IFT 6390 - Homework 1 - Solutions

October 11, 2018

Introduction

- Answers should consist of a report in the `.pdf` format supporeted by the appropriate code files `.ipynb` or `.py`. Answers in `.docx` or other format were given the grade 0. You should submit an archive (preferrably `.zip`) only if you have more than 3 files.
- For each sub-question, the answer should be between 1 and 10 lines long, as illustrated by this solution example. Questions are purposely kept simple and so should be the answers.

1 Small exercise on probabilities

Notation. Our group of study is entirely made of women in their 40s who took the routine test. For samples in this group, let us note C the event of having a breast cancer, and P the event that the test is positive. We respectively note $\neg C$ and $\neg P$ the complementary of these events : not having a cancer and having a negative test.

Data. The proportion of women with a cancer is $\mathbb{P}(C) = 0.015$. The *statistical power* of the test is $\mathbb{P}(P|C) = 0.87$. The *significance level* of the test is $\mathbb{P}(P|\neg C) = 0.096$. We want to know the probability of having a cancer given a positive test $\mathbb{P}(C|P)$.

Bayes Rule.

$$\mathbb{P}(C|P) = \frac{\mathbb{P}(P|C)\,\mathbb{P}(C)}{\mathbb{P}(P)} \tag{1}$$

where $\mathbb{P}(P) = \mathbb{P}(P \cap C) + \mathbb{P}(P \cap \neg C) = \mathbb{P}(P|C)\,\mathbb{P}(C) + \mathbb{P}(P|\neg C)\,\mathbb{P}(\neg C)$

Answer. Using a precise calculator gives: $\mathbb{P}(C|P) = 0.12127125731 \approx 0.12 = 12\%$ keeping only 2 digits. The correct answer is E. Doctors probably trusted the high statistical power of the test without worrying about its high significance level combined with the low proportion of cancers in the population.

2 Curse of dimensionality and geometric intuition in higher dimensions

1. The volume of the hypercube of dimension d with side length c is $V = \text{Vol}([0, c]^d) = c^d$

2. There are several valid definitions of a uniform distribution. Let's take this one: a uniform distribution over the hypercube H has a density $p(x) = k$ constant over H. This density should integrate to 1 over the space so:

$$1 = \int p(x)dx = \int_H kdx = k \int_H 1dx = kV \tag{2}$$

Where for the last equality we used that the volume of a set is defined as the integral of 1 over this set. It results that

$$\forall x \in H, p(x) = \frac{1}{V} = \frac{1}{c^d} \tag{3}$$

3. For a uniform density the probability of falling in a certain set S is proportional to its volume. This can be deduced from the definition above by $\mathbb{P}(X \in S) = \int_S kdx = \frac{\text{Vol}(S)}{V}$. So the probability of falling into the smaller hypercube $\bar{H}$ of side length $c - 2 \times 0.03c = 0.94c$ and volume $(0.94c)^d$ is

$$\mathbb{P}(X \in \bar{H}) = \frac{(0.94c)^d}{c^d} = 0.94^d \, . \tag{4}$$

The probability of falling into its complementary, the border B is

$$\mathbb{P}(X \in B) = \mathbb{P}(X \in H) - \mathbb{P}(X \in \bar{H}) = 1 - 0.94^d \, . \tag{5}$$

4. We round the results to two digits.

d	1	2	3	5	10	100	1000
$\mathbb{P}(X \in B)$	0.06	0.12	0.17	0.27	0.46	1.00	1.00

5. In higher dimensions, the probability of falling close from the border of the hypercube is very high. This is in contrast to dimension 1, 2 and 3 where this probability is quite low: it remains lower than 1 in 5 for the border width we picked.

[Optional] There are several ways to understand this phenomenon, which all reflect the calculation we did. One of them is to say that if each dimension of X is itself a sample, the odd of having at least one of these samples fall close to the border grows with the dimension. A bad explanation is that the density is no longer uniform or that points are more drawn towards the border in higher dimension. It's the opposite: the density remains uniform but the border grows bigger.

3 Parametric Gaussian density estimation, v.s. Parzen window density estimation

3.1

(a)

the parameters are the mean μ of dimensionality d and variance σ^2 of dimensionality 1

- also accepted the covariance matrix parameter Σ of dimensionality $d \times d$ if you mentioned that $\Sigma = \sigma^2 * I$
- *note:* x is **not** a parameter

(b)

$$\mu = \frac{1}{n} \sum_{i}^{n} x^i$$

for the variance, you can either use the general formula

$$\Sigma = \frac{1}{n} \sum_{i}^{n} (x^i - \mu)(x^i - \mu)^\intercal$$

alternatively, you could write the formula for σ^2

$$\sigma^2 = \frac{1}{dn} \sum_{i}^{n} \sum_{j}^{d} (x_j^i - \mu_j)^2$$

- *note*: don't forget to divide by d in the latter case!
- *note*: you can choose to use either Σ or σ^2 here but you have to be consistent across all of question 3.1

(c)

- the time complexity for μ is $O(nd)$ since you have to perform n additions for vectors of length d

if you used Σ

- the time complexity is $O(nd^2)$ since you are multiplying vectors to create matrices of size $d \times d$ and then averaging all n matrices.

if you used σ^2

- then the time complexity is $O(nd)$ since you have a subtraction and then a squaring of a scalar ($O(1)$) and you do this in a sum over all n and over all d

(d)

if you used Σ then generally

$$\hat{p}_{gauss-isotrop}(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\intercal \Sigma^{-1}(x-\mu)}$$

but if you used σ^2 then for isotropic

$$\hat{p}_{gauss-isotrop}(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sigma^d} e^{-\frac{1}{2}\frac{\|x-\mu\|^2}{\sigma^2}}$$

(e)

if you used Σ, this is $O(d^2)$

- because you need to calculate the matrix product with Σ which is $d \times d$

if you used σ^2, this is $O(d)$

- you only need to work with x which is of dimension d
- *note*: inverting Σ is $O(d)$ because we know it is isotropic, so this is simply $\Sigma \to \Sigma^{-1} \implies \sigma^2 \to \frac{1}{\sigma^2}$

3.2

(a)

the training portion is simply loading the training points into memory. I also accepted

- nothing
- memorizing the training data
- setting the means of the gaussians

(b)

$$\hat{p}_{parzen} = \frac{1}{n}\sum_{i}^{n} \frac{1}{(2\pi)^{\frac{d}{2}}\sigma^d} e^{-\frac{1}{2}\frac{\left\|x-x^{(i)}\right\|^2}{\sigma^2}}$$

(c)

similar to question 3.1.e, except you have to iterate over all n examples

- if you used Σ then $O(nd^2)$
- if you used σ^2 then $O(nd)$

3.3

(a)

A model's expressivity can be thought of as how many possible functions it can express. Parzen has more expressivity since it can be seen of as a combination of n fixed gaussians which can express much more complex functions (given enough examples n) than the parametric method which will be able to only express the space of isotropic gaussian functions.

- *note*: having hyperparameters is not an indicator of a model with more expressivity

(b)

Parzen is more likely to be overfitting in the case that we set σ^2 to be very small. In that situation, the parzen method does not find general patterns but puts a lot of weight to each point and the very small area around it.

(c)

The parzen's method σ^2 is a hyperparameter, which means we don't learn it on the dataset. Since it controls the capacity of the model, if we were to optimize σ^2 on the training data, we would always be making it as small as possible to fit the training data as closely as possible, but this would be overfitting! Instead, we can learn to optimize it by using a validation dataset. For the parametric method, σ^2 is a parameter so it is learned on the training data.

- *note:* full points only for justifying that we can't learn σ^2 on the dataset, explaining how we do choose it (validation), or explaining that it controls the capacity of the model

3.4

(a)

$$\hat{p}_{gauss-diagonal} = \frac{1}{(2\pi)^{\frac{d}{2}} \prod_j^d \sigma_{jj}} e^{-\frac{1}{2} \sum_j^d \frac{(x_j - \mu_j)^2}{\sigma_{jj}^2}}$$

with parameters are mean μ of dimensionality d and covariances $\sigma_{11} \dots \sigma_{nn}$, in total dimensionality d

- also accepted parameter Σ with dimensionality $d \times d$ if you mentioned that $\sigma_{ij} = 0$ where $i \neq j$
- *note:* you had to write the equation for the diagonal here and couldn't reuse the general equation

(b)

$P_{xy}(x, y) = P_x(x)P_y(Y)$ implies that x and y are independent

$$\begin{aligned}
\hat{p}(x) &= \frac{1}{(2\pi)^{\frac{d}{2}} \prod_j^d \sigma_{jj}} e^{-\frac{1}{2}\sum_j^d \frac{(x_j-\mu_j)^2}{\sigma_{jj}^2}} \\
&= \prod_j^d \frac{1}{(2\pi)^{\frac{d}{2}} \sigma_{jj}} e^{-\frac{1}{2}\frac{(x_j-\mu_j)^2}{\sigma_{jj}^2}} \\
&= \prod_j^d p_j(x_j)
\end{aligned}$$

therefore the components $x_1 \dots x_d$ are independent random variable

- *note:* even though independence implies that covariance is 0, the converse (covariance 0 implies independence) is not necessarily true!

(c)

$$\begin{aligned}
\hat{R}(p) &= \frac{1}{n}\sum_i^n -\log p(x^{(i)}) \\
&= \frac{1}{n}\sum_i^n -\log \prod_j^d p_j(x_j^{(i)}) \\
&= \frac{1}{n}\sum_i^n \sum_j^d -\log p_j(x_j^{(i)}) \\
&= \frac{1}{n}\sum_i^n \sum_j^d -\log \frac{1}{(2\pi)^{\frac{1}{2}} \sigma_{jj}} e^{-\frac{1}{2}\frac{(x_j^{(i)}-\mu_j)^2}{\sigma_{jj}^2}} \\
&= \frac{d}{2}\log 2\pi + \frac{1}{n}\sum_i^n \sum_j^d -\log \frac{1}{\sigma_{jj}} e^{-\frac{1}{2}\frac{(x_j^{(i)}-\mu_j)^2}{\sigma_{jj}^2}} \\
&= \frac{d}{2}\log 2\pi + \frac{1}{n}\sum_i^n \sum_j^d \log \sigma_{jj} + \frac{1}{2}\frac{(x_j^{(i)}-\mu_j)^2}{\sigma_{jj}^2}
\end{aligned}$$

our objective is minimization wrt μ, Σ so we can remove the constant term

$$argmin_{\mu,\Sigma} \frac{1}{n}\sum_i^n \sum_j^d \log \sigma_{jj} + \frac{1}{2}\frac{(x_j^{(i)}-\mu_j)^2}{\sigma_{jj}^2}$$

(d)

to solve for μ we find the partial derivative and set it to 0, we will solve for μ_j

$$\begin{aligned} 0 &= \frac{d\hat{R}}{d\mu_j} \\ &= \frac{1}{n}\sum_i^n -\frac{(x_j^{(i)} - \mu_j)^2}{\sigma_{jj}^2} \\ &= \frac{1}{n}\sum_i^n -(x_j^{(i)} - \mu_j)^2 \\ \mu_j &= \frac{1}{n}\sum_i^n x_j^{(i)} \end{aligned}$$

we can extrapolate this to all j

$$\mu = \frac{1}{n}\sum_i^n x^{(i)}$$

to solve for σ we find the partial derivative and set it to 0, we will solve for σ_{jj}

$$\begin{aligned} 0 &= \frac{d\hat{R}}{d\sigma_{jj}} \\ &= \frac{1}{n}\sum_i^n \frac{1}{\sigma_{jj}} - \frac{(x_j^{(i)} - \mu_j)^2}{\sigma_{jj}^3} \\ \frac{1}{\sigma_{jj}} &= \frac{1}{n}\sum_i^n \frac{(x_j^{(i)} - \mu_j)^2}{\sigma_{jj}^3} \\ \sigma_{jj}^2 &= \frac{1}{n}\sum_i^n (x_j^{(i)} - \mu_j)^2 \end{aligned}$$

- $\sigma_{jj}^2 = \frac{1}{n}\sum_i^n (x_j^{(i)} - \mu_j)^2$ **does not imply** $\Sigma = \frac{1}{n}\sum_i^n (x^{(i)} - \mu)(x^{(i)} - \mu)^\intercal$ because everything other than the diagonals is 0
- *note:* you needed to show the derivation for full marks

4 Practical part: density estimation

4.1

You were asked to implement a Diagonal Gaussian density. The most frequent error were a wrong scaling of the Gaussian: this can be spotted by looking at

the plots since the pdf must integrate to 1. Some used the densities instead of the log densities: this is fine as long as it is consistent with your plots.
Here is a proposed implementation:

```
class GaussDiagonal:
    def __init__(self, n_dims):
        # n_dims can also be inferred from train_data at train time
        self.n_dims = n_dims

    def train(self, data):
        self.mean = data.mean(axis=0, keepdims=True)
        self.sigma2 = data.var(axis=0, keepdims=True)

    def predict(self, test_data):
        c = - self.n_dims * np.log(2*np.pi) / 2.0 - np.log(np.prod(self.sigma2)) / 2.0
        # since we used keepdims in train then self.mean will be broadcasted to match \
        # test_data's shape
        log_prob = c - np.sum((test_data -  self.mean)**2.0 / (2.0 * self.sigma2), axis=1)
        return log_prob
```

4.2

You were asked to implement a Parzen density. The most frequent error was at implementing distances between each train and test samples.
Here is a proposed implementation:

```
def logsumexp(X, axis=0):
    # for numerical stability:
    # return np.log(np.exp(X - X.max(axis=axis, keepdims=True)).sum(axis=axis)) + X.max(axis=axis)
    # but this one worked as well:
    return np.log(np.exp(X).sum(axis=axis))

class ParzenGaussIsotropic:
    def __init__(self, n_dims, sigma2):
        self.n_dims = n_dims
        self.sigma2 = sigma2

    def train(self, data):
        self.train_data = data

    def predict(self, test_data):
        # squarred distance between each train and each test:
        # this notation adds broadcast axes to do everything in a single line
        # but of course we accepted answers using 1 or 2 for loops
        ds = ((self.train_data[None, :, :] - test_data[:, None, :])**2).sum(axis=2)
        c = - self.n_dims * np.log(2*np.pi)/2.0 - np.log(self.sigma2**self.n_dims) / 2.0
        # log probs of each example of the test set for all gaussian models in the train set
        # this is a n_test x n_train matrix array :
        indiv_log_probs = c - 1. / (2. * self.sigma2) * ds
        # summing over all train examples
        log_prob = logsumexp(indiv_log_probs, axis=1) - np.log(self.train_data.shape[0])
        return log_prob
```

4.3

(a)

(b)

(c)

(d)

(e)

Most frequent errors:

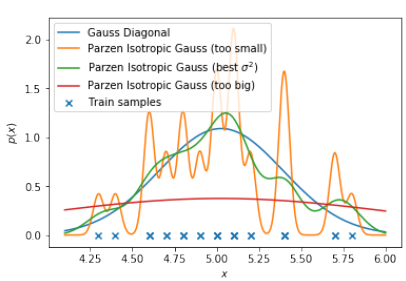
- Train set samples were not displayed
- Some people used test set samples to plot the densities. It is more informative to instead generate points `np.arange` or `np.linspace` and compute the densities for all intermediate points (see proposed solution below)

Training:

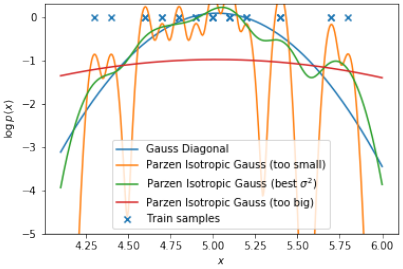
```
iris = np.loadtxt('iris.txt')

iris_train = iris[:30, :]
iris_valid = iris[30:50, :]
train_cols = [0]

model_gaussdiagonal = GaussDiagonal(len(train_cols))
model_parzen_small = ParzenGaussIsotropic(len(train_cols), .001)
model_parzen_best = ParzenGaussIsotropic(len(train_cols), .01)
model_parzen_big = ParzenGaussIsotropic(len(train_cols), 1.)

model_gaussdiagonal.train(iris_train[:, train_cols])
model_parzen_small.train(iris_train[:, train_cols])
model_parzen_best.train(iris_train[:, train_cols])
model_parzen_big.train(iris_train[:, train_cols])
```

Plotting:

```
min_x = iris_train[:, train_cols[0]].min() - .2
max_x = iris_train[:, train_cols[0]].max() + .2
x_pdfs = np.linspace(min_x, max_x, 300)
x_pdfs = x_pdfs.reshape(-1, 1)
pylab.scatter(iris_train[:, train_cols[0]], np.zeros_like(iris_train[:, train_cols[0]]),
              marker='x', label='Train samples')
y_pdfs_gaussdiagonal = model_gaussdiagonal.predict(x_pdfs)
y_pdfs_parzen_small = model_parzen_small.predict(x_pdfs)
y_pdfs_parzen_best = model_parzen_best.predict(x_pdfs)
y_pdfs_parzen_big = model_parzen_big.predict(x_pdfs)
pylab.plot(x_pdfs, np.exp(y_pdfs_gaussdiagonal), label='Gauss Diagonal')
pylab.plot(x_pdfs, np.exp(y_pdfs_parzen_small), label='Parzen Isotropic Gauss (too small)')
pylab.plot(x_pdfs, np.exp(y_pdfs_parzen_best), label='Parzen Isotropic Gauss (best $\sigma^2$)')
pylab.plot(x_pdfs, np.exp(y_pdfs_parzen_big), label='Parzen Isotropic Gauss (too big)')
pylab.ylabel('$p\\left(x\\right)$')
pylab.xlabel('$x$')
pylab.legend()
```



(a) Pdfs for Gauss diagonal, and Parzen

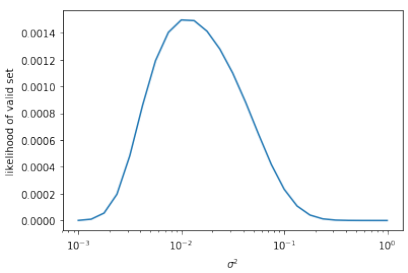


(b) Log densities: Also accepted, but more difficult to interpret.

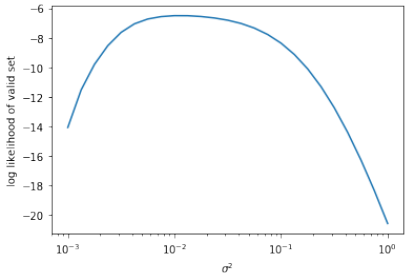
(f)

You obtained full mark is you used a validation set to optimize σ^2. Some students proposed other techniques: They were given half points. See proposed answer below:

```
# find most appropriate sigma using validation set

candidate_sigma2s = 10 ** np.linspace(-3, 0, 25)
lls = []
for sigma2 in candidate_sigma2s:
    model_parzen = ParzenGaussIsotropic(len(train_cols), sigma2)
    model_parzen.train(iris_train[:, train_cols])
    # LL of valid set:
    ll = model_parzen.predict(iris_valid[:, train_cols]).sum()
    lls.append(ll)

pylab.plot(candidate_sigma2s, lls)
pylab.xscale('log')
pylab.xlabel('$\sigma^2$')
pylab.ylabel('log likelihood of valid set')
pylab.show()
pylab.plot(candidate_sigma2s, np.exp(lls))
pylab.xscale('log')
pylab.xlabel('$\sigma^2$')
pylab.ylabel('likelihood of valid set')
pylab.show()
```



(a) Likelihood of points in the valid set



(b) Log Likelihood of points in the valid set

4.4

(a)

(b)

(c)

(d)

Most frequent errors were:

- In order to generate the points (x, y) where to compute the densities that you needed to pass to `contour` you had to use `np.meshgrid` then reshape appropriately (see proposed solution below)
- Some forgot to plot the points in the train set. In that case it is hard to tell if your model does a good job at modeling the densities.

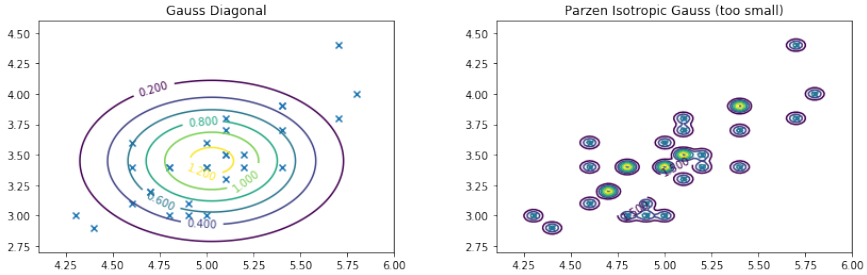
Training:

```
train_cols = [0, 1]

model_gaussdiagonal = GaussDiagonal(len(train_cols))# gaussienne diagonale
model_parzen_small = ParzenGaussIsotropic(len(train_cols), .001)
model_parzen_best = ParzenGaussIsotropic(len(train_cols), 5e-2)
model_parzen_big = ParzenGaussIsotropic(len(train_cols), 1.)

#entrainement
model_gaussdiagonal.train(iris_train[:, train_cols])
model_parzen_small.train(iris_train[:, train_cols])
model_parzen_best.train(iris_train[:, train_cols])
model_parzen_big.train(iris_train[:, train_cols])
```

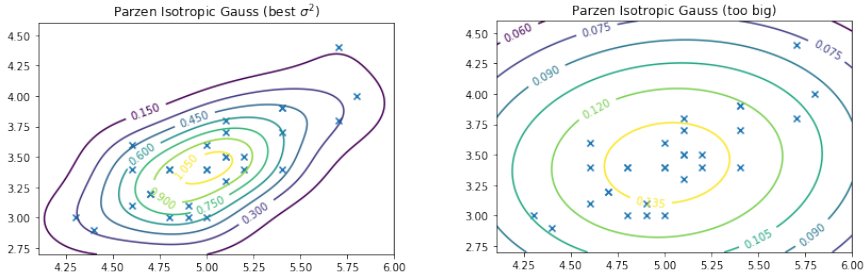
Plotting:

```
min_x = iris_train[:, train_cols[0]].min() - .2
max_x = iris_train[:, train_cols[0]].max() + .2
min_y = iris_train[:, train_cols[1]].min() - .2
max_y = iris_train[:, train_cols[1]].max() + .2
x_pdfs = np.linspace(min_x, max_x, 100)
y_pdfs = np.linspace(min_y, max_y, 100)
```

```
X, Y = np.meshgrid(x_pdfs, y_pdfs)
support = np.concatenate([X.reshape(-1, 1), Y.reshape(-1, 1)], axis=1)

z_pdfs_gaussdiagonal = model_gaussdiagonal.predict(support)
z_pdfs_parzen_small = model_parzen_small.predict(support)
z_pdfs_parzen_best = model_parzen_best.predict(support)
z_pdfs_parzen_big = model_parzen_big.predict(support)

pylab.figure()
pylab.scatter(iris_train[:, train_cols[0]], iris_train[:, train_cols[1]], marker='x')
c = pylab.contour(x_pdfs, y_pdfs, np.exp(z_pdfs_gaussdiagonal.reshape(100, 100)))
pylab.clabel(c)
pylab.title('Gauss Diagonal')
pylab.show()

pylab.figure()
pylab.scatter(iris_train[:, train_cols[0]], iris_train[:, train_cols[1]], marker='x')
c = pylab.contour(x_pdfs, y_pdfs, np.exp(z_pdfs_parzen_small.reshape(100, 100)))
pylab.clabel(c)
pylab.title('Parzen Isotropic Gauss (too small)')
pylab.show()

pylab.figure()
pylab.scatter(iris_train[:, train_cols[0]], iris_train[:, train_cols[1]], marker='x')
c = pylab.contour(x_pdfs, y_pdfs, np.exp(z_pdfs_parzen_best.reshape(100, 100)))
pylab.clabel(c)
pylab.title('Parzen Isotropic Gauss (best $\sigma^2$)')
pylab.show()

pylab.figure()
pylab.scatter(iris_train[:, train_cols[0]], iris_train[:, train_cols[1]], marker='x')
c = pylab.contour(x_pdfs, y_pdfs, np.exp(z_pdfs_parzen_big.reshape(100, 100)))
pylab.clabel(c)
pylab.title('Parzen Isotropic Gauss (too big)')
pylab.show()
```



(a) (b)

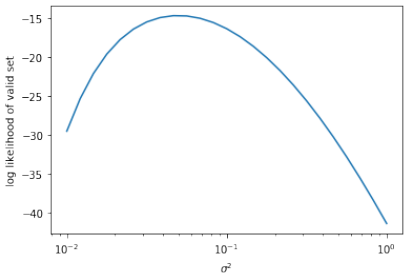


(c) (d)

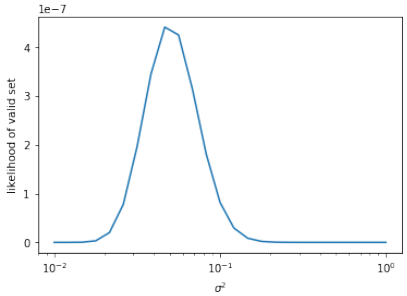
(e)

Similarly as 4.3.f, you had to use the valid set:

```
# find most appropriate sigma using validation set

candidate_sigma2s = 10 ** np.linspace(-2, 0, 25)
lls = []
for sigma2 in candidate_sigma2s:
    model_parzen = ParzenGaussIsotropic(len(train_cols), sigma2)
    model_parzen.train(iris_train[:, train_cols])
    # LL of valid set:
    ll = model_parzen.predict(iris_valid[:, train_cols]).sum()
    lls.append(ll)

pylab.plot(candidate_sigma2s, lls)
pylab.xscale('log')
pylab.xlabel('$\sigma^2$')
pylab.ylabel('log likelihood of valid set')
pylab.show()
pylab.plot(candidate_sigma2s, np.exp(lls))
pylab.xscale('log')
pylab.xlabel('$\sigma^2$')
pylab.ylabel('likelihood of valid set')
pylab.show()
```



(a) Log Likelihood of points in the valid set



(b) Likelihood of points in the valid set