Machine Learning Lab 1

Akhilendra Pratap 211112438

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import os

train_dirs=[]
test_dirs=[]
headers=[7, 7, 9, 7, 8]
for dir in os.listdir("./"):
    if(dir.find("5-fold")!=-1):
        train_dirs.append("./"+dir+"/train/")
        test_dirs.append("./"+dir+"/test/")
```

Linear Regression

```
def LinearRegression(train_file, test_file, header):
    train_df = pd.read_csv(train_file, header=header, delimiter=",")
    test_df = pd.read_csv(test_file, header=header, delimiter=",")

    X_train = train_df.iloc[:, :-1].values
    y_train = train_df.iloc[:, -1].values

    X_test = test_df.iloc[:, :-1].values
    y_test = test_df.iloc[:, -1].values

    from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()

    regressor.fit(X_train, y_train)

    y_pred = regressor.predict(X_test)

    from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2score = r2_score(y_test, y_pred)
    return np.array([mse, mae, r2score])


for train_dir, test_dir, header in zip(train_dirs, test_dirs,
headers):
    train_files=os.listdir(train_dir)
    test_files=os.listdir(test_dir)
    val=np.zeros(3)
    for train, test in zip(train_files, test_files):
        val+=LinearRegression(train_dir+train, test_dir+test, header)
    print(train_dir)
    val/=len(train_files)
    val[0]=math.sqrt(val[0])
    val=pd.DataFrame(val, index=["RMSE", "MSE", "R2"],
columns=["Values"])
    print(val)
    print("-----------------------------\n")
```

```
./diabetes-5-fold/train/
        Values
RMSE  0.639498
MSE   0.501970
R2   -0.000552
-----------------------------

./ele-1-5-fold/train/
          Values
RMSE  649.533859
MSE   421.387017
R2      0.682405
-----------------------------

./laser-5-fold/train/
         Values
RMSE  23.300207
MSE   15.579874
R2     0.746418
-----------------------------

./plastic-5-fold/train/
        Values
RMSE  1.531465
MSE   1.232442
R2    0.798437
-----------------------------

./quake-5-fold/train/
        Values
RMSE  0.189132
MSE   0.148620
R2    0.002162
-----------------------------
```

Polynomial Regression of degree 2 and 3

```
def PolynomialRegression(train_file, test_file, header, degree):
    train_df = pd.read_csv(train_file, header=header, delimiter=",")
    test_df = pd.read_csv(test_file, header=header, delimiter=",")

    X_train = train_df.iloc[:, :-1].values
    y_train = train_df.iloc[:, -1].values

    X_test = test_df.iloc[:, :-1].values
    y_test = test_df.iloc[:, -1].values

    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures

    poly_reg=PolynomialFeatures(degree=degree)
    X_poly=poly_reg.fit_transform(X_train)

    regressor = LinearRegression()
    regressor.fit(X_poly, y_train)

    y_pred = regressor.predict(poly_reg.transform(X_test))

    from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score

    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2score = r2_score(y_test, y_pred)
    return np.array([mse, mae, r2score])
```

Polynomial Regression of degree 2

```
for train_dir, test_dir, header in zip(train_dirs, test_dirs,
headers):
    train_files = os.listdir(train_dir)
    test_files = os.listdir(test_dir)

    val = np.zeros(3)
    for train, test in zip(train_files, test_files):
        val += PolynomialRegression(train_dir + train, test_dir +
test, header, 2)
    print(train_dir)
    val /= len(train_files)
    val[0] = math.sqrt(val[0])
    val = pd.DataFrame(val, index=["RMSE", "MSE", "R2"],
columns=["Values"])
    print(val)
    print("------------------------------\n")
```

```
./diabetes-5-fold/train/
        Values
RMSE  0.561297
MSE   0.456880
R2    0.226230
```

```
-----------------------------

./ele-1-5-fold/train/
          Values
RMSE  625.020558
MSE   416.170802
R2      0.704994
-----------------------------

./laser-5-fold/train/
         Values
RMSE  10.954232
MSE    6.686850
R2     0.944316
-----------------------------

./plastic-5-fold/train/
        Values
RMSE  1.528545
MSE   1.226209
R2    0.799254
-----------------------------

./quake-5-fold/train/
        Values
RMSE  0.189590
MSE   0.148552
R2   -0.002729
-----------------------------
```

Polynomial Regression of degree 3

```
for train_dir, test_dir, header in zip(train_dirs, test_dirs,
headers):
    train_files = os.listdir(train_dir)
    test_files = os.listdir(test_dir)

    val = np.zeros(3)
    for train, test in zip(train_files, test_files):
        val += PolynomialRegression(train_dir + train, test_dir +
test, header, 3)
    print(train_dir)
    val /= len(train_files)
    val[0] = math.sqrt(val[0])
    val = pd.DataFrame(val, index=["RMSE", "MSE", "R2"],
columns=["Values"])
    print(val)
    print("-----------------------------\n")
```

```
./diabetes-5-fold/train/
        Values
RMSE  0.838511
MSE   0.620331
R2   -0.519498
-----------------------------

./ele-1-5-fold/train/
          Values
RMSE  737.354993
MSE   427.489641
R2      0.588328
-----------------------------

./laser-5-fold/train/
        Values
RMSE  7.379026
MSE   3.270549
R2    0.975378
-----------------------------

./plastic-5-fold/train/
        Values
RMSE  1.473863
MSE   1.166224
R2    0.813267
-----------------------------

./quake-5-fold/train/
        Values
RMSE  0.189523
MSE   0.149088
R2   -0.002150
-----------------------------
```

Regularization in Linear Regression

```
def Regularization(train_file, test_file, header):
    train_df = pd.read_csv(train_file, header=header, delimiter=",")
    test_df = pd.read_csv(test_file, header=header, delimiter=",")

    X_train = train_df.iloc[:, :-1].values
    y_train = train_df.iloc[:, -1].values

    X_test = test_df.iloc[:, :-1].values
    y_test = test_df.iloc[:, -1].values

    from sklearn.linear_model import Ridge
    alphas = np.array([2**i for i in range(-18, 51, 2)])
    best_mse, best_alpha=float('inf'), None

    from sklearn.metrics import mean_squared_error
    for alpha in alphas:
        regressor=Ridge(alpha=alpha)
        regressor.fit(X_train, y_train)
        y_pred=regressor.predict(X_test)
        mse=mean_squared_error(y_test, y_pred)

        if mse<best_mse:
            best_mse, best_alpha=mse, alpha
    return np.array([best_mse, best_alpha])

for train_dir, test_dir, header in zip(train_dirs, test_dirs,
headers):
    train_files = os.listdir(train_dir)
    test_files = os.listdir(test_dir)

    val = np.zeros(2)
    for train, test in zip(train_files, test_files):
        val = Regularization(train_dir + train, test_dir + test,
header)
    print(train_dir)
    val[0]=math.sqrt(val[0])
    val = pd.DataFrame(val, index=["Best RMSE", "Best Alpha"],
columns=["Values"])
    print(val)
    print("-----------------------------\n")
```

```
./diabetes-5-fold/train/
              Values
Best RMSE   0.634371
Best Alpha  0.000004
-----------------------------

./ele-1-5-fold/train/
                Values
Best RMSE   577.159707
Best Alpha    0.000004
-----------------------------

./laser-5-fold/train/
               Values
Best RMSE   23.499185
Best Alpha   0.000004
-----------------------------

./plastic-5-fold/train/
              Values
```

```
Best RMSE   1.510579
Best Alpha  0.000004
-----------------------------

./quake-5-fold/train/
                   Values
Best RMSE        0.192593
Best Alpha  262144.000000
-----------------------------
```

```
import numpy as np
import math
```

Gradient Descent

```
def gradient_descent(X, y, learing_rate=0.01, iterations=1000):
    m, a, b=len(y), 0, 0
    costs=[]
    for _ in range(iterations):
        y_pred=a+b*X
        error=y_pred-y
        a=a-learing_rate*sum(error)/m
        b=b-learing_rate*error.dot(X)
        cost=(1/(2*m))*sum(error**2)
        costs.append(cost)
    return a, b, costs
```

Root Mean Squared Error

```
def root_mean_sqaured_error(y_pred, y):
    error=y-y_pred
    error=error**2
    s=sum(error)
    s/=len(y)
    return math.sqrt(s)
```

Mean Absolute Error

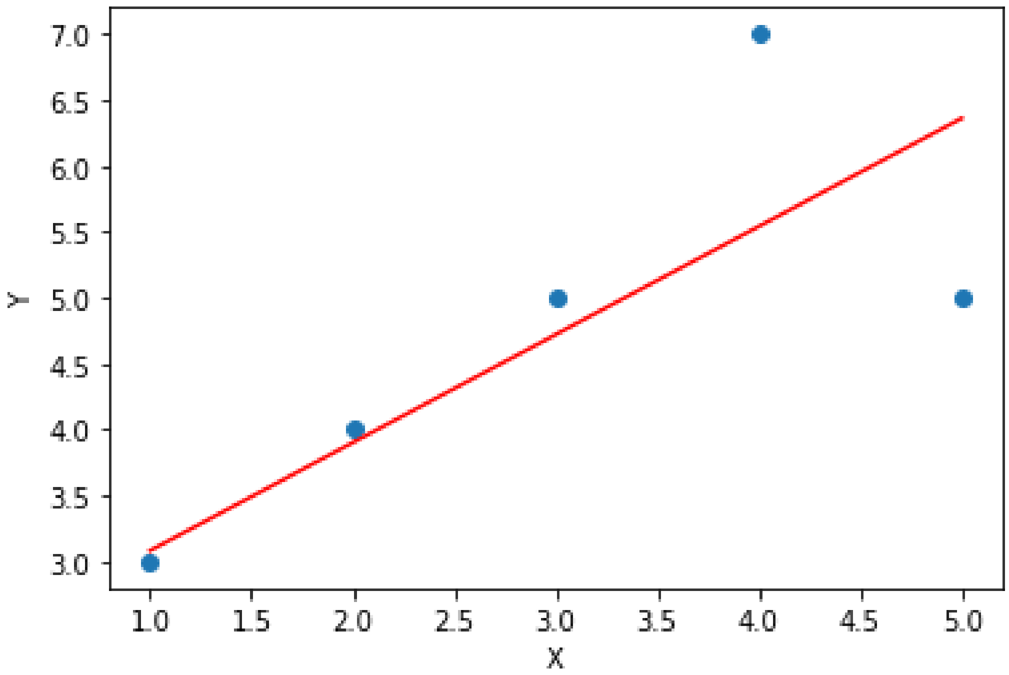
```
def mean_absoulute_error(y_pred, y):
    error=abs(y-y_pred)
    return sum(error)/len(y)
```

R2 Score

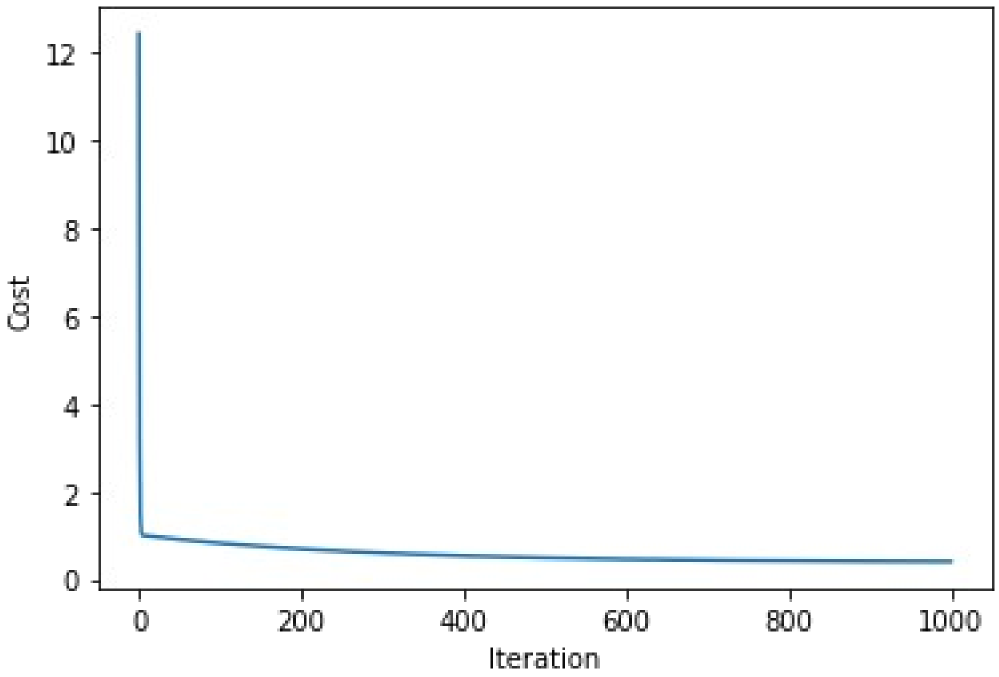
```
def r2_score(y_pred, y):
    avg=y.mean()
    SSres=sum((y-y_pred)**2)
    SStot=sum((y-avg)**2)
    return 1-SSres/SStot

X=np.arange(1, 6)
Y=np.array([3, 4, 5, 7, 5])

a, b, costs=gradient_descent(X, Y)
y_pred=a+b*X

print(a, b, root_mean_sqaured_error(y_pred, Y),
mean_absoulute_error(y_pred, Y), r2_score(y_pred, Y))
```

```
2.263493307840752 0.8194363043824198 0.9025768411894179
0.6556395558023976 0.537133548720757
```

```
import matplotlib.pyplot as plt

plt.scatter(X, Y)
plt.plot(X, y_pred, color="red")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



```
plt.plot(costs)
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.show()
```



Ridge

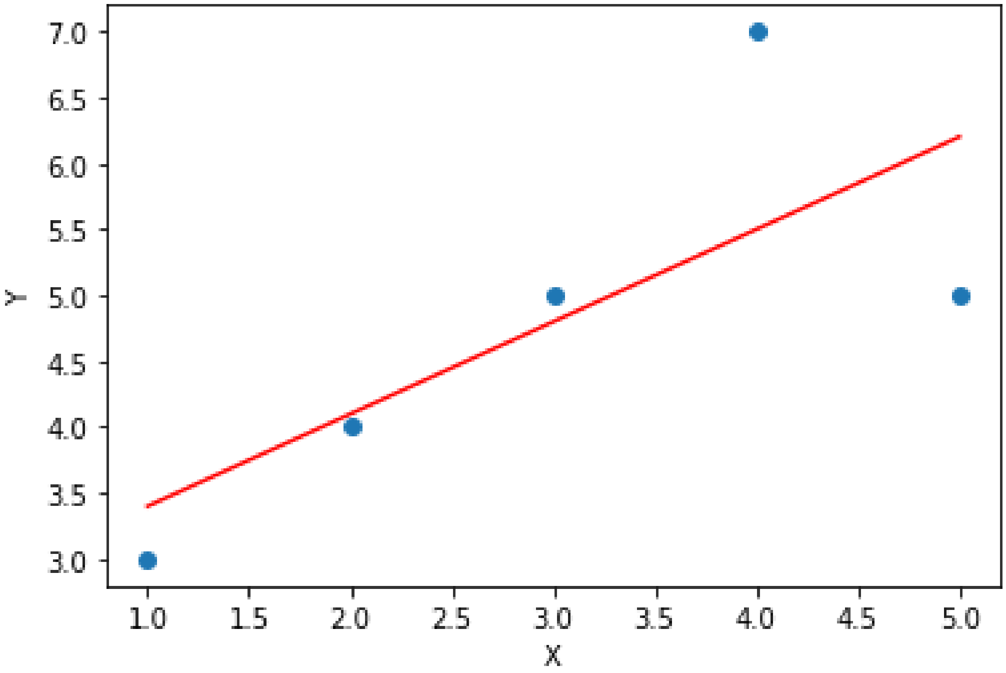
```python
def Regularization(X, Y):
    from sklearn.linear_model import Ridge

    alphas = np.array([2**i for i in range(-18, 51, 2)])

    best_mse, best_alpha = float("inf"), None

    from sklearn.metrics import mean_squared_error

    for alpha in alphas:
        regressor = Ridge(alpha=alpha)
        regressor.fit(X, Y)
        y_pred = regressor.predict(X)
        mse = mean_squared_error(Y, y_pred)

        if mse < best_mse:
            best_mse, best_alpha = mse, alpha
    return np.array([best_mse, best_alpha])

X=X.reshape(-1, 1)
Y=Y.reshape(-1, 1)
mse, alpha=Regularization(X, Y)

print(mse, alpha)
```

```
0.7800000000001427 3.814697265625e-06
```

```
from sklearn.linear_model import Ridge

regressor=Ridge(alpha=alpha)
regressor.fit(X, Y)
```

```
Ridge(alpha=3.814697265625e-06)
```

```
import matplotlib.pyplot as plt

plt.scatter(X, Y)
plt.plot(X, regressor.predict(X), color="red")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



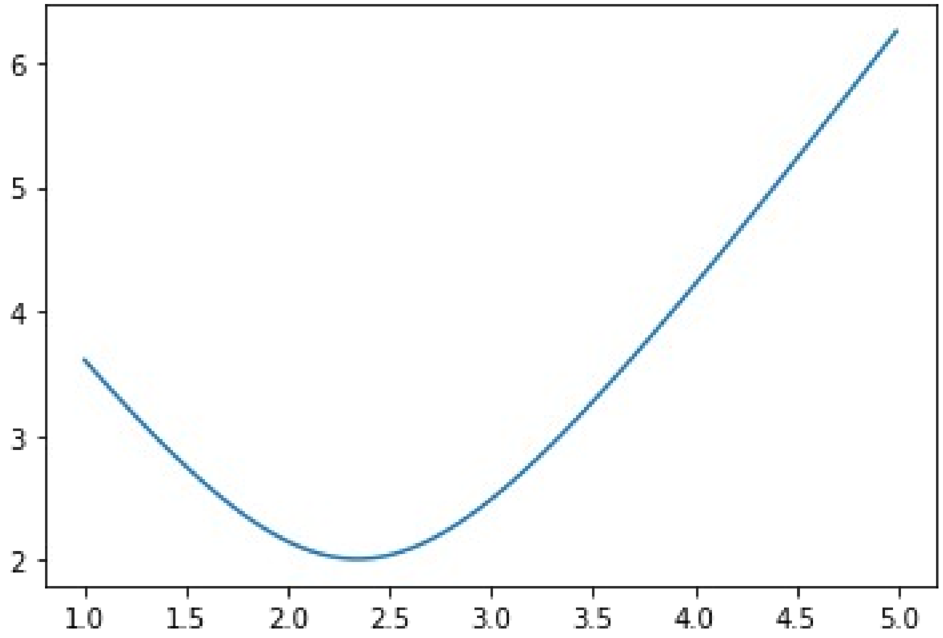
```
from sklearn.metrics import mean_squared_error
mse=mean_squared_error(Y, regressor.predict(X))
print(mse)
```

```
0.7800000000001427
```

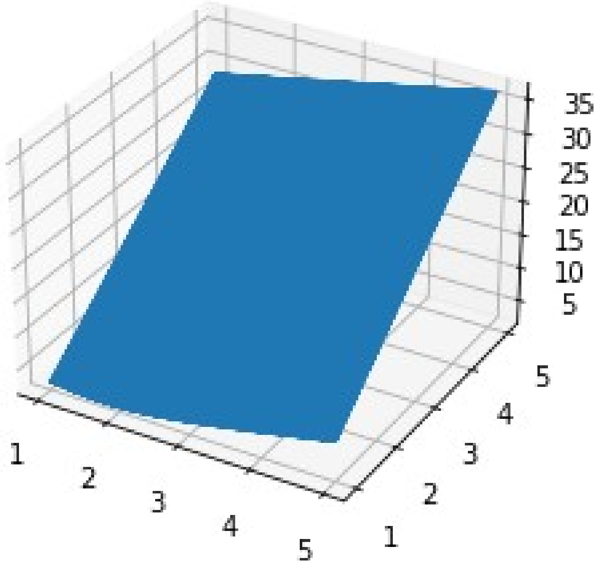
```
costs=[]
A=np.arange(1, 5, 0.01)
for a in A:
    y_pred=a+X*b
    error=math.sqrt(sum((Y-y_pred)**2))
    costs.append(error)

costs=np.array(costs)
```

```
plt.plot(A, costs)
```

```
[<matplotlib.lines.Line2D at 0x24e897cb340>]
```



```
A = np.arange(1, 5, 0.01)
B = np.arange(1, 5, 0.01)
x, y=[], []
costs = []
for a in A:
    for b in B:
        y_pred = a + X * b
        x.append(a)
        y.append(b)
        error = math.sqrt(sum((Y - y_pred) ** 2))
        costs.append(error)

x=np.array(x)
y=np.array(y)
costs=np.array(costs)

ax=plt.axes(projection="3d")
ax.plot3D(x, y, costs)
plt.show()
```


35
30
25
20
15
10
5
5
4
3
2
1
1
2
3
4
5