Temperature Forecast Project using ML

Introduction :

Temperature prediction using three machine learning models - Multiple Linear Regression (MLR), Artificial Neural Network (ANN) and Support Vector Machine (SVM), through a comparative analysis using the weather data collected from Central Kerala during the period 2007 to 2015. The experimental results are evaluated using Mean Error (ME), Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Correlation Coefficients (CC). The error metrics and the CC shows that MLR is a more precise model for temperature prediction than ANN and SVM.

Data Pre-processing Done:

The purpose of preprocessing is to convert raw data into a form that fits machine learning. Structured and clean data allows a data scientist to get more precise results from an applied machine learning model. The technique includes data formatting, cleaning, and sampling.

- Hardware and Software Requirements and Tools Used
- **Tools: Python 3.8.5, Jupyter Notebook, Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, Scipy**
- **Techniques: regression, Linear Regression ,Decision Tree Regressor, Lasso, Ridge, SVR, KNeighborsRegressor, RandomForestRegressor**
- **Hardware: I3 processor, 4GB RAM**

- Model/s Development and Evaluation

  I have used the below models for regression:

  1. Linear Regression
  2. Decision Tree Regressor
  3. Lasso
  4. Ridge
  5. KNeighborsRegressor
  6. SVR
  7. RandomForestRegressor

- Identification of possible problem-solving approaches (methods)

- Read the data (from csv)
- Identify the dependent and independent variables.
- Check if the data has missing values or the data is categorical or not.
- If yes, apply basic data preprocessing operations to bring the data in a go to go format.
- Now split the data into the groups of training and testing for the respective purpose.
- After splitting data, fit it to a most suitable model. (How to find a suitable model is answered below)

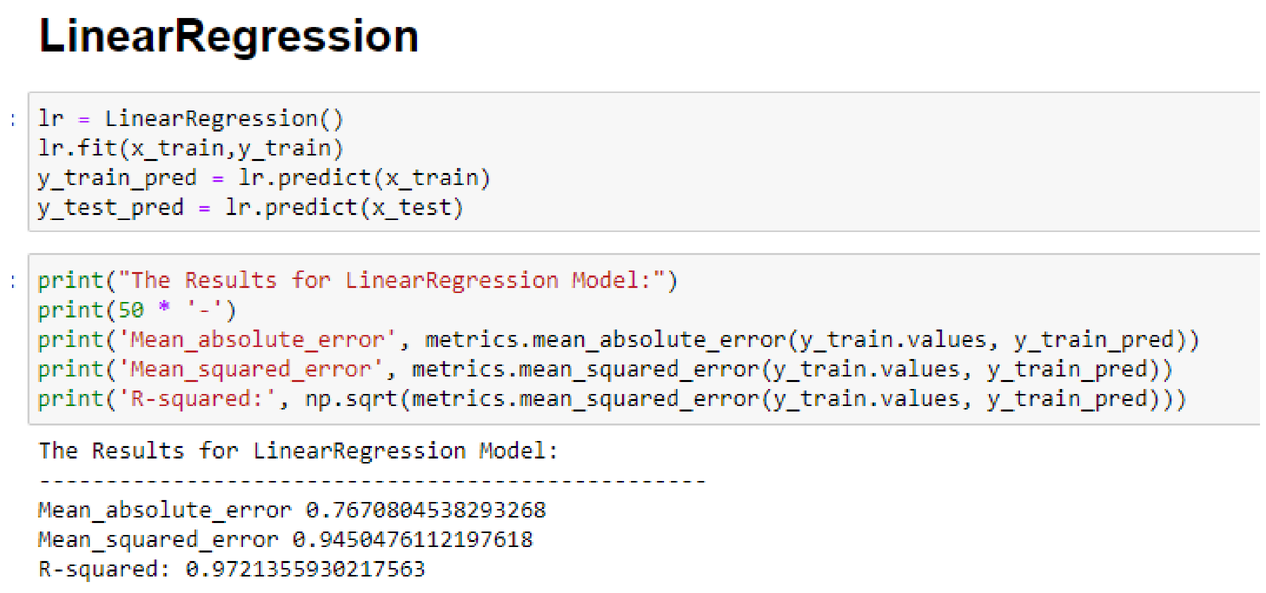
- Validate the model. If satisfactory, then go with it, else tune the parameters and keep testing. In a few cases, you can also try different algorithms for the same problem to understand the difference between the accuracies.
- From step 7 one can also learn about accuracy paradox.
- Visualize the data.

• Testing of Identified Approaches (Algorithms)

1. Cross validation
2. Linear Regression
3. Decision Tree Regression
4. r2_score
5. mean_squared_error, mean_absolute_error

• Run and Evaluate selected models

I have used the below models for regression:



LinearRegression

```
lr = LinearRegression()
lr.fit(x_train,y_train)
y_train_pred = lr.predict(x_train)
y_test_pred = lr.predict(x_test)
```

```
print("The Results for LinearRegression Model:")
print(50 * '-')
print('Mean_absolute_error', metrics.mean_absolute_error(y_train.values, y_train_pred))
print('Mean_squared_error', metrics.mean_squared_error(y_train.values, y_train_pred))
print('R-squared:', np.sqrt(metrics.mean_squared_error(y_train.values, y_train_pred)))
```

```
The Results for LinearRegression Model:
--------------------------------------------------
Mean_absolute_error 0.7670804538293268
Mean_squared_error 0.9450476112197618
R-squared: 0.9721355930217563
```

RandomForestRegressor

```
rdr = RandomForestRegressor()
rdr.fit(x_train,y_train)
y_train_pred = rdr.predict(x_train)
y_test_pred = rdr.predict(x_test)
```

```
print("The Results for Random Forest Regressor Model:")
print(50 * '-')
print('Mean_absolute_error', metrics.mean_absolute_error(y_train, y_train_pred))
print('Mean_squared_error', metrics.mean_squared_error(y_train, y_train_pred))
print('R-squared:', np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
```

```
The Results for Random Forest Regressor Model:
--------------------------------------------------
Mean_absolute_error 0.2101264344633356
Mean_squared_error 0.07854704004696006
R-squared: 0.28026244851381726
```

Ridge

```
]: rd = Ridge()
   rd.fit(x_train,y_train)
   y_train_pred = rd.predict(x_train)
   y_test_pred = rd.predict(x_test)
```

```
]: print("The Results for Ridge Model:")
   print(50 * '-')
   print('Mean_absolute_error', metrics.mean_absolute_error(y_train, y_train_pred))
   print('Mean_squared_error', metrics.mean_squared_error(y_train, y_train_pred))
   print('R-squared:', np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
```

```
The Results for Ridge Model:
--------------------------------------------------
Mean_absolute_error 0.7670475496994559
Mean_squared_error 0.9450536266180147
R-squared: 0.9721386869259009
```

Lasso

```
lss = Lasso()
lss.fit(x_train,y_train)
y_train_pred = lss.predict(x_train)
y_test_pred = lss.predict(x_test)
```

```
print("The Results for Lasso Model:")
print(50 * '-')
print('Mean_absolute_error', metrics.mean_absolute_error(y_train, y_train_pred))
print('Mean_squared_error', metrics.mean_squared_error(y_train, y_train_pred))
print('R-squared:', np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
```

```
The Results for Lasso Model:
--------------------------------------------------
Mean_absolute_error 0.8799626959200997
Mean_squared_error 1.2388876014669188
R-squared:  1.1130532788087544
```

Decision Tree Regressor

```
dtc = DecisionTreeRegressor()
dtc.fit(x_train,y_train)
y_train_pred = dtc.predict(x_train)
y_test_pred = dtc.predict(x_test)
```

```
print("The Results for Decision Tree Regressor Model:")
print(50 * '-')
print('Mean_absolute_error', metrics.mean_absolute_error(y_train, y_train_pred))
print('Mean_squared_error', metrics.mean_squared_error(y_train, y_train_pred))
print('R-squared:', np.sqrt(metrics.mean_squared_error(y_train, y_train_pred)))
```

```
The Results for Decision Tree Regressor Model:
--------------------------------------------------
Mean_absolute_error 3.142835543354664e-17
Mean_squared_error 1.116559482509652e-31
R-squared:  3.3414958963159776e-16
```

K Neighbors Regressor

```
knn = KNeighborsRegressor()
knn.fit(x_train,y_train)
y_train_pred = knn.predict(x_train)
y_test_pred = knn.predict(x_test)
```

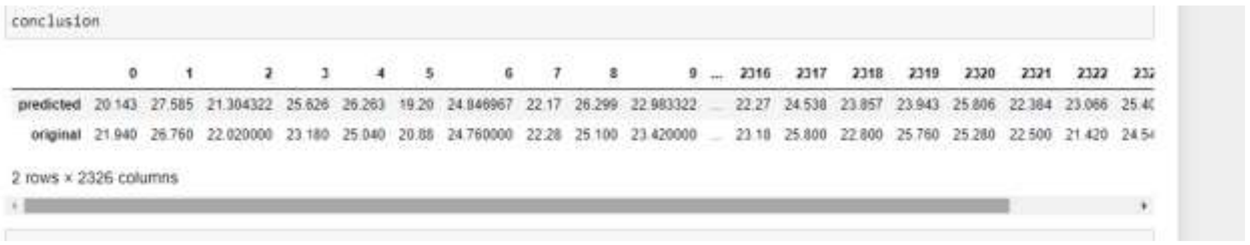
```
print("The Results for K Neighbors Regressor Model:")
print(50 * '-')
print('Mean_absolute_error', metrics.mean_absolute_error(y_train.values, y_train_pred))
print('Mean_squared_error', metrics.mean_squared_error(y_train.values, y_train_pred))
print('R-squared:', np.sqrt(metrics.mean_squared_error(y_train.values, y_train_pred)))
```

```
The Results for K Neighbors Regressor Model:
--------------------------------------------------
Mean_absolute_error 1.0606583958096996
Mean_squared_error 1.9147962928359366
R-squared: 1.3837616459621709
```

- ## Conclusion:

The five models produced adequate results on their own, averaging a relatively low RMSE for each dataset. Through this advance algorithm and including pruning and other measures fosr over fitting, we were able to as a Decision Tree Regressor that performed pretty well on their own. These shows that these machine learning techniques can be applied well to predicting sports data. By assembling multiple methods, we where able to create an even more accurate model. Decision Tree Regressor is the best model of this dataset, because

Decision Tree Regressor has better r2_score than linear regression model , lasso, ridge, Elastic Net for this dataset, r2_score is 82% it may also denote it is over fitting as it even classifies the outliers perfectly.



```
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	2316	2317	2318	2319	2320	2321	2322	232
predicted	20.143	27.585	21.304322	25.626	26.263	19.20	24.846967	22.17	26.299	22.983322	...	22.27	24.538	23.857	23.943	25.806	22.384	23.066	25.4
original	21.940	26.760	22.020000	23.180	25.040	20.88	24.760000	22.28	25.100	23.420000	...	23.18	25.800	22.800	25.760	25.280	22.500	21.420	24.5

2 rows × 2326 columns