

Predicting Point Spread in NFL Games

By: Christina Wadsworth (cwads@stanford.edu) & Francesca Vera (fvera@stanford.edu)
Project TA: Michael Zhu

Introduction

Since 1985, the Harris Poll¹ has conducted a survey that asks American adults what their favorite sport is. In every single poll, pro-football topped the list. Most recently in late 2015, 33% of the group responded with pro-football, a resounding win over over baseball, which trailed by eighteen percentage points. Thus, the National Football League is arguably the most valuable professional sports league across the country. More specifically, over \$1 billion is legally bet on NFL games every year. Being able to predict the winner of a game according to Vegas guidelines can be highly profitable. We were thus motivated to find ways in which we could use machine learning principles to thrive in this industry.

We approached this by considering the two main ways to bet on an NFL game.

- I. You can place a **line** bet by simply selecting a winning team.
- II. You can bet on a game's point **spread**, which is the difference in points between opposing teams. For example: if Vegas predicts team A to beat team B by 7 points, but the final outcome shows team A won by only 5 points, those who bet on team B win.

For all our models, we input data for one game, which included the teams playing and Vegas prediction - among other features. We first used classifier models to output a predicted winning team for that game. To check accuracy, we compared our predictions to the actual winners of the games, as you would in Vegas. We then used regression models to output a predicted point spread for that game. While we also compared this to the actual spreads of each game, Vegas is only concerned about falling on the "right" side of the spread. Following from the above example, this means any prediction 6 points or less would have won. We used whether or not we beat Vegas as our error metric. "Beating Vegas" more than 50% of the time is incredibly difficult, so our goal was to do it consistently enough to make a profit.

Related Works

Due to the popularity of the NFL, there is no shortage of websites and online publications that attempt to guide bettors on how to predict spreads and winners of games. However, because most of these sources write in a blog-like style, a lot of their considerations involve coach and player choices, momentum streaks, and specific one-off details (e.g. a particular quarterback not liking cold weather) - things that are not easily quantifiable. Though there may be some machine learning involved, it usually stays hidden and so is not a useful reference for this project other than looking at what features sports writers focus on.

A popular publication that is more transparent about how it numerically calculates point spread is FiveThirtyEight, which uses "Elo Ratings" - a metric FiveThirtyEight founder Nate Silver is famous for. After obtaining the team's ratings, a simple equation is used:

$$P(\text{team A wins}) = \frac{1}{1 + 10^{-\frac{EloDiff}{400}}},$$

where $EloDiff = Elo\ Rating_{(Team\ A)} - Elo\ Rating_{(Team\ B)}$. Elo Rating mostly relies on final game scores and home/away location. However, even then, the website doesn't suggest using the NFL Elo to "beat Vegas"² since it only achieves about 51% accuracy, which isn't enough to cover the house cut.

Some past CS229³ projects have also tried to predict the outcomes of NFL games, but those projects focused on simply predicting a winner. There were two key differences between those projects and ours. First, we opted to use ensemble methods whereas both did not. Second, we found that they used many more features than we did: we kept ours relatively simple with only 7 while one project used 33. This was because we felt some facets of the game like quarterback performance and personnel changes were encapsulated by the ESPN/expert opinion. Our models performed better than all past related CS229 projects.

¹ http://www.theharrispoll.com/sports/Americas_Fav_Sport_2016.html

² <http://fivethirtyeight.com/datalab/introducing-nfl-elo-ratings/>

³ <http://cs229.stanford.edu/proj2006/BabakHamadani-PredictingNFLGames.pdf>,
<http://cs229.stanford.edu/proj2011/Shau-PredictingOutcomesOfNFLGames.pdf>

Data Set

Most of our data came from Pro Football Reference⁴, a website that compiles statistics on various NFL players, teams, and seasons. For our project, we used a scraper that created a spreadsheet such that each row represents one game, with columns for Away Team, Home Team, Away Score, Home Score, Vegas Line etc. We also created a second sheet that had each team's previous season's ranking, and ESPN's preseason "Power Ranking"⁵. The "Power Rankings" were created by a panel of more than 80 writers, editors, and personalities before being published by ESPN, and so offer an expert perspective. We ended up with data dating all the way back to the 2002 NFL season (almost 4000 games), but we decided to focus on seasons 2010-onwards to keep the prediction as relevant to today as possible.

An example is shown below:

Week	homeTeam	awayTeam	startTime	weather	surface	vegasLine
1	Minnesota Vikings (6-10-0)	New Orleans Saints (11-5-0)	8.41pm	73 degrees	outdoor	New Orleans Saints -5.0

Year	Team	PrevRank	ESPNRank
2010	Minnesota Vikings	2	5
2010	New Orleans Saints	1	2

We then made a few scripts to pull out each team's streak, as well as cleaned the rest of the data. Start time was rounded to the nearest hour, vegasLine expressed the point differential for the favored team, and homeTeam and awayTeam were only expressed as their rankings.

Features

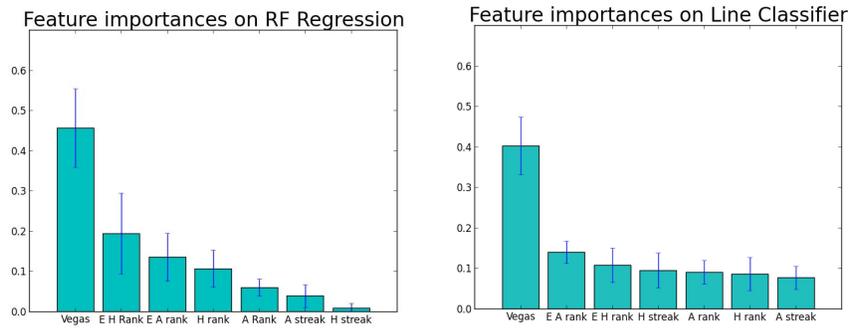
When considering which features to use, we thought about what aspects the game most directly impact the outcome. After experimenting with various factors like weather, surface, and day, our preliminary results shows that these features didn't correlate with correct predictions. Another set of features we looked at was the average yards gained, time of possession, and turnovers for each team. Initially these appeared useful, but we stuck with features that were known before the game was even played so our model could be used to bet on a winning team or a winning spread before play begins. We ended up using features the subset of features that most correlated with our models predicting correctly, as follows:

- **Each team's win streak:** Because this roughly translates to a team's momentum, it is a good indicator of how well a team has been performing coming into a game.
- **Each team's NFL ranking from the previous season:** It is realistic that certain franchises are consistently good across seasons, reflected by a consistently high previous season ranking.
- **Each team's ESPN preseason "Power Ranking":** This provides an expert's perspective on a team's potential for the upcoming season.
- **The predicted Vegas spread:** This is the basis on which we are judging whether or not a prediction is correct.

On the next page, we show a feature importance graph of our Random Forest spread regression model. As you can see, our most impactful feature was Vegas' predicted spread. This makes sense because Vegas is generally regarded as unbeaten. ESPN's Power Ranking is the next most impactful feature, which would suggest that ESPN Power Rankings are a better indicator of a team's upcoming performance than the team's performance in the previous season. It also reflects how the average fan typically trusts an expert opinion. The remaining features were somewhat helpful with our prediction, and if we removed the previous season's ranking or team streak, our accuracy dropped by a few percentage points.

⁴ <http://www.pro-football-reference.com/>

⁵ http://www.espn.com/nfl/powerrankings/_year/2010/week/1



Above, feature graphs show the Gini importance of each feature used. On the left hand, our regression model that predicted spread, and on the right hand, our line classifier that predicted an outright winner

Methods

We used classification algorithms from scikit learn for the **line** prediction, which simply picks a winner. After testing **SVMs**, **Stochastic Gradient Descent**, **Logistic Regression**, and **tree based** methods, we found the most promising results from Logistic Regression and the tree based methods. This was because we had to deal with overfitting - with tree-based models especially, it was very easy to focus on refining hyperparameters to optimize these models.

Logistic regression takes the output of linear regression and maps it to $\{0,1\}$ based on the output, turning a regression into a classifier. We used the L1 regularization, as it performed better than the L2 norm. The L1 norm, as used in our model:

$$\min_{w \in \mathbb{R}^p} \frac{1}{n} \|\hat{X}w - \hat{Y}\|^2 + \lambda \|w\|_1$$

As mentioned earlier, our methods also stood out from other projects because we chose to focus on ensemble models. An ensemble model uses multiple learning models, then aggregates or averages them. The idea is that combining several base estimators will give a better prediction and a more generalized prediction than just one estimator. The methods discussed below are ensemble methods.

The first tree based method we tried was a **Random Forest** Classifier. Random Forest randomly samples from the data set with replacement, running a tree based classifier on each subset. Scikit-learn's default implementation of splitting at each node is gini impurity, which calculates how often a randomly chosen element in the set would be labeled incorrectly if labeled randomly from the label distribution in that randomly chosen subset. Instead, we used entropy (aka information gain) as the splitting method on our classifier, which performed better. Entropy and gini impurity, below:

$$I_E(f) = - \sum_{i=1}^m f_i \log_2 f_i \quad I_G(f) = \sum_{i=1}^J f_i(1 - f_i) = \sum_{i=1}^J (f_i - f_i^2) = \sum_{i=1}^J f_i - \sum_{i=1}^J f_i^2 = 1 - \sum_{i=1}^J f_i^2 = \sum_{i \neq k} f_i f_k$$

where f_i is the probability that the i th example is chosen. This basically means that entropy is splitting based on the metric that adds the most information gain to our method. Information gain is used to reduce a decision tree's bias. We also prevented overfitting by selecting a maximum tree depth of 5.

We explored various other tree-based methods: **Extra-trees**, **Gradient Boosting**, and **Ada Boosting**. Extra-trees is similar to Random Forest, except the splitting metric is random. Gradient Boosting is similar to Random Forest as well. However, Gradient Boosting is a bagging ensemble method, while Random Forest is an averaging method. This means that Gradient Boosting builds many weak learners on our data set and updates one classifier with the weak learners' information. Random Forest, on the other hand, builds many full decision tree methods on subsets of our data and averages them.

Ada Boosting is also a bagging ensemble method and uses decision trees as weak learners, just as Gradient Boosting does, but the two methods have different loss functions. Ada Boosting gives more weighting to samples that fit the worst; as a result, it is sensitive to noisy data or outliers. When combining weak learners, Ada Boosting weights as follows, where each $h(x_i)$ is a hypothesis for each sample in the training set. The sum over the training error of the resulting classifier (as follows) is minimized. $F_{t-1}(x_i)$ is the previous classifier that has been built up.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

We modified each of these ensemble methods' hyperparameters to prevent overfitting. We modified Gradient Boosting to have a maximum tree depth of 1, Ada Boosting to have a maximum number of 200 estimators, and Extra-Trees has a maximum depth of 4 and uses a maximum number of 4 trees.

For our **spread** model, we tried the regression versions of the ensemble models detailed above as well as various other models, such as SVMs, Linear Regression, and Stochastic Gradient Descent. The ensemble models gave the best results. We ended up using a Random Forest Regressor, but gini impurity worked better for a splitting algorithm than the information gain algorithm that our classifier used. We also modified the max depth to also equal 5. Gradient boosting and Extra-trees used the same depths as their classifier correspondents, at 1 and 4 respectively. Ada boosting used only 100 estimators.

Experimental results and analysis

Note: Part way through our analysis, when we added ESPN Power Rankings to our model, our prediction rate jumped through the roof. We tested our code, and attributed this to really good ESPN experts whose rankings were highly correlated with how a team would do in the upcoming season. At this point in time, we were looking at a ~75% chance at beating Vegas on our spread model. During the poster session, we were met with some shock at our results. We thought about what beating Vegas 75% of the time really means (hint: it doesn't happen), and we dug into our code to figure out if something was wrong. Sure enough, we have learned to never add magic numbers to our code again. And, on the bright side, our numbers are still pretty good.

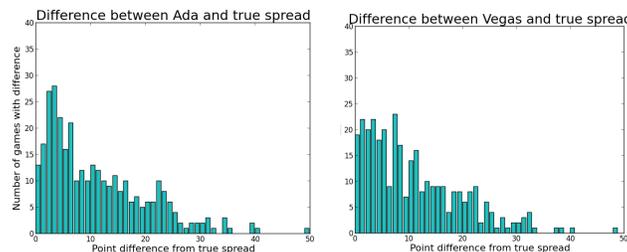
After deciding on a feature set, we ran an **SVM** and **Stochastic Gradient Descent**, and were looking at 60-65% winner prediction on our test set and ~55% winner prediction on our training set. Our spread fared no better. Both models were beating Vegas close to 50% of the time, and we might as well have done a coin flip.

When we ran **Logistic Regression** and our **decision tree ensemble methods**, we got much better results. The line results are a percentage of the time we predicted the winning team. The spread results are calculated based on whether we would have won the Vegas spread. We ran our models with a **training set of size 655** and a **test set of 328**.

Model	Line train	Line test	Spread train	Spread test
Logistic Regression	.7	.68	N/A	N/A
Random forest	.72	.7	.65	.55
Extra-trees	.71	.69	.62	.56
Gradient Boosting	.72	.72	.6	.56
Ada Boosting	.76	.72	.6	.59

Our best classifier (relevant to **line** bets) is Ada Boosting, although all the other classifiers we looked at are around 70-75%, which are better than any past 229 results. Previous projects only attempted logistic regression with results that were on par with the logistic regression model we ran.

Our best regressor (relevant to **spread** bets) is the same model, Ada Boosting. We're able to beat Vegas almost 60% of the time on test. Here it is important to keep our metric in mind. This isn't the percentage of the time that we predict the spread correctly or that we are within 5 points of the spread. This is the percentage of the time that we are *beating Vegas*, making it more useful for real-life applications.



We also looked at the differences between predicted spread and the game's true spread for both our prediction and Vegas' prediction, which are graphed above. As you can see, our results are more left-skewed than Vegas' differentials. This

suggests that we predict closer to the true spread than Vegas does, which would support the above results that we are beating Vegas 55-60% of the time.

Since we chose to tackle a popular problem, we were able to compare our results with the results of different sources. As mentioned earlier, Nate Silver's ELO Rating model only achieved 51% accuracy, yet was considered by our group as one of the more valuable related works. The main reason that our model is better than his is that he is using Naive Bayes and not taking into account the same features that we are. Silver focuses on ranking and wins thus far in the season. Our ESPN Power Ranking feature takes into account injuries, coach switches, etc. Our winning streak is also correlated to some of these features. While Nate Silver's 51% is admittedly not great, not many college students can say they are beating him.

Another interesting comparison is between our results and how well ESPN/sport experts tend to do since the Power Ranking was one of our most crucial features. ESPN mostly predicts the outright winner of the game, staying away from spread prediction. They are correct about 68-70% of the time, which is about as good as our Logistic Regression and the past CS229 projects that also predicted NFL winners. Our best model's test set prediction is a few points higher at 72%.

To put our results into perspective, assuming 1:1 odds, we're expected to make a consistent profit after the house cut. While our success rate is pretty good, there is still much room for improvement. One source of potential error is feature set. The possible features available was somewhat limited. Our two data sets were all information from before the season started, and ideally we would have had enough time to build a pipeline that funneled week by week data to us in order to take into account a team's performance in the first few weeks of the season when looking at a later game. One thing we did do was calculated win streak, which was somewhat correlated to our predicted values, but the least correlated of the features we ended up using. Since the features from before the season were more correlated than the week-by-week features we did try, we decided it wasn't worth it to build a pipeline that updates each week with game outcomes.

Additionally, our initial predictions were massively overfitting because we used sparse vectors with strings for team names instead of the team's current rank. We were able to rectify that by selecting a feature set that was only integers, but this was an important consideration as to whether or not to add other features that would have been potentially helpful. One thing we weren't able to explore was specific quarterbacks or starting lineups as a consideration since it would add too many branches to the tree models we used. Overall, we're really excited about our results, but have definitely thought a lot about various improvements that could be made.

Future Work

We have yet to use our models to predict a week's worth of games this season. Doing so would add value to this project, and eventually allow us to try our hand at sports betting - which is what we set out to explore in the first place.

Additionally, our initial data set considered many other features, such as yards per game, players, and team efficiency, that we were not able to use for this project. Thus, we're excited about the possibility of a model that includes these features that experts and fans often turn to when rating a team's performance.

Moreover, we could look into creating a model that updates as the game progresses (even including injuries or personnel change) to dynamically predict each team's percent chance of winning. This is still relevant to sports betting since betting is often open for the entire game - with the odds simply changing as the game goes on. A similar thing is done for World Series poker, where the probability of each player winning adjusts as more cards are revealed.

Conclusion

Considering the inherent difficulty of predicting a game that is affected by so many unquantifiable and unpredictable factors, we present promising results. Further, our project predicts both the winner and the point spread, whereas other related works would opt to only focus on one type of bet.

Overall, our success is validated when we compare it to the results of other groups doing the same type of work. Our best classifier, Ada Boosting, predicts correctly around 70-75% of the time, beating previous 229 project results. Our best regressor beats Vegas around 60% of the time, which not only would turn a profit, but also performs better than reputable sports analyst Nate Silver. We ultimately attest this to our very selective set of features, and the use of ensemble methods.

References

The Harris Poll (2016, January 26). *Pro Football is Still America's Favorite Sport* [Online]. Available: http://www.theharrispoll.com/sports/Americas_Fav_Sport_2016.html

Sports Reference LLC (2016). *Pro Football & NFL History* [Online]. Available: <http://www.pro-football-reference.com/years/>

ESPN's Power Panel (2010, September 8). *2010 NFL Power Rankings: Week 1* [Online]. Available: http://www.espn.com/nfl/powerrankings/_/year/2010/week/1

Silver, N (2014, September 14). *Introducing NFL Elo Ratings* [Online]. Available: <http://fivethirtyeight.com/datalab/introducing-nfl-elo-ratings/>

Shau, A., "Predicting outcomes of NFL Games," CS229: Machine Learning, Stanford, CA. December 16, 2011.

Hamadani, B., "Predicting the outcome of NFL games using machine learning," CS229: Machine Learning, Stanford, CA. 2006.