CS 3190 Homework 5

Kyle Kazemini

November 25, 2020

(1)

Euclidean distance for two points $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ is defined as $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. This distance will be denoted $\phi_S(x)$. This formulation is equivalent to the one given in the textbook, but is specific to points in $\mathbb{R}^2$.

(a)

$\phi_{s_1}(x_1) = 1$
$\phi_{s_2}(x_1) = \sqrt{13}$
$\phi_{s_3}(x_1) = \sqrt{13}$

Thus, the closest site to x_1 is s_1.

(b)

$\phi_{s_1}(x_2) = \sqrt{53}$
$\phi_{s_2}(x_2) = 5$
$\phi_{s_3}(x_2) = \sqrt{17}$

Thus, the closest site to x_2 is s_3.

(c)

$\phi_{s_1}(x_3) = 7$
$\phi_{s_2}(x_3) = \sqrt{17}$
$\phi_{s_3}(x_3) = 5$

Thus, the closest site to x_3 is s_2.

(d)

$\phi_{s_1}(x_4) = \sqrt{40}$
$\phi_{s_2}(x_4) = \sqrt{82}$
$\phi_{s_3}(x_4) = \sqrt{106}$

Thus, the closest site to x_4 is s_1.

(e)

$\phi_{s_1}(x_5) = \sqrt{5}$
$\phi_{s_2}(x_5) = \sqrt{17}$

$\phi_{s_3}(x_5) = \sqrt{41}$

Thus, the closest site to x_5 is s_1.

(f)

From the textbook, the probability density function of a d-dimensional Gaussian distribution is defined as

$$f_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{d/2}} \frac{1}{\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

where $|\Sigma|$ is the determinant of Σ. In our specific case, $x, \mu \in \mathbb{R}^2$ and $\Sigma \in \mathbb{R}^{2x2}$. This gives the following probability density functions.

$f_{\mu_1,\Sigma_1}(x) = \frac{1}{2\pi} \exp\left(-\frac{1}{2}(x-\mu)^T (x-\mu)\right)$

where $\Sigma_1 = I_2$ (the 2-dimensional identity matrix).

$f_{\mu_2,\Sigma_2}(x) = \frac{1}{4\pi} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$

where $\Sigma_2 = 2I_2$. (2 * the 2-dimensional identity matrix).

$f_{\mu_3,\Sigma_3}(x) = \frac{1}{18\pi} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$

where $\Sigma_3 = 3I_2$. (3 * the 2-dimensional identity matrix).

NOTE: In g-k I won't include the formulas with vectors plugged in because they're really hard to read. I'll include the code I used for multiplying the matrices and vectors together and then give an exact solution.

(g)

```
import numpy as np

x = np.array([0, 0])
y = np.array([0, 1])

answer = np.array(x - y).transpose() @ np.array(x - y)
answer

x = np.array([0, 0])
y = np.array([3, 2])
z = np.array(x - y)

m = np.matrix([[1/2, 0], [0, 1/2]])

answer = z.transpose() @ m @ z
answer
```

```
x = np.array([0, 0])
y = np.array([3, -2])
z = np.array(x - y)

m = np.matrix([[1/3, 0], [0, 1/3]])

answer = z.transpose() @ m @ z
answer
```

$$w_1(x_1) = \frac{\frac{1}{2\pi}\exp(-1/2)}{\frac{1}{2\pi}\exp(-1/2) + \frac{1}{4\pi}\exp(-9/4) + \frac{1}{18\pi}\exp(-13/6)}.$$

$$w_2(x_1) = \frac{\frac{1}{4\pi}\exp(-9/4)}{\frac{1}{2\pi}\exp(-1/2) + \frac{1}{4\pi}\exp(-9/4) + \frac{1}{18\pi}\exp(-13/6)}.$$

$$w_3(x_1) = \frac{\frac{1}{18\pi}\exp(-13/6)}{\frac{1}{2\pi}\exp(-1/2) + \frac{1}{4\pi}\exp(-9/4) + \frac{1}{18\pi}\exp(-13/6)}.$$

(h)

```
x = np.array([7, -1])
y = np.array([0, 1])

answer = np.array(x - y).transpose() @ np.array(x - y)
print(answer)

x = np.array([7, -1])
y = np.array([3, 2])
z = np.array(x - y)

m = np.matrix([[1/2, 0], [0, 1/2]])

answer = z.transpose() @ m @ z
answer

x = np.array([7, -1])
y = np.array([3, -2])
z = np.array(x - y)

m = np.matrix([[1/3, 0], [0, 1/3]])

answer = z.transpose() @ m @ z
answer
```

$$w_1(x_2) = \frac{\frac{1}{2\pi}\exp(-53/2)}{\frac{1}{2\pi}\exp(-53/2) + \frac{1}{4\pi}\exp(-25/4) + \frac{1}{18\pi}\exp(-14/5)}.$$

$$w_2(x_2) = \frac{\frac{1}{4\pi}\exp(-25/4)}{\frac{1}{2\pi}\exp(-53/2) + \frac{1}{4\pi}\exp(-25/4) + \frac{1}{18\pi}\exp(-14/5)}.$$

$$w_3(x_2) = \frac{\frac{1}{4\pi}\exp(-14/5)}{\frac{1}{2\pi}\exp(-53/2) + \frac{1}{4\pi}\exp(-25/4) + \frac{1}{4\pi}\exp(-14/5)}.$$

(i)

```
x = np.array([7, 1])
y = np.array([0, 1])

answer = np.array(x - y).transpose() @ np.array(x - y)
answer

x = np.array([7, 1])
y = np.array([3, 2])
z = np.array(x - y)

m = np.matrix([[1/2, 0], [0, 1/2]])

answer = z.transpose() @ m @ z
answer

x = np.array([7, 1])
y = np.array([3, -2])
z = np.array(x - y)

m = np.matrix([[1/3, 0], [0, 1/3]])

answer = z.transpose() @ m @ z
answer
```

$$w_1(x_3) = \frac{\frac{1}{2\pi}\exp(-49/2)}{\frac{1}{2\pi}\exp(-49/2) + \frac{1}{4\pi}\exp(-17/4) + \frac{1}{18\pi}\exp(-83/20)}.$$

$$w_2(x_3) = \frac{\frac{1}{4\pi}\exp(-17/4)}{\frac{1}{2\pi}\exp(-49/2) + \frac{1}{4\pi}\exp(-17/4) + \frac{1}{18\pi}\exp(-83/20)}.$$

$$w_3(x_3) = \frac{\frac{1}{18\pi}\exp(-83/20)}{\frac{1}{2\pi}\exp(-49/2) + \frac{1}{4\pi}\exp(-17/4) + \frac{1}{18\pi}\exp(-83/20)}.$$

(j)

```
x = np.array([-6, 3])
y = np.array([0, 1])

answer = np.array(x - y).transpose() @ np.array(x - y)
answer

x = np.array([-6, 3])
y = np.array([3, 2])
z = np.array(x - y)

m = np.matrix([[1/2, 0], [0, 1/2]])

answer = z.transpose() @ m @ z
answer

x = np.array([-6, 3])
y = np.array([3, -2])
```

```
z = np.array(x - y)

m = np.matrix([[1/3, 0], [0, 1/3]])

answer = z.transpose() @ m @ z
answer
```

$$w_1(x_4) = \frac{\frac{1}{2\pi}\exp(-20)}{\frac{1}{2\pi}\exp(-20) + \frac{1}{4\pi}\exp(-41/2) + \frac{1}{18\pi}\exp(-353/20)}.$$

$$w_2(x_4) = \frac{\frac{1}{4\pi}\exp(-41/2)}{\frac{1}{2\pi}\exp(-20) + \frac{1}{4\pi}\exp(-41/2) + \frac{1}{18\pi}\exp(-353/20)}.$$

$$w_3(x_4) = \frac{\frac{1}{18\pi}\exp(-353/20)}{\frac{1}{2\pi}\exp(-20) + \frac{1}{4\pi}\exp(-41/2) + \frac{1}{18\pi}\exp(-353/20)}.$$

(k)

```
x = np.array([-1, 3])
y = np.array([0, 1])

answer = np.array(x - y).transpose() @ np.array(x - y)
answer

x = np.array([-1, 3])
y = np.array([3, 2])
z = np.array(x - y)

m = np.matrix([[1/2, 0], [0, 1/2]])

answer = z.transpose() @ m @ z
answer

x = np.array([-1, 3])
y = np.array([3, -2])
z = np.array(x - y)

m = np.matrix([[1/3, 0], [0, 1/3]])

answer = z.transpose() @ m @ z
answer
```

$$w_1(x_5) = \frac{\frac{1}{2\pi}\exp(-5/2)}{\frac{1}{2\pi}\exp(-5/2) + \frac{1}{4\pi}\exp(-17/4) + \frac{1}{18\pi}\exp(-34/5)}.$$

$$w_2(x_5) = \frac{\frac{1}{4\pi}\exp(-17/4)}{\frac{1}{2\pi}\exp(-5/2) + \frac{1}{4\pi}\exp(-17/4) + \frac{1}{18\pi}\exp(-34/5)}.$$

$$w_3(x_5) = \frac{\frac{1}{18\pi}\exp(-34/5)}{\frac{1}{2\pi}\exp(-5/2) + \frac{1}{4\pi}\exp(-17/4) + \frac{1}{18\pi}\exp(-34/5)}.$$

(2)

Let $X = \{x_1 = (0,0),\ x_2 = (1,1),\ x_3 = (2,2),\ x_4 = (3,3)\}$. Let $S = \{s_1 = (0,1),\ s_2 = (1,0)\}$. Lloyd's algorithm will converge for S after some number of steps $R < \infty$ since there are finite ways any arbitrary set of points can be assigned to different clusters (thus, the algorithm terminates after a finite number of steps). However, X gets stuck in a local minimum for the set S at s_1. This is fine because we know that Lloyd's algorithm is not guaranteed to find the optimal set of sites, but it does increase $cost(X, S)$ because the algorithm will not move from that point.

Now take $S' = \{s_1 = (0,1),\ s_2 = (1,0)\}$. Lloyd's algorithm still converges for S' by the same logic. However, $cost(X, S) > cost(X, S')$ because $\forall\ s_i \in S$ and $\forall\ s'_i \in S'$, the following holds.

$$\|\phi_S(x) - x\|^2 > \|\phi_{S'}(x) - x\|^2$$

where $x \in X$. Thus, we've shown that Lloyd's algorithm will converge for both S and S' but $cost(X, S') < cost(X, S)$.

(3)

(a)

In plain English, the function

$$\Lambda(g_{w,b}, (x_i, y_i))$$

is a cost function for a family of linear classifiers. The cost function returns 1 for points that are misclassified and meet the condition $|g_{w,b}(x_i)| > 1$. The cost function returns 1/2 for points that meet the condition $0 \le |g_{w,b}(x_i)| \le 1$, and finally, the cost function returns 0 for points that are classified correctly and meet the condition $|g_{w,b}(x_i)| > 1$.

As stated in the textbook, $z_i = y_i g_\alpha(x_i)$ is a clever expression for handling when the function $g_\alpha(x_i)$ correctly predicts the positive or negative example in the same way. That is, $z_i > 0$ when $y_i = 1$ and $g_\alpha(x_i) > 0$ or when $y_i = -1$ and $g_\alpha(x_i) < 0$. Of course for this problem, $g_\alpha(x_i) = g_{w,b}(x_i)$. Thus, the function $\Lambda(g_{w,b}, (x_i, y_i))$ as a function of $z_i = y_i g_\alpha(x_i)$ can be interpreted as follows.

$$\Lambda(g_{w,b}, (x_i, y_i)) = \begin{cases} 1 & z_i < 0 \\ 1/2 & 0 \le z_i \le 1 \\ 0 & z_i > 1 \end{cases}$$

This is essentially a clever way to think about $\Lambda(g_{w,b}, (x_i, y_i))$.

(b)

Define $\ell_\Lambda(z)$ to be

$$\ell_\Lambda(z) = \begin{cases} 0 & z_i \ge 1 \\ (1-z)^2/2 & 0 < z_i < 1 \\ 1/2 - z & z_i \le 0 \end{cases}$$

It's states in the textbook that this (the smoothed hinge loss function) is convex. There are two points of speculation for defining the derivative of $\ell_\Lambda(z)$: 0 and 1.
$\ell'_\Lambda(0) = -1$
$\ell'_\Lambda(1) = 0$

Thus, the derivative of $\ell_\Lambda(z)$ is well defined on $\mathbb{R}$.

It's easy to see that $\ell_\Lambda(z) \geq \Lambda(z)\ \forall\ z$ because of our construction of z in part (a). Thus, we've shown that $\ell_\Lambda(z)$ is a proxy for $\Lambda(z)$ that is convex, has a derivative defined for all z, and satisfies $\ell_\Lambda(z) \geq \Lambda(z)\ \forall\ z$.

(4)

(a)

Let S be the set of points given by $x1$, $x2$, $y1$, and $y2$ in the following code where the terms of $x1$ and $x2$ form a point in $\mathbb{R}^2$ and the terms of $y1$ and $y2$ form a point in $\mathbb{R}^2$. The set S is not linearly separable because it would require two lines to separate the points in S. That does not agree with the definition of linear separability, so S is not linearly separable.

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

x1 = [1, 2, 2, 10, 9, 8]
x2 = [1, 1, 1.5, 8, 10, 10]
y1 = [2, 3, 4, 4, 5, 6]
y2 = [5, 4, 3, 4, 3, 2]

plt.scatter(x1, x2, c = "red")
plt.scatter(y1, y2, c = "blue")
plt.show()
```

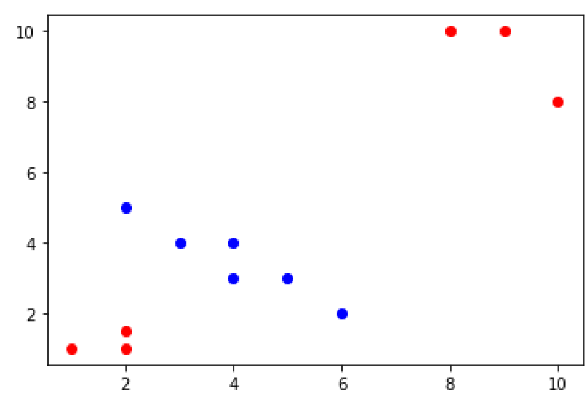

Figure 1: Plot

(b)

The perceptron algorithm will only converge for a set that is linearly separable through the origin. The data set S is not linearly separable through the origin, so the perceptron algorithm won't converge for S. Since we know from the problem that there's no upper bound on the number of steps T for the algorithm, it will never terminate.

(c)

Linear classification via gradient descent would provide an acceptable classifier for S. As mentioned in the textbook, we can define a cost function that uses the identity function in order to represent the number of misclassified points.

$$\Delta(g_\alpha, (X, y)) = \sum_{i=1}^{n} (1 - \mathbb{1}(sign(y_i) = sign(g(x_i)))$$

Then we can use a loss function as a proxy in order to run gradient descent feasibly.

$$f(\alpha) = \sum_{i=1}^{n} \ell(g_\alpha, (x_i, y_i))$$

More than one of the loss functions given in the textbook would work here, but let's say we choose hinge loss.

$$\ell(z) = \max(0, 1 - z)$$

Linear classification via gradient descent does not require a set to be linearly separable through the origin, so this method would provide an acceptable classifier for S.