CSE 382M: Found. Data Science & ML **Spring 2025**

Problem Set 4

Student Name: Noah Reef

Problem 1

Let $\mathcal{H}$ be the set of linear hypothesis defined as

$$\mathcal{H} = \{h_a(x) := a^T x : ||a||_2 \leq B\}$$

and suppose that for all $(x, y) \in \mathcal{D}$ we have that

$$||x||_2 \leq X \quad \text{and} \quad ||y||_2 \leq Y$$

Then we can define the class of linear least square hypothesis as

$$\mathcal{L} = \{f(x) = \ell(h_a(x), y) := \frac{1}{2}(h_a(x) - y)^2 : h_a \in \mathcal{H}\}$$

Then we get that for the Radamacher complexity that

$$\begin{aligned}
\mathcal{R}_S(\mathcal{H}) &= \mathbb{E}_\sigma \left[\sup_{h_a \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{m} \sigma_i h_a(x_i) \right] = \frac{1}{n} \mathbb{E}_\sigma \left[\sup_{||a||_2 \leq B} \sum_{i=1}^{m} \sigma_i a^T x_i \right] \\
&\leq \frac{1}{n} \mathbb{E}_\sigma \left[\sup_{||a||_2 \leq B} \left| a^T \sum_{i=1}^{n} \sigma_i x_i \right| \right] \\
&\leq \frac{1}{n} \mathbb{E}_\sigma \left[\sup_{||a||_2 \leq B} ||a||_2 \left|\left| \sum_{i=1}^{n} \sigma_i x_i \right|\right|_2 \right] \\
&\leq \frac{B}{n} \mathbb{E}_\sigma \left[\left|\left| \sum_{i=1}^{n} \sigma_i x_i \right|\right|_2 \right] \\
&\leq \frac{B}{n} \sqrt{n} X
\end{aligned}$$

and hence we have that

$$\mathcal{R}_S(\mathcal{H}) \leq \frac{BX}{\sqrt{n}}$$

lastly we can define the loss function as

$$\phi_y(z) = \ell(z, y) = \frac{1}{2}(z - y)^2$$

with $z = h_a(x)$, and compute the derivative as

$$\phi_y'(z) = (z - y)$$

and get that

$$|\phi'_y(z)| \leq |z| + |y| = |h_a(x)| + |y| \leq BX + Y$$

and hence we have that

$$\mathcal{R}_S(\mathcal{L}) \leq \frac{BX}{\sqrt{n}}(BX + Y)$$

Problem 2

```python
import numpy as np
import pandas as pd

def nystrom_approx(X, sigma, r):
    K = kernel_matrix(X,sigma)
    A = K[0:r, 0:r]
    B = K[r:, 0:r]
    Z = np.block([[A],[B]])

    return Z @ np.linalg.pinv(A) @ Z.T

def fourier_approx(X, sigma, r):
    n, d = X.shape
    # Sample random frequencies from N(0, 1/sigma^2 I)
    W = np.random.normal(loc=0.0, scale=1.0/sigma, size=(d, r))
    # Sample random offsets from Uniform[0, 2pi]
    b = np.random.uniform(0, 2*np.pi, size=r)
    # Compute the feature mapping
    Phi = np.sqrt(2.0 / r) * np.cos(X @ W + b)
    return Phi @ Phi.T



def kernel_matrix(X, sigma=1):
    # Compute the squared norms of each row in X (shape: (n, 1))
    sum_X = np.sum(X ** 2, axis=1, keepdims=True)

    # Compute the pairwise squared Euclidean distances in a vectorized
    manner
    dists = sum_X + sum_X.T - 2 * X @ X.T

    # Compute the Gaussian kernel matrix
    K = np.exp(-dists / (2 * sigma ** 2))
    return K

# Create pandas DataFrame
df = pd.DataFrame(columns=["d", "n", "r", "Nystrom Error", "Fourier Error"
    ])

# Test the Nystrom approximation and Fourier approximation
ds = [2, 4 , 8 , 16]
ns = [1024,4096,16384]
rs = [128,512,1024]

```

```
for d in ds:
    for n in ns:
        # Generate random data
        mu = 0
        sigma = 1
        shape = (n, d)
        X = np.random.normal(mu,sigma,shape)

        # Construct total kernel matrix
        K = kernel_matrix(X)

        for r in rs:

            # Construct Nystrom approximation
            K_nys = nystrom_approx(X, 1, r)

            # Construct Fourier approximation
            K_fourier = fourier_approx(X, 1, r)

            # Compute the error
            nystrom_error = np.linalg.norm(K - K_nys) / np.linalg.norm(K)
            fourier_error = np.linalg.norm(K - K_fourier) / np.linalg.norm
    (K)


            # Append results to DataFrame
            df = pd.concat([df, pd.DataFrame({"d": [d], "n": [n], "r": [r
    ], "Nystrom Error": [nystrom_error], "Fourier Error": [fourier_error]})
    ], ignore_index=True)

# Save DataFrame to CSV
df.to_csv("kernel_approximation_errors.csv", index=False)

print(df.to_latex(
    index=False,
    float_format="%.4f",
    column_format="|c|c|c|c|c|",
    escape=False,
    caption="Kernel Approximation Errors",
    label="tab:kernel_approximation_errors"
))
```

d	n	r	Nystrom Error	Fourier Error
2	1024	128	0.0033	0.1743
2	1024	512	0.0013	0.0774
2	1024	1024	0.0006	0.0639
2	4096	128	0.0025	0.1860
2	4096	512	0.0013	0.0716
2	4096	1024	0.0009	0.0812
2	16384	128	0.0023	0.1383
2	16384	512	0.0012	0.0857
2	16384	1024	0.0008	0.0693
4	1024	128	0.0698	0.3975
4	1024	512	0.0141	0.2148
4	1024	1024	0.0000	0.1406
4	4096	128	0.0785	0.3973
4	4096	512	0.0130	0.1987
4	4096	1024	0.0054	0.1463
4	16384	128	0.0684	0.4436
4	16384	512	0.0121	0.2051
4	16384	1024	0.0046	0.1589
8	1024	128	0.6445	1.6959
8	1024	512	0.3354	0.8601
8	1024	1024	0.0000	0.6053
8	4096	128	0.6695	2.0762
8	4096	512	0.3766	1.0520
8	4096	1024	0.2622	0.7291
8	16384	128	0.6157	2.1504
8	16384	512	0.3460	1.0789
8	16384	1024	0.2422	0.7586
16	1024	128	0.9349	2.8316
16	1024	512	0.7059	1.4132
16	1024	1024	0.0000	0.9979
16	4096	128	0.9839	5.6222
16	4096	512	0.9333	2.8146
16	4096	1024	0.8624	1.9895
16	16384	128	0.9953	11.0760
16	16384	512	0.9805	5.5400
16	16384	1024	0.9619	3.9167

Problem 3

Suppose we have the following Kernel regression problem $Kw = y + \eta$ where η is some noise. If K is a full-rank kernel matrix we have that

$$K = U\Sigma V^T \quad \text{with} \quad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0$$

and hence the solution the above regression problem is given by,

$$w = K^{-1}(y+\eta) = \sum_{i=1}^{n} \frac{1}{\sigma_i} u_i^T (y+\eta) v_i$$

then for a low-rank approximation of our kernel matrix we have that $K_r = U_r \Sigma_r V_r^T$ and our solution vector is given by

$$w_r = \sum_{i=1}^{r} \frac{1}{\sigma_i} u_i^T (y+\eta) v_i = \sum_{i=1}^{r} \frac{1}{\sigma_i} u_i^T y v_i + \sum_{i=1}^{r} \frac{1}{\sigma_i} u_i^T \eta v_i = w_r^* + \delta w_r$$

then we get that

$$||\delta w_r|| \leq \frac{||\eta||}{\sigma_r}$$

and so

$$\frac{||\delta w_r||}{||w_r^*||} \leq \kappa_r \frac{||\eta||}{||y||}$$

Lastly to the error is not too large we require that

$$\kappa_r \frac{||\eta||}{||y||} \leq 1 \implies \sigma_r \geq \sigma_1 \frac{||\eta||}{||y||} \implies \sigma_r \geq \sigma_1 \frac{||\eta||}{||y||}$$

Problem 4

Part a

```python
import numpy as np
import pandas as pd
from sklearn.kernel_approximation import Nystroem
from sklearn.model_selection import KFold


def linear_regression_regularized(A, b, theta):
    r = A.shape[1]
    Z = A.T @ A + theta * np.eye(r)
    y = A.T @ b
    w = np.linalg.solve(Z, y)
    return w

df = pd.DataFrame(columns=["Dataset", "Best Theta", "Test Error"])

for i in range(2):

    # Load the in the dataset
    fl=np.load('CSE_382M/ps_4/ps_4 code/datasets/dataset{}.npz'. format(i+1))
    xtrain = fl['xtrain'].T
    ytrain = fl['ytrain'].T

```

```
    # regualarization parameter theta
    thetas = [1e-4,1e-3,1e-2,1e-1,1,10,100]

    # errors
    err = {}

    # Create K-Fold Cross-Validation split of the training dataset
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    kf.get_n_splits(xtrain)
    for train_index, test_index in kf.split(xtrain):
        x_train, x_test = xtrain[train_index], xtrain[test_index]
        y_train, y_test = ytrain[train_index], ytrain[test_index]

        # Loop over the regularization parameters
        for theta in thetas:
            feature_map = Nystroem(kernel='rbf', gamma=1, n_components
=100)
            x_train_transformed = feature_map.fit_transform(x_train)
            x_test_transformed = feature_map.transform(x_test)

            # Train the model
            w = linear_regression_regularized(x_train_transformed, y_train
, theta)

            # Make predictions
            y_pred = x_test_transformed @ w
            # Compute the error
            error = np.mean((y_test - y_pred) ** 2)
            # Store the error
            if theta not in err:
                err[theta] = []
            err[theta].append(error)
    # Compute the average error for each theta
    for theta in err:
        err[theta] = np.mean(err[theta])

    # pick best theta
    best_theta = min(err, key=err.get)

    # test the best theta
    yTest = fl['ytest'].T
    xTest = fl['xtest'].T
    xtrain = fl['xtrain'].T
    ytrain = fl['ytrain'].T

    feature_map = Nystroem(kernel='rbf', gamma=1, n_components=100)
    xtrain_transformed = feature_map.fit_transform(xtrain)
    xTest_transformed = feature_map.transform(xTest)
    w = linear_regression_regularized(xtrain_transformed, ytrain,
best_theta)
    y_pred = xTest_transformed @ w

    # Compute the error
    error = np.mean((yTest - y_pred) ** 2)
```

```
    # Append the results to the DataFrame
    df = pd.concat([df, pd.DataFrame({"Dataset": [i+1], "Best Theta": [best_theta], "Test Error": [error]})], ignore_index=True)

df.to_csv("dataset_errors.csv", index=False)

print(df.to_latex(index=False,
                  float_format="{:.3E}".format,
))
```

Part b

Dataset	Best Theta	Test Error
1	1.000E-03	9.973E-09
2	1.000E+00	1.226E-09