CME 466: Design of a Digital System

Winter 2024

Machine Learning – Linear Regression [part 2]

KHAN A. WAHID, PhD, PEng, SMIEEE

Professor
Electrical and Computer Engineering
University of Saskatchewan
Email: khan.wahid@usask.ca

1. Build regression model using multiple features (Diabetes dataset)

- In the last section, we built a linear regression model using the Diabetes dataset where only one feature was used. Now, we will rebuild the model using multiple features. The object X_train will contain all 10 features.

1.1 Using built-in method to spilt the train/test dataset[1]

- Instead of manually splitting the dataset between train and test, we can use train_test_split method that will do it automatically for us. In that case, we can choose several parameters. If *test_size* and *train_size* are not given, the default split is 25% (i.e., 75% of the data will be used in training and the remaining 25% is kept for testing).

- Also note that, by default the data is shuffled. If we do not want shuffle, we must use *shuffle=False*. Parameter *random_state* (integer value) controls the shuffling.

```
diabDS = load_diabetes()
print(diabDS.data.shape)

from sklearn.model_selection import train_test_split
# from here, we are changing the names to be consistent with other codes
X_train, X_test, y_train, y_test = train_test_split(diabDS.data, diabDS.target,
test_size=0.1, random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)


from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
print(lin_reg)
```

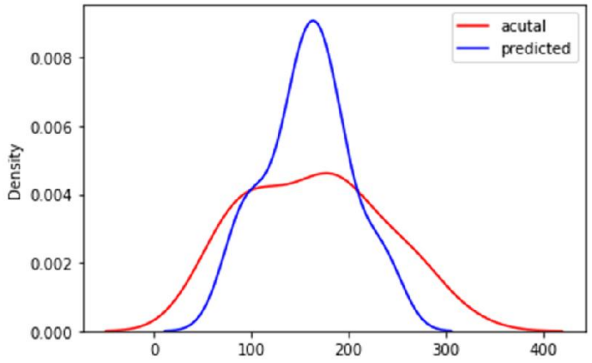
[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
print(y_pred)
# see evaluation parameters
print("Coefficients: \n", lin_reg.coef_)
print("Mean squared error: %.2f" % mse(y_test, y_pred))
print("Coefficient of determination: %.2f" % r2(y_test, y_pred))
```

```
Coefficients:
 [ -23.99554987 -203.06216444  495.1922928   303.69217352  -59.89808336
  -73.77823728 -186.47331851  102.19474559  444.8128758    92.81740705]
Mean squared error: 3152.25
Coefficient of determination: 0.35
```

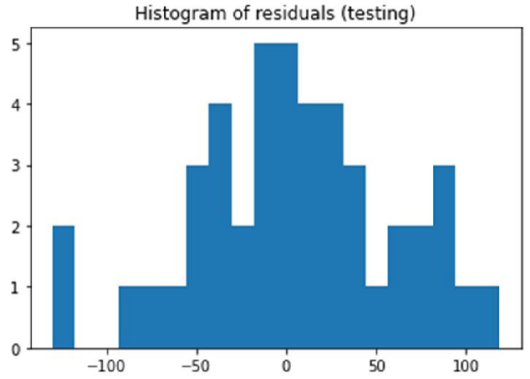
- Did our performance improve with 10 features? The coefficient of determination has gone down, compared with the previous example. The mean squared error has improved. **What is the conclusion? Is this model better or worse?**

```
# see kernel density estimate (KDE) plot
import seaborn as sns
sns.kdeplot(y_test, color='red', label='acutal')
sns.kdeplot(y_pred, color='blue', label='predicted')
plt.legend()
```

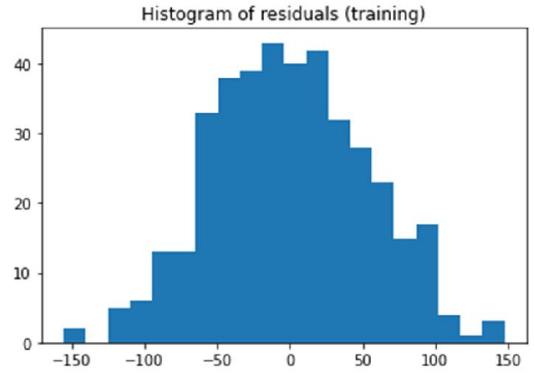


- We can no longer scatter plot the training data points like before, because they are multi-D array (397 x 10 array). However, plotting the residuals (differences between each target value and the corresponding predicted value) is another good way to see the performance.

```
res_y = y_test - y_pred
plt.hist(res_y, bins=20)
plt.title("Histogram of residuals (testing)")
print([np.mean(res_y)])
```




```
ytr_pred = lin_reg.predict(X_train)
res_ytr = y_train - ytr_pred
plt.hist(res_ytr, bins=20)
plt.title("Histogram of residuals (training)")
print([np.mean(res_ytr)])
```



- Use the train_test_split and try different splitting ratio. You can also try using different *random_state*.
- Retrain your model and see if you can improve the performance.
- Redo the experiment with other regression models and observe the performance.

2. How to find the best features while training (Boston dataset)?

- From previous experiments, no conclusion could be drawn as to which feature(s) to take and how many to improve the model's performance. Therefore, in the next section, we will learn how to choose the best feature during a training stage when linear regression is used. This technique can be applied to other machine learning models as well.

- We will use Boston house prices dataset[2]. This dataset is part of scikit-learn version 0.15. It may still be available in newer versions. The full database can be found here: http://lib.stat.cmu.edu/datasets/boston

- The dataset has 13 attributes (e.g., per capita crime rate, land zone, non-retail business area, no of rooms, property tax, etc.) for 506 homes in Boston area with the target being the median value of homes in $1000's.

```
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.data.shape)
print(boston.DESCR)
```

- It is often easier to create a dataframe to visualize the data points better. Let us do it here.

```
import pandas as pd
# Create a dataframe for easier visualization
df = pd.DataFrame(boston.data, columns = boston.feature_names)
df['target'] = boston.target
df
```

[2] Boston house prices dataset

2.1 Exploratory data analysis

- Exploratory data analysis is a very important step before training the model. It will help us choose the features that are more useful and relevant. We will plot some observations to understand the relationship of the target variable with other variables (or features).

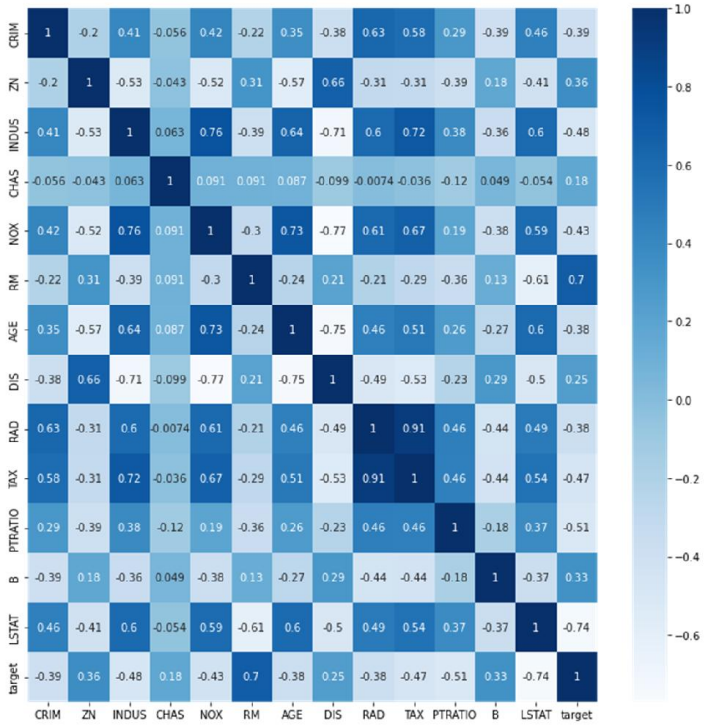
- Let us first plot the histogram of the target values to see the distribution. Look for any outliers.

```
plt.hist(boston['target'], bins=30)
```

- Next, we create a correlation matrix that measures the linear relationships between the variables. The correlation matrix can be easily formed by using the panda dataframe library. We will then use the heatmap function from the seaborn library to plot the correlation matrix.

- The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two variables. If it is close to -1, the variables have a strong negative correlation. They are both important.

- To better fit a model, it is recommended to select the features that have a high correlation with the target.




```
import seaborn as sns
corr_matrix = df.corr()
# print(corr_matrix)
plt.subplots(figsize=(12,12))
sns.heatmap(data=corr_matrix, annot=True,
cmap='Blues')
```

- From the heatmap, we can see that RM has a strong positive correlation with the target, whereas LSTAT has a high negative correlation with it. Therefore, we will use RM and LSTAT as our features.

- **Check for multi-collinearity:** It is a situation where two or more predictors are highly linearly related. In general, an absolute correlation coefficient of >0.7 among two or more predictors indicates the presence of multi-collinearity.

- The features RAD, TAX have a very strong correlation of 0.91. These feature pairs should not be selected together during training, because they tend to have similar effects on the model and it would be difficult to differentiate.

- We can plot the two (or more) chosen features separately to observe the distribution *w.r.t* the target.

```
plt.scatter(df['RM'], df['target'])
plt.scatter(df['LSTAT'], df['target'])

# prepare the data for training
X = pd.concat([df['LSTAT'], df['RM']], axis=1)
y = df['target']
print(X.shape)
print(y.shape)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=5)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# we use default split 75-25

# load evaluation parameters
from sklearn.metrics import mean_squared_error, r2_score
```

```
mse=mean_squared_error
from sklearn import linear_model
lin_reg = linear_model.LinearRegression()
#lin_reg = linear_model.Ridge(alpha=0.1)
#lin_reg = linear_model.Lasso(alpha=0.5)

lin_reg.fit(X_train,y_train)

y_pred = lin_reg.predict(X_test)
print(y_pred)

print("Coefficients: \n", lin_reg.coef_)
print("Mean squared error: %.2f" % mse(y_test, y_pred))
print("Coefficient of determination: %.2f" % r2_score(y_test, y_pred))
```

```
Coefficients:
 [-0.67758923  4.91580979]
Mean squared error: 30.59
Coefficient of determination: 0.63
```

- Using two best features, the coefficient of determination is 63%. You can now use two worse features, **like DIS and RAD (or TAX)**, and see the change in the performance. To apply all the features in your model, you can use the following code and continue as described earlier.

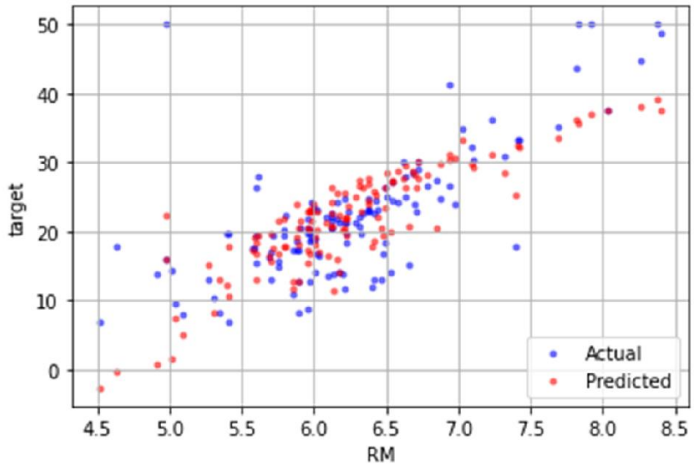
- In addition, you can try different regression models (like Ridge and Lasso) to measure the performance.

- Change the `random_state=5` and redo the experiment. What do you see?

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target,
random_state=5)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

- In addition to performance metrics, we can plot the real values with the predicted values to better visualize the performance of the model.

```
#plot real vs predicted
x_axis = X_test.RM
plt.scatter(x_axis, y_test, c = 'b', alpha = 0.5, marker = '.', label = 'Actual')
plt.scatter(x_axis, y_pred, c = 'r', alpha = 0.5, marker = '.', label =
'Predicted')

plt.xlabel('RM')
plt.ylabel('target')

plt.grid(linestyle = 'solid')

plt.legend(loc = 'lower right')
plt.show()
```



- Finally, we can create a new table by appending the predicted value to others, and compare it with the actual value (target).

```
import warnings
```

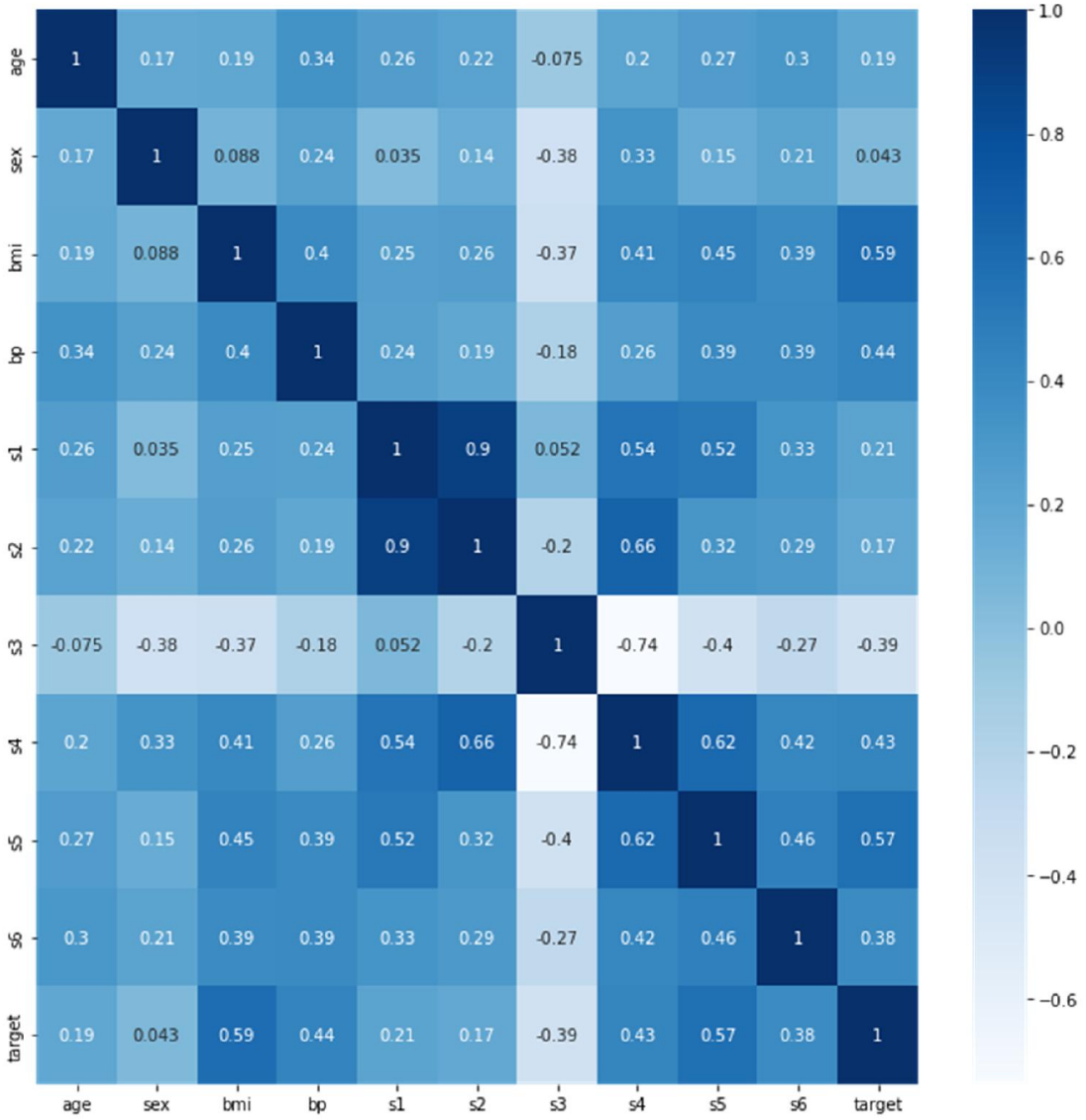
```
warnings.filterwarnings('ignore')

new_table = X_test
new_table['target'] = y_test
new_table['prediction'] = y_pred.tolist()
new_table.head()
```

2.2 Can you find the best feature(s) in the Diabetes dataset?

- You can follow the instructions above to generate a heatmap of the correlation matrix, and find the best feature for the Diabetes dataset.
- Comment on your finding.

```
# need to import numpy to handle data array
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
# import linear regression class from scikit-learn
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
print(lin_reg)

from sklearn.datasets import load_diabetes
diabDS = load_diabetes()
print(diabDS.data.shape)
```

```
df = pd.DataFrame(diabDS.data, columns = diabDS.feature_names)
df['target'] = diabDS.target
df

corr_matrix = df.corr()
plt.subplots(figsize=(12,12))
sns.heatmap(data=corr_matrix,
annot=True, cmap='Blues')
```



3. Performance evaluation of regression models using diamonds dataset

- Another very popular dataset is the diamond dataset[3]. This dataset contains information about 50,000 round-cut diamonds. There are 10 variables measuring various pieces of information about the diamonds. Let us repeat the experiment for this dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import linear_model
lin_reg = linear_model.LinearRegression()
#lin_reg = linear_model.Ridge(alpha=0.1)
#lin_reg = linear_model.Lasso(alpha=0.5)
print(lin_reg)

# load evaluation parameters
from sklearn.metrics import mean_squared_error, r2_score
mse=mean_squared_error
r2=r2_score

diamonds = pd.read_csv('diamonds.csv')
df = diamonds
print(df.shape)
df.head(10)
```

[3] https://bookdown.org/yih_huynh/Guide-to-R-Book/diamonds.html

- **Preprocessing of dataset:** However, after loading the dataset, we notice that the first column has an "Unnamed" feature. This may create issues in our model. It is highly common to have unnecessary and/or incomplete features in a dataset. Therefore, we need to remove it.

```
df = df.drop(['Unnamed: 0'], axis = 1)
# delete Unnamed column, axis = 1 means we are dropping column
df.head(10)
```

- We will now use LabelEncoder[4] to preprocess the dataset. Note that, some features are descriptive. For example, 'cut', 'color' and 'clarity' have descriptive measurements. We can replace them with numeric values using LabelEncoder.

- The method `fit_transform` fits to data, then transforms it. In other words, it fits transformer to X and y with optional parameters `fit_params`, and returns a transformed version of X.

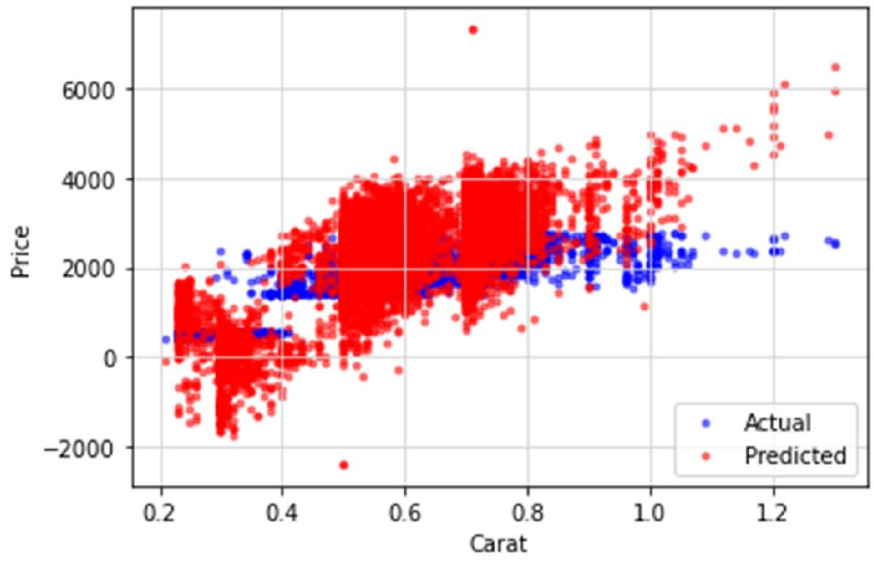
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# change feature names to numeric values
categorical_features = ['cut', 'color', 'clarity']
for i in range(3):
    new = le.fit_transform(df[categorical_features[i]])
    df[categorical_features[i]] = new
df.head(10)
```

- The dataset is now ready for the experiment. The rest of the process is the same as before.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

```
X = df[['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z']]
y = df[['price']]
print(X.shape)
print(y.shape)

plt.scatter(df['carat'], df['price'])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False, random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)


lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
print(y_pred)

print("Coefficients: \n", lin_reg.coef_)
print("Mean squared error: %.2f" % mse(y_test, y_pred))
print("Coefficient of determination: %.2f" % lin_reg.score(X_train, y_train))


#plot real vs predicted
x_axis = X_test.carat
plt.scatter(x_axis, y_test, c = 'b', alpha = 0.5, marker = '.', label =
'Actual')
plt.scatter(x_axis, y_pred, c = 'r', alpha = 0.5, marker = '.', label =
'Predicted')
```

```
plt.xlabel('Carat')
plt.ylabel('Price')

plt.grid(linestyle = 'solid')

plt.legend(loc = 'lower right')
plt.show()
```




```
import warnings
warnings.filterwarnings('ignore')

new_table = X_test
new_table['price'] = y_test
new_table['prediction'] = y_pred.tolist()
new_table.head()
```