Amir Hossein Eyvazkhani
1747696
Ex4

Task 1) A python jupyter notebook is uploaded for this and also is attached at the end of this pdf.

Task 2)

(a) $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \left(f_\theta(x_n) - y_n \right)^2 + \lambda \|\theta\|_2^2$

$$\frac{dL(\theta)}{d\theta} = \frac{1}{N} (-X)^T 2 (y - X\theta) + 2\lambda\theta \quad : \text{from slides}$$

we put θ^* $\Rightarrow$ $\frac{dL(\theta^*)}{d\theta} = \frac{-2}{N} X^T \left(y - X\left(X^TX + N\lambda I\right)^{-1} X^T y \right) + 2\lambda \left(X^TX + N\lambda I \right)^{-1} X^T Y$

$$= \frac{-2}{N} X^T y + \frac{2}{N} \left(X^TX + \lambda I N \right) \left(X^TX + \lambda N I \right)^{-1} (X^T y) = \frac{-2}{N} \left(X^T y - X^T y \right)$$

$= 0$

(b) $\nabla^2 L = \frac{2}{N} X^T X + 2\lambda$

we had $\lambda > 0 \Rightarrow 2\lambda > 0$ (II)

$X^T X$ is positive indefinite if there is $\theta \in \mathbb{R}^n$ that $\theta^T X^T X \theta > 0$

so, $\theta^T X^T X \theta = (\theta X)^T (\theta X) = \|\theta X\|_2^2 > 0 \Rightarrow X^T X$ is possitive indefinite (I)

(I), (II) $\longrightarrow$ Hession is positive indefinite.

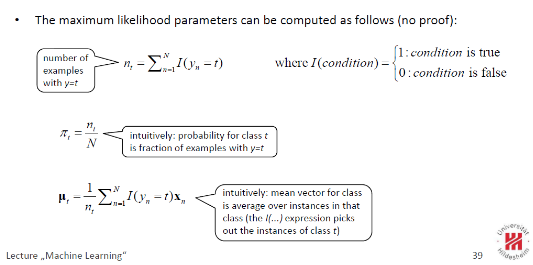
Task 3) A Python notebook and a pdf is attached.

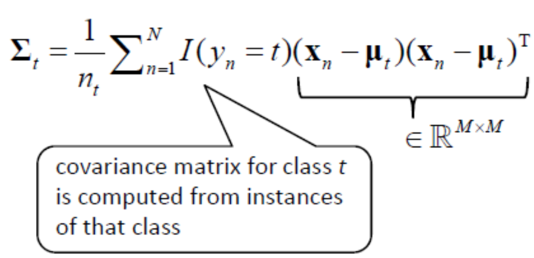
```python
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Task 1

Part a)

The formulas are taken from the slides






```python
x = np.array(
    [
        [1, 3],
        [2, 4],
        [1, 4],
        [2, 2],
        [3, 3],
        [3, 2]
    ]
)
y = np.array(
    [
        [2], [2], [2], [1], [1], [1]
    ]
)
# parameters
n_1 = len([value[0] for value in y if value[0] == 1])
n_2 = len([value[0] for value in y if value[0] == 2])
N = y.shape[0]
pi_1 = n_1/N
pi_2 = n_2/N
Mu_1 = np.mean(
    [value for index, value in enumerate(x) if y[index,0] == 1], axis=0
```

```
)
Mu_2 = np.mean(
    [value for index, value in enumerate(x) if y[index,0] == 2], axis=0
)
covar_matrix_1 = np.mean(
    [((value - Mu_1).reshape((len(value), 1)))@((value - Mu_1).reshape((len(value),
    , axis=0)
covar_matrix_2 = np.mean(
    [((value - Mu_2).reshape((len(value), 1)))@((value - Mu_2).reshape((len(value),
    , axis=0)
print(f'n_1: {n_1}')
print(f'n_2: {n_2}')
print(f'pi_1: {pi_1}')
print(f'pi_2: {pi_2}')
print(f'Mu_1: {Mu_1}')
print(f'Mu_2: {Mu_2}')
print(f'covariance matrix 1: {covar_matrix_1}')
print(f'covariance matrix 2: {covar_matrix_2}')
```

```
n_1: 3
n_2: 3
pi_1: 0.5
pi_2: 0.5
Mu_1: [2.66666667 2.33333333]
Mu_2: [1.33333333 3.66666667]
covariance matrix 1: [[0.22222222 0.11111111]
 [0.11111111 0.22222222]]
covariance matrix 2: [[0.22222222 0.11111111]
 [0.11111111 0.22222222]]
```

Part b)

In [ ]:

```
X = np.array([[1.5, 3]])

decision_function_class1 = -0.5 * np.log(np.linalg.det(covar_matrix_1)) - \
                           0.5 * np.sum(np.dot((X - Mu_1), np.linalg.inv(co
                    np.log(pi_1)

decision_function_class2 = -0.5 * np.log(np.linalg.det(covar_matrix_2)) - \
                           0.5 * np.sum(np.dot((X - Mu_2), np.linalg.inv(co
                    np.log(pi_2)

predictions = (decision_function_class1 > decision_function_class2).astype(int) + 1

print(f'The predicted class is: {predictions[0]}')
```

```
The predicted class is: 1
```

Task 3

Part a)

In [ ]:

```
np.random.seed(123)
dataset = fetch_california_housing()
Xdata = dataset["data"]
Ydata = dataset["target"]
```

```python
N, M = Xdata.shape
# Split into training and validation sets
ridx = np.random.permutation(N)
split = int(0.8*N)
Xtrain = Xdata[ridx[:split]]
Ytrain = Ydata[ridx[:split]]
Xvalid = Xdata[ridx[split:]]
Yvalid = Ydata[ridx[split:]]
reg = LinearRegression().fit(Xtrain, Ytrain) # fit model
intcp = reg.intercept_ # get intercept
coefs = reg.coef_ # get coefficients
# Get Predictions for training and validation sets
pred_train = reg.predict(Xtrain)
pred_val = reg.predict(Xvalid)
# Calculate Loss on training and validation sets
train_loss = mean_squared_error(pred_train, Ytrain)
val_loss = mean_squared_error(pred_val, Yvalid)
print(f'Train loss with sklearn: {train_loss}')
print(f'Test loss with sklearn: {val_loss}')
```

```
Train loss with sklearn: 0.5257192588422376
Test loss with sklearn: 0.5308531303306663
```

In [ ]:

```python
class regression:
    def __init__(self, x, y, x_valid, y_valid):
        self.x = np.column_stack((x, np.ones(len(x))))
        self.y = y
        self.x_valid = np.column_stack((x_valid, np.ones(len(x_valid))))
        self.y_valid = y_valid
        self.theta = None

    def fit(self):
        A = self.x.T @ self.x
        B = self.x.T @ self.y
        self.theta = np.linalg.lstsq(A, B, rcond=None)[0]

    def predict(self, data):
        return data@self.theta

    def validate(self):
        self.train_error = mean_squared_error(
            self.predict(self.x), self.y
        )
        self.valid_error = mean_squared_error(
            self.predict(self.x_valid), self.y_valid
        )

    def print_results(self):
        print(f'Train loss with regression: {self.train_error}')
        print(f'Test loss with regression: {self.valid_error}')


reg = regression(Xtrain, Ytrain, Xvalid, Yvalid)
reg.fit()
reg.validate()
reg.print_results()
```

```
Train loss with regression: 0.5257192588422377
Test loss with regression: 0.530853130330998
```

They match the results from sklearn

Part b)

In [ ]:
```
def forward_search():
    selected_variables = []
    remaining_variables = list(range(Xtrain.shape[1]))
    while remaining_variables:
        best_variable = None
        best_loss = float('inf')
        best_train_loss = None # just for printing as the question wanted
        for variable in remaining_variables:
            selected_vars = selected_variables + [variable]
            model = regression(Xtrain[:, selected_vars], Ytrain, Xvalid[:, selected
            model.fit()
            model.validate()
            loss_val = model.valid_error
            if loss_val < best_loss:
                best_loss = loss_val
                best_variable = variable
                best_train_loss = model.train_error
        selected_variables.append(best_variable)
        remaining_variables.remove(best_variable)

        print(f"Selected variables: {selected_variables}")
        print(f"Training loss: {best_train_loss}")
        print(f"Validation loss: {best_loss}")
        print("\n")
```

In [ ]:
```
forward_search()
```

```
Selected variables: [0]
Training loss: 0.7057142611240419
Validation loss: 0.6828174174305537


Selected variables: [0, 1]
Training loss: 0.6565408330102636
Validation loss: 0.6421602037629112


Selected variables: [0, 1, 6]
Training loss: 0.6454676255374441
Validation loss: 0.6327294459135423


Selected variables: [0, 1, 6, 7]
Training loss: 0.5412654842398897
Validation loss: 0.5379723120836373


Selected variables: [0, 1, 6, 7, 3]
Training loss: 0.5356213901380866
Validation loss: 0.5284987345411443


Selected variables: [0, 1, 6, 7, 3, 2]
Training loss: 0.5277066774799533
Validation loss: 0.5191577098527762


Selected variables: [0, 1, 6, 7, 3, 2, 4]
Training loss: 0.5275119429508665
Validation loss: 0.5198925555672947


Selected variables: [0, 1, 6, 7, 3, 2, 4, 5]
Training loss: 0.5257192588422377
Validation loss: 0.5308531303309508
```

In [ ]:

```
def backward_search():
    selected_variables = list(range(Xtrain.shape[1]))

    while len(selected_variables) > 1:
        best_variable = None
        best_loss = float('inf')
        best_train_loss = None # just for printing as the question wanted
        for variable in selected_variables:
            remaining_vars = selected_variables.copy()
            remaining_vars.remove(variable)
            model = regression(Xtrain[:, remaining_vars], Ytrain, Xvalid[:, remaini
            model.fit()
            model.validate()
            loss_val = model.valid_error
```

```
            if loss_val < best_loss:
                best_loss = loss_val
                best_variable = variable
                best_train_loss = model.train_error

        selected_variables.remove(best_variable)

        print(f"Selected variables: {selected_variables}")
        print(f"Training loss: {best_train_loss}")
        print(f"Validation loss: {best_loss}")
        print("\n")
```

In [ ]:

```
backward_search()
```

```
Selected variables: [0, 1, 2, 3, 4, 6, 7]
Training loss: 0.5275119429508663
Validation loss: 0.5198925555672468


Selected variables: [0, 1, 2, 3, 6, 7]
Training loss: 0.5277066774799533
Validation loss: 0.5191577098527814


Selected variables: [0, 1, 3, 6, 7]
Training loss: 0.5356213901380866
Validation loss: 0.5284987345411519


Selected variables: [0, 1, 6, 7]
Training loss: 0.5412654842398897
Validation loss: 0.5379723120836373


Selected variables: [0, 6, 7]
Training loss: 0.5552242366588557
Validation loss: 0.5470577783938771


Selected variables: [0, 6]
Training loss: 0.6948452870930405
Validation loss: 0.6729338841159962


Selected variables: [0]
Training loss: 0.7057142611240419
Validation loss: 0.6828174174305537
```

In [ ]: