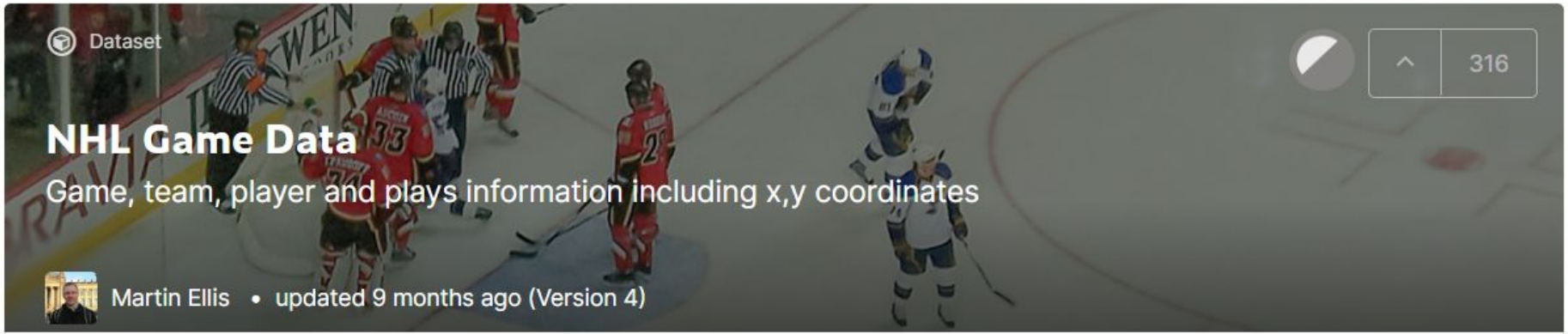
NHL fantasy prediction With SPARK

By Jean-Philippe Jacques

Because we are missing sports



Data Tasks Kernels (31) Discussion (8) Activity Metadata Download (1 GB) New Notebook

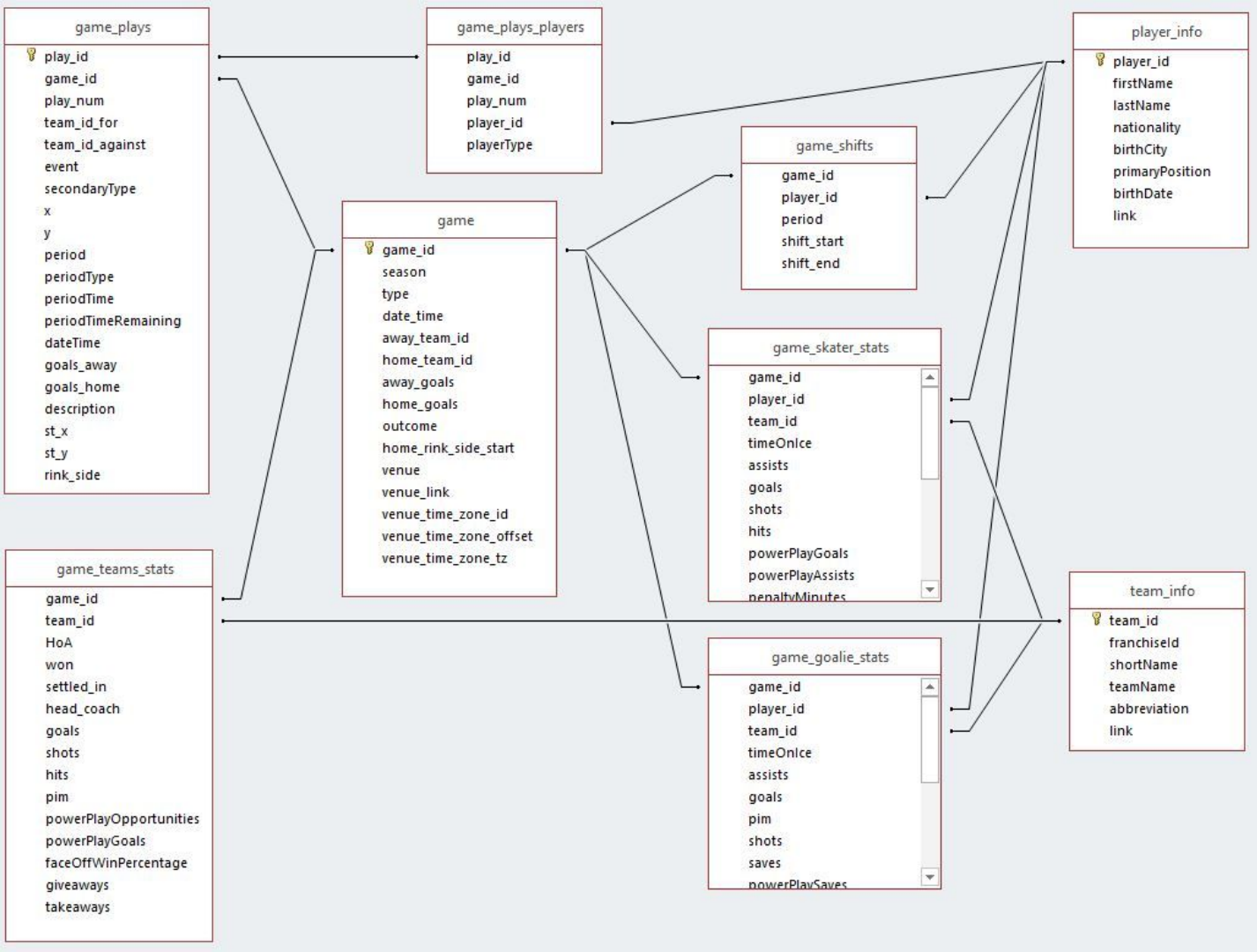
Usability 8.5 License Other (specified in description) Tags statistics, online media, sports, mathematics, video games and 3 more

The Data

It represent represents all the official metrics measured for each game in the NHL in the past 6 years.

Compared to other sports, advanced statistics in Hockey are still in infancy. It has been suggested that the best models can only predict the winner 62% of the time due to variances in talent and "puck luck".

NHL stats API was used to gather this data. Spark streaming would be a very useful and Spark ML will be very useful tool for live sports betting.


game_plays
play_id
game_id
play_num
team_id_for
team_id_against
event
secondaryType
x
y
period
periodType
periodTime
periodTimeRemaining
dateTime
goals_away
goals_home
description
st_x
st_y
rink_side
game_plays_players
play_id
game_id
play_num
player_id
playerType
player_info
player_id
firstName
lastName
nationality
birthCity
primaryPosition
birthDate
link
game_shifts
game_id
player_id
period
shift_start
shift_end
game
game_id
season
type
date_time
away_team_id
home_team_id
away_goals
home_goals
outcome
home_rink_side_start
venue
venue_link
venue_time_zone_id
venue_time_zone_offset
venue_time_zone_tz
game_skater_stats
game_id
player_id
team_id
timeOnIce
assists
goals
shots
hits
powerPlayGoals
powerPlayAssists
penaltyMinutes
game_teams_stats
game_id
team_id
HoA
won
settled_in
head_coach
goals
shots
hits
pim
powerPlayOpportunities
powerPlayGoals
faceOffWinPercentage
giveaways
takeaways
team_info
team_id
franchiseId
shortName
teamName
abbreviation
link
game_goalie_stats
game_id
player_id
team_id
timeOnIce
assists
goals
pim
shots
saves
powerPlaySaves

Main Objective

Forecast the yearly total of points by players

Read and Join the Data

```
val team=  sparkSession.read.option("header", "true")
  .option("delimiter", ",")
  .option("inferSchema", true)
  .csv("../NHLSPARK/src/NHLSPARKPACK/team_info.csv")

 val skater_stats =  sparkSession.read.option("header", "true")
  .option("delimiter", ",")
  .option("inferSchema", true)
  .csv("../NHLSPARK/src/NHLSPARKPACK/game_skater_stats.csv")

 val game =  sparkSession.read.option("header", "true")
  .option("delimiter", ",")
  .option("inferSchema", true)
  .csv("../NHLSPARK/src/NHLSPARKPACK/game.csv")

  val SS_Game = skater_stats.join(game, skater_stats.col("game_id").equalTo(game("game_id")))
  .drop(game.col("game_id"))
  .orderBy(game("date_time"))

  val SS_Game_Pyer = SS_Game.join(player, SS_Game.col("player_id").equalTo(player("player_id")))
  .drop(player.col("player_id"))
  .orderBy(SS_Game("date_time").desc).filter("type == 'R'")
  SS_Game_Pyer.cache()
```

Feature Engineering, Lag Data, last 10 games

Spark is very easy to use and very fast to create new feature with aggregation over time series with Window.

```
val last10 = Window.partitionBy('player_id).orderBy('date_time).rowsBetween(-11, -1)
val last20 = Window.partitionBy('player_id).orderBy('date_time).rowsBetween(-21, -1)
val lagdataset = SS_Game_Pyer.withColumn("l10_assists", sum('assists) over last10).withColumn(
                                         "L10_goals", sum('goals) over last10).withColumn(
                                         "L10_shots", sum('shots) over last10).withColumn(
                                         "L10_faceOffWins", sum('faceOffWins) over last10).withColumn(
                                         "L10_takeaways", sum('takeaways) over last10).withColumn(
                                         "L10_plusMinus", sum('plusMinus) over last10).withColumn(
```

Yearly Stats with GoupBy and more features

```
val dataByYearPlayers = lagdataseta.groupBy("player_id", "season").agg(countDistinct('game_id),
    sum('assists),mean('assists),stddev_samp('assists),
    sum('goals),mean('goals),stddev_samp('goals),
    sum('points),mean('points),stddev_samp('points),
    sum('timeOnIce),mean('timeOnIce),stddev_samp('timeOnIce),
    sum('shots),mean('shots),stddev_samp('shots),
    sum('powerPlayGoals),mean('powerPlayGoals),stddev_samp('powerPlayGoals),
```

One Train set , one Test set and One to Forecast

Use filter and randomSplit to split dataframe

```
val traintmp = dataByYearPlayerNext.filter("nexty_Team >0").filter("rank<400")
val tofroctmp = dataByYearPlayerNext.filter("season == '20182019'").filter("rank<400")
val train =traintmp.na.fill(0)
val tofroc =tofroctmp.na.fill(0)
val colt = train.schema.names.clone()
val colf = colt.filter(! _.contains("nexty_points"))
val split = train.randomSplit(Array(0.8, 0.2), seed = 11L)
val train_data = split(0).cache()
val test_data = split(1).cache()
```

Use 3 regression types

Generalized Linear Regression

```
///GenLinReg
val glr = new GeneralizedLinearRegression()
  .setFamily("gaussian")
  .setLink("identity")
  .setMaxIter(10)
  .setRegParam(0.3)
  .setLabelCol("nexty_points")
  .setFeaturesCol("features")
```

(RMSE)= 12.08195

RandomForestRegressor

```
///RandomForestRegressor
val dtc = new RandomForestRegressor()
.setLabelCol("nexty_points")
.setFeaturesCol("features")
.setMaxDepth(8)
```

(RMSE)= 12.2388

Gradient-Boosted Trees

```
///GBTRegressor
val gbt = new GBTRegressor()
.setLabelCol("nexty_points")
.setFeaturesCol("features")
.setCheckpointInterval(10)
```

(RMSE)= 14.02523

Conclusion

Preprocessing and feature Engineering was very fast and very easy to do with spark.

The machine learning training part was different. Much slower than most of python libraries and also use lot of memory. I wasn’t able to deploy deep forest because of memory limitation. Spark machine learning tool are also limited and harder to use with limited amount of code exemple.

It will be useful to found some fast data pipeline that link spark preprocessing data to Python machine learning training and that bring back python model in spark scala for processing live streaming data on that model and return forecast.