CS189

1. (a) By myself. consulting piazza.

(b) I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Yao Cao

HW4 Yao Cai

2. (a) $P(X=\vec{x}) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\det(\Sigma)|}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{u})^T \Sigma^{-1} (\vec{x}-\vec{u})\right]$

$$ll = \log L = \sum_{i=1}^{n} \log P(X=\vec{x}_i)$$

$$= \sum_{i=1}^{n} \left(\log \frac{1}{(\sqrt{2\pi})^d \sqrt{|\det(\Sigma)|}} - \frac{1}{2}(\vec{x}_i-\vec{u})^T \Sigma^{-1} (\vec{x}_i-\vec{u}) \right)$$

$$= n \log \frac{1}{(\sqrt{2\pi})^d \sqrt{|\det(\Sigma)|}} - \frac{1}{2} \sum_{i=1}^{n} (\vec{x}_i-\vec{u})^T \Sigma^{-1} (\vec{x}_i-\vec{u})$$

(b) $\frac{\partial ll}{\partial \vec{u}} = 0 + \frac{1}{2} \sum_{i=1}^{n} 2\Sigma^{-1}(\vec{x}_i-\vec{u})$

$$= \Sigma^{-1} \sum_{i=1}^{n} (\vec{x}_i-\vec{u}) = 0$$

$$\Rightarrow n\vec{u} = \sum_{i=1}^{n} \vec{x}_i$$

$$\hat{u} = \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i$$

$$\frac{\partial ll}{\partial \Sigma} = \frac{\partial}{\partial \Sigma} \left[n\left(-d \log\sqrt{2\pi} - \frac{1}{2}\log|\det(\Sigma)|\right) - \frac{1}{2}\sum_{i=1}^{n} (\vec{x}_i-\vec{u})^T \Sigma^{-1} (\vec{x}_i-\vec{u}) \right]$$

$$= -\frac{1}{2} n (\Sigma^{-1}) + \frac{1}{2} \sum_{i=1}^{n} \Sigma^{-1} (\vec{x}_i-\vec{u})(\vec{x}_i-\vec{u})^T \Sigma^{-1} = 0$$

$$\Rightarrow n\Sigma^{-1} = \Sigma^{-1} \left(\sum_{i=1}^{n} (\vec{x}_i-\vec{u})(\vec{x}_i-\vec{u})^T \right) \Sigma^{-1}$$

$$\Rightarrow n = \Sigma^{-1} \left(\sum_{i=1}^{n} (\vec{x}_i-\vec{u})(\vec{x}_i-\vec{u})^T \right)$$

$$\Rightarrow \hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (\vec{x}_i-\hat{u})(\vec{x}_i-\hat{u})^T$$

2c

when simga = [[20, 0], [0, 10]], the MLE mean is [14.62972319 5.13279893], and MLE covariance is [[13.93459582 2.83012681]
 [2.83012681 9.93336424]]
the singular values for sigma matrix is [20. 10.]

when simga = [[20, 14], [14, 10]], the MLE mean is [14.98469043 5.04061468],
and MLE covariance is [[20.46621341 14.35034222]
 [14.35034222 10.23210116]]
the singular values for sigma matrix is [29.86606875 0.13393125]

when simga = [[20, -14], [-14, 10]], the MLE mean is [14.35926801 5.38991362], and MLE covariance is [[16.75876404 -11.60036104]
 [-11.60036104 8.29960636]]
the singular values for sigma matrix is [29.86606875 0.13393125]

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

def mean(samples):
	return np.average(samples, axis = 0)

def covariance(samples, mu_hat):
	n = samples.shape[0]
	d = samples.shape[1]
	mu_hat = mu_hat.reshape((d, 1))
	return (samples.T-mu_hat) @ (samples.T-mu_hat).T/n

if __name__ == '__main__':
	mu = [15, 5]
	sigmas = [[[20, 0], [0, 10]], [[20, 14], [14, 10]], [[20, -14], [-14, 10]]]
	for i, sigma in enumerate(sigmas):
		samples = np.random.multivariate_normal(mu, sigma, size=100)
		mu_hat = mean(samples)
		sigma_hat = covariance(samples, mu_hat)
		print('when simga = {}, the MLE mean is {}, and MLE covariance is
{}'.format(sigma, mu_hat, sigma_hat))
		print('the singular values for sigma matrix is
{}'.format(np.linalg.svd(sigma, compute_uv = False)))
		fig = plt.figure()
		plt.scatter(samples[:, 0], samples[:, 1])
		plt.savefig('sigma{}.jpg'.format(i))
```

3. (a) $Y|x,w \sim N(x^T w, 1)$

(b) $P(w|Y_1 \cdots Y_n, x_1 \cdots x_n)$

$$\propto P(Y_1 \cdots Y_n | w, x_1 \cdots x_n) P(w|x_1 \cdots x_n)$$

$$\propto \prod_{i=1}^{n} P(Y_i|w, x_i) \cdot P(w)$$

$$\propto \prod_{i=1}^{n} \exp\left(-\frac{(Y_i - x_i^T w)^2}{2}\right) \cdot \exp\left(-\frac{1}{2} w^T \Sigma^{-1} w\right)$$

$$= \exp\left(-\frac{1}{2}\sum_{i=1}^{n}(Y_i - x_i^T w)^2 - \frac{1}{2} w^T \Sigma^{-1} w\right)$$

$$= \exp\left(-\frac{1}{2}\|\vec{Y} - Xw\|_2^2 - \frac{1}{2} w^T \Sigma^{-1} w\right)$$

$$\boxed{\begin{array}{l} \vec{Y} = (Y_1 \cdots Y_n)^T \in \mathbb{R}^{n\times 1} \\ X = \begin{pmatrix} - x_1^T - \\ \vdots \\ - x_n^T - \end{pmatrix} \in \mathbb{R}^{n\times d} \end{array}}$$

$$= \exp\left(-\frac{1}{2}\left(\vec{Y}^T\vec{Y} + w^T(X^TX + \Sigma^{-1})w) + \vec{Y}^T X w\right)\right)$$

$$= \exp\left(-\frac{1}{2}\left(w - (X^TX+\Sigma^{-1})^{-1}X^T\vec{Y}\right)^T (X^TX+\Sigma^{-1})\left(w - (X^TX+\Sigma^{-1})^{-1}X^T\vec{Y}\right)\right) + \text{const.}$$

$$w|Y_1 \cdots Y_n, x_1 \cdots x_n \sim N\left((X^TX+\Sigma^{-1})^{-1}X^T\vec{Y}, (X^TX+\Sigma^{-1})^{-1}\right)$$

$$\hat{w}_{MAP} = (X^TX+\Sigma^{-1})^{-1}X^T\vec{Y}$$

We can compare this result with HW3-2h where we derived multi-dimensional ridge regression, the only difference is that instead of $\sigma^2 I$ for ridge, we use Σ for Tikhonov. In ridge, all components of w are iid, but in Tikhonov, since covariance Σ is not generally diagonal, components are not iid and have dependence on each other, which is more realistic since features may have dependence on each other.

(c) $V \sim N(0, I)$

$$Z = \Sigma_z^{1/2} V + u_z \sim N(u_z, \Sigma_z)$$

$$Y = Xw + Z = Xw + \Sigma_z^{1/2} V + u_z$$

$$\Rightarrow Y - u_z = Xw + \Sigma_z^{1/2} V$$

$$\Sigma_z^{-1/2}(Y - u_z) = \Sigma_z^{-1/2} Xw + V \quad ①$$

Let $\tilde{Y} = \Sigma_z^{-1/2}(Y - u_z)$

Let $\tilde{X} = \Sigma_z^{-1/2} X$

① becomes $\tilde{Y} = \tilde{X} w + V$, then we can use the results from (b)

$$w \mid Y_1 \cdots Y_n, X_1 \cdots X_n \sim N\left((\tilde{X}^T\tilde{X} + \Sigma^{-1})^{-1}\tilde{X}^T\tilde{Y}, (\tilde{X}^T\tilde{X} + \Sigma^{-1})^{-1}\right)$$

$$\sim N\left((X^T\Sigma_z^{-1}X + \Sigma^{-1})^{-1} X^T\Sigma_z^{-1}(Y - u_z), (X^T\Sigma_z^{-1}X + \Sigma^{-1})^{-1}\right)$$

3d

As the number of date point increases, the posterior mean becomes closer to true w, and posterior variance becomes smaller.

```
import matplotlib
matplotlib.use('Qt5Agg')
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
np.random.seed(0)
from math import sqrt
```

```
def generate_data(n):
    """
    This function generates data of size n.
    """
    X = np.random.multivariate_normal([0, 0], [[5, 0], [0, 5]], size = n)
    z = np.random.normal(0, 1, size = n).reshape((n, 1))
    y = X @ np.array([[1], [1]]) + z
    print(X.shape)
    print(z.shape)
    print(y.shape)

    return (X,y)

def tikhonov_regression(X,y,Sigma):
    """
    This function computes w based on the formula of tikhonov_regression.
    """
    mux, muy, dummy, dummy, dummy = compute_mean_var(X,y,Sigma)
    return [mux, muy]

def compute_mean_var(X,y,Sigma):
    """
    This function computes the mean and variance of the posterior
    """
    pos_var = inv(X.T @ X + inv(Sigma))
    pos_mu = pos_var @ X.T @ y
    mux = pos_mu[0, 0]
    muy = pos_mu[1, 0]
    sigmax = sqrt(pos_var[0, 0])
    sigmay = sqrt(pos_var[1, 1])
    sigmaxy = pos_var[0, 1]
    return mux,muy,sigmax,sigmay,sigmaxy

Sigmas = [np.array([[1,0],[0,1]]), np.array([[1,0.25],[0.25,1]]),
          np.array([[1,0.9],[0.9,1]]), np.array([[1,-0.25],[-0.25,1]]),
          np.array([[1,-0.9],[-0.9,1]]), np.array([[0.1,0],[0,0.1]])]
names = [str(i) for i in range(1,6+1)]


fig, ax = plt.subplots(3, 6, figsize = (24, 12), sharex = True, sharey = True)

for j, num_data in enumerate([5,50,500]):
    X,Y = generate_data(num_data)
    for i,Sigma in enumerate(Sigmas):
```

```
    mux,muy,sigmax,sigmay,sigmaxy = compute_mean_var(X,Y,Sigma)

    x = np.arange(0.5, 1.5, 0.01)
    y = np.arange(0.5, 1.5, 0.01)
    X_grid, Y_grid = np.meshgrid(x, y)

    Z = matplotlib.mlab.bivariate_normal(X_grid,Y_grid, sigmax, sigmay, mux, muy, sigmaxy)



    # plot
    #plt.figure(figsize=(10,10))
    #CS = plt.contour(X_grid, Y_grid, Z,
    #            levels = np.concatenate([np.arange(0,0.05,0.01),np.arange(0.05,1,0.05)]))
    #plt.clabel(CS, inline=1, fontsize=10)
    #plt.xlabel('X')
    #plt.ylabel('Y')
    #plt.title('Sigma'+ names[i] + ' with num_data = {}'.format(num_data))
    #plt.savefig('Sigma'+ names[i] + '_num_data_{}.png'.format(num_data))

    CS = ax[j, i].contour(X_grid, Y_grid, Z,
            levels = np.concatenate([np.arange(0,0.05,0.01),np.arange(0.05,1,0.05)]))
    ax[j, i].clabel(CS, inline=1, fontsize=6)
    ax[j, i].set_xlabel('X')
    ax[j, i].set_ylabel('Y')
    #ax[j, i].axis('equal')
    ax[j, i].set_title('Sigma'+ names[i] + ' with num_data = {}'.format(num_data))
plt.tight_layout()
plt.savefig('3d.jpg')
```

3e

1\ As the number of training data increases, we have smaller MSE.

2\ When number of training data is small, MSE depends on which prior we choose. Different prior ends up with different MSE.

3\ As the number of training data increases, MSE becomes less dependent on the prior we choose. This is because as the number of training data increases, the effect of prior on posterior distribution decreases, and information from the training data becomes dominant.

4\Here we don't have control on prior mean which is [0, 0], we only have control on Sigma, and the true is at [1, 1]. A good prior distribution should have a large density at point [1, 1]. Since the Guassian prior center is at [0, 0], we need a Sigma which makes the prior distribution centered at [0, 0] but points to [1, 1]. And such a sigma is strongly positively correlated(Sigma3).

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv
np.random.seed(0)
w = [1.0,1.0]
n_test = 500
n_trains = np.arange(5,205,5)
n_trails = 100

Sigmas = [np.array([[1,0],[0,1]]), np.array([[1,0.25],[0.25,1]]),
        np.array([[1,0.9],[0.9,1]]), np.array([[1,-0.25],[-0.25,1]]),
        np.array([[1,-0.9],[-0.9,1]]), np.array([[0.1,0],[0,0.1]])]
names = ['Sigma{}'.format(i+1) for i in range(6)]

def generate_data(n):
    """
    This function generates data of size n.
    """
    X = np.random.multivariate_normal([0, 0], [[5, 0], [0, 5]], size = n)
    z = np.random.normal(0, 1, size = n).reshape((n, 1))
    y = X @ np.array([[1], [1]]) + z
    return (X,y)

def compute_mean_var(X,y,Sigma):
    """
    This function computes the mean and variance of the posterior
    """
    pos_var = inv(X.T @ X + inv(Sigma))
    pos_mu = pos_var @ X.T @ y
    mux = pos_mu[0, 0]
    muy = pos_mu[1, 0]
    sigmax = sqrt(pos_var[0, 0])
    sigmay = sqrt(pos_var[1, 1])
    sigmaxy = pos_var[0, 1]
    return mux,muy,sigmax,sigmay,sigmaxy

def tikhonov_regression(X,y,Sigma):
    """
    This function computes w based on the formula of tikhonov_regression.
    """
```

```
    mux, muy, dummy, dummy, dummy = compute_mean_var(X,y,Sigma)
    return np.array([[mux], [muy]])

def compute_mse(X,Y, w):
    """
    This function computes MSE given data and estimated w.
    """
    n = X.shape[0]
    Y_hat = X @ w
    mse = (Y-Y_hat).T @ (Y-Y_hat)/n
    return mse

def compute_theoretical_mse(w):
    """
    This function computes theoretical MSE given estimated w.
    """
    theoretical_mse = 5*(w[0, 0]-1)**2 + 5*(w[1, 0]-1)**2 + 1
    return theoretical_mse

# Generate Test Data.
X_test, y_test = generate_data(n_test)

mses = np.zeros((len(Sigmas), len(n_trains), n_trails))

theoretical_mses = np.zeros((len(Sigmas), len(n_trains), n_trails))

for seed in range(n_trails):
    np.random.seed(seed)
    for i,Sigma in enumerate(Sigmas):
        for j,n_train in enumerate(n_trains):
            #TODO implement the mses and theoretical_mses
            X_train, y_train = generate_data(n_train)
            w_hat = tikhonov_regression(X_train, y_train, Sigma)
            mses[i, j, seed] = compute_mse(X_test, y_test, w_hat)
            theoretical_mses[i, j, seed] = compute_theoretical_mse(w_hat)

# Plot
plt.figure()
for i,_ in enumerate(Sigmas):
    plt.plot(n_trains, np.mean(mses[i],axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('MSE on Test Data')
plt.legend()
plt.savefig('MSE.png')
```

```
plt.figure()
for i,_ in enumerate(Sigmas):
    plt.plot(n_trains, np.mean(theoretical_mses[i],axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('Theoretical MSE on Test Data')
plt.legend()
plt.savefig('theoretical_MSE.png')


plt.figure()
for i,_ in enumerate(Sigmas):
    plt.loglog(n_trains, np.mean(theoretical_mses[i]-1,axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('Theoretical MSE on Test Data')
plt.legend()
plt.savefig('log_theoretical_MSE.png')
```

4. (a) $\min_{w,r} \max_{\alpha} \left[\frac{1}{2}\|\vec{r}\|_2^2 + \frac{\lambda}{2}\|\vec{w}\|_2^2 + \alpha^T(\vec{r} - X\vec{w} + \vec{y}) \right]$

① for inner maximization, if $\vec{r} \neq X\vec{w} - \vec{y}$ we can always choose a $\vec{\alpha}$ to make $\alpha^T(\vec{r} - X\vec{w} + \vec{y})$ blow up, so that the case $\vec{r} \neq X\vec{w} - \vec{y}$ is ignored by minimization.

② if $\vec{r} = X\vec{w} - \vec{y}$, $\alpha^T(\vec{r} - X\vec{w} + \vec{y}) = 0$
then the problem becomes $\min_{w,r} \left[\frac{1}{2}\|\vec{r}\|_2^2 + \frac{\lambda}{2}\|\vec{w}\|_2^2 \right]$

these make the minimization-maximization problem equivalent to the original problem.

(b) solve $\min_{w,r} L(w, r, \alpha)$

$$\frac{\partial}{\partial r} L(w, r, \alpha) = \vec{r} + \vec{\alpha} = 0 \quad \Rightarrow \vec{r}^* = -\vec{\alpha}$$

$$\frac{\partial}{\partial w} L(w, r, \alpha) = \lambda \vec{w} - X^T \vec{\alpha} = 0 \quad \Rightarrow \vec{w}^* = \frac{1}{\lambda} X^T \vec{\alpha}$$

$$L(w^*, r^*, \alpha) = \frac{1}{2}\|\vec{\alpha}\|_2^2 + \frac{\lambda}{2} \cdot \frac{1}{\lambda^2} \vec{\alpha}^T X X^T \vec{\alpha} + \vec{\alpha}^T \left(-\vec{\alpha} - X \cdot \frac{1}{\lambda} X^T \vec{\alpha} + y\right)$$

$$= -\frac{1}{2}\|\vec{\alpha}\|_2^2 - \frac{1}{2\lambda} \vec{\alpha}^T X X^T \vec{\alpha} + \vec{\alpha}^T y = -\frac{1}{2\lambda} \vec{\alpha}^T (\lambda I + X X^T) \vec{\alpha} + \vec{\alpha}^T y$$

$$\underset{\alpha}{\operatorname{argmax}} L(w^*, r^*, \alpha) = \underset{\alpha}{\operatorname{argmax}} \left(-\frac{1}{2\lambda} \vec{\alpha}^T (\lambda I + X X^T) \vec{\alpha} + \vec{\alpha}^T y \right)$$

$$= \underset{\alpha}{\operatorname{argmin}} \left(\frac{1}{2} \vec{\alpha}^T (\lambda I + X X^T) \vec{\alpha} - \lambda \vec{\alpha}^T y \right)$$

$$= \underset{\alpha}{\operatorname{argmin}}\, g(\vec{\alpha})$$

(c) $$\frac{\partial g(\vec{\alpha})}{\partial \vec{\alpha}} = (\lambda I + X X^T) \vec{\alpha} - \lambda y = 0$$

$$\vec{\alpha}^* = \lambda (\lambda I + X X^T)^{-1} y$$

$$\Rightarrow \vec{w}^* = \frac{1}{\lambda} X^T \vec{\alpha}^* = \frac{1}{\lambda} X^T \lambda (\lambda I + X X^T)^{-1} y$$

$$= X^T (\lambda I + X X^T)^{-1} y$$

(d) Tikhonov regularization (kernel form)

$$w^* = \Sigma_w X'^T (X' \Sigma_w X'^T + \lambda I)^{-1} \vec{y}$$

$$\hat{y} = x'^T w^* = x'^T \Sigma_w X'^T (X' \Sigma_w X'^T + \lambda I)^{-1} \vec{y} \quad ①$$

$X' \in \mathbb{R}^{n \times d}$ $\quad d = \binom{l+2}{2}$

is polynomial feature matrix

$\Sigma_w \in \mathbb{R}^{d \times d}$

$w^* \in \mathbb{R}^{d \times 1}$

x' the new polynomial feature vector

kernel ridge regression:

$$w^* = X^T (K + \lambda I)^{-1} \vec{y}$$

$$\hat{y} = k(x, X^T)(K + \lambda I)^{-1} \vec{y} \quad ②$$

Compare ① & ②, we need to equate K and $X' \Sigma_w X'^T$

we can just look at one element of K

$$k(x, z) = (1 + x^T z)^2 \qquad x, z \in \mathbb{R}^{l \times 1}$$

$$= (x^T z)^2 + 2x^T z + 1$$

$$= (x_1 z_1 + x_2 z_2 + \cdots x_l z_l)^2 + 2(x_1 z_1 + \cdots + x_l z_l) + 1$$

$$= x_1^2 z_1^2 + x_2^2 z_2^2 + \cdots + x_l^2 z_l^2 + 2 \sum_{i \neq j} x_i z_i x_j z_j + 2 \sum_{i=1}^{l} x_i z_i + 1$$

$$= [x_1^2, x_2^2, \cdots x_l^2, x_1 x_2, x_1 x_3 \cdots x_{l-1} x_l, x_1, \cdots x_l, 1].$$

$l \times l$

$\frac{l(l-1)}{2} \times \frac{l(l-1)}{2}$

$l \times l$

$$\begin{bmatrix} 1 & & & & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & & & \\ & & \ddots & & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & & \\ & & & & 2 & & & & & & & & & & & \\ & & & & & \ddots & & & & & & & & & & \\ & & & & & & 2 & & & & & & & & & \\ & & & & & & & 2 & & & & & & & & \\ & & & & & & & & \ddots & & & & & & & \\ & & & & & & & & & 2 & & & & & & \\ & & & & & & & & & & 1 & & & & & \\ & & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & & \ddots & & & \\ & & & & & & & & & & & & & 1 & & \\ & & & & & & & & & & & & & & 1 & \\ & & & & & & & & & & & & & & & 1 \end{bmatrix} \begin{bmatrix} z_1^2 \\ \vdots \\ z_l^2 \\ z_1 z_2 \\ \vdots \\ z_{l-1} z_l \\ z_1 \\ \vdots \\ z_l \\ 1 \end{bmatrix}$$

if we let $\Sigma_w =$

$$\begin{bmatrix} 1 & & & & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & & & \\ & & \ddots & & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & & \\ & & & & 2 & & & & & & & & & & & \\ & & & & & \ddots & & & & & & & & & & \\ & & & & & & 2 & & & & & & & & & \\ & & & & & & & 2 & & & & & & & & \\ & & & & & & & & \ddots & & & & & & & \\ & & & & & & & & & 2 & & & & & & \\ & & & & & & & & & & 1 & & & & & \\ & & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & & \ddots & & & \\ & & & & & & & & & & & & & 1 & & \\ & & & & & & & & & & & & & & 1 & \\ & & & & & & & & & & & & & & & 1 \end{bmatrix}$$

then $K = X' \Sigma_w X'^T$

and the two approaches are equivalent

(e) if we do Tikhonov regression in non-kernelized form.

$$w^* = (X^T X + \Sigma_w^{-1})^{-1} X^T \vec{y}$$

$X \in \mathbb{R}^{n \times d}$ $\quad d = O(l^p)$

Computing $X^T X$ takes $O(nd^2)$

inversion takes $O(d^3)$

computing $X^T \vec{y}$ takes $O(nd)$

computing $\mathbb{R}^{d \times d} \times \mathbb{R}^{d \times 1}$ takes $O(d^2)$

the first two dominate and for finding w^*

it takes $\underline{\underline{O(d^3 + nd^2)}}$

$\hat{y} = x^T w^*$ takes $\underline{\underline{O(d)}}$ time

If we do kernel trick.

$\alpha \propto (K + \lambda I)^{-1} y$

compute $K = (1 + x_i^T x_j)^p$ takes $O((l + \log p) n^2)$

inversion takes $O(n^3)$

computing $\mathbb{R}^{n \times n} \times \mathbb{R}^{n \times 1}$ takes $O(n^2)$

finding α takes $\underline{\underline{O(n^3 + n^2(l + \log p))}}$

predication : $\hat{y} = k(x, X^T)\, \alpha$

compute $k(x, X^T)$ takes $O((l + \log p) n)$

compute $\mathbb{R}^{1 \times n} \times \mathbb{R}^{n \times 1}$ takes $O(n)$

so predication takes $\underline{\underline{O(l + \log p) n)}}$ time

5a

5b

poly_loss_circle
training loss
valid loss

poly_loss_heart
training loss
valid loss

poly_loss_asymmetric
training loss
valid loss

p=2
p=4
p=6
p=8
p=10
p=12

p=2
p=4
p=6
p=8
p=10
p=12

p=2
p=4
p=6
p=8
p=10
p=12

5c

We only want high-order polynomial when we have a lot of training data.

5d

poly_loss_vs_n_asymmetric
lambda = 0.0001, p = 5
lambda = 0.001, p = 5
lambda = 0.01, p = 5
lambda = 0.0001, p = 6
lambda = 0.001, p = 6
lambda = 0.01, p = 6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
10^2
10^3
10^4

5e

validation error is the smallest at sigma = 0.1. Validatin error first decreases and the increases as sigma increases.

5f

When p = 16, # of features = 153, which is much smaller than the number of points, so the non-kernelized one should be more efficient.

It is not possible to implement rbf as non-kernelized ridge regression, since if we do Taylor expansion for the exponential function, there are infinite terms, which means the number of features is infinite.

We would prefer kernel ridge when the number of polynomial features are much larger than the number of samples.

5g

For polynomial kernel, if we don't clip, we can see that as polynomial order increases, the polynomial explodes very quickly. RBF kernel doesn't have this problem, since the RBF kernel is always smaller than 1.

```
#!/usr/bin/env python3

import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
from numpy.linalg import norm
from math import exp

def lstsq(A, b, lambda_=0):
    return np.linalg.solve(A.T @ A + lambda_ * np.eye(A.shape[1]), A.T @ b)

def ploss(X, y, w):
    return norm(X @ w - y)**2/X.shape[0]

def assemble_feature(x, D):
    n_feature = x.shape[1]
    Q = [(np.ones(x.shape[0]), 0, 0)]
    i = 0
    while Q[i][1] < D:
        cx, degree, last_index = Q[i]
        for j in range(last_index, n_feature):
            Q.append((cx * x[:, j], degree + 1, j))
        i += 1
    return np.column_stack([q[0] for q in Q])

def poly_kernel(x, z, p):
    return (1 + x.T @ z)**p

def rbf_kernel(x, z, sigma):
    return exp(-norm(x-z)**2/2/sigma**2)

def kernel_matrix(X, Y, kernel, parameter):
    """
    calculate the kernel matrix given kernel type and kernel parameter
    X, Y have same feature dimension
    kernel : can be 'poly' or 'rbf'
    parameter: if 'poly' then this is p; if 'rbf' then this is sigma
    """
    Z = np.zeros((X.shape[0], Y.shape[0]))
    for i in range(X.shape[0]):
        for j in range(Y.shape[0]):
            if kernel == 'poly':
                Z[i, j] = poly_kernel(X[i, :].T, Y[j, :].T, parameter)
            if kernel == 'rbf':
```

```
            Z[i, j] = rbf_kernel(X[i, :].T, Y[j, :].T, parameter)
    return Z

def kernel_ridge(X, y, lambda_, kernel, parameter):
    K = kernel_matrix(X, X, kernel, parameter)
    return np.linalg.solve(K + lambda_ * np.eye(X.shape[0]), y)

def kloss(X, y, alpha, X_train, kernel, parameter):
    K = kernel_matrix(X, X_train, kernel, parameter)
    return norm(K @ alpha - y)**2/X.shape[0]

def visualize(X, y, ax):
    pos = y[:] == +1.0
    neg = y[:] == -1.0
    ax.scatter(X[pos, 0], X[pos, 1], c='red', marker='+')
    ax.scatter(X[neg, 0], X[neg, 1], c='blue', marker='v')
    return ax

def heatmap(X, y, ax, f, clip=5):
    # example: heatmap(lambda x, y: x * x + y * y)
    # clip: clip the function range to [-clip, clip] to generate a clean plot
    #   set it to zero to disable this function

    xx = yy = np.linspace(np.min(X), np.max(X), 72)
    x0, y0 = np.meshgrid(xx, yy)
    x0, y0 = x0.ravel(), y0.ravel()
    z0 = f(x0, y0)


    if clip:
        z0[z0 > clip] = clip
        z0[z0 < -clip] = -clip

    hb = ax.hexbin(x0, y0, C=z0, gridsize=50, cmap=cm.jet, bins=None)
    #plt.colorbar(ax = ax)
    cs = ax.contour(
        xx, yy, z0.reshape(xx.size, yy.size), [-2, -1, -0.5, 0, 0.5, 1, 2], cmap=cm.jet)
    ax.clabel(cs, inline=1, fontsize=10)

    visualize(X, y, ax)

    return hb, ax
```

```
#def main():
    # example usage of heatmap
#    heatmap(lambda x, y: x * x + y * y)

def a():
    fig, axs = plt.subplots(1, 3, figsize=(12, 4))
    for i, name in enumerate(['circle', 'heart', 'asymmetric']):
        data = np.load(name+'.npz')
        X = data["x"]
        y = data["y"]
        X /= np.max(X)  # normalize the data
        visualize(X, y, axs[i])
    plt.savefig('5a.jpg')
    plt.show()

def get_train_valid(name, split):
    data = np.load(name+'.npz')
    X = data["x"]
    y = data["y"]
    X /= np.max(X)  # normalize the data

    n_train = int(X.shape[0] * split)
    X_train = X[:n_train:, :]
    X_valid = X[n_train:, :]
    y_train = y[:n_train]
    y_valid = y[n_train:]
    return X, y, X_train, y_train, X_valid, y_valid

def b(name):
    SPLIT = 0.8
    LAMBDA = 0.001
    ps = np.arange(1, 17, 1)

    X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

    train_loss = np.zeros(ps.shape[0])
    valid_loss = np.zeros(ps.shape[0])
    for p in ps:
        w = lstsq(assemble_feature(X_train, p), y_train, LAMBDA)
        train_loss[p-1] = ploss(assemble_feature(X_train, p), y_train, w)
        valid_loss[p-1] = ploss(assemble_feature(X_valid, p), y_valid, w)
    print('for p = {} the training loss is {} and validation loss is {}'.format(ps, train_loss, valid_loss))
    plt.semilogy(ps, train_loss, label = 'training loss')
    plt.semilogy(ps, valid_loss, label = 'valid loss')
```

```
    plt.legend()
    plt.title('poly_loss_'+name)
    plt.savefig('poly_loss_'+name+'.jpg')
    plt.show()

    #ps = np.arange(2, 13, 2)
    #fig, axs = plt.subplots(1, ps.shape[0], figsize = (ps.shape[0]*4, 4))
    #for i, p in enumerate(ps):
    #    w = lstsq(assemble_feature(X_train, p), y_train, LAMBDA)
    #    f = lambda x, y: assemble_feature(np.hstack((x.reshape(x.shape[0], 1),
y.reshape(y.shape[0], 1))), p) @ w
    #    heatmap(X, y, axs[i], f, clip=5)
    #    axs[i].set_title('p=' + str(p))
    #plt.savefig('poly_heatmap_'+name+'.jpg')
    #plt.show()

def c(name):
    SPLIT = 0.8
    LAMBDA = 0.001
    ps = np.arange(1, 17, 1)

    X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

    #train_loss = np.zeros(ps.shape[0])
    #valid_loss = np.zeros(ps.shape[0])
    #for p in ps:
    #    alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='poly', parameter=p)
    #    train_loss[p-1] = kloss(X_train, y_train, alpha, X_train, kernel='poly', parameter=p)
    #    valid_loss[p-1] = kloss(X_valid, y_valid, alpha, X_train, kernel='poly', parameter=p)
    #print('for p = {} the training loss is {} and validation loss is {}'.format(ps, train_loss,
valid_loss))
    #plt.semilogy(ps, train_loss, label = 'training loss')
    #plt.semilogy(ps, valid_loss, label = 'valid loss')
    #plt.legend()
    #plt.title('poly_kernel_loss_'+name)
    #plt.savefig('poly_kernel_loss_'+name+'.jpg')
    #plt.show()

    ps = np.arange(2, 13, 2)
    fig, axs = plt.subplots(1, ps.shape[0], figsize = (ps.shape[0]*5, 4))
    for i, p in enumerate(ps):
        alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='poly', parameter=p)
        f = lambda x, y: kernel_matrix(np.hstack((x.reshape(x.shape[0], 1), y.reshape(y.shape[0],
1))),\
```

```
                                X_train, kernel='poly', parameter=p) @ alpha
        hb, ax = heatmap(X, y, axs[i], f, clip=0)
        axs[i].set_title('p=' + str(p))
        fig.colorbar(hb, ax=ax)
    plt.savefig('poly_kernel_heatmap_noclip_'+name+'.jpg')
    plt.show()

def c_part2(name):
    SPLIT = 0.15
    LAMBDA = 0.001
    ps = np.arange(1, 25, 1)

    X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

    train_loss = np.zeros(ps.shape[0])
    valid_loss = np.zeros(ps.shape[0])
    for p in ps:
        alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='poly', parameter=p)
        train_loss[p-1] = kloss(X_train, y_train, alpha, X_train, kernel='poly', parameter=p)
        valid_loss[p-1] = kloss(X_valid, y_valid, alpha, X_train, kernel='poly', parameter=p)
    print('for p = {} the training loss is {} and validation loss is {}'.format(ps, train_loss, valid_loss))
    plt.semilogy(ps, train_loss, label = 'training loss')
    plt.semilogy(ps, valid_loss, label = 'valid loss')
    plt.legend()
    plt.title('few_training_poly_kernel_loss_'+name)
    plt.savefig('few_training_poly_kernel_loss_'+name+'.jpg')
    plt.show()

def d(name):
    SPLIT = 0.8
    LAMBDAS = [0.0001, 0.001, 0.01]
    ps = [5, 6]

    X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)


    for j, p in enumerate(ps):
        for i, LAMBDA in enumerate(LAMBDAS):
            ns = np.arange(100, X_train.shape[0], 100)
            valid_loss = np.zeros(ns.shape[0])
            for k, n in enumerate(ns):
                loss = np.zeros(10)
                for h in range(10):
                    sample_index = np.random.choice(X_train.shape[0], size=n, replace=False)
```

```
            X_train_sample = X_train[sample_index, :]
            y_train_sample = y_train[sample_index]
            w = lstsq(assemble_feature(X_train_sample, p), y_train_sample, LAMBDA)

            loss[h] = ploss(assemble_feature(X_valid, p), y_valid, w)

          valid_loss[k] = np.mean(loss)

        plt.semilogx(ns, valid_loss, label = 'lambda = {}, p = {}'.format(LAMBDA, p))
  plt.legend()
  plt.title('poly_loss_vs_n_'+name)
  plt.savefig('poly_loss_vs_n_'+name+'.jpg')
  plt.show()

def e(name):
  SPLIT = 0.8
  LAMBDA = 0.001
  sigmas = [10, 3, 1, 0.3, 0.1, 0.03]

  X, y, X_train, y_train, X_valid, y_valid = get_train_valid(name, split=SPLIT)

  #train_loss = np.zeros(len(sigmas))
  #valid_loss = np.zeros(len(sigmas))
  #for i, sigma in enumerate(sigmas):
  #    alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='rbf', parameter=sigma)
  #    train_loss[i] = kloss(X_train, y_train, alpha, X_train, kernel='rbf', parameter=sigma)
  #    valid_loss[i] = kloss(X_valid, y_valid, alpha, X_train, kernel='rbf', parameter=sigma)
  #print('for sigma = {} the training loss is {} and validation loss is {}'.format(sigmas, train_loss,
valid_loss))
  #plt.loglog(sigmas, train_loss, label = 'training loss')
  #plt.loglog(sigmas, valid_loss, label = 'valid loss')
  #plt.legend()
  #plt.title('rbf_kernel_loss_'+name)
  #plt.savefig('rbf_kernel_loss_'+name+'.jpg')
  #plt.show()

  fig, axs = plt.subplots(1, len(sigmas), figsize = (len(sigmas)*5, 4))
  for i, sigma in enumerate(sigmas):
    alpha = kernel_ridge(X_train, y_train, LAMBDA, kernel='rbf', parameter=sigma)
    f = lambda x, y: kernel_matrix(np.hstack((x.reshape(x.shape[0], 1), y.reshape(y.shape[0],
1))),\
                          X_train, kernel='rbf', parameter=sigma) @ alpha
    hb, ax = heatmap(X, y, axs[i], f, clip=0)
    axs[i].set_title('sigma=' + str(sigma))
```

```
        fig.colorbar(hb, ax=ax)
    plt.savefig('rbf_kernel_heatmap_no_clip_'+name+'.jpg')
    plt.show()

if __name__ == "__main__":
    #a()

    #b('circle')
    #b('heart')
    #b('asymmetric')

    #c('circle')
    #c('heart')
    #c_part2('circle')

    #d('asymmetric')
    e('heart')
```