EBOOK

Build an End-to-End Machine Learning Pipeline for Live Sports with Apache Spark™ and Databricks

A how-to guide including code samples and notebooks

databricks

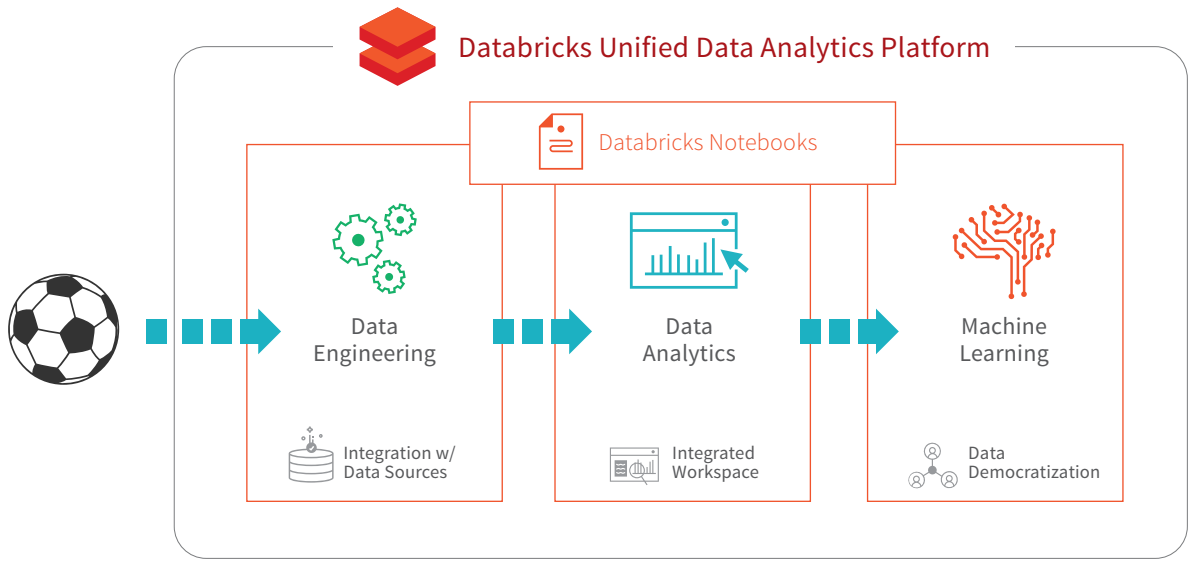
Introduction

The global sports market is huge, comprised of players, teams, leagues, fan clubs, sponsors, etc., and all of these entities interact in myriad ways generating an enormous amount of data. Some of that data is used internally to help make better decisions, and there are a number of use cases within the media industry that use the same data to create better products and attract/retain viewers.

A few ways that the sports and media industries have started utilizing big data are:

- Analyze on-field conditions and events (passes, player positions, etc.) that lead to soccer goals, football touchdowns, or baseball home runs etc.
- Assess the win-loss percentage with different combinations of players in different on-field positions.
- Track a sportsperson's or team's performance graph over the years/seasons.

Ultimately, these industries need to build an end-to-end data pipeline comprised of these three functional components: data engineering, data analysis, and machine learning. To extract these insights from their big data, sports and media companies need to build end-to-end data pipelines, our approach to addressing these questions is by selecting a unified platform that offers these capabilities. Databricks provides a Unified Data Analytics Platform that brings together big data and AI and allows the different personas of your organization to come together and collaborate in a single workspace.



European Soccer Leagues Data

Soccer or football in most countries has massive global appeal, an amazing combination of skills and strategy, and the extreme fans that add an extra element to the game.

In this ebook, we'll see how you could use Apache Spark and Databricks to create an end-to-end data pipeline with European Soccer games data – including facets from data engineering, data analysis, and machine learning, to help answer business questions. We'll use a dataset from Kaggle , that provides a granular view of 9,074 games, from the biggest 5 European soccer leagues: England, Spain, Germany, Italy, and France, for the 2011 to 2016 seasons.

The primary dataset is specific events from the games in chronological order, including key information like:

- id_odsp – unique identifier of game
- time – minute of the game
- event_type – primary event
- event_team – the team that produced the event
- player – name of the player involved in the main event
- shot_place – placement of the shot, 13 possible placement locations
- shot_outcome – 4 possible outcomes
- location – location on the pitch where the event happened, 19 possible locations
- is_goal – binary variable if the shot resulted in a goal (own goals included)
- And more..

The second smaller dataset includes high-level information and advanced stats with one record per game. Key attributes are "League", "Season", "Country", "Home Team", "Away Team" and various market odds.

Data Engineering

We start out by creating an ETL notebook, where the two CSV datasets are transformed and joined into a single Parquet data layer, which enables us to utilize DBIO caching feature for high-performance big data queries.

Extraction

The first task is to create a DataFrame schema for the larger game events dataset, so the read operation doesn't spend time inferring it from the data. Once extracted, we'll replace "null" values for interesting fields with data-type specific constants as noted in the code snippet below.

```
from pyspark.sql.types import *

schema = (StructType().
          add("id_odsp", StringType()).add("id_event", StringType()).add("sort_order", IntegerType()).
          add("time", IntegerType()).add("text", StringType()).add("event_type", IntegerType()).
          add("event_type2", IntegerType()).add("side", IntegerType()).add("event_team", StringType()).
          add("opponent", StringType()).add("player", StringType()).add("player2", StringType()).
          add("player_in", StringType()).add("player_out", StringType()).add("shot_place", IntegerType()).
          add("shot_outcome", IntegerType()).add("is_goal", IntegerType()).add("location", IntegerType()).
          add("bodypart", IntegerType()).add("assist_method", IntegerType()).add("situation", IntegerType()).
          add("fast_break", IntegerType())
         )

eventsDf = (spark.read.csv("/data/eu-soccer-events/input/events.csv",
                           schema=schema, header=True,
                           ignoreLeadingWhiteSpace=True,
                           ignoreTrailingWhiteSpace=True,
                           nullValue='NA'))

eventsDf = eventsDf.na.fill({'player': 'NA', 'event_team': 'NA', 'opponent': 'NA',
                             'event_type': 99, 'event_type2': 99, 'shot_place': 99,
                             'shot_outcome': 99, 'location': 99, 'bodypart': 99,
                             'assist_method': 99, 'situation': 99})
display(eventsDf)
```

Data Engineering

This is what the raw data (with some NULLs replaced) looks like:

time	text	event_type	event_type2	side	event_team	opponent	player	player2	player_in	player_out	shot_place	shot_outcome	is_goal	location
32	Roman Weidenfeller (Borussia Dortmund) wins a free kick in the defensive half.	8	99	1	Borussia Dortmund	Hamburg SV	roman weidenfeller	null	null	null	99	99	0	2
32	Corner, Hamburg. Conceded by Ilkay Gundogan.	2	99	2	Hamburg SV	Borussia Dortmund	ilkay gundogan	ilkay gundogan	null	null	99	99	0	99
32	Foul by Gojko Kacar (Hamburg).	3	99	2	Hamburg SV	Borussia Dortmund	gojko kacar	null	null	null	99	99	0	99
33	Foul by Robert Lewandowski (Borussia Dortmund).	3	99	1	Borussia Dortmund	Hamburg SV	robert lewandowski	null	null	null	99	99	0	99

Showing the first 1000 rows.

We also read the second dataset into a DataFrame, as it includes the country name which we'll use later during analysis.

```
gameInfDf = (spark.read.csv("/data/eu-soccer-events/input/ginf.csv",
                            inferSchema=True, header=True,
                            ignoreLeadingWhiteSpace=True,
                            ignoreTrailingWhiteSpace=True,
                            nullValue="NA"))
display(gameInfDf)
```

▸ gameInfDf: pyspark.sql.dataframe.DataFrame = [id_odsp: string, link_odsp: string ... 16 more fields]

id_odsp	link_odsp	adv_stats	date	league	season	country	ht	at	fthg	ftag	odd_h	odd_d	odd_a	od
UFot0hit/	/soccer/germany/bundesliga-2011-2012/dortmund-hamburger-UFot0hit/	true	2011-08-05T00:00:00.000+0000	D1	2012	germany	Borussia Dortmund	Hamburg SV	3	1	1.56	4.41	7.42	nu
Aw5DfILH/	/soccer/germany/bundesliga-2011-2012/augsburg-freiburg-Aw5DfILH/	true	2011-08-06T00:00:00.000+0000	D1	2012	germany	FC Augsburg	SC Freiburg	2	2	2.36	3.6	3.4	nu
bkjpaC6n/	/soccer/germany/bundesliga-2011-2012/werder-bremen-kaiserslautern-bkjpaC6n/	true	2011-08-06T00:00:00.000+0000	D1	2012	germany	Werder Bremen	Kaiserslautern	2	0	1.83	4.2	4.8	nu
CzPV312a/	/soccer/france/ligue-1-2011-2012/paris-sg-lorient-CzPV312a/	true	2011-08-06T00:00:00.000+0000	F1	2012	france	Paris Saint-Germain	Lorient	0	1	1.55	4.5	9.4	nu
GUOdmtIl/	/soccer/france/ligue-1-2011-2012/caen-valenciennes-	true	2011-08-06T00:00:00.000+0000	F1	2012	france	Caen	Valenciennes	1	0	2.5	3.4	3.45	nu

Showing the first 1000 rows.

Data Engineering

Transformation

The next step is to transform and join the DataFrames into one. Many fields of interest in the game events DataFrame have numeric IDs, so we define a generic UDF that could use look-up tables for mapping IDs to descriptions.

```
def mapKeyToVal(mapping):
    def mapKeyToVal_(col):
        return mapping.get(col)
    return udf(mapKeyToVal_, StringType())
```

The mapped descriptions are stored in new columns in the DataFrame. So once the two DataFrames are joined, we'll filter out the original numeric columns to keep it as sparse as possible. We'll also use QuantileDiscretizer to add a categorical "time_bin" column based on "time" field.

```
gameInfDf = gameInfDf.withColumn("country_code", mapKeyToVal(countryCodeMap)("country"))

display(gameInfDf['id_odsp','country','country_code'])
```

This next code snippet performs a lookup using UDFs and joining DataFrames.

```
eventsDf = (
            eventsDf.
            withColumn("event_type_str", mapKeyToVal(evtTypeMap)("event_type")).
            withColumn("event_type2_str", mapKeyToVal(evtTyp2Map)("event_type2")).
            withColumn("side_str", mapKeyToVal(sideMap)("side")).
            withColumn("shot_place_str", mapKeyToVal(shotPlaceMap)("shot_place")).
            withColumn("shot_outcome_str", mapKeyToVal(shotOutcomeMap)("shot_outcome")).
            withColumn("location_str", mapKeyToVal(locationMap)("location")).
            withColumn("bodypart_str", mapKeyToVal(bodyPartMap)("bodypart")).
            withColumn("assist_method_str", mapKeyToVal(assistMethodMap)("assist_method")).
            withColumn("situation_str", mapKeyToVal(situationMap)("situation"))
          )

joinedDf = (
  eventsDf.join(gameInfDf, eventsDf.id_odsp == gameInfDf.id_odsp, 'inner').
  select(eventsDf.id_odsp, eventsDf.id_event, eventsDf.sort_order, eventsDf.time, eventsDf.event_type,
eventsDf.event_type_str, eventsDf.event_type2, eventsDf.event_type2_str, eventsDf.side, eventsDf.side_
str, eventsDf.event_team, eventsDf.opponent, eventsDf.player, eventsDf.player2, eventsDf.player_in,
eventsDf.player_out, eventsDf.shot_place, eventsDf.shot_place_str, eventsDf.shot_outcome, eventsDf.
shot_outcome_str, eventsDf.is_goal, eventsDf.location, eventsDf.location_str, eventsDf.bodypart,
eventsDf.bodypart_str, eventsDf.assist_method, eventsDf.assist_method_str, eventsDf.situation,
eventsDf.situation_str, gameInfDf.country_code)
)
```

Data Engineering

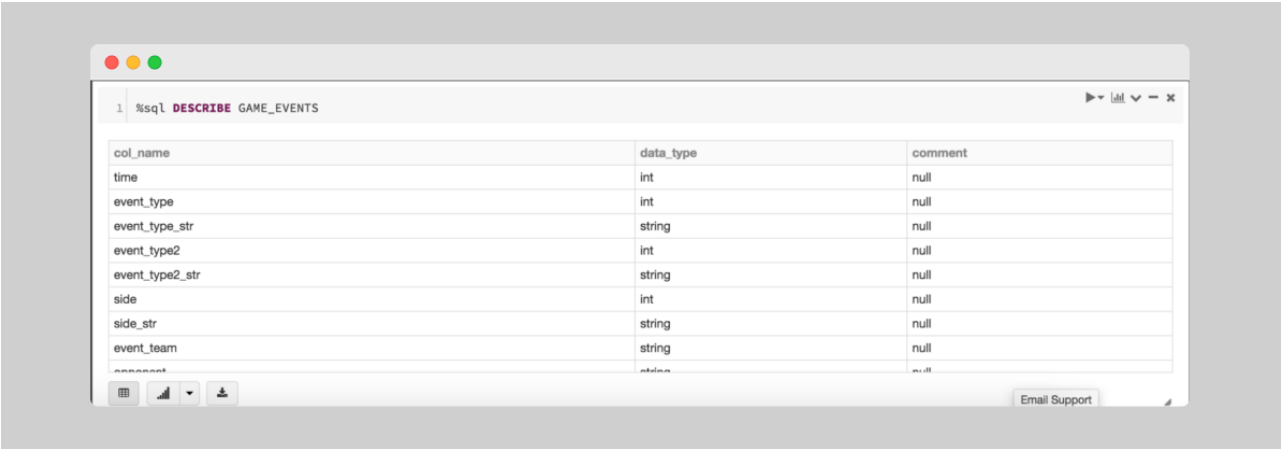
Loading

Once the data is in the desired shape, we'll load it as Parquet into a Spark table that would reside in a domain-specific database. The database and table will be registered with internal Databricks metastore, and the data will be stored in DBFS. We'll partition the Parquet data by "country_code" during write.

```
%sql
-- Create Databricks/Spark database
CREATE DATABASE IF NOT EXISTS EURO_SOCCER_DB
LOCATION "dbfs:/FileStore/databricks-abhinav/eu-soccer-events/interm"

-- Set the database session
USE EURO_SOCCER_DB

-- Load transformed game event data into a Databricks/Spark table
joinedDf.write.saveAsTable("GAME_EVENTS", format = "parquet", mode = "overwrite",
partitionBy = "COUNTRY_CODE", path = "dbfs:/FileStore/databricks-abhinav/eu-
soccer-events/interm/tr-events")
```

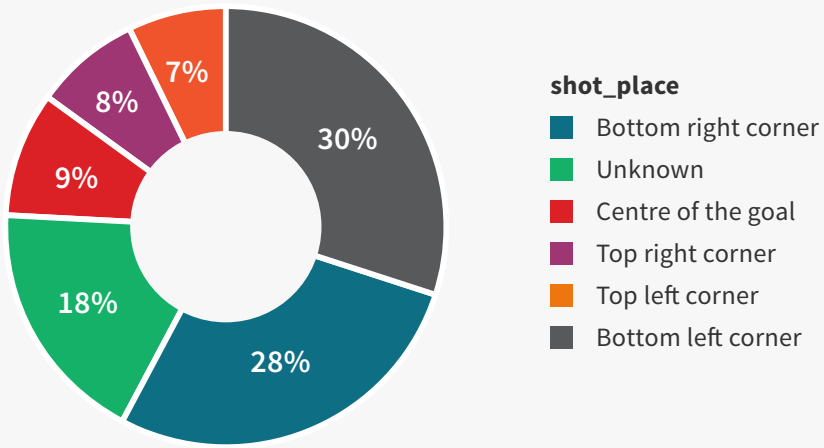


Data Analysis

Now that the data shape and format is all set, it's time to dig in and try and find answers to a few business questions. We'll use plain-old super-strong SQL (Spark SQL) for that purpose, and create a second notebook from the perspective of data analysts.

For example, if one wants to see the distribution of goals by shot placement, then it could look like this simple query and resulting pie-chart (or alternatively viewable as a data-grid).

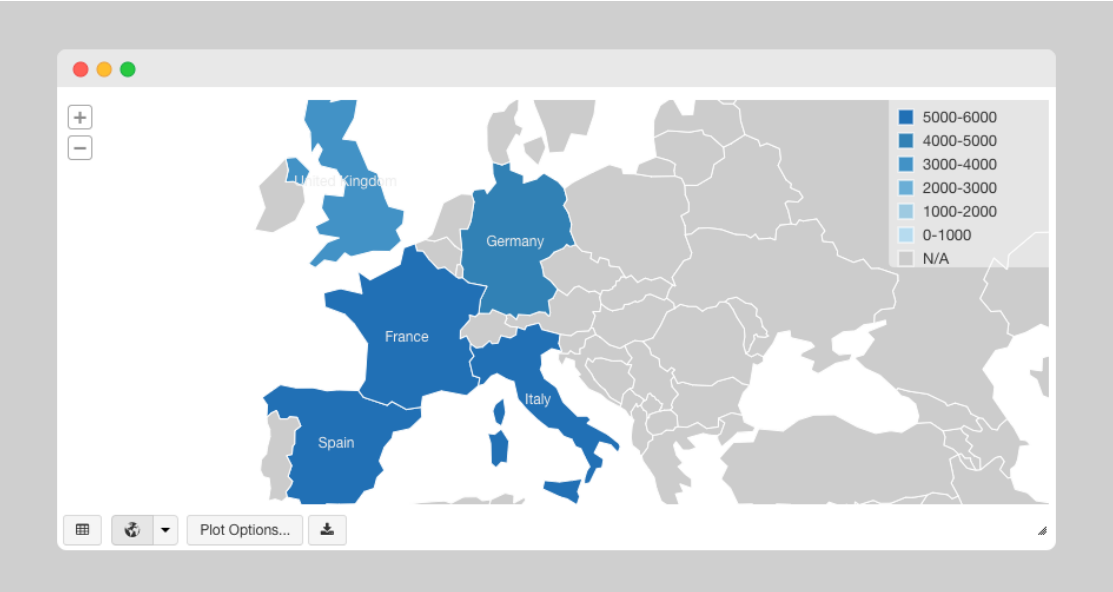
```
%sql
SELECT CASE WHEN shot_place_str == 'NA' THEN 'Unknown' ELSE shot_place_str END
shot_place, COUNT(1) AS TOT_GOALS
  FROM GAME_EVENTS
 WHERE is_goal = 1
 GROUP BY shot_place_str
```



Data Analysis

Or, if the requirement is to see the distribution of goals by countries/leagues, it could look like this map visualization (which needs ISO country codes, or US state codes as a column).

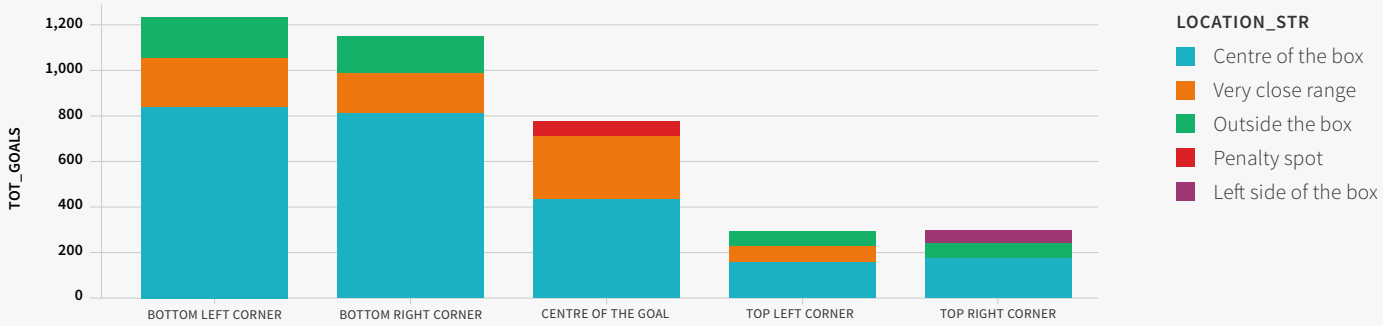
```
%sql
SELECT country_code, COUNT(1) AS TOT_GOALS
  FROM GAME_EVENTS
 WHERE is_goal = 1
 GROUP BY country_code
```



Data Analysis

Once we observe that Spanish league has had most goals over the term of this data, we could find the top 3 goals locations per shot place from the games in Spain, by writing a more involved query using Window functions in Spark SQL. It would be a stepwise nested query:

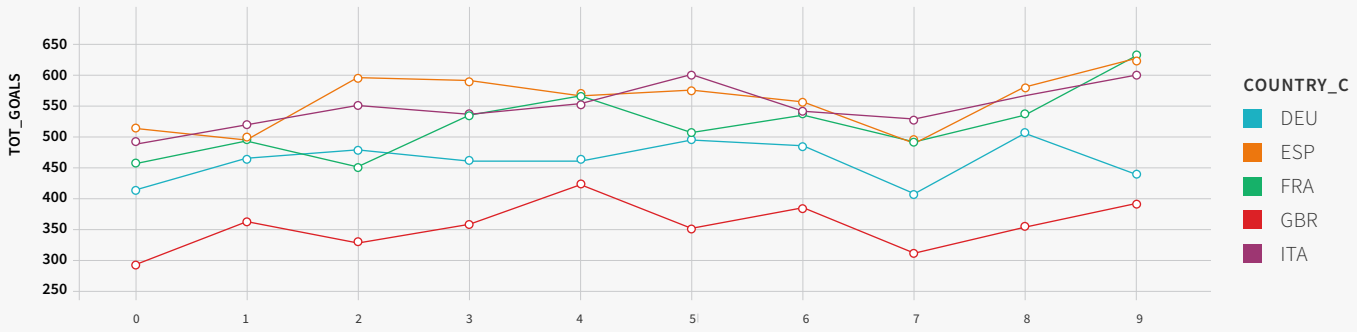
```
%sql
SELECT SHOT_PLACE_STR, LOCATION_STR, TOT_GOALS
  FROM (
     SELECT SHOT_PLACE_STR, LOCATION_STR, TOT_GOALS,
            RANK() OVER (PARTITION BY SHOT_PLACE_STR ORDER BY TOT_GOALS DESC) goals_rank
       FROM (
              SELECT CASE WHEN LOCATION_STR == 'NA' THEN 'Unknown' ELSE LOCATION_STR END LOCATION_STR,
                     CASE WHEN SHOT_PLACE_STR == 'NA' THEN 'Unknown' ELSE SHOT_PLACE_STR END SHOT_PLACE_STR,
                     COUNT(1) AS TOT_GOALS
              FROM GAME_EVENTS
              WHERE is_goal = 1 AND COUNTRY_CODE = 'ESP'
             GROUP BY SHOT_PLACE_STR, LOCATION_STR
       ) tmp_in
       WHERE TOT_GOALS IS NOT NULL AND TOT_GOALS <> 0
     ) tmp_out
WHERE goals_rank <= 3 AND LOCATION_STR != 'Unknown' AND SHOT_PLACE_STR != 'Unknown'
ORDER BY SHOT_PLACE_STR
```



Data Analysis

We could do time-based analysis as well, e.g. by observing the total number of goals over the course of a game (0-90+ minutes), across all games in the five leagues. We could use the “time_bin” column created as part of the transformation process earlier, rather than a continuous variable like “time”.

```
%sql
SELECT COUNTRY_CODE, TIME_BIN, COUNT(1) TOT_GOALS
  FROM GAME_EVENTS
 WHERE is_goal = 1
 GROUP BY COUNTRY_CODE, TIME_BIN
 ORDER BY COUNTRY_CODE, TIME_BIN
```



Machine Learning

As we saw, doing descriptive analysis on big data has been made super easy with Spark SQL and Databricks. But what if you're a data scientist who's looking at the same data to find combinations of on-field playing conditions that lead to "goals"?

We'll now create a third notebook from that perspective, and see how one could fit a GBT classifier Spark ML model on the game events training dataset. In this case, our binary classification label will be field "is_goal", and we'll use a mix of categorical features like "event_type_str", "event_team", "shot_place_str", "location_str", "assist_method_str", "situation_str" and "country_code".

First, we need to do the necessary imports from Spark ML:

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

Then the following three-step process is required to convert our categorical feature columns to a single binary vector:

- Convert string features to indices using StringIndexer
- Transform feature indices to binary vectors using OneHotEncoder
- Assemble different binary vector columns into a single vector using VectorAssembler

```
# Create a list of categorical features
categFeatures = ["event_type_str", "event_team", "shot_place_str", "location_str",
"assist_method_str", "situation_str", "country_code"]

# Encode categorical string cols to label indices
stringIndexers = [StringIndexer().setInputCol(baseFeature).setOutputCol(baseFeature +
"_idx") for baseFeature in categFeatures]

# Convert categorical label indices to binary vectors
encoders = [OneHotEncoder().setInputCol(baseFeature + "_idx").setOutputCol(baseFeature
+ "_vec") for baseFeature in categFeatures]

# Combine all columns into a single feature vector
featureAssembler = VectorAssembler()
featureAssembler.setInputCols([baseFeature + "_vec" for baseFeature in categFeatures])
featureAssembler.setOutputCol("features")
```

Machine Learning

Finally, we'll create a Spark ML Pipeline using the above transformers and the GBT classifier. We'll divide the game events data into training and test datasets, and fit the pipeline to the former.

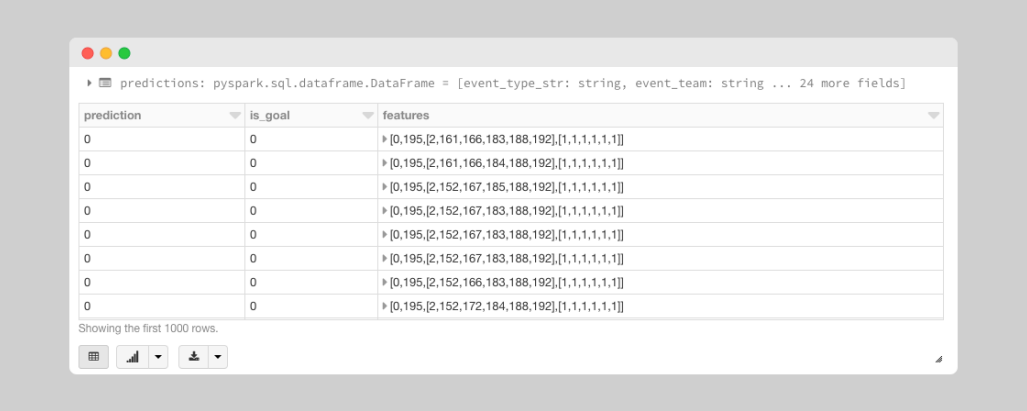
```
# Create Spark ML pipeline using a GBT classifier
gbtClassifier = GBTClassifier(labelCol="is_goal", featuresCol="features", maxDepth=5,
maxIter=20)

pipelineStages = stringIndexers + encoders + [featureAssembler, gbtClassifier]
pipeline = Pipeline(stages=pipelineStages)

# Split dataset into training/test, and create a model from training data
(trainingData, testData) = gameEventsDf.randomSplit([0.75, 0.25])
model = pipeline.fit(trainingData)
```

Now we can validate our classification model by running inference on the test dataset. We could compare the predicted label with actual label one by one, but that could be a painful process for lots of test data. For scalable model evaluation in this case, we can use BinaryClassificationEvaluator with area under ROC metric. One could also use the Precision-Recall Curve as an evaluation metric. If it was a multi-classification problem, we could've used the MulticlassClassificationEvaluator.

```
# Validate the model on test data, display predictions
predictions = model.transform(testData)
display(predictions.select("prediction", "is_goal", "features"))
```


predictions: pyspark.sql.dataframe.DataFrame = [event_type_str: string, event_team: string ... 24 more fields]

prediction	is_goal	features
0	0	[0,195,[2,161,166,183,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,161,166,184,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,152,167,185,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,152,167,183,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,152,167,183,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,152,167,183,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,152,166,183,188,192],[1,1,1,1,1,1]]
0	0	[0,195,[2,152,172,184,188,192],[1,1,1,1,1,1]]

Showing the first 1000 rows.

```
# Evaluate the model using areaUnderROC metric
evaluator = BinaryClassificationEvaluator(
    labelCol="is_goal", rawPredictionCol="prediction")
evaluator.evaluate(predictions)

# Output
Out[11]: 0.7774092428726642
```

Summary

We demonstrated how to build the three functional components of data engineering, data analysis, and machine learning using the Databricks Unified Data Analytics Platform. We've illustrated how you can run your ETL, analysis, and visualization, and machine learning pipelines all within a single Databricks notebook. By removing the data engineering complexities commonly associated with such data pipelines with the Databricks Unified Data Analytics Platform, this allows different sets of users i.e. data engineers, data analysts, and data scientists to easily work together to find hidden value in big data from any sports.

But this is just the first step. A sports or media organization could do more by running model-based inference on real-time streaming data processed using Structured Streaming, in order to provide targeted content to its users. And then there are many other ways to combine different Spark/Databricks technologies, to solve different big data problems in sport and media industries.

Download our European Soccer Events Notebook to begin modeling this data yourself

Learn More

Databricks, founded by the original creators of Apache Spark™, is on a mission to accelerate innovation for our customers by unifying data science, engineering and business teams. The Databricks Unified Data Analytics Platform powered by Apache Spark enables data science teams to collaborate with data engineering and lines of business to build data and machine learning products. Users achieve faster time-to-value with Databricks by creating analytic workflows that go from ETL through interactive exploration. We also make it easier for users to focus on their data by providing a fully managed, scalable, and secure cloud infrastructure that reduces operational complexity and total cost of ownership.

To learn how you can build scalable, real-time data and machine learning pipelines: **databricks.com/trial**.