CSE 158/258, Fall 2019: Homework 1

Zhankui He - A53312511

Tasks - Regression

Problem 1

- We load this dataset in to pandas dataframe, and finish the data preprocessing by using `len` and converting "Y/N" into 1/0 to take advantage of 'verified purchase' and 'the length of the review' features.
- The number of rows of the dataset is 148310. After dropping `NaN` values in this dataset, we get 148304 rows.

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

plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

path = "amazon_reviews_us_Gift_Card_v1_00.tsv"
df = pd.read_csv(path, sep="\t")
df.head()

print("# Rows:", len(df))
df = df.dropna()
print("# Rows:", len(df))

review_length = np.array(df["review_body"].apply(lambda x: len(str(x))))
verified = np.array(df["verified_purchase"].apply(lambda x: int(x=='Y')))
assert len(review_length) == len(verified)
```

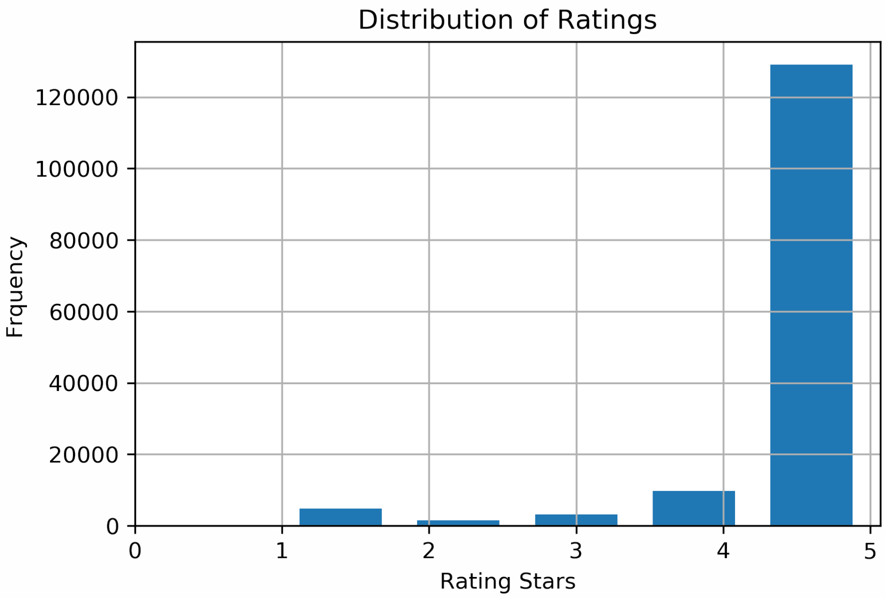
```
# Rows: 148310
# Rows: 148304
```

We plot the distribution of ratings, and that the most ratings is 5-star (about 87.00%), and the least ratings is 2-star (about 1.05%). From this we know the dataset is imbalanced extremely and concentrating on 5-star ratings.

```
for i in range(1,6):
    print("Percentage of %d stars: %.4f%%" % (i,
100*sum(df["star_rating"]==i)/len(df)))
```

```
Percentage of 1 stars: 3.2137%
Percentage of 2 stars: 1.0519%
Percentage of 3 stars: 2.1220%
Percentage of 4 stars: 6.6128%
Percentage of 5 stars: 86.9997%
```

```
# Problem 1
plt.hist(
    df["star_rating"],
    rwidth=0.7,
    bins=5
)
plt.xticks(np.arange(6), range(6))
plt.xlabel("Rating Stars")
plt.ylabel("Frquency")
plt.title("Distribution of Ratings")
plt.grid()
plt.show()
```



Problem 3

We construct a class of Linear Regression (without regularization) and solve this model using closed-form solution. Here:

$$star\ rating \simeq \theta_0 + \theta_1 \times [review\ is\ verified] + \theta_2 \times [review\ length]$$

Now, we can represent the parameters θ_1, θ_2, θ_3 as vector Θ and represent the data points of constant value 1 and other two features as matrix X, the star ratings as Y. The model can be expressed as:

$$Y \simeq X\Theta$$

We can estimate the parameters Θ as:

$$\Theta = (X^T X)^{-1} X^T Y$$

```
class LinearRegression():
    def __init__(self, x, y):
        self.x = x # (b, n)
        self.y = y # (b, 1)
        self.theta = np.zeros((self.x.shape[1], 1)) # (n, 1)

    def solve(self):
        self.theta = np.dot(np.dot(np.linalg.inv(np.dot(self.x.T, self.x)),
self.x.T), self.y)

    def inference(self, x):
        return np.dot(x, self.theta)

def split(x, y, ratio, shuffle=False, seed=5583):
    assert len(x) == len(y)

    indices = np.arange(len(x))
    if shuffle:
        np.random.seed(seed)
        np.random.shuffle(indices)

    idx_train = indices[:int(len(x)*ratio)]
    idx_test = indices[int(len(x)*ratio):]

    return x[idx_train], y[idx_train], x[idx_test], y[idx_test]
```

We will report the values of θ_0, θ_1, and θ_2.

param	value	interpretation
θ_0	4.8450	The basic rating stars bias of every item, which means if there is no "purchase verified" and "reviews length" is zero, this is the predicted rating score in this model.
θ_1	0.0499	This parameter reprensts the replationship of "purchase verified" and rating stars. i.e. if it's true, the predicted score will increase by 0.0499 compared with the case where it's false, and vice versa.
θ_2	-0.0012	This parameter reprensts the replationship of "length of review" and rating stars. if the increase of every character in reviews can represent the decrease of 0.0012 predicted score, and vice versa.

```
x = np.vstack(([1]*len(verified), verified, review_length)).T
y = np.array(df["star_rating"])[:, None]

model = LinearRegression(x, y)
model.solve()

for i, t in zip(range(len(model.theta)), model.theta):
    print("theta %d: %.4f" % (i, t))
```

```
theta 0: 4.8450
theta 1: 0.0499
theta 2: -0.0012
```

Problem 4

In problem 4, we only use the feature of "review is verified", i.e.

$$star\ rating \simeq \theta_0 + \theta_1 \times [review\ is\ verified]$$

The theta values are:

param	value
θ_0	4.5781
θ_1	0.1680

In the last problem, the θ_0 and θ_1 represent the same features as this problem's. But the value of θ_0 decreases by 0.2669, and the value of θ_1 increases by 0.1181.

Interpretation: Though θ_0 and θ_1 represent the same features, but the meaning of these coefficients are different slightly.

- For θ_0, the value means the basic rating score if "purchase verified"=N instead of cosidering "purcase verified"=N and "review length"=0. Also, from the last model we know when "purchase verified"=N and "review length" increasing, the predicted score will decrease, so in our model, the value of θ_0 should be smaller than the last one for compensation.
- For θ_1, when "purchase verified"=Y, our model predicted score will be lower than the last one if θ_1 remain as the same. So to compensate the case, the value of θ_1 should be larger than the last.

From the analysis above, we will know it makes sense for why these coefficients might vary so significantly.

```
x = np.vstack(([1]*len(verified), verified)).T
y = np.array(df["star_rating"])[:, None]

model = LinearRegression(x, y)
model.solve()

for i, t in zip(range(len(model.theta)), model.theta):
    print("theta %d: %.4f" % (i, t))
```

```
theta 0: 4.5781
theta 1: 0.1680
```

Problem 5

In this scenario, we know MSE stands for Mean Squared Error:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Here MSE value for training and test dataset is:

dataset	MSE
train	0.6211
test	0.9550

```
x = np.vstack(([1]*len(verified), verified, review_length)).T
y = np.array(df["star_rating"])[:, None]

def split_train(x, y, ratio):
    x_train, y_train, x_test, y_test = split(x, y, ratio)
    model = LinearRegression(x_train, y_train)
    model.solve()
    y_pred = model.inference(x_train)
    mse_train = np.mean((y_pred-y_train)**2)
    y_pred = model.inference(x_test)
    mse_test = np.mean((y_pred-y_test)**2)
    return mse_train, mse_test

print("MSE of training %.4f \nMSE of testing %.4f" % split_train(x, y, 0.9))
```

```
MSE of training 0.6211
MSE of testing 0.9550
```

Problem 7

We conduct the experiment with varying the size of the training and test fractions from 5% to 95% with step of 5% to show the relationship between MSE with the size of training and test dataset using plots.

Description: Actually the size of the training set make a significant difference in testing performance. We find that, with the increasing of the ratio of training set in the whole dataset, the MSE of test set is increasing generally and the MSE of training set is decresing until the ratio=45% and increasing after that. And when ratio=5%, MSE of the test is the least one which is meaning the best generalization of this model.

Interpretation: Generally speaking, when we increase the ratio of training set, it will be helpful for the generalization of our model so we will get the better performance on test set. However, in this problem, the result is not the same as we expected. Why? We conjecture these training set and test set might not be independent and identically distributed, and the distribution difference varies with the size ratio because we should keep the order of data when splitting. We draw a simple plot to illusrate this point as below.

```
step = 0.05
ratios = np.arange(0.05, 0.95+step, step)
train_res, test_res = [], []
for r in ratios:
    tr, te = split_train(x, y, r)
    train_res.append(tr)
    test_res.append(te)

plt.plot(train_res, 'o-', label="train mse", markersize=5)
plt.plot(test_res, 's-', label="test mse", markersize=5)
plt.xticks(range(len(ratios)), ["%.2f" % i for i in ratios], rotation=-90)
plt.title("MSE with Different Ratio of Train-Test-Split Dataset")
plt.legend()
plt.xlabel("Ratio of Train-Test-Split Dataset")
plt.ylabel("MSE")
plt.grid()
plt.show()
```

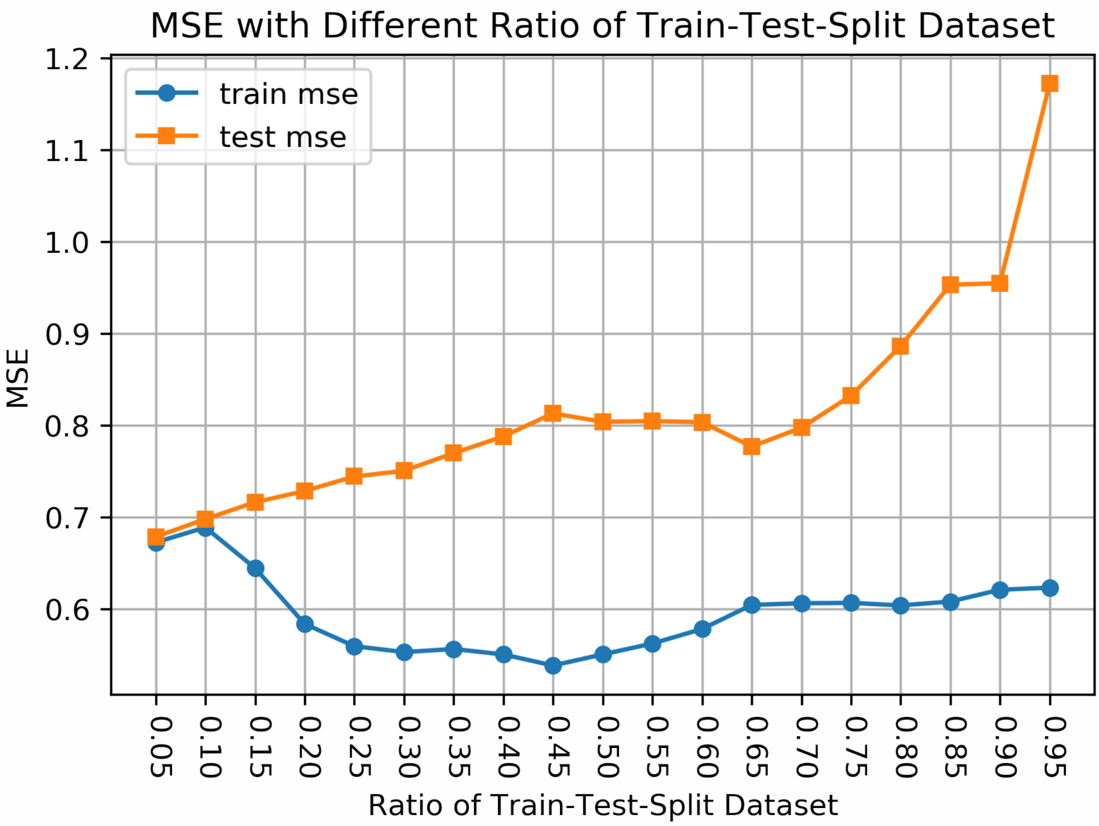


Illustration: We define the proportion of 5-star ratings in training and test set as P:

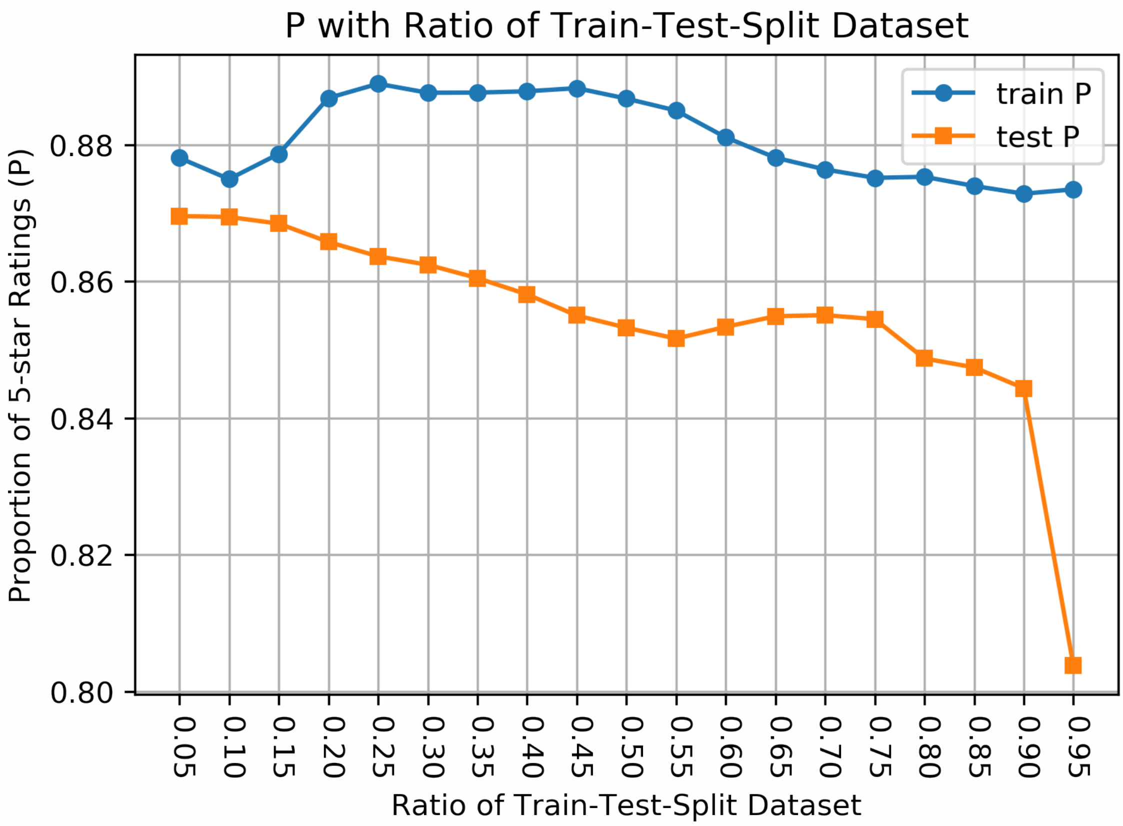
$$P = \frac{\#\{5star\ ratings\}}{\#\{ratings\}}$$

to plot the relationship between P with Ratio of Train-Test-Split Dataset. Here P is simple but good enough to show the difference between training and test set distribution. And when ratio=5%,10%, the distribution is more similar than ratio increasing. When ratio=95%, the proportion of 5-star ratings in test drops to about 81% sharply while that's 88% in training set. It's reasonable why the generalization performance becomes much worse when ratio=95% in the plot above.

```
train_ratio, test_ratio = [], []
for r in ratios:
    x_train, y_train, x_test, y_test = split(x, y, ratio=r)
    train_ratio.append(sum(y_train==5)/len(y_train))
    test_ratio.append(sum(y_test==5)/len(y_test))

plt.plot(train_ratio, 'o-', label="train P", markersize=5)
plt.plot(test_ratio, 's-', label="test P", markersize=5)

plt.xticks(range(len(ratios)), ["%.2f" % i for i in ratios], rotation=-90)
plt.title("P with Ratio of Train-Test-Split Dataset")
plt.legend()
plt.xlabel("Ratio of Train-Test-Split Dataset")
plt.ylabel("Proportion of 5-star Ratings (P)")
plt.grid()
plt.show()
```



Tasks - Classification

Problem 8

It's a classification task with logistic regression model. We can formulate the model as:

$$p(review\ is\ verified) \simeq \sigma(\theta_0 + \theta_1 \times [star\ rating] + \theta_2 \times [review\ length])$$

Here $\sigma(x) = (1+\exp(-x))^{-1}$ and we can write this model in simpler notations as:

$$P = \sigma(X\Theta)$$

where we represent the parameters θ_1, θ_2, θ_3 as vector Θ and represent the data points of constant value 1 and other two features as matrix X, the predict probability as P.

- In **Trial 1**, we realize the `LogisticRegressor` by hand and estimate the parameters using SGD with L2 regularization, where we set learning rate as 0.01, epochs as 1000 and the weight of L2 regularization as 0.01.
- In **Trial 2**, we call the `LogisticRegression` function in `scikit-learn` toolbox and train our model using the default hyper-parameters.

```python
import warnings
from sklearn.metrics import classification_report

warnings.filterwarnings('ignore')

x = np.vstack(([1]*len(verified), np.array(df["star_rating"]),
review_length)).T
y = verified[:, None]

x_train, y_train, x_test, y_test = split(x, y, 0.9)
```

Trial 1

The results of trial 1 is :

Metrics	Value
Accuracy	55.9571%
Proportion of labels that are positive	55.9571%
Proportion of predictions that are positive	100%

```python
class LogisticRegressor():
    def __init__(self, x, y, seed=5583):
        self.x = x # (b,n)
        self.y = y # (b,1)

        np.random.seed(seed)
        self.theta = np.random.rand(x.shape[1],1) # (n,1)

    def solve(self, lr=0.1, reg=0, epochs=100):
        for epoch in range(epochs):
            y_ = self.inference(self.x) # (b,1)
            delta_theta = np.dot(self.x.T, ((-1)**self.y) * y_ * (1-y_)) #
(n,b)*(b,1) = (n,1)
            self.theta -= lr*delta_theta + reg*self.theta

    def inference(self, x):
        return 1/(1+np.exp(-np.dot(x, self.theta))) # (b,1)

model = LogisticRegressor(x_train, y_train)
model.solve(lr=0.01, epochs=1000, reg=0.01)

y_ = (model.inference(x_test) > 0.5).astype(np.int)

print("Classification Report:\n\n", classification_report(np.squeeze(y_test),
np.squeeze(y_)))
```

```
print("Accuracy: %.4f%%" % (100*sum(np.squeeze(y_) ==
np.squeeze(y_test))/len(y_test)))

print("Proportion of labels that are positive %.4f%%: " %
(100*sum(np.squeeze(y_test)==1)/len(y_test)))
print("Proportion of predictions that are positive %.4f%%: " %
(100*sum(np.squeeze(y_)==1)/len(y_test)))
```

```
Classification Report:

             precision    recall  f1-score   support

          0       0.00      0.00      0.00      6532
          1       0.56      1.00      0.72      8299

avg / total       0.31      0.56      0.40     14831

Accuracy: 55.9571%
Proportion of labels that are positive 55.9571%:
Proportion of predictions that are positive 100.0000%:
```

Trial 2

The results of trial 2 is :

Metrics	Value
Accuracy	55.9571%
Proportion of labels that are positive	55.9571%
Proportion of predictions that are positive	99.8989%

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=5583).fit(x_train,
np.squeeze(y_train))
y_ = model.predict(x_test)

print("Classification Report:\n\n", classification_report(np.squeeze(y_test),
y_))
print("Accuracy: %.4f%%" % (100*model.score(x_test, np.squeeze(y_test))))

print("Proportion of labels that are positive %.4f%%: " %
(100*sum(np.squeeze(y_test)==1)/len(y_test)))
print("Proportion of predictions that are positive %.4f%%: " %
(100*sum(y_==1)/len(y_test)))
```

```
Classification Report:

             precision    recall  f1-score   support

          0       0.60      0.00      0.00      6532
          1       0.56      1.00      0.72      8299

avg / total       0.58      0.56      0.40     14831

Accuracy: 55.9773%
Proportion of labels that are positive 55.9571%:
Proportion of predictions that are positive 99.8989%:
```

Problem 9

1. Shffule the data

As we analyzed in Problem 7, we know the order of data split make the training set and test set not 'independent and identically distributed'. So to address this problem, we firstly shuffle our dataset and then split then as the same ratio as before. This makes a significant difference in this two settings.

Metrics	Value
Accuracy	91.2750%
Proportion of labels that are positive	91.4369%
Proportion of predictions that are positive	99.6629%

```
x_train, y_train, x_test, y_test = split(x, y, 0.9, shuffle=True)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=5583).fit(x_train,
np.squeeze(y_train))
y_ = model.predict(x_test)

print("Classification Report:\n\n", classification_report(np.squeeze(y_test),
y_))
print("Accuracy: %.4f%%" % (100*model.score(x_test, np.squeeze(y_test))))

print("Proportion of labels that are positive %.4f%%: " %
(100*sum(np.squeeze(y_test)==1)/len(y_test)))
print("Proportion of predictions that are positive %.4f%%: " %
(100*sum(y_==1)/len(y_test)))
```

```
Classification Report:

             precision    recall  f1-score   support

          0       0.26      0.01      0.02      1270
          1       0.91      1.00      0.95     13561

avg / total       0.86      0.91      0.87     14831

Accuracy: 91.2750%
Proportion of labels that are positive 91.4369%:
Proportion of predictions that are positive 99.6629%:
```

2. Use Weighted Logsitic Model

After shuffling, the most difficulty of our model training might be the imbalance of positive and negative data, where about 91.43% data are 5-star ratings. So we employ the weighted strategy in `LogisticRegression` in `scikit-learn` and the performance will be:

Metrics	Value
Accuracy	74.0476%
Proportion of labels that are positive	91.4369%
Proportion of predictions that are positive	73.4408%

But the results become worse compared with only shuffling model.

```
x_train, y_train, x_test, y_test = split(x, y, 0.9, shuffle=True)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=5583,
class_weight="balanced").fit(x_train, np.squeeze(y_train))
y_ = model.predict(x_test)

print("Classification Report:\n\n", classification_report(np.squeeze(y_test),
y_))
print("Accuracy: %.4f%%" % (100*model.score(x_test, np.squeeze(y_test))))

print("Proportion of labels that are positive %.4f%%: " %
(100*sum(np.squeeze(y_test)==1)/len(y_test)))
print("Proportion of predictions that are positive %.4f%%: " %
(100*sum(y_==1)/len(y_test)))
```

```
Classification Report:

              precision    recall  f1-score   support

          0       0.17      0.54      0.26      1270
          1       0.95      0.76      0.84     13561

avg / total       0.88      0.74      0.79     14831

Accuracy: 74.0476%
Proportion of labels that are positive 91.4369%:
Proportion of predictions that are positive 73.4408%:
```