A Predictive Model for Betting On NFL Teasers

Marion Sudvarg, Arivan Thillaikumaran
December 9, 2016

Under Instruction of Dr. Yixin Chen, CSE 543

Abstract

Predicting NFL betting lines is a difficult task; there is much variability that can affect a game, and unlike other sports, there are only 16 games per season, so it's hard to create a model to predict the outcome of these games. We instead attempt to create models involving nonlinear optimization techniques to predict the outcome of teaser bets. We find that certain models work better than others, and our results show that the model that we find can in fact predict the outcomes of games based on the lines with an acceptable accuracy.

Background

Betting on the NFL is just a fun pastime for some. Office pools, Superbowl wagers, fantasy teams, and online contests like Fanduel are all very popular outlets for people to sink their cash for the sake of a little competition (or the hopes of a big payout). But in Las Vegas, a whole industry banks on making money from bets. It starts with line makers, statisticians and sports analyzers who attempt to estimate the spread between final scores of each game. The sportsbooks then take bets against these lines, shifting the lines as conditions change. The bookies aren't competing against the betters; instead, they attempt to settle the line so that equal pools of money are bet on both teams. In the end, losers owe the bookie $110 for every $100 they would have won. That 10% cut is where the bookies make their money.

The lines themselves represent the predicted difference in the final score. A team's line is added to the actual final score of the team being bet on; if that sum is greater than the other team's score, the wager is good. For example, a game between the Falcons and the Saints might set the line at -11 for Saints (which corresponds to +11 for the Falcons). This means that the Saints are predicted to win by 11 points. Any bets placed on the Saints would require them to win by more than 11 points for the bet to pay off. For a final score of Saints 54 to Falcons 44, bets on the Falcons would actually win, because the Falcons' final score plus the line (44+11=55) beats the Saints' score.

Predicting these lines is a difficult task. With so much variability that can affect a game, and given the relatively few games played per season, it's hard to create a model to accurately predict the final spread. As Michael Konik has revealed to his readers, however, line makers have gotten better over time. Lines have been regressing toward the actual final point spreads of games. This opens the door to a whole new type of bet: the teaser.

Teaser bets work by assuming that the final point spread of a game is close to the betting line for the game. Lines are teased up by 6 points. Using the example above, a teaser bet on the Saints (who were favored by 11 points) would use a -5 instead of a -11 line. This means that the Saints would have to win by more than 5 points (instead of 11). A teaser bet on the underdog, the Falcons, would have the line teased from +11 to +17. A teaser bet on either team for a game with a final point spread within 6 points of the line, then, would be successful.

There's a tease to the teaser bet, however: one must actually select two teams to bet against the adjusted line, and both selections must be correct, to actually win the money. Since every loss means losing 110% of the money gained for a win, we calculate the required expectation of winning to break even as follows:

$$p-1.1(1-p)=0 \rightarrow p\approx 0.524$$

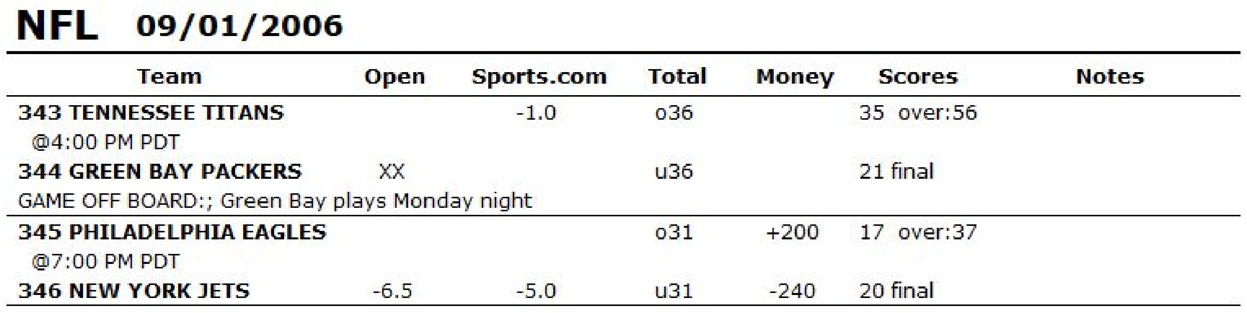
So to at least break even on standard bets, one would have to only place bets when there is at least a 52.4% expectation of winning. On a teaser, since two bets must be correct, this expectation becomes 72.4% (since $0.724^2\approx 0.524$).

That brings us to our goal: to find a predictive model that determines which teaser bets have at least a 72.4% expectation of success.

Collecting Data

Game data was scraped from the website http://www.scoresandodds.com. Its archive provides game dates, times, home and away teams, lines, and final scores from September 1, 2006 to the present. Data was collected through November 28, 2016.

For games occurring between 9/1/2006 and 9/11/2006, game data is accessed via URLs like the following: http://archive.scoresandodds.com/pgrid_20060901.html, where 20060901 corresponds to games on the date 9/1/2006. These pages display data like in Fig. 1 below. Games are separated by horizontal lines. Within each game, the away team is listed at the top, and the home team is at the bottom. The Sports.com column only lists the line for the favored team, so in Fig. 1 the Tennessee Titans are at -1.0 and the New York Jets are at -5.0. The underdog team has the opposite line, so the Green Bay Packers are at +1.0 and the Philadelphia Eagles are at +5.0. The final scores are listed in the Scores column, so the Tennessee Titans beat the Green Bay Packers 35-21.

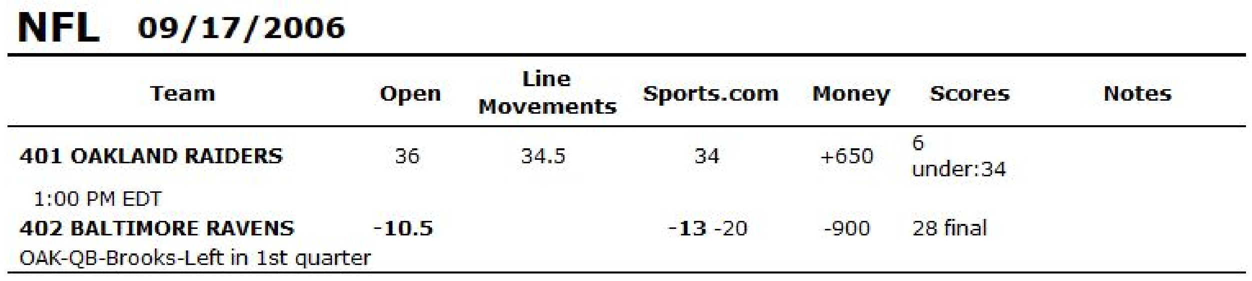


NFL 09/01/2006

Team	Open	Sports.com	Total	Money	Scores	Notes
343 TENNESSEE TITANS		-1.0	o36		35 over:56	
@4:00 PM PDT						
344 GREEN BAY PACKERS	XX		u36		21 final	
GAME OFF BOARD:; Green Bay plays Monday night						
345 PHILADELPHIA EAGLES			o31	+200	17 over:37	
@7:00 PM PDT						
346 NEW YORK JETS	-6.5	-5.0	u31	-240	20 final	

Figure 1: Scores and Odds Data for 9/1/2016

For games occurring between 9/17/2006 and 10/1/2006, the URL format remains the same. The only difference in page format can be seen in Fig. 2, where a Line Movements column has been added. The game line for the favored team is in bold in the Sports.com column. So in Fig. 2, for example, the Baltimore Ravens are at -13, so the Oakland Raiders are at +13.

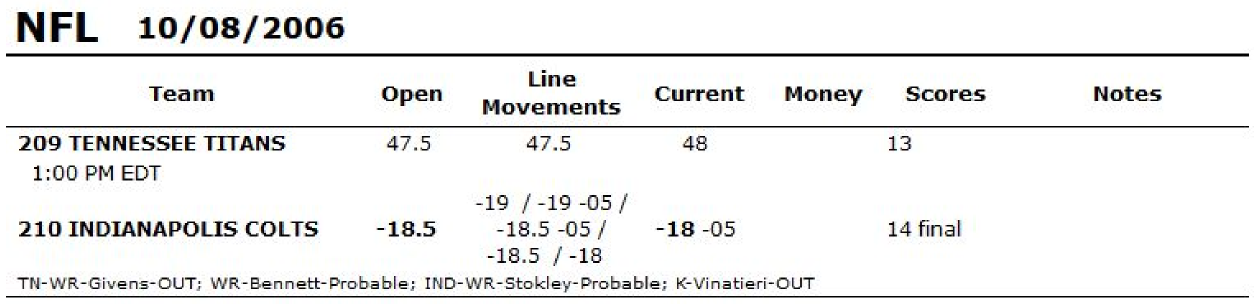


NFL 09/17/2006

Team	Open	Line Movements	Sports.com	Money	Scores	Notes
401 OAKLAND RAIDERS	36	34.5	34	+650	6 under:34	
1:00 PM EDT						
402 BALTIMORE RAVENS	**-10.5**		**-13** -20	-900	28 final	
OAK-QB-Brooks-Left in 1st quarter						

Figure 2: Scores and Odds Data for 9/17/2006

For games occurring between 10/2/2006 and 1/18/2009, we again see a change in columns. Now, as seen in Fig. 3, the Sports.com column has been replaced with a Current column, but otherwise the data is displayed the same as in Fig. 2.

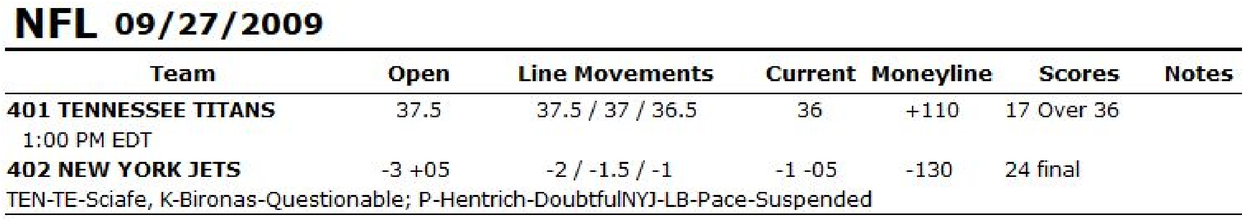


NFL 10/08/2006

Team	Open	Line Movements	Current	Money	Scores	Notes
209 TENNESSEE TITANS	47.5	47.5	48		13	
1:00 PM EDT						
210 INDIANAPOLIS COLTS	**-18.5**	-19 / -19 -05 / -18.5 -05 / -18.5 / -18	**-18** -05		14 final	
TN-WR-Givens-OUT; WR-Bennett-Probable; IND-WR-Stokley-Probable; K-Vinatieri-OUT						

Figure 3: Scores and Odds Data for 10/8/2006

For games occurring between 9/3/2009 and the present, the URLs now look like http://www.scoresandodds.com/pgrid_20090903.html. Otherwise, the data is displayed very similarly as before. One notable difference, however, is that the line is no longer displayed in bold. So in Fig. 4 below, the line for the New York Jets is -1.



NFL 09/27/2009

Team	Open	Line Movements	Current	Moneyline	Scores	Notes
401 TENNESSEE TITANS	37.5	37.5 / 37 / 36.5	36	+110	17 Over 36	
1:00 PM EDT						
402 NEW YORK JETS	-3 +05	-2 / -1.5 / -1	-1 -05	-130	24 final	
TEN-TE-Sciafe, K-Bironas-Questionable; P-Hentrich-DoubtfulNYJ-LB-Pace-Suspended						

Figure 4: Scores and Odds Data for 9/27/2009

One last item of note: Where the line is listed as "PK," instead of being listed as a number, neither team is favored. In this case, a standard bet on the winning team, with no score adjustment, would be successful. A teaser bet would adjust the score by +6; if this beats the opposing (unadjusted) score, the teaser bet is good.

Methodology

Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification, or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic regression. As an optimization problem, L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2}w^T w + C\sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

Support Vector Regression

Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, and a vector $y \in \mathbb{R}^n$ ε-SVR solves the following primal problem:

$$\min_{w,b,\zeta,\zeta^*} \frac{1}{2}w^T w + C\sum_{i=1}^{n}(\zeta_i + \zeta_i^*)$$

$$\text{subject to } y_i - w^T\phi(x_i) - b \le \varepsilon + \zeta_i,$$
$$w^T\phi(x_i) + b - y_i \le \varepsilon + \zeta_i^*,$$
$$\zeta_i, \zeta_i^* \ge 0, i = 1, ..., n$$

It's dual is:

$$\min_{\alpha,\alpha^*} \frac{1}{2}(\alpha - \alpha^*)^T Q(\alpha - \alpha^*) + \varepsilon e^T(\alpha + \alpha^*) - y^T(\alpha - \alpha^*)$$

$$\text{subject to } e^T(\alpha - \alpha^*) = 0$$
$$0 \le \alpha_i, \alpha_i^* \le C, i = 1, ..., n$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

Stochastic Gradient Descent Regression

This algorithm implements a plain stochastic gradient descent learning routine with loss function set to the squared loss. The gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate.)

Neural Network/Perceptron Learning

The backpropagation algorithm for neural networks solves a nonlinear optimization problem. The training or learning of the model is to determine the weights w from the input layer to the hidden layer and the weights w 0 from the hidden layer to the output layer to minimize a loss function over the training data D. Here (x1, x2, · · · , xp) ∈ D is a given data point of p dimension. Here x0 = 1 is introduced as a bias term, which is something similar to the constant term in linear regression. What functions f used here determine what kind of perceptrons we use. A common nonlinear function is the sigmoid function y=f(u)=$\frac{1}{1+e^{-ku}}$, u=$\sum_i x_i\, w_i$, where 1/k is the temperature parameter. This is simply a logistic regression that typically works well on data with nonlinear structures. We define the loss function as

$$L(w) = ½(y_d - t_d)^2$$

where t_d is the target value (or label) of a given example d, y_d is the prediction of the perceptron on d, and w are the weights of links in the network used to evaluate yd. Note that L(w) is a function of w since y_d depends on w. *Gradient descent* is perhaps the most general and effective paradigm for optimization over complex functions. It is often the method to use for problems where there is no closed form solutions. Our objective is to find the local minimum of the smallest value that we can find with the given computation. To achieve our objective, we make a move downward along the direction of the gradient, ∇w, at our current position w on the cost surface toward a local minimum. The backpropagation algorithm for neural networks works as the following: Initially, the network is initialized with a set of random initial values of the model parameters w. Next, for a given example d, predict the outcome of the model yd by computing values through the network from the input layer to the output layer. Compute the error $½(y_d - t_d)^2$, where t_d is the actual value of d. Finally, back propagate the error at the output layer backward into the network to compute gradient: Compute the gradient of the loss function L with respect to the parameter w backward from the output layer back to the input layer. Consider the weight w'_{ij} from a hidden node h_i to an output node y_j . In the following, we also use y_j for the output value from node y_j for a given example d whose actual value at node y_j is simplified to t_j . We need to compute

$$\nabla w'_{ij} = \partial \mathrm{L} \frac{\partial L}{\partial w'_{ij}} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial u'_j} \frac{\partial u'_j}{\partial w'_{ij}}, \text{ where}$$

$$\frac{\partial L}{\partial y_j} = y_j - t_j \text{ , } \frac{\partial y_j}{\partial u'_j} = y_j(1-y_j), \text{ and } \frac{\partial u'_j}{\partial w'_{ij}} = \frac{\partial}{\partial w'_{ij}} \sum_{i=1}^{K} w'_{ij}\, h_i = h_i$$

We can now write:

$$\nabla w'_{ij} = \frac{\partial L}{\partial w'_{ij}} (y_j - t_j) y_j (1 - y_j) h_i$$

The second equation arises from the derivative of the sigmoid function, and the K in the third equation is the number of output nodes that link to the hidden node i. Now we consider the weight wki from an input node k to a hidden node i. We need to compute:

$$\nabla w_{ki} \frac{\partial L}{\partial w_{ki}} = \sum_{j=1}^{M} \left(\frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial u'_j} \frac{\partial u'_j}{\partial h_i}\right) \left(\frac{\partial h_i}{\partial u_i} \frac{\partial u_i}{\partial w_{ki}}\right)$$

where M is the number of output nodes linking to hidden node i

From earlier, $\frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial u'_j} = (y_j - t_j) y_j (1 - y_j)$ and $\frac{\partial u'_j}{\partial h_i} = w'_{ij}$

Since $h_i = \mathrm{f}(u_i) = 1/(1+\mathrm{e} - u_i)$ and $y_j = \sum_{k=1}^{p} w_{ki}\, x_k$

$$\frac{\partial h_i}{\partial u_i} = h_i(1 - h_i) \text{; } \frac{\partial u_i}{\partial w_{ki}} = x_k$$

Add up, and we have:

$$\nabla w_{ki} \frac{\partial L}{\partial w_{ki}} = h_i(1 - h_i)\, x_k \sum_{j=1}^{M} (y_j - t_j) y_j (1 - y_j)\, w'_{ij}$$

Finally, we can update our model parameters:

$$w'_{ij} \leftarrow w'_{ij} - \eta \nabla w'_{ij} \text{; } w_{ki} \leftarrow w_{ki} - \eta \nabla w_{ki} .$$

Alternate among examples: Repeat steps 2 to 4 on a new example d until convergence, i.e., no change to any parameter, or some stopping criterion is met, e.g., running out of time or diminishing gradient. Repeat this entire process with a new set of (random) initial parameters w for a given number of times; return the model parameters that give rise to the best solution found.

Experiments and Analysis

Implementation

We used Python and the SciKit-Learn package to implement all of our algorithms. The pandas package provides flexible data structures and is the fundamental, high-level building block in doing complex, real world data analysis in Python. SciKit-Learn employs the Limited Memory BFGS algorithm to solve nonlinear optimization problems.

Evaluation

We use cross-validation to assess the quality of our estimator models. The goal of cross-validation is to define a dataset to "test" the model in the training phase in order to limit problems such as overfitting or to give an insight on how the model will generalize to an independent dataset. The first round of cross validation involves partitioning a sample of data into complementary subsets, performing the training on one subset and validating the training analysis on the other set (the test set.) To reduce variability, multiple rounds of cross-validation are performed using different partitions.

We calculate the mean accuracy on the given test data and labels. We calculate the coefficient of determination, or R^2 score, for our logistic regression, stochastic gradient descent, and support vector regression model's predictions. The R^2 score provides a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model. The coefficient R^2 is defined as (1 - u/v), where u is the regression sum of squares and v is the residual sum of squares. The best possible score is 1.0. When applying a model to test data, the R^2 value can be negative, since the model can be arbitrarily worse that that which would be obtained from training over the test data. R^2 values are returned for the logistic regression, stochastic gradient descent regression, and support vector regression models, since this is a standard method measuring how well these models perform in the statistics field. For the perceptron model, however, SciKit-Learn does not supply an R^2 value. It instead returns a score based on mean accuracy on the given test data.

Initial Models

We split our data into two sets. The first, comprising of data from the 2006-2014 seasons, constitutes training data that we used to create prediction models. The second, comprising of data from the 2015 season through 11/28/2016, was used to test the accuracy of our models.

For each game, we calculated a teaser value for both teams. This value attempts to represent how well a teaser bet on that team would have done. It is calculated as follows:

$$\text{Teaser} = \text{Score} + \text{Line} - \text{Opposing Score} + 6$$

For example, when the Pittsburgh Steelers played the Baltimore Ravens on January 3, 2013, the Steelers were the favored team with a line of -3. The Ravens ended up beating the Steelers with a score of 30 to the Steelers' 17. The teaser score for the Steelers was thus:

$$17 + (-3) - 30 + 6 = -10$$

The teaser score for the Ravens was then:

$$30 + (+3) - 17 + 6 = 22$$

A negative teaser score means that a teaser bet placed on that team would have been a failure, while a positive score corresponds with a successful bet.

We then attempted to construct a predictive model for teaser scores based on game season, time, timezone, home team, away team, and home team line. The away team line was ignored since it is linearly dependent on the home team line. The home team, away team, timezone, and season were all treated as categorical variables, while the line was treated as a numerical variable. Game time was converted to a decimal value between 0 and 1, using a 24 hour period as the scale (e.g. a noon game would have a time value of 0.5).

For our first model, we used SciKit-Learn's linear regression model against our training data with the away team teaser score as the dependent variable. This resulted in an R^2 score of 0.0048, which is very poor. Despite this, we tested the coefficients found against our test data set. This resulted in an even worse R^2 score: -0.0033. In fact, the linear model predicted a teaser score between about 3.97 and 8.31 for every away team in the test data. Since this value is always positive, it would mean that we should please a teaser bet on any away team, so it hardly constitutes a predictive model at all!

We then attempted a support vector machine analysis of the same data. When the model was run against the test data, the R^2 score was -0.0073, indicative of an even worse model. Furthermore, the mean absolute error resulting in this analysis was approximately 9.45, a large value considering that the average away team teaser score is only about 6.0.

Finally, we implemented SciKit-Learn's neural network to create a best fit model on our training data. This was run against the test data, and the R^2 score was calculated. We found it to be very close to that calculated by SVM, again approximately -0.0073.

We remodeled our data, this time using the season as a numerical parameter (instead of treating it as categorical and label encoding it). Only very slightly better results were to be had. The SVM model had an R^2 value of 0.0043 and a mean absolute error of 9.41. The neural network analysis had an R^2 value of -0.0055.

Additional attempts were made with the same modeling approaches, this time using the home team teaser score as the dependent variable. Similarly poor results were found. These initial models suggest that teaser scores are too spread out to be predicted with any confidence as variables dependent on game data such as game season, time, teams, and starting lines.

Appendix A: initial_models.py shows the Python code used to construct these initial models. Appendix B: initial_data.csv shows a sample of the data used to construct these models. These are real records from the training data, though just a subset. The testing data has the same structure.

Corrected Models

With unsuccessful first attempts, a different approach was taken to analyzing the data. Rather than attempting to predict teaser scores, we attempted to model the expectation of accurately selecting a teaser bet. We did away with the parameters of game time, time zone, and which teams played. Instead, for each distinct line value, we analyzed the ratio of teaser wins to games played for each season. The data was analyzed from 4 perspectives: predicting teasers wins for the home team, away team, favored team, and underdog team.

We put our initial data into a SQL table, and ran the following query to get a new dataset of away team teaser win ratios based on season and the away team line:

```
SELECT T.away_line, T.Season, Sum(T.away_win) AS Wins, Count(T.away_win) AS
Totals, [Wins]/[Totals] AS Ratio
FROM Game_Data As T
GROUP BY T.away_line, T.Season
ORDER BY T.away_line, T.Season;
```

A similar query was run against the home team lines to get home team teaser ratios, and again for the favored team and underdog team for each game. This resulted in four data sets to analyze.

A cursory linear regression analysis was performed in Excel. Selecting a few lines for away teams, a linear regression of teaser ratios by season was plotted for each line over time. Several of these gave fairly good fits, as can be seen in Fig. 5 below. With such promising results, we began more complex, nonlinear modeling on the four data sets.

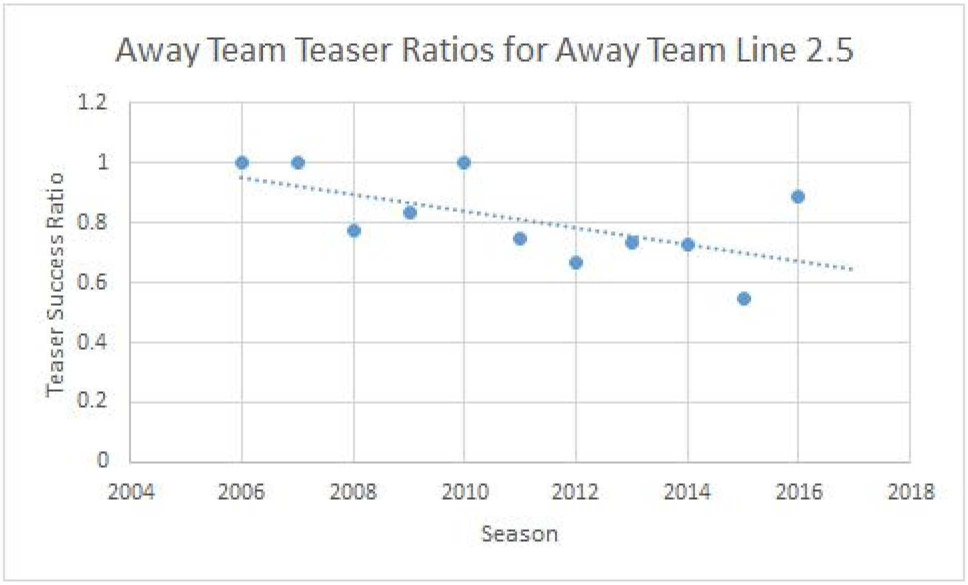


Figure 5: Teaser Success Ratio (Away Team Line 2.5) by Season with Linear Best Fit

We used these four models to model the data collected for our new approach: logistic regression, support vector regression with the radial basis function kernel, stochastic gradient descent, and a perceptron neural network. Of these, the logistic regression estimator and the perceptron returned very promising results. Results are shown below in Fig. 6, taken over average iterations of differing cross validation values.

Teaser Set	Logistic Regression (R^2)	Support Vector Machine (R^2)	Stochastic Gradient Descent (R^2)	Perceptron (Accuracy)
Away Teasers	0.677419354	-0.22868373	-6.23783735	0.67741935
Home Teasers	0.69585253	0.002976771	-1.19063271	0.304147465
Favored Teasers	0.765151512	-0.0363167	-.000000001	0.772727272
Underdog Teasers	0.840909091	-0.246485	-.000000004	0.818181818

Figure 6: Model Results

The Stochastic Gradient Descent Regression algorithm did not work well, and we believe this is because it is most useful when the number of samples and the number of features is very large;

however, as can be seen from our datasets, the number of features in our problem is very small. The support vector machine also returned neutral results; an R^2 score of zero represents a constant model that always predicts the expected value of the objective function, regardless of input features; thus this model is not useful.

Appendix C: final_models.py shows the Python code used to construct these final models. Appendix D: final_data.csv shows a sample of the data used to construct these models. These are real records from the training data, though just a subset. The testing data has the same structure.

Conclusion

We ran four different learning algorithms that employ nonlinear optimization techniques in finding their solution, and find that two return results that indicate that our data can be modeled in a functional manner.

Given a season and line, we can use the logistic regression function to predict teaser win ratios with high confidence. This ratio, then, is the expectation that a teaser bet will win. So given lines before a game, we can determine with high confidence which favored and underdog team teaser selections have a probability over 72.4% of being winners. Picking based on these results gives a high expectation of winning the bet. The same can be said for the perceptron model.

Picking favored and underdog teasers is sufficient: for any game, given the opening line, we know which team is favored and which is the underdog. Picking based on home team and away team is then unnecessary, which is fortunate since these models give lower confidence values based on that methodology of selection.

Using this model, we hope to provide sports betters (including ourselves) a profitable means of betting on NFL teasers.

Future Work

While we've gotten good results on logistic regression and perceptron models for predicting teaser picks, there is still room for improvement. With R^2 values and accuracies under 1.0, we know that our models are imperfect. Other parameters are also influencing the teaser success ratios.

To improve our analysis of the data, other dimensions could be added to our data set. Analyzing time of day, game day (if it falls on a Saturday, Sunday, or Monday), and game number are all viable parameters. However, care will need to be taken to determine a way to allow a teaser success ratio model to still exist with additional parameters added.

The additional parameters mentioned are all easily derived from the data already scraped. Even further dimensions could potentially be analyzed, with more specific football statistics like coaches, quarterbacks, game delays, injuries, etc. all being analyzed. We hope to continue tweaking these models to pick teasers with better and better confidence.

Contributions

Marion Sudvarg was responsible for scraping data from the Scores and Odds website. Arivan Thillaikumaran wrote the initial Python code to analyze the data using SciKit-Learn's linear regression, SVM, and neural network models. Arivan and Marion worked together to implement the different models and review their efficacy. When the initial approach didn't work, Arivan and Marion worked together to formulate a different means of approaching the data. Ultimately, Marion came up with the idea of analyzing the pattern of teaser ratios by season for each line value. He created the SQL database and queries to calculate these ratios and did the initial linear regression tests in Excel. Arivan took this data and implemented the nonlinear optimization techniques to create more accurate predictive models. Both contributed to the writing of this report.

References

Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2007. Print.

Konik, Michael. *Smart Money: How the World's Best Sports Bettors Beat the Bookies out of Millions*. New York: Simon & Schuster, 2005. Print.

Schmidt, Mark, Le Roux, Nicolas, and Bach, Francis. "Minimizing Finite Sums with the Stochastic Average Gradient." INRIA - SIERRA Project. Departement d'Informatique de l'Ecole Normale Superieure. Paris, 2016.

Smola, Alex and Schölkopf, Bernhard. "A Tutorial on Support Vector Regression." *Statistics and Computing* 14: 199-222, 2004.

Zhang, Weixiong. Washington University in St. Louis Fall 2016 CSE 514 Lecture Notes

Appendix A: initial_models.py

```
from sklearn import preprocessing
from sklearn import linear_model
from sklearn import svm
from sklearn import neural_network
from sklearn import metrics
import pandas as pd

##Away Team Teaser Training Data
trainset = pd.read_csv('python_train.csv',low_memory=False,
        usecols=['Season','Time','Timezone','home team','away team','home team line','Away
        Team Teaser'])
x_train = trainset.drop('Away Team Teaser', axis=1)
le_home = preprocessing.LabelEncoder()
le_away = preprocessing.LabelEncoder()
le_season = preprocessing.LabelEncoder()
le_timezone= preprocessing.LabelEncoder()
x_train['home team'] = le_home.fit_transform(x_train['home team'])
x_train['away team'] = le_away.fit_transform(x_train['away team'])
x_train['Timezone'] = le_timezone.fit_transform(x_train['Timezone'])
x_train['Season'] = le_season.fit_transform(x_train['Season'])
y_train = trainset['Away Team Teaser']

#Linear Regression Model
print ("Away Team Teaser Linear Regression")
linreg = linear_model.LinearRegression()
linreg.fit(x_train,y_train)
print (linreg.coef_)
score = linreg.score(x_train,y_train)
print (score)

#Testing Linear Regression Model
testset = pd.read_csv('python_test.csv',low_memory=False,
        usecols=['Season','Time','Timezone','home team','away team','home team line','Away
        Team Teaser'])
x_test = testset.drop('Away Team Teaser', axis=1)
x_test['home team'] = le_home.fit_transform(x_test['home team'])
x_test['away team'] = le_away.fit_transform(x_test['away team'])
x_test['Timezone'] = le_timezone.fit_transform(x_test['Timezone'])
x_test['Season'] = le_season.fit_transform(x_test['Season'])
y_test = testset['Away Team Teaser']
score = linreg.score(x_test,y_test)
print (score)

for i in linreg.predict(x_test):
        print (i)

#Support Vector Machine
print ("Away Team Teaser SVM")
svr = svm.SVR()
svr.fit(x_train, y_train)
testset = pd.read_csv('python_test.csv', low_memory=False,
        usecols=['Season','Time','Timezone','home team','away team','home team line','Away
        Team Teaser'])
x_test = testset.drop('Away Team Teaser', axis=1)
x_test['home team'] = le_home.fit_transform(x_test['home team'])
x_test['away team'] = le_away.fit_transform(x_test['away team'])
x_test['Timezone'] = le_timezone.fit_transform(x_test['Timezone'])
x_test['Season'] = le_season.fit_transform(x_test['Season'])
y_test = testset['Away Team Teaser']
preds = svr.predict(x_test)
print (metrics.mean_absolute_error(y_test, preds))
score = svr.score(x_test,y_test)
print (score)

#Neural Network
print ("Away Team Teaser Neural Network")
nn = neural_network.MLPRegressor()
nn.fit(x_train,y_train)
nn.predict(x_test)
score = nn.score(x_test,y_test)
print (score)

#Do not treat Season as Categorical
trainset = pd.read_csv('python_train.csv',low_memory=False,
        usecols=['Season','Time','Timezone','home team','away team','home team line','Away
        Team Teaser'])
x_train = trainset.drop('Away Team Teaser', axis=1)
le_home = preprocessing.LabelEncoder()
le_away = preprocessing.LabelEncoder()
le_timezone= preprocessing.LabelEncoder()
x_train['home team'] = le_home.fit_transform(x_train['home team'])
x_train['away team'] = le_away.fit_transform(x_train['away team'])
x_train['Timezone'] = le_timezone.fit_transform(x_train['Timezone'])
y_train = trainset['Away Team Teaser']

#Support Vector Machine
print ("Away Team Teaser SVM")
svr = svm.SVR()
svr.fit(x_train, y_train)
```

```
testset = pd.read_csv('python_test.csv', low_memory=False,
        usecols=['Season','Time','Timezone','home team','away team','home team line','Away
        Team Teaser'])
x_test = testset.drop('Away Team Teaser', axis=1)
x_test['home team'] = le_home.fit_transform(x_test['home team'])
x_test['away team'] = le_away.fit_transform(x_test['away team'])
x_test['Timezone'] = le_timezone.fit_transform(x_test['Timezone'])
y_test = testset['Away Team Teaser']
preds = svr.predict(x_test)
print (metrics.mean_absolute_error(y_test, preds))
score = svr.score(x_test,y_test)
print (score)

#Neural Network
print ("Away Team Teaser Neural Network")
nn = neural_network.MLPRegressor()
nn.fit(x_train,y_train)
nn.predict(x_test)
score = nn.score(x_test,y_test)
print (score)
```

Appendix B: initial_data.csv

```
date,Season,Time,Timezone,home team,away team,home team line,away team line,Home Team Teaser,Away Team Teaser,Favored Team Teaser
9/3/2015,2015,0.791666667,EDT,indianapolis colts,cincinnati bengals,-1.5,1.5,1.5,10.5,1.5
9/3/2015,2015,0.791666667,EDT,new york jets,philadelphia eagles,3,-3,15,-3,-3
9/3/2015,2015,0.791666667,EDT,miami dolphins,tampa bay buccaneers,-3.5,3.5,-2.5,14.5,-2.5
9/3/2015,2015,0.791666667,EDT,green bay packers,new orleans saints,-2.5,2.5,31.5,-19.5,31.5
9/3/2015,2015,0.791666667,EDT,atlanta falcons,baltimore ravens,-2.5,2.5,4.5,7.5,4.5
9/3/2015,2015,0.8125,EDT,detroit lions,buffalo bills,-3.5,3.5,9.5,2.5,9.5
9/3/2015,2015,0.8125,EDT,new england patriots,new york giants,1,-1,4,8,8
9/3/2015,2015,0.8125,EDT,pittsburgh steelers,carolina panthers,1.5,-1.5,-9.5,21.5,21.5
9/3/2015,2015,0.8125,EDT,washington redskins,jacksonville jaguars,-3.5,3.5,1.5,10.5,1.5
9/3/2015,2015,0.833333333,EDT,tennessee titans,minnesota vikings,-3,3,10,2,10
9/3/2015,2015,0.833333333,EDT,st. louis rams,kansas city chiefs,-3,3,-4,16,-4
9/3/2015,2015,0.833333333,EDT,dallas cowboys,houston texans,3,-3,16,-4,-4
9/3/2015,2015,0.833333333,EDT,chicago bears,cleveland browns,-1,1,29,-17,29
9/3/2015,2015,0.875,EDT,denver broncos,arizona cardinals,-5,5,-1,13,-1
9/3/2015,2015,0.916666667,EDT,san francisco 49ers,san diego chargers,-3,3,5,7,5
9/3/2015,2015,0.916666667,EDT,seattle seahawks,oakland raiders,-2.5,2.5,13.5,-1.5,13.5
9/10/2015,2015,0.861111111,EDT,new england patriots,pittsburgh steelers,-7,7,6,6,6
9/13/2015,2015,0.541666667,EDT,chicago bears,green bay packers,6,-6,4,8,8
9/13/2015,2015,0.541666667,EDT,houston texans,kansas city chiefs,0,0,-1,13,13
9/13/2015,2015,0.541666667,EDT,new york jets,cleveland browns,-3.5,3.5,23.5,-11.5,23.5
9/13/2015,2015,0.541666667,EDT,buffalo bills,indianapolis colts,0,0,19,-7,-7
9/13/2015,2015,0.541666667,EDT,washington redskins,miami dolphins,4,-4,3,9,9
9/13/2015,2015,0.541666667,EDT,jacksonville jaguars,carolina panthers,3,-3,-2,14,14
9/13/2015,2015,0.541666667,EDT,st. louis rams,seattle seahawks,3.5,-3.5,12.5,-0.5,-0.5
9/13/2015,2015,0.670138889,EDT,arizona cardinals,new orleans saints,-2,2,16,-4,16
9/13/2015,2015,0.670138889,EDT,san diego chargers,detroit lions,-3.5,3.5,7.5,4.5,7.5
9/13/2015,2015,0.684027778,EDT,tampa bay buccaneers,tennessee titans,-3,3,-25,37,-25
9/13/2015,2015,0.684027778,EDT,oakland raiders,cincinnati bengals,3,-3,-11,23,23
9/13/2015,2015,0.684027778,EDT,denver broncos,baltimore ravens,-5,5,7,5,7
9/13/2015,2015,0.854166667,EDT,dallas cowboys,new york giants,-7,7,0,12,0
9/14/2015,2015,0.798611111,EDT,atlanta falcons,philadelphia eagles,3,-3,11,1,1
9/14/2015,2015,0.930555556,EDT,san francisco 49ers,minnesota vikings,3,-3,26,-14,-14
9/17/2015,2015,0.850694444,EDT,kansas city chiefs,denver broncos,-3,3,-4,16,-4
9/20/2015,2015,0.541666667,EDT,carolina panthers,houston texans,-3,3,10,2,10
```

Appendix C: final_models.py

```
from sklearn import linear_model
from sklearn import svm
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

#Away Team Teaser Linear Regression
trainset = pd.read_csv('teasers_underdog.csv',low_memory=False)
train_set = trainset.drop('Wins', axis=1)
x_train = train_set.drop('Totals', axis=1)
x= x_train.drop('Ratio', axis=1)

y= trainset['Ratio'].astype(np.int)
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=.4, random_state=0)

#Logistic Regression
print ("Logistic Regression")
logreg = linear_model.LogisticRegression()
logreg.fit(X_train,Y_train)
print ('R^2 score')
print (logreg.score(X_test, Y_test))
print ("coefficients")
print (logreg.coef_)

#SVM
svr = svm.SVR()
svr.fit(X_train, Y_train)
print ('SVR R^2 score')
print (svr.score(X_test,Y_test))

#stochastic gradient descent
print ("SGD Regression")
sgd = linear_model.SGDRegressor()
sgd.fit(X_train, Y_train)
print ('SGD R^2 score')
print (sgd.score(X_test,Y_test))

#perceptron
print ("Perceptron")
perceptron = linear_model.Perceptron()
perceptron.fit(X_train, Y_train)
print ('perceptron R^2 score')
print (perceptron.score(X_test,Y_test))
```

Appendix D: final_data.csv

```
away_line,Season,Wins,Totals,Ratio
-2.5,2006,3,5,0.6
-2.5,2007,6,11,0.545454545
-2.5,2008,1,3,0.333333333
-2.5,2009,6,7,0.857142857
-2.5,2010,3,3,1
-2.5,2011,5,6,0.833333333
-2.5,2012,6,8,0.75
-2.5,2013,8,10,0.8
-2.5,2014,3,3,1
-2.5,2015,3,8,0.375
-2.5,2016,6,11,0.545454545
-2,2006,6,6,1
-2,2007,4,4,1
-2,2008,2,3,0.666666667
-2,2009,0,1,0
-2,2010,4,5,0.8
-2,2011,3,5,0.6
-2,2012,1,6,0.166666667
-2,2013,2,4,0.5
-2,2014,1,1,1
-2,2015,5,6,0.833333333
-2,2016,2,2,1
```