## Introduction

In ODI format of cricket, many factors play a significant role in deciding what the final score will be in an innings. Following are some of the key factors:

* Number of wickets left
* Number of balls left
* On how much scores are the current batsman batting?
* How much the team had scored in last 5 overs?
* How much the team had lost wickets in last 5 overs?
* The nature of the pitch
* How strong is the batting and bowling team?

I analyzed a similar set of factors to predict final score of a team in an ODI cricket match.

## Dataset

I have downloaded the dataset from [cricsheet](https://cricsheet.org/downloads/). The site gives us ball by ball details of over 1000 ODI matches. Data contains 15 non-null columns/features which can be seen below:

* mid: Each match is given a unique number
* date: When the match happened
* venue: Stadium where match is being played
* bat\_team: Batting team name
* bowl\_team: Bowling team name
* batsman: Batsman name who faced that ball
* bowler: Bowler who bowled that ball
* runs: Total runs scored by team at that instance
* wickets: Total wickets fallen at that instance
* overs: Total overs bowled at that instance
* runs\_last\_5: Total runs scored in last 5 overs
* wickets\_last\_5: Total wickets that fell in last 5 overs
* striker: max (runs scored by striker, runs scored by non-striker)
* non-striker: min (runs scored by striker, runs scored by non-striker)
* total: Total runs scored by batting team after first innings

## Feature Selection

I carefully selected following features using Lasso tool from scikit-learn package:

* current score
* runs scored in last 5 overs
* striker’s score
* non-striker’s score

One can use a different combination of features and test the code on them.

**Label Used**: total

## Experimental Setup

Firstly, I scaled the features for better performance and then split the data into training and test sets with the ratio of 75/25.

Further, I used two regression models with their default parameters from scikit-learn package; LinearRegression and RandomForestRegressor.

For evaluation, I used mean absolute error to see how close is the predicted score. Also, I wrote a custom accuracy measure. It takes predicted score, actual score and a threshold value. If the difference between both scores falls under the threshold value, I count it as correct. After little experimentation, I set the threshold value as 25.

## Results and Discussion

Following table summarize the results of both models:

|  |  |  |
| --- | --- | --- |
|  | **Mean Absolute Error** | **Custom Accuracy** |
| **Random Forest** | 25.43% | 64.51% |
| **Linear Regression** | 43.35% | 36.69% |

It appears that random forest perform way better than the other model. To further improve the results, one can either try fine tuning these models or try fitting a complex model like neural network.