AML Homework 1

Yue (Billy) Liu (yl992)

Teammate: Chenran Ning (cn257)

PART I : House Price OLS Regression Problem

Question 1

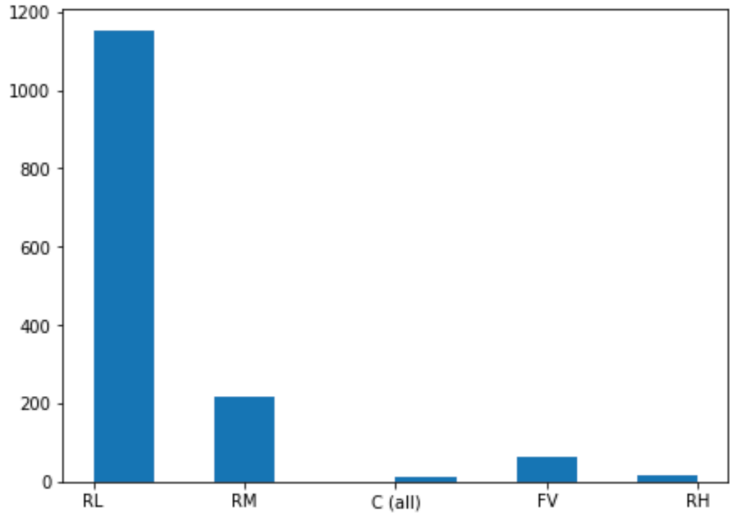
Join the House Prices - Advanced Regression Techniques competition on Kaggle. Download the training and test data.

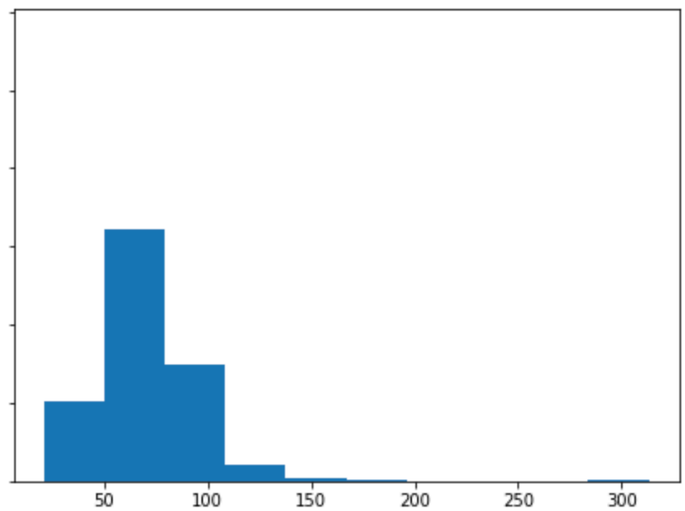
```
df = pd.read_csv('train.csv')
trainDF = df
```

Question 2

Give 3 examples of continuous and categorical features in the dataset; choose one feature of each type and plot the histogram to illustrate the distribution.

```
#Read df into a pandas dataframe
print(df.info())
#From the datatypes we can see:
#Categorical Features: MSZoning, SaleType, SaleCondition
#Continuous Features: LotFrontage, MasVnrArea, GarageYrBlt
fig, axs = plt.subplots(1,2,sharey=True,tight_layout=True)
fig.set_figheight(10), fig.set_figwidth(20)
#Plotting Categorical Features
axs[0].hist(df["MSZoning"])
#Plotting Continuous Features
axs[1].hist(df["LotFrontage"])
```





Question 3

Pre-process your data, explain your pre-processing steps, and the reasons why you need them.

```
#Lets divide the data into 3 types:categorical, categorical_simple, and continuous
#categorical: non-ordinal categorical variables, they will be one-hot encoded
#categorical_ordinal: ordinal categorical variables, they will be label encoded
#continuous: continuous variables, these will be converted into float
categorical = {"MSZoning","LotShape","LotConfig",
            "Neighborhood","BldgType","HouseStyle","RoofStyle","RoofMatl","Exterior1st",

"Exterior2nd","MasVnrType","ExterCond","Foundation","BsmtQual","BsmtCond","BsmtExposure",
            "BsmtFinType1","BsmtFinType2","Heating","Electrical",
            "GarageType","GarageFinish","GarageCond","PavedDrive","Fence","MiscFeature",
            "SaleType","SaleCondition"}
categorical_ordinal =
{"Street","Alley","Utilities","LandSlope","LandContour","CentralAir","PoolQC",

 "Condition1","Condition2","ExterQual","KitchenQual","HeatingQC","Functional","GarageQual",
             "FireplaceQu"}
continuous =
{"MSSubClass","LotFrontage","LotArea","OverallQual","OverallCond","YearBuilt","YearRemodAdd"
,"MasVnrArea","BsmtFinSF1",

"BsmtFinSF2","BsmtUnfSF","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLivArea","Bsm
tFullBath",

"BsmtHalfBath","FullBath","HalfBath","Bedroom","Kitchen","TotRmsAbvGrd","Fireplaces","Garage
YrBlt",

"GarageCars","GarageArea","WoodDeckSF","OpenPorchSF","EnclosedPorch","3SsnPorch","ScreenPorc
h","PoolArea",
```

```
            "MiscVal","MoSold","YrSold","BedroomAbvGr","KitchenAbvGr","SalePrice"}


#Function to find unique categories within a categorical feature
def find_Category(column):
    categories = set()
    for value in column:
        categories.add(value)
    return categories

#Function to preprocess all the data in the dataset in one run
def preprocessing(df,start,end):
    #creating a list of dataframe columns
    columns = list(df)[start:end]
    data_cleaned = [] #Array that contains cleaned data
    for column in columns:
        tmp = []
        #Takes care of non-ordinal categorical features
        if column in categorical:
            cats = find_Category(trainDF[column])
            cats = list(cats)+["NA"]
            for value in df[column]:
                tmp2 = [0]*len(cats)
                if value != value:
                    tmp2[len(cats)-1] = 1
                else:
                    tmp2[cats.index(value)]=1
                tmp.append(tmp2)
        #Takes care of ordinal categorical features
        elif column in categorical_ordinal:
            cats = find_Category(trainDF[column])
            cats = list(cats)+["NA"]
            for value in df[column]:
                if value != value:
                    tmp.append(len(cats)-1)
                else:
                    tmp.append(cats.index(value))

        #Normalizes numerical value
        elif column in continuous:
            for value in df[column]:
                #convert each value into float and replace any missing value with 0
                tmp.append(float(value) if value == value else 0)
        tmp = np.array(tmp)
        if tmp.ndim == 1:
            tmp = tmp.reshape(-1,1)
```

```
        if data_cleaned == []:
            data_cleaned = tmp
        else:
            data_cleaned = np.hstack((data_cleaned, tmp))#Append the cleaned column to new
dataframe
    return data_cleaned

data = preprocessing(df,1,-1)
```

Question 4

One-hot encoding

Question 4 is already answered in Question 3 as the pre-processing function one-hot encoded all the non-ordinal categorical value at once.

I used all of the features to train the model and the model performed quite well. I tried to cherry-pick features to improve the result but ended up being worse the using all of the features.

Question 5

Using ordinary least squares (OLS), try to predict house prices on this dataset. Choose the features (or combinations of features) you would like to use or ignore, provided you justify your choice. Evaluate your predictions on the training set using the MSE and the R2 score. For this question, you need to implement OLS from scratch without using any external libraries or packages.

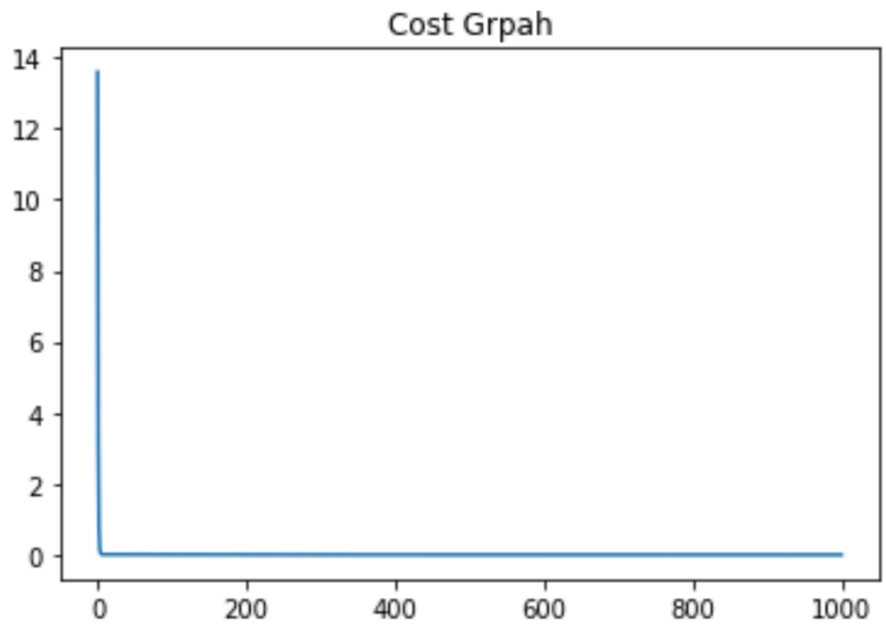
```
class OLS:
    def __init__(self, x, y, learning_rate, iteration):
        self.x = x
        self.y = y
        self.lr = learning_rate
        self.iteration = iteration
        self.theta = (1/(x.shape[1]+1))*np.ones((x.shape[1]+1,1))#Mean Initialization

    def fit(self):
        x = np.hstack([np.ones(len(self.x))[:, np.newaxis], self.x])#Account for y-
intersection
        n = self.y.shape[0]
        theta = self.theta
        cost_list = []
        for i in range(iteration):
            y_pred = np.dot(x, theta)
            #Calculating cost using the cost function
            cost = (1/(2*n))*np.sum(np.square(y_pred - y))
```

```
            #gradient decent
            d_theta = (1/n)*np.dot(x.T, y_pred - y)
            theta = theta - self.lr*d_theta
            cost_list.append(cost)

        self.theta = theta
        self.y_pred = np.dot(x, theta)

        plt.plot(cost_list)
        plt.title("Cost Grpah")
        plt.show()

        return theta, cost_list, self.y_pred

    def predict(self,data):
        x = np.hstack([np.ones(len(data))[:, np.newaxis], data])
        y_pred = np.dot(x, theta)
        return y_pred

    def evaluate(self):
        r2 = r2_score(self.y, self.y_pred)
        mse = mean_squared_error(self.y, self.y_pred)
        print("R2_Score: ", r2)
        print("Mean Squared Error: ",mse)
```

Question 6

Train your classifier using all of the training data, and test it using the testing data. Submit your results to Kaggle.

```
learning_rate = 5E-1
y = preprocessing(df,-1,9999)

data = normalize(data,axis=0)
y = np.log10(y)

model_OLS = OLS(data,y,learning_rate, iteration)
theta, cost_list, y_pred = model_OLS.fit()
OLS.evaluate(model_OLS)
```




```
R2_Score:  0.8274347137752431
Mean Squared Error:  0.0051898332994723 4
```

The loss drops very rapidly.

```
test_df = pd.read_csv('test.csv')

test_data = preprocessing(test_df,1,80)
test_data = normalize(test_data,axis=0)
predictions = model_OLS.predict(test_data)


with open("submission.csv","w") as f:
    writer = csv.writer(f)
    row = ["Id", "SalePrice"]
    writer.writerow(row)
    for i in range(len(predictions)):
        row = [i+1461, 10**predictions[i][0]]
        writer.writerow(row)
```

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	4 days ago	1 seconds	0 seconds	0.18297

Complete

Jump to your position on the leaderboard

Part II: Titanic Logistic Regression Classification Problem

Question 1

Join the Titanic: Machine Learning From Disaster competition on Kaggle. Download and pre- process the data.

```
train = pd.read_csv('titanic/train.csv')
test = pd.read_csv('titanic/test.csv')

#Data-Preprocessing
#Did some tweaking and reading discussions on Kaggle, these columns appear to be quite
useless, let's disgard them
train.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin','Embarked'], axis=1,
inplace=True)
test.drop(columns= ['PassengerId', 'Name', 'Ticket', 'Cabin','Embarked'], axis=1, inplace=
True)

#Upon investigaing the data, there are a lot of null values, let's fill them with median
train['Age'].fillna(train['Age'].median(), inplace=True)
test['Age'].fillna(test['Age'].median(), inplace=True)
test['Fare'].fillna(test['Fare'].median(), inplace=True)
#Label-encode sex column
def label_encode(data):
    tmp = []
    for line in data:
        if line == 'male':
            tmp.append(0)
        elif line == 'female':
            tmp.append(1)
    return tmp

x_train= train.iloc[:, 1:]
y_train= train['Survived'].values.reshape(-1,1).astype(int)

label_List1 = label_encode(x_train['Sex'])
label_List2 = label_encode(test['Sex'])
x_train.drop('Sex', axis = 1, inplace = True)
x_train['Sex'] = label_List1
test.drop('Sex', axis = 1, inplace = True)
test['Sex'] = label_List2

#Feature Normalization
features= ['Age', 'SibSp', 'Fare']

x_train[features] = normalize(x_train[features])
test[features] = normalize(test[features])
```

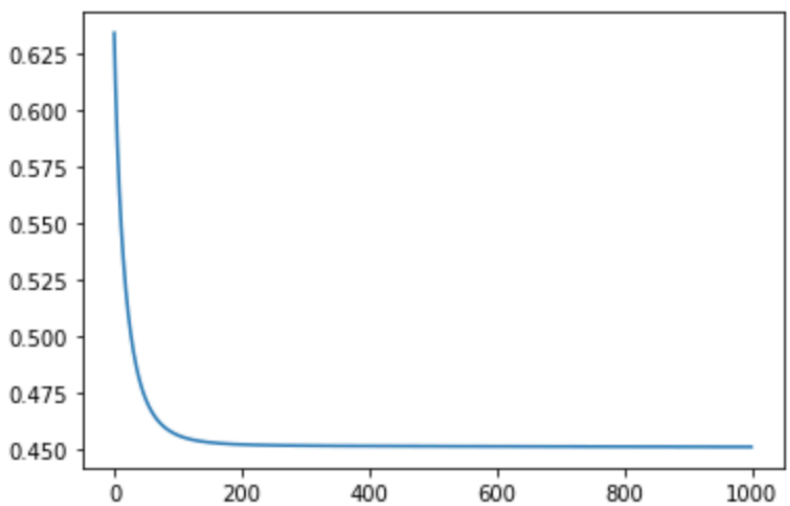
Question 2

Using logistic regression, try to predict whether a passenger survived the disaster. Choose the features (or combinations of features) you would like to use or ignore, provided you justify your choice.

```
class Logistic:

    def __init__(self, x, y, learning_rate, iteration):
        self.x = x
        self.y = y
        self.lr = learning_rate
        self.iteration = iteration

    def sigmoid (self,z):
        return 1/(1 + e**(-z))

    def cost_function(self, x, y, weights):
        z = np.dot(x, weights)
        predict = y * np.log(self.sigmoid(z)) + (1-y) * np.log(1-self.sigmoid(z))
        return -np.sum(predict) / len(self.x)

    def fit(self):
        # X: N*Feature
        #Weight: Feature*2
        cost = []
        weights = np.zeros((self.x.shape[1],1))
        N = len(self.x)

        for i in range(iteration):

            y_hat = self.sigmoid(np.dot(self.x, weights))
            y_true = self.y
            weights -= learning_rate * np.dot(self.x.T,  (y_hat - y_true)) / N
            cost.append(self.cost_function(self.x, self.y, weights))

        plt.plot(cost)
        self.weights = weights
        y_pred = self.sigmoid(np.dot(self.x, weights))
        return weights, cost

    def predict(self,data):
        # Predicting with sigmoid function
        z = np.dot(data, self.weights)
        result = [int(i>0.5) for i in self.sigmoid(z)]

        return np.array(result)
```

```
    def evaluate(self):
        y_pred = self.predict(self.x)
        y = self.y.reshape(-1,)
        f1 = f1_score(self.y, y_pred )
        acc = np.sum(y==y_pred)/self.y.shape[0]
        print("Accuracy: ",acc)
        print("F1-Score: ",f1)

iteration = 1000
learning_rate = 0.5
model_Logistic = Logistic(x_train,y_train,learning_rate, iteration)
weights, cost_list = model_Logistic.fit()

model_Logistic.evaluate()
```

```
Accuracy:  0.7845117845117845
F1-Score:  0.710843373493976
```



Question 3

Train your classifier using all of the training data, and test it using the testing data. Submit your results to Kaggle.

```
result = model_Logistic.predict(test)
with open("gender_submission.csv","w") as f:
    writer = csv.writer(f)
    row = ["PassengerId", "Survived"]
    writer.writerow(row)
    for i in range(len(test)):
        row = [i+892, result[i]]
        writer.writerow(row)
```

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
gender_submission.csv	a day ago	1 seconds	0 seconds	0.74641

Complete

Jump to your position on the leaderboard

Part III: Written Exercise

Question 1

$$
\begin{aligned}
&\arg\min_\theta \mathbb{E}_{\hat{p}(x)}[KL(\hat{p}(y|x)||p_\theta(y|x))] \\
&= \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)}[log\hat{p}(y|x) - logp_\theta(y|x)]] \\
&= \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)}[log\hat{p}(y|x)]] - \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)}[logp_\theta(y|x)]] \\
&= \arg\min_\theta -\mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)}[logp_\theta(y|x)]] \\
&= \arg\max_\theta \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)}[logp_\theta(y|x)]] \\
&= \arg\max_\theta \mathbb{E}_{\hat{p}(x)\hat{p}(y|x)}[logp_\theta(y|x)] \\
&= \arg\max_\theta \mathbb{E}_{\hat{p}(x,y)}[logp_\theta(y|x)]
\end{aligned}
$$

Question 2

(a)

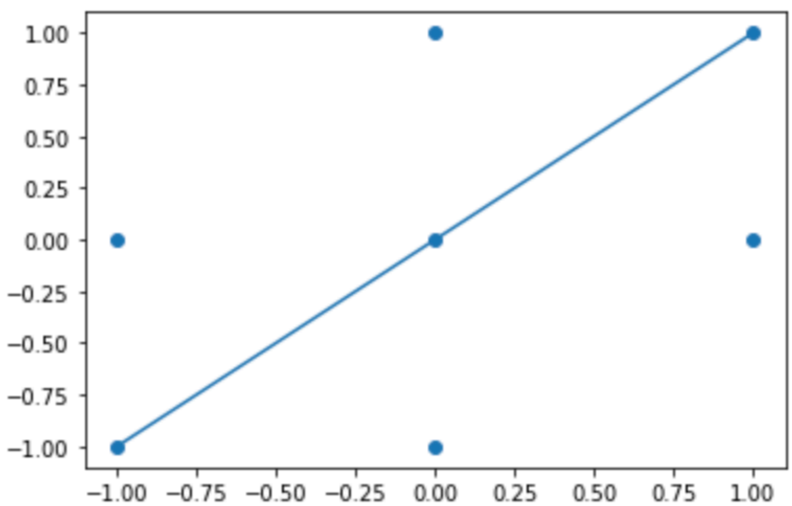
$$
\frac{\mathrm{d}\sigma(a)}{\mathrm{d}a} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}} \cdot \frac{1+e^{-x}-1}{1+e^{-x}} = \sigma(a)\cdot(1-a)
$$

(b)

$$\begin{aligned}
\ell(\theta) &= ylog\sigma(\theta^T x) + (1-y)log(1-\sigma(\theta^T x)) \\
\nabla\ell(\theta) &= \frac{\mathrm{d}\ell(\theta)}{\mathrm{d}\theta} \\
&= \frac{\mathrm{d}[ylog\sigma(\theta^T x) + (1-y)log(1-\sigma(\theta^T x))]}{\mathrm{d}\theta} \\
&= \frac{\mathrm{d}[ylog\sigma(\theta^T x)]}{\mathrm{d}log\sigma(\theta^T x)} \cdot \frac{\mathrm{d}log\sigma(\theta^T x)}{\mathrm{d}\theta} + \frac{\mathrm{d}[(1-y)log(1-\sigma(\theta^T x))]}{\mathrm{d}[log(1-\sigma(\theta^T x))]} \cdot \frac{d[log(1-\sigma(\theta^T x))]}{\mathrm{d}\theta} \\
&= y \cdot \frac{dlog\sigma(\theta^T x)}{\mathrm{d}\theta} + (1-y) \cdot \frac{\mathrm{d}[log(1-\sigma(\theta^T x))]}{\mathrm{d}\theta} \\
&= y \cdot \frac{1}{\sigma(\theta^T x)} \cdot \sigma(\theta^T x) \cdot (1-\sigma(\theta^T x)) \cdot x + (1-y) \cdot \frac{1}{1-\sigma(\theta^T x)} \cdot (1-\sigma(\theta^T x)) \cdot \sigma(\theta^T x) \cdot x \cdot (-1) \\
&= y \cdot (1-\sigma(\theta^T x)) \cdot x - (1-y) \cdot \sigma(\theta^T x) \cdot x \\
&= (y-\sigma(\theta^T x)) \cdot x
\end{aligned}$$

Question 3

(a)



The best fit line is y = x, where the slope is 1 and intercept is 0.

(b)

$$We\ have:\ \hat{\beta}=(X^TX)^{-1}X^TY$$

$$X=\begin{bmatrix}-1 & 1\\ -1 & 1\\ 0 & 1\\ 0 & 1\\ 0 & 1\\ 1 & 1\\ 1 & 1\end{bmatrix},\ Y=\begin{bmatrix}-1\\ 0\\ -1\\ 0\\ 1\\ 0\\ 1\end{bmatrix}$$

$$\hat{\beta}=\begin{bmatrix}4 & 0\\ 0 & 7\end{bmatrix}^{-1}\cdot\begin{bmatrix}-1 & -1 & 0 & 0 & 0 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}\cdot\begin{bmatrix}-1\\ 0\\ -1\\ 0\\ 1\\ 0\\ 1\end{bmatrix}=\begin{bmatrix}\frac{1}{2}\\ 0\end{bmatrix}$$

Therefore,the best fit line is y = 0.5x, where the slope is 0.5 and intercept is 0.

The best fit line has to go through the origin to minimize the MSE of the three points on the y-axis. To minimize the MSE of the rest 4 points, the line has to go through the middle between (1,1) and (1,0), and between (-1,-1) and (-1,0).

(c)

$$MAE=\frac{\sum_{i=1}^{N}|y_i-x_i|}{N}$$

As long as the line goes through (0,0) and bounded by (1,1) and (1,0), the mean absolute erro is the same, therefore, the best fit line is:

$$y=k\cdot x,\ k\in[0,1]$$

Thanks for grading!