İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

2010

EEE 450- Introduction to Machine Learning First Application Assignment

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING,
IZMIR KATIP ÇELEBI UNIVERSITY

Turhan Can Kargın | 150403005 | Deadline:31-10-2019

Assignment#1:

You are required to use Python and Scikit-learn for this project. Classify the Wikipedia 300 dataset (150 articles about Video games, 150 about Programming) using machine learning. The dataset can be downloaded at the Datasets page. For text classification the bag-of-words approach where you convert an article to word counts are typically used. An improvement is TF-IDF (Term Frequency-Inverse Document Frequency) which is used to convert from word counts to word frequencies. TF-IDF is especially useful if the size of the articles varies a lot. Suitable algorithms for text classification are Multinomial Naïve Bayes (MultinomialNB) and Support Vector Machines with linear kernels (LinearSVC).

a) Classify the dataset using MultinomailNB and LinearSVC with the bag-of-words approach.
b) Evaluate classification accuracy on the same data as used for training the algorithms.
c) Evaluate classification accuracy using 10-fold cross validation.d) Use TF-IDF to convert from word counts to word frequencies.
e) Does TF-IDF improve classification accuracy when using cross-validation?

a-b) Classifying Multinominal Naive Bayes with the bag-of-word approach and Accuracy:

Bag of Words convert a collection of text documents to a matrix of token counts and to implement this to our code we should use from sklearn.feature_extraction.text import CountVectorizer library code.You can see my codes and outputs below with the explanations.

CODE:

```
import numpy as np
import pandas as pd # library for loading dataset
wikipedia_300 = pd.read_csv("C:/Program Files/Weka-3-8/data/wikipedia_300.csv")#loading dataset
print(wikipedia_300.head())#showing dataset to evaluate
wikipedia_300_x=wikipedia_300["Text"]
wikipedia_300_y=wikipedia_300["Category"]
from sklearn.feature_extraction.text import CountVectorizer#importing bag-of-wordslibrary
cv = CountVectorizer()
bag_of_words = cv.fit_transform(wikipedia_300_x)
print('Bag Of Words = ' ,bag_of_words)#showing that I convert an article to word counts are typically used
from sklearn.model_selection import train_test_split#Train-Test split
x_train, x_test, y_train, y_test = train_test_split(bag_of_words, wikipedia_300_y, test_size=0.2, random_state=4)
from sklearn.naive_bayes import MultinomialNB#importing model
MNB = MultinomialNB()#multinominal naivbe bayes model
MNB.fit(x_train,y_train)
y_pred = MNB.predict(x_test)
from sklearn import metrics
print('Accuracy= ' ,metrics.accuracy_score(y_test,y_pred))#Finding Accuracy
```

OUTPUT:

	Text	Category
0	agile software development software developmen...	Programming
1	debugging debug redirects here for other uses ...	Programming
2	spanning tree protocol internet protocol suite...	Programming
3	cathode cathoderay tube as found in an oscillo...	Games
4	luigi federico menabrea luigi federico menabre...	Programming

```
Bag Of Words =     (0, 6125)       124
   (0, 42761)     108
   (0, 15481)     128
   (0, 13621)     2
   (0, 5753)      3
   (0, 37085)     4
   (0, 39265)     18
   (0, 15354)     17
   (0, 17876)     12
   (0, 13298)     1
   (0, 45661)     14
   (0, 14843)     1
   (0, 15286)     2
   (0, 29165)     3
   (0, 34731)     3
   (0, 6820)      172
   (0, 31012)     4
   (0, 12047)     1
   (0, 24232)     9
   (0, 37389)     2
   (0, 43256)     2
   (0, 30993)     17
   (0, 49618)     7
   (0, 30224)     4
   (0, 20243)     3
   :       :
   (299, 24074)   1
   (299, 13511)   1

          Accuracy=  0.9333333333333333
```

Classifying Support Vector Machines with linear kernels with Bag of Word and Accuracy:

CODE:

```
import numpy as np
import pandas as pd # library for loading dataset
wikipedia_300 = pd.read_csv("C:/Program Files/Weka-3-8/data/wikipedia_300.csv")#loading dataset
wikipedia_300_x=wikipedia_300["Text"]
wikipedia_300_y=wikipedia_300["Category"]
from sklearn.feature_extraction.text import CountVectorizer#importing bag-of-wordslibrary
cv = CountVectorizer()
bag_of_words = cv.fit_transform(wikipedia_300_x)
from sklearn.model_selection import train_test_split#Train-Test split
x_train, x_test, y_train, y_test = train_test_split(bag_of_words, wikipedia_300_y, test_size=0.2, random_state=4)
from sklearn.svm import LinearSVC#importing model
SVC = LinearSVC()#multinominal naivbe bayes model
SVC.fit(x_train,y_train)
y_pred = SVC.predict(x_test)
from sklearn import metrics
print('Accuracy= ' ,metrics.accuracy_score(y_test,y_pred))#Finding Accuracy
```

CODE:

```
Accuracy=  0.9
```

c)Classification Accuracy using 10-fold Cross Validation Multinominal Naive Bayes with the bag-of-word approach:

```
import numpy as np
import pandas as pd # library for loading dataset
wikipedia_300 = pd.read_csv("C:/Program Files/Weka-3-8/data/wikipedia_300.csv")#loading dataset
wikipedia_300_x=wikipedia_300["Text"]
wikipedia_300_y=wikipedia_300["Category"]
from sklearn.feature_extraction.text import CountVectorizer#importing bag-of-words library
cv = CountVectorizer()
bag_of_words = cv.fit_transform(wikipedia_300_x)
from sklearn.model_selection import cross_val_score#importing Cross Validation library
from sklearn.naive_bayes import MultinomialNB#importing model
MNB = MultinomialNB()#multinominal naive bayes model
accuracy = cross_val_score(MNB,bag_of_words,wikipedia_300_y,scoring='accuracy', cv=10)
print('Accuracy with 10 C.V.=',accuracy.mean() * 100) #printing Accuracy
```

OUTPUT:

```
Accuracy with 10 CV= 95.33333333333334
```

Classification Accuracy using 10-fold Cross Validation Support Vector Machines with linear kernels with the bag-of-word approach:

```
import numpy as np
import pandas as pd # library for loading dataset
wikipedia_300 = pd.read_csv("C:/Program Files/Weka-3-8/data/wikipedia_300.csv")#loading dataset
wikipedia_300_x=wikipedia_300["Text"]
wikipedia_300_y=wikipedia_300["Category"]
from sklearn.feature_extraction.text import CountVectorizer#importing bag-of-words library
cv = CountVectorizer()
bag_of_words = cv.fit_transform(wikipedia_300_x)
from sklearn.model_selection import cross_val_score#importing Cross Validation library
from sklearn.svm import LinearSVC #importing model
SVC = LinearSVC() #multinominal naive bayes model
accuracy = cross_val_score(SVC,bag_of_words,wikipedia_300_y,scoring='accuracy', cv=10)
print('Accuracy with 10 C.V.=',accuracy.mean() * 100) #printing Accuracy
```

OUTPUT:

```
Accuracy with 10 C.V.= 91.66666666666667
```

d)Classification Accuracy using 10-fold Cross Validation Multinominal Naive Bayes with the Term Frequency-Inverse Document Frequency:

Term Frequency-Inverse Document Frequency convert a collection of raw documents to a matrix of TF-IDF features and to implement this to our code we should use from sklearn.feature_extraction.text import TfidfVectorizer library code.

CODE:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer# importing TF-IDF library
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from sklearn import svm
from sklearn.model_selection import cross_val_score #importing cross validation library
wikipedia_300 = pd.read_csv("C:/Program Files/Weka-3-8/data/wikipedia_300.csv")
wikipedia_300_x=wikipedia_300["Text"]
wikipedia_300_y=wikipedia_300["Category"]
vectorizer = TfidfVectorizer()#Term Frequency-Inverse Document Frequency
TF_IDF = vectorizer.fit_transform(wikipedia_300_x)
MNB = MultinomialNB()
accuracy = cross_val_score(MNB,TF_IDF,wikipedia_300_y,scoring='accuracy', cv=10)#applying 10 Cross Validation
print('Accuracy with 10 CV (TF-IDF)=',accuracy.mean() * 100)
```

OUTPUT:

```
Accuracy with 10 CV (TF-IDF)= 95.66666666666667
```

Classification Accuracy using 10-fold Cross Validation Support Vector Machines with linear kernels with the Term Frequency-Inverse Document Frequency:

CODE:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer# importing TF-IDF library
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn import svm
from sklearn.model_selection import cross_val_score #importing cross validation library
wikipedia_300 = pd.read_csv("C:/Program Files/Weka-3-8/data/wikipedia_300.csv")
wikipedia_300_x=wikipedia_300["Text"]
wikipedia_300_y=wikipedia_300["Category"]
vectorizer = TfidfVectorizer()#Term Frequency-Inverse Document Frequency
TF_IDF = vectorizer.fit_transform(wikipedia_300_x)
SVC = LinearSVC()
accuracy = cross_val_score(SVC,TF_IDF,wikipedia_300_y,scoring='accuracy', cv=10)#applying 10 Cross Validation
print('Accuracy with 10 CV=',accuracy.mean() * 100)
```

OUTPUT:

```
Accuracy with 10 CV= 96.03758169934642
```

e) Consequently, We can say that using TF-IDF is better than using bag-of-word because for 10-fold Cross Validation Multinominal Naive Bayes with the bag-of-word approach we have 95.33 % accuracy and for 10-fold Cross Validation Support Vector Machinewith linear kernels with the bag-of-word approach we have 91.66 % accuracy but when we use TF-IDF we have respectively 95.66 % and 96.03 % accuracy.Therefore, we can easily say that TD-IDF improved classification accuracy when using cross-validation.

Assignment#2:

Text classification in the WEKA tool.

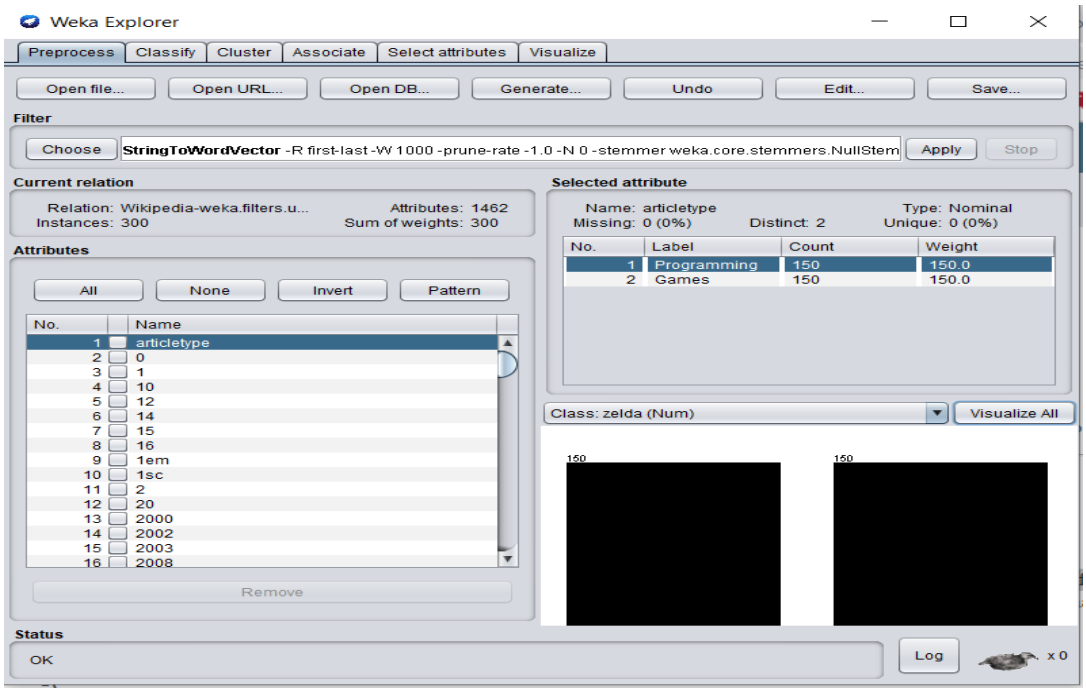
Classify the Wikipedia 300 dataset (150 articles about Video games, 150 about Programming) using machine learning. The dataset can be downloaded at the Datasets page.

- a)Apply the StringToWordVector filter in the Preprocess tab, and select Articletype in the target attribute dropdown list in the Classify tab.

- b) Classify the dataset in Weka using the algorithms NaiveBayes and NaiveBayesMultinomial and 10-fold cross validation. What are the differences between the two classifiers, and why do you think one is better than the other?

SOLUTİON:

a) Firstly, from preprocess tab I applied the StringToWordVector filter tab, and selected Articletype in the target attribute dropdown list in the Classify tab. **StringToWordVector** converts string attributes into a set of numeric attributes representing word occurrence information from the text contained in the strings. The dictionary is determined from the first batch of data filtered (typically training data). Note that this filter is not strictly unsupervised when a class attribute is set because it creates a separate dictionary for each class and then merges them.



b)

NaiveBayes

```
Time taken to build model: 0.25 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         278               92.6667 %
Kappa statistic                          0.8533
Mean absolute error                      0.0734
Root mean squared error                  0.2707
Relative absolute error                 14.6855 %
Root relative squared error             54.1415 %
Total Number of Instances              300

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,953    0,100    0,905      0,953    0,929      0,855    0,964     0,939     Programming
                 0,900    0,047    0,951      0,900    0,925      0,855    0,974     0,960     Games
Weighted Avg.    0,927    0,073    0,928      0,927    0,927      0,855    0,969     0,950

=== Confusion Matrix ===

   a   b   <-- classified as
 143   7 |   a = Programming
  15 135 |   b = Games
```

NaiveBayesMultinomial

```
Time taken to build model: 0.05 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         287               95.6667 %
Kappa statistic                          0.9133
Mean absolute error                      0.0441
Root mean squared error                  0.2014
Relative absolute error                  8.8115 %
Root relative squared error             40.2887 %
Total Number of Instances              300

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,960    0,047    0,954      0,960    0,957      0,913    0,979     0,967     Programming
                 0,953    0,040    0,960      0,953    0,957      0,913    0,980     0,974     Games
Weighted Avg.    0,957    0,043    0,957      0,957    0,957      0,913    0,979     0,970

=== Confusion Matrix ===

   a   b   <-- classified as
 144   6 |   a = Programming
   7 143 |   b = Games
```

They have different correctly classified instances percentage.NaiveBayes is 92.6667 % and NaiveBayesMultinominal is 95.6667 % which means NaiveBayesMultinominal classifier is slightly better than NaiveBayes classifier. It is better because NaiveBayesMultinominal has more complicated algorithm.

Assignment#4:

Classification of Iris dataset in the Weka tool.

- Classify the Iris dataset in Weka using the algorithm k-Nearest Neighbor (lazy/IBk), Decision Trees (trees/J48) and Naïve Bayes (bayes/NaiveBayes)
- The Iris dataset can be found in the data folder in the Weka installation or can be downloaded from the Datasets page
- Which algorithm gives the best result?

SOLUTİON:

k-Nearest Neighbor

```
Correctly Classified Instances         143               95.3333 %
Kappa statistic                          0.93
Mean absolute error                      0.0399
Root mean squared error                  0.1747
Relative absolute error                  8.9763 %
Root relative squared error             37.0695 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Iris-setosa
                 0,940    0,040    0,922      0,940    0,931      0,896    0,952     0,887     Iris-versicolor
                 0,920    0,030    0,939      0,920    0,929      0,895    0,947     0,894     Iris-virginica
Weighted Avg.    0,953    0,023    0,953      0,953    0,953      0,930    0,966     0,927

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

Decision Trees

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         144               96      %
Kappa statistic                          0.94
Mean absolute error                      0.035
Root mean squared error                  0.1586
Relative absolute error                  7.8705 %
Root relative squared error             33.6353 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,980    0,000    1,000      0,980    0,990      0,985    0,990     0,987     Iris-setosa
                 0,940    0,030    0,940      0,940    0,940      0,910    0,952     0,880     Iris-versicolor
                 0,960    0,030    0,941      0,960    0,950      0,925    0,961     0,905     Iris-virginica
Weighted Avg.    0,960    0,020    0,960      0,960    0,960      0,940    0,968     0,924

=== Confusion Matrix ===

  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  2 48 |  c = Iris-virginica
```

Naïve Bayes

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         144               96      %
Kappa statistic                          0.94
Mean absolute error                      0.0342
Root mean squared error                  0.155
Relative absolute error                  7.6997 %
Root relative squared error             32.8794 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Iris-setosa
               0,960    0,040    0,923      0,960    0,941      0,911    0,992     0,983     Iris-versicolor
               0,920    0,020    0,958      0,920    0,939      0,910    0,992     0,986     Iris-virginica
Weighted Avg.  0,960    0,020    0,960      0,960    0,960      0,940    0,994     0,989

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 48  2 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

→ I used 10 folds Cross-Validation test option and I saw that k-Nearest Neighbor has 95.3333% and Decision Trees, Naïve Bayes have the same percentage which is 96 % . We can say that Decision Trees and Naïve Bayes better than k-Nearest Neighbor.

(Optional)Assignment#5:

Write Java code for classifying the Iris data using the Weka.jar library.

- I did this optional homework to have a small idea about using java for machine learning algorithms. I didn't do homework 3 and 8 but I did homework 9.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.Reader;
import java.util.Random;

import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.Evaluation;
import weka.core.Instances;

public class iris {

	public static void main(String[] args) throws Exception {

	BufferedReader breader = null;
	breader = new BufferedReader(new FileReader("/Program Files/Weka-3-
8/data/iris.arff"));

	Instances train = new Instances (breader);
	train.setClassIndex(train.numAttributes() -1);

	breader.close();

	NaiveBayes nB = new NaiveBayes();
	nB.buildClassifier(train);
	Evaluation eval = new Evaluation(train);
	eval.crossValidateModel (nB,train,10,new Random(1));
	System.out.println(eval.toSummaryString("\nResults\n======\n",true));
	 System.out.println(eval.fMeasure(1) + " " + eval.precision(1) + " " +
eval.recall(1));

	}

}
```

SOLUTİON:

I wrote the following code to classify the iris data with Naive Bayes Classifier and I wrote the code of 10 folds Cross-Validation test option .I also added the output of the code below . We have %96 correctly classified instances.

OUTPUT:

```
Results
======

Correctly Classified Instances         144               96      %
Kappa statistic                          0.94
K&B Relative Info Score                 93.2874 %
K&B Information Score                  221.7855 bits      1.4786 bits/instance
Class complexity | order 0             237.7444 bits      1.585  bits/instance
Class complexity | scheme               28.5868 bits      0.1906 bits/instance
Complexity improvement     (Sf)        209.1576 bits      1.3944 bits/instance
Mean absolute error                      0.0342
Root mean squared error                  0.155
Relative absolute error                  7.6997 %
Root relative squared error             32.8794 %
Total Number of Instances              150
```

Assignment#6:

Classification of Iris dataset in Scikit.

- Classify the Iris dataset in Scikit using the k-Nearest Neighbour algorithm

- Experiment with different values for k. Which setting gives the best accuracy?

- Test and compare the results when using a Decision Tree classifier instead. Which gives the best accuracy.

SOLUTİON:

K Nearest Neighbor(KNN) is a very simple, easy to understand, versatile and one of the topmost machine learning algorithms. KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. In Credit ratings, financial institutes will predict the credit rating of customers. In loan disbursement, banking institutes will predict whether the loan is safe or risky. In political science, classifying potential voters in two classes will vote or won't vote. KNN algorithm used for both classification and regression problems. KNN algorithm based on feature similarity approach.

I wrote the following code to classify the iris data with k-Nearest Neighbour algorithm with Scikit library in Python.I added comments for explaining my code.

CODE:

```
#Let's first load the required      dataset from scikit-learn datasets.
from sklearn.datasets import load_iris
#Load dataset
iris = load_iris()
#After I have loaded the dataset, I want to know a little bit more about it. You can check feature and target names.
print(iris.feature_names)# print the names of the features
print(iris.target_names)# print the label species(class_0, class_1, class_2)
print(iris.data[0:5])# print the iris data (top 5 records)
print(iris.target)# print the iris labels (0:Class_0, 1:Class_1, 2:Class_3)
print(iris.data.shape)# print data(feature)shape
print(iris.target.shape)# print target(or label)shape
#To understand model performance, dividing the dataset into a training set and a test set is a good strategy.
#Let's split dataset by using function train_test_split(). We need to pass 3 parameters features, target, and test_set size. Additionally, we can use random_state to select records randomly.
from sklearn.model_selection import train_test_split# Import train_test_split function
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3) #  Split dataset into training set and test set,70% training and 30% test
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier
#Create KNN Classifier and we will change it to see the difference
knn = KNeighborsClassifier(n_neighbors=3)
#Train the model using the training sets
knn.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = knn.predict(X_test)
print(knn)
#Import scikit-learn metrics module for accuracy calculation,let's estimate, how accurately the classifier or model can predict the type of cultivars.
#Accuracy can be computed by comparing actual test set values and predicted values.
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

k=3

First,I used k=3,later I increased k=7.You can see the two different output below.

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
(150, 4)
(150,)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
Accuracy: 0.9333333333333333
```

k=7

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
(150, 4)
(150,)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
Accuracy: 0.9555555555555556
```

Here, I have increased the number of neighbors in the model and accuracy got increased. So,we decided that higher k number , better accuracy. However, we can’t say that when you increase the k number you will get always better accuracy.

Decision Tree classifier : As a marketing manager, you want a set of customers who are most likely to purchase your product. This is how you can save your marketing budget by finding your audience. As a loan manager, you need to identify risky loan applications to achieve a lower loan default rate. This process of classifying customers into a group of potential and non-potential customers or safe or risky loan applications is known as a classification problem. Classification is a two-step process, learning step and prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret. It can be utilized for both classification and regression kind of problem.

Decision Tree classifier Algorithm:

```
#Let's first load the required      dataset from scikit-learn datasets.
from sklearn.datasets import load_iris
#Load dataset
iris = load_iris()
from sklearn.model_selection import train_test_split# Import train_test_split function
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3) #  Split dataset into training set and test set,70% training and 30% test
from sklearn.tree import DecisionTreeClassifier
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output: `Accuracy: 0.9777777777777777`

Therefore,we can say that decision treee classifier is better than k-Nearest Neighbour classifier.

Assignment#7:

Classification of Iris dataset in R

• Classify the Iris dataset in R using Decision Trees (CART) and k-Nearest Neighbour (kNN) algorithms

• Which algorithm gives the best result?

• Does the result from kNN match the results from Scikit and Weka? If not, what are the reasons for the differences in result?

Decision Trees Algorithm in R Tool

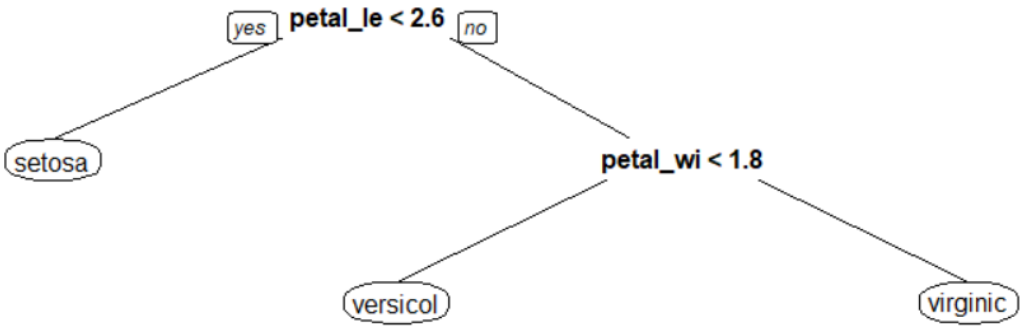
```
#İmporting necessary libraries
library(caTools)
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
library(VGAM)
iris = read.csv("iris.csv",header = F) #Loading required dataset
names = c("sepal_length","sepal_width", "petal_length","petal_width"
,"class")
names(iris) = names
# To create training and testing data set,we'll be using library caT
ools.Here,65% of original dataset will act as training dataset while
remaining 35% will be testing dataset.
split = sample.split(iris$class,SplitRatio = 0.65)
train = subset(iris,split == T)
test = subset(iris,split == F)
#Decision Trees
tree = rpart(class ~ . ,data = train,method = "class")
prp(tree)
preds = predict(tree, newdata = test,type = "class")
#To create confusion matrix
table(test$class,preds)
#Accuracy of model is given by
((18+16+16)/nrow(test))*100
```

SOLUTİON:

You can see the code which I wrote for Decision Trees.I also added comments to explain my codes.

Output

```
            preds
             setosa versicolor virginica
  setosa         18          0         0
  versicolor      0         15         3
  virginica       0          1        17
> ((18+16+16)/nrow(test))*100
[1] 92.59259
```



Later,I wrote code for k-Nearest Neighbour classification.You can see the code and output of the code.

k-Nearest Neighbour Algorithm

```
library(knitr)
library(class)
# Normalization of all columns except Species
dataNorm <- iris
dataNorm[, -5] <- scale(iris[, -5])
set.seed(1234)

# 70% train and 30% test
ind <- sample(2, nrow(dataNorm), replace=TRUE, prob=c(0.7,0.3))
trainData <- dataNorm[ind==1,]
testData <- dataNorm[ind==2,]
# Execution of k-NN with k=3
KnnTestPrediction_k1 <- knn(trainData[,-5], testData[,-5], trainData$
Species, k=3, prob=TRUE)
# Confusion matrix of KnnTestPrediction_k1
table(testData$Species, KnnTestPrediction_k1)
# Classification accuracy of KnnTestPrediction_k1
sum(KnnTestPrediction_k1==testData$Species)/length(testData$Species)*
100
```

Output:

```
            KnnTestPrediction_k1
             setosa versicolor virginica
setosa           10          0         0
versicolor        0         12         0
virginica         0          2        14
```

```
[1] 94.73684
```

Therefore,we can see that k-Nearest Neighbour classification gives better accuracy than Decision Trees classification.

We classify our iris dataset with k-Nearest Neighbour in Weka,Scikit and R tools.For all platform our accuracy is around %95 so, we can say that our results is matching.

Assignment#9:

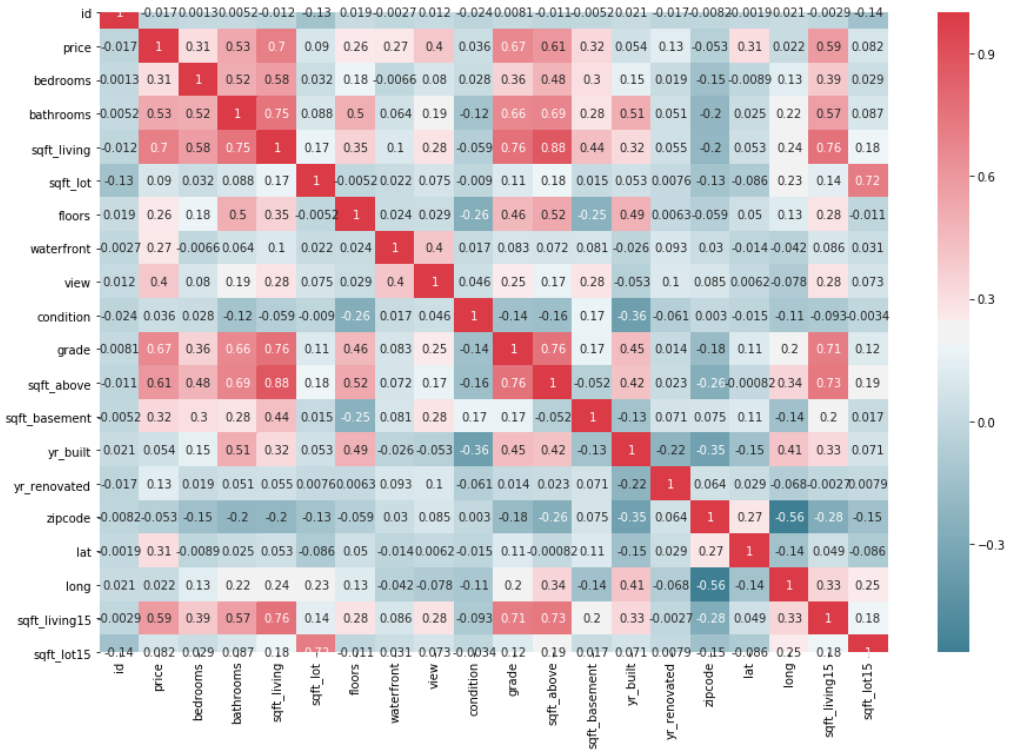
Utilize at least 5 linear regression models available in Scikit-learn to predict house price using regression based on the given dataset.

- Download House Sales in King Country
https://www.kaggle.com/harlfoxem/housesalesprediction

- Utilize linear regression models available in Scikit-learn.

1. Simple Linear Regression

Linear regression models are used to show or predict the relationship between two variables or factors. The factor that is being predicted (the factor that the equation solves for) is called the dependent variable. The factors that are used to predict the value of the dependent variable are called the independent variables. For linear Regression, we are using linear_model from sklearn library. I imported train_test_split. This splits the data into required ratio of which a certain ratio is considered for training the data, and the rest for testing the data. The data is trained to predict a line and then the test data is used to see if the line fits perfectly or not. I added to my code heat map of the dataset to have information about dataset and comments to show information about code. You can see my code,heat map ,linear regression model graph and output of the code below.



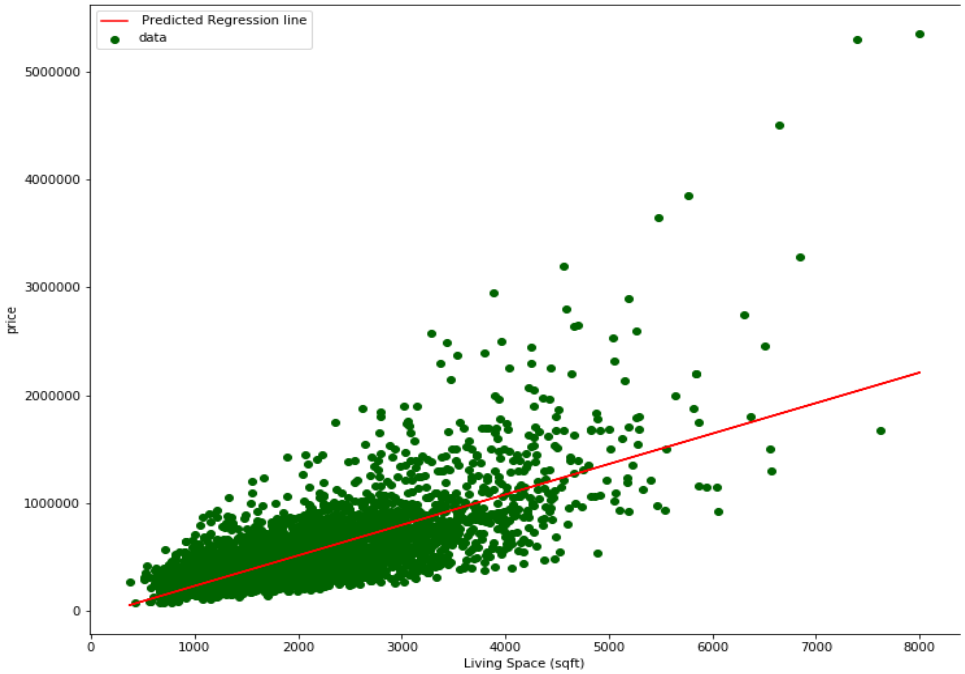
Heat Map of the Dataset

```python
#importing necessary library for loading data,plotting
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#loading data
df = pd.read_csv("C:/Program Files/Weka-3-8/data/kc_house_data.csv")

#Heatmap code
def correlation_heatmap(df1):
    _,ax=plt.subplots(figsize=(15,10))
    colormap=sns.diverging_palette(220,10,as_cmap=True)
    sns.heatmap(df.corr(),annot=True,cmap=colormap)
correlation_heatmap(df)

#importing library for linear regression models
from sklearn.model_selection import train_test_split #training the data, and the rest for testing the data.
from sklearn import linear_model #linear models
#In KNeighborsRegressor the target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.
from sklearn.neighbors import KNeighborsRegressor
#PolynomialFeatures generates a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics #The metrics is imported as the metric module implements functions assessing prediction error for specific purposes.
from mpl_toolkits.mplot3d import Axes3D
train_data,test_data=train_test_split(df,train_size=0.8,random_state=3) #Training and Testing data %80-%20
reg=linear_model.LinearRegression() # Applying simple linear regression model for prince vs living space
x_train=np.array(train_data['sqft_living']).reshape(-1,1)
y_train=np.array(train_data['price']).reshape(-1,1)
reg.fit(x_train,y_train)
# Code for finding Sqaured mean error,R squared training,R sqaured testing
x_test=np.array(test_data['sqft_living']).reshape(-1,1)
y_test=np.array(test_data['price']).reshape(-1,1)
pred=reg.predict(x_test)
print('linear model')
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('Sqaured mean error', round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(x_train,y_train),3))
print('R sqaured testing',round(reg.score(x_test,y_test),3) )
print('intercept',reg.intercept_)
print('coefficient',reg.coef_)

# Code for linear regression model graph
ax = plt.subplots(figsize= (12, 10))
plt.scatter(x_test, y_test, color= 'darkgreen', label = 'data')
plt.plot(x_test, reg.predict(x_test), color='red', label= ' Predicted Regression line')
plt.xlabel('Living Space (sqft)')
plt.ylabel('price')
plt.legend()
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```



Simple Linear Regression Model of Price vs Living Space

OUTPUT

```
linear model
Sqaured mean error 254289.15
R squared training 0.492
R sqaured testing 0.496
intercept [-47235.8113029]
coefficient [[282.2468152]]
```

2. Multiple Linear Regression

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

First I wrote the code of the boxplot is plotted for 'grade', 'bedrooms' and 'bathrooms' with respective to 'price'. So, now the features we are considering are 'bedrooms', 'grade', 'sqft_living' and 'sqft_above'. These are considered are one feature namely features1. Now the data is fitted into the model and test_data of features1 are used for prediction. Mean squared error is calculated for y_test. The mean squared error is rounded of upto 2 decimals. R squared error for both training and test is calculated. The intercept of the line is calculated along with coefficient of individual feature.

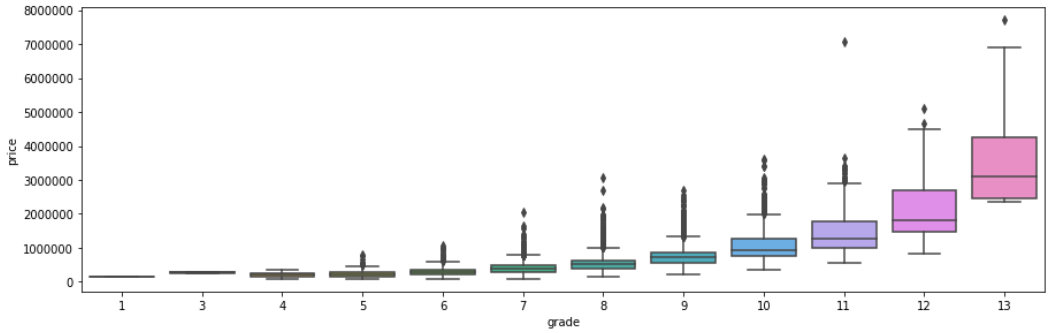
CODE:

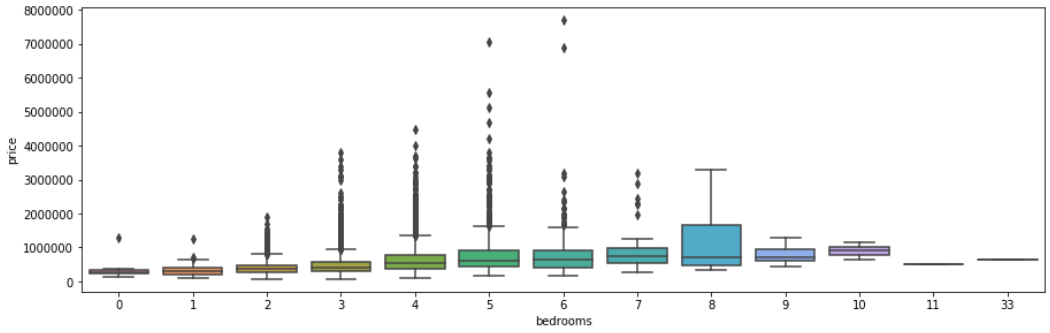
```
#importing necessary library for loading data,plotting
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from mpl_toolkits.mplot3d import Axes3D
# importing data set
df = pd.read_csv("C:/Program Files/Weka-3-8/data/kc_house_data.csv")
#Boxplots
fig,ax=plt.subplots(2,1,figsize=(15,10))
sns.boxplot(x=train_data['grade'],y=train_data['price'],ax=ax[0])
sns.boxplot(x=train_data['bedrooms'],y=train_data['price'],ax=ax[1])
_ , axes = plt.subplots(1, 1, figsize=(15,10))
sns.boxplot(x=train_data['bathrooms'],y=train_data['price'])
#Multiple Linear regression
features1=['bedrooms','grade','sqft_living','sqft_above']
reg=linear_model.LinearRegression()
reg.fit(train_data[features1],train_data['price'])
pred=reg.predict(test_data[features1])
print('complex_model 1')
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('mean squared error(MSE)', round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(train_data[features1],train_data['price']),3))
print('R squared training', round(reg.score(test_data[features1],test_data['price']),3))
print('Intercept: ', reg.intercept_)
print('Coefficient:', reg.coef_)
```

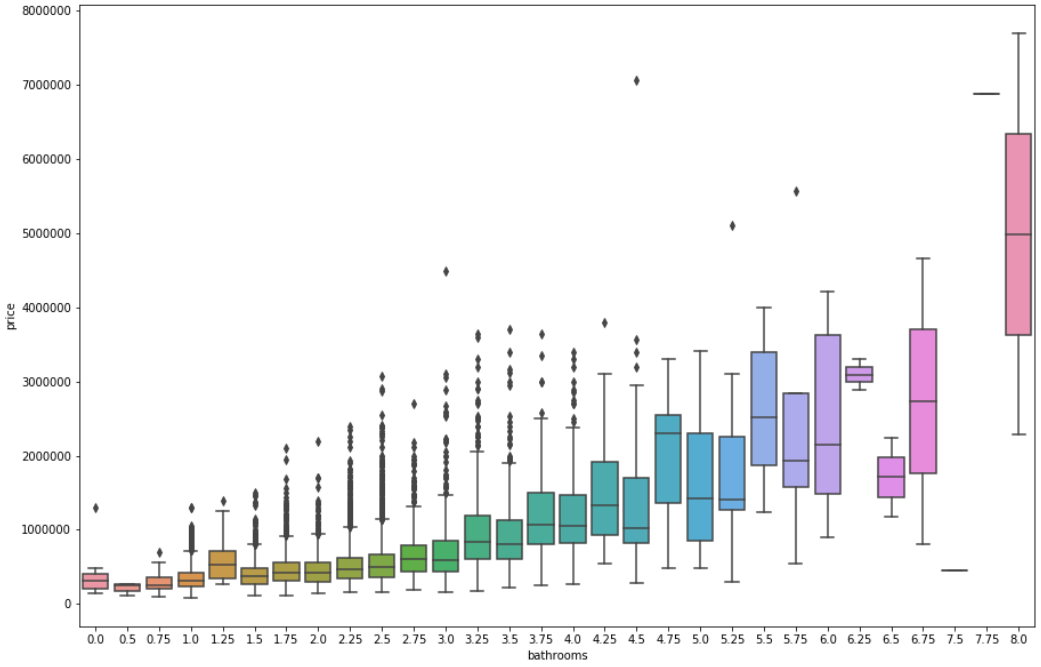
Output

```
complex_model 1
mean squared error(MSE) 239014.4
R squared training 0.548
R squared training 0.555
Intercept:  -523645.7841468266
Coefficient: [-4.33050242e+04  1.03455986e+05  2.73023590e+02 -8.38875593e+01]
```

The boxplots are plotted for ‘grade’, ‘bedrooms’ and ‘bathrooms’ with respective to ‘price’.







3. Polynomial Regression

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x).

First,for degree=2, the linear modelis built. the mean squared error is calculated and r squared is found for training and testing.

Then, for degree=3, the linear model is built. The mean squared error is calculated and r squared is found for training and testing.

CODE:

```
#importing necessary library for loading data,plotting
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from mpl_toolkits.mplot3d import Axes3D
# importing data set
df = pd.read_csv("C:/Program Files/Weka-3-8/data/kc_house_data.csv")
#For degree=2, the linear modelis built
polyfeat=PolynomialFeatures(degree=2)
xtrain_poly=polyfeat.fit_transform(train_data[features1])
xtest_poly=polyfeat.fit_transform(test_data[features1])
poly=linear_model.LinearRegression()
poly.fit(xtrain_poly,train_data['price'])
polypred=poly.predict(xtest_poly)
print('Complex Model_3')
mean_squared_error = metrics.mean_squared_error(test_data['price'], polypred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(poly.score(xtrain_poly, train_data['price']), 3))
print('R-squared (testing) ', round(poly.score(xtest_poly, test_data['price']), 3))
#For degree=3
polyfeat=PolynomialFeatures(degree=3)
xtrain_poly=polyfeat.fit_transform(train_data[features1])
xtest_poly=polyfeat.fit_transform(test_data[features1])
poly=linear_model.LinearRegression()
poly.fit(xtrain_poly,train_data['price'])
polypred=poly.predict(xtest_poly)
print('complex model_4')
mean_squared_error=metrics.mean_squared_error(test_data['price'],polypred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(poly.score(xtrain_poly, train_data['price']), 3))
print('R-squared (testing) ', round(poly.score(xtest_poly, test_data['price']), 3))
```

OUTPUT:

```
Complex Model_3
Mean Squared Error (MSE)  221965.07
R-squared (training)  0.614
R-squared (testing)  0.616
complex model_4
Mean Squared Error (MSE)  226057.06
R-squared (training)  0.627
R-squared (testing)  0.602
```

Complex Model_3 gives us R-squared (testing) score of 0.616. From above reports, we can say that at Polynomial regression for degree=2, is better than degree=3.

4. Decision Trees for Regression

Decision tree regression is similar to decision tree classification,however uses Mean Squared Error.In statistics, the mean squared error (MSE) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is actually estimated.

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

The Mean Squared Error is given by:

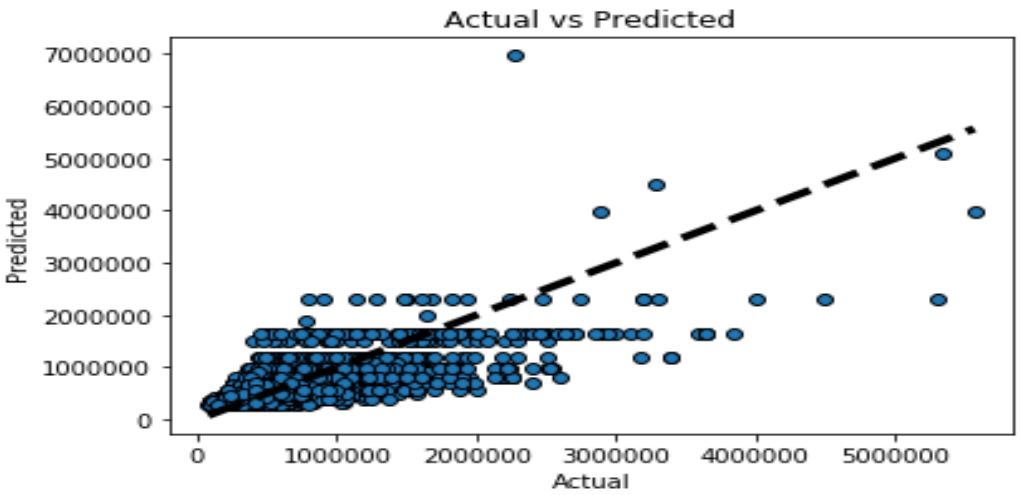
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

To predict the median prices of homes located King Country area when other attributes of the house are given.

You can find my code with the definitions and outputs below.

CODE:

```
#Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
#loading data
data = pd.read_csv("C:/Program Files/Weka-3-8/data/kc_house_data.csv")
#target variables
y = data['price']
X = data[['bedrooms','bathrooms','sqft_living','sqft_lot','sqft_lot15']]
#train test split
x_training_set, x_test_set, y_training_set, y_test_set = train_test_split(X,y,test_size=0.3,random_state=42,
shuffle=True)
#Training/model fitting
model =  DecisionTreeRegressor(max_depth=5,random_state=0)
model.fit(x_training_set, y_training_set)
#Model parameters study
from sklearn.metrics import mean_squared_error, r2_score
model_score = model.score(x_training_set,y_training_set)
# Have a look at R sq to give an idea of the fit
# Explained variance score: 1 is perfect prediction
print(' coefficient of determination R^2 of the prediction.: ',model_score)
y_predicted = model.predict(x_test_set)
# The mean squared error
print("Mean squared error: %.2f"% mean_squared_error(y_test_set, y_predicted))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(y_test_set, y_predicted))
# So let's run the model against the test data
#Visualization
from sklearn.model_selection import cross_val_predict
fig, ax = plt.subplots()
ax.scatter(y_test_set, y_predicted, edgecolors=(0, 0, 0))
ax.plot([y_test_set.min(), y_test_set.max()], [y_test_set.min(), y_test_set.max()], 'k--', lw=4)
ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
ax.set_title("Actual vs Predicted")
plt.show()
```



OUTPUT:

```
Coefficient of determination R^2 of the prediction.:  0.5971178148090541
Mean squared error: 70000653979.55
Test Variance score: 0.52
```

5. Random Forest Regression

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

CODE:

```
#Import pandas library and read the CSV file.
import pandas as pd
import numpy as np
dataset = pd.read_csv("C:/Program Files/Weka-3-8/data/kc_house_data.csv") # loading data
y = dataset['price']
X = dataset[['bedrooms','bathrooms','sqft_living','sqft_lot','sqft_lot15']]
from sklearn.model_selection import train_test_split #train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.ensemble import RandomForestRegressor # import random forest regressor
regressor = RandomForestRegressor(n_estimators=40, random_state=0)#The n_estimators parameter defines the number of trees in the random forest.
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
from sklearn import metrics #Evaluating the random forest regressor model algorithm using the error metrics.
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

OUTPUT for n=20

```
Mean Absolute Error: 158213.22880815523
Mean Squared Error: 63144760747.80991
Root Mean Squared Error: 251286.2128088406
```

OUTPUT for n=40

```
Mean Absolute Error: 156526.4510229116
Mean Squared Error: 61455470979.348045
Root Mean Squared Error: 247902.13992490675
```

OUTPUT for n=60

```
Mean Absolute Error: 155951.5432862072
Mean Squared Error: 60955114470.99541
Root Mean Squared Error: 246890.89588519745
```

OUTPUT for n=80

```
Mean Absolute Error: 155526.80004051427
Mean Squared Error: 61014567343.54112
Root Mean Squared Error: 247011.26966910055
```

OUTPUT for n=100

```
Mean Absolute Error: 155384.0483186831
Mean Squared Error: 61211700152.67589
Root Mean Squared Error: 247409.98393895887
```

To get a better model,we can try different tree size using the n_estimators parameter and compute the error metrics. I should take the model with lower RMSE value. In this, I tried various tree size and 60 tree size gives me a better result.

In conclusion, I created a machine learning regression models such as linear regression, multiple linear regression, polynomial regression, decision trees regression, random forest regression using the python with sklearn libraries.