

CS322:Big Data

Final Class Project Report

**Project (FPL Analytics ): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Date: \_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Class/Section |
| 1 | Aronya Baksy | PES1201800002 | 5-D |
| 2 | Ansh Sarkar | PES1201800275 | 5-D |
| 3 | Vishesh P | PES1201800314 | 5-I |
| 4 | Aditeya Baral | PES1201800366 | 5-D |

## Introduction

The aim of this project is to stream data related to English Premier League matches, accept this streaming data using Spark, and process it in real time. Each batch of data streamed consists of a single match information, and a list of events that occurred in each match. The aim of the project is to process these events, and accumulate the performance of each player (on the basis of these ratings). At the end of the streaming computation, the user should be able to request for player profile (his performance over the matches), match events (per player, their performance in a given match) or for a comparison between 2 teams that the user has made.

## Related work

Spark Streaming online documentation: https://spark.apache.org/docs/latest/streaming-programming-guide.html

## Design

We use pair RDDs to store all the streaming data. The raw streaming data is first split into event and match data based on the keys present in it (using the filter function on the input Dstream). The event data is passed through a series of functions that first compute each metric for each event, given the player as a key (events in this case are free kicks, foul events, pass events and shot events). The count of each event is accumulated per player using the updateStateByKey method of the Dstream.

Once the per match number of events are aggregated per player, the computation of the metric values per player can be undertaken using these values, and the final rating is computed using the metric values that were computed earlier. These operations are all performed using the updateStateByKey method of Dstream objects in PySpark.

For the computation of player chemistry (ie. a measure of how well two players interact with each other, which is a function of whether they are in the same team, and whether their ratings increased or fell after one single match), the transformWith and cartesian methods are used to get a pair of players as the key, and their respective chemistry values. Using updateStateByKey, these can be updated for every match.

The UI functions parse the input JSON file, read its format, and access the relevant information from the relevant RDDs that are stored (using the saveAsTextFile method). The player and team information that is required to service the UI tasks is retrieved from the DataFrame structures that hold the CSV files that were read into the Spark job.

## Results

## Problems

1. Problems with initializing the streaming job and environment configuration for streaming
2. Problems with storing data persistently for the final computation, which were solved with the saveAsTextFiles method of the Dstream object in PySpark.

## Conclusion

The learnings from this project are to implement an analytics pipeline that uses streaming data in real-time, as well as implement complex operations on that data and provide useful insights to the final user.

## EVALUATIONS:

|  |  |  |  |
| --- | --- | --- | --- |
| SNo | Name | SRN | Contribution (Individual) |
| 1 | Aronya Baksy | PES1201800002 | UI code |
| 2 | Ansh Sarkar | PES1201800275 | Metrics Calculation |
| 3 | Vishesh P | PES1201800314 | Metrics Calculation |
| 4 | Aditeya Baral | PES1201800366 | Streaming Envt. Setup |

## (Leave this for the faculty)

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Evaluator | Comments | Score |
|  |  |  |  |

## CHECKLIST:

|  |  |  |
| --- | --- | --- |
| SNo | Item | Status |
| 1. | Source code documented |  |
| 2. | Source code uploaded to GitHub – (access link for the same, to be added in status ) |  |
| 3. | Instructions for building and running the code. Your code must be usable out of the box. |  |