DATA7202 Assignment 1

Longpeng Xu

March 23, 2023

Contents

1 Question 1 code 1

2 Question 2 2

3 Question 3 2

4 Question 4 code 3

5 Question 5 code 4

6 Question 6 code 5

7 Question 7 code 5

8 Appendix 6

1 Question 1 code

Inferences about the coefficients

- X_3 ('internet') has the greatest contribution to the sales, for its greatest coefficient 5.6428. X_2 ('tv') has the medium contribution to the model, for its coefficient 3.6350. X_1 ('radio') has the smallest contribution to the model, for its smallest coefficient 0.4449.
- All the coefficients are individually significant, since the p-values of the t-tests are significantly small (less than 10^{-3}), rejecting the null hypothesis that a coefficient is equal to zero.
- The p-value of F-test is less than 10^{-2}, rejecting the null hypothesis that all of the coefficients are equally zero.

Linear regression mean squared error: 0.375525

Random forest regressor mean squared error: 0.415045 (Both rounded to six decimal places)

2 Question 2

Given $X_1, ..., X_n \sim N(\mu, \sigma^2)$, the hypothesis class is

$$\mathcal{H} = \{f(\mathbf{x}, \theta);\ \theta \in \Theta\} = \{f(\mathbf{x}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\mathbf{x}-\mu)^2}{2\sigma^2}};\ \mu \in \mathbb{R},\ \sigma^2 \in \mathbb{R}^+ \cup \{0\}\}$$

where θ and Θ are

$$\theta = \{\mu, \sigma^2\},\ \Theta = \mathbb{R} \times (\mathbb{R}^+ \cup \{0\})$$

Notes

$\triangle$ As an extreme case, σ^2 can be 0.

$\triangle$ Since $\sigma = \sqrt{\sigma^2}$, $\sigma \in \mathbb{R}^+ \cup \{0\}$.

3 Question 3

Proof Let $\mathcal{H}$ be the set of binary classifiers g. Set domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y} \sim \mathcal{D}$ ($\mathcal{D}$ is unknown) where $\boldsymbol{z_1}, ..., \boldsymbol{z_m}$ in sample $\mathcal{T}$ are i.i.d. samples from $\mathcal{D}$. $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$ is a loss function.

$$\begin{aligned}
\text{Loss}_{\mathcal{T}}(g) &\overset{\text{def}}{=\!=} \frac{1}{m}\sum_{i=1}^{m} l(g, \boldsymbol{z_i}) \\
\mathbb{E}_{\mathcal{T}}\text{Loss}_{\mathcal{T}}(g) &= \mathbb{E}_{\mathcal{T}}\frac{1}{m}\sum_{i=1}^{m} l(g, \boldsymbol{z_i}) \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_i}) \\
&\overset{\triangle}{=\!=} \frac{1}{m} \cdot m \cdot \mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_i}) \\
&= \mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_i}) \\
&\overset{\triangle}{=\!=} \mathbb{E}_{\boldsymbol{Z}\sim\mathcal{D}} l(g, \boldsymbol{Z}) \\
&\overset{\text{def}}{=\!=} \text{Loss}_{\mathcal{D}}(g) \qquad \blacksquare
\end{aligned}$$

Notes

$\triangle$ Since $\boldsymbol{z_1}, ..., \boldsymbol{z_m} \sim^{\text{i.i.d}} \mathcal{D}$, $\mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_1}) = ... = \mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_m}) = \mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_m})$

$\triangle$ When m, the size of $\mathcal{T}$, is infinitely large, $\mathbb{E}_{\mathcal{T}} l(g, \boldsymbol{z_i})$ converges to $\mathbb{E}_{\boldsymbol{Z}\sim\mathcal{D}} l(g, \boldsymbol{Z})$, since $\mathcal{T} \to \boldsymbol{Z} \sim \mathcal{D}$ as $m \to \infty$

We also note it is the binary classifiers that $\mathcal{H}$ contains. If the loss functions are expressed as specific zero-one losses rather than the generalized losses as above, with an strong assumption, the proof can go as below:

Proof ′ The training set is $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^m$ where $(x_i)_{i=1}^m$ follows distribution $\mathcal{D}$. Assume the true labelling function $g^* : (x_1, ..., x_m) \to (y_1, ..., y_m)$, $g^* \in \mathcal{H}$ exists, as in Lecture 1.

$$\begin{aligned}
\mathbb{E}_{\mathcal{T}}\text{Loss}_{\mathcal{T}}(g) &= \mathbb{E}_{\mathcal{T}} \frac{\sum_{i=1}^m \mathbb{1}\{g(x_i) \neq y_i\}}{m} \\
&= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{\mathcal{T}}\mathbb{1}\{g(x_i) \neq y_i\} \\
&\overset{\triangle}{=} \mathbb{E}\ \mathbb{E}[\mathbb{1}\{g(X) \neq Y\}|Y = g^*(X)] \\
&\overset{\triangle}{=} \mathbb{E}_{\mathcal{D}}\mathbb{1}\{g(X) \neq g^*(X)\} \\
&= \mathbb{P}_{X\sim\mathcal{D}}[g(X) \neq g^*(X)] \\
&= \text{Loss}_{\mathcal{D}}(g) \qquad \blacksquare
\end{aligned}$$

Notes ′

△ Given $g^* : (x_1, ..., x_m) \to (y_1, ..., y_m)$, indicator $\mathbb{1}\{g(X) \neq Y\}$ is conditional on $Y = g^*(X)$, as any hypothesis other than g^* cannot satisfy the mapping. Not conditional on X as $(x_i)_{i=1}^m \sim \mathcal{D}$ is predefined.

△ By Adam's Law: For any r.v.s. U and V, $\mathbb{E}[\mathbb{E}[V|U]] = \mathbb{E}[V]$.

4 Question 4 code

(a)

Model	β_0	β_1
Model_1	$\frac{11}{5}$	0
Model_2	0	0.7333

△ For Model_1, $\mathbb{E}\varepsilon = 0 \Rightarrow \mathbb{E}[y_i - \beta_0] = 0 \Rightarrow \frac{1}{n}\sum_{i=1}^n (y_i - \beta_0) = 0 \Rightarrow \sum_{i=1}^n (y_i - \beta_0) = 0 \Rightarrow \sum_{i=1}^n y_i = n\beta_0 \Rightarrow \beta_0 = \frac{1}{5}(3 + 2 + 1 + 2 + 3) = \frac{11}{5} = 2.2$

△ For Model_2, fit the data into `statsmodels.api.OLS` in Python yields $\beta_1 = 0.7333$ (See appendix)

(b) For average losses,

Model	squared error loss	absolute error loss	$L_{1.5}$ loss
Model_1	0.56	0.64	0.5849
Model_2	2.1733	1	1.4094

$\text{Model}_1 :\ y = \frac{11}{5} + \varepsilon$

○ Squared error loss: $\frac{1}{5}[2 \cdot (3 - \frac{11}{5})^2 + 2 \cdot (2 - \frac{11}{5})^2 + (1 - \frac{11}{5})^2] = 0.56$

○ Absolute error loss: $\frac{1}{5}[2 \cdot |3 - \frac{11}{5}| + 2 \cdot |2 - \frac{11}{5}| + |1 - \frac{11}{5}|] = 0.64$

○ $L_{1.5}$ loss: $\frac{1}{5}[2 \cdot |3 - \frac{11}{5}|^{1.5} + 2 \cdot |2 - \frac{11}{5}|^{1.5} + |1 - \frac{11}{5}|^{1.5}] \approx 0.5849$

$\text{Model}_2 :\ y = 0.7333x_1 + \varepsilon$

○ Squared error loss: $\frac{1}{5}[(3 - 0.7333 \cdot 4)^2 + (2 - 0.7333 \cdot 3)^2 + (1 - 0.7333 \cdot 2)^2 + (2 - 0.7333 \cdot 1)^2 + (3 - 0.7333 \cdot 0)^2] \approx 2.1733$

- Absolute error loss: $\frac{1}{5}[|3-0.7333\cdot4|+|2-0.7333\cdot3|+|1-0.7333\cdot2|+|2-0.7333\cdot1|+|3-0.7333\cdot0|] = 1$
- $L_{1.5}$ loss: $\frac{1}{5}[|3-0.7333\cdot4|^{1.5}+|2-0.7333\cdot3|^{1.5}+|1-0.7333\cdot2|^{1.5}+|2-0.7333\cdot1|^{1.5}+|3-0.7333\cdot0|^{1.5}] \approx 1.4094$

(c) Without introducing x_1, Model$_1$ generates smaller absolute error loss, $L_{1.5}$ loss and squared error loss. By introducing x_1, Model$_2$ has more uncertainty which leads to the higher losses, compared to those of the Model$_1$ without x_1.

5 Question 5 ^{code}

(a) See Appendix.

(b)
- ▶ The cardinalities of 'League', 'Division' and 'NewLeague' are 2 (i.e. Each column has only 2 categorical values). Hence, using LabelEncoder, by encoding 'League' = N as 1 and 'League = A' as 0, for example, does not introduce the issue of hierarchy which brings misinterpretation.
- ▶ LabelEncoder is more space efficient, using only one column for each categorical variable (i.e., each original column), while OneHotEncoder costs k column for a categorical variable of cardinality k.

(c) 10-fold cross-validation mean square error $CV_{10} = \sum_{k=1}^{10} \frac{n_k}{n} l_{\mathcal{T}_{-k}} \approx 116613.23$

Notes:

1. The problem does not indicate whether an intercept must be included in linear regression. Hence, the data is fitted by `statsmodels.api.OLS` which is default to exclude the intercept, rather than `sklearn.linear_model.LinearRegression` where `fit_intercept=False` should be specified.
2. $CV_K \approx 116613.23$ is calculated by taking the mean of the $l_{\mathcal{T}_{-k}}$ list, i.e., assume all folds share the same weight, despite the intricate fold-splitting method is specified by scikit-learn developers.

6 Question 6 code

Method

Algorithm 1 Generation of a discrete uniform variable

function DISC_UNIF(n)

input: the number of iteration n

output: a random variable which follows discrete uniform distribution

    create a variable res default to be 0
    create a list $int\_list = \{0, 1, ..., n\}$
    create a r.v. int_sample from Unif$(0, 1)$ and transform it to an integer in $[0, n]$
    for i in $\{0, 1, ..., n\}$ **do**
        if $int\_list[i]$ equals to int_sample **then**
            res is $int\_list[i]$
        else
            $i = i + 1$
        end if
    end for
    return res

end function

Explanation

- The time complexity of the algorithm is $\boldsymbol{O}(n)$, since there are n iterations in the for-loop, each iteration has fixed number of operations, and the operations outside for-loop is counted as an constant.
- Hence, the total number of operations is proportional to n especially when n is large, which gives $\boldsymbol{O}(n)$.

7 Question 7 code

- The 95% confidence interval for l is [0.239381, 0.241285], rounded to six decimal places.
- The obtained CMC estimation for 0.240333 (without setting a random seed), while the true value l is 0.240301, both rounded to six decimal places. Hence, the CMC estimation is very similar to the true value l.

8 Appendix

Listing 1: **Code for Questions**

```
# ----------------------------------------------
# Question 1
# ----------------------------------------------

import numpy as np
import pandas as pd
import statsmodels.api as sm

def CreateDataset(N, seed):
    np.random.seed(seed)

    X1 = np.random.rand(N)
    X2 = np.random.rand(N) * (1-X1)
    # X1 = np.random.uniform(low=0.0, high=1.0, size=N)
    # X2 = np.array([np.random.uniform(low=0.0, high=1-x) for x in X1])
    X3 = 1 - X1 - X2

    Y = np.zeros((N,))

    for i in  range(N):
        # noise = np.random.uniform(low = -1.0, high = 1.0)
        if(np.random.rand()<0.5):
            noise = np.random.rand()
        else:
            noise = -np.random.rand()
        Y[i] = 0.5*X1[i] + 3*X2[i] + 5*X3[i] + 5*X2[i]*X3[i] + 2*X1[i]*X2[i]*X3[i] + noise

    data = np.array([X1, X2, X3 ,Y]).T
    df = pd.DataFrame(data,columns=['radio','tv','internet','sales'])
    return df

if __name__ == "__main__":

    df_train = CreateDataset(1000, 1)
    df_test = CreateDataset(1000, 2)
    X_train = df_train[['radio','tv','internet']]
    y_train = df_train[['sales']].values.reshape(-1,)
    X_test = df_test[['radio','tv','internet']]
    y_test = df_test[['sales']].values.reshape(-1,)

    lr = sm.OLS(y_train, X_train).fit()
    print(lr.summary())
    print('*' * 20)

    y_pred_lr = lr.predict(X_test)
    print("Linear regression mean squared error: ",  round(np.mean(np.power((y_test - y_pred_lr).values, 2)), 6))
    print('*' * 20)

```

```
    from sklearn.ensemble import RandomForestRegressor
    rfr = RandomForestRegressor(500, random_state=0)
    rfr.fit(X_train, y_train)
    y_pred_rfr = rfr.predict(X_test)
    print("Random forest regressor mean squared error: ",  round(np.mean(np.power((y_test - y_pred_rfr), 2)), 6))

# -----------------------------------------------
# Question 4
# -----------------------------------------------

import statsmodels.api as sm

X_1_ = [4,3,2,1,0]
Y_ = [3,2,1,2,3]
lm_ = sm.OLS(Y_, X_1_).fit()
print(lm_.summary())

# -----------------------------------------------
# Question 5
# -----------------------------------------------

import pandas as pd
import numpy as np
from sklearn import preprocessing
import statsmodels.api as sm
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

hitters = pd.read_csv('Hitters.csv', header=0)
hitters

hitters.dtypes

le = preprocessing.LabelEncoder()
change = ['League', 'Division', 'NewLeague']
for colname in change:
    hitters[colname] = le.fit_transform(hitters[colname])
hitters

X_ = hitters.drop(['Salary'], axis=1)
y_ = hitters['Salary']

X_

y_

def Validate(X,Y):
    kf = KFold(n_splits = 10)
    kf.get_n_splits(X)
    mse_cv = []
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index, :], X.iloc[test_index, :]
```

```
        y_train, y_test = Y.iloc[train_index], Y.iloc[test_index]

        lm = sm.OLS(y_train, X_train).fit()

        y_pred = lm.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        mse_cv.append(mse)

    return 'Mean square error for each fold: ', [ round(i,2) for i in mse_cv],\
           '10-Fold cross validation mean square error: ',  round(np.mean(mse_cv), 2)

Validate(X_, y_)

# ----------------------------------------------
# Question 6
# ----------------------------------------------
import numpy as np

def disc_unif(n):
    res = 0
    int_list =  list( range(n+1))
    int_sample =  int(np.random.uniform(low=0, high=1) * n)
    for i in  range( len(int_list)):
        if int_list[i] == int_sample:
            res += int_list[i]
        else:
            i += 1
    return res

disc_unif(100)

# ----------------------------------------------
# Question 7
# ----------------------------------------------

from math import sqrt
from math import atan
import scipy.stats as stats
import numpy as np

def l_func(x, a=1, b=2, c=3, low=0, high=1):
    return 1 / (a*x**2 + b*x + c) * (high - low)

def cmc(N=10000, alpha=0.05):
    sum_of_l = 0
    list_l = []
    z = stats.norm.ppf(1 - alpha / 2)
    for i in  range(N):
        x = np.random.uniform(low=0.0, high=1.0)
        sum_of_l += l_func(x, a=1, b=2, c=3, low=0, high=1)
        list_l += [l_func(x, a=1, b=2, c=3, low=0, high=1)]
    mean_l = sum_of_l / N
```

```
    std_l = sqrt( sum([(l - mean_l) ** 2 for l in list_l]) / N)
    lower = mean_l - z * std_l / sqrt(N)
    upper = mean_l + z * std_l / sqrt(N)
    print('The CMC estimate for l is',  round(mean_l, 6), 'with',  int((1-alpha)*100),\
          '% confidence interval', [ round(lower, 6),  round(upper, 6)])
    return None

cmc(N=10000, alpha=0.05)

def primitive_l(x, a=1, b=2, c=3):
    return (2 / sqrt(4*a*c - b**2)) * atan((2*a*x + b) / sqrt(4*a*c - b**2))

def true_val_l(low=0, high=1):
    return  round(primitive_l(high) - primitive_l(low), 6)

print('The true value of l is', true_val_l(low=0, high=1))
```