Machine Learning CAP6610

Assignment 1 :- / Homework 1

① Regression to Mean :-

Given, $x_i \Rightarrow$ parent's height

$y_i \Rightarrow$ child's height

$\mu_x = \mu_y = \mu$ $\quad 0 < \rho < 1$

$\sigma_x = \sigma_y = \sigma$ $\quad \hat{y} = f(x)$

S.t, $\mu < \hat{y} < x_i$ when $x > \mu$

or $x < \hat{y} < \mu$ when $x < \mu$

Since x is scalar, the prediction model is a straight line, that fits the dataset (x_i, y_i) with least loss.

the line can be given as $f(x) = ax + b$

Using "Hint", the prediction model is,

$$\hat{y} = \mu_y + \rho \frac{\sigma_y}{\sigma_x}(x - \mu_x)$$

substituting $\sigma_y = \sigma_x = \sigma$ and $\mu_y = \mu_x = \mu$.

$$\hat{y} = \mu + \rho \frac{\cancel{\sigma}}{\cancel{\sigma}}(x - \mu)$$

~~$\hat{y} = \mu + \rho x - \rho \mu$~~

$$(\hat{y} - \mu) = \rho (x - \mu)$$

Since $0 < \rho < 1$

if $x - \mu > 0$

or $x > \mu$,

$$(\hat{y} - \mu) = \rho(x - \mu)$$

can be expressed as,

$$0 < \hat{y} - \mu < x - \mu$$

adding μ,

Case 1: $\rightarrow$ $\boxed{\mu < \hat{y} < x \quad \text{when} \quad x > \mu}$

if $x - \mu < 0$, or $x < \mu$,

$$x - \mu < \hat{y} - \mu < 0$$

adding μ,

Case 2: $\rightarrow$ $\boxed{x < \hat{y} < \mu \quad \text{when} \quad \mu > x}$

Case 3:

when $\mu = x$, $(\hat{y} - x) = \rho(x - x)$

$$\hat{y} = x$$

$\therefore$ when $\mu = x$, there's a perfect correlation between $\hat{y}$ x and y, i.e, parent's and child's height.

(2) Given,

Least Square Regression

model 1 $\Rightarrow f(x) = \theta_1 \varphi_1(x) + \dots + \theta_m \varphi_m(x)$ $\rightarrow$ m basis fn

model 2 $\Rightarrow \tilde{f}(x) = \tilde{\theta}_1 \varphi_1(x) + \dots + \tilde{\theta}_m \varphi_m(x) + \tilde{\theta}_{m+1} \varphi_{m+1}$

$\Rightarrow (m+1)$ basis fn.

st;
$$\frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \geq \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{f}(x_i))^2$$

$\rightarrow$

Let ϕ be ~~column~~ matrix for ~~mod~~ m basis functions,

$$\phi = \begin{bmatrix} | & | & & | \\ \varphi_1 & \varphi_2 & \cdots & \varphi_m \\ | & | & & | \end{bmatrix}_{n \times m}$$

Assuming n is ~~size~~ size of data or number of samples.

Similarly, let $\tilde{\phi}$ be matrix for m+1 basis functions,

$$\tilde{\phi} = \begin{bmatrix} | & | & & | & | \\ \varphi_1 & \varphi_2 & \cdots & \varphi_m & \varphi_{m+1} \\ | & | & & | & | \end{bmatrix}_{n \times (m+1)}$$

Since $\tilde{\phi}$ has all m basis function similar to ϕ, we can say, column space of $\tilde{\phi}$ includes column space of ϕ, or column space of ϕ is a subspace of column space of $\tilde{\phi}$.

$$col(\phi) \subseteq col(\tilde{\phi})$$

The least square problem now becomes about projecting y into column space of ϕ or $\tilde{\phi}$ to minimize error.

Since col($\tilde{\phi}$) includes col(ϕ) ~~it is going~~ ~~to project~~ ~~of~~ the projection of y in col($\tilde{\phi}$) is going to be much closer than the projection of y in col(ϕ)

Let $P_\phi(y) \Rightarrow$ projection of y in ϕ

& $P_{\tilde{\phi}}(y) \Rightarrow$ projection of y in $\tilde{\phi}$

$$\therefore \|y - P_{\tilde{\phi}}(y)\|^2 \leq \|y - P_\phi(y)\|^2$$

The two ~~err~~ errors in projections are equal when $\theta_{m+1} = 0$, i.e, the $(m+1)$ basis function doesn't contribute to the projection.

~~Expressing Projections in terms of f(x) and f̃(x) as errors, we get~~ using $f(x)$ and $\tilde{f}(x)$ as $P_\phi(y)$ and $P_{\tilde{\phi}}(y)$

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \geq \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{f}(x_i))^2$$

$\therefore$ More basis always reduces training error, but if not regularized may lead to overfitting and not perform well on test data.

Assignment 1_3

February 4, 2025

0.0.1 3

[10]:
```
phi = randn(40,10);
psi = randn(40);
```

[11]:
```
theta_star = phi\psi;
```

[12]:
```
using LinearAlgebra;
loss = norm(phi*theta_star - psi)^2;
```

[13]:
```
theta_star
```

[13]:
```
10-element Vector{Float64}:
 0.09445657568075688
 0.09505880102567173
 0.339523301054718
 0.08158984213513429
 0.07741292050163236
 0.03167319737476269
 0.2908630264721047
 0.010331296031684714
 0.04356279550166443
 0.34397346549075386
```

[19]:
```
print("Loss while estimating theta(theta_star): ", loss)
```

```
Loss while estimating theta(theta_star): 21.976225886019318
```

[35]:
```
# verifying inquality || phi * (theta_star + sigma) - psi || ^2 > || phi␣
↪*theta_star - psi || ^2
function verifyInequality()
    sigma = randn(10);
    print("Testing inequlity with sigma as : \n", sigma, "\n")
    lhs = norm(phi*(theta_star + sigma) - psi) ^ 2;
    rhs = loss;
    print("Does the inequality hold thus proving that no approximation of theta␣
↪other than theta_star has lower loss?")
    lhs > rhs
end
```

[35]: verifyInequality (generic function with 1 method)

[36]:
```
print("Verifying inequality || phi * (theta_star + sigma) - psi || ^2 > || phi␣
↪*theta_star - psi || ^2 ")
```

```
Verifying inequality || phi * (theta_star + sigma) - psi || ^2 > || phi
*theta_star - psi || ^2
```

[37]:
```
verifyInequality()
```

```
Testing inequlity with sigma as :
[0.20546735761226462, 0.04258280531378687, 0.7272327498768144,
-0.35086792902882535, -1.0819662582607297, 0.975170287364697,
-1.5224711318291555, 1.1978334861771822, 0.009280002293177056,
0.0984519130758678]
Does the inequality hold thus proving that no approximation of theta other than
theta_star has lower loss?
```

[37]: true

[38]:
```
verifyInequality()
```

```
Testing inequlity with sigma as :
[-0.1139782803337731, 0.6158918590428364, 2.201795467165284, 0.333883111171912,
0.11537499801982071, -1.5550850204473206, -0.6264512177973692,
-0.9911144148957922, 0.09955737889305871, 1.076660670669673]
Does the inequality hold thus proving that no approximation of theta other than
theta_star has lower loss?
```

[38]: true

[39]:
```
verifyInequality()
```

```
Testing inequlity with sigma as :
[-0.3421842416523434, 0.9326139271256874, -0.36010491178037296,
2.0162017160321426, -0.6868668189542076, -1.3867325178905994,
-1.4448523071897577, -2.2003030283697367, -0.040000106074750186,
-1.9003356862996872]
Does the inequality hold thus proving that no approximation of theta other than
theta_star has lower loss?
```

[39]: true

[40]:
```
verifyInequality()
```

```
Testing inequlity with sigma as :
[0.0539281707246908, -0.23677946714303807, 1.1466172184860852,
0.5623905191431423, 0.3269112146444907, 0.37238427685715597,
0.22890575040606476, 1.9757464403480802, 1.5042207219925152, 0.8614238784722701]
```

Does the inequality hold thus proving that no approximation of theta other than theta_star has lower loss?

[40]: true

[41]:
```
verifyInequality()
```

Testing inequlity with sigma as :
[-1.2271962315890752, 1.0186090920037258, 0.845130222602616,
0.21576243970813883, -0.10079657499241743, -0.4476669158500418,
-0.6538591194735733, -0.30920125260659764, -1.885126228257937,
0.2293229222274903]
Does the inequality hold thus proving that no approximation of theta other than theta_star has lower loss?

[41]: true

[ ]:

Assignment_1_4

February 4, 2025

0.0.1 4

[1]:
```python
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
import numpy as np
```

[7]:
```python
df = pd.read_csv("wine+quality/winequality-red.csv", sep=';')
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, y_train = X[:1400], y[:1400]
X_test, y_test = X[1400:], y[1400:]

# Standardize the features (x-mean(x))/std(x)
scaler = StandardScaler()
# Use X_train to get the mean and standard deviation
X_train_std = scaler.fit_transform(X_train)
# Use the mean and standard deviation from X_train to standardize X_test
# as having X_train and X_test with same standardization terms makes sense.
X_test_std = scaler.transform(X_test)

def addBiasToSamples(data):
    return np.hstack([data, np.ones((data.shape[0], 1))])

X_train_std = addBiasToSamples(X_train_std)
X_test_std = addBiasToSamples(X_test_std)

lambdas = [0, 10**-1, 10**-2, 10**-3]
```

[3]:
```python
def trainModel(regularization_coefficient = 0):
    model = Ridge(alpha = regularization_coefficient, fit_intercept=False)
    model.fit(X_train_std, y_train)
    return model
```

[4]:
```python
for i in lambdas:
    model = trainModel(regularization_coefficient = i)
    y_pred = model.predict(X_test_std)
    mse = np.square(np.subtract(y_test, y_pred)).mean()
```

```
    print("MSE for predictions using lambda =", i, "is", mse)
```

```
MSE for predictions using lambda = 0 is 0.4868691102549516
MSE for predictions using lambda = 0.1 is 0.48679037671598435
MSE for predictions using lambda = 0.01 is 0.48686122087563066
MSE for predictions using lambda = 0.001 is 0.4868683211567339
```

[5]:
```
print(y_pred)
```

```
[4.96968956 4.96968956 6.48022124 6.24348    5.63512459 6.47637574
 6.46109026 6.02312003 6.89340766 6.02312003 5.46661258 5.78955934
 6.46109026 5.65754314 5.81636801 5.49213338 5.81636801 6.4858995
 5.45928528 4.99971898 5.45928528 5.42065771 6.13049355 5.73302682
 5.80205261 5.80205261 6.46573438 5.98007449 5.73193801 6.55401253
 5.75838624 5.43865946 6.38829602 5.80489126 5.12262846 5.12262846
 5.03487309 5.29849859 5.52115807 5.64123401 6.24890783 4.88163005
 5.49439433 5.9155035  5.6504892  4.89088525 5.49439433 5.39623184
 5.4979915  6.27302043 6.24890783 6.13299239 6.13077909 5.09326975
 5.97816429 5.46479226 5.57130752 5.09326975 5.94152713 6.67692155
 5.78388268 5.25063352 5.52565753 5.58974503 5.44456581 5.44456581
 5.60922918 5.20959451 5.60922918 4.92164916 5.3607052  6.07610532
 6.12068586 5.65823825 5.11819883 6.66163544 5.11819883 6.67089063
 5.14817026 5.92353078 5.41030272 5.92353078 5.43703254 6.03564813
 5.4251022  5.26327806 5.45832632 5.73961589 5.86960525 5.91860042
 6.56346192 5.86960525 5.9184233  5.05574706 5.87604842 6.02692571
 5.05574706 5.81454868 5.35158744 5.81454868 5.24524079 4.94936123
 5.27307185 5.92059197 5.96361459 5.27970716 5.51249091 5.96361459
 6.01107977 6.36950504 5.74679358 5.38331099 5.4211661  5.56146144
 4.89482979 4.89945739 6.13267612 5.85236808 5.66991921 5.31143101
 5.85236808 5.14986791 6.13267612 5.63604178 5.7493092  5.57614418
 5.56575644 5.88257051 5.73469342 5.45081879 6.11977941 5.35983716
 5.70049255 5.26546232 6.01578337 5.40934096 5.5923982  5.56253454
 5.9415341  5.72713178 5.94434946 6.18087881 5.44623096 5.83650948
 6.32927604 5.50006971 5.60417372 5.99304497 5.76466264 6.27083328
 5.14403345 5.148213   5.6844273  5.14590846 5.46554709 5.74804687
 5.0883812  5.46554709 4.85902189 5.10385055 5.10385055 5.10385055
 5.39039244 5.39039244 5.39039244 5.86671578 6.29045623 5.39039244
 5.26232752 5.94473069 6.35346603 6.10780072 4.95790905 6.09495147
 5.5555253  6.15586868 6.1272792  5.82643018 5.74015994 5.89219546
 6.26680807 5.89219546 5.78357829 5.37212228 6.4130757  6.28950899
 6.35690954 5.72260439 6.19578207 4.94999131 6.20273905 5.6177888
 5.95113282 5.50020717 5.54474851 5.96881715 5.95113282 5.49756939
 6.02443865]
```

[6]:
```
y_test
```

[6]:
```
array([5, 5, 6, 8, 6, 7, 6, 6, 7, 6, 6, 6, 6, 5, 5, 5, 5, 7, 5, 5, 5, 5,
       6, 4, 6, 6, 6, 5, 5, 5, 5, 6, 6, 7, 6, 6, 5, 5, 5, 6, 7, 6, 5, 5,
```

```
6, 6, 5, 5, 5, 8, 7, 7, 7, 5, 6, 6, 6, 5, 5, 7, 6, 4, 6, 6, 5, 5,
7, 4, 7, 3, 5, 5, 6, 5, 5, 7, 5, 7, 3, 5, 4, 5, 4, 5, 4, 5, 5, 5,
5, 6, 6, 5, 5, 5, 7, 6, 5, 6, 6, 6, 5, 5, 5, 6, 6, 3, 6, 6, 6, 5,
6, 5, 6, 6, 6, 6, 5, 6, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 6, 6, 6, 5,
6, 5, 7, 6, 6, 6, 5, 5, 6, 7, 6, 6, 7, 6, 5, 5, 5, 8, 5, 5, 6, 5,
6, 7, 5, 6, 5, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6,
6, 6, 6, 5, 6, 5, 5, 5, 7, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 5,
6])
```

[ ]:

Assignment_1_5

February 4, 2025

0.0.1 5

[1]:
```
import pandas as pd
from sklearn.linear_model import RidgeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np
```

[14]:
```
# fetch data from https://archive.ics.uci.edu/dataset/52/ionosphere
# used the code mentioned in the website
# we will treat good as "1" and bad as "-1" in the class labels.

from ucimlrepo import fetch_ucirepo

# fetch dataset
ionosphere = fetch_ucirepo(id=52)

# data (as pandas dataframes)
X = ionosphere.data.features.values
Y = ionosphere.data.targets.values

y = []
# convert your to binary
for i in range(len(Y)):
    y.append(1 if Y[i] == 'g' else -1)
```

[15]:
```
X_train, y_train = X[:300], y[:300]
X_test, y_test = X[300:], y[300:]

# Standardizing features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

# adding constant term to each sample
X_train_std = np.hstack([np.ones((X_train_std.shape[0], 1)), X_train_std])
X_test_std = np.hstack([np.ones((X_test_std.shape[0], 1)), X_test_std])
```

```
lambdas = [0, 10**-1, 10**-2, 10**-3]
```

[16]:
```
# RidgeClassifier converts class/target variable to +1/-1 internally
# and outputs contain original class value and not +1/-1.
for _lambda in lambdas:
    model = RidgeClassifier(alpha=_lambda)
    model.fit(X_train_std, y_train)
    y_pred = model.predict(X_test_std)
    print("Accuracy with lambda =", _lambda, "is" , accuracy_score(y_pred,␣
 ↪y_test))
```

```
Accuracy with lambda = 0 is 0.9803921568627451
Accuracy with lambda = 0.1 is 1.0
Accuracy with lambda = 0.01 is 1.0
Accuracy with lambda = 0.001 is 1.0
```

[17]:
```
model.coef_
```

[17]:
```
array([[ 0.        ,  0.2319215 ,  0.        ,  0.18686956,  0.10769536,
         0.17877135,  0.09732167,  0.15415996,  0.19582622,  0.17179077,
         0.05284186, -0.05137689, -0.02870209, -0.04701518,  0.01539959,
         0.04125438, -0.01826716,  0.05579179,  0.05036423, -0.13846341,
         0.07718832, -0.02540749, -0.23438013,  0.18273207,  0.05848456,
         0.08582776,  0.11103445, -0.17830319, -0.08222595,  0.05609893,
         0.0708795 ,  0.13426188,  0.053514  , -0.09828889, -0.17625284]])
```