self.train_data)
                    self.d = 1

                self.mu = np.sum(self.train_data, axis=0) / self.n
                self.sigma = np.cov(self.train_data.T) * np.eye(self.d)

            def predict(self, test_data, nll=True):

                # if only one test_data is passed, add dummy dimension
                if len(np.shape(test_data)) == 1:
                    test_data = np.expand_dims(test_data, axis=1).T
        self.test_data = test_data
        n_inputs = np.shape(test_data)[0]
        densities = np.zeros(n_inputs)

        sigma_inv = np.linalg.inv(self.sigma)
        sigma_det = np.linalg.det(self.sigma)

        normalizer = 1 / ((2* np.pi)**(self.d / 2) * np.sqrt(sigma_det))

        # we treat each input test_data independently
        for i in range(n_inputs):

            diff = self.test_data[i, :] - self.mu
            exponent = (-0.5) * (diff).T.dot(sigma_inv).dot(diff)

            if nll:
                p = -np.log(normalizer)*exponent
            else:
                p = normalizer*np.exp(exponent)

            # handle edge case where p(x)=0
            if p == 0:
                p = np.finfo(float).eps

            densities[i] = p

        return(densities)

class parzen_density_estimator:
    """
    2. Implement a Parzen density estimator with an isotropic Gaussian
    kernel. It will have to work for data of arbitrary dimension d.
    Likewise it should have a train() method and a predict() method
    that computes the log density.
    """
    def __init__(self):
        pass

    def train(self, train_data, sigma=0):
        self.train_data = np.expand_dims(train_data, axis=1)

        try:
            self.n, self.d = np.shape(self.train_data)
        except:
            self.n = np.shape(self.train_data)
            self.d = 1
        if sigma == 0:
            self.sigma = np.std(self.train_data) # std because isotropic Gaussian
        else:
            self.sigma = sigma

    def predict(self, test_data, nll=True):

        # if only one test_data is passed, add dummy dimension
        if len(np.shape(test_data)) == 1:
            test_data = np.expand_dims(test_data, axis=1).T

        self.test_data = test_data
        n_inputs = np.shape(self.test_data)[0]
        n_train = np.shape(self.train_data)[0]
        densities = np.zeros(n_inputs)
        normalizer = 1/((2*np.pi*self.sigma)**(self.d / 2))

        for i in range(n_inputs):

            # calculate avg dist b/t this training point and all test points
            p = 0
            for j in range(n_train):

                # for gaussian kernel we use euclidean distance
                distance = np.sum((self.test_data[i, :] - self.train_data[j, :])**2)
                exponent = (-0.5) * (distance / self.sigma)

                if nll:
                    p += -np.log(normalizer)*exponent
                else:
                    p += normalizer * np.exp(exponent)

            # handle edge case where p(x)=0
            if p == 0:
                p = np.finfo(float).eps

            # save the average kernel values across all training points
            densities[i] = p/n_train

        return(densities)
```

```
Populating the interactive namespace from numpy and matplotlib


/opt/python/sci_36/lib/python3.6/site-packages/IPython/core/magics/pylab.py:160: UserWarning: p
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

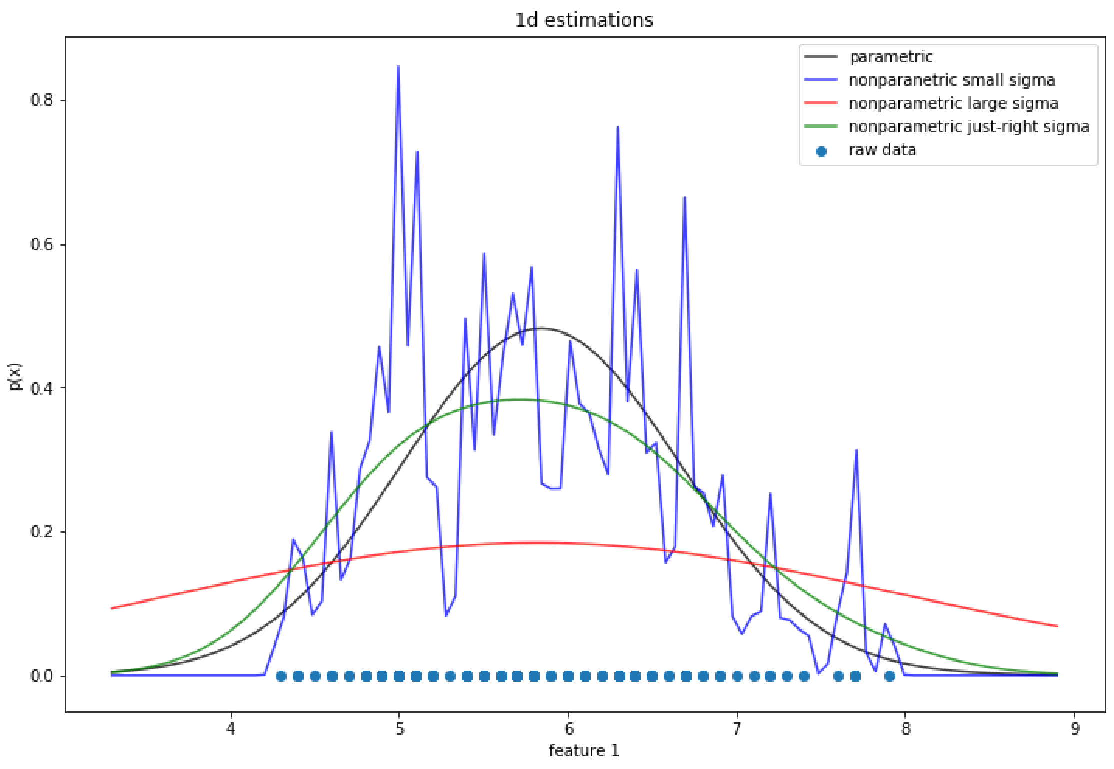
0.1 1D Densities

From the Iris dataset examples, choose a subset corresponding to one of the classes (of your choice), and one of the characteristic features, so that we will be in dimension d = 1 and produce a single graph (using the plot function) including:

- the data points of the subset (displayed on the x axis).
- a plot of the density estimated by your parametric Gaussian estimator.
- a plot of the density estimated by the Parzen estimator with a hyper-parameter (standard deviation) too small.
- a plot of the density estimated by the Parzen estimator with the hyper-parameter being a little too big.
- a plot of the density estimated by the Parzen estimator with the hyper-parameter that you consider more appropriate. Use a different color for each plot, and provide your graph with a clear legend.
- Explain how you chose your hyper-parameter .

```
In [6]: # load data
        iris = np.loadtxt("iris.txt")
        iris_subset = iris[:,0]
        n = len(iris_subset)

        # general plotting variables
        n_bins = 100
        axes_min = np.min(iris_subset)-1
        axes_max = np.max(iris_subset)+1
        alpha=0.75

        small_sig = 0.001
        large_sig = 4
        moyen_sig = 0.25

        x = np.atleast_2d(np.linspace(axes_min, axes_max, n_bins)).T

        # (a) -- raw data
        plt.figure(figsize=(12, 8))
        plt.scatter(iris_subset, np.zeros(len(iris_subset)))

        # (b) -- parametric model
        flower = diagonal_gaussian_parametric()
        flower.train(iris_subset)
        density = flower.predict(x, nll=False)
        plt.plot(x, density, color='black', alpha=alpha)

        # (c) -- nonparametric small sigma
        flower = parzen_density_estimator()
        flower.train(iris_subset, sigma=small_sig)
        density2 = flower.predict(x, nll=False)
        plt.plot(x, density2, color='blue', alpha=alpha)
```

```
# (d) -- nonparametric large sigma
flower = parzen_density_estimator()
flower.train(iris_subset, sigma=large_sig)
density = flower.predict(x, nll=False)
plt.plot(x, density, color='red', alpha=alpha)

# (e) -- nonparametric just-right sigma
flower = parzen_density_estimator()
flower.train(iris_subset, sigma=moyen_sig)
density = flower.predict(x, nll=False)
plt.plot(x, density, color='green', alpha=alpha)

# legend
plt.legend(['parametric',
            'nonparanetric small sigma',
            'nonparametric large sigma',
            'nonparametric just-right sigma',
            'raw data'])

plt.title('1d estimations')
plt.xlabel('feature 1')
plt.ylabel('p(x)')
plt.show()
```



We chose sigma based on how it looked on the plot. The very large sigma (4) was chosen to encompase lots of empty feature space and the very small sigma (0.001) was chosen to be very sensitive to each data point. For the 'just right' sigma (0.25), we chose a value that mostly covered the data distribution without it encompasing too much empty feature space.

0.2 2D Densities

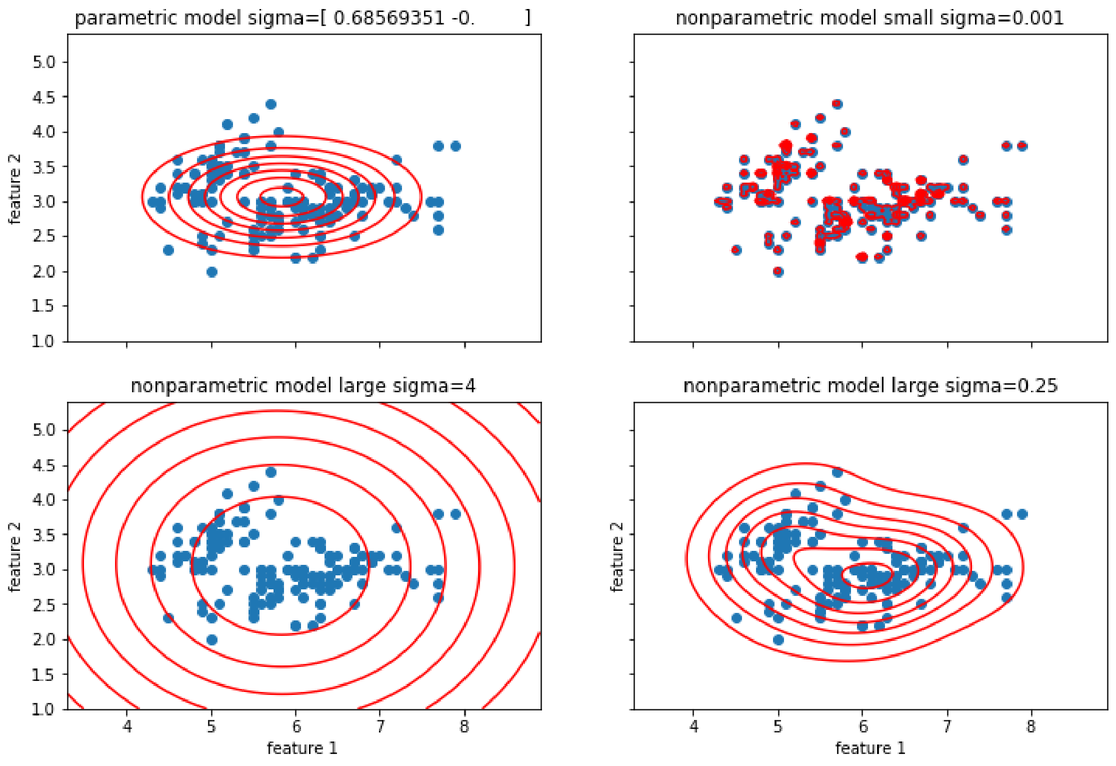
Now add a second characteristic feature of Iris, in order to have entries in d = 2 and produce 4 plots, each displaying the points of the subset of the data (with the plot function), and the contour lines of the density estimated (using the contour function):

- by the diagonal Gaussian parametric estimator.
- by the Parzen estimator with the hyper-parameter (standard deviation) being too small.
- by the Parzen estimator with the hyper-parameter being a little too big.
- by the Parzen estimator with the hyper-parameter that you consider more appropriate.
- Explain how you chose your hyper-parameter

```
In [7]: def make_dimensions(x_min, x_max, y_min, y_max, n_bins):
            X = np.atleast_2d(np.linspace(x_min, x_max, n_bins)).T
            Y = np.atleast_2d(np.linspace(y_min, y_max, n_bins)).T
            l = len(X) * len(Y)
            Z = np.zeros([l, 2])
            i = 0
            for x in X:
                for y in Y:
                    Z[i, 0] = x
                    Z[i, 1] = y
                    i += 1

            return X, Y, Z
```

```
In [9]: # new data subset
        iris_subset = iris[:,0:2]

        # general plotting variables
        n_bins = 100
        x_min = np.min(iris_subset[:,0])-1
        x_max = np.max(iris_subset[:,0])+1
        y_min = np.min(iris_subset[:,1])-1
        y_max = np.max(iris_subset[:,1])+1
        alpha=0.75

        # plot time
        f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 8), sharex=True,
                                                                          sharey=True)

        # add raw data
```

```
ax1.scatter(iris_subset[:, 0], iris_subset[:, 1])
ax2.scatter(iris_subset[:, 0], iris_subset[:, 1])
ax3.scatter(iris_subset[:, 0], iris_subset[:, 1])
ax4.scatter(iris_subset[:, 0], iris_subset[:, 1])

# (b) -- parametric model
flower = diagonal_gaussian_parametric()
flower.train(iris_subset)
X, Y, Z = make_dimensions(x_min, x_max, y_min, y_max, n_bins)
X, Y = np.meshgrid(X, Y)
density = flower.predict(Z, nll=False).reshape(n_bins, n_bins)
ax1.contour(X, Y, density.T, colors='red')
ax1.set_title('parametric model sigma={}'.format(flower.sigma[0]))
ax1.set_ylabel('feature 2')

# (c) -- nonparametric model, small sigma
flower = parzen_density_estimator()
flower.train(iris_subset, sigma=small_sig)
X, Y, Z = make_dimensions(x_min, x_max, y_min, y_max, n_bins)
X, Y = np.meshgrid(X, Y)
density = flower.predict(Z, nll=False).reshape(n_bins, n_bins)
ax2.contour(X, Y, density.T, colors='red')
ax2.set_title('nonparametric model small sigma={}'.format(small_sig))

# (d) -- nonparametric model, large sigma
flower = parzen_density_estimator()
flower.train(iris_subset, sigma=large_sig)
X, Y, Z = make_dimensions(x_min, x_max, y_min, y_max, n_bins)
X, Y = np.meshgrid(X, Y)
density = flower.predict(Z, nll=False).reshape(n_bins, n_bins)
ax3.contour(X, Y, density.T, colors='red')
ax3.set_title('nonparametric model large sigma={}'.format(large_sig))
ax3.set_xlabel('feature 1')
ax3.set_ylabel('feature 2')

# (e) -- nonparametric model, just right sigma
flower = parzen_density_estimator()
flower.train(iris_subset, sigma=moyen_sig)
X, Y, Z = make_dimensions(x_min, x_max, y_min, y_max, n_bins)
X, Y = np.meshgrid(X, Y)
density = flower.predict(Z, nll=False).reshape(n_bins, n_bins)
ax4.contour(X, Y, density.T, colors='red')
ax4.set_title('nonparametric model large sigma={}'.format(moyen_sig))
ax4.set_xlabel('feature 1')
ax4.set_ylabel('feature 2')

plt.show()
```



We chose sigma based on how it looked on the plot. The very large sigma (4) was chosen to encompase lots of empty feature space and the very small sigma (0.001) was chosen to tightly wrap each data point. For the 'just right' sigma (0.25), we chose a value that mostly covered the data distribution without it encompasing too much empty feature space.