CMP9794M Advanced Artificial Intelligence – Workshop Week 5

UNIVERSITY OF LINCOLN

Summary: In this workshop you will get familiarised with Gaussian distributions and with Gaussian Processes (GPs) for applying them to data containing continuous inputs and discrete outputs. An example implementation of GPs is provided and applied to multiple classification tasks. The workshop materials from this week (Zip file in Blackboard under week 5) require installing the following dependencies: `pip install numpy; scipy; pandas; matplotlib`

Task 1: **Familiarisation wit Gaussian probability distributions**

a. What is the value of $f(x)$ using the following equations with $\mu = 4.86$ and $\sigma = 0.94$?

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Answering the question above using the following Python code:

```
import numpy as np

mean = 4.86
stdev = 0.94
var = stdev**2

def get_gaussian_probability_density_v1(x, mean, stdev):
    e_val = -0.5*np.power((x-mean)/stdev, 2)
    return (1/(stdev*np.sqrt(2*np.pi))) * np.exp(e_val)

def get_gaussian_probability_density_v2(x, mean, var):
    e_val = -np.power((x-mean), 2)/(2*var)
    return (1/(np.sqrt(2*np.pi*var))) * np.exp(e_val)

for x in range(0, 10):
    fx_v1 = get_gaussian_probability_density_v1(x, mean, stdev)
    fx_v2 = get_gaussian_probability_density_v2(x, mean, var)
    print("x=%s f(x)_v1=%s f(x)_v2=%s" % (x, fx_v1, fx_v2))
```

Are you able to see that the closer x is to the mean the higher the probability density?

b. Execute the code below to visualise randomly generated samples from a 1-dimensional Gaussian.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
```

```
mean = 4.86
stdev = 0.94
samples = np.random.normal(mean, stdev, size=(1, 1000))
sns.distplot(samples, hist=True, rug=True, fit=norm, kde=False,
color="r", vertical=False)
plt.xlabel('$x$', fontsize = 16)
plt.ylabel('$f(x)$', fontsize = 16)
plt.show()
```

c. Execute the code below to visualise randomly generated samples from 2-dimensional (or bi-variate) Gaussians with different covariances. The X-axis refers to random variable x_1 and the Y-axis to x_2.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

mean = [2., 2.]
positive_cov = [[1.0, 0.7], [0.7, 1.0]]
negative_cov = [[1.0, -0.7], [-0.7, 1.0]]
no_cov = [[1.0, 0.0], [0.0, 1.0]]

data_pos = np.random.multivariate_normal(mean, positive_cov, 1000)
data_neg = np.random.multivariate_normal(mean, negative_cov, 1000)
data_no = np.random.multivariate_normal(mean, no_cov, 1000)
df_pos = pd.DataFrame(data_pos, columns=["x1", "x2"])
df_neg = pd.DataFrame(data_neg, columns=["x1", "x2"])
df_no = pd.DataFrame(data_no, columns=["x1", "x2"])

plt.subplot(1, 3, 1)
plt.scatter(df_pos["x1"], df_pos["x2"], marker="+")
plt.title("positive covariance")

plt.subplot(1, 3, 2)
plt.scatter(df_neg["x1"], df_neg["x2"], marker="+")
plt.title("negative covariance")

plt.subplot(1, 3, 3)
plt.scatter(df_no["x1"], df_no["x2"], marker="+")
plt.title("no covariance")

plt.show()
```

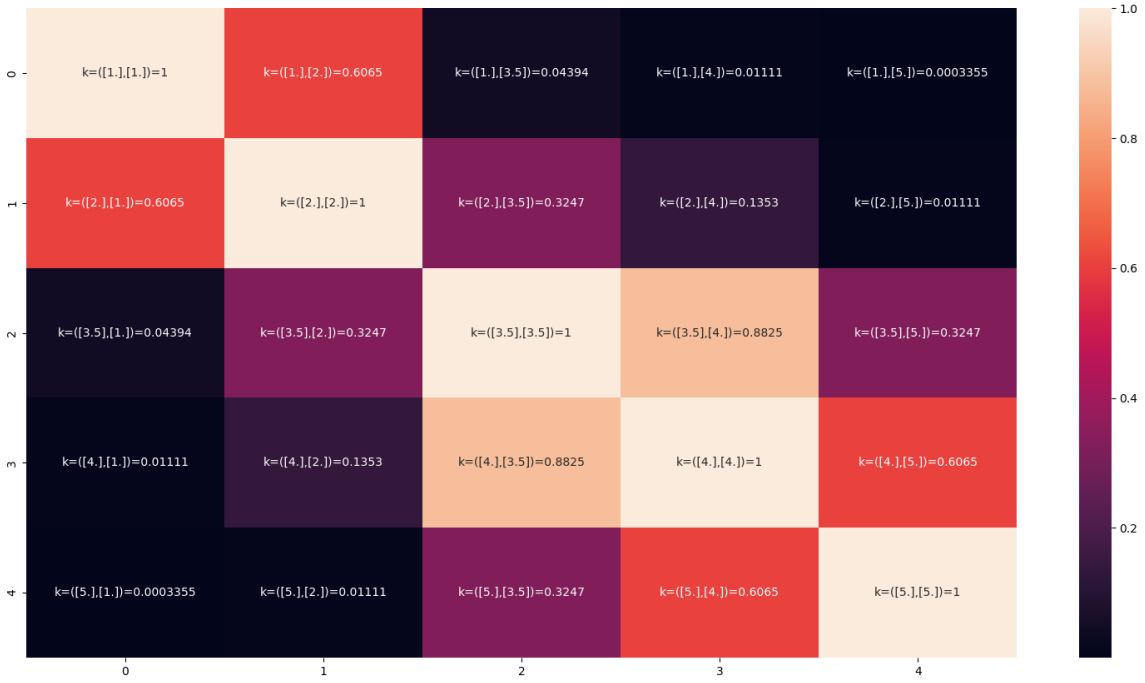
Task 2: **Understanding and training Gaussian Process classifiers**

a. The following program illustrates the RBF kernel applied to 1D and 2D data:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

def kernel(X1, X2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(X1**2, 1).reshape(-1, 1) + np.sum(X2**2, 1) - 2 *
np.dot(X1, X2.T)
    return sigma_f**2 * np.exp(-0.5 / l**2 * sqdist)
```

```
X=[1, 2, 3.5, 4, 5]
X = np.asarray(X).reshape(-1, 1)
K = kernel(X, X)
print("\n1D data: kernel(X, X)=",K)

pairs = []
for i in range(0,len(X)):
    for j in range(0, len(X)):
        pairs.append("k=(%s,%s)=%.4g" % (str(X[i]), str(X[j]),
K[i][j]))
pairs = np.asarray(pairs).reshape(len(X), len(X))
sns.heatmap(K, data=X, annot=pairs, fmt='')
plt.show()

A=[[0.36, 0.11],
   [1.48, 0.55]]
B=[[1.50, 0.90]]
A = np.asarray(A)
B = np.asarray(B)
K = kernel(A, B)
print("\n2D data: kernel(A, B)=",K)
```

Save and run the code above, which should generate the heatmap shown below for the 1D data. Look at the original and kernelised values, where the closer/similar x_i and x_j are the higher the kernel value (towards 1) – and the more dissimilar x's are the kernel values get closer to zero. This is useful information for a Gaussing Process to distinguish when training examples (or data points) look similar—in order to later on generate similar outputs. In other words, if two data points look similar, they will potentially have a similar output.



Once you have looked at the above, modify one of the 2D vectors to make it similar or dissimilar to the others. Do you see any pattern in the kernel values as the vectors change?

b. This task requires you to make use of three datasets, already included in the Zip file of the workshop materials, to train and test Gaussian Process models:

- 2D continuous inputs with discrete output, **linearly separable** toy data.
- 2D continuous inputs with discrete output, **not linearly separable** toy data.
- 4D continuous outputs with discrete output for **bank note authentication**. Source: https://archive.ics.uci.edu/dataset/267/banknote+authentication

You can do that via the following commands:

- `python GaussianProcess.py ..\data\data-linearlyseparable-train.csv ..\data\data-linearlyseparable-test.csv`
- `python GaussianProcess.py ..\data\data-nonlinearlyseparable-train.csv ..\data\data-nonlinearlyseparable-test.csv`
- `python GaussianProcess.py ..\data\data_banknote_authentication-train.csv ..\data\data_banknote_authentication-test.csv`

c. The program `GaussianProcess.py` implements the baseline mentioned in the lecture with two variants. The first variant normalises predicted mean values ignoring the variance:

$$P(\boldsymbol{y}_* = 1|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) \approx \frac{\boldsymbol{\mu}_* - \min(\boldsymbol{\mu}_*)}{max(\boldsymbol{\mu}_*) - \min(\boldsymbol{\mu}_*)}$$
$$P(\boldsymbol{y}_* = 0|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) = 1 - P(\boldsymbol{y}_* = 1|\mathbf{X}, \mathbf{y}, \mathbf{x}_*)$$

The second variant uses the predicted means and variances to calculate normalised probability densities $f(y; \mu, \sigma^2)$, i.e., this variant takes the mean and variance into account:

$$P(y_{*i} = 1|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) \approx \frac{f(y_{*\_i} = 1;\ \mu_{*i}, \sigma_{*i}^2)}{f(y_{*_i} = 1;\ \mu_{*i}, \sigma_{*i}^2) + f(y_{*\_i} = 0;\ \mu_{*i}, \sigma_{*i}^2)}$$
$$P(y_{*i} = 0|\mathbf{X}, \mathbf{y}, \mathbf{x}_*) \approx \frac{f(y_{*\_i} = 0;\ \mu_{*i}, \sigma_{*i}^2)}{f(y_{*_i} = 1;\ \mu_{*i}, \sigma_{*i}^2) + f(y_{*\_i} = 0;\ \mu_{*i}, \sigma_{*i}^2)}$$

d. To observe the effects of the choice of hyperparameters, re-run the commands of task (b) but set the following values of `l_opt=1, sigma_f_opt=1, noise=0.0001`. See lines 71 to 76 of `GaussianProcess.py`. You can also set your own values randomly between a 0.01 and 10 for example. Overall, did you get better or worse results than the optimised ones?

e. The class `GaussianProcess.py` is powered by the code in `gaussian_processes_util`. In your own time, explore the block of methods labelled as "GPs for regression utils".

Task 3: **Homework**

Apply the `GaussianProcess.py` to the datasets of the assignment. If you are inquisitive to find out about the performance of other tools, use baseline above (and the data of the assignment if possible) to compare against some other libraries such as GPy, GPyTorch, GPFlow, Scikit-Learn.