Homework #4
Due Th. 11/15

Please prepare a report including the source code, captures of output and send it along with related source files as a zip file to my blackboard. The zip file should be named by your family name (i.e., family-name.zip). Please print the report and bring the hard copy by the due date to the class. The selective grading may be used for homeworks of this course.

1. Answer the following questions:
   (a) What are the overfitting and underfitting problems in machine learning? Support your answer with an example.
   (b) What is the regularization term utilized by support vector machines?
   (c) Suppose you want to perform a prediction task, explain when you will use regression technique versus nearest neighborhood classification?

2. Regression is the process of estimating the relationship between input and output variables. One thing to note is that the output variables are continuous-valued real numbers. In regression, it is assumed that the output variables depend on the input variables, so we want to see how they are related. Consequently, the input variables are called independent variables, also known as predictors, and output variables are called dependent variables, also known as criterion variables.

   Regression analysis helps us in understanding how the value of the output variable changes when we vary some input variables while keeping other input variables fixed. In linear regression, we assume that the relationship between input and output is linear. This puts a constraint on our modeling procedure, but it's fast and efficient. Let's start implementing regression in Python.

   Create a new Python file and import the following packages.

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures
```

   We will use the file `data_multivar_regr.txt` provided to you.:

```
# Input file containing data
input_file = 'data_multivar_regr.txt'
```

   This is a comma-separated file, so we can load it easily with a one-line function call::

```
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

   Split the data into training and testing:

```
# Split data into training and testing
num_training = int(0.8 * len(X))
num_test = len(X) - num_training
```

```
# Training data
X_train, y_train = X[:num_training], y[:num_training]

# Test data
X_test, y_test = X[num_training:], y[num_training:]
```

Create and train the linear regressor model:

```
# Create the linear regressor model
linear_regressor = linear_model.LinearRegression()

# Train the model using the training sets
linear_regressor.fit(X_train, y_train)
```

Predict the output for the test dataset:

```
# Predict the output
y_test_pred = linear_regressor.predict(X_test)
```

Print the performance metrics:

```
# Measure performance
print("Mean absolute error =", round(sm.mean_absolute_error(y_test,y_test_pred),
2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_test_pred),
2))
print("Explained variance score =", round(sm.explained_variance_score(y_test,
y_test_pred), 2))
```

Run the code and chapter screen the output for your report.
What are the mean absolute error and Mean squared errors? Explain their differences in your report

3. Let's see how to use the SVM concept to build a regressor to estimate the housing prices. We will use the dataset available in sklearn where each data point is define, by 13 attributes. Our goal is to estimate the housing prices based on these attributes.

   Create a new Python file and import the following packages:

   ```
   import numpy as np
   from sklearn import datasets
   from sklearn.svm import SVR
   from sklearn.metrics import mean_squared_error, explained_variance_score
   from sklearn.utils import shuffle
   ```

   Load the housing dataset:

   ```
   # Load housing data
   data = datasets.load_boston()
   ```

   Let's shuffle the data so that we don't bias our analysis:

   ```
   # Shuffle the data
   X, y = shuffle(data.data, data.target, random_state=7)
   ```

Split the dataset into training and testing in an 80/20 format:

```
# Split the data into training and testing datasets
num_training = int(0.8 * len(X))
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]
```

Create and train the Support Vector Regressor using a linear kernel. The C parameter represents the penalty for training error. If you increase the value of C, the model will fine-tune it more to fit the training data. But this might lead to overfitting and cause it to lose its generality. The epsilon parameter specifies a threshold; there is no penalty for training error if the predicted value is within this distance from the actual value::

```
# Create Support Vector Regression model
sv_regressor = SVR(kernel='linear', C=1.0, epsilon=0.1)

# Train Support Vector Regressor
sv_regressor.fit(X_train, y_train)
```

Evaluate the performance of the regressor and print the metrics:

```
# Evaluate performance of Support Vector Regressor
y_test_pred = sv_regressor.predict(X_test)
mse = mean_squared_error(y_test, y_test_pred)
evs = explained_variance_score(y_test, y_test_pred)
print("\n #### Performance #### ")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Let's take a test data point and perform prediction:

```
# Test the regressor on test datapoint
test_data = [3.7, 0, 18.4, 1, 0.87, 5.95, 91, 2.5052, 26, 666, 20.2, 351.34,
15.27]
print("\n Predicted price:", sv_regressor.predict([test_data])[0])
```

Run the code and chapter screen the output for your report.

4. Let's see how to build a classifier using Decision Trees in Python. Create a new Python file and import the following packages:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation
from sklearn.tree import DecisionTreeClassifier

from utilities import visualize_classifier
```

We will be using the data in the `data_decision_trees.txt` file that's provided to you. In this file, each line contains comma-separated values. The first two values correspond to the

input data and the last value corresponds to the target label. Let's load the data from that file:

```
# Load input data
input_file = 'data_decision_trees.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Let's load the data from this file:

```
# Load data from input file
data = np.loadtxt(input_file, delimiter=',')
# X, y = data[:, :-1], data[:, -1]
```

Separate the input data into two separate classes based on the labels:

```
# C# Separate input data into two classes based on labels
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```

Let's visualize the input data using a scatter plot:

```
# Visualize input data
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black', edgecolors='black',
linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white', edgecolors='black',
linewidth=1, marker='o')
plt.title('Input data')
```

We need to split the data into training and testing datasets:

```
# Split data into training and testing datasets
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y,
test_size=0.25, random_state=5)
```

Create, build, and visualize a decision tree classifier based on the training dataset. The random_state parameter refers to the seed used by the random number generator required for the initialization of the decision tree classification algorithm. The max_depth parameter refers to the maximum depth of the tree that we want to construct:

```
# Decision Trees classifier
params = {'random_state': 0, 'max_depth': 4}
classifier = DecisionTreeClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
```

Compute the output of the classifier on the test dataset and visualize it:

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')
```

Evaluate the performance of the classifier by printing the classification report:

```
# Evaluate classifier performance
class_names = ['Class-0', 'Class-1']
```

```
    print("\n" + "#"*40)
    print("\n Classifier performance on training dataset \n")
    print(classification_report(y_train, classifier.predict(X_train),
target_names=class_names))
    print("#"*40 + "\n")
    print("#"*40)
    print("\n Classifier performance on test dataset\n")
    print(classification_report(y_test, y_test_pred, target_names=class_names))
    print("#"*40 + "\n")
    plt.show()
```

The performance of a classifier is characterized by precision , recall, and f1-scores. Capture screen the output and explain these metrics in your report .