Assignment 3

Sophia J Quinton

Grand Canyon University

DSC-540: Machine Learning

Dr. Aiman Darwiche

17 November 2021

Assignment 3

Part 1

Proof 1

Given:

$$P(X|y_q) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu_q)^T\Sigma^{-1}(x-\mu_q);\ q = 1,2$$

Prove:

$g_q(x) = ln(p(X|y_q))P(y_q)) = ln(p(p(X|y_q)) + ln(P(y_q))$ Equation 3.61 (Gopal, 2019)

It is shown that the natural logarithm is used as a discriminant.

Now plug in the given formula

$$g_q(x) = ln(\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu_q)^T\Sigma^{-1}(x-\mu_q)) + ln(P(y_q));\ q = 1,2$$

$$= -ln((2\pi)^{n/2}|\Sigma|^{1/2}) - (-\frac{1}{2}(x-\mu_q)^T\Sigma^{-1}(x-\mu_q)) + ln(P(y_q));\ q = 1,2$$

Simplify

$$= -ln((2\pi)^{n/2}|\Sigma|^{1/2}) + \mu_q^T\Sigma^{-1}x - \frac{1}{2}\mu_q^T\Sigma^{-1}\mu_q + ln(P(y_q));\ q = 1,2$$

Constant is irrelevant as equation will max out (PennStats, nd)

$$g_q(x) = \mu_q^T\Sigma^{-1}x - \frac{1}{2}\mu_q^T\Sigma^{-1}\mu_q + ln(P(y_q));\ q = 1,2$$

$$\therefore$$

Proof 2

Given:

$$g(x) = w^Tx + w_0 = 0$$

Prove:

Using the equation: $g(x) = g_1(x) - g_2(x)$ equation 3.62 (Gopal, 2019) and $g_q(x) = \mu_q^T \Sigma^{-1} x - \frac{1}{2} \mu_q^T \Sigma^{-1} \mu_q + ln(P(y_q));\ q = 1{,}2$

$$g(x) = \mu_1^T \Sigma^{-1} x - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + ln\big(P(y_1)\big) - \mu_2^T \Sigma^{-1} x - \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + ln\big(P(y_2)\big)$$

Simplify natural logs

$$= \mu_1^T \Sigma^{-1} x - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} x - \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + ln(\frac{P(y_1)}{P(y_2)})$$

Simplify by combining terms

$$(\mu_1^T - \mu_2^T)\, \Sigma^{-1} x - \frac{1}{2} (\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2) + ln(\frac{P(y_1)}{P(y_2)})$$

Plug into given equation

$$g(x) = w^T x + w_0 = (\mu_1^T - \mu_2^T)\, \Sigma^{-1} x - \frac{1}{2} (\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2) + ln(\frac{P(y_1)}{P(y_2)})$$

Part 2

For this example of gradient descent, the equation was first defined as a function and graphed in a 3D plot. Figure 1 defines this first step following the code from VanderPlas (nd). The next step was to create a gradient descent function following the instructions in RealPython (2021). It uses a starting point of [2,-2]. The first and second point were chosen. The learning rate of 0.08 was chosen because it would be more precise. Figure 2 shows the function and the graph.

```
In [77]: #(VanderPlas, nd)
         def f(x,y):
             return 2.0*x**2.0 + 2.0*x*y + 5.0*y**2.0
         x = np.linspace(-20, 20,10)
         y = np.linspace(-20,20, 10)
         X,Y = np.meshgrid(x,y)
         Z = f(X,Y)
         ax = plt.axes(projection='3d')
         ax.contour3D(X,Y,Z, 50,cmap='binary')
Out[77]: <matplotlib.contour.QuadContourSet at 0x2705fd244c0>
```

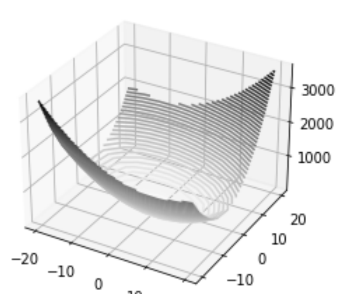


Figure 1: Plot of function

```
#(RealPython, 2021)
start_point = [2, -2, f(2,-2)]
def gradient_descent(gradient, start, learn_rate, n_itter=2):
    vector = start
    for _ in range(n_itter):
        diff = -learn_rate * gradient(vector)
        vector += diff
    return vector

first = gradient_descent(gradient=lambda v: np.array([2*v[0]**2 + 2*v[0]*v[1] + 5*v[1]**2]), start= start_point, learn_rate=0.08,
second = gradient_descent(gradient=lambda v: np.array([2*v[0]**2 + 2*v[0]*v[1] + 5*v[1]**2]), start= start_point, learn_rate=0.0
```

```
first
```

```
array([ 0.4, -3.6, 18.4])
```

```
second
```

```
array([-4.5792, -8.5792, 13.4208])
```

```
#graph (Kite, nd)
x_points = [start_point[0], first[0], second[0]]
y_points = [start_point[1], first[1], second[1]]
z_points = [start_point[2], first[2], second[2]]
ax = plt.axes(projection='3d')
ax.contour3D(X,Y,Z, 10,cmap='binary')

ax.scatter(x_points, y_points, z_points, color='green')
ax.plot(x_points, y_points, z_points, color='green')
```

```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x2705ff88970>]
```

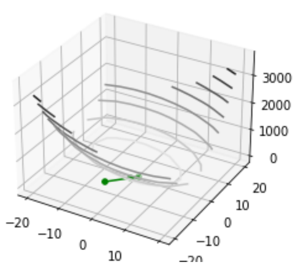


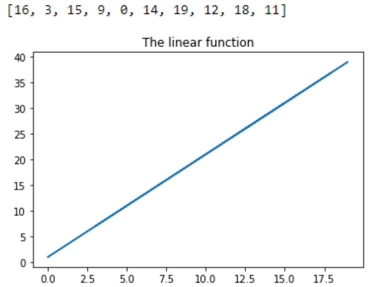
Figure 2: Contour plot with gradient points

Part 3

```
In [30]: import random
def f3(x,b, m):
    final_result = []
    for i in range(len(x)):
        result = x[i]*m + b
        final_result.append(result)
    return final_result

x2 = random.sample(range(0,20), 10) #(PYNative, 2021)
print(x2)
b = 1
m = 2
lin_funct = f3(x=x2, b=b, m=m)
plt.plot(x2, lin_funct)
plt.title("The linear function")
plt.show()
```

```
[16, 3, 15, 9, 0, 14, 19, 12, 18, 11]
```




```
In [29]: ##part 3 (nbshare notebooks, nd)
def f2(x, x0, k, L):
    return L/(1+np.exp(-k*(x-x0)))

x = np.arange(start=-4, stop=4, step=0.1)
x0 = 0
L=4
log_funct = f2(x=x, x0=x0, k=2, L=L)
plt.plot(x, log_funct)
plt.title("The Logistic Function")
plt.show()
#x0 is starting
#x is the array of random numbers
#L = max
#k is the step
```

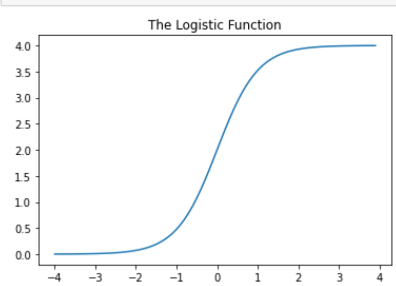


Figure 3: logistic and linear functions

The logistic regression function is represented by the s shaped function and the linear regression function is represented by the straight-line function. Logistic regression is used for binary problems. This means problems that have a yes or no, one or two, true or false etc., as their response variable. The linear regression is used to help solve problems that are focused on predicting continuous data. An example of this type of question would be predicting the weight of a kiwi bird. Another issue is that logistic regression cannot predict negative values. “If both linear regression and logistic regression make a prediction on the probability, linear model can even generate negative prediction, while logistic regression does not have such problem” (S., 2021).

Looking at the mathematical functions show that logistic regression is a nonlinear regression.

$$f(x) = \frac{L}{1+ e^{-k(x-x_0)}}$$

When dividing by a variable, the function takes on a nonlinear shape. It is not possible to treat a logistic discrimination in terms of an equivalent linear regression because the response variable is a binary value, and sometimes it is a categorical variable. One cannot make a “nominal response into a numeric response”. Linear functions are focused on continuous data. Looking at the graph of the logistic function, there is a distinct separation between the ‘top’ and ‘bottom’ data points. This represents a clear distinction of two groups. This cannot be applied to the linear graph.

References

Gopal, M. (2019). *Applied machine learning*. McGraw-Hill Education.

Kite. (n.d.). *Code faster with line-of-code completions, cloudless processing*. Kite. Retrieved November 16, 2021, from https://www.kite.com/python/answers/how-to-make-a-connected-scatter-plot-in-matplotlib-in-python.

nbshare notebooks. (n.d.). *Understanding Logistic Regression Using Python*. Understanding logistic regression using python. Retrieved November 18, 2021, from https://www.nbshare.io/notebook/415235001/Understanding-Logistic-Regression-Using-Python/.

PennStats. (n.d.). 9.2 - discriminant analysis. Retrieved November 18, 2021, from https://online.stat.psu.edu/stat508/book/export/html/645.

PYNative. (2021, November 13). *Python random randrange() and randint() to generate random* ... Retrieved November 18, 2021, from https://pynative.com/python-random-randrange/.

Real Python. (2021, January 19). *Stochastic gradient descent algorithm with python and NumPy*. Real Python. Retrieved November 16, 2021, from https://realpython.com/gradient-descent-algorithm-python/.

S, Y. (2021, September 6). *An introduction to logistic regression*. Medium. Retrieved November 18, 2021, from https://towardsdatascience.com/an-introduction-to-logistic-regression-8136ad65da2e.

VanderPlas, J. (n.d.). *Three-dimensional plotting in Matplotlib*. Three-Dimensional Plotting in Matplotlib | Python Data Science Handbook. Retrieved November 15, 2021, from https://jakevdp.github.io/PythonDataScienceHandbook/04.12-three-dimensional-plotting.html.