1920-CT4101 Machine Learning

Assignment 3

Sampritha Hassan Manjunath
1MAI
19232922

ML Package selection:

Scikit Learn:

For completing this assignment, I choose Scikit Learn ML package.

Package description:

- Scikit Learn is simple ML package written in Python that offers variety of ML algorithms.
- It is an open source package.
- Scikit Learn allows us to pass math arrays like NumPy arrays or pandas arrays to handle datasets directly into the algorithm.
- Provides different types of data pre-processing options like MinMaxScaler, StandardScaler etc.,
- Provides various feature selection methods. Also provides methods for test_train_split and Cross-Validation methods.
- It includes metrics like Confusion Metrix and Classification Metrix
- Most of the ML algorithms for classification, regression and clustering are available in Scikit Learn.

Reason for choosing Scikit Learn: Along with the above-mentioned features of Scikit Learn, it provides a flexible environment for coding. Since from Assignment 1 I have been using Scikit Learn, I can work more efficiently on this Package than any other (like Weka, Orange)

Data Preparation:

- Copied steel.txt from requirement specification to a **steelData.csv** file. **Added headers** to each column (in order as provided in assignment requirement) to the CSV.
- Read the CSV as Data Frame using pandas

```
# data loading
import pandas as pd
data = pd.read_csv('./steelData.csv')
```

- **Feature selection:** To determine the most relevant features which contributes to the prediction of the target, the following steps are taken. Features which contribute less than 10% in target prediction is dropped.

```
# data pre processing
from sklearn.preprocessing import MinMaxScaler
#Using Pearson Correlation
correlation = data.corr()
#Correlation with output variable
correlation_target = abs(correlation["tensile_strength"])
#Selecting features which are atleast 10% contributing in prediction
relevant_features = correlation_target[correlation_target>0.1]
# drop the features which do not contribute to predict the target
relevant_features_data = data.drop(['percent_chromium','manufacture_yea
r','percent_nickel','percent_manganese'], axis = 1)
```

Algorithm Description:

K Nearest Neighbour:

- KNN is the simplest yet efficient algorithm to pick when it comes to both classification and regression.
- When the target value is continuous numerical data, KNN regression can be used to predict the target.
- KNN algorithm uses the **'features similarity'** to predict the new data point i.e., it analyses each set of features and compares it against the new data point's feature.
- Based on the closed data points, the new data point is closer, target is predicted.
- To decide on the closest resemblance, **distance between each data point** in training set is calculated first [1]
    - Distance can be calculated by any of 3 methods (Euclidean Distance, Manhattan Distance, Hamming Distance)
- For the given K value, algorithm considers the K nearest data points of the new data point. The **average of K nearest points will then be the prediction of the new data point** [2].
- For the algorithm to predict more accurately, optimum K value is required. This can be obtained by calculating the validation error/R2/RMSE for different values of K.
- K values affect the fitting of the model [1]
    - Higher K value → Model Underfits, Lower K Value → Model Overfits

Support Vector Regression (SVR):

- SVM offers an algorithm which can be used to predict continuous data and is called as SVR (Support Vector Regression)
- The basic idea of the SVM is do **define a hyperplane** which has the maximum margin i.e., the margin should be as close as to the data point making it as far as from the hyperplane.
- The **two margins generated are then kept as a reference for prediction** (say +m and -m). Each data point which is next to +m margin are said to belong to a class and each data point which is after -m margin are said to not belong to the class.
- In case of Regression, **a margin of tolerance (epsilon)** [3] is set along with the above-mentioned parameters.
- Since the prediction has be numerical prediction, we calculate the value of target along with the error tolerance
- Aim is to ***try to fit the error within a certain threshold*** while predicting the value of new data point**.**
- In other words, if the error falls between the +m and -m, the prediction is said to be accurate and if the error falls out of either +m or -m, the predicted value is said to be inaccurate and the R2/RMSE is calculated based on these errors.

Algorithm Development:

K Nearest Neighbour:

Data standardization and test_train_split: Once the relevant features has been selected, now we need to scale the features to fit the model better. **MinMaxScaler** has been used to achieve this task. Later divide the whole data into Test and Train set in 40:60 ratio.

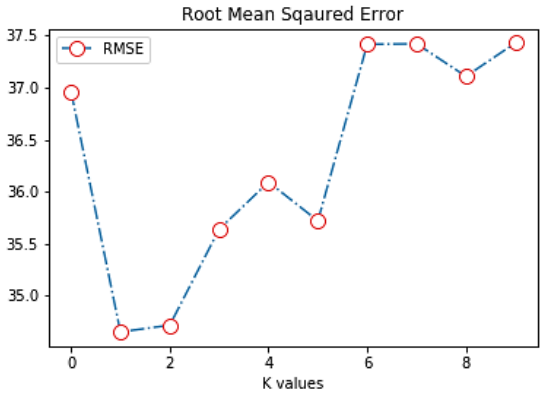
```
# data standardization
X = relevant_features_data.drop(['tensile_strength', 'sample_id'],axis=1)
scaler = MinMaxScaler(feature_range=(0, 1))
X = scaler.fit_transform(X)
y = relevant_features_data['tensile_strength'].values
# split data into test and train set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

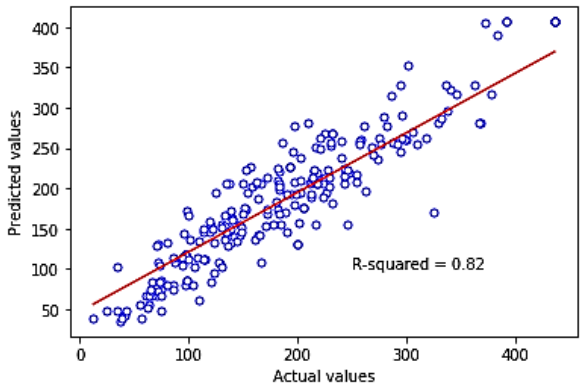
Evaluate Model for different values of K and calculate RMSE and R2 Error.

```
# calculate error rate for each k
r2_score_val = []
rmse_val = [] #to store rmse values for different k
rmse_val2 = []
for K in range(10):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)
    model.fit(X_train, y_train)  #fit the model
    pred=model.predict(X_test) #make prediction on test set
    y_pred=model.predict(X_train) #make prediction on train set
    rmse1 = sqrt(metrics.mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(rmse1)
    rmse2 = sqrt(metrics.mean_squared_error(y_train,pred)) #calculate rmse
    rmse_val2.append(rmse2)
    r2_score = metrics.r2_score(y_test, pred)
    r2_score_val.append(r2_score)
print("Average RMSE on test data: ", np.mean(rmse_val))
print("Average RMSE on train data: ", np.mean(rmse_val2))
print("Average R2 Score : ", np.mean(r2_score_val)))
```

```
Average RMSE on test data :  36.31496277883473
Average RMSE on train data :  29.852378537985135
Average R2 Score :  0.8229481220695462
```

Algorithm Analysis: It is observed that as the K value increases the model tends to overfit and as K value is decreased so less, the model tends to underfit.





Observing the RMSE values from the above graph, model is overfitting as K value go beyond 5 and model is underfitting as the K value falls below 2.

Optimal K value is = 2 (based on the RMSE curve)

Support Vector Regression:

No feature selection is done in SVR as it decreases the accuracy of the model by around 6%. So, we skip the feature selection step and move ahead with data standardization.

Kernel Selection: Out of available kernels (linear, poly, rbf, sigmoid), linear kernel is the one which gives comparatively less error. Hence, I have chosen linear kernel.

Data standardization and test_train_split: StandardScaler has been used to achieve this task (as the R2 score is much better by using StandardScaler than by using MinMaxScaler). Later divide the whole data into Test and Train set in 40:60 ratio.

```
# data standardization
scaler = StandardScaler()
X = data.drop(['tensile_strength', 'sample_id'],axis=1)
X = scaler.fit_transform(X)
y = data['tensile_strength'].values
# split test and train data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

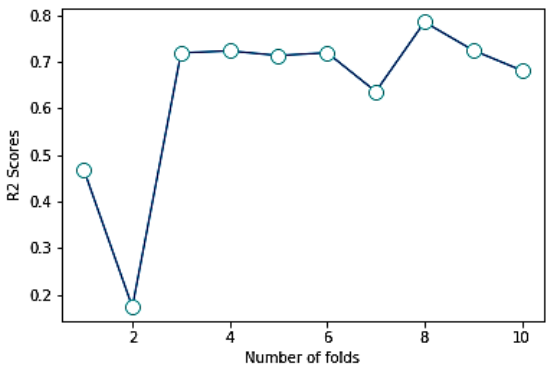
Cross Validation return the R2 score when using the regression algorithm. So below is the code snippet which shows r2 calculation

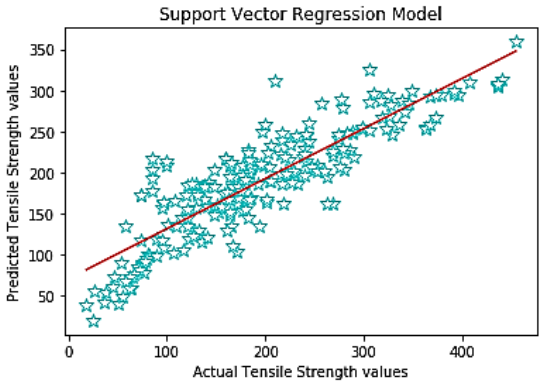
```
regressor = sklearn.svm.SVR(kernel = 'linear', gamma = 'auto')
# train model with cv of 10
cv_scores = sklearn.model_selection.cross_val_score(regressor, X, y, cv=10)
print('R2 Score : ' np.mean(cv_scores))
```

Evaluate the model to calculate RMSE Score. Plot the graph.

```
regressor.fit(X_train, y_train) #fit the model
y_pred = regressor.predict(X_test) #make prediction on test set
y2_pred = regressor.predict(X_train)
rmse_val = sqrt(metrics.mean_squared_error(y_test,y_pred)) #calculate rmse
rmse_val2 = sqrt(metrics.mean_squared_error(y_train,y2_pred))
print('RMSE value for test data = ', rmse_val)
print('RMSE value for train data = ', rmse_val2)
```

```
R2 Score :  0.6348296608021867
RMSE value for test data =  49.73129623482556
RMSE value for train data =  46.1206514119394
```





Conclusion:

After running KNN and SVR for the given dataset steelData.csv, below are the observations:

Sl. No	Algorithm	$RMSE_{test}$	$RMSE_{train}$	R2 Score
1	KNN	36.31	29.85	0.83
2	SVR	49.71	46.12	0.63

- KNN regression predicted the target accurately compared to SVR.
- Since the parameter settings in KNN (K value) is easily settable, algorithm can be evaluated to get highest accuracy and relatively low error.
- But when compared to error (RMSE), SVR performed better compared to KNN. Evaluation metrics is as below

If $RMSE_{test} > RMSE_{train}$ → Underfit on train data

If $RMSE_{test} < RMSE_{train}$ → Overfit on test data

If $RMSE_{test}$ approx(=) $RMSE_{train}$ → Model is accurate

---

References:

[1]: Lecture Notes ML_03_SimilarityBasedLearning_notes.pdf

[2]: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[3]: http://www.saedsayad.com/support_vector_machine_reg.htm?source=post_page-----8eb3acf6d0ff----------