assn2

March 9, 2022

1 Intermediate Machine Learning: Assignment 2

Deadline

Assignment 2 is due Wednesday, March 9 11:59pm. Late work will not be accepted as per the course policies (see the Syllabus and Course policies on Canvas).

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged.

You should start early so that you have time to get help if you're stuck. The drop-in office hours schedule can be found on Canvas. You can also post questions or start discussions on Ed Discussion. The assignment may look long at first glance, but the problems are broken up into steps that should help you to make steady progress.

Submission

Submit your assignment as a pdf file on Gradescope, and as a notebook (.ipynb) on Canvas. You can access Gradescope through Canvas on the left-side of the class home page. The problems in each homework assignment are numbered. Note: When submitting on Gradescope, please select the correct pages of your pdf that correspond to each problem. This will allow graders to more easily find your complete solution to each problem.

To produce the .pdf, please do the following in order to preserve the cell structure of the notebook:

Go to "File" at the top-left of your Jupyter Notebook Under "Download as", select "HTML (.html)" After the .html has downloaded, open it and then select "File" and "Print" (note you will not actually be printing) From the print window, select the option to save as a .pdf

Topics

- Convolutional neural networks
- Gaussian processes
- Dirichlet processes

This assignment will also help to solidify your Python and Jupyter notebook skills.

1.1 Problem 1: Filter through (15 points)

In this problem, we will "open the black box" and inspect the filters and feature maps learned by a convolutional neural network trained to classify handwritten digits, using the MNIST database.

[ ]:
```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

1.1.1 1.1 Visualizing the filters

To begin, we load the dataset with 60000 training images and 10000 test images.

[ ]:
```
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train_binary = keras.utils.to_categorical(y_train, num_classes)
y_test_binary = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Next, we initialize our convolutional neural network similar to the network we used for Assignment 1 Problem 4 except that we now have a few more layers.

[ ]:
```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(5, 5), activation="relu", name='conv1'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(32, kernel_size=(5, 5), activation="relu", name='conv2'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
WARNING:tensorflow:From
/home/xiaoranzhang/anaconda3/envs/tf_directml/lib/python3.6/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 8, 8, 32)          25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 10)                5130
=================================================================
Total params: 31,594
Trainable params: 31,594
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
batch_size = 128
epochs = 1

init = tf.global_variables_initializer()
sess = tf.compat.v1.Session()
sess.run(init)

model.compile(loss="categorical_crossentropy", optimizer="adam",␣
 ↪metrics=["accuracy"])
```

```
model.fit(x_train, y_train_binary, batch_size=batch_size, epochs=epochs,␣
 ↪validation_split=0.1)
```

```
Train on 54000 samples, validate on 6000 samples
54000/54000 [==============================] - 7s 126us/sample - loss: 0.1220 -
acc: 0.9635 - val_loss: 0.0502 - val_acc: 0.9872
```

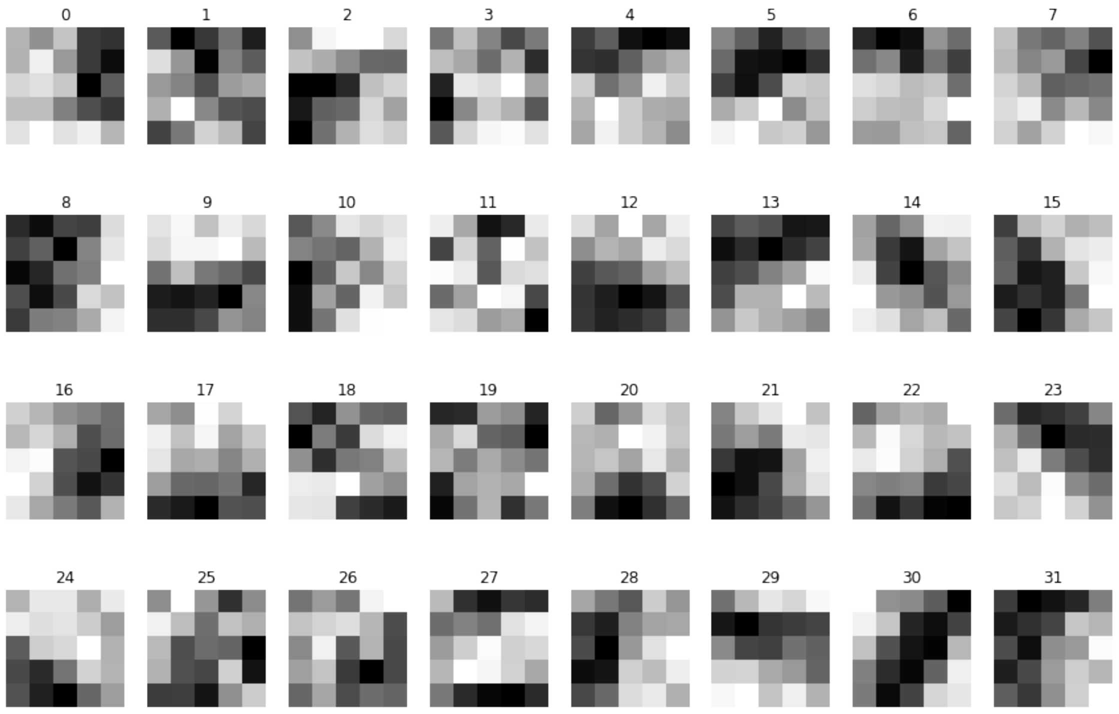
```
[ ]: <tensorflow.python.keras.callbacks.History at 0x7f44cd728f60>
```

```
[ ]: score = model.evaluate(x_test, y_test_binary, verbose=0)
     print("Test loss:", score[0])
     print("Test accuracy:", score[1])
```

```
Test loss: 0.05162191509041004
Test accuracy: 0.98340005
```

Now that we've trained and tested the model, let's look at the filters learned in the first convolutional layer.

```
[ ]: filters_conv1 = model.get_layer(name='conv1').get_weights()[0]

     fig, axs = plt.subplots(4, 8)
     fig.set_figheight(10)
     fig.set_figwidth(15)

     for i in range(4):
         for j in range(8):
             f = filters_conv1[:, :, 0, 8*i+j]
             axs[i, j].imshow(f[:, :], cmap='gray')
             axs[i, j].axis('off')
             axs[i, j].set_title(8*i+j)
```



Describe what you see. Do (some of) the learned filters make sense to you?

Hint: Many filters have been designed and widely applied in image processing. Here are some examples of edge detection filters and their effect on the image. You can find the details about each filter by clicking the links at the bottom.

Answer: The filters make sense as many of them seem to be a edge detection filter. For instance, filters 9, 17, and 22 seem to detect horizontal edges and filter 28 for vertical edges, which shares a similar role to the Scharr and Prewitt filter.

1.1.2 1.2 Visualizing the feature maps

We can also look at the corresponding feature map for each filter. There are 32 kernels at the first convolutional layer, so there are 32 feature maps for each sample. feature_map_conv1 is a 4D matrix where the first dimension is the index of the sample and the last dimension is the index of the correpsonding filter.

```
[ ]: conv1_layer_model = keras.Model(inputs=model.input, outputs=model.
      ↪get_layer('conv1').output)
     feature_map_conv1 = conv1_layer_model(x_test)
```

Randomly draw 16 samples for visualization.

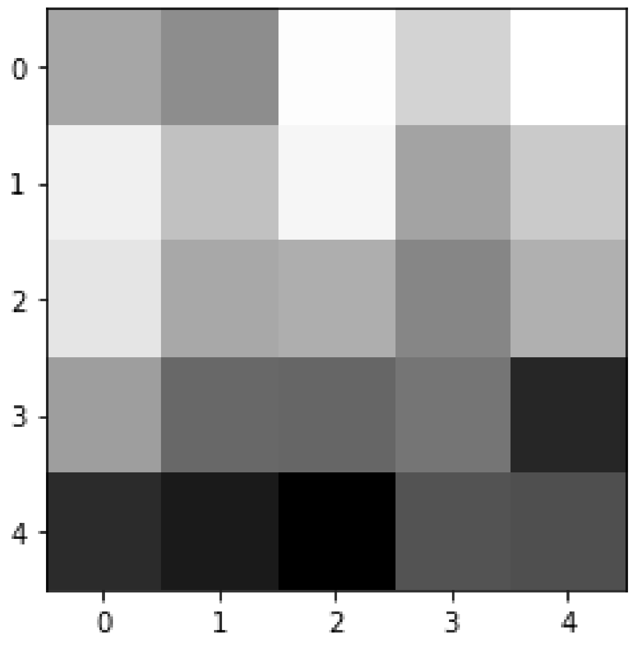
```
[ ]: sample_index = random.sample(range(1, len(x_test)), 16)
```

Choose two filters among all 32 filters from 3.1, visualize their feature maps. There is no need to modify the code, just run the four cells below.

```
[ ]: filter_n1 = 17#
     filter_n2 = 30#
```

```
[ ]: plt.imshow(filters_conv1[:, :, 0, filter_n1], cmap='gray')
```

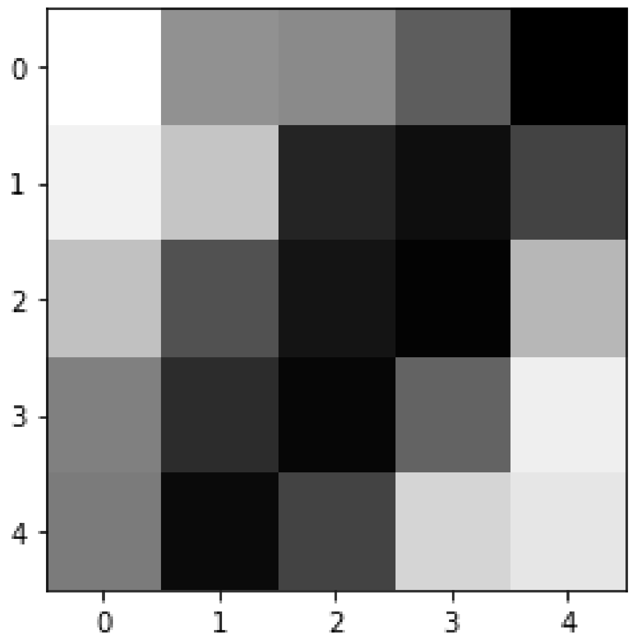
```
[ ]: <matplotlib.image.AxesImage at 0x7f437850fc88>
```



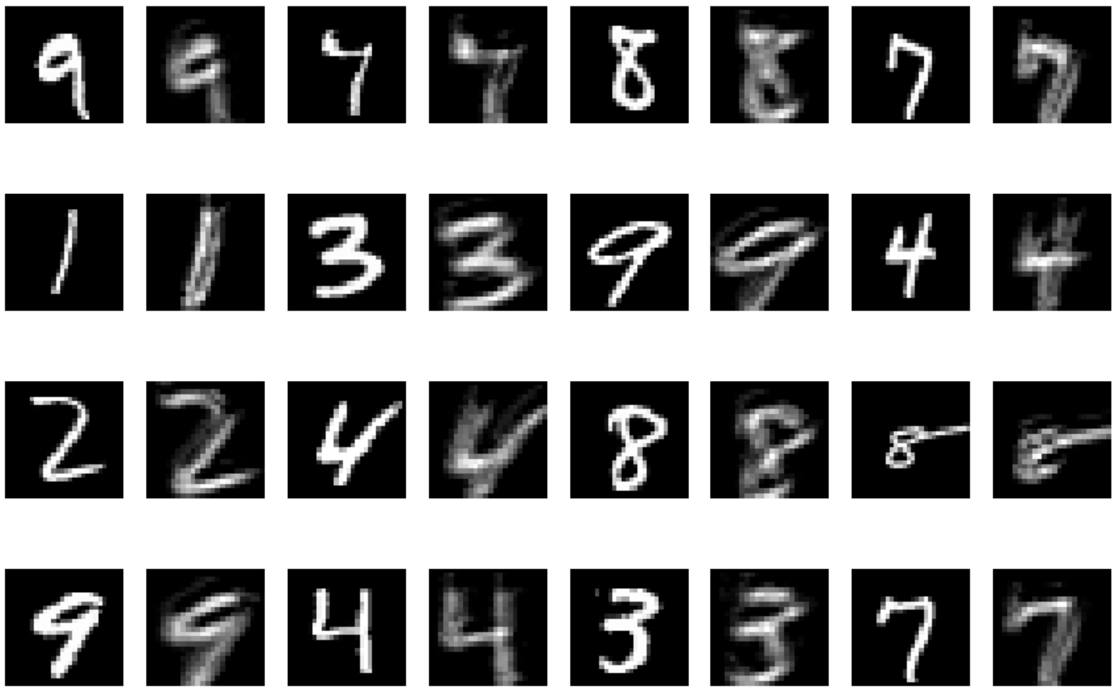
```
[ ]: fig, axs = plt.subplots(4, 8)
     fig.set_figheight(10)
     fig.set_figwidth(15)

     ix=0
     for i in range(4):
         for j in range(4):
             axs[i, 2*j].imshow(x_test[sample_index[4*i+j], :, :, 0], cmap='gray')
             axs[i, 2*j].axis('off')
             axs[i, 2*j+1].imshow(feature_map_conv1[sample_index[4*i+j], :, :,␣
      ↪filter_n1].eval(session=sess), cmap='gray')
             axs[i, 2*j+1].axis('off')
```

```
[ ]: plt.imshow(filters_conv1[:, :, 0, filter_n2], cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f445c033a20>
```



```
[ ]: fig, axs = plt.subplots(4, 8)
     fig.set_figheight(10)
     fig.set_figwidth(15)

     ix=0
     for i in range(4):
         for j in range(4):
             axs[i, 2*j].imshow(x_test[sample_index[4*i+j], :, :, 0], cmap='gray')
             axs[i, 2*j].axis('off')
             axs[i, 2*j+1].imshow(feature_map_conv1[sample_index[4*i+j], :, :,␣
      ↪filter_n2].eval(session=sess), cmap='gray')
             axs[i, 2*j+1].axis('off')
```



Comment on what you see in the feature maps. * How do they correspond to the original images? * How do they correspond to the filters? * Why might the feature maps be helpful for classifying digits?

1.1.3 Answer:

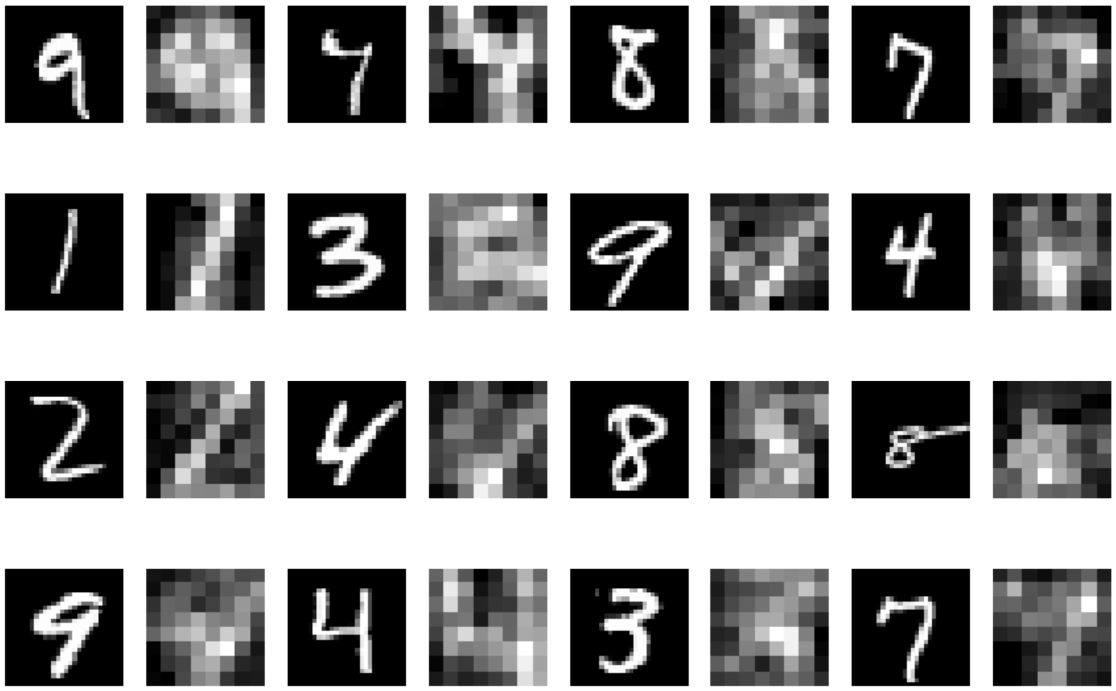
1. Both feature maps of filters 17 and 30 correspond to the original images. We are still able to tell the digits based on the feature maps.
2. For the first filter (#17), the feature maps seem to be the results of horizontal edge detection. For instance, the feature of number 9 in the second row has more emphasis on its horizontal axis. The second filter (#30) seems to detect edges in the oblique direction.
3. The feature maps are helpful to classify digits as their combination represent unique charac-

teristics of distinct digits. For instance, the feature map of digit 1 might be more sensible than other digits when filtered by a vertical edge detector.

1.1.4 1.3 Fitting a logistic regression model on feature maps

The features of the images are further summarized after the second convolutional layer.

```
[ ]: conv2_layer_model = keras.Model(inputs=model.input, outputs=model.
      ↪get_layer('conv2').output)
     feature_map_conv2 = conv2_layer_model(x_test)

     fig, axs = plt.subplots(4, 8)
     fig.set_figheight(10)
     fig.set_figwidth(15)

     ix=0
     for i in range(4):
         for j in range(4):
             axs[i, 2*j].imshow(x_test[sample_index[4*i+j], :, :, 0], cmap='gray')
             axs[i, 2*j].axis('off')
             axs[i, 2*j+1].imshow(feature_map_conv2[sample_index[4*i+j], :, :, 0].
      ↪eval(session=sess), cmap='gray')
             axs[i, 2*j+1].axis('off')
```



Build and test a logistic regression model to classify two digits of your choice using the features maps at the second convolutional layer as the input. You may use logistic regression functions such as LogisticRegression in sklearn. Use 80% of the data for training and 20% for test.

- How many features are there in your input X? Show the derivation of this number based on the architecture of the convolutional neural network.
- How is your logistic regression model related to the fully connected layer and softmax layer in the convolutional neural network?
- What is the accuracy of your model? Is this expected, or surprising?
- Comment on any other aspects of your findings that are interesting to you.

1.2 Answer:

1. I choose digit 1 and 3 for classification. The number of features can be calculated as ((28-4)/2-4)^2*32=2048.
2. Logistic regression shares many similar aspects than the fully connected layer and the softmax layer. The logistic regression computes the odds ratio using a linear transformation, which is similar to the fully-connected layer. The softmax function is similar to the logistic function, which is to compute the probability of certain prediction.
3. The logistic regression achieves 100% in testing, which is expected.
4. The feature maps in the second convolution layer is more high-level than the first convolution layer.

[ ]:
```
X_lr = np.reshape(feature_map_conv2.eval(session=sess),(np.
 ↪shape(feature_map_conv2.eval(session=sess))[0],-1))
y_lr = y_test

# extract feature maps for selected digits
digit_1 = 1
digit_2 = 3

indice_digit_1 = np.where(y_lr == digit_1)[0]
indice_digit_2 = np.where(y_lr == digit_2)[0]
indices = np.concatenate((indice_digit_1, indice_digit_2))
numOfSamples = indices.shape[0]

# construct training set
X_lr_train = X_lr[indices[:int(0.8*numOfSamples)]]
y_lr_train = y_lr[indices[:int(0.8*numOfSamples)]]

# construct testing set
X_lr_test = X_lr[indices[int(0.8*numOfSamples):]]
y_lr_test = y_lr[indices[int(0.8*numOfSamples):]]
```

[ ]:
```
# train the logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
clf = LogisticRegression(random_state=0).fit(X_lr_train, y_lr_train)
pred_lr_test = clf.predict(X_lr_test)
```

```
print('testing accuracy is',accuracy_score(pred_lr_test, y_lr_test))
```

```
testing accuracy is 1.0
```

1.3 Problem 2: Going for gold (15 points)

The 2022 Winter Olympics just finished, and this problem will have you looking forward to summer. You will use Gaussian process regression to model the trends in gold medal performances of selected events in the summer Olympics. The objectives of this problem are for you to:

- Gain experience with Gaussian processes, to better understand how they work
- Explore how posterior inference depends on the properties of the prior mean and kernel
- Use Bayesian inference to identify unusual events
- Practice making your Python code modular and reusable

For this problem, the only starter code we provide is to read in the data and extract one event. You may write any GP code that you choose to, but please do not use any package for Gaussian processes; your code should be "np.complete" (using only basic `numpy` methods). You are encouraged to start from the GP demo code used in class.

When we ran the GP demo code from class on the marathon data, it generated the following plot:

Note several properties of this plot: * It shows the Bayesian confidence of the regression, as a shaded area. This is a 95% confidence band because it has width given by $\pm 2\sqrt{V}$, where V is the estimated variance. The variance increases at the right side, for future years.

- The gold medal time for the 1904 marathon is outside of this confidence band. In fact, the 1904 marathon was an unusual event, and this is apparent from the model.

- The plot shows the posterior mean, and also shows one random sample from the posterior distribution.

Your task in this problem is generate such a plot for six different Olympic events by writing a function

`def gp_olympic_event(year, result, kernel, mean, noise, event_name):     ...`

where the input variables are the following:

- `year`: a numpy array of years (integers)
- `result`: a numpy array of numerical results, for the gold medal performances in that event
- `kernel`: a kernel function
- `mean`: a mean function
- `noise`: a single float for the variance of the noise, σ^2
- `event_name`: a string used to label the y-axis, for example "marathon min/mile (men's event)"

Your function should compute the Gaussian process regression, and then display the resulting plot, analogous to the plot above for the men's marathon event.

You will then process **six** of the events, three men's events and three women's events, and call your function to generate the corresponding six plots.

For each event, you should create a markdown cell that describes the resulting model. Comment on such things as:

- How you chose the kernel, mean, and noise.
- Why the plot does or doesn't look satisfactory to you
- If there are any events such as the 1904 marathon that are notable.
- What happens to the posterior mean (for example during WWII) if there are gaps in the data

Use your best judgement to describe your findings; post questions to EdD if things are unclear. And have fun!

In the remainder of this problem description, we recall how we processed the marathon data, as an example. The following cell reads in the data and displays the collection of events that are included in the dataset.

```
[ ]: import numpy as np
     import pandas as pd

     dat = pd.read_csv('https://raw.githubusercontent.com/YData123/sds365-sp22/main/
      ↪demos/gaussian_processes/olympic_results.csv')
     events = set(np.array(dat['Event']))
     print(events)
```

```
{'400M Women', '110M Hurdles Men', '1500M Men', 'High Jump Men', '100M Hurdles
Women', 'Discus Throw Men', '4X100M Relay Women', 'High Jump Women', '1500M
Women', 'Marathon Women', 'Heptathlon Women', 'Decathlon Men', 'Long Jump Men',
'3000M Steeplechase Women', 'Hammer Throw Men', '20Km Race Walk Women', '800M
Women', '5000M Women', '800M Men', '200M Men', '400M Men', 'Shot Put Women',
'100M Men', '4X400M Relay Men', 'Discus Throw Women', 'Triple Jump Women', '400M
Hurdles Men', '400M Hurdles Women', 'Hammer Throw Women', '4X100M Relay Men',
'Javelin Throw Men', 'Long Jump Women', 'Shot Put Men', 'Marathon Men', '10000M
Women', '20Km Race Walk Men', '5000M Men', 'Pole Vault Men', 'Pole Vault Women',
'Triple Jump Men', '50Km Race Walk Men', 'Javelin Throw Women', '100M Women',
'3000M Steeplechase Men', '10000M Men', '200M Women', '4X400M Relay Women'}
```

We then process the time to compute the minutes per mile (without checking that the race was actually 26.2 miles!)

```
[ ]: marathon = dat[dat['Event'] == 'Marathon Men']
     marathon = marathon[marathon['Medal']=='G']
     marathon = marathon.sort_values('Year')
     time = np.array(marathon['Result'])
     mpm = []
     for tm in time:
         t = np.array(tm.split(':'), dtype=float)
         minutes_per_mile = (t[0]*60*60 + t[1]*60 + t[2])/(60*26.2)
         mpm.append(minutes_per_mile)

     marathon['Minutes per Mile'] = np.round(mpm,2)
     marathon = marathon.drop(columns=['Gender', 'Event'], axis=1)
     marathon.reset_index(drop=True, inplace=True)
```

```
year = np.array(marathon['Year'])
result = np.array(marathon['Minutes per Mile'])
marathon
```

```
[ ]:              Location  Year Medal                 Name Nationality  \
0                 Athens  1896     G        Spyridon LOUIS         GRE
1                  Paris  1900     G         Michel THÃ ATO        FRA
2               St Louis  1904     G          Thomas HICKS         USA
3                 London  1908     G            John HAYES         USA
4              Stockholm  1912     G  Kennedy Kane MCARTHUR         RSA
5                Antwerp  1920     G     Hannes KOLEHMAINEN         FIN
6                  Paris  1924     G         Albin STENROOS         FIN
7              Amsterdam  1928     G      BoughÃ¨ra EL OUAFI         FRA
8            Los Angeles  1932     G      Juan Carlos ZABALA         ARG
9                 Berlin  1936     G              Kitei SON         JPN
10                London  1948     G          Delfo CABRERA         ARG
11              Helsinki  1952     G           Emil ZÃ TOPEK        TCH
12  Melbourne / Stockholm  1956     G           Alain MIMOUN         FRA
13                  Rome  1960     G           Abebe BIKILA         ETH
14                 Tokyo  1964     G           Abebe BIKILA         ETH
15                Mexico  1968     G             Mamo WOLDE         ETH
16                Munich  1972     G  Frank Charles SHORTER         USA
17              Montreal  1976     G    Waldemar CIERPINSKI         GDR
18                Moscow  1980     G    Waldemar CIERPINSKI         GDR
19           Los Angeles  1984     G           Carlos LOPES         POR
20                 Seoul  1988     G         Gelindo BORDIN         ITA
21             Barcelona  1992     G        Young-Cho HWANG         KOR
22               Atlanta  1996     G         Josia THUGWANE         RSA
23                Sydney  2000     G        Gezahegne ABERA         ETH
24                Athens  2004     G        Stefano BALDINI         ITA
25               Beijing  2008     G    Samuel Kamau WANJIRU         KEN
26                London  2012     G      Stephen KIPROTICH         UGA
27                   Rio  2016     G   Eliud Kipchoge ROTICH         KEN

       Result  Minutes per Mile
0     2:58:50              6.83
1   2:59:45.0              6.86
2   3:28:53.0              7.97
3   2:55:18.4              6.69
4   2:36:54.8              5.99
5   2:32:35.8              5.82
6   2:41:22.6              6.16
7     2:32:57              5.84
8     2:31:36              5.79
9   2:29:19.2              5.70
10  2:34:51.6              5.91
11  2:23:03.2              5.46
```

```
12     2:25:00              5.53
13   2:15:16.2              5.16
14   2:12:11.2              5.05
15   2:20:26.4              5.36
16   2:12:19.8              5.05
17   2:09:55.0              4.96
18   2:11:03.0              5.00
19     2:09:21              4.94
20     2:10:32              4.98
21     2:13:23              5.09
22     2:12:36              5.06
23     2:10:11              4.97
24     2:10:55              5.00
25     2:06:32              4.83
26     2:08:01              4.89
27    02:08:44              4.91
```

Enter your code and markdown following this cell.

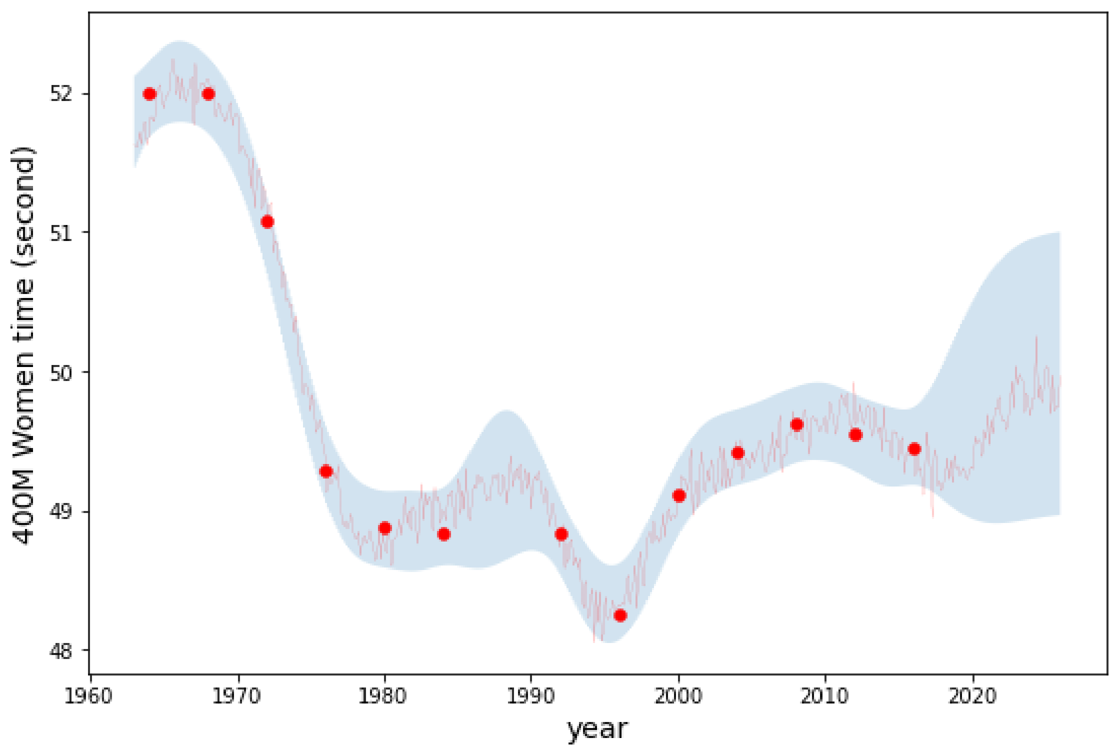
```
[ ]: import matplotlib.pyplot as plt
     import numpy as np
     from numpy.linalg import cholesky

     def gaussian_sample(mu, Sigma):
         A = cholesky(Sigma)
         Z = np.random.normal(loc=0, scale=1, size=len(mu))
         return np.dot(A, Z) + mu

     def gp_olympic_event(year, result, kernel, mean, noise, event_name):
         X_train = year
         f_train = result
         xs = np.linspace(min(year)-1, max(year)+10, 500) # new data points

         K = kernel(X_train, X_train)
         Ks = kernel(X_train, xs)
         sigma2 = noise
         Kss = kernel(xs, xs) + sigma2 * np.eye(len(xs))
         Ki = np.linalg.inv(K + sigma2 * np.eye(len(X_train)))

         postMu = mean(xs) + Ks.T @ Ki @ (f_train - mean(X_train))
         postCov = Kss - Ks.T @ Ki @ Ks

         fig, ax = plt.subplots(1, 1, figsize=(9, 6))
         S2 = np.diag(postCov)
         ax.fill_between(xs, postMu - 2*np.sqrt(S2), postMu + 2*np.sqrt(S2),␣
      ↪step="pre", alpha=0.2, label='posterior 95% confidence')
         fs = gaussian_sample(postMu, postCov)
```

```
    ax.plot(xs, fs, c='r', linestyle='-', linewidth=.1)
    ax.scatter(X_train, f_train, color='red', marker='o', linewidth=.4)
    ax.set_xlabel('year', fontsize=14)
    ax.set_ylabel("{}".format(str(event_name)), fontsize=14)
```

1.4 Event 1: 400M Women

Dicussion: We choose Gaussian kernel with $h = 4$ and noise $\sigma^2 = 0.01$ empirically. The mean function ($\mu = 50$) of the prior distribution is chosen based on the rough estimate of the time that a female athelet running for 400 meters. From the figure below, it seems that posterior distribution fits quite well with the true data. We also observe a gap around 1970, which shows that the atheletes' performance improve quite significantly. It is also worthwhile noting that the current olympic record was set before 2000, which is quite impressive.

```
[ ]: def mu_fun(x, mu=50):
         return mu * np.ones(len(x))

     def K_fun(x, z, h=4):
         K = np.zeros(len(x)*len(z)).reshape(len(x), len(z))
         for j in np.arange(K.shape[1]):
             K[:,j] = (1/h)*np.exp(-(x-z[j])**2/(2*h**2))
         return K

     fourhundred_women = dat[dat['Event'] == '400M Women']
     fourhundred_women = fourhundred_women[fourhundred_women['Medal']=='G']
     fourhundred_women = fourhundred_women.sort_values('Year')
     year = np.array(fourhundred_women['Year'])
     result = np.array(fourhundred_women['Result'], dtype=float)
     event_label = '400M Women time (second)'

     gp_olympic_event(year, result, K_fun, mu_fun, noise=0.01,␣
      ↪event_name=event_label)
```

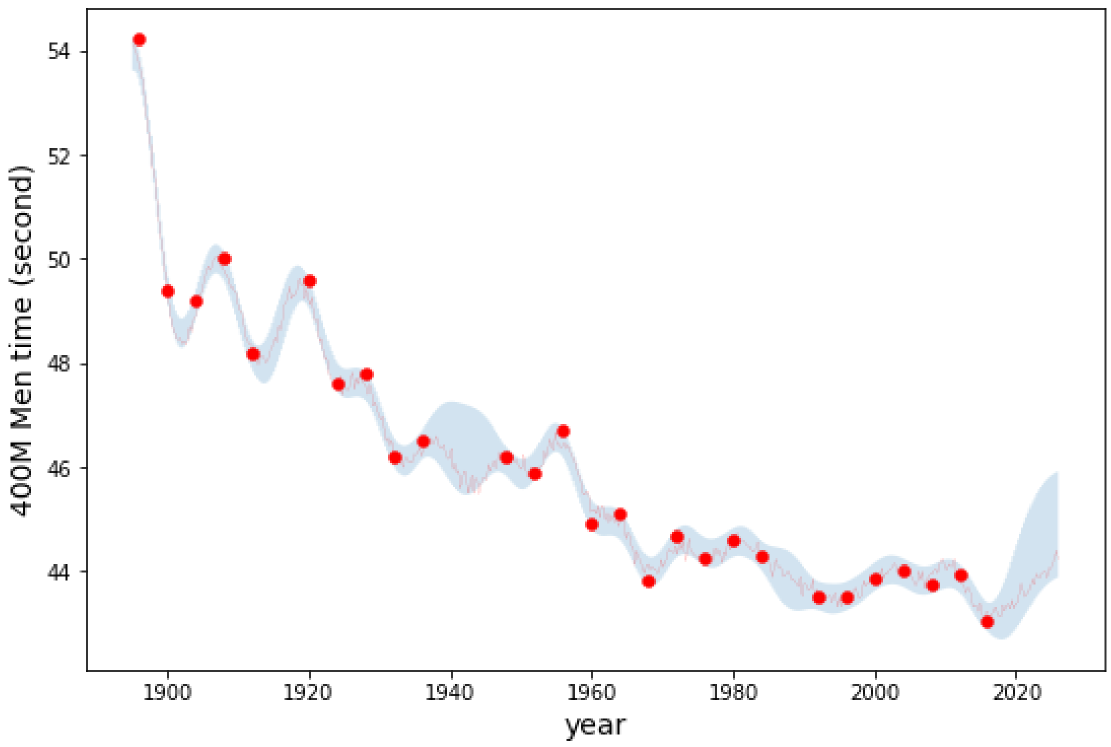


1.5 Event 2: 400M Men

Discussion: We choose Gaussian kernel with $h = 4$ and noise $\sigma^2 = 0.01$ empirically. The mean function ($\mu = 45$) of the prior distribution is chosen based on the rough estimate of the time that a male athelet running for 400 meters. From the figure below, it seems that posterior distribution fits quite well with the true data. The atheles performance are also improving after 1900.

[ ]:
```
def mu_fun(x, mu=45):
    return mu * np.ones(len(x))

def K_fun(x, z, h=4):
    K = np.zeros(len(x)*len(z)).reshape(len(x), len(z))
    for j in np.arange(K.shape[1]):
        K[:,j] = (1/h)*np.exp(-(x-z[j])**2/(2*h**2))
    return K

fourhundred_men = dat[dat['Event'] == '400M Men']
fourhundred_men = fourhundred_men[fourhundred_men['Medal']=='G']
fourhundred_men = fourhundred_men.sort_values('Year')
year = np.array(fourhundred_men['Year'])

result_list = []
time = np.array(fourhundred_men['Result'])
```

```
for tm in time:
    if float(tm.split(':')[0])!=0:
        t = float(tm.split(':')[0])
    else:
        t = float(tm.split(':')[1])
    result_list.append(t)
result = np.array(result_list)
event_label = '400M Men time (second)'
result
gp_olympic_event(year, result, K_fun, mu_fun, noise=0.01,␣
 ↪event_name=event_label)
```

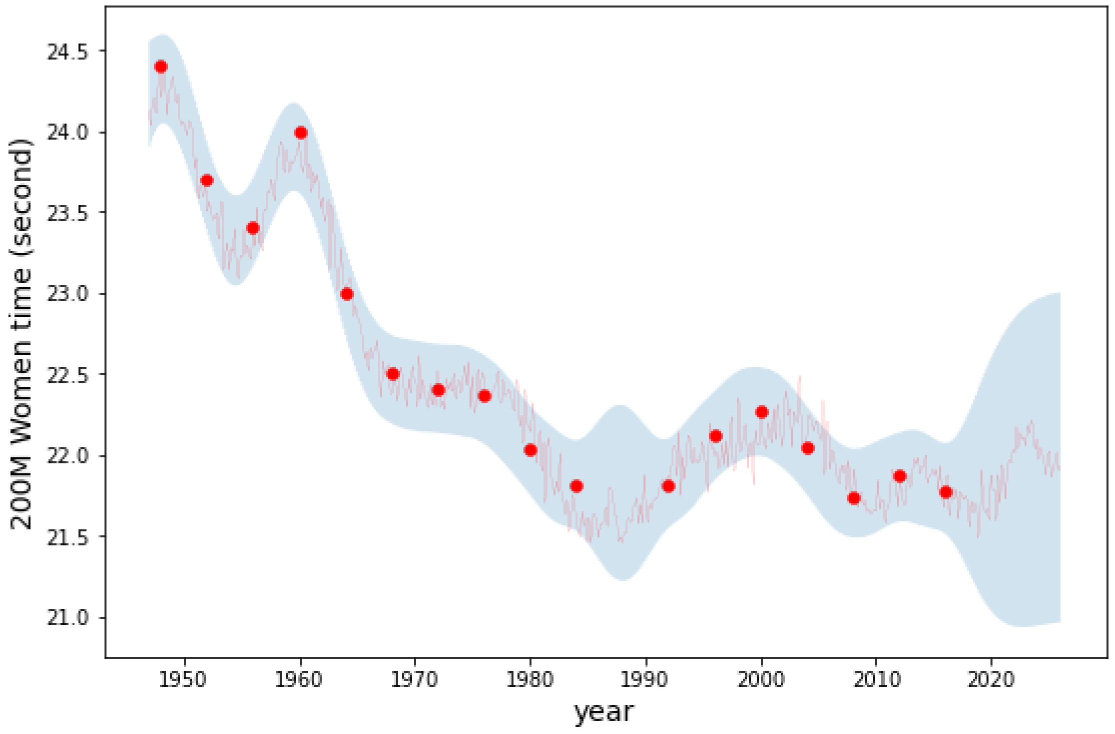


1.6 Event 3 200M Women

Discussion: We choose Gaussian kernel with $h = 4$ and noise $\sigma^2 = 0.01$ empirically. The mean function ($\mu = 22$) of the prior distribution is chosen based on the rough estimate of the time that a female athelet running for 200 meters. From the figure below, it seems that posterior distribution fits quite well with the true data. The female athelets performance are also improving throughout the years.

```
[ ]: def mu_fun(x, mu=22):
         return mu * np.ones(len(x))
```

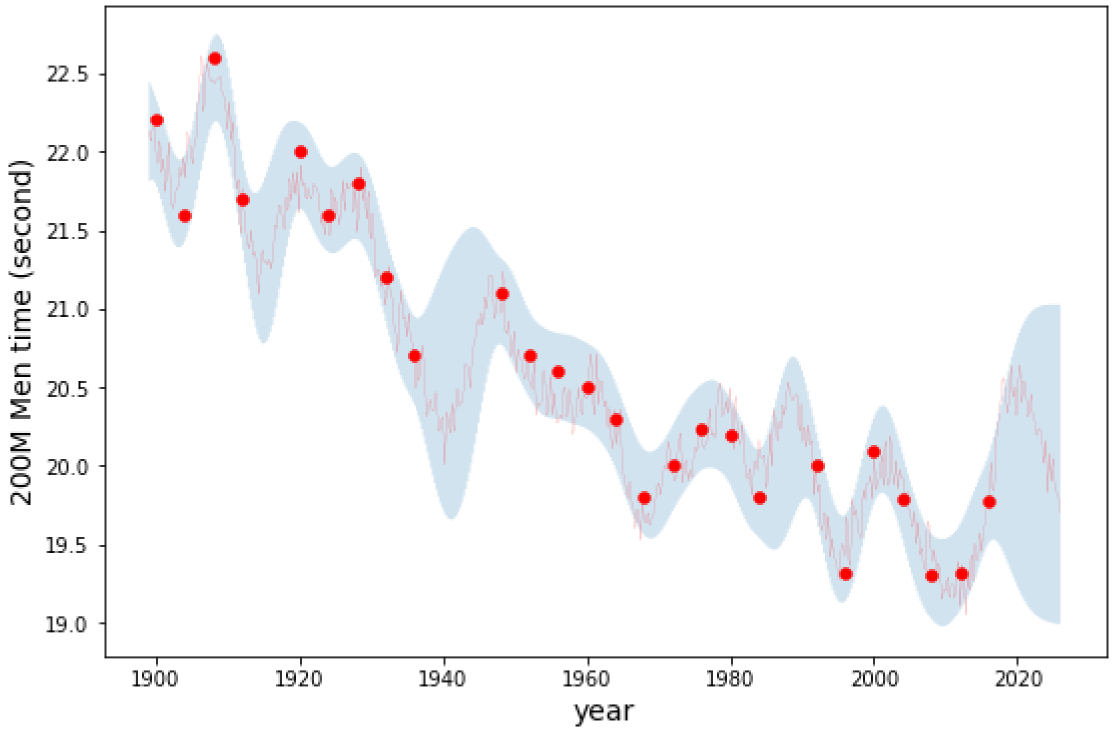
```
def K_fun(x, z, h=4):
    K = np.zeros(len(x)*len(z)).reshape(len(x), len(z))
    for j in np.arange(K.shape[1]):
        K[:,j] = (1/h)*np.exp(-(x-z[j])**2/(2*h**2))
    return K

twohundred_women = dat[dat['Event'] == '200M Women']
twohundred_women = twohundred_women[twohundred_women['Medal']=='G']
twohundred_women = twohundred_women.sort_values('Year')
year = np.array(twohundred_women['Year'])
result = np.array(twohundred_women['Result'], dtype=float)
event_label = '200M Women time (second)'

gp_olympic_event(year, result, K_fun, mu_fun, noise=0.01,␣
↪event_name=event_label)
```



1.7 Event 4 200 Men

Discussion: We choose Gaussian kernel with $h = 4$ and noise $\sigma^2 = 0.01$ empirically. The mean function ($\mu = 20$) of the prior distribution is chosen based on the rough estimate of the time that a male athelet running for 200 meters. From the figure below, it seems that posterior distribution fits quite well with the true data. The athletes' performances are also improving.

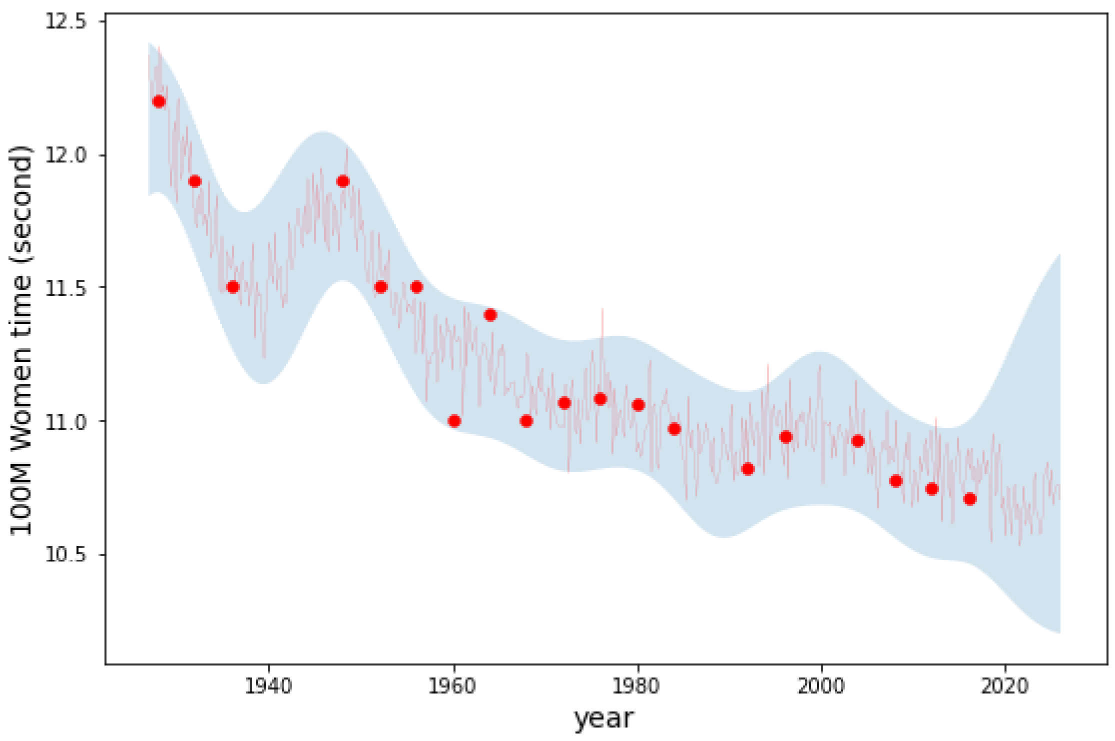
```
[ ]: def mu_fun(x, mu=20):
         return mu * np.ones(len(x))

     def K_fun(x, z, h=4):
         K = np.zeros(len(x)*len(z)).reshape(len(x), len(z))
         for j in np.arange(K.shape[1]):
             K[:,j] = (1/h)*np.exp(-(x-z[j])**2/(2*h**2))
         return K

     twohundred_men = dat[dat['Event'] == '200M Men']
     twohundred_men = twohundred_men[twohundred_men['Medal']=='G']
     twohundred_men = twohundred_men.sort_values('Year')
     year = np.array(twohundred_men['Year'])
     result = np.array(twohundred_men['Result'], dtype=float)
     event_label = '200M Men time (second)'

     gp_olympic_event(year, result, K_fun, mu_fun, noise=0.01,␣
      ↪event_name=event_label)
```



1.8 Event 5 100 Women

Discussion: We choose Gaussian kernel with $h = 7$ and noise $\sigma^2 = 0.01$ empirically. The mean function ($\mu = 11$) of the prior distribution is chosen based on the rough estimate of the time that a female athelet running for 100 meters. From the figure below, it seems that posterior distribution fits quite well with the true data. We can see that the atheles performances improve quite significantly from 1940 to 1960.

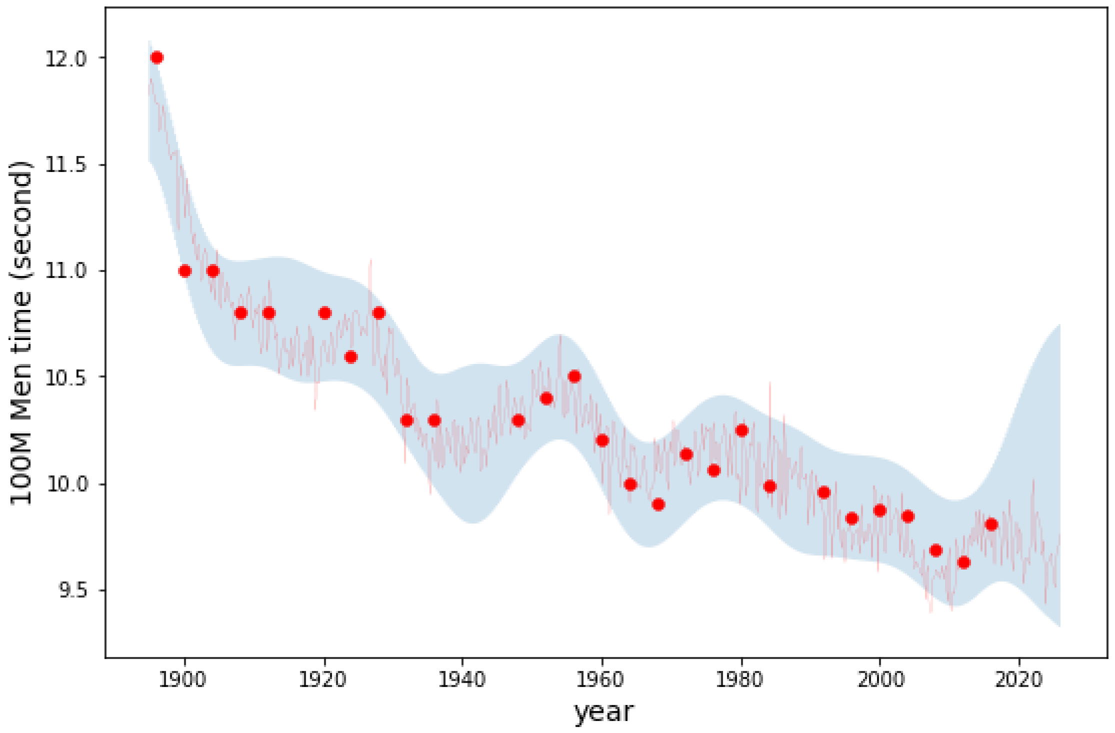
```
[ ]: def mu_fun(x, mu=11):
         return mu * np.ones(len(x))

     def K_fun(x, z, h=7):
         K = np.zeros(len(x)*len(z)).reshape(len(x), len(z))
         for j in np.arange(K.shape[1]):
             K[:,j] = (1/h)*np.exp(-(x-z[j])**2/(2*h**2))
         return K

     onehundred_women = dat[dat['Event'] == '100M Women']
     onehundred_women = onehundred_women[onehundred_women['Medal']=='G']
     onehundred_women = onehundred_women.sort_values('Year')
     year = np.array(onehundred_women['Year'])
     result = np.array(onehundred_women['Result'], dtype=float)
     event_label = '100M Women time (second)'

     gp_olympic_event(year, result, K_fun, mu_fun, noise=0.01,␣
      ↪event_name=event_label)
```



1.9 Event 6 100 Men

Discussion: We choose Gaussian kernel with $h = 7$ and noise $\sigma^2 = 0.01$ empirically. The mean function ($\mu = 10$) of the prior distribution is chosen based on the rough estimate of the time that a male athelet running for 100 meters. From the figure below, it seems that posterior distribution fits quite well with the true data. There seems to be a large gap around 1900 and athelets performances are steadly improving since 1980.

```
[ ]: def mu_fun(x, mu=10):
         return mu * np.ones(len(x))

     def K_fun(x, z, h=7):
         K = np.zeros(len(x)*len(z)).reshape(len(x), len(z))
         for j in np.arange(K.shape[1]):
             K[:,j] = (1/h)*np.exp(-(x-z[j])**2/(2*h**2))
         return K

     onehundred_men = dat[dat['Event'] == '100M Men']
     onehundred_men = onehundred_men[onehundred_men['Medal']=='G']
     onehundred_men = onehundred_men.sort_values('Year')
     year = np.array(onehundred_men['Year'])
     result = np.array(onehundred_men['Result'], dtype=float)
     event_label = '100M Men time (second)'

     gp_olympic_event(year, result, K_fun, mu_fun, noise=0.01,␣
      ↪event_name=event_label)
```



1.10 Problem 3: It's the process, not the product (15 points)

In this problem, you will explore the properties of the Dirichlet process, and get some practice drawing samples from a DP.

1.10.1 3.1: Be consistent

Let $F \sim \text{DP}(\alpha, F_0)$ be drawn from a Dirichlet process with parameters α and F_0. Show that $\mathbb{E}(F) = F_0$.

1.11 Answer:

From the stick breaking process, we have s_i drawn from F_0. For the pmf of distribution F as a random variable over s_i, we have

$$\mathbb{E}[F(s_i)] = \mathbb{E}[p(s_i, w_i)] \tag{1}$$

As w_i is drawn independently with s_i and thus we have

$$\begin{aligned} \mathbb{E}[F(s_i)] &= \mathbb{E}_{w_i \sim W}[p(s_i)] & (2)\\ &= F_0(s_i) \sum_{j=1}^{N} w_j & (3)\\ &= F_0(s_i) & (4) \end{aligned}$$

The above can be interepted as the expected stick length (w_i) on location s_i, which needs to be averaged across all possible assignments of w_i, which is independent of s_i. Thus, we have $\mathbb{E}[F] = F_0$.

1.11.1 3.2: Concentrate

If $F \sim \text{DP}(\alpha, F_0)$, show that the prior gets more concentrated around F_0 as $\alpha \to \infty$.

1.12 Answer:

As each value draw from the DP is Dirichlet distributed. From the definition of Dirichlet distribution, the variance can be calculated as

$$\text{Var}[F] = \frac{F_0(1 - F_0)}{1 + \alpha} \tag{5}$$

When $\alpha \to \infty$, the variance of F approximates to zero. Thus the prior gets more concentrated.

1.12.1 3.3: Repeat after me

Let $F \sim \text{DP}(\alpha, F_0)$ be drawn from a Dirichlet process with parameters α and F_0, and let $X_1, X_2, \ldots, X_n \mid F \sim F$ be drawn from F. What is the posterior mean of $F(x)$ given $X_1, \ldots, X_n$?

1.13 Answer:

As the Drichelet distribution is a conjugate distribution, the posterior of DP is also a DP. The posterior mean of $F(x)$ can be expressed as

$$\bar{F}_n = \frac{n}{n+\alpha} F_n + \frac{\alpha}{n+\alpha} F_0 \tag{6}$$

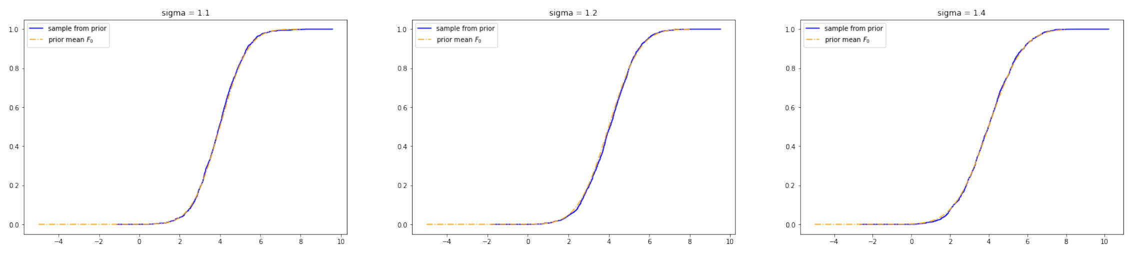
which can be explained from the Chinese resturant process. F_n here is the empirical distribution of $X_1, ..., X_n$ The posterior mean is a weighted sum of MLE estimator and base distribution. When the number of samples gets larger, the estimator relies more on the empirical distribution and vice versa.

1.13.1 3.4: Seeing is believing

Write Python code to illustrate property 3.1 above: $\mathbb{E}F(x) = F_0(x)$. Take the DP demo code used in class as a starting point. You should draw many samples from the prior, average them, and then compare to F_0. You can let F_0 be Gaussian or another distribution. Try several different values of α.

[ ]:
```python
from scipy.stats import norm

def stick_break(alpha, N):
    V = np.random.beta(1, alpha, size=N)
    w = np.zeros(N)
    w[0] = V[0]
    d = 1
    for i in np.arange(1, N):
        d = d*(1-V[i-1])
        w[i] = V[i]*d
    return w/np.sum(w)

def sample_dirichlet_process(alpha, mu, sigma):
    N = 1000
    s = np.random.normal(loc=mu, scale=sigma, size=N)
    w = stick_break(alpha, N)
    ind = np.argsort(s)
    s = s[ind]
    w = w[ind]
    return s, w

mu0 = 4
alpha = 10
n_rep = 100

sigma_list = [1.1, 1.2, 1.4]

fig, axes = plt.subplots(1, 3)
fig.set_size_inches((30, 6))
```

```
for idx, sigma0 in enumerate(sigma_list):
    s_list, w_list = [], []
    for idx_rep in range(n_rep):
        s, w = sample_dirichlet_process(alpha=alpha, mu=mu0, sigma=sigma0)

        # average across past distribution draws of F
        for idx_ele in range(len(s)):
            s_list.append(s[idx_ele])
            w_list.append(w[idx_ele])

    s_arr = np.array(s_list)
    w_arr = np.array(w_list)
    indices = np.argsort(s_arr)
    s_arr = s_arr[indices]
    w_arr = w_arr[indices]
    w_arr /= n_rep

    axes[idx].step(s_arr, np.cumsum(w_arr), label='sample from prior',␣
 ↪color='blue')

    xs = np.linspace(-5, 8, 500)
    axes[idx].plot(xs, norm.cdf(xs, loc=mu0, scale=sigma0),␣
 ↪linestyle='dashdot', color='orange', label=r'prior mean $F_0$')
    axes[idx].set_title('sigma = {}'.format(str(sigma0)))

    axes[idx].legend()
```


sigma = 1.1
sigma = 1.2
sigma = 1.4
sample from prior
prior mean $F_0$