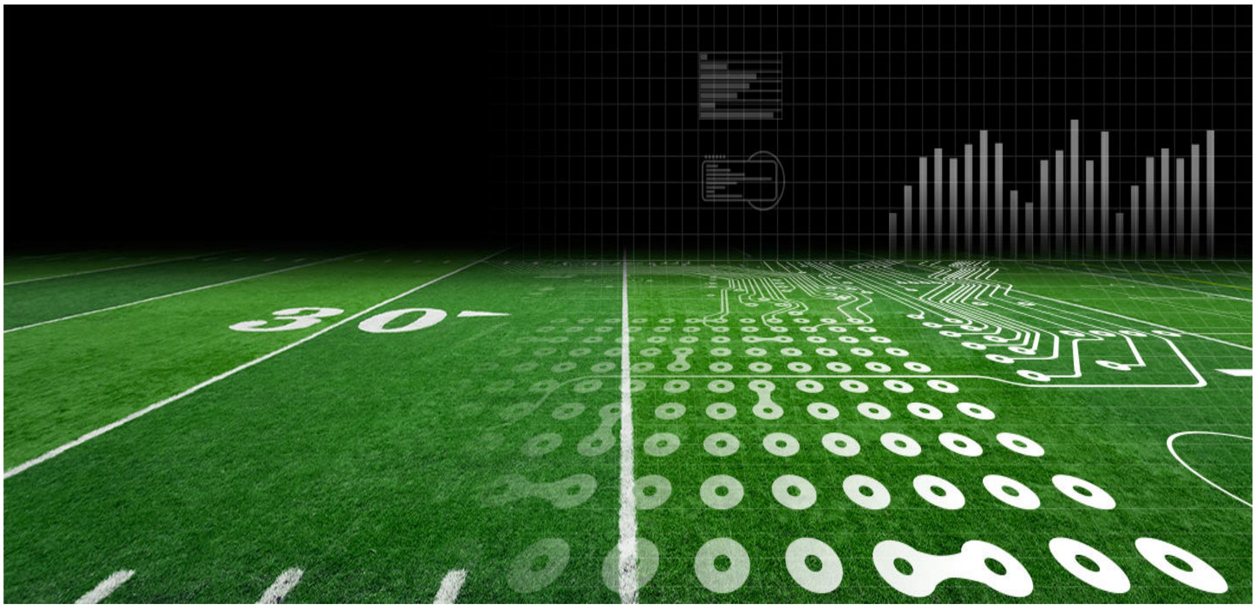
NFL Play Predictions with (Simulated) Streaming



Romith Challa

Data Description

An NFL dataset, consisting of comprehensive play-by-play data, was downloaded from *Kaggle* and uploaded into the DBFS of *DataBricks* to kickstart this project. The source data was originally put together by Carnegie Mellon University, through the use of *nflscrapR* library. The data encompasses nine seasons (2009-2018) worth of individual play data, spanning a total of 255 variables for each play record. To scale down the scope of this project, I decided to focus on data for only the recent seasons, with the features of most personal interest. To that extent, the full raw data was initially loaded and filtered in order to: a) remove data before the 2014-2015 season, b) keep only the variables involving the team, time, yardage, and play information, c) establish a four-level classification problem (including only 'run', 'pass', 'punt', or 'field-goal' plays). This truncated dataset was then written out to a new *csv* file, and subsequently loaded with a pre-defined schema, to be used as the official dataset for the remainder of the project.

Variable	Type	Format	Description
game_id	feature	numeric	ID contains season; used for data splitting
pos_team	feature	string	Team on offense (with possession) for play
def_team	feature	string	Team on defense for play
yardline_100	feature	numeric	Yards left till endzone
yds_till_first	feature	numeric	Yards left till first-down conversion
half_sec_remaining	feature	numeric	Time (in sec) remaining till end of that half
score_differential	feature	numeric	Offense's score – defense's score
pos_team_timeouts	feature	numeric	Timeouts remaining for team on offense
play_type	label	string	'run', 'pass', 'field-goal', or 'punt'

The final data cleansing step, before any pre-processing for modeling took place, was to update stale team abbreviations. This could have resulted from an NFL team moving cities (i.e., St. Louis Rams to Los Angeles Rams) or just a change in abbreviations (i.e., JAC to JAX). The newer name was chosen to keep consistent across all records. A data-partitioning step followed after, that split the records into a training set encompassing 2014-2017 seasons, and a test set for the 2017-2018 season.

Classification Problem

The objective for this project is to accurately classify NFL plays, into one of four levels, based on the various team, time, and yardage features. A tree-based model was chosen for this multi-level classification problem, as they are known to be more robust in this setting. Specifically, a random forest was chosen, which was manually tuned to balance runtime and performance. However, before training the model, a series of transformations needed to occur to pre-process the dataset and prepare it for any distributed application of machine learning techniques.

To that extent, a pipeline was assembled consisting of several estimator and transformer objects. As the team info and play types were in string format, they were first indexed into numeric form through *StringIndexer* method. The score differential information was set to undergo binning, through a *Bucketizer* that categorizes the numeric data into buckets that represent the following for the team on offense: down by multiple possessions, down by single possession, close game, up by single possession, up by multiple possessions. Afterwards, the data was then put into a *VectorAssembler* to format it as necessary for modeling. Lastly, as some of the numeric features were on wildly different scales which could affect some machine learning algorithms, a min-max scaler was also instantiated to take the input vectorized feature set and output a scaled version. The *RandomForestClassifier* algorithm was added as the remaining step to this pipeline, before fitting and transforming the training data. To observe how well the model performs on the training data itself, a quick transformation was done, which yielded an accuracy of 61%. A static version of test set predictions were also evaluated for baseline, and also yielded similar results. Despite generalizing well, the model outputted an underwhelming accuracy score which can be accounted for by the lack of proper model tuning and further feature engineering. Some sample results are displayed below:

```
Accuracy for (static) train set predictions:  0.6099967496117592
+---------+-----+----------+
|play_type|label|prediction|
+---------+-----+----------+
|      run|  1.0|       0.0|
|     pass|  0.0|       0.0|
|     pass|  0.0|       0.0|
|     pass|  0.0|       0.0|
|      run|  1.0|       0.0|
+---------+-----+----------+
only showing top 5 rows

Accuracy for (static) test set predictions:  0.6087643859871

Command took 1.90 minutes -- by romith.challa@du.edu at 3/3/2022, 9:12:31 PM on 3-2
```

Streaming Simulation

As part of the objective of this project, streaming was set up for the test set (2017-2018 season) to simulate real-time play predictions for a new season, based on the model that was trained on previous seasons. This was accomplished by first assessing the underlying default partitioning of the test data. Based on the game ID, there were 224 unique games for the test season. The data was written out to a directory that will consist of a *csv* file for each partition. However, as I was unable to find a proper dataset with timestamps to conduct a windowed stream, and due to the technical capacity of *Databricks* 'community edition' that made it unrealistic to stream a single play at a time, the simulation was setup to handle bunch of plays from a game at once instead.

This directory was used as the stream source, from which the "new" season data is retrieved from in order to predict NFL plays in "real-time". Before making the headway with this, an initial query was performed on the training set results, to create a SQL table that can be dynamically updated for each set of streamed test predictions. Each set of stream data is read in from the source directory created earlier and undergoes the necessary pre-processing data transformations through the assembled pipeline. The constructed model is then used to transform the test dataset and yield predictions based on the new streaming data. A query that looks at the results for each set is then written out to a sink, with a *memory* format and a trigger of 10 seconds. As the desired output is the predictions themselves, and not any aggregated computation, the output mode of "append" was the only viable option. A separate query outputs the most recent game situation information, with the corresponding predicted play, to ideally inform coaching staff to prepare for next play. These dynamic results can be observed as below:

game_id	pos_team	def_team	yardline_100	yds_till_first	half_sec_remaining	score_differential	play_type	label	probability	prediction
2018090902	IND	CIN	52	2	1263	-3	pass	0.0	[0.55507112008713...	0.0
2018090902	CIN	IND	27	5	1498	0	pass	0.0	[0.57806055963257...	0.0
2018090902	IND	CIN	54	4	1300	-3	run	1.0	[0.55780996847830...	0.0
2018090902	CIN	IND	89	6	1652	0	pass	0.0	[0.55768038257395...	0.0
2018090902	CIN	IND	32	10	1537	0	run	1.0	[0.47704000557439...	1.0
2018090902	CIN	IND	24	12	1410	0	pass	0.0	[0.71085970856887...	0.0
2018090902	IND	CIN	60	10	1329	-3	pass	0.0	[0.48291952870497...	1.0
2018090902	CIN	IND	63	3	1751	0	pass	0.0	[0.55504374626867...	0.0
2018090902	CIN	IND	93	10	1690	0	run	1.0	[0.46906501691038...	1.0
2018090902	CIN	IND	60	10	1615	0	run	1.0	[0.47307677146019...	1.0
2018090902	CIN	IND	53	3	1577	0	pass	0.0	[0.55504374626867...	0.0
2018090902	CIN	IND	22	10	1460	0	run	1.0	[0.47704000557439...	1.0
2018090902	CIN	IND	24	12	1415	0	pass	0.0	[0.71085970856887...	0.0
2018090902	IND	CIN	75	10	1400	-3	pass	0.0	[0.48291952870497...	1.0
2018090902	IND	CIN	72	7	1365	-3	run	1.0	[0.56427136965793...	0.0
2018090902	CIN	IND	70	10	1793	0	run	1.0	[0.46652623979684...	1.0
2018090902	IND	CIN	41	7	1191	-3	pass	0.0	[0.56599347874765...	0.0
2018090902	IND	CIN	7	7	1741	0	run	1.0	[0.57234009163206...	0.0

Command took 0.14 seconds -- by romith.challa@du.edu at 3/3/2022, 8:43:56 PM on 3-2

Challenges

This project was designed to be a fairly scaled-down implementation of intersecting distributed machine learning with streaming. This is because the modeling process doesn't involve any hyperparameter tuning, validation, nor selection, and the streaming process is "simulated" to simplify the process. However, there were still some unexpected hurdles in executing this project.

One challenge faced was the excessive time to read in and work with the original raw dataset. As it consisted of over 400,000 records and 255 variables, it was not feasible to build this study, even in a distributed environment, with the basic functionality of the *community* version. Therefore, I made the choice to truncate the dataset substantially which would simplify the process of building a manual schema, as well as cut down runtime throughout the executions.

Another unexpected challenge was some data cleansing steps that needed to take place such as standardizing team names to account for any changes over the seasons. In fact, this issue was only caught after the initial modeling stage, as I ran into an error that the tree-based models only allow up to 32 categories for any feature. This led me to observe that the dataset had 35 distinct teams, rather than the expected 32 NFL teams.

Furthermore, due to the nature of the dataset, it was frustrating to not be able to incorporate windowing in the streaming process. As the query for the pipeline-transformed test stream source was a selection of output columns, as the goal was to output predictions for a play scenario rather than some aggregated accuracy score, there were issues trying to set up a "complete" output mode for the sink. Instead, only "append" was the viable option. However, this led to problems since the separate query of the dynamic table shows the first twenty rows and the new information is at the end of the table. It was challenging to find workarounds to this situation. After numerous different attempts, I went back to retain the 'game_id' column and designed the dynamic table query to sort in descending order for that feature, to yield the most recent information. Ideally, with timestamps, these issues would be avoided, and each individual output would serve better for the objective of this project.

However, this gives an opportunity in the future to improve upon this project and build a more robust machine learning model, and find ways to incorporate streaming, even with the other complexities. Although the predictive power was underwhelming, the process shed light on the possibilities of distributed learning and was enjoyable to dig into the realm of sports analytics.