MAS374 Optimization theory
Homework #3

20150597 Jeonghwan Lee

Department of Mathematical Sciences, KAIST

October 6, 2021

I worked on this programming assignment by using Python 3 (version 3.7.7). I utilized PyCharm 2021.1 Community Edition as an integrated development environment (IDE).

(a) The next code defines a function `my_lstsq(A, y)` which takes $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{y} \in \mathbb{R}^{m}$ as its input and employs the singular value decomposition of $\mathbf{A}$ to compute and return the optimal solution $\boldsymbol{\theta}^* = \mathbf{A}^{\dagger}\mathbf{y} \in \mathbb{R}^{n}$:

```python
import numpy as np

def my_lstsq(A, y):
    m = A.shape[0]
    n = A.shape[1]
    r = np.linalg.matrix_rank(A)
    u, s, vh = np.linalg.svd(A, full_matrices=True)
    pseudo_smat = np.zeros((n, m))
    for i in range(r):
        pseudo_smat[i, i] = 1/s[i]
    pseudo_inv_A = np.dot(np.transpose(vh), np.dot(pseudo_smat, np.transpose(u)))
    theta = np.dot(pseudo_inv_A, y)
    return theta
```

For a simpler implementation of the function `my_lstsq(A, y)`, we may use the following code which defines a function `my_lstsq_simpler(A, y)` that uses the Moore-Penrose pseudo-inverse $\mathbf{A}^{\dagger} \in \mathbb{R}^{n \times m}$ directly. Note that the function `my_lstsq_simpler(A, y)` does not use the SVD of $\mathbf{A}$ in the code:

```python
import numpy as np

def my_lstsq_simpler(A, y):
    pseudoinv_A = np.linalg.pinv(A)
    theta = np.dot(pseudoinv_A, y)
    return theta
```

It is worth to notice that both the functions `my_lstsq(A, y)` and `my_lstsq_simpler(A, y)` requires no full column-rank condition of $\mathbf{A}$; $m \geq n = \mathsf{rank}(\mathbf{A})$. So the above codes provide the optimal solution $\boldsymbol{\theta}^* = \mathbf{A}^{\dagger}\mathbf{y}$ under the fully general case.

(b) Here, we generate n samples $\left\{\mathbf{x}^{(i)} := \left(x_1^{(i)}, x_2^{(i)}\right) : i \in [n]\right\}$ from $\mathsf{Unif}\left([-2,2] \times [-2,2]\right)$. Note that we are letting $n = 250$ in this problem. We first label these sample points according to the rule

$$y_i = \begin{cases} +1 & \text{if } \left(x_1^{(i)}\right)^2 + \left(x_2^{(i)}\right)^2 \leq 1; \\ -1 & \text{otherwise,} \end{cases} \tag{1}$$

and let $\mathbf{y} := \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^\top \in \mathbb{R}^n$. The construction of the samples $\left\{\mathbf{x}^{(i)} := \left(x_1^{(i)}, x_2^{(i)}\right) : i \in [n]\right\}$ and the label vector $\mathbf{y} \in \mathbb{R}^n$ (here, $n = 250$) can be implemented through the following code:

```python
import numpy as np
import random

def label(a, b):     # label a given point in \mathbb{R}^2
    if a**2 + b**2 <= 1:
        return 1
    else:
        return -1

length = 250     # number of samples
first_coordinate_1 = np.dot(4, np.random.rand(length)) - np.dot(2, np.ones(length))    # the first coordinates of 250 samples chosen uniformly at random from [-2, 2] \times [-2, 2]
second_coordinate_1 = np.dot(4, np.random.rand(length)) - np.dot(2, np.ones(length))    # the second coordinates of 250 samples chosen uniformly at random from [-2, 2] \times [-2, 2]
y_1 = np.zeros(length)     # initialize vector of labels for part (b)
num_1 = 0
for i in range(length):
    y_1[i] = label(first_coordinate_1[i], second_coordinate_1[i])     # label the 250 random sample points according to the rule (4)
    if y_1[i] > 0:
        num_1 = num_1 + 1
```

Next, we would like to construct our coefficient matrix $\mathbf{A} \in \mathbb{R}^{n \times 6}$ from the samples $\left\{\mathbf{x}^{(i)} := \left(x_1^{(i)}, x_2^{(i)}\right) : i \in [n]\right\}$ as follows: for every $i \in [n]$, the i-th row vector of $\mathbf{A}$ is given by

$$\mathbf{A}_{i*} := \begin{bmatrix} 1 & x_1^{(i)} & x_2^{(i)} & x_1^{(i)} x_2^{(i)} & \left(x_1^{(i)}\right)^2 & \left(x_2^{(i)}\right)^2 \end{bmatrix}$$

The construction of the coefficient matrix $\mathbf{A} \in \mathbb{R}^{n \times 6}$ can be implemented through the following code:

```python
import numpy as np

def generate_data_matrix(first_coordinate, second_coordinate):     # construct a proper data matrix (2-dimensional array) of shape length \times 6
    input_length = first_coordinate.shape[0]
    data_matrix = np.zeros((length, 6))
    for i in range(input_length):
        data_matrix[i, 0] = 1
        data_matrix[i, 1] = first_coordinate[i]
        data_matrix[i, 2] = second_coordinate[i]
```

```
        data_matrix[i, 3] = first_coordinate[i]**2
        data_matrix[i, 4] = first_coordinate[i]*second_coordinate[i]
        data_matrix[i, 5] = second_coordinate[i]**2
    return data_matrix
```

By utilizing the coefficient matrix $\mathbf{A} \in \mathbb{R}^{n\times 6}$ and the label vector $\mathbf{y} \in \mathbb{R}^n$, we can compute the LS solution $\boldsymbol{\theta}^* = \mathbf{A}^\dagger \mathbf{y} \in \mathbb{R}^6$, which is an optimal solution to the following least-squares problem:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^6} \|\mathbf{y} - \mathbf{A}\boldsymbol{\theta}\|_2^2 . \tag{2}$$

This part can be performed by using the function `my_lstsq(A, y)`:

```
import numpy as np

A_1 = generate_data_matrix(first_coordinate_1, second_coordinate_1)
theta_1 = my_lstsq(A_1, y_1)  # compute the optimal order-2 polynomial that minimizes (3)
print(theta_1)
```

Lastly, we would like to determine how the decision boundary would look like in a quantitative way, and visualize it by using matplotlib.pyplot. To this end, we first introduce how to classify the conic sections by using its discriminant. We consider the conic section $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$. The discriminant of this conic section is defined by $\Delta := B^2 - 4AC$. Then it is well-known that

1. if $A = C$ and $B = 0$, then the equation $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$ represents a circle, which is a special case of an ellipse;
2. if $\Delta < 0$, then the equation $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$ represents an ellipse;
3. if $\Delta = 0$, then the equation $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$ represents a parabola;
4. if $\Delta > 0$, then the equation $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$ represents a hyperbola.

See [1] for further details of the classification of conic sections. Utilizing these facts, one can design a function `conic_section_discriminant(A, B, C, D, E, F)` as follows:

```
def conic_section_discriminant(A, B, C, D, E, F):        # discriminate the conic section
    Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0
    discriminant = B**2 - 4*A*C
    if A == C and B == 0:
        return "Circle"
    elif discriminant < 0:
        return "Ellipse"
    elif discriminant == 0:
        return "Parabola"
    else:
        return "Hyperbola"
```

Upon establishing this function, we are now able to verify the shape of the decision boundary by using the above facts and visualize it by implementing the following code:

```
import numpy as np
import random
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt

def axes():
    plt.axhline(0, alpha=.1)
    plt.axvline(0, alpha=.1)

print(conic_section_discriminant(theta_1[3], theta_1[4], theta_1[5], theta_1[1], theta_1[2], theta_1[0]))      # discriminate the conic section formed as the zero set of the optimal order-2 polynomial that minimizes (3) (= the decision boundary)

x = np.linspace(-5, 5, 4000)
y = np.linspace(-5, 5, 4000)
x, y = np.meshgrid(x, y)
axes()
plt.contour(x, y, (theta_1[3]*x**2 + theta_1[4]*x*y + theta_1[5]*y**2 + theta_1[1]*x + theta_1[2]*y + theta_1[0]), [0], colors='k')
plt.show()
```

To sum up, the overall code for part (b) can be encapsulated as follows:

```
import numpy as np
import random
import matplotlib as mpl
import matplotlib.pyplot as plt

def axes():
    plt.axhline(0, alpha=.1)
    plt.axvline(0, alpha=.1)

def label(a, b):     # label a given point in \mathbb{R}^2 according to the rule (4)
    if a**2 + b**2 <= 1:
        return 1
    else:
        return -1

def generate_data_matrix(first_coordinate, second_coordinate):      # construct a proper data matrix (2-dimensional array) of shape length \times 6
    input_length = first_coordinate.shape[0]
    data_matrix = np.zeros((length, 6))
    for i in range(input_length):
        data_matrix[i, 0] = 1
        data_matrix[i, 1] = first_coordinate[i]
        data_matrix[i, 2] = second_coordinate[i]
        data_matrix[i, 3] = first_coordinate[i]**2
        data_matrix[i, 4] = first_coordinate[i]*second_coordinate[i]
        data_matrix[i, 5] = second_coordinate[i]**2
    return data_matrix

def conic_section_discriminant(A, B, C, D, E, F):       # discriminate the conic section Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0
```

```python
    discriminant = B**2 - 4*A*C
    if A == C and B == 0:
        return "Circle"
    elif discriminant < 0:
        return "Ellipse"
    elif discriminant == 0:
        return "Parabola"
    else:
        return "Hyperbola"

length = 250     # number of samples
first_coordinate_1 = np.dot(4, np.random.rand(length)) - np.dot(2, np.ones(length))    # the first coordinates of 250 samples chosen uniformly at random from [-2, 2] \times [-2, 2]
second_coordinate_1 = np.dot(4, np.random.rand(length)) - np.dot(2, np.ones(length))    # the second coordinates of 250 samples chosen uniformly at random from [-2, 2] \times [-2, 2]
y_1 = np.zeros(length)     # initialize vector of labels for part (b)
num_1 = 0
for i in range(length):
    y_1[i] = label(first_coordinate_1[i], second_coordinate_1[i])     # label the 250 random sample points according to the rule (4)
    if y_1[i] > 0:
        num_1 = num_1 + 1

A_1 = generate_data_matrix(first_coordinate_1, second_coordinate_1)
theta_1 = my_lstsq(A_1, y_1)     # compute the optimal order-2 polynomial that minimizes (3)
print(theta_1)
print(conic_section_discriminant(theta_1[3], theta_1[4], theta_1[5], theta_1[1], theta_1[2], theta_1[0]))     # discriminate the conic section formed as the zero set of the optimal order-2 polynomial that minimizes (3) (= the decision boundary)

x = np.linspace(-5, 5, 4000)
y = np.linspace(-5, 5, 4000)
x, y = np.meshgrid(x, y)
axes()
plt.contour(x, y, (theta_1[3]*x**2 + theta_1[4]*x*y + theta_1[5]*y**2 + theta_1[1]*x + theta_1[2]*y + theta_1[0]), [0], colors='k')
plt.show()
```

Finally, it's time to demonstrate some simulation results of the above code. The following figures show the visualizations of decision boundaries when we sample $\left\{\mathbf{x}^{(i)} := \left(x_1^{(i)}, x_2^{(i)}\right) : i \in [250]\right\}$ uniformly at random from $[-2, 2] \times [-2, 2]$ over 5 trials. We note that the image on the right side of each trial is the corresponding output result produced by `print()` functions in the above code. The first array together with the string "`Ellipse`" are the implementation result of the code for part (b), while the second array together with the string "`Hyperbola`" are the implementation result of the code for part (c).

Figure 1: The decision boundary and outputs for part (b); Trial #1

Figure 2: The decision boundary and outputs for part (b); Trial #2

Figure 3: The decision boundary and outputs for part (b); Trial #3

Figure 4: The decision boundary and outputs for part (b); Trial #4

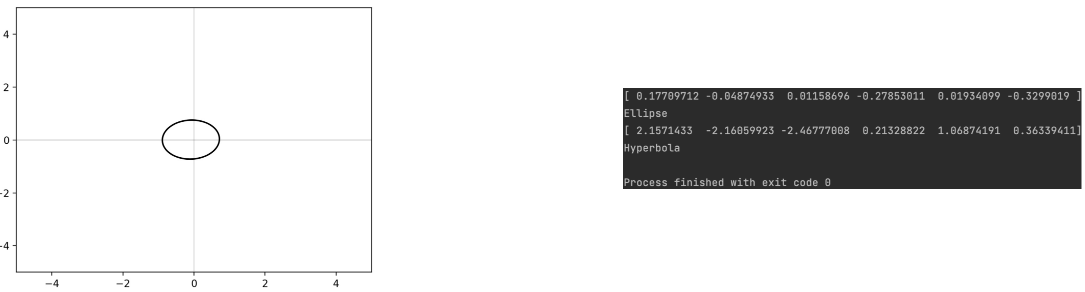


Figure 5: The decision boundary and outputs for part (b); Trial #5

As Figure 1–5 show, we can corroborate that the decision boundary $f(\mathbf{x}) = 0$ of part (b) forms an *ellipse* in $\mathbb{R}^2$ whose center is located nearby the origin $(0, 0)$ via a simple straightforward visualization. Why would the decision boundary $f(\mathbf{x}) = 0$ for part (b) have this shape? One can make a fairly reasonable guess based on the following intuition: since the optimal polynomial

$$f(\mathbf{x}) = f(x_1, x_2) := \theta_1^* + \theta_2^* x_1 + \theta_3^* x_2 + \theta_4^* x_1^2 + \theta_5^* x_1 x_2 + \theta_6^* x_2^2 \tag{3}$$

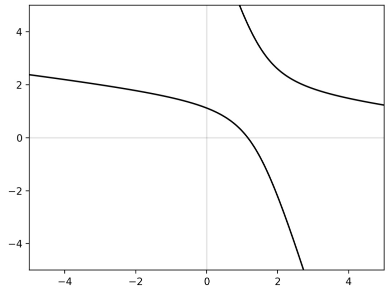
should fit well with the training data $\left\{\left(\mathbf{x}^{(i)}, y_i\right) : i \in [250]\right\}$, one can anticipate that the decision boundary $f(\mathbf{x}) = 0$ for part (b) closely resembles the boundary of the labeling rule (1) $\left(= \left\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\right\}\right)$ for part (b) within the region $[-2, 2] \times [-2, 2]$. In this context, we may guess that the decision boundary $f(\mathbf{x}) = 0$ for part (b) would look like an ellipse in $\mathbb{R}^2$ whose center is located nearby the origin $(0, 0)$.

(c) The only difference of the part (c) from the part (b) is the sampling scheme of the measurements: we sample $\left\{\mathbf{x}^{(i)} := \left(x_1^{(i)}, x_2^{(i)}\right) : i \in [250]\right\}$ uniformly at random from $[0, 2] \times [0, 2]$ in lieu of $[-2, 2] \times [-2, 2]$. So the corresponding overall code for part (c) can be summarized as below:

```
import numpy as np
import random
import matplotlib as mpl
import matplotlib.pyplot as plt

def axes():
    plt.axhline(0, alpha=.1)
    plt.axvline(0, alpha=.1)

def label(a, b):      # label a given point in \mathbb{R}^2 according to the rule (4)
    if a**2 + b**2 <= 1:
        return 1
    else:
        return -1

def generate_data_matrix(first_coordinate, second_coordinate):      # construct a proper data matrix (2-dimensional array) of shape length \times 6
    input_length = first_coordinate.shape[0]
    data_matrix = np.zeros((length, 6))
    for i in range(input_length):
        data_matrix[i, 0] = 1
```

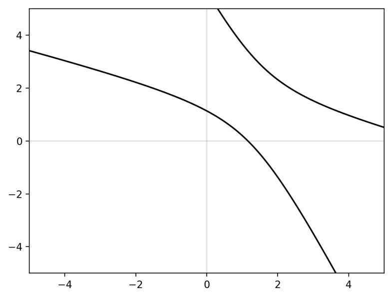
```
        data_matrix[i, 1] = first_coordinate[i]
        data_matrix[i, 2] = second_coordinate[i]
        data_matrix[i, 3] = first_coordinate[i]**2
        data_matrix[i, 4] = first_coordinate[i]*second_coordinate[i]
        data_matrix[i, 5] = second_coordinate[i]**2
    return data_matrix

def conic_section_discriminant(A, B, C, D, E, F):        # discriminate the conic section Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0
    discriminant = B**2 - 4*A*C
    if A == C and B == 0:
        return "Circle"
    elif discriminant < 0:
        return "Ellipse"
    elif discriminant == 0:
        return "Parabola"
    else:
        return "Hyperbola"

length = 250     # number of samples
first_coordinate_2 = np.dot(2, np.random.rand(length))     # the first coordinates of 250 samples chosen uniformly at random from [0, 2] \times [0, 2]
second_coordinate_2 = np.dot(2, np.random.rand(length))    # the second coordinates of 250 samples chosen uniformly at random from [0, 2] \times [0, 2]
y_2 = np.zeros(length)     # initialize vector of labels for part (c)
num_2 = 0
for i in range(length):
    y_2[i] = label(first_coordinate_2[i], second_coordinate_2[i])     # label the 250 random sample points according to the rule (4)
    if y_2[i] > 0:
        num_2 = num_2 + 1

A_2 = generate_data_matrix(first_coordinate_2, second_coordinate_2)
theta_2 = my_lstsq(A_2, y_2)     # compute the optimal order-2 polynomial that minimizes (3)
print(theta_2)
print(conic_section_discriminant(theta_2[3], theta_2[4], theta_2[5], theta_2[1], theta_2[2], theta_2[0]))     # discriminate the conic section formed as the zero set of the optimal order-2 polynomial that minimizes (3) (= the decision boundary)

x = np.linspace(-5, 5, 4000)
y = np.linspace(-5, 5, 4000)
x, y = np.meshgrid(x, y)
axes()
plt.contour(x, y, (theta_2[3]*x**2 + theta_2[4]*x*y + theta_2[5]*y**2 + theta_2[1]*x + theta_2[2]*y + theta_2[0]), [0], colors='k')
plt.show()
```

Moreover, the corresponding numerical simulation results are obtained as follows over 5 trials, which were also conducted in the demonstration of Figure 1–5:



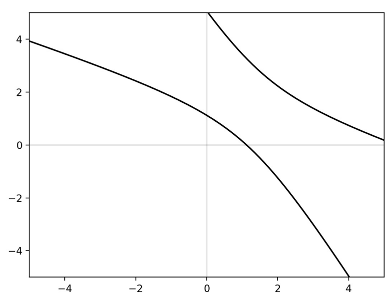

```
[ 0.22136734 -0.01107393 -0.00160513 -0.30279146  0.01953938 -0.31016869]
Ellipse
[ 2.39095029 -2.27012348 -2.41631035  0.17609717  1.16161241  0.25071671]
Hyperbola

Process finished with exit code 0
```

Figure 6: The decision boundary and outputs for part (c); Trial #1



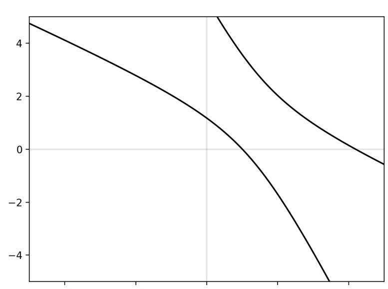

```
[ 0.10635102  0.0093856   0.03371123 -0.26687212  0.01929558 -0.27077483]
Ellipse
[ 2.46329425 -2.51432628 -2.59915563  0.33875255  1.11776776  0.38191455]
Hyperbola

Process finished with exit code 0
```

Figure 7: The decision boundary and outputs for part (c); Trial #2



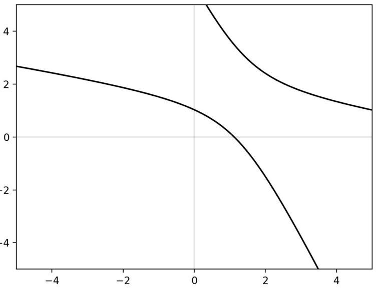

```
[ 0.08153645  0.02563064 -0.05074008 -0.26344132  0.01459541 -0.27096089]
Ellipse
[ 2.44617507 -2.63661919 -2.66345798  0.40771009  1.11977809  0.43030099]
Hyperbola

Process finished with exit code 0
```

Figure 8: The decision boundary and outputs for part (c); Trial #3




```
[ 0.03532633  0.02337575  0.0373171  -0.24028077 -0.01709515 -0.2657974 ]
Ellipse
[ 2.47039615 -3.04722482 -2.53690046  0.58586328  1.20773101  0.37125046]
Hyperbola

Process finished with exit code 0
```

Figure 9: The decision boundary and outputs for part (c); Trial #4



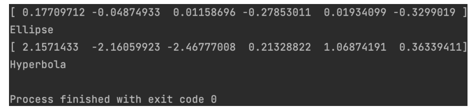


Figure 10: The decision boundary and outputs for part (c); Trial #5

As Figure 6–10 demonstrate, one may empirically confirm that the decision boundary $f(\mathbf{x}) = 0$ of part (c) forms a *hyperbola* in $\mathbb{R}^2$. At this point, as part (b), one can make a guess of the shape of the decision boundary $f(\mathbf{x}) = 0$ for part (c) with the aid of an analogous intuition: since the optimal polynomial (3) should fit well with the training examples $\left\{\left(\mathbf{x}^{(i)}, y_i\right) : i \in [250]\right\}$, we may expect that the decision boundary $f(\mathbf{x}) = 0$ for part (c) looks similar with the boundary of the labeling rule (1) $\left(= \left\{(x, y) \in \mathbb{R}^2 : x, y \geq 0 \ \& \ x^2 + y^2 = 1\right\}\right)$ for part (c) within the region $[0, 2] \times [0, 2]$. It is worth to notice that the decision boundary $f(\mathbf{x}) = 0$ for part (c) has no need for being akin to the remaining parts of the unit circle in $\mathbb{R}^2$. Indeed, we can find that the intersections of the hyperbolas in Figure 6–10 and the rectangular region $[0, 2] \times [0, 2]$ look very similar to the first quadrant part of the unit circle $\left(= \left\{(x, y) \in \mathbb{R}^2 : x, y \geq 0 \ \& \ x^2 + y^2 = 1\right\}\right)$. So one can argue that the empirical results in Figure 6–10 match fairly well with our intuition. However, in my opinion, it would be difficult to determine the exact shape of the conic section $f(\mathbf{x}) = 0$ for part (c), which is determined by its discriminant (or its eccentricity), solely based on such intuitions.

References

[1] Murray H Protter and Philip E Protter. *Calculus with analytic geometry*. Jones & Bartlett Learning, 1988.