## **NFL Predictions - from Machine Learning Model to Production**

## ***Lessons learned in predicting NFL football scores and the end-to-end process***

It’s Wednesday morning, November 6, 2019, and our weekly NFL predictions are due for the football pool. We crank up our program, ai\_infinity, and out pops the predicted winners for the sixteen games.

The football application was chosen as we wanted to win the football pool but also for the learning experience. Similar to the business environment, the focus was on the end-to-end process and [weekly] results. Another advantage of the NFL football application choice is that there is a ton of data and many people can identify with the topic and data. The downside is that games are so unpredictable, many decided within the last seconds. We did learn that cursing the algorithm is as effective as cursing the TV.

My coding buddy and collaborator, David Fai, and I initiated this project as a pilot for some other applications that we wanted to pursue. The planning and scope of this project include:

* Scrape the websites for pertinent NFL statistics
* Incorporate data into a single structured database
* Build an ai / machine learning model to make predictions for each game in the 2019 season
* Put the model into production for weekly predictions
* Win the weekly football pool
* Learn a few things from the end-to-end implementation.

There are two main topics or lessons learned to share:

1. The end-to-end process, focusing on production (business results), fits well with Agile software development processes and,
2. An automated test documentation database was so crucial and natural for both the iterative process involved with machine learning modeling and the Agile software development process.

## End-to-End Process (business results) using Agile Software Development Processes

The end-to-end process (starting a project from scratch and producing business results) is business-critical. Many of the machine learning articles and courses focus on creating and evaluating different models using the various algorithms; but how does one put the model (predictions) into a production environment? Industry will take this approach; for example, Tesla or other autonomous developer collects data million of miles of driving and uses this data create and test models. Once the model is refined, it is then implemented in an individual commercial car that you and I might use (eventually); real time AI in this case. Our model, predicting NFL football scores for a football pool (the production step), is a much simpler process but is intended to outline the getting-to-production process.

Agile development for software development is characterized by the division of tasks into short phases of work, evaluation of results, and adaptation of plans. Given the iterative process of machine learning (the modeling), Agile development was an ideal process to use. Agile development was also ideal for the end-to-end focus as early results were the focus.

The following are the six major steps in our process. From an Agile development perspective, we iterated over these six steps many times, starting simply and adding functionality and performance with each iteration. Python was used for all the programming.

1. Extract and parse information from various sources (websites).
   1. Several websites were used as sources including NFL, The Football Database, ESPN, NFL Penalty Tracker, Pro Football Reference. Data included the outcomes for each regular season game along with corresponding team and player information going back seven years. The first iteration included only one website and a few football statistics.
   2. Data was programmatically scraped from the websites, cleaned, and inserted into a structured database (SQLite). Statistical data from each year, 2012-2018, was used. This required a bit of programming. Utilities, like Beautiful Soup, were somewhat helpful but since most of data was in tables, parsing into variables was a bit difficult. Other problems included inconsistency team names, cities, abbreviations, and team moves; for example, NYG (New York Giants), NYJ (New York Jets), LA Chargers vs San Diego Chargers, St. Louis Rams vs Los Angeles, etc. The solution was to immediate convert and store as team name, i.e. ‘Chargers’, ‘Giants’, ‘Jets’, ‘Rams’, etc.
   3. Teams and players turnover year-to-year, so our analysis in using such history was focus on determining what factors matter the most to winning and losing. These factors are discussed later as machine learning features.
   4. A scraping sample, ML-Predict-scrape, is provided on GitHub (<https://github.com/dsfai2020/nfl_machine_learning> ).
2. Insert data into the database (SQLite) with some minor calculations.
   1. Consolidated multiple data (statistics from the multiple websites) into a database was key in understanding, structuring, and extracting inputs. Approximately 80% of the programming was related to data preparation (items 1 and 2 in this list).
   2. The database included data on 1,500+ games. A good data structure was important for execution efficiencies and ease of development.
3. Determine what features to engineer and use.
   1. We started with 6 and ended with 64 features. Features included the teams win rate at home, team win rate away, quarterback statistics, …. We also engineered some features, like ‘momentum’ which we calculated as performance in preceding three games.
   2. Some of the feature engineering didn’t work and some did. For example, number of impact players (game changers from the NFL Top 100 players from prior year) contributed little to the predictions. When only the top 25 players were included, the predictions improved. In another feature engineering example, we thought percentage of game points scored in 4Q would be a good proxy for endurance, resolve, etc; modeling showed this as significant. A study of turnovers (takeaways and giveaways) showed more relevance than penalties for and against; although both turnovers and penalties contributed to model performance. Other features related to offense and defense differences; passing performance showed more relevance than running performance. Figure 1 shows some of this analysis.

|  |
| --- |
| **Figure 1. Feature Analysis (Relative Importance) Sample** |
| |  |  |  |  | | --- | --- | --- | --- | | **Feature** | | **Relative Importance** | **Description (all features were normalized)** | | **2** | AptsPos | 0.73 | Average points scored | | **22** | ATOnet | 0.69 | Turnovers Net (takeaways - giveaways) | | **4** | AqbrTot | 0.66 | Quarterback ranking | | **5** | AqbPts | 0.65 | Quarterback points | | **8** | AL5WinR | 0.64 | Prior 5 games win rate | | **6** | AqbPass | 0.56 | Quarterback passing | | **9** | AL5PtsP | 0.5 | Prior 5 games points scored | | **14** | AoScore | 0.48 | Offense ranking | | **23** | ATOpos | 0.48 | Turnovers positive (takeaways) | | **15** | AdScore | 0.31 | Defense ranking | | **16** | AimpactS1 | 0.3 | Impact players (ranking for tier 1) | | **17** | AimpactN1 | 0.29 | Impact players (number in tier 1) | | **30** | AFGgood | 0.19 | Field goal success rate | | **19** | AimpactN2 | 0.19 | Impact players (number in tier 2) | | **7** | AqbRun | 0.16 | Quarterback run | | **20** | AupSpH | 0.15 | Upset positive at home | | **12** | AL5upSp | 0.15 | Upset positive prior 5 games | | **18** | AimpactS2 | 0.15 | Impact players (number in tier 2) | | **31** | AFG50plus | 0.13 | Field goal 50+ yards | | **28** | ApenPosYds | 0.02 | Penalties (positive; drawn) | |

* 1. Sometimes, all of the features were used in modeling and other times, a subset was used in order to examine particular aspects or modeling behaviors. Feature lists were formalized and recorded to facilitated reproducibility. All features were normalized.

1. Extract input data from database to create the machine learning input.
   1. The database included a variety of data; for example, team and player statistics for a number of years. Each modeling test could include different inputs; for example, sometimes the first five games of the season were excluded. The feature set (football stats) used varied. The database was also updated weekly with scores, performance data, and roster changes (such as those for injuries).
2. Run various machine learning algorithms against the input data.
   1. Initially, the major algorithms from Scikit-learn were used in the modeling and testing. These included Logistical Regression, Random Forest, Support Vector Machine, Decision Trees, and Neural Networks. After about twenty features, when Offense and Defense ranking were added as features, the Neural Network algorithm performed the best. Neural Network models with PCA (Principal Components Analysis, to consolidate dimensions or features) were completed. For Neural Networks, SciKit-learn algorithm was initially used but we switched to Tensorflow to get more control. Tensorflow was abandon due to conflicts in the environment. The Pytorch Neural Network algorithm was then used and eventually established in the final model and production.
   2. Switching basic algorithms was programmatically easy; however, each algorithm has options that add complexity. The combinations of options were huge and, combined with changes to features and the number of features, created significant problems in determining and tracking what works, what doesn’t work, and in what circumstances. For example, with the Pytorch Neural Network, the number of hidden layers made a performance difference and the number of iterations had an effect. Trial and error, as usual, was a big part of the modeling process and the GRID search option was used to optimize.
   3. With machine learning, input data is divided into a training set (input used to train the model) and a test set (input used to test the model). Input, in our case, was from 1500+ games from 2012 to 2018. Typically, one randomly selects 66% of data from the input for training and the 34% remaining for testing. This approach was used initially but dividing input by year (2012 – 2016 for training and 2017-2018 for testing) also worked with no significant difference in results.
   4. A modeling sample, ML-Modeling-v8, is provided on GitHub (<https://github.com/dsfai2020/nfl_machine_learning> ).
3. Analyze and record the test results.
   1. The recording of test results was automated early in the development cycle. The next section provides more details on this automated test recording.
   2. From an Agile development perspective, we iterated through these six steps many times. For example, the first iteration included only one website with a couple of statistics, six features, and one algorithm that produced some very preliminary, but celebratory, predictions. Subsequent iterations added websites, football statistics, changes to data structure, features, algorithms, and new predictions.
   3. During the modeling, we continued to statistically look at common causes (random) of variation verses special causes of variation. In general, common causes are inherent to process (unsolvable); special causes are opportunities for feature engineering. Regarding the first few games of the season, we focused on ‘impact players’; that is, the game-changers. After each season, a list of Top 100 players, ranked 1 to 100, is available. For 2018, these include Tom Brady, Todd Gurley, Ezekeil Elliot, and Patrick Mahomes. As part of analysis and feature engineering, the impact player list was divided into Tier 1 (top 25), and Tier 2 (other 75). Tier 1 had impact; Tier 2 impact was minimal. These results and other are illustrated in Figure xx. The impact player list was updated weekly to account for injuries, suspensions, etc, and roster changes.
   4. Intuition, confirmed by testing, showed that the first few games of the season are harder to predict than the sequent games. The first games are influenced by seasonal changes to rosters and strategies. The last game(s) are also problematic - influenced by playoff bound teams playing second strings to reduce injury risks and contending teams going all-in.
   5. Variability of predictions indicates that additional feature engineering could be beneficial. For example, in our modeling we frequently use years 2002 to 2017 for model development (training) and year 2018 for model validation (testing). In testing, performance was measured weekly for the 15 to 16 games played per week; that is, accuracy for week = games correctly predicted / total game played. Higher variance of weekly accuracy indicated that additional feature work would be beneficial. Figure 2 illustrates the test performance.

|  |
| --- |
| **Figure 2. Test Predictions, Sample, Year: 2018, Weeks 6 to 17** |
| Predict year:2018, week: 6, Correct %: 80.0, Missed:3  Predict year:2018, week: 7, Correct %: 64.3, Missed:5  Predict year:2018, week: 8, Correct %: 85.7, Missed:2  Predict year:2018, week: 9, Correct %: 76.9, Missed:3  Predict year:2018, week: 10, Correct %: 85.7, Missed:2  Predict year:2018, week: 11, Correct %: 92.3, Missed:1  Predict year:2018, week: 12, Correct %: 86.7, Missed:2  Predict year:2018, week: 13, Correct %: 68.8, Missed:5  Predict year:2018, week: 14, Correct %: 75.0, Missed:4  Predict year:2018, week: 15, Correct %: 81.2, Missed:3  Predict year:2018, week: 16, Correct %: 62.5, Missed:6  Predict year:2018, week: 17, Correct %: 75.0, Missed:4  **Errors totals: 40, Games total: 178, Error %: 22.5** |

## Automated test documentation database

Machine learning is an iterative process based on many variables, choices, and trials and errors. Originally, we didn’t plan on a test documentation database for this simple application but quickly the iterations and additions of function overwhelmed our tracking of what works, what didn’t, and under what circumstances. To track testing, an automated test documentation database was created to track tests – algorithm used, key algorithm options, input data used, features used, and test results. The test database easily allowed us to compare results of different modeling tests and allowed us to quickly reproduce prior tests, when needed.

Over 200 modeling tests were documented in the test database (this does not address code testing and debugging). To reduce clutter, some modeling testing were not recorded and, in some cases, modeling tests were purged. Figure 3 is an illustration from the test database.

|  |  |
| --- | --- |
| **Figure 3. Test documentation database snippet and description of fields** | |
|  | |
| **Field** | **Description** |
| date | Date of test |
| inCnt | Number of inputs (games) |
| testAcc | Accuracy of test set (% of games predicted correctly) |
| trainAcc | Accuracy of training set (% of games predicted correctly) |
| period1 | Period of years used (range from 2012 to 2018) |
| period2 | Period of weeks used (regular seasons included up to 17). Sometimes the first 5 games of each season or the last two were excluded. |
| num\_of\_feat | Number of features (ranged from 6 to 64) |
| features | Pointer to one of the 14 different feature sets. |
| testsize | Usually 30% to 35% of input (games) used in test (remainder used for training |
| alg | Algorithm used (about 14 total including major options) |
| script | Name and version of modeling program |
| notes | More specific options and parameters used in specific test |
| precision | Standard statistical measure |
| recall | Standard statistical measure |
| f1 | Standard statistical measure |
| true pos | Standard statistical measure |
| false pos | Standard statistical measure |
| false neg | Standard statistical measure |
| true neg | Standard statistical measure |

The test database was created early in the development cycle and our development cycle time improved significantly. The process for updating test database was automated and fairly simple – collect the information in variables (such as test size) and, at end of test, write the data to the database as illustrated in Figure 4.

|  |
| --- |
| **Figure 4. Automating the test documentation** |
| # Save test results to TestDB  if save\_rpt == 'Y': # this is set at beginning (key inputs section)  # Insert into TestDB (as record in 'test' table)  cur.execute("""INSERT OR IGNORE INTO test  (project,date,inCnt,testAcc,trainAcc,  period1,period2,num\_of\_feat,features, testsize,alg,script,notes,  truePos, falsePos, falseNeg, trueNeg,precision,recall,f1)  VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)""" ,  (project,runDate,inCnt,test\_acc,train\_acc,  period1,period2,Num\_of\_feat,features,testsize,alg,script,notes,  true\_pos, false\_pos, false\_neg, true\_neg, precision, recall, f1 ))  print ('\nReport saved.')  conn.commit() # flush to database  conn.close() # Close database |
| Note: This code is from ML-Modeling-v8, which is on GitHub (<https://github.com/dsfai2020/nfl_machine_learning>). |

This concept of a test document database was subsequently copied into our gaming app to collect data on human and bot behaviors (player data) that we plan to use as input to machine learning to create a super-bot for the game. The existing bots are mathematic (but not ai based) algorithms. This process and behavioral data will allow us to create a new Neural Network ai bot.

## Conclusion … and having a good coding partner

The end-to-end project developed a variety of technical skills and a systems appreciation. This project developed skills for web scraping, database design and implementation, feature engineering, modeling, and analysis. Subject matter expertise was also required and developed; for example, understanding the basic aspects of football (the specific industry and application). A great team, coding buddy, or co-founder was critical; for example, when the project was in serious trouble and struggling with problems – such as in the example the slow cycle time on the Agile development cycle (the overwhelming numbers of modeling parameters and features). Stepping back, collaborating, and brainstorming winning plays, such as, the automated test documentation, was part of the playbook. David Fai, my coding partner, and I started in May 2019 (pictured in Figure 5) and finished for the game one kickoff on September 5, 2019. Besides David’s technical contributions, football knowledge, and computer gaming experience, David’s endless optimism was ever-present. Now, we are ready to move onto our next project.

|  |  |
| --- | --- |
| **Figure 5. Brainstorming the Production Script** | |
|  |  |
| *David Fai and Raymond Ernst* | *Production Script Outline* |

Historically, modeling of this nature was limited to large corporations, universities, and government agencies. The skills, compute infrastructure, software tools, and other resources were limited and expensive. Fortunately, advanced modeling can now be run on a standard PC and the software tools (like the database and machine learning algorithms identified herein) are open source and readily available. And, at a minimum, we’ve proven that out-of-pocket expenses can pretty much be limited to Starbucks. More complex or image-based models will over-power the standard PC but consumer-level solutions such as Graphic Processing Units (GPUs) and cloud-base subscriptions are available. Skills and techniques can be learned through videos, publications (such as Medium) and free, or inexpensive, online courses, such as those from Coursera, Edx, and Udacity. Small businesses, entrepreneurs, individuals, and citizen scientists, be they in California or Ghana, can readily participate and contribute. Unimaginable opportunities and improvements in healthcare, medicine, economics, life sciences, environment, finance, etc. are being addressed … and await.