Programming Assignment I :

Gaussian Process Regression

Amatya Sharma

17CS30042

Objective

Input

CSV data file new_cases_World-India containing four columns: day number, date, number of new cases in world, number of new cases in India.

Goal

Predict the number of new COVID-19 cases in India and the World using Gaussian Process Regression.

Output

For both World and India individually, we ouput the following:

- A model with Self-Fed or guessed Hyper-Parameters
- A model with optimized hyper-parameters

For each model, we output following:

- Visualization of fitted curve on training data
- Visualization of predicted curve on test data along with train data
- Zoomed in Visualization of predicted curve on test data
- List of Means and Variances for predictions on Test data

We also visualize the following kernels using histograms:

- Kernel (Xtrain, Xtrain)
- Kernel (Xtest, Xtest)
- Kernel (Xtrain, Xtest)

We also visualize 4 random draws from a multivariate Gaussian prior.

Reading and Partitioning Data

- We use **pandas** library to read data from CSV file
- We then partition data in 2 sets :
  - Training Data : Contains first 246 entries (till begining of september)
  - Test Data : Conatains rest of entries (september month)
- We also consider Ytrain for world and india separately.

GP Prior

We take prior with 0 mean.

- To decide up on Kerel function we choose from two standard functions :
  1. RBF Kernel :

```
def rbf_kernel_util(sigma_f, x1, x2):
return np.exp(-(x1-x2)**2/(2* sigma_f**2))
```

  2. Squared Exponential Kernel:

```
def sqexp_kernel_util(x1, x2, l, sigma_f):
return sigma_f*sigma_f * np.exp(-0.5/(l*l) * (x1-x2)*(x1-x2))
```

- We observe that (2) Squared Exponential Kernel fits more closely to the data and thus we use it for the rest of the assignment for calculating kernels.
- For calculating Kernel for vectors we call `sqexp_kernel_util()` for each element of resulting 2-D vector :

```
def kernel(X1, X2, l, sigma_f):
  l1 = len(X1)
  l2 = len(X2)
  K = np.zeros((l1, l2))

  #iterate over all indices of the covariance matrix to fill it with k(i,j)
  for i in range(l1):
      for j in range(l2):
          K[i][j] = sqexp_kernel_util(X1[i], X2[j], l, sigma_f)
  return K
```

- For visualizing the kernel we use `matplotlib.pyplot.imshow()` function
- We make 5 draws from a multivariate Gaussian pdf with Mean as 0 and covariance given by the Square Exp Kernel from above.
    - For drawing we use numpy function

```
rand_draw_prior = np.dot(Kplot, np.random.normal(size=(numplot, 5)))
```

    - And then plot it using **matplotlib.pyplot**

Posterior Predictive Distribution

- We directly use the fact that PPD is Gaussian with:
    - Mean Vector

$$mu_s = K_s^T . K^{-1} . Ytrain$$

    - Covarinace Matrix

$$sigma_s = K_s s - K_s^T . K^{-1} . K_s$$

  where
    - K_s = kernel(Xtrain, Xtest)
    - K_ss = kernel(Xtest, Xtest)
    - K = kernel(Xtrain, Xtrain) + (sigma_y)^2 I
    - sigma_y (=0.1)= given variance of noise-error

  The implementation is as given :

```
def Cinv(K, sigma_y):
  K_inv = np.linalg.inv(K + (sigma_y*sigma_y)*np.eye(len(Xtrain)))
  return K_inv
```

```
def posterior_predictive(Xtrain, Ytrain, l, sigma_y, sigma_f, X):
  cov_s = 0.0
  K_ss = kernel(X, X, l, sigma_f)
  K_s = kernel(Xtrain, X, l, sigma_f)

  mu_s = 0.0
  K = kernel(Xtrain, Xtrain, l, sigma_f)
  K_inv = Cinv(K, sigma_y)
```

```
    cov_s = K_ss - K_s.T.dot(K_inv).dot(K_s)
    mu_s = K_s.T.dot(K_inv).dot(Ytrain)

    return mu_s, cov_s
```

Plotting of Predictions

- We use user-defined function `predict_and_plot(Xtrain, Ytrain, l, sigma_f, sigma_y, Xtest, Ytest)` for plotting the data.
- The function is used to :
    - Plot the Fitted Curve on Training Set
    - Plot the predicted curve on test data along with train data
    - Plot zoomed in visualization of predicted curve on test data
- Internally the function employs `matplotlib.pyplot` library functions for plotting graphs.

Hyper Parameters

Prediction with Self-Fed or Guessed Hyper-Parameters

- We manually twitch the parameters and call `predict_and_plot(Xtrain, Ytrain, l, sigma_f, sigma_y, Xtest, Ytest)` to plot and fit the curve to train data as much as possible.

Prediction with Optimized Hyper-Parameters

- With initial guess same as our previous self-fed values we, iterate within a range (sigma_f-10, sigma_f+10) and (l-100, l+100) to find the minima in the range.
- The goal of mimization function `hyperparameter_opt(sigma_f, l, Xtrain, Ytrain, Xtest, Ytest)` is to minimize the error over the given range.
- We employ Root Mean Square Error as our objective error/cost function.

Conclusions

- Despite being able to fit the posterior mean perfectly, the choice of the wrong prior took its toll. The GP here, is underestimating its variance (over confident) which is not good for practical applications. This, was easily rectified by either scaling the GP by eye-balling or performing MLE.
- For Hyper-Parameter optimization, we have used a very naive method of local search based on intial guess, which can be improved by replacing it with **Negative Logarithmic Likelihod** optimization and minimize function from scipy libraries (couldn't do as use of only basic

libraries is permitted). However, as an extra confirmational measure, we optimized hyperparameters with **NLL optimization** and found quite less difference between its output and our output.

- As a part of estimation we have plotted **MAP** as data points i.e. **mu_s**. For a clearer visualization we could have used `plot_gp()` from GP Library to plot the confidence interval along with means using variances.

References

[1] https://ourworldindata.org/coronavirus-source-data Website for procuring neccesary data.
[2] Kevin P. Murphy. Machine Learning, A Probabilistic Perspective, Chapters 4, 14 and 15.
[3] Christopher M. Bishop. Pattern Recognition and Machine Learning, Chapter 6.
[4] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian Processes for Machine Learning.
[5] Paper on **Hands-on Experience with Gaussian Processes (GPs): Implementing GPs in Python - I by Kshitij Tiwaria** (Intelligent Robotics Group, Department of Electrical Engg. & Automation, Aalto University, Espoo, 02150, Finland). [Link - https://arxiv.org/pdf/1809.01913.pdf]