CS5785 Applied Machine Learning Homework 1

Yixuan (Rylee) Li yl2557

September 2021

1. *CODING EXERCISES*

Part I. The Housing Prices

Question 1 Load Dataset

I downloaded the data set and successfully imported it by implementing following code:

```python
# load the Housing Prices dataset
df_test  = pd.read_csv("test.csv");
df_train = pd.read_csv("train.csv");
```

Question 2 Continuous and Categorical Features

3 examples of continuous features: LotArea, GrLivArea, TotalBsmtSF
3 examples of categorical features: LotShape, BldgType, MasVnrType

- **Histogram of TotalBsmtSF - Continuous**

```python
# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(df_train['TotalBsmtSF'])
# Adding extra features
plt.xlabel("TotalBsmtSF: Total square feet of basement area")
plt.ylabel("Number of Houses - train set")
plt.savefig("hist_cont_TotalBsmtSF") # save pics
```

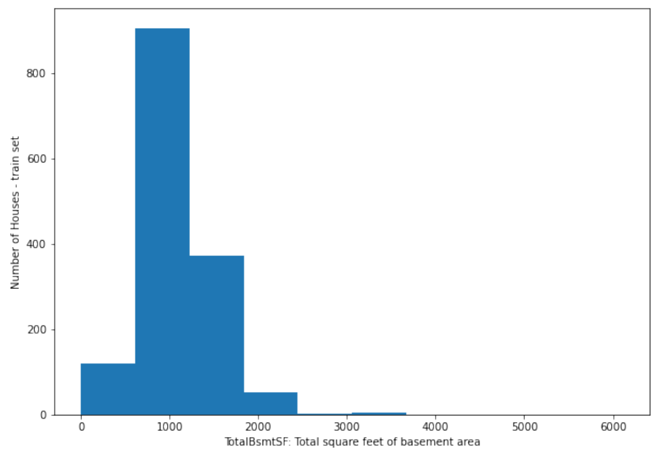


Figure 1.1: Histogram of TotalBsmtSF - Continuous

- **Histogram of LotShape - Categorical**

```
# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(df_train['LotShape'])
# Adding extra features
plt.xlabel("LotShape: General shape of property")
plt.ylabel("Number of Houses - train set")
plt.savefig("hist_cat_LotShape") # save pics
```

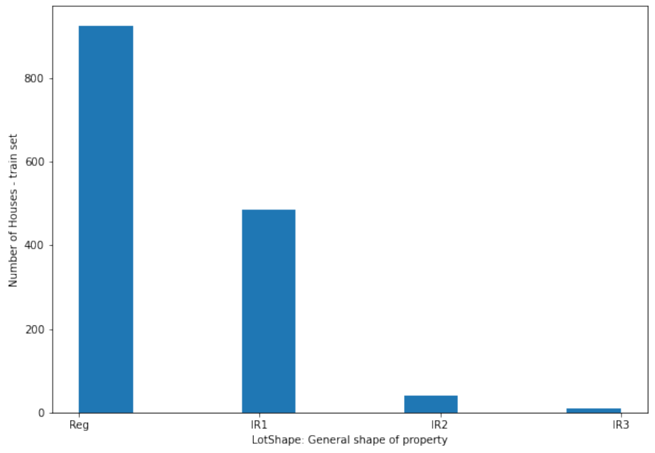


Figure 1.2: Histogram of LotShape - Categorical

Question 3,4 Preprocess Data

In this section, steps of pre-processing included preparing the datasets, dealing with missing values, normalizing numerical values, dealing with categorical values.

- **Prepare the Datasets**

To pre-process data, I concatenated train and test to facilitate the process. In this way, I could deal with both datasets at the same time.

```
# concat train and test for data processing
df = pd.concat([df_train, df_test], ignore_index = True);
# drop the target variable saleprice in place
target = df_train['SalePrice']
```

- **Normalize Data**

Let's inspect the distribution of SalePrice first.

```
# draw the distribution of salesprice
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(target, bins=100)
# Adding extra features
plt.xlabel("SalePrice")
plt.ylabel("Distribution")
plt.savefig("dist_sale_price") # save pics
```

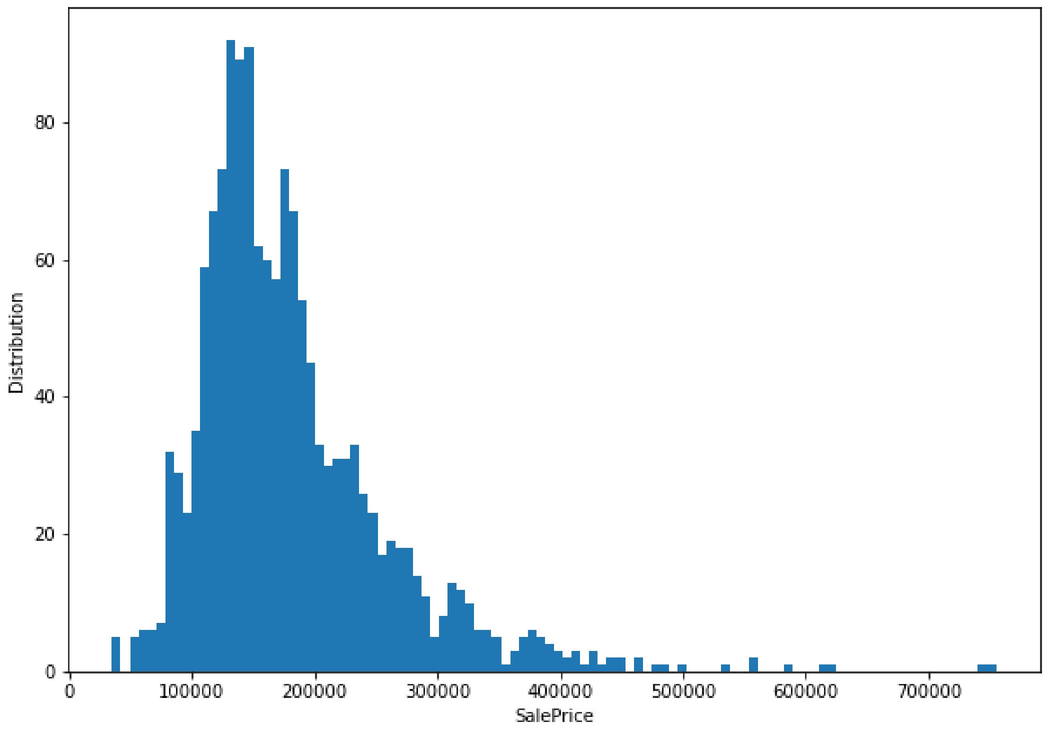


Figure 1.3: Distribution of Sale Price

From the Fig1.3, We can see that the distribution is skewed. I took the log-transform of SalePrice to remove this skewness.

```
# get the log of sale price
y_train  = np.log1p(df_train["SalePrice"])
df.drop(['SalePrice'], axis = 1, inplace = True)
```

- **Categorize data**

Different methods of pre-processing would be applied for categorical data and numerical data. So I classified data into these two types first for future processing.

```
# get all the categorical columns
cat_col = df.select_dtypes(include=['object']).columns.tolist()
# dataframe with categorical features
data_cat = df[cat_col]
# dataframe with numerical features
data_num = df.drop(cat_col, axis = 1)
```

- **Deal with missing value in Categorical Columns**

I figured out the number of missing value for each column, checked out the columns that had missing values and sorted the list in descending order.

```
# figure out the number of missing value for each column
missing_values = data_cat.isnull().sum()
# check out the columns that have missing values and sort the list in descending order
missing_values = missing_values[missing_values > 0].sort_values(ascending = False)
```

After going through the data description file, I found that some of the missing values indicate that there's no such facilities (eg. BsmtCond) but some may imply the data has not been recorded. I handled these two situations in different ways. For the first siutation, I replaced missing values with NA (other/none for some columns like MasVnrType, Exterior1st and Exterior2nd). For the later siutation, I replaced missing values with the mode of that column. I replaced missing values in Column MSZoning utilizing data in the Neighborhood Column since these two features are highly correlated.

1. None: MasVnrType
2. Other: Exterior2nd
3. NA: BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, GarageType, GarageCond, GarageFinish, GarageQual, Alley, FireplaceQu, PoolQC, Fence, MiscFeature
4. Mode: Utilities, Exterior1st, Electrical, KitchenQual, Functional, SaleType
5. Other Method: MSZoning

```
# fill missing values according to the schema described above
# replace missing values with NA
df.MasVnrType.fillna('None', inplace = True)
df.Exterior2nd.fillna('Other', inplace = True)
df.BsmtQual.fillna('NA', inplace = True)
df.BsmtCond.fillna('NA', inplace = True)
df.BsmtExposure.fillna('NA', inplace = True)
df.BsmtFinType2.fillna('NA', inplace = True)
df.BsmtFinType1.fillna('NA', inplace = True)
df.GarageType.fillna('NA', inplace = True)
df.GarageCond.fillna('NA', inplace = True)
df.GarageFinish.fillna('NA', inplace = True)
df.GarageQual.fillna('NA', inplace = True)
df.Alley.fillna('NA', inplace = True)
df.FireplaceQu.fillna('NA', inplace = True)
df.PoolQC.fillna('NA', inplace = True)
df.Fence.fillna('NA', inplace = True)
df.MiscFeature.fillna('NA', inplace = True)

# replace missing values with the mode of that column
df.Utilities.fillna(df.Utilities.mode()[0], inplace = True)
df.Exterior1st.fillna(df.Exterior1st.mode()[0], inplace = True)
df.Electrical.fillna(df.Electrical.mode()[0], inplace = True)
df.KitchenQual.fillna(df.KitchenQual.mode()[0], inplace = True)
df.Functional.fillna(df.Functional.mode()[0], inplace = True)
df.SaleType.fillna(df.SaleType.mode()[0], inplace = True)

# replace missing values in Column MSZoning utilizing data in the Neighborhood Column
for neighbor in df.Neighborhood.unique():
    if df.MSZoning[df.Neighborhood == neighbor].isnull().values.any():
        # fill in missing values in Column MSZoning
        # using the mode of those houses that are in the same neighborhood
        df.loc[df.Neighborhood == neighbor,'MSZoning'] = \
        df.loc[df.Neighborhood == neighbor,'MSZoning'].fillna(
            df.loc[df.Neighborhood == neighbor,'MSZoning'].mode()[0])

# check if there are any missing values
df[cat_col].isnull().values.any()
```

- **Deal with missing value in Numerical Columns**

I figured out the number of missing value for each column, checked out the columns that have missing values and sorted the list in descending order.

```
# figure out the number of missing value for each column
missing_values = data_num.isnull().sum()
# check out the columns that have missing values and sort the list in descending order
missing_values = missing_values[missing_values > 0].sort_values(ascending = False)
```

After going through the data description file, I thought most missing values in the numerical category were not resulted from lack of record. So I did not drop any numerical column at this point. And I found that most missing values indicated that there's no such facilities (eg. MasVnrArea) but few may imply the data had not been recorded. I handled these two situations in different ways. For the first situation, I replaced missing values with zero. For the later situation, I replaced missing values with the median of that column. For GarageYrBlt, I replaced missing values of the year garage was built with the year the house was built.

1. Median: LotFrontage
2. Zero: MasVnrArea, BsmtFullBath, BsmtHalfBath, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, GarageCars, GarageArea
3. Other Method: GarageYrBlt

```
# replace missing values with Median
df.LotFrontage.fillna(df.LotFrontage.median(), inplace=True)

# replace missing values with Zero
df.MasVnrArea.fillna(0, inplace = True)
df.BsmtHalfBath.fillna(0, inplace = True)
df.BsmtFullBath.fillna(0, inplace = True)
df.GarageArea.fillna(0, inplace = True)
df.GarageCars.fillna(0, inplace = True)
df.TotalBsmtSF.fillna(0, inplace = True)
df.BsmtUnfSF.fillna(0, inplace = True)
df.BsmtFinSF2.fillna(0, inplace = True)
df.BsmtFinSF1.fillna(0, inplace = True)

# replace missing values of the year garage was built with the year the house was built
df.GarageYrBlt.fillna(df.YearBuilt, inplace = True)

# check if there are any missing values
df.isnull().values.any()
```

- **Deal with categorical values**

First, I conducted integer encoding to deal with categorical variables that had order or relationship. Then I conducted OHE for those barley have any order or relationship. Some examples of features that I think should use a one-hot encoding are MSZoning, BldgType, MasVnrType, etc.. This is because these features don't have any ranking for category values.

```
# STEP 1: Integer encoding
# dealing with categorical variables that have order or relationship
df.LotShape = df.LotShape.map({'IR3':0, 'IR2':1, 'IR1':2, 'Reg':3})
df.LandSlope = df.LandSlope.map({'Sev':1, 'Mod':2, 'Gtl':3})
df.ExterQual = df.ExterQual.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.ExterCond = df.ExterCond.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.BsmtQual = df.BsmtQual.map({'NA':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.BsmtCond =  df.BsmtCond.map({'NA':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.BsmtExposure = df.BsmtExposure.map({'NA':0, 'No':1, 'Mn':2, 'Av':3, 'Gd':4})
df.BsmtFinType1 = df.BsmtFinType1.map({'NA':0, 'Unf':1, 'LwQ':2, 'Rec':3,
    'BLQ':4, 'ALQ':5, 'GLQ':6})
df.BsmtFinType2 = df.BsmtFinType2.map({'NA':0, 'Unf':1, 'LwQ':2, 'Rec':3, 'BLQ':4,
    'ALQ':5, 'GLQ':6})
df.HeatingQC = df.HeatingQC.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.KitchenQual = df.KitchenQual.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.Functional = df.Functional.map({'Sal':1, 'Sev':2, 'Maj2':3, 'Maj1':4,
    'Mod':5, 'Min2':6, 'Min1':7, 'Typ':8})
df.FireplaceQu = df.FireplaceQu.map({'NA':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.GarageCond = df.GarageCond.map({'NA':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.GarageQual = df.GarageQual.map({'NA':0, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})
df.GarageFinish = df.GarageFinish.map({'NA':0, 'Unf':1, 'RFn':2, 'Fin':3})
df.PavedDrive = df.PavedDrive.map({'N':1, 'P':2, 'Y':3})
df.PoolQC = df.PoolQC.map({'NA':0, 'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

# STEP 2: use a one-hot encoding (OHE) to map categorical data to integers
df = pd.get_dummies(df,columns=['MSZoning', 'Street', 'Alley', 'RoofStyle',
            'LandContour', 'Utilities',
            'LotConfig', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
            'HouseStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
            'Foundation', 'Heating', 'CentralAir', 'Electrical', 'GarageType',
            'Fence', 'MiscFeature', 'SaleType', 'SaleCondition'],drop_first=True)
```

I visualized the results of OHE by plotting feature histograms of the original feature and its new one-hot encoding (Fig 1.4, Fig 1.5).

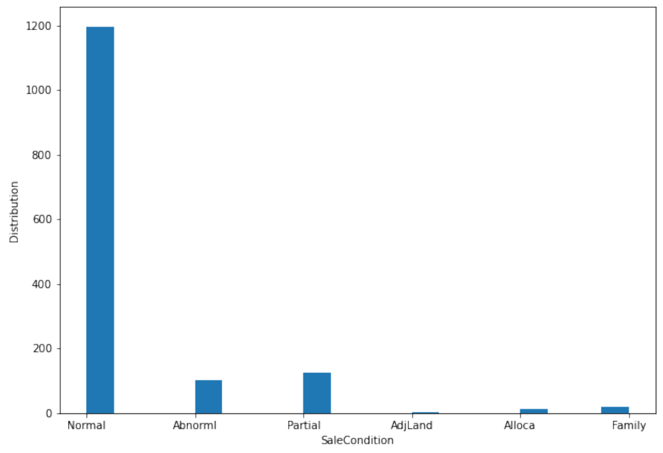


Figure 1.4: Histogram of Sale Condition (Original Feature)

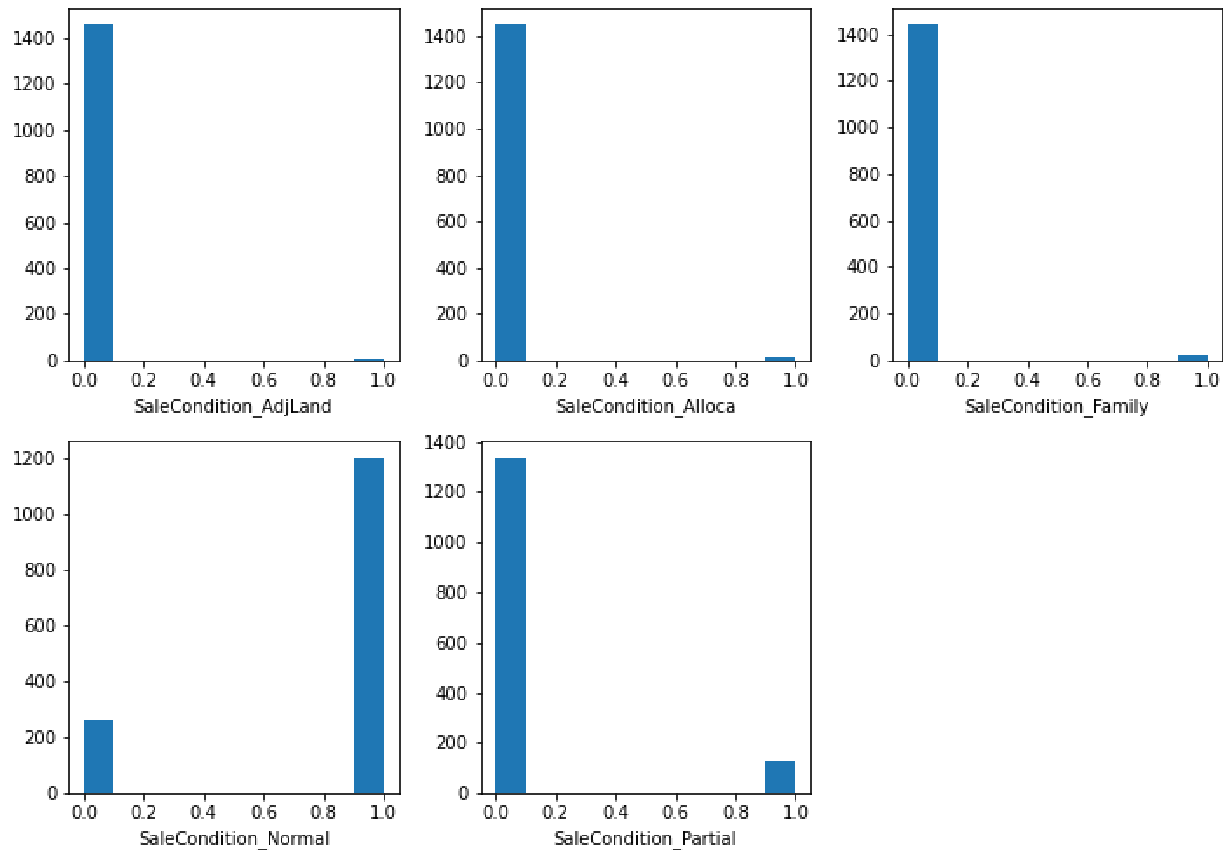


Figure 1.5: Histogram of Sale Condition Features (After OHE)

- **Organize Data**

Splited the data frame into train and test. Got Xtrain, ytrain, and Xtest.

```
# get train set and test set
df_train = df.iloc[:len(df_train)]
df_train = df_train.join(target)
df_test = df.iloc[len(df_train):]
# Get Xtrain, ytrain, and Xtest.
X_train  = df_train.drop(['SalePrice','Id'], axis=1)
# remove duplicate columns
X_train = X_train.loc[:,~X_train.columns.duplicated()]
```

Question 5 Feature Selection

After one-hot encoding, there were more than 200 features. In order to avoid over fitting, feature selection would be implemented. Here, I used Lasso Regression to filter out the least important features (with coef = 0) and keep the important ones. The process of feature selection kept 96 features and dropped the other 109 features.

```
#Define RMSE
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y_train,
        scoring="neg_mean_squared_error", cv = 5))
    return(rmse)

#Lasso Regression
model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y_train)

#Implemented feature selection
coef = pd.Series(model_lasso.coef_, index = X_train.columns)
```

Dropped features that were filtered out by LASSO Regression.

```
# drop features
droped_features = [];
for i in range(len(coef)):
    if coef[i] == 0:
        droped_features.append(X_train.columns[i])
X_train.drop(droped_features, axis = 1, inplace=True)
```

Question 5 Modeling and Predictions

- **OLS From Scratch**

I implemented OLS from scratch for modeling. Our goal here is to obtain θ^* that minimizes the normal equations $(X^TX)\theta = X^Ty$. The value θ^* is given by: $\theta^* = (X^TX)^{-1}X^Ty$. The application of normal equations can be found here:

```
# Use ordinary least squares (OLS) for Modeling (build from scratch)
mat = pd.DataFrame(np.linalg.pinv(np.matrix(X_train.T.dot(X_train))))
theta_best = pd.DataFrame(mat.values.dot(X_train.T).dot(y_train))
arr_d = (np.asarray(theta_best)).flatten()
```

- **Evaluate Predictions on Training set**

I generated predictions on the training set and evaluated predictions on the training set using the MSE and the R^2 score. The result is: MSE: 0.013690668658049349, R^2: 0.9141384048093214. Both MSE and R^2 tell us that the model works well for the training set. The mean square error is close to 0, implying that the regression line is close to the set of data points from the training set. The high R^2 indicates a good fit for the model. An R^2 of 91.41 percent reveals that 91.41 percent of the data fit the regression model.

```
# generate predictions on the training set
y_train_pred = X_train.dot(arr_d)
# Evaluate predictions on the training set using the MSE and the R^2 score
print('MSE:', metrics.mean_squared_error(y_train, y_train_pred))
print('R^2:', metrics.r2_score(y_train, y_train_pred))
```

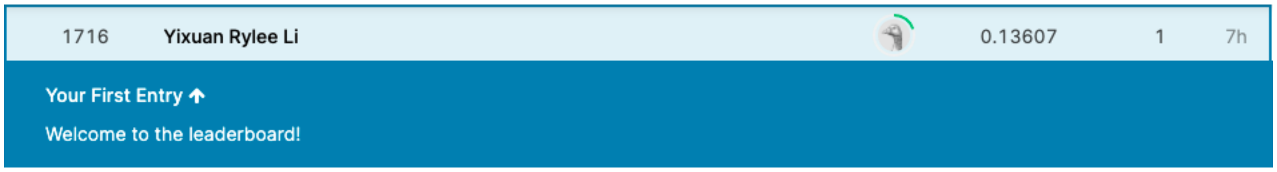
- **Predictions on Testing set**

I used the regression model to predict the sale price with the test set and took the exp of the sale price since the log of sale price were used for modeling.

```
# generate predictions
y_test_pred = X_test.dot(arr_d)
# take the exp of the sale price
y_test_pred_exp = np.exp(y_test_pred)
submission_predictions = pd.Series(y_test_pred_exp).array
# save data
```

```
res=pd.DataFrame(columns = ['Id', 'SalePrice'])
res['Id'] = df_test.index + 1
res['SalePrice'] = submission_predictions
res.to_csv('submission1.csv',index=False)
```

Question 6 Kaggle Results



```
1716    Yixuan Rylee Li    0.13607    1    7h
Your First Entry ↑
Welcome to the leaderboard!
```

Figure 1.6: Kaggle Submission - Housing Prices

Part II. Titanic

Question 1 Load Dataset

I downloaded the data set and successfully imported it by implementing following code:

```
# load the Housing Prices dataset
df_test  = pd.read_csv("test.csv");
df_train = pd.read_csv("train.csv");
```

Question 1 Preprocess Data

- **Prepare datasets**

To pre-process data, I concatenated train and test to facilitate the process. In this way, I could deal with both datasets at the same time.

```
# concat train and test for data processing
df = pd.concat([df_train, df_test], ignore_index = True);
# drop the target variable Survived in place
target = df_train['Survived']
df.drop(['Survived'], axis = 1, inplace = True)
```

- **Categorize data**

Different methods of pre-processing would be applied for categorical data and numerical data. So I classified data into these two types first for future processing.

```
# get all the categorical columns
cat_col = df.select_dtypes(include=['object']).columns.tolist()
# dataframe with categorical features
data_cat = df[cat_col]
# dataframe with numerical features
data_num = df.drop(cat_col, axis = 1)
```

- **Deal with missing value in Categorical Columns**

I figured out the number of missing value for each column, checked out the columns that had missing values and sorted the list in descending order.

```
# figure out the number of missing value for each column
missing_values = data_cat.isnull().sum()
# check out the columns that have missing values and sort the list in descending order
missing_values = missing_values[missing_values > 0].sort_values(ascending = False)
```

Missing values appeared in the following two categorical columns: cabin and embarked. After going through the data description, I replaced missing values in Cabin with NA, implying lack of record. And I replaced missing values in Embarked with the mode of that column.

```
# replace missing values using the above schema
df.Cabin.fillna('NA', inplace = True)
df.Embarked.fillna(df.Embarked.mode()[0], inplace = True)
# check if there are any missing values
df[cat_col].isnull().values.any()
```

- **Deal with missing value in Numerical Columns**

I figured out the number of missing value for each column, checked out the columns that had missing values and sorted the list in descending order.

```
# figure out the number of missing value for each column
missing_values = data_num.isnull().sum()
# check out the columns that have missing values and sort the list in descending order
missing_values = missing_values[missing_values > 0].sort_values(ascending = False)
```

Missing values appeared in the following two numerical columns: age and fare. After going through the data description, I replaced missing values in Age and Fare with the mean of that column.

```
# replace missing values with mean
df.Age.fillna(df.Age.mean(), inplace=True)
df.Fare.fillna(df.Fare.mean(), inplace=True)
# check if there are any missing values
df.isnull().values.any()
```

- **Deal with categorical values**

First, I'll conduct integer encoding to deal with categorical variables that have order or relationship. Then I'll conduct OHE for those barley have any order or relationship.

```
# STEP 1: Integer encoding
# dealing with categorical variables that have order or relationship
df.Sex = df.Sex.map({'female':0, 'male':1})
# STEP 2: use a one-hot encoding (OHE) to map categorical data to integers
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)
print("The transform data using get_dummies")
```

- **Prepare the Dataset**

Splited the dataframe to dftrain and dftest.

```
# get train set and test set
df_train = df.iloc[:len(df_train)]
df_train = df_train.join(target)
df_test = df.iloc[len(df_train):]
```

Question 2 Feature Selection

First, I dropped PassengerId which obviously didn't relate to the chance of survival. I conducted feature selection by looking into each feature's correlation to our target variable (Survived). Both the heat map and the correlation coefficient demonstrated how each kept feature correlates with the variable Survived. I dropped those features that had little correlations.

```
# drop PassengerId which obviously doesn't relate to the chance of survival
df_train.drop(['PassengerId'], axis=1, inplace=True)
#correlation matrix
corrmat = df_train.corr()
#Plot the heatmap
fig, ax = plt.subplots(figsize=(30, 19))
sns.set(font_scale=1)
sns.heatmap(corrmat, square=True,cmap='coolwarm');
# get the correltion coefficients and sort them by values
correlations = corrmat["Survived"].sort_values(ascending=False)
```

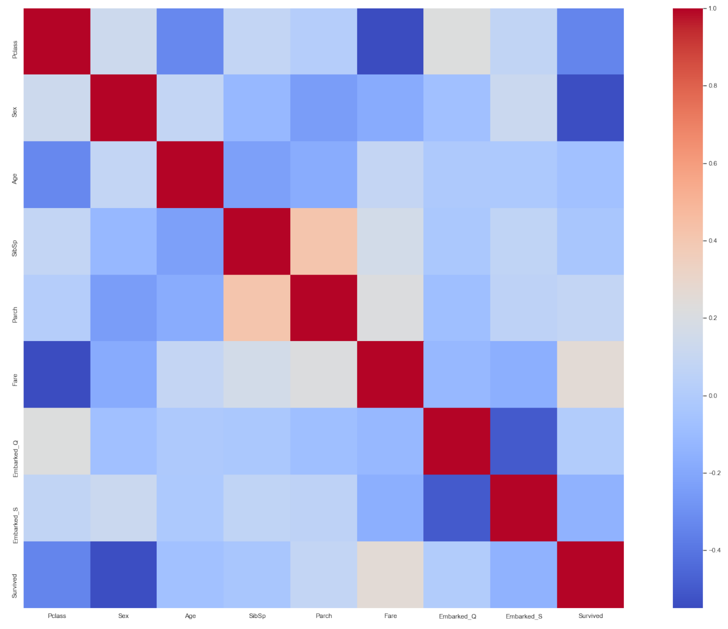
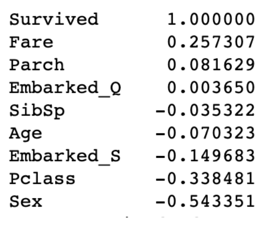


Figure 1.7: Heat Map Showing Correlations between Kept Features and Target Variables



```
Survived       1.000000
Fare           0.257307
Parch          0.081629
Embarked_Q     0.003650
SibSp         -0.035322
Age           -0.070323
Embarked_S    -0.149683
Pclass        -0.338481
Sex           -0.543351
```

Figure 1.8: Correlation Table (Features Kept)

Dropped features that provided too little information and correlated too weak (ie.'Ticket', 'Name', 'Cabin') according to the correlations illustrated above.

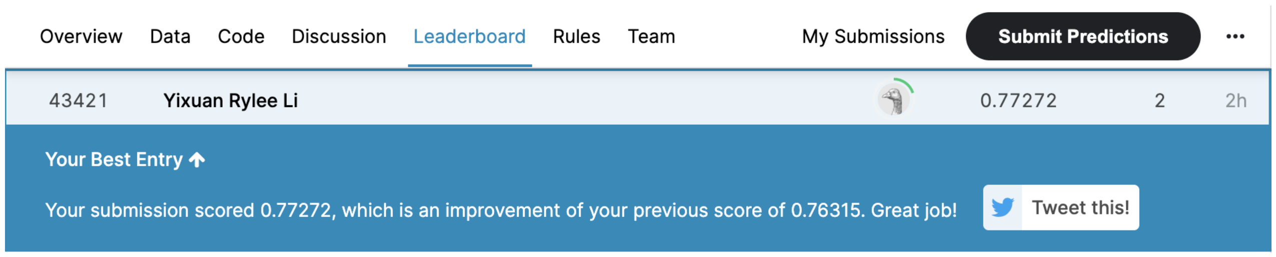
```
# drop features
X_train.drop(['Ticket', 'Name', 'Cabin'], axis = 1, inplace=True)
X_test.drop(['Ticket', 'Name', 'Cabin'], axis=1, inplace=True)
```

Question 3 Build Model and Make Predictions

I built model using logistic regression and made predictions using the test set.

```
# Build model
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
# Predict using the test set
submission_predictions = clf.predict(X_test)
# save predictions
res=pd.DataFrame(columns = ['PassengerId', 'Survived'])
res['PassengerId'] = df_test.index + 1
res['Survived'] = submission_predictions
res.to_csv('my_submission.csv',index=False)
```

Question 3 Kaggle Results



43421 Yixuan Rylee Li 0.77272 2 2h

Your Best Entry

Your submission scored 0.77272, which is an improvement of your previous score of 0.76315. Great job!

Figure 1.9: Kaggle Submission - Titanic

2. *WRITTEN EXERCISES*

Question 1: Maximum Likelihood and KL Divergence.

$$RHS = \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[KL(\hat{p}(y|x))||p_\theta(y|x)] = \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)}[\log \hat{p}(y|x) - \log p_\theta(y|x)]] \tag{2.1}$$

The first part can be dropped since it doesn't relate to θ. Then we have:

$$RHS = \arg\min_\theta \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)} - \log p_\theta(y|x)]] \tag{2.2}$$

$$= \arg\max_\theta \mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x)]] \tag{2.3}$$

And we have:

$$\mathbb{E}_{\hat{p}(x)}[\mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x)]] = \sum_x \sum_y \hat{p}(x)\hat{p}(y|x) \log p_\theta(y|x) \tag{2.4}$$

$$= \sum_{x,y} \hat{p}(x,y) \log p_\theta(y|x) = \mathbb{E}_{\hat{p}(x,y)} \log p_\theta(y|x) \tag{2.5}$$

So equation (2.3) becomes:

$$\arg\max_\theta \mathbb{E}_{\hat{p}(x,y)}[\log p_\theta(y|x)] = LHS \tag{2.6}$$

Question 2: Gradient and log-likelihood for logistic regression

a. Derivative of Sigmoid Function

$$\frac{d\sigma(a)}{da} = \frac{d}{da}\left[\frac{1}{1+e^{-a}}\right] \tag{2.7}$$

$$= \frac{\frac{d}{da}[e^{-a}+1]}{(e^{-a}+1)^2} \tag{2.8}$$

$$= \frac{e^{-a} \cdot \frac{d}{da}[-a]}{(e^{-a}+1)^2} \tag{2.9}$$

$$= \frac{e^{-a} \cdot 1}{(e^{-a}+1)^2} \tag{2.10}$$

$$= \frac{1}{1+e^{-a}} \cdot \left(\frac{e^{-a}}{1+e^{-a}}\right) \tag{2.11}$$

$$= \frac{1}{1+e^{-a}} \cdot \left(1 - \frac{1}{1+e^{-a}}\right) \tag{2.12}$$

$$= \sigma(a) \cdot (1 - \sigma(a)) \tag{2.13}$$

b. Gradient of the Log Likelihood

$$\frac{d\ell(\theta)}{d\theta} = \frac{d}{d\theta} y \log \sigma(\theta^T x) + \frac{d}{d\theta}(1-y)\log(1-\sigma(\theta^T x)) \quad (2.14)$$

$$= [\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)}]\frac{d}{d\theta}\sigma(\theta^T x) \quad (2.15)$$

From part a) the derivative of Sigmoid function and chain rule, we have:

$$= [\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)}]\sigma(\theta^T x)[1-\sigma(\theta^T x)]x \quad (2.16)$$

$$= [\frac{y-\sigma(\theta^T x)}{\sigma(\theta^T x)[1-\sigma(\theta^T x)]}]\sigma(\theta^T x)[1-\sigma(\theta^T x)]x \quad (2.17)$$

$$= [y-\sigma(\theta^T x)]x \quad (2.18)$$

Question 3: Linear regression and OLS.

(a) Plot the seven data points. Without performing any calculations, what do you think is the result of fitting a linear model to this dataset using the MSE (i.e., what is the slope and intercept of the best fit line)?

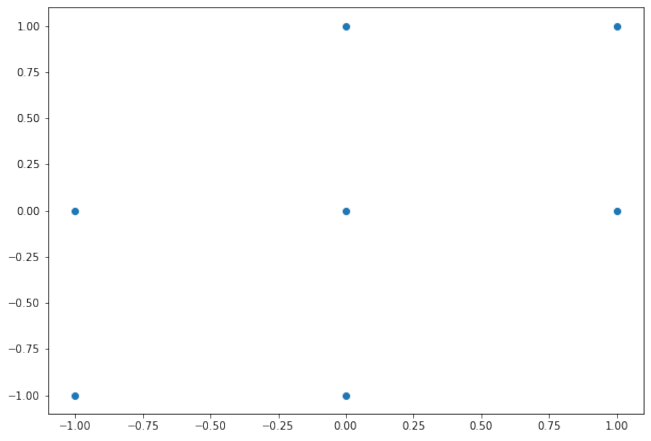


Figure 2.1: Plot the Seven Data Points

I think the result of fitting a linear model to this dataset would be y = 0.5x (slope: 0.5; intercept: 0). From my observation, I think this is the line yielding a small MSE as the distance to the line is minimized overall.

(b) Calculate the slope and the intercept of a linear model fit to this dataset using the MSE. Explain your answer via a mathematical argument or via code that computes the model parameters. If the answer is different from what you wrote in (a), comment on why that's the case.

I implemented the Least Mean Squares (LMS)[1] with gradient descent algorithm. The slope obtained from MSE method is 0.50000027, and intecept is 0. It's close to my prediction in part a).

```
#The linear model we are trying to fit
def f(X, theta):
    """The linear model we are trying to fit.

    Parameters:
    theta (np.array): d-dimensional vector of parameters
    X (np.array): (n,d)-dimensional data matrix

    Returns:
    y_pred (np.array): n-dimensional vector of predicted targets
    """
    return np.dot(X,theta)

#Define mean squared error
def mean_squared_error(theta, X, y):
    """The cost function, J, describing the goodness of fit.

    Parameters:
    theta (np.array): d-dimensional vector of parameters
    X (np.array): (n,d)-dimensional design matrix
    y (np.array): n-dimensional vector of targets
    """
    return 0.5*np.mean((y-f(X, theta))**2)

#Define the gradient of cost function
def mse_gradient(theta, X, y):
    """The gradient of the cost function.

    Parameters:
    theta (np.array): d-dimensional vector of parameters
    X (np.array): (n,d)-dimensional design matrix
    y (np.array): n-dimensional vector of targets

    Returns:
    grad (np.array): d-dimensional gradient of the MSE
    """
    return np.mean((f(X, theta) - y) * X.T, axis=1)

# input
X = np.array([[-1], [-1],[0], [0], [0], [1], [1]])
y = np.array([-1, 0, -1, 0, 1, 0, 1])

# using Widrow-Hoff learning rule to train our linear model with gradient descent algorithm
threshold = 1e-7
step_size = 4e-1
theta, theta_prev = np.array([10]), np.array([1])
opt_pts = [theta]
opt_grads = []
iter = 0

while np.linalg.norm(theta - theta_prev) > threshold:
    theta_prev = theta
    gradient = mse_gradient(theta, X, y)
    theta = theta_prev - step_size * gradient
    opt_pts += [theta]
    opt_grads += [gradient]
    iter += 1

x_line = np.stack([np.linspace(-1, 1, 3)])
y_line = opt_pts[-1].dot(x_line)

# plot data points
fig, ax = plt.subplots(figsize =(5, 5))
ax.scatter(X, y,  color='black')
ax.plot(x_line[0], y_line)
plt.xlabel('X')
plt.ylabel('y')
plt.savefig("seven_data_point_mse")
```

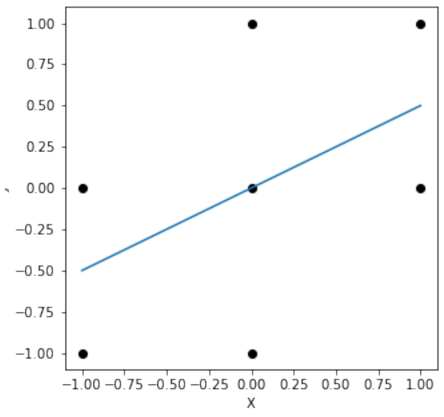


Figure 2.2: Plot the Line with Smallest MSE

(c) Calculate the slope and the intercept of a linear model fit to this dataset using the mean absolute error (i.e., the L1 loss). Briefly explain how you obtained your answer and comment on the result.

The slope should be [0, 1]; the intercept should be 0.
Let suppose $y = kx + b$. We can calculate the mean absolute error using the seven data points:

$$MAE = \frac{\sum_n |y_{pred_i} - y_i|}{n} = \frac{1}{7} * [|(k+b)-1| + |k+b| + |b+1| + |b| + |b-1| + |k+b| + |1-(k+b)|]$$

$|(k+b)-1|+|k+b|+|k+b|+|1-(k+b)|$ is minimized when $0 \leq k+b \leq 1$. $|b+1|+|b|+|b-1|$ is minimized when $b = 0$. Thus, MAE can be minimized when $k \in [0,1], b = 0$

Reference

[1] Volodymyr Kuleshov. *Linear Regression*. URL: `https://github.com/kuleshov/cornell-cs5785-2021-applied-ml/blob/main/notebooks/lecture3-linear-regression.ipynb`. (accessed: 09.16.2021).