CS418_Lab3

October 31, 2022

1 Lab 3: Scikit Learn and Regression

1.0.1 Read an follow the prelab before answering the questions here

1.1 Academic Integrity Guidelines

This lab is an individual assignment (like all the other labs). You should NOT work with your project teammate nor any other classmate. If you have questions, please ask us on Piazza. You must reference (including URLs) of any resources other than those linked to in the assignment, or provided by the professor or TA.

##To submit this assignment: 1. Print your Jupyter Notebook as a PDF and upload it to Gradescope. **Make sure that each line of code has 80 characters maximum and that all your plots and text are properly displayed in the pdf.** 2. Export your Jupyter Notebook as a python file (.py) and upload it to Gradescope. **If your pdf does not show your text and plots properly, submit your jupyter notebook file also (.ipynb)**

Exercise 3.1 (20 pts) Perform the following tasks (TODO):

1. Load this Boston housing dataset:

- info
- housing.csv

2. Display the first 10 rows of the dataset with column names as headers as it is shown in the info file

3. Using your own judgment, select one of the 13 columns as a predictor for MEDV(Median value of owner-occupied homes in $1000's).

[41]:
```
# Your code goes here
# TODO: Load the dataset
import pandas as pd
df = pd.read_csv('housing.csv')
# import pandas as pd
# df = pd.read_html('https://raw.githubusercontent.com/jbrownlee/Datasets/
 ↪master/housing.csv')
df.columns =['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',␣
 ↪'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
# TODO: Display first 10 rows
df.head(10)
```

[41]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX
0	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0
1	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0
2	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0
3	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0
4	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0
5	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311.0
6	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0
7	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311.0
8	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311.0
9	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311.0

	PTRATIO	B	LSTAT	MEDV
0	17.8	396.90	9.14	21.6
1	17.8	392.83	4.03	34.7
2	18.7	394.63	2.94	33.4
3	18.7	396.90	5.33	36.2
4	18.7	394.12	5.21	28.7
5	15.2	395.60	12.43	22.9
6	15.2	396.90	19.15	27.1
7	15.2	386.63	29.93	16.5
8	15.2	386.71	17.10	18.9
9	15.2	392.52	20.45	15.0

####Which column did you choose and why?

I have chosen the column - RM. Because number of rooms is usually directly proportional to a home's value.

4. Make a linear regression model with the feature you chose. Use `train_test_split` to keep 20% of the data for testing.
5. Use your model to predict values for test set and print the predictions for the first 10 instances of the test data and compare them with actual values. (e.g. a table with columns predicted value and actual value).
6. Print the coefficient value and its corresponding feature name (e.g. CRIM 0.00632)

7. Calculate training-MSE, testing-MSE, and R-squared value.

[71]:
```
# Your code goes here
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
# TODO: Select ONE FEATURE for linear regression
X = df[['RM']]
y = df[['MEDV']]
# TODO: Split the data into training/testing sets
X_train,X_test , y_train, y_test = train_test_split(
```

```
                                        X, y, test_size = 0.20)
lr = LinearRegression()
# df_X_train.shape
# df_y_train.shape
# TODO: Train the model
lr.fit(X_train, y_train)

# TODO: Make predictions on test data
pred = lr.predict(X_test)
# print(tabulate(data, headers=["Pos", "Predicted Values", "Actual Values"]))
print('Predicted values vs Actual values')
d = {'Predicted Values': pred[:10], 'Actual Values': y_test[:10]}
print(d)
print('------------------------------------------------------------------')
# TODO: print the model coefficient with feature names
print('Model coefficient with feature names')
# print(X_train.columns)
for col in X_train.columns:
    print(col)
print(lr.coef_)
print('------------------------------------------------------------------')
# TODO: print the model intercept
print('Model intercept')
print(lr.intercept_)
print('------------------------------------------------------------------')
# TODO: print the r-squared
from sklearn.metrics import r2_score
r2 = r2_score(y_test, pred)
print('r-squared')
print(r2)
print('------------------------------------------------------------------')
# TODO: print the mean squared error for training
from sklearn.metrics import mean_squared_error
pred_train = lr.predict(X_train)
print('MSE for training : ' + str(mean_squared_error(y_train, pred_train)))
# TODO: print the mean squared error for testing
print('MSE for testing : ' + str(mean_squared_error(y_test, pred)))
print('------------------------------------------------------------------')
```

```
Predicted values vs Actual values
{'Predicted Values': array([[34.54026669],
       [24.51628901],
       [31.7317866 ],
       [20.09160811],
       [19.43806243],
       [17.21247444],
       [14.83674757],
```

```
       [32.99471947],
       [24.05704069],
       [23.58896068]]), 'Actual Values':      MEDV
281  46.0
158  23.3
262  31.0
293  21.7
15   23.1
404   5.0
488   7.0
267  43.5
129  19.2
90   22.0}
------------------------------------------------------------------
Model coefficient with feature names
RM
[[8.83169839]]
------------------------------------------------------------------
Model intercept
[-32.97806754]
------------------------------------------------------------------
r-squared
0.5634007372139881
------------------------------------------------------------------
MSE for training : 45.97878512680088
MSE for testing : 34.68826382068348
------------------------------------------------------------------
```

8. Make a linear regression model with **all the features** in the dataset (MEDV is not a feature, it is the value to predict). Use `train_test_split` to keep 20% of the data for testing.
9. Use your model to predict values for test set and print the predictions for the first 10 instances of the test data and compare them with actual values.
10. Print the coefficient values and their corresponding feature name
11. Calculate training-MSE, testing-MSE, and R-squared value.

[61]:
```
# Your code goes here

# TODO: Select ALL THE FEATURES for linear regression
X1 = df[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT']]
y1 = df['MEDV']
# TODO: Split the data into training/testing sets
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2)
# TODO: Train the model
lm = LinearRegression()
lm.fit(X1_train,y1_train)
# TODO: Make predictions on test data
pred1 = lm.predict(X1_test)
```

```
# TODO: print the model coefficient with feature names
print('Model coefficient with feature names')
print(str(X1_train.columns) + '  ' + str(lm.coef_))
print('-------------------------------------------------------------')
# TODO: print the model intercept
print('Model intercept')
print(lm.intercept_)
print('-------------------------------------------------------------')
# TODO: print the r-squared
# from sklearn.metrics import r2_score
r2_1 = r2_score(y1_test, pred1)
print('r-squared')
print(r2_1)
print('-------------------------------------------------------------')
# TODO: print the mean squared error for training
# from sklearn.metrics import mean_squared_error
pred1_train = lm.predict(X1_train)
print('MSE for training : ' + str(mean_squared_error(y1_train, pred1_train)))
# TODO: print the mean squared error for testing
print('MSE for testing : ' + str(mean_squared_error(y1_test, pred1)))
print('-------------------------------------------------------------')
```

```
Model coefficient with feature names
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')  [-1.10395242e-01  3.29866184e-02  1.87221062e-02
3.17535918e+00
 -2.01023731e+01  3.18090448e+00  2.25493116e-02 -1.36785138e+00
  3.16303679e-01 -9.95787688e-03 -1.10042829e+00  7.90313769e-03
 -6.40724075e-01]
-------------------------------------------------------------
Model intercept
43.55405096702418
-------------------------------------------------------------
r-squared
0.7545604410992675
-------------------------------------------------------------
MSE for training : 22.753865549060905
MSE for testing : 20.414164512408018
-------------------------------------------------------------
```

Compare the two models (with all the features versus one feature). Which model has a better performance? Why? The model with all the features is the better performing one. It's mean squared error is lesser than the one with a single feature

Exercise 3.2 (15 pts)

1. Use `RFE` to select the two most important features.

2. Make a Linear regression model using the features that RFE selected.
3. Calculate and print both train and test MSE and R-squared values, and compare this model with the models from 3.1.

[62]:
```
# Your code goes here
from sklearn.feature_selection import RFE

model = LinearRegression()
rfe = RFE(model, n_features_to_select=2)
rfe.fit(X1,y1)
Xrfe = rfe.transform(X1)
# Split the data into training/testing sets
Xrfe_train, Xrfe_test, yrfe_train, yrfe_test = train_test_split(Xrfe, y1,␣
 ↪test_size=0.2)
# Train the model
lrrfe = LinearRegression()
lrrfe.fit(Xrfe_train,yrfe_train)
# Make predictions on test data
predrfe = lrrfe.predict(Xrfe_test)
# print the r-squared
r2_rfe = r2_score(yrfe_test, predrfe)
print('r2 Score : ' + str(r2_rfe))
print('----------------------------------------------------------------')
# print the mean squared error for training
predrfe_train = lrrfe.predict(Xrfe_train)
print('MSE for training : ' + str(mean_squared_error(yrfe_train,␣
 ↪predrfe_train)))
# print the mean squared error for testing
print('MSE for testing : ' + str(mean_squared_error(yrfe_test, predrfe)))
print('----------------------------------------------------------------')
# Print the columns selected by rfe
print('columns selected by rfe')
for i in range(X1.shape[1]):
  if rfe.support_[i] == True:
          print('Column: %d' % (i))
print('----------------------------------------------------------------')
```

```
r2 Score : 0.5556249881405007
----------------------------------------------------------------
MSE for training : 39.660419359513234
MSE for testing : 37.90592035165051
----------------------------------------------------------------
columns selected by rfe
Column: 4
Column: 5
----------------------------------------------------------------
```

Explain your observation. What are the important features found using RFE? The two most important features found were columns 5 and 6 (considering indexing from 0) i.e. NOX and RM.

Compare the three models (with one feature, two features and all the features versus). Which model has a better performance? The model with all the features seem to have the least MSE thus best performance. The model with two features performs better than the one with single feature but occassionally performs bad. This could be due to the changing samples in the train-test split.

[ ]:
```
# Your code goes here
```

1.1.1 Linear regression on the Boston house price dataset

Now it's your turn to perform a linear regression on the **Boston housing** dataset.

Exercise 3.3 (15 pts)

Let's build models with different training sizes to predict the house prices for boston house dataset. Each model should use all the available features. The goal is to train multiple linear regression models for the following train-test splits and calculate and visualize their MSE:

1. 30% training, 70% testing
2. 40% training, 60% testing
3. 50% training, 50% testing
4. 60% training, 40% testing
5. 70% training, 30% testing
6. 80% training, 20% testing. (done previously)

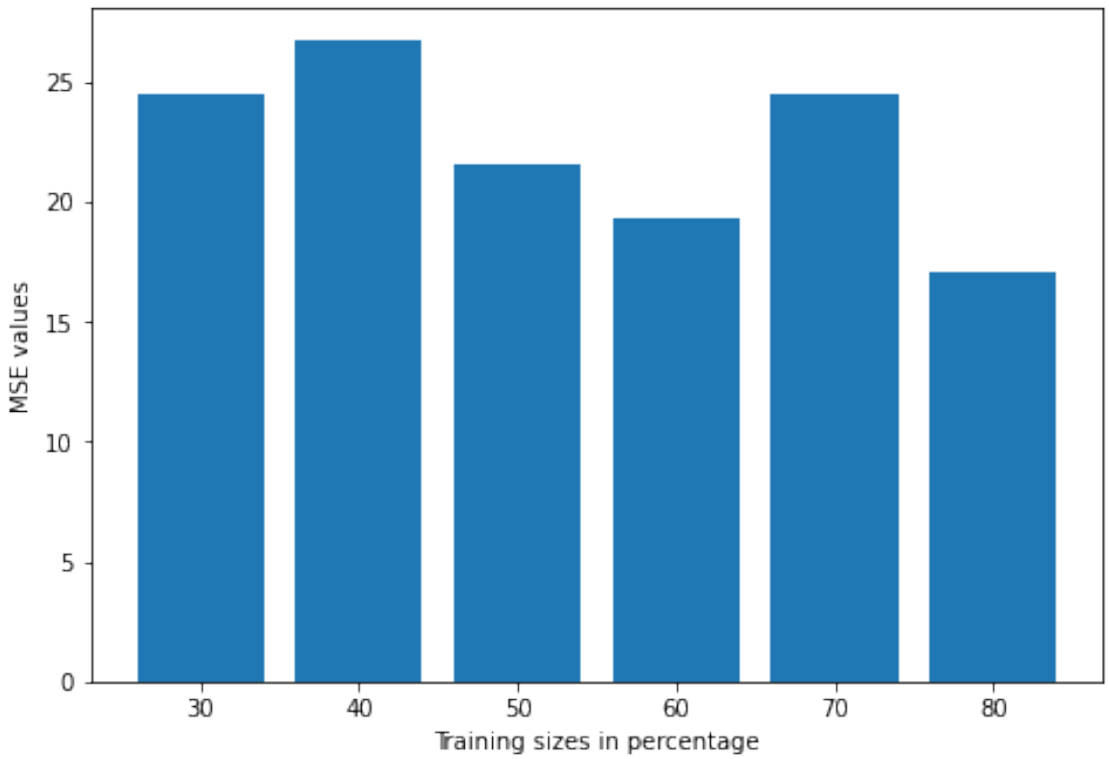
Note: For simplicity, make a function that builds and fits the linear model and returns MSE for test.

[27]:
```
#Your code goes here
def modelBuilding(x, Y, train_size):
  # Split the data into training/testing sets
  x_train, x_test, Y_train, Y_test = train_test_split(x, Y,␣
 ↪test_size=(1-train_size))
  # TODO: Train the model
  lm = LinearRegression()
  lm.fit(x_train,Y_train)
  # TODO: Make predictions on test data
  pred1 = lm.predict(x_test)
  # print the mean squared error for testing
  return (mean_squared_error(Y_test, pred1))
```

Plot the test MSE values for all models with respect to the training size.

[30]:
```
#Your code goes here
import matplotlib.pyplot as plt
fig = plt.figure()
```

```
ax = fig.add_axes([0,0,1,1])
xaxis = ['30','40','50','60','70','80']
mse_values = []
for i in xaxis:
  tr_size = (int(i)/100)
  mse_values.append(modelBuilding(X1,y1,tr_size))
ax.set_ylabel('MSE values')
ax.set_xlabel('Training sizes in percentage')
ax.bar(xaxis,mse_values)
plt.show()
```



Exercise 3.4 (30 pts)

Next, we want to use `RFE` to find the best set of features for prediction. When we used `RFE` in 3.2, we chose an arbitrary value for number of features. In practice, we don't know the best number of features to select for `RFE`, so we have to test different values. In this case, the number of features is the hyperparameter that we want to tune. Typically, we use cross validation for tuning hyperparameters.

Recall that cross validation is a technique where we split our data into equal folds and then use one fold for testing and the rest for training. We can use `KFold` Documentation form scikit learn `model_selection` to split our data into desired folds. We can also use `cross_val_score` Documentation to evaluate a score by cross validation.

For this exercise, use `RFE` with cross-validation (K = 5 aka 5fold) to find the best set of features for prediction.

- Make an RFE model with i number of features
- Create a 5-fold CV on the data set
- Fit the model with Cross validation using the best features and store the MSE values
- Draw a box plot for MSE values of each model and pick the best number of features

Note that the Boston housing data set contains 13 features, so you must create 13 models (one per each i best features selected with RFE) and use that model with 5-fold CV. At the end, you should have 13 models with 5 MSE values for each model (results of CV).

Plot a side-by-side boxplot and compare the distribution of MSE values for these 13 models.

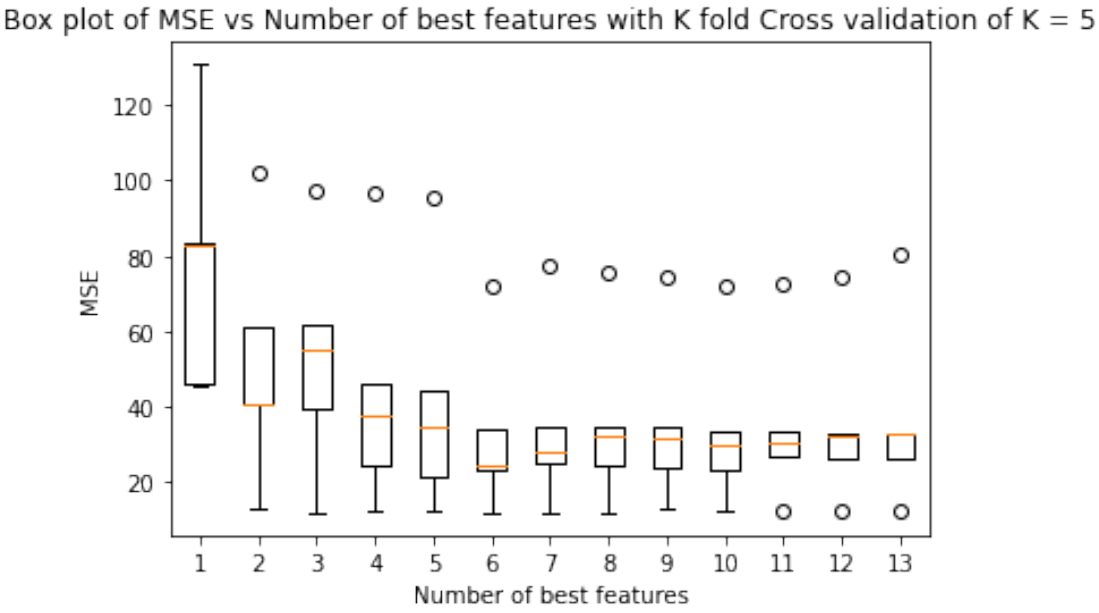
[39]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

#Your code goes here
list_scores = []
for i in range(13):
  mdl = LinearRegression()
  rfekf = RFE(mdl, n_features_to_select=i+1)
  rfekf.fit(X1,y1)
  Xrfekf = rfekf.transform(X1)
  folds = KFold(n_splits = 5)
  scores = cross_val_score(lm, Xrfekf, y1, scoring='neg_mean_squared_error',␣
 ↪cv=folds)
  scores = scores * -1
  list_scores.append(scores)

fig, ax = plt.subplots()
ax.boxplot(list_scores,meanline = True)

ax.set_title('Box plot of MSE vs Number of best features with K fold Cross␣
 ↪validation of K = 5')
ax.set_xlabel('Number of best features')
ax.set_ylabel('MSE')
xticklabels=['1','2','3','4','5','6','7','8','9','10','11','12','13']
ax.set_xticklabels(xticklabels)

plt.show()
```



Based on the previous plot, pick the model with lowest average MSE as the best result. What is the best number of features? Which features are selected?

The model with the 6 best features has the lowest average MSE

[36]:
```
# Print the columns selected by rfe
md = LinearRegression()
rfe = RFE(mdl, n_features_to_select=6)
rfe.fit(X1,y1)
print('Columns selected by rfe')
for i in range(X1.shape[1]):
  if rfe.support_[i] == True:
          print('Column: %d' % (i))
```

```
Column: 3
Column: 4
Column: 5
Column: 7
Column: 10
Column: 12
```

The columns 4,5,6,8,11 and 13 (considering zero indexing) i.e. CHAS, NOX, RM, DIS, PTRATIO and LSTAT have been selected as the 6 best features.

Exercise 3.5 (10 pts) Another way of doing RFE with CV is using `RFECV` Documentation which performs the same task as `RFE` while performing a cross validation task. Use `RFECV` to find the best number of features for a regression model for Boston housing data set. Which features are selected? (you can use `ranking_` attribute)

Some Notes:

- For `scoring` parameter you can use `neg_mean_squared_error` to get MSE. You can read model evaluation documentation to learn what values you can pass for this parameter
- The default value for `cv` parameter in `RFECV` model is 5-fold cross-validation
- Your results for RFE may vary (values with different numerical precision) given the stochastic nature of the algorithm and evaluation process.
- Specifying random_state parameter ensures the same random partitions are used across different runs
- Read the documentation to learn more about parameters and attributes for each model we discussed.

[40]:
```
#Your code goes here
from sklearn.feature_selection import RFECV
mdl = LinearRegression()
selector = RFECV(mdl, cv=5, scoring ='neg_mean_squared_error' )
selector = selector.fit(X1, y1)
selector.ranking_
```

[40]:
```
array([3, 4, 6, 1, 1, 1, 8, 1, 2, 5, 1, 7, 1])
```

###Which features are selected?

The features selected by RFECV are the columns 4,5,6,8,11 and 13 i.e. CHAS, NOX, RM, DIS, PTRATIO and LSTAT

Exercise 3.6 (10 pts) The Boston housing prices dataset has an ethical problem. Describe here why.

###**The inclusion of the column 'B' which stands for Black proportion of population is the ethical problem here. The authors of the dataset, in their paper talk about their assumption that increase in B should negatively affect MEDV. The inclusion of this column also will lead to systemic racism if used official systems.**[1]

REFERENCES:

[1] https://medium.com/@docintangible/racist-data-destruction-113e3eff54a8

[ ]: