**Cricket Score Prediction Model**

Sahil Kumar Rathour

23N0266

IIT Bombay , Analytics Club

I used Jupyter notebook for most of my Python Coding Work .

I initiated the project by importing my data set as Pandas Data Frame , it is a huge data set and has about 1.25 M entries , the first observation was the ‘low\_memory=False’ parameter of ‘read\_csv’ function of Pandas , I used this function to load my data set as a whole , if you set ‘low\_memory=True’ then your data set will be loaded in chunks, and the reason I didn’t go for ‘True‘ is that sometimes it leads to mixed data types .

The data set has exactly 12.65 lakh rows and 23 columns , consisting of various features like ‘match\_id’, ’venue’,’bowling\_team’,’batting\_team’ etc.

**Visualization-**

* I was quite well aware of the data set since I have good domain knowledge and understanding of independent features , and thus I didn’t undergo extensive data visualization

**Data Cleaning AND Feature Construction**

* Firstly I dropped some features which don’t contribute to the overall score of the cricket match

'season','start\_date','striker','non\_striker','bowler','player\_dismissed','other\_wicket\_type','other\_player\_dismissed' and 'cricsheet\_id

* There are columns named runs\_off\_bat, extras, wides, noball , byes, legbyes, penalty , I noticed that the column ‘extras’ has the sum of runs scored because of wides, noballs, byes , legbyes , penalty , so I decided to drop these columns (‘wides’, ’noballs’ , ’byes’, ’legbyes’) because such rare event don’t usually effect the total runs scored in a match .
* **Removing Small Teams** – The data set has data of 28 teams , but not all the teams played significant number of matches , thus I decided to pick top 10 teams which played the most number of matches , and I chose the subset of the data set where these top 10 teams played the match among themselves.

**Steps**- Firstly I added an extra column ‘count’ where all the entries are 1 and then

-*df.groupby(['batting\_team'],as\_index=False)['count'].sum().sort\_values(by='count',ascending=False).head(10)*

this is how I got the top 10 teams [*India , Sri Lanka, Australia, Pakistan, England , West Indies, South Africa, New Zeland, Bangladesh , Zimbabwe*]

Then to filter out-

*df\_f = df[df['batting\_team'].isin(teams\_list)]*

*df\_f=df\_f[df\_f['bowling\_team'].isin(teams\_list)], post this the shape of the data set is (950173, 9)*

* Removing ‘venue’- wanted to extract venue of the match as well but because of the mixed data in this feature it was near impossible to extract something feasible from it , for some samples city is mentioned and for some samples the name of the stadium without the city , so I decided to drop this feature.

**FEATURE CONSTRUCTION**

* The next step I undertook was of constructing a few relevant features , from the pre existing ‘ball’ feature , this feature has values of the form say 2.3 where the integral part represents over number and the decimal part represents the number of balls bowled in that over.

I extracted over number and the ball number from this feature.

*df\_f['over'] = df\_f['ball'].apply(lambda x:str(x).split(".")[0])*

*df\_f['ball\_no'] = df\_f['ball'].apply(lambda x:str(x).split(".")[1])*

Then I changed the data type of the features to ‘int ’ using the *.astype(‘int’)* function.

* **Balls Bowled Feature-**  used the following code to get this feature

*df\_f['balls\_bowled'] = (df\_f['over']\*6) + df\_f['ball\_no']*

* **Current Score –** i got this feature using the following code

*df\_f['current\_score'] = df\_f.groupby(['match\_id','innings'])['ball\_runs'].cumsum()*

The thing to note down over here is that I have used match id and innings both to perform group by, this is because both the innings were having the same match id, thus I used it , so that I get current score of both the innings separately .

* **Wickets dropped -**  used the following code to get this feature

*df\_f['wicket\_dropped']=df\_f.groupby(['match\_id','innings'],as\_index=False)['wicket\_type'].cumsum()*

Here I have used ‘wicket\_type’ feature which has 0,1 entries , 0 if no wicket was taken in that ball , 1 if wicket was taken .

* Now as I previously mentioned that same match id was representing both the innings and I wanted to have a match id which identified innings as well

so I made a formula on the basis of my observation where I combined the previous match id and the innings number to create a new match id .

*df\_f['innings']=df\_f['innings']\*100000000*

*df\_f['ID']=(df\_f['match\_id']+df\_f['innings'])* , and then I dropped the old match id

* **Runs In Last Five Over –** this feature is very important because it plays a huge role in determining the overall score of the team , constructing it was a challenging task

*match\_ids = df\_f['ID'].unique()*

*last\_five = []*

*for match\_id in match\_ids:*

# Filter the DataFrame for the current match\_id

*match\_data = df\_f[df\_f['ID'] == match\_id]*

# Use rolling window to calculate the cumulative sum over a window of 30 balls

*cumulative\_sum = match\_data['ball\_runs'].rolling(window=30).sum()*

# Extend the last\_five list with the cumulative sum values

*last\_five.extend(cumulative\_sum.values.tolist())*

*df\_f['last\_five'] = last\_five , this gave a feature where we had the runs scored in last 30 balls*

Also for the first five overs we dropped NaN entries , so **CCA**

* **Current Run Rate**- *final\_df['CRR']=final\_df['current\_score']/(final\_df['balls\_bowled']/6)*

**FEATURE TRANSFORMATION**

* Performed One-Hot Encoding on the columns ‘batting\_team’ and ‘bowling team’
* Performed scaling using MinMaxScaler and scaled the whole data set between 0 and 1
* Observed that most of the columns were right skewed using .skew() function and kdeplot analysis and used PowerTranformer, ’yeo-johanson’ to reduce right skewness of the data

**Used Column Transformers for the transformation and connected various transformers using a pipeline and also integrated the StackingRegressor into the pipeline**

**5-fold StackingRegressor**

1. Trained the RandomForestRegressor , tuned it using GridSearchCV and got an r2\_score of about 92.5 percent .
2. Trained and tuned KNeighboursRegressor and got an r2\_score of 93.4 percent, figured out that the optimal number of n\_neighbours=5 and used ‘weights=’distance’’
3. The ‘weights=’distance’’ is a very uniqe parameter , this is basically weighted average , the neighbour which is far away from the query point will have a lesser say in the output and the neighbour which is closer wll have more say on the overall output .

To understand this better , say we have 3 selected 3 nearest neighbours , now say D1,D2,D3 are their respective distance from the query point-

we will calculate 1/D1=s1 ,1/D2=s2 similarly s3 , and the average will be calculated --

s1\*X1/(s1+s2+s3) + s2\*X2/(s1+s2+s3) + s3\*X3/(s1+s2+s3). this is the average as per the distance of the neighbour . If the neighbour is fqar away its Di will be large but si will be small and ultimately its weight si/(s1+s2…) will be very small

1. Used XGBRegressor as the final model ,tuned it a bit, and got a max r2\_score of 96 percent.

**Observations**

1. Linear models like LinearRegression, Ridge, ElaticNet were not apt for the data set , got -ve r2 score with these models which is understandable as well because the data is not linear
2. Tried using SVM also but wasn’t able to train it , device was crashing and google collab also got timed out , that is why didn’t go for SVM
3. XGBRegressor alone post tuning was giving a max r2\_score of 83 percent
4. Got the best results when used XGBRegressor as the final model in the stacking structure
5. **Tried using blending +stacking as well but it wasn’t performing in par with k-fold stacking so I used the usual k-fold stacking**

**ALSO OBSERVED THAT ADDING NEW FEATURES SIGNIFICANTLY IMPROVED THE ACCURACY OF THE MODEL**

**Model Export**

* Employed a pickle file for deploying the model on Streamlit
* Learnt basics of Streamlit and gained valuable experience
* Created a new virtual environment to solve the various problems faced while using Streamlit

**Possible Improvements** –

* As per my observation I could have constructed a few more features and it would have surely improved the accuracy , also I could have improved the performance by using Imputation techniques like ‘Arbitrary value imputation’ instead of performing CCA.
* The model could be improved significantly