MACHINE LEARNING 2

By Rishabh Jain

Introduction: ML package I have chosen for my regression task is **Linear Regression model** and **K-Nearest Neighbour regression model**. I am choosing **Scikit-learn library in Python**. This is because I am already familiar with Scikit-learn library.

Linear Regression is a predictive analysis algorithm which is used for finding a relationship between dependent variable and one or more independent predictors which are linear in nature. A simple linear regression equation is of form **$y = mx + b$** where y is the dependent variable, x is the independent variable, m is the slope and b is the intercept. It is a model based on supervised learning. If we plot independent variable on the x-axis and dependent variable on the y-axis, we can perform linear regression over this dataset to get the best fitted line for our linear regression model. This concept can me expended for multiple linear regression as **$y = b_0 + m_1b_1 + m_2b_2 + \dots m_nb_n$**. This is also an equation for the hyperplane.

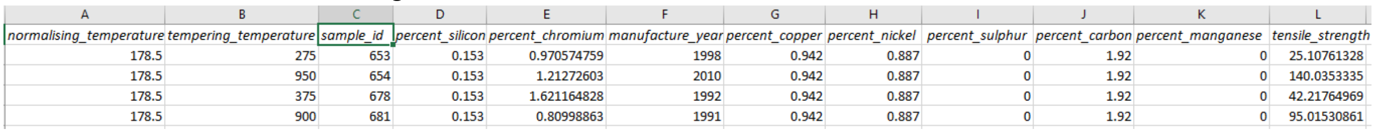
I am using linear regression for my regression task because linear regression is easy to implement and understand. It can be trained fast as compared to other algorithms. It is good for predicting data where Y is continuous in nature (like out strength data). It is good for solving problems with complex nature. Though it might not be the best algorithm to be used on data with real life scenarios. I want to try and compare it with KNN algorithm. Also, we need to calculate RMSE in the end for evaluation, that's another reason to use linear regression as it makes more sense to calculate mean squared error as the metric of loss in Linear regression model.

K-Nearest Neighbour is a supervised learning Machine learning model. Supervised learning models works when Data is already provided. It can be used for both classification and regression. Here we are using KNN for regression. KNN algorithm assumes that the similar data points are close to each other. KNN algorithm is based on similarity of data and classifying them accordingly into groups. In KNN algorithm, we initialize a K which is equal to the number of neighbours to that cluster. We calculate the distance between the data points and put them accordingly in an ordered fashion. Then we sort that ordered fashion in ascending order by calculation the distances between them. Then we pick first K entries and label them return the mean of the K labels for regression.

I am using K-Nearest Neighbour because it is easy to implement and a simple machine learning model. There are only few parameters to tune when it comes to KNN. I have also used KNN for classification for my first Machine Learning assignment, therefore I was curious to see how it can be used for regression as well. Choosing the correct K value is very important for KNN to give accurate predictions. KNN is used in recommender systems (e.g. Netflix, Spotify etc.)

Preparing the data for ML package:

1. Download the **"steel.txt"** file from blackboard.
2. Open excel and import the .txt file.
3. Use excel to **Delimit** the file as **"Characters such as commas or tabs separate each field".**
4. **Add the title** to each file as following:

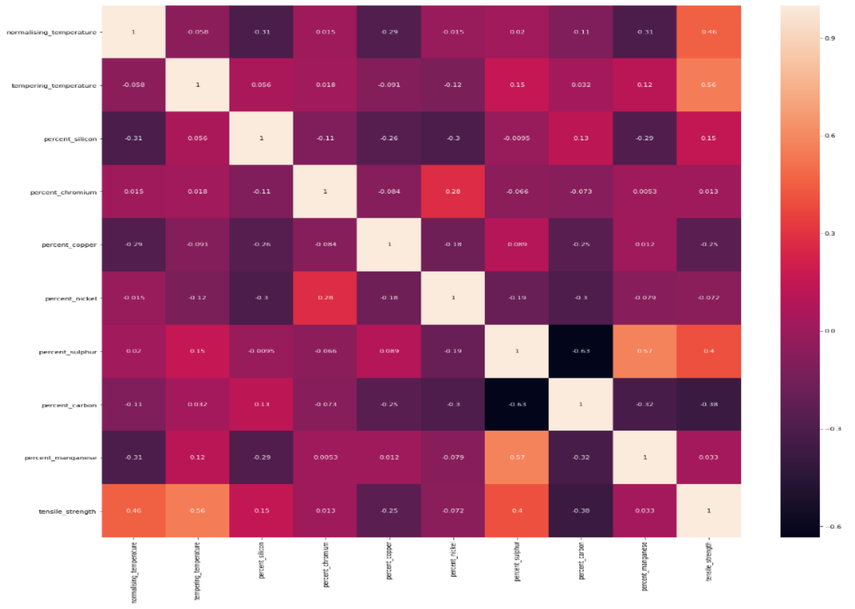


A	B	C	D	E	F	G	H	I	J	K	L
normalising_temperature	tempering_temperature	sample_id	percent_silicon	percent_chromium	manufacture_year	percent_copper	percent_nickel	percent_sulphur	percent_carbon	percent_manganese	tensile_strength
178.5	275	653	0.153	0.970574759	1998	0.942	0.887	0	1.92	0	25.10761328
178.5	950	654	0.153	1.21272603	2010	0.942	0.887	0	1.92	0	140.0353335
178.5	375	678	0.153	1.621164828	1992	0.942	0.887	0	1.92	0	42.21764969
178.5	900	681	0.153	0.80998863	1991	0.942	0.887	0	1.92	0	95.01530861

5. Save the file as **.CSV** file. The data preparation to input data in ML package is complete. I have saved the .CSV file as **steel.csv.**

CORRELATION BETWEEN X AND Y

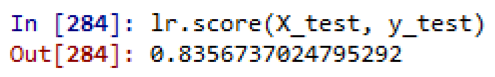
z = dataset.iloc[:, [0,1,3,4,6,7,8,9,10,11]]
#ALL VALUES OF X and Y except Manufacturing_Year and Sample_ID
plt.subplots(figsize=(20,20))
sns.heatmap(z.corr(),annot=True)
plt.show()



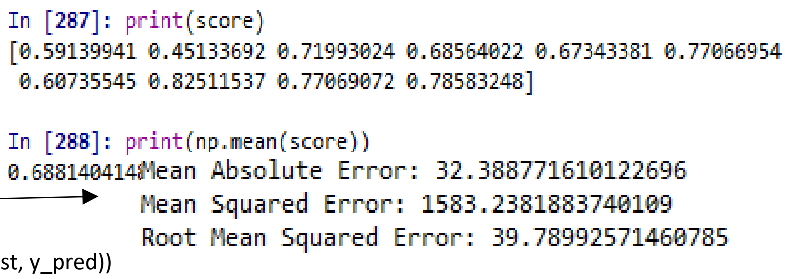
ALGORITHM 1 – LINEAR REGRESSION: Linear Regression is a predictive linear model. It is one of the most commonly used machine learning models and used in lots of business applications. It explains the relationship between a dependent variable(y) and one or more independent variable(x). It is used for finding a relationship between dependent variable and one or more independent predictors which are linear in nature.

CODE(PYTHON):

```
#####################################################################
# Importing the dataset
dataset = pd.read_csv('steel.csv')
X = dataset.iloc[:, [0,1,3,4,6,7,8,9,10]]  #Excluding Sample_ID and Manufacturing_year
y = dataset.iloc[:, 11].values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
#Training data contains 387 Values and Testing Data contains 166 data as test_size is 0.3
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
# Fit the classifier to the data
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
y_pred
lr.score(X_test, y_test)
# 10-fold Cross Validation
from sklearn.model_selection import cross_val_score
#train model with cv of 10
cv_scores = cross_val_score(lr, X, y, cv=10)
#print each cv score (accuracy) and average them
print(cv_scores)
print(np.mean(cv_scores))
#RMSE AND MAE
from sklearn import 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)))
#Plot between actual and predicted values
plot = sns.distplot(y_test, hist=False, color="r", label="Actual Value")
sns.distplot(y_pred, hist=False, color="b", label="Fitted Values" ,
ax=plot)
############################################################
```

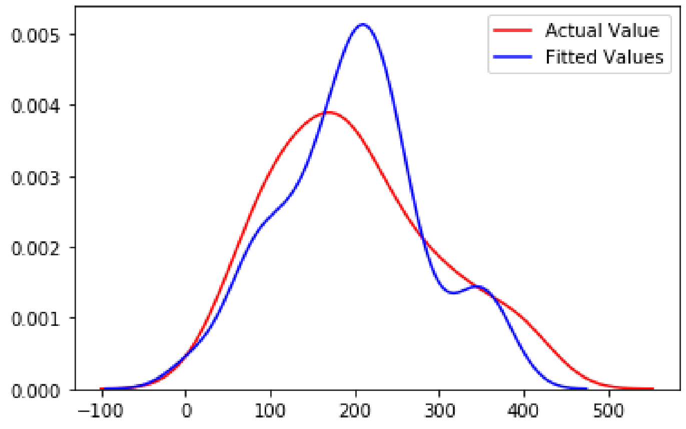


```
In [284]: lr.score(X_test, y_test)
Out[284]: 0.8356737024795292
```



```
In [287]: print(score)
[0.59139941 0.45133692 0.71993024 0.68564022 0.67343381 0.77066954
 0.60735545 0.82511537 0.77069072 0.78583248]

In [288]: print(np.mean(score))
0.688140414Mean Absolute Error: 32.388771610122696
           Mean Squared Error: 1583.2381883740109
           Root Mean Squared Error: 39.78992571460785
```

lr.predict predict the values for test data based on linear regression trained model. **lr.score** print the accuracy of the model which in our case is coming to be **83.5.** Mean aaccuracy after performing the 10-fold cross validation is coming to be **68.8%. Mean Absoluter Error** is **32.3%**, and **Root Mean Squared Error** is coming to be **39.7%**. Later we are plotting the actual values against fitted values.



Process for Developing Linear Regression Model and choosing the parameters: For Linear Regression Model, I am taking most values as default parameters. Test size for training and testing data is taken to be 0.3 as under this setting I was getting the best accuracy for my model. Apart from that I am performing cross validation to cover other test parameters as well. The parameters of the Linear regression Model are as follows:

`LinearRegression`(*fit_intercept=True*, *normalize=False*, *copy_X=True*, *n_jobs=None*)

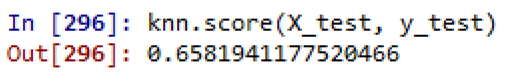
- Fit_intercept=True (by default) to calculate the intercept for this model.
- Normalize= False (by default) as there is no need for normalization.
- Copy_X = True (by default) as there is no need for overwriting X.
- N_jobs= Number of jobs for the computation which is none (by default) as we are not performing parallel backend job.

I tried and experimenting with different parameter settings for my linear regression model. By changing test size, by normalizing my data, by overwriting X values and changing the number of jobs for the computation. But mostly on the default setting itself, I got the best values among other values for my model. Hence above settings were my chosen parameter setting for the model.

ALGORITHM 2 – K NEAREST NEIGHBOUR (KNN): K-Nearest Neighbour is a supervised learning Machine learning model. Supervised learning models works when Data is already provided. It learns from Data given by training the model on test data. After the training is complete, the test data is inputted to predict the output values. This model works by taking a data point and looking for the K-closest neighbour to that data point. K can be any number from 1 to n. Accuracy of the model varies depending on the value of K. For my assignment, I am taking K as 3. After that, most of the data point are given a label and clustered accordingly.

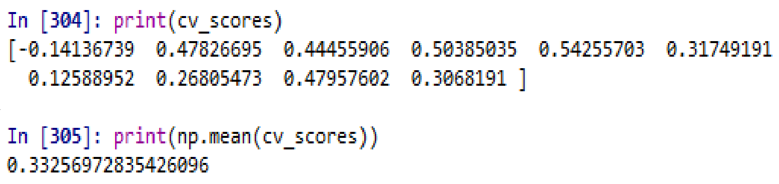
CODE(PYTHON):

```
########################################################
# Importing the dataset
dataset = pd.read_csv('steel.csv')
X = dataset.iloc[:, [0,1,3,4,6,7,8,9,10]]
y = dataset.iloc[:, 11].values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
#Training data contains 387 Values and Testing Data contains 166 data as test_size is 0.3
# Create KNN classifier taking K=3 for this dataset
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=3)
# Fit the classifier to the data
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
y_pred
knn.score(X_test, y_test)
```



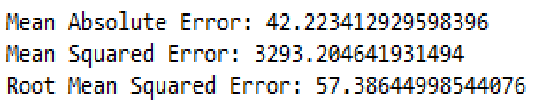
```
In [296]: knn.score(X_test, y_test)
Out[296]: 0.6581941177520466
```

```
# 10-fold Cross Validation
from sklearn.model_selection import cross_val_score
#train model with cv of 10
cv_scores = cross_val_score(knn, X, y, cv=10)
#print each cv score (accuracy) and average them
print(cv_scores)
print(np.mean(cv_scores))
```



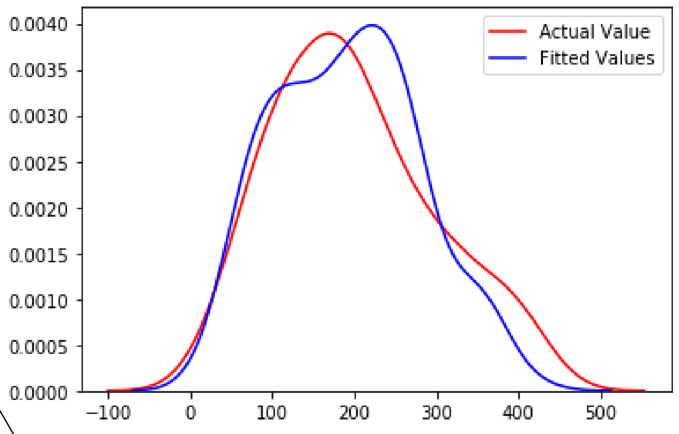
```
In [304]: print(cv_scores)
[-0.14136739  0.47826695  0.44455906  0.50385035  0.54255703  0.31749191
  0.12588952  0.26805473  0.47957602  0.3068191 ]

In [305]: print(np.mean(cv_scores))
0.33256972835426096
```

```
#RMSE AND MAE
from sklearn import 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)))
```

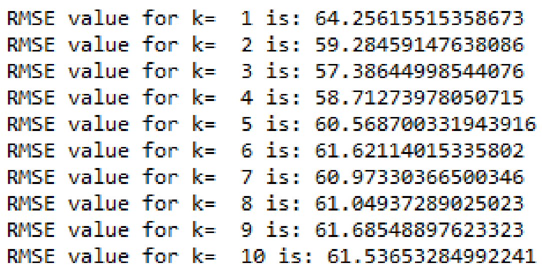


```
Mean Absolute Error: 42.223412929598396
Mean Squared Error: 3293.204641931494
Root Mean Squared Error: 57.38644998544076
```

```
#Plot between actual and predicted values
plot = sns.distplot(y_test, hist=False, color="r", label="Actual Value")
sns.distplot(y_pred, hist=False, color="b", label="Fitted Values" ,
ax=plot)
```



```
#Code to get best value of K
rmse_val = [] #to store rmse values for different k
for K in range(50):
   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
   error = sqrt(mean_squared_error(y_test,pred)) #RMSE
   rmse_val.append(error) #store rmse values
   print('RMSE value for k= ' , K , 'is:', error)
#####################################################################
```



```
RMSE value for k=  1 is: 64.25615515358673
RMSE value for k=  2 is: 59.28459147638086
RMSE value for k=  3 is: 57.38644998544076
RMSE value for k=  4 is: 58.71273978050715
RMSE value for k=  5 is: 60.568700331943916
RMSE value for k=  6 is: 61.62114015335802
RMSE value for k=  7 is: 60.97330366500346
RMSE value for k=  8 is: 61.04937289025023
RMSE value for k=  9 is: 61.68548897623323
RMSE value for k=  10 is: 61.53653284992241
```

knn.predict predict the values for test data based on knn trained model. **Knn.score** print the accuracy of the model which in our case is coming to be **65%.** Mean accuracy after performing the 10-fold cross validation is **33%. Mean Absoluter Error** is **42.2%**, and **Root Mean Squared Error** is **57.3%**. Later we are plotting the actual values against fitted values to see how accurate our prediction is in plot. We are also calculating the **RMSE values** for all the values of K from 1 to 50 to figure out the best K.

Process for Developing K-Nearest Neighbour Regression Model and choosing the parameters: For K-Nearest Neighbour Regression Model, I am taking most values as default parameters. Test size for training and testing data is taken to be

0.3 as under this setting I was getting the best accuracy for my model. Apart from that I am performing cross validation to cover other test parameters as well. I also calculated all the RMSE values for different values of K (1 to 50) to get the best value of K for my model. After calculating different RMSE, I chose K=3 for my model because at this value of K, I was getting the least error as compared to other values of K. At K = 3, RMSE is **57.3%.** The parameters of the Linear regression Model are as follows:

`KNeighborsRegressor(`*n_neighbors=3*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*`)`

- N_neighbours = 3, explained above as I am getting least RMSE at this value of K.
- Weights = 'uniform' (by default) as all points in the neighbourhood are weighted equally in this case and I don't need to calculate weight points by inverse of their distance.
- Algorithm='auto' (by default) as it will decide the best algorithm among BallTree, KDTree and Brute based on fit.
- Leaf_size=30 (by default) as it can affect the speed as well as memory required to store the Tree. So, I am taking optimum value by default.
- P=2 as we are using Euclidean distance for our Minkowski metric.
- Metric = 'minkowski' (by default), we can use Euclidean metric as well
- Metric_params = None (by default) as there are no additional parameter being used for metric.
- N_jobs= Number of jobs for the computation which is none (by default) as we are not performing parallel backend job.

I tried and experimenting with different parameter settings for my linear regression model. For selecting the parameters for my regression model, I choose test size as 0.3 for best case scenario. K=3 as it gave the least RMSE for this value. Weights are uniform by default and metric used is minkowski metric. Most values are used as default because these values were giving the best result in the form of accuracy for my model.

Underfitting and Overfitting Monitoring:

Underfitting means that training error rate in the model is too high while overfitting means that the result of error rate of model training is lower that the result of rate of testing dataset. To get the best value of training and testing dataset, I have chosen my test size as 0.3, to divide the training and test data in 70-30 ratio. I came to this conclusion by randomly selecting the different values of test size and calculating the accuracy of my model based on that test size. After that I started plotting the graph between actual and fitted values as seen above to compare both the data. I also performed 10- fold cross validation to avoid the problem of underfitting and overfitting of data and to get 10 different accuracy result and taking there mean for my result. There won't be much of overfitting as this is a small dataset. But to avoid underfitting, we have taken test size as 0.3.

Performance Metric and Conclusion:

From the results, we can see that KNN given the accuracy of **65%** and Linear Regression predicts an accuracy of **83%**. Based on this prediction we can say that; Linear Regression might be a better algorithm than Linear Regression for prediction the variety of our steel dataset. As the dataset contains only **553 values**, the model is trained on **387 values** and testing is done on **166 values**. It won't be a good criterion to access the model based on this single test. Therefore, later we performed a 10-fold Cross Validation on both our algorithm to get a different set of accuracies for different set of test data build by Cross Validation as **cv=10**. Taking mean of these accuracy for comparing both the algorithms, we found that **KNN** gave an average accuracy of **35%** after 10-fold cross validation while **Linear Regression** gave an average accuracy of **68.8%.** We still can't say for sure as both results are not great for comparison. Now, we are performing RMSE and MAE on our predicted and test values. MAE measures the overall magnitude of the errors in a set of predictions. It is the average of the absolute differences between predicted and actual values where all singular distances have equal weights. It doesn't consider the direction. RMSE is the square root of average of squared differences between actual and predicted values. RMSE is more useful when large errors are undesirable.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \quad \mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

For my model, I am performing both the metric for evaluation, RMSE AND MAE. The RMSE for Linear Regression is **39.7** and MAE for Linear Regression is **32.3**. While RMSE for KNN is **57.38** and MAE for KNN is **42.2**. Lower the value of error, better will be the model. As we can see RMSE and MAE are both low for Linear Regression and comparatively higher for KNN. Both the models give very different results in terms of accuracy, RMSE and MAE. This result is due to that fact that KNN is slower when we have a real-world scenario. We need to provide a proper scaling for fair treatment among features of KNN. Hyperparameters like K-value and Distance function also effect the model accuracy. Whereas Linear Regression can be used easily for real-world problems. It can also be easily implemented for space complex solution. It can perform well when there are large number of features as compared to KNN which might be the problem in our dataset. Given such large number of features might be difficult for KNN to make a good prediction model. Also, KNN is slower than Linear regression as Linear regression can easily get output values from the already tuned coefficients while KNN have to keep a track of all the training data and finding the neighbour node. Considering these factors, we can conclude that **Linear Regression is a better regression algorithm than KNN for our 'steel.txt' dataset.**

REFERENCES AND ACKOWLEDGEMENT:

- https://www.udemy.com/course/machinelearning/ (Machine Learning A-Z™: Hands-On Python & R in Data Science)
- https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a
- https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
- https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/
- https://www.statisticssolutions.com/what-is-linear-regression/
- https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86
- https://machinelearningmastery.com/linear-regression-for-machine-learning/
- https://datafai.com/2017/10/31/python-machine-learning-linear-regression-with-scikit-learn/
- https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d
- https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/
- https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7
- https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- https://d2l.ai/chapter_multilayer-perceptrons/underfit-overfit.html