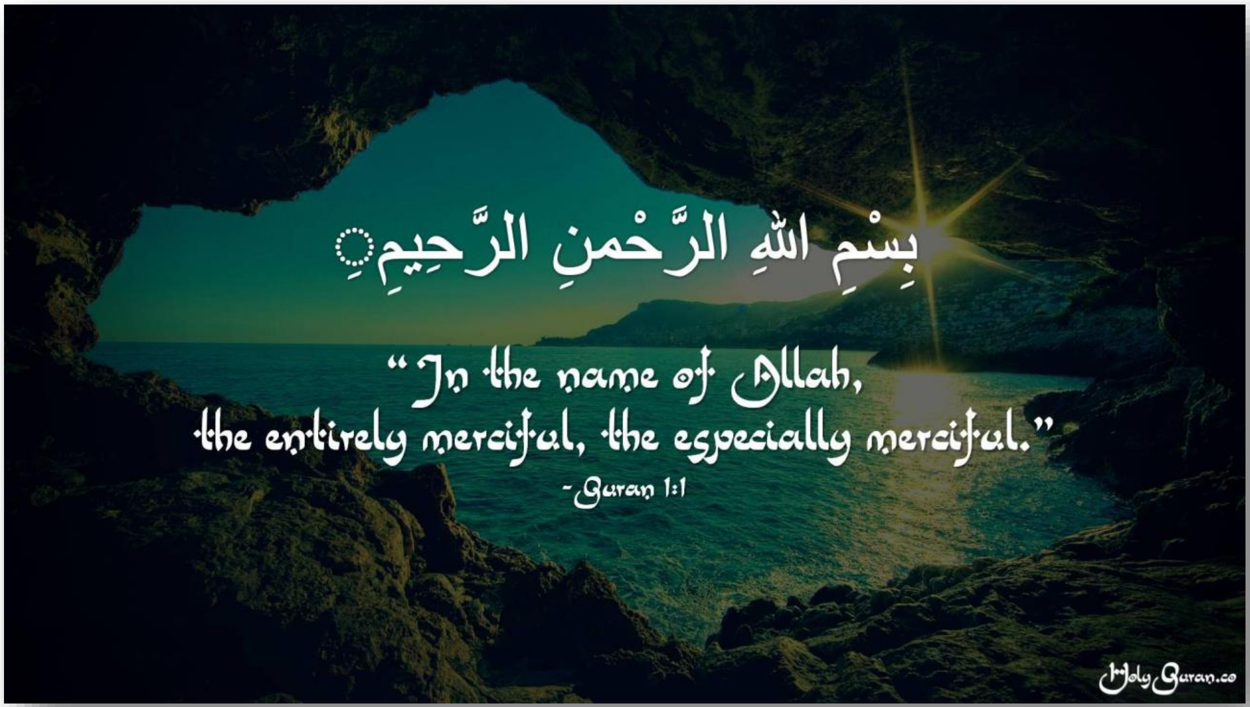
بِسْمِ اللهِ الرَّحْمنِ الرَّحِيمِ
"In the name of Allah,
the entirely merciful, the especially merciful."
-Quran 1:1
HolyQuran.co

SCIENTIFIC COMPUTATION PROJECT

Final project Of Scientific computation course

Using California housing dataset

By: Mojtaba Zolfaghari

Professor :Dr.Ali Shakiba

http://alishakiba.ir

Prepared in the Faculty of Mathematics and Computer Science

Vali-e-Asr University of Rafsanjan

Feb. 2021

In this code, we try to implement different models of linear regression on the California Database. The code written in the two main parts is composed as follows:

1) In the first part, it is generally on the data that is in the current folder, we use and implement various models.

2) In the second part, we use the same data that is also located in the SKLEARN module. Select the two columns called Average Income and Label and then implement the models on these two columns.

Import all libraries

First, we enter the desired libraries into the program as follows:

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

from sklearn.linear_model import LinearRegression, Lasso, Ridge, SGDRegressor, El
asticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sco
re, cross_val_predict, KFold
from sklearn.preprocessing import StandardScaler, PolynomialFeatures, LabelEncode
r,MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer
```

Turn on xkcd sketch-style drawing mode. This will only have effect on things drawn after this function is called. For best results, the "Humor Sans" font should be installed: it is not included with matplotlib. As follows:

```
plt.xkcd()
```

Load the dataset

Read the “housing.csv” file from the folder into the program, as follows:

```
data = pd.read_csv('housing.csv')
```

Preprocessing and preparing the data

But first, before doing anything, we need to make the database understandable to machine learning algorithms, or in other words, do the preprocessing step.For execute preprocessing on “data”. we send it to the pre-processing function as follows:

```
X, y = preprocessing(data)
```

This function takes the dataset as an input argument. Separates the inputs (X) and the label or output (y), then fills in the missing values of the label column with the average of the corresponding column, and finally encodes categorical data and And the returns X and y.

```
def preprocessing(data):

    # Extract input (X) and output (y) data from the dataset

    X = data.iloc[:, :-1].values
    y = data.iloc[:, [-1]].values

    # Handle missing values:
            # Fill the missing values with the mean of the respective column:

    imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
    X[:, :-1] = imputer.fit_transform(X[:, :-1])
    y = imputer.fit_transform(y)
    # Encode categorical data:
        # Convert categorical column in the dataset to numerical data
    X_labelencoder = LabelEncoder()
    X[:, -1] = X_labelencoder.fit_transform(X[:, -1])

    return X, y
```

Generate Polynomial and interaction features

To create polynomial features, we do the following:

```
poly_features = PolynomialFeatures(degree = 3, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

Split the dataset

Here we divide the data into 80% training data and 20% testing data, as follows:

```
Xp_train, Xp_test, yp_train, yp_test = train_test_split(X_poly, y,
                                                    test_size = 0.2,
                                                    random_state = 0)
```

Standardize data

Standardize training and test datasets:

```
scaler = StandardScaler()
Xp_train = scaler.fit_transform(Xp_train)
Xp_test = scaler.transform(Xp_test)
yp_train = scaler.fit_transform(yp_train)
yp_test = scaler.transform(yp_test)
```

So far we have done the data set to use the machine learning functions. Next, we build different models and evaluate them using the plot_learning_curve() and model() functions. But first we will give a brief description of these functions and how to use them.

plot_learning_curve() description

This function plots the learning curve based on the size of the training set. In this way, it first receives the desired parameters from the input and then, by creating train-error and validation-error empty lists, calculates the error of the training set and validation set based on the input model in step-length number and append the error within these two lists. Finally, it draws the curves of the training and evaluation data. The code is as follows:

```
def plot_learning_curve(model,X_train,y_train,X_val,y_val,test_size=0.2,step_length= 100):

    train_err, val_err = [], []
    for m in range(1, len(X_train), step_length):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_err.append(mean_squared_error(y_train[:m], y_train_predict))
        val_err.append(mean_squared_error(y_val, y_val_predict))

    plt.plot(np.sqrt(train_err), "r-+", label="Training")
    plt.plot(np.sqrt(train_err), "b-", label="Validation")
    plt.legend(loc="upper right")
    plt.xlabel("Size of the training set")
    plt.ylabel("RMSE")
    plt.show()
```

model() description

This function takes the pipeline model, the desired parameters, the training set data and the general data from the input. Then it applies input parameters using a GridSearchCV() to report the best performance along with the input parameters. Next, prints the best performance results at each step for different input parameters. It then builds and evaluates 5-Fold cross validation. Finally, we can draws the results on a graph. The desired code is as follows:

```
def model(pipeline, parameters, X_train, y_train, X, y):

    grid_obj = GridSearchCV(estimator=pipeline,
                            param_grid=parameters,
                            cv=3,
                            scoring='r2',
                            verbose=2,
                            n_jobs=1,
                            refit=True)
    grid_obj.fit(X_train, y_train)

    '''Results'''

    results = pd.DataFrame(pd.DataFrame(grid_obj.cv_results_))
    results_sorted = results.sort_values(by=['mean_test_score'], ascending=False)

    print("##### Results")
    print(results_sorted)

    print("best_index", grid_obj.best_index_)
    print("best_score", grid_obj.best_score_)
    print("best_params", grid_obj.best_params_)

    '''Cross Validation'''

    estimator = grid_obj.best_estimator_
    '''
    if estimator.named_steps['scl'] == True:
        X = (X - X.mean()) / (X.std())
        y = (y - y.mean()) / (y.std())
```

```python
'''
shuffle = KFold(n_splits=5,
                shuffle=True,
                random_state=0)

cv_scores = cross_val_score(estimator,
                            X,
                            y.ravel(),
                            cv=shuffle,
                            scoring='r2')
print("##### CV Results")
print("mean_score", cv_scores.mean())

'''Show model coefficients or feature importances'''

try:
    print("Model coefficients: ", list(zip(list(X), estimator.named_steps['clf'].coef_)))
except:
    print("Model does not support model coefficients")

try:
    print("Feature importances: ", list(zip(list(X), estimator.named_steps['clf'].feature_importances_)))
except:
    print("Model does not support feature importances")

'''Predict along CV and plot y vs. y_predicted in scatter'''

y_pred = cross_val_predict(estimator, X, y, cv=shuffle)

plt.scatter(y, y_pred)
xmin, xmax = plt.xlim()
ymin, ymax = plt.ylim()
plt.plot([xmin, xmax], [ymin, ymax], "g--", lw=1, alpha=0.4)
plt.xlabel("True prices")
plt.ylabel("Predicted prices")
plt.annotate(' R-squared CV = {}'.format(round(float(cv_scores.mean()), 3)), size=9,
                         xy=(xmin,ymax), xytext=(10, -15), textcoords='offset points')
plt.annotate(grid_obj.best_params_, size=9,
             xy=(xmin, ymax), xytext=(10, -35), textcoords='offset points', wrap=True)
plt.title('Predicted prices (EUR) vs. True prices (EUR)')
plt.show()
```

Building and testing model with different types of Regressors

Now it's time to implement the described items on each model, let's start ...

Task1) KNeighborsRegressor:

Set pipeline and Parameters for KNN and Execute model hyperparameter tuning and crossvalidation as follows:

```
knn = KNeighborsRegressor(n_neighbors=1)

pipe_knn = Pipeline([('clf', KNeighborsRegressor())])

param_knn = {'clf__n_neighbors':[5, 10, 15, 25, 30]}

model(pipe_knn, param_knn, Xp_train, yp_train, X, y)
```

Task2) SGDRegressor:

Learning curve for SGDRegressor:

```
sgd = SGDRegressor()

plot_learning_curve(sgd,
                    Xp_train,
                    yp_train.ravel(),
                    Xp_test,
                    yp_test.ravel(),
                    test_size=0.2,
                    step_length = 100)
```

Pipeline and Parameters for SGDRegressor and Execute model hyperparameter tuning and crossvalidation:

```
pipe_sgd = Pipeline([('clf', SGDRegressor())])

param_sgd = {'clf__alpha': [0.0001, 0.00001, 0.1, 1, 10]}


model(pipe_sgd, param_sgd, Xp_train, yp_train.ravel(), X, y.ravel())
```

Task3) Polynomial regression:

Plotting earning curve for LinearRegression and also use pipeline and set Parameters for Polynomial Regression and Execute model hyperparameter tuning and crossvalidation:

```
linear_regression= LinearRegression()


# ##### Learning curve for Plynomial Regression


plot_learning_curve(linear_regression,
                    Xp_train,
                    yp_train.ravel(),
                    Xp_test,
                    yp_test.ravel(),
                    test_size=0.2,
                    step_length = 100)


# #### Pipeline and Parameters for Polynomial Regression and Execute model hyperp
arameter tuning and crossvalidation
pipe_poly = Pipeline([('clf', LinearRegression())])

param_poly = {}


model(pipe_poly, param_poly, Xp_train, yp_train, X, y)
```

Task4) Ridge regression:

Plotting earning curve for Ridge Regression and also use pipeline and set Parameters for Ridge Regression and Execute model hyperparameter tuning and crossvalidation:

```
ridge_regression= Ridge()


# ##### Learning curve for LinearRegression



plot_learning_curve(ridge_regression,
                    Xp_train,
                    yp_train,
                    Xp_test,
                    yp_test,
                    test_size=0.2,
                    step_length = 100)


# #### Pipeline and Parameters for Ridge and Execute model hyperparameter tuning
and crossvalidation



pipe_ridge = Pipeline([('clf', Ridge())])

param_ridge = {'clf__alpha': [0.01, 0.1, 1, 10]}


model(pipe_ridge, param_ridge, Xp_train, yp_train, X, y)
```

Task5) Lasso regression:

Plotting earning curve for Lasso Regression and also use pipeline and set Parameters for Lasso Regression and Execute model hyperparameter tuning and crossvalidation:

```
lasso_regression = Lasso(max_iter=1500)


# ##### Learning curve for Lasso
plot_learning_curve(lasso_regression,
                    Xp_train,
                    yp_train,
                    Xp_test,
                    yp_test,
                    test_size=0.2,
                    step_length = 100)


# Pipeline and Parameters - Lasso # Execute model hyperparameter tuning and cross
validation


pipe_lasso = Pipeline([('clf', Lasso(max_iter=1500))])

param_lasso = {'clf__alpha': [0.01, 0.1, 1, 10]}


model(pipe_lasso, param_lasso, Xp_train, yp_train, X, y)
```

Task6) ElasticNet regression:

Plotting earning curve for ElasticNet Regression and also use pipeline and set Parameters for ElasticNe tRegression and Execute model hyperparameter tuning and crossvalidation:

```
elastic_net = ElasticNet()


# ##### Learning curve for ElasticNet


plot_learning_curve(elastic_net,
                    Xp_train,
                    yp_train,
                    Xp_test,
                    yp_test,
                    test_size=0.2,
                    step_length = 100)


# #### Pipeline and Parameters for Lasso and Execute model hyperparameter tuning
and crossvalidation

pipe_elasticnet = Pipeline([('clf', ElasticNet())])

param_lasso = {'clf__alpha': [0.01, 0.1, 1, 10]}


model(pipe_elasticnet, param_lasso, Xp_train, yp_train, X, y)
```

...The End...