NFL Big Data Bowl
How many yards will an NFL player gain after receiving a handoff?

OR-568: Applied Predictive Analytics

George Mason University

Final Team Project Report

Subject: NFL Big Data Bowl

Data Analysts Working on this Project:

Venkata Lakshmi Nikhila Katiki
Bhagirath Reddy Vulupala
Sindhuja Pulluri
Revanth Pavan Nimmala
Gnana Prasuna Pulipaka
Kausik Valeti

NFL Big Data Bowl
How many yards will an NFL player gain after receiving a handoff?

Abstract:

American football is a complex sport. From the 22 players on the field to specific characteristics that ebb and flow throughout the game, it can be challenging to quantify the value of specific plays and actions within a play. Fundamentally, the goal of football is for the offense to run (rush) or throw (pass) the ball to gain yards, moving towards, then across, the opposing team's side of the field in order to score. And the goal of the defense is to prevent the offensive team from scoring. Deeper insight into rushing plays will help teams, media, and fans better understand the skill of players and the strategies of coaches. It will also assist the NFL and its teams evaluate the ball carrier, his teammates, his coach, and the opposing defense, in order to make adjustments as necessary. Additionally, the winning model will be provided to the NFL's Next Gen Stats group to potentially share with teams.

Introduction:

This is a featured code competition announced by Kaggle, the data we are working on is extracted from the National football league site. This dataset contains Next Gen Stats tracking data for running plays. The prediction we must do is How many yards will an NFL player gain after receiving a handoff?

The variables included in this dataset are as follows: (177.21 MB and 49 columns)

DATA SET VARIABLES AND ITS DETAILS:

GameId - a unique game identifier	**Dis** - distance traveled from prior time point, in yards	**YardLine** - the yard line of the line of scrimmage	**HomeScoreBeforePlay** - home team score before play started	**PlayDirection** - direction the play is headed	**PlayerCollegeName** - where the player attended college	**WindDirection** - wind direction
PlayId - a unique play identifier	**Orientation** - orientation of player	**Quarter** - game quarter (1-5, 5 == overtime)	**VisitorScoreBeforePlay** - visitor team score before play started	**TimeHandoff** - UTC time of the handoff	**Position** - the player's position	**WindSpeed** - wind speed in miles/hour
Team - home or away	**Dir** - angle of player motion	**Game Clock** - time on the game clock	**NFL Id Rusher** - the NFL Id of the rushing player	**TimeSnap** - UTC time of the snap	**HomeTeamAbbr** - home team abbreviation	**Humidity** - humidity
X - player position along the long axis of the field.	**NFL Id** - a unique identifier of the player	**Possession Team** - team with possession	**Offense Formation** - offense formation	**Yards** - the yardage gained on the play	**VisitorTeamAbbr** - visitor team abbreviation	**Temperature** - temperature (deg F)
Y - player position along the short axis of the field	**DisplayName** - player's name	**Down** - the down (1-4)	**Offense Personnel** - offensive team positional grouping	**PlayerHeight** - player height (ft-in)	**Week** - week into the season	**GameWeather** - description of the game weather
S - speed in yards/second	**Jersey Number** - jersey number	**Distance** - yards needed for a first down	**DefendersInTheBox** - number of defenders lined up near the line of scrimmage, spanning the width of the offensive line	**PlayerWeight** - player weight (lbs)	**Stadium** - stadium where the game is being played	**Turf** - description of the field surface
A - acceleration in yards/second^2	**Season** - year of the season	**Field Position** - which side of the field the play is happening on	**DefensePersonnel** - defensive team positional grouping	**PlayerBirthDate** - birth date (mm/dd/yyyy)	**Location** - city where the game is being player	**Stadium Type** - description of the stadium environment

Fig: 1 – Data set Variables Before Pre-Processing

Data Pre-Processing:

- The sample of our data set, size of 176 MB having 509,763 rows and 49 columns. Once we pre-process the data, the rows are cut down to 375,786 rows.
- We pre-processed the data by removing the text columns which is not necessary for predicting our response variable (Yards), we also removed Null values which does not impact much on our dataset to predict our response variable. Because the null values are in our response variable, So we thought of having enough data with more than 300 thousand rows.
- These are the columns that we have used after pre-processing:

NFL Big Data Bowl
How many yards will an NFL player gain after receiving a handoff?

X,Y,S,A,Dis,Orientation,Dir,Yardline,Quarter,Down,Distance,HomeScoreBeforePlay,Visitor ScoreBeforePlay,DefendersInTheBox,Yards,PlayerHeight,PlayerWeight,WindSpeed,Humidit y,Temperature. Which are all numeric.

Correlation Plot:

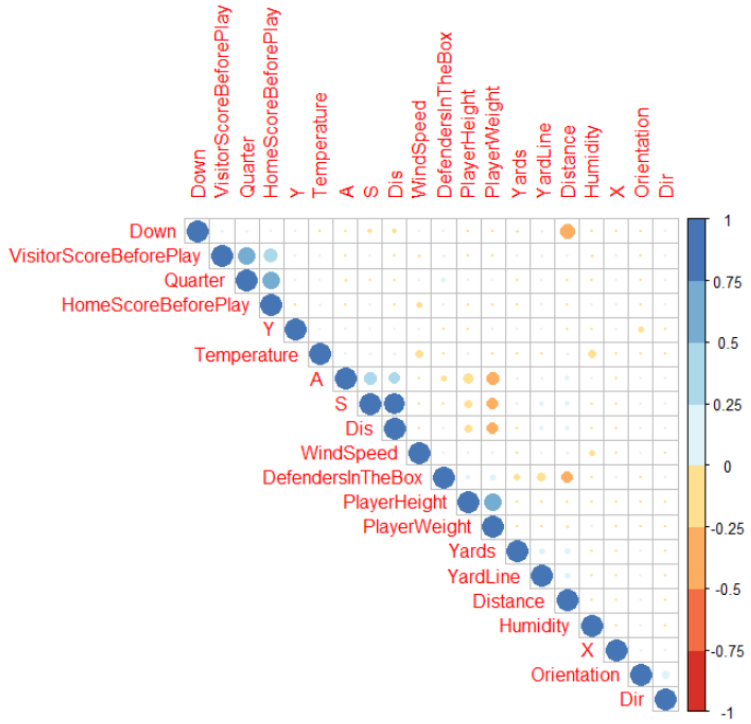

Fig: 2 – Correlation plot after pre-processed variables.

From Fig-2, our response variable Yards neither positively correlated nor negatively correlated to other variables in our dataset. So, we would like to further explore how our response variable is related to other variables.

This analysis uses the data and the methods listed below to address our question:

Linear Models: Partial Least Squares (PLS), Principal Component Regression (PCR), Principal Component Analysis (PCA), Penalized Regression Model (Ridge & Lasso)

Non-Linear Model: Extreme Gradient Boosting (XGBOOST), Random Forest, Neural networks.

PLS:

PLS is known as 'Projection to latent structures', but popularly known as 'Partial Least Squares'. It is kind of statistical method that bears some relation to principal components regression. Instead of finding hyperplanes of maximum variance between the response and independent variables, it finds a linear regression model by projecting the predicted variables and the observable variables to a new space. The PLS family of methods are known as bilinear factor models. Partial least squares Discriminant Analysis **(PLS-DA)** is a variant used when the Y is categorical.

NFL Big Data Bowl
How many yards will an NFL player gain after receiving a handoff?

PLS is a "*Dimension Reduction Technique"* with some similarity to principal component analysis. This technique reduces the predictors to a smaller set of uncorrelated components and performs least square regression on these components instead of on original data. PLS is a supervised procedure and using library 'pls'. PLS find the number of components that simultaneously summarize the variation of predictors while at the same time being optimally correlated with outcome. These components are then used to fit the regression model.

Note on PLS:

- We can preprocess with centered and scaled to enhance performance.
- Tuning parameter: ncomp
- It doesn't assume that predictors are fixed unlike multiple regression. Which means that predictors can be measured with error, making PLS more robust to measurement uncertainty.
- In PLS, the emphasis is on developing models, which means that it is not usually used to screen out variables that are not useful in explaining the response.
- The number of principle components are generally chosen by cross validation
- Predictor and outcome variables should be generally standardized, to make variables comparable.

Before Cross Validation:

- Here we are training the data with method pls and we not using any cross validation and preprocessing here. We see that output has set the ncomp = 3 as best optimal value and having the smallest vales where RMSE value is '6.506' and R Squared values as 0.004

```
> pls <- train(x = data, y = nfl_train$Yards, method = "pls")
> pls
Partial Least Squares

250527 samples
    23 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 250527, 250527, 250527, 250527, 250527, 250527, ...
Resampling results across tuning parameters:

  ncomp  RMSE      Rsquared      MAE
  1      6.518283  0.0006334809  3.903584
  2      6.512282  0.0024692192  3.898676
  3      6.506683  0.0041924154  3.893781

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was ncomp = 3.
```

After Cross Validation:

Here we have applied cross validation using 'trControl' and we see a decrease in RMSE value to '6.483'. The difference in RMSE value before and after cross validation is ***'0.023'***.

```
> pls <- train(x = data, y = nfl_train$Yards, method = "pls", tuneGrid = expand.grid(ncomp = 3), trControl = ctrl)
> pls
Partial Least Squares

250527 samples
    23 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 225474, 225475, 225475, 225475, 225473, 225475, ...
Resampling results:

  RMSE      Rsquared     MAE
  6.483276  0.004762665  3.888682

Tuning parameter 'ncomp' was held constant at a value of 3
```

- Predicting the value of pls using 'predict' function and we see RMSE as '6.465'.

```
> plsPred = predict(plsFit, nfl_test, ncomp = 3)
> plsValue = data.frame(obs = nfl_test$Yards, pred = plsPred[,,1])
> defaultSummary(plsValue)
       RMSE    Rsquared         MAE
6.465145528 0.005170411 3.884827128
```

- Summary of test pls with RMSE as '6.465'

```
> testpls <- data.frame(obs = nfl_test$Yards, pred = predict(pls, nfl_test))
> defaultSummary(testpls)
       RMSE    Rsquared         MAE
6.465145528 0.005170411 3.884827128
```

- Tuning pls with tune grid ratio of ncomp from 1 to 20. And we see RMSE value as '6.429' the lowest at ncomp = 20. We observe that as the number of ncomp of increases, RMSE value is gradually decreasing.

```
> plsTune <- train(x = data, y = nfl_train$Yards, method = "pls", tuneGrid = expand.grid(ncomp = 1:20), trControl = ctrl)
> plsTune
Partial Least Squares

250527 samples
    23 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 225474, 225475, 225475, 225474, 225473, 225475, ...
Resampling results across tuning parameters:

  ncomp  RMSE      Rsquared     MAE
   1     6.494830  0.000964279  3.898982
   2     6.490774  0.002203486  3.895540
   3     6.482741  0.004660424  3.888830
   4     6.481459  0.005051543  3.887530
   5     6.478955  0.005825277  3.883247
   6     6.474030  0.007335950  3.874463
   7     6.466036  0.009794180  3.855115
   8     6.464125  0.010384672  3.851711
   9     6.463536  0.010566863  3.848977
  10     6.462063  0.011008234  3.845383
  11     6.445659  0.016055051  3.814033
  12     6.445076  0.016232922  3.813588
  13     6.444875  0.016299394  3.813790
  14     6.442496  0.017021431  3.812501
  15     6.441330  0.017375528  3.812333
  16     6.439048  0.018068582  3.812118
  17     6.435859  0.019028263  3.810717
  18     6.433540  0.019731234  3.810149
  19     6.430950  0.020506743  3.807984
  20     6.429797  0.020859951  3.807734

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was ncomp = 20.
```

Fig: 3 – PLS Tuning grid

- Variable Importance: It ranks the variables based on the normalized weight of predictor. 'varImp' function is used to find the variable importance. We found that Defenders in the Box ranks top among all the existing values, which means that if there is a certain number of defenders in the box, there is more probability of scoring Yards or Lose Yards.

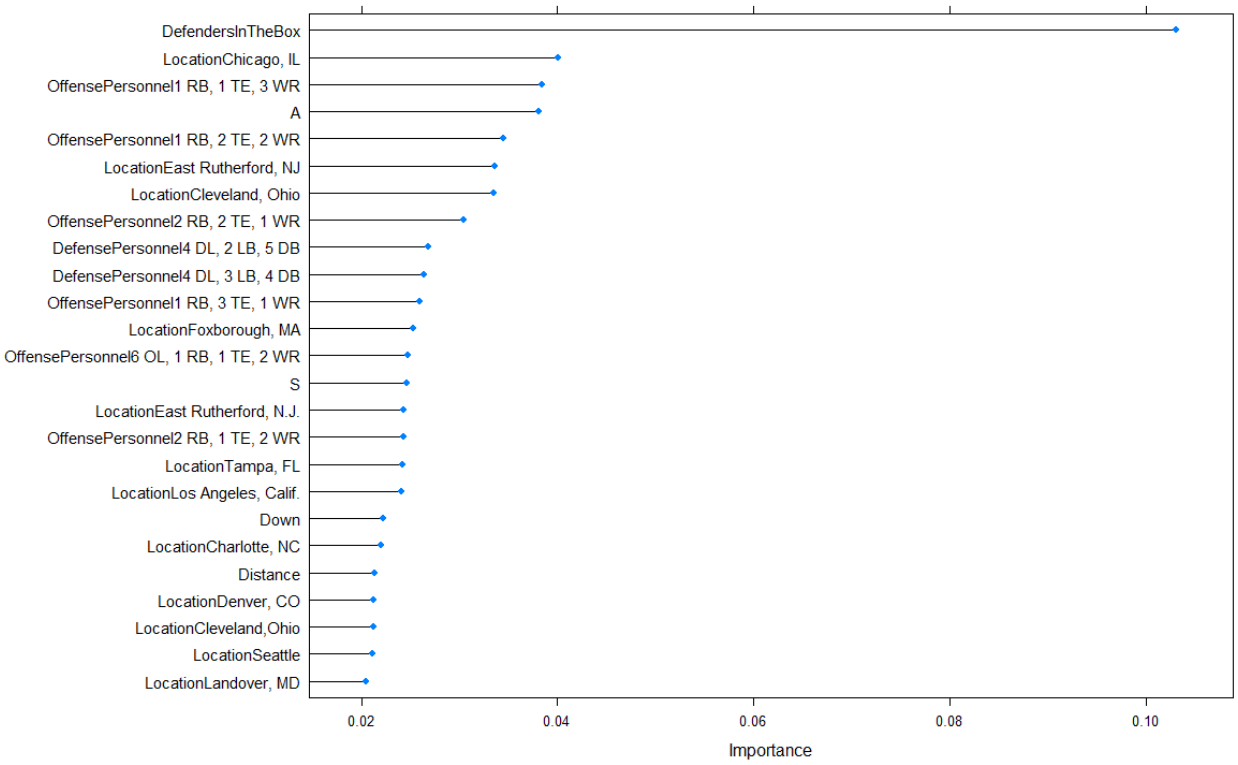

Fig: 4 – PLS Variable Importance Plot

Linear Models:

PCA:

Principal Component Analysis is used to draw important information from large datasets with multiple inter-correlated quantitative variables as a set of new variables called Principal Components. These new variables are the linear combination of the original variables. The number of resultant Principal Components obtained can be less than or equal to the number of original variables. The main aspect of PCA is to reduce the dimensionality of multivariate data and obtain principal components with minimal loss of information.

Here, for the NFL dataset, there are 20 variables after cleaning and on using the function prcomp() on the whole cleaned dataset, 20 Principal Components are obtained as shown below out of which about 8 components show more than 80% of variance on the data.

Below is the summary of PCA on the dataset and the scree plot.

```
> summary(NFL.PCA)
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9
Standard deviation     1.5773 1.4710 1.26122 1.18379 1.08399 1.08034 1.06598 1.03098 1.00162
Proportion of Variance 0.1244 0.1082 0.07953 0.07007 0.05875 0.05836 0.05682 0.05315 0.05016
Cumulative Proportion  0.1244 0.2326 0.31212 0.38219 0.44094 0.49930 0.55611 0.60926 0.65942
                          PC10    PC11    PC12    PC13    PC14    PC15    PC16   PC17    PC18
Standard deviation     0.99143 0.97282 0.91999 0.90525 0.89284 0.86051 0.77094 0.6648 0.58023
Proportion of Variance 0.04915 0.04732 0.04232 0.04097 0.03986 0.03702 0.02972 0.0221 0.01683
Cumulative Proportion  0.70857 0.75589 0.79821 0.83918 0.87904 0.91606 0.94578 0.9679 0.98471
                          PC19   PC20
Standard deviation     0.48970 0.2570
Proportion of Variance 0.01199 0.0033
Cumulative Proportion  0.99670 1.0000
```

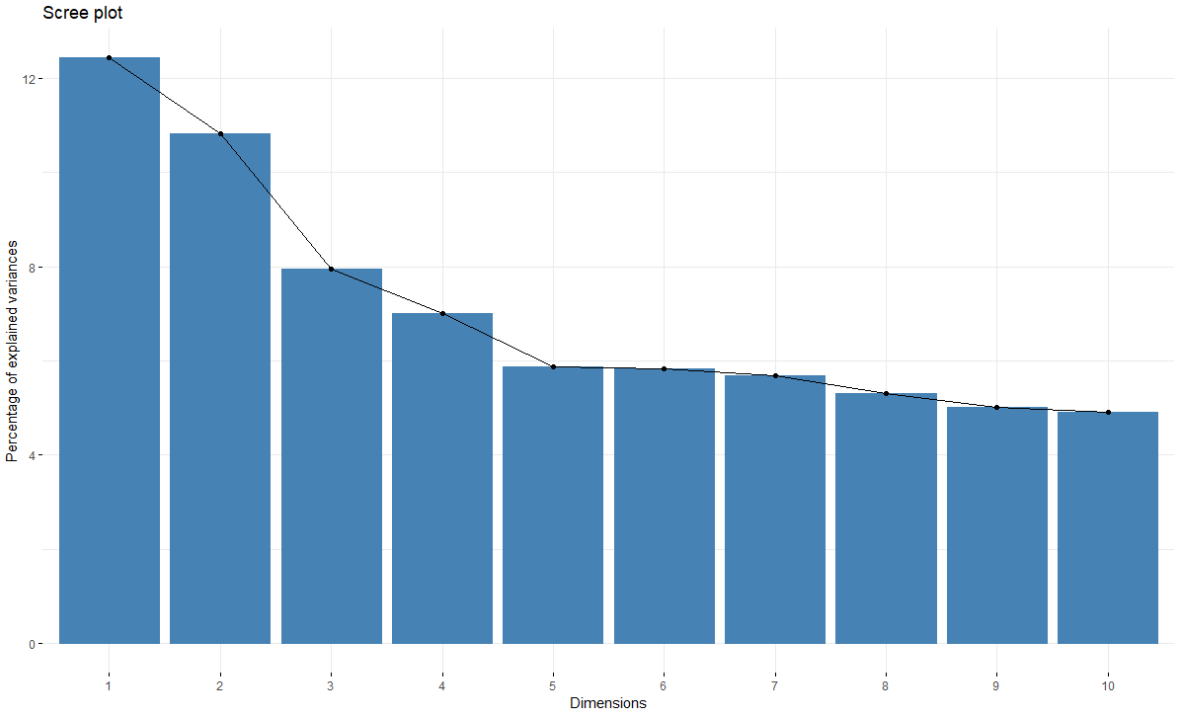

Fig: 5 – PCA Scree Plot

PCR:

Principal Component Regression is regression technique based on dimension reduction. The PCR first performs Principal Component Analysis on the data to obtain new variables which are Principal Components and then regression model is built using the new variables or the Principal Components.

The tuning parameter for PCR is ncomp which is the number of optimal principal components is chosen by cross-validation(cv).

Using the train() function and method as pcr, intially Principal Components are generated for the response and the predictor variables and regression model is built with the same. We have used Pre-processiong and 10-fold cross validation and tune length as 36. The tuning parameter ncomp obtained is 18 out of 19 predictors that means only two of the components are much correlated which are acceleration and speed and there is not much correlation between all other predictors. We can conclude that PCR is not much used for our dataset.

```
model <- train(Yards~.,data = nfl_train,method = "pcr",
               trControl = trainControl("cv",number = 10),
               preProc = c("center","scale"),tuneLength = 36)
```

```
> model$bestTune
   ncomp
18    18
> summary(model$finalModel)
Data:   X dimension: 250527 19
        Y dimension: 250527 1
Fit method: svdpc
Number of components considered: 18
TRAINING: % variance explained
           1 comps   2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps
X         13.06536  24.45303  32.7175  40.0839  46.2335   52.310   58.226   63.805    69.08
.outcome   0.03994   0.04276   0.7575   0.8873   0.8885    1.143    1.482    1.488     1.49
          10 comps  11 comps  12 comps  13 comps  14 comps  15 comps  16 comps  17 comps
X           74.258    78.714    83.029    87.246    91.150     94.28     96.61    98.388
.outcome     1.491     1.495     1.539     1.594     1.595      1.62      1.62     1.621
          18 comps
X           99.652
.outcome     1.623
```

The below plot shows the number of components vs RMSE after cross validation. We could observe that on using about less than 15 components the RMSE is higher and the model is not accurate. The optimal number of components for the model to be accurate is 18.

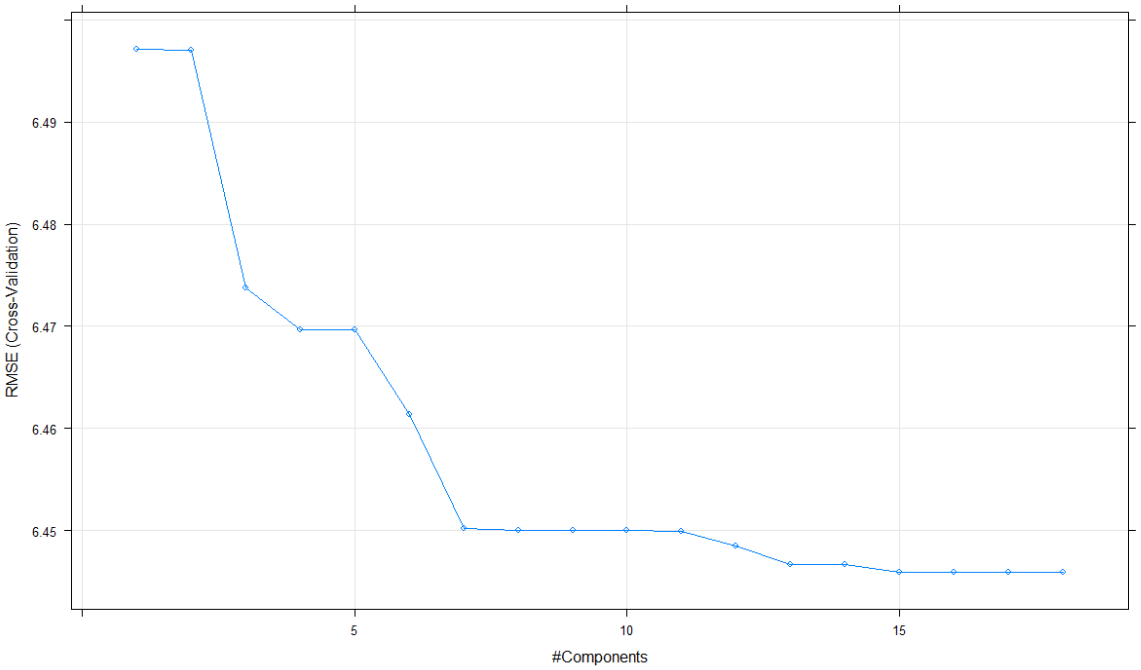

Fig: 6 – PCR RMSE Vs Components Plot

Penalized Regression models (Ridge, Lasso):

The Penalized regression models are the advanced forms of standard linear models and creates a linear regression model that is penalized with different penalty terms to reduce or shrink the coefficient values towards zero based on the model used.

The tuning parameter for both Ridge and Lasso is Lambda which is the amount of shrinkage. The amount of penalty for each model can be fine-tuned using the lambda. It can have a value from 0 to infinity and is different for Ridge and Lasso.

The main difference between ridge and lasso is ridge regression shrink the coefficients with minor contribution to zero whereas lasso set the coefficients to zero. That means Ridge regression includes all

the predictors independent of their contribution but in Lasso, the number of predictors in the final model may be less than actual number of predictors depending on their contribution to the response.

Glmnet() is used for the penalized regression models and the models can be differentiated based on the values for alpha. When alpha = 0, the regression model is considered as ridge and alpha =1 as Lasso.

Lasso (Least Absolute Shrinkage and Selection Operator):

Without Tuning parameter:

When the tuning parameter is not considered (lambda = NULL), the regression is built as shown below and the RMSE is 6.4351.

```
> #Lasso without tuning parameters
> model <- glmnet(x,y,alpha = 1,lambda = NULL)
> x.test <- model.matrix(Yards~.,nfl_test)[,-1]
> predictions <- model %>% predict(x.test) %>% as.vector()
> data.frame(
+   RMSE = RMSE(predictions,nfl_test$Yards)
+ )
      RMSE
1 6.435151
```

With tuning and preprocessing:

The regression is built using preprocessing and 10-fold cross validation with 5 repeats and a range of values for lambda (tuning parameter). The optimal model which has a tuning parameter at which the RMSE is low is considered. The value of lambda and RMSE for the optimal model is 0.003511 and 6.4283 respectively.

```
#Lasso using grid
lambda = 10^seq(-3,3,length=100)
set.seed(540)
lasso <- train(Yards~.,data = nfl_train,method = "glmnet",
               trControl = trainControl(method = "repeatedcv",number = 10,repeats = 5),
               preProc=c("center","scale"),
               tuneGrid = expand.grid(alpha = 1, lambda = lambda))
coef(lasso$finalModel,lasso$bestTune$lamdba)
predictions <- lasso %>% predict(nfl_test)
postResample(predictions, nfl_test$Yards)
```

```
> predictions <- lasso %>% predict(nfl_test)
> postResample(predictions, nfl_test$Yards)
      RMSE   Rsquared        MAE
6.42839597 0.01645286 3.81101321
> lasso$bestTune
   alpha      lambda
10     1 0.003511192
```

Ridge:

Without tuning parameters:

When the tuning parameter is not considered (lambda = NULL), the regression is built as shown below and the RMSE is 6.4483

```
> #ridge without tuning parameters
> model <- glmnet(x,y,alpha = 0,lambda = NULL)
> x.test <- model.matrix(Yards~.,nfl_test)[,-1]
> predictions <- model %>% predict(x.test) %>% as.vector()
> data.frame(
+   RMSE = RMSE(predictions,nfl_test$Yards)
+ )
      RMSE
1 6.448387
```

Ridge with tuning parameters:

The regression is built using preprocessing and 10-fold cross validation with 5 repeats and a range of values for lambda (tuning parameter). The optimal model which has a tuning parameter at which the RMSE is low is considered. The value of lambda and RMSE for the optimal model is 0.06579 and 6.4284 respectively.

```
#Ridge using grid
lambda = 10^seq(-3,3,length=100)
set.seed(540)
ridge <- train(Yards~.,data = nfl_train,method = "glmnet",
               trControl = trainControl(method = "repeatedcv",number = 10,repeats = 5),
               preProc=c("center","scale"),
               tuneGrid = expand.grid(alpha = 0, lambda = lambda))
coef(ridge$finalModel,ridge$bestTune$lamdba)
predictions <- ridge %>% predict(nfl_test)
postResample(predictions, nfl_test$Yards)
```

```
> predictions <- ridge %>% predict(nfl_test)
> postResample(predictions, nfl_test$Yards)
      RMSE   Rsquared        MAE
6.42845453 0.01643394 3.81115498
> ridge$bestTune
   alpha     lambda
31     0 0.06579332
```

Conclusion: We could observe from both models the difference in each model with and without considering the tuning parameter and preprocessing. The optimal model (model with low RMSE) is only built when tuning and reprocessing are considered.

On comparing both the regression models ridge and lasso, we could also conclude that Lasso has slight low RMSE compared to Ridge as Lasso considers only the important predictors but ridge considers all the predictors to build the regression model. Lasso is the best model for our data.

```
> models <- list(ridge = ridge,lasso = lasso)
> resamples(models)%>%summary(metric="RMSE")

Call:
summary.resamples(object = ., metric = "RMSE")

Models: ridge, lasso
Number of resamples: 50

RMSE
          Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
ridge 6.243677 6.388490 6.435920 6.445808 6.484771 6.745372    0
lasso 6.243583 6.388444 6.435878 6.445799 6.484775 6.745399    0
```

Random Forest:

In our problem we used random forest as a regression to predict our response variable Yards. As you can see from the plot (Fig:7) if the number of trees started to increase from 0 to 100 there is a sudden drop of an error. When the tree size is above 100, we can observe there is only small amount of error rate drop and is steady till 200th tree. Here we have used mtry =7 as tuning parameter.

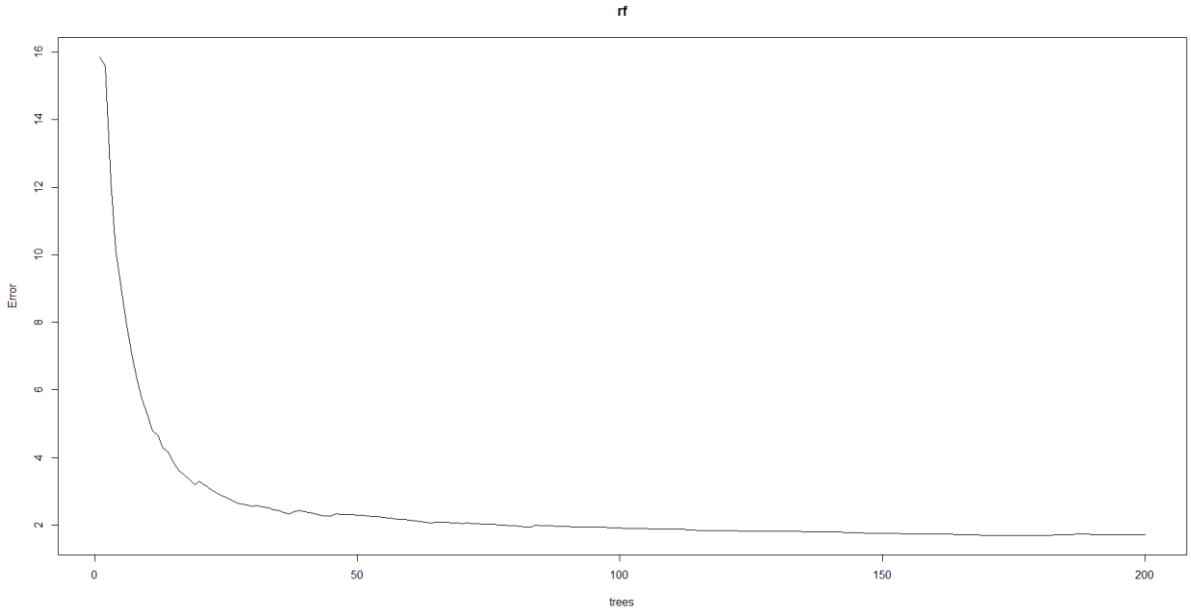

Fig: 7 – Random Forest Error vs Tree plot.

From the below graph (Fig: 8) you can see the ranking of each variable which is in descending order, i.e. variable which is on top of the graph is having most importance in predicting our response variable Yards. In our case Yard line, Humidity, Temperature, Homescorebeforeplay, Visitorscorebeforeplay, Windspeed plays a crucial role in predicting Yards. Playerweight, Playerheight, Speed and Acceleration had less importance from our model which doesn't show much impact in predicting the yards after handoff. We have a train rmse of 1.690553 and test rmse of 1.257535. From all the models so far we have done, Random Forest gives us best RMSE value and most important predictors in predicting our response variable. Unfortunately, we are not able to do produce results of random forest with Cross Validation by the report time. We performed Random Forest with out CV but able to achieve minimum rmse value than with the other models which performed with CV.

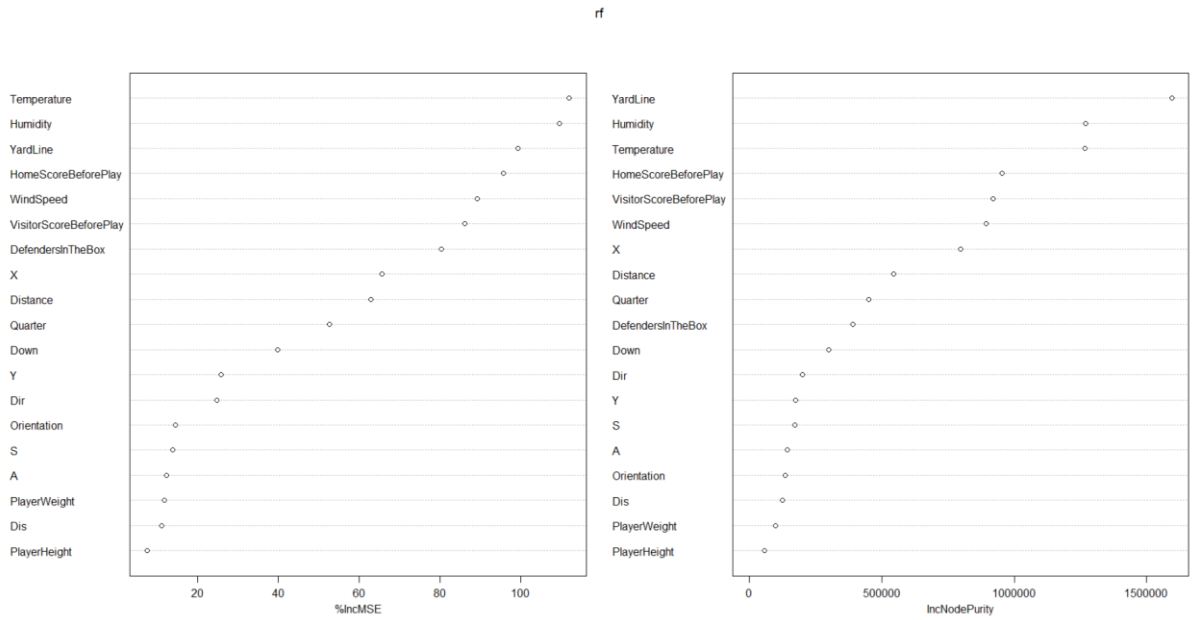

Fig: 8 – Random Forest Variable Importance Plot.

XGBOOST:

The next model that we are going to use is Extreme Gradient Boosting which is advanced model and has won many Kaggle competitions for its accuracy. From Randomforest we have high variance and low bias by using xgboost we can achieve high accuracy with more performance and avoid high variance. Here we have used xgboost with 10 – Fold Cross Validation and achieved the below results which are optimal.

```
> xgb.cv$bestTune
    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
108     150         3 0.4     0              0.8                1         1
> result
```

Parameters for Tree Booster

1. **nrounds[default=100]**
   - It controls the maximum number of iterations. For classification, it is similar to the number of trees to grow.
   - Should be tuned using CV
2. **eta[default=0.3] [range: (0,1)]**
   - It controls the learning rate, i.e., the rate at which our model learns patterns in data. After every round, it shrinks the feature weights to reach the best optimum.
3. **gamma[default=0] [range: (0,Inf)]**
   - It controls regularization (or prevents overfitting). The optimal value of gamma depends on the data set and other parameter values.
   - Higher the value, higher the regularization. Regularization means penalizing large coefficients which don't improve the model's performance. default = 0 means no regularization.

- *Tune trick:* Start with 0 and check CV error rate. If you see train error >>> test error, bring gamma into action. Higher the gamma, lower the difference in train and test CV. If you have no clue what value to use, use gamma=5 and see the performance. Remember that gamma brings improvement when you want to use shallow (low max_depth) trees.

4. **max_depth[default=6] [range: (0, Inf)]**
   - It controls the depth of the tree.
   - Larger the depth, more complex the model; higher chances of overfitting. There is no standard value for max_depth. Larger data sets require deep trees to learn the rules from data.
   - Should be tuned using CV
5. **min_child_weight[default=1] [range:(0, Inf)]**
   - In regression, it refers to the minimum number of instances required in a child node. In classification, if the leaf node has a minimum sum of instance weight (calculated by second order partial derivative) lower than min_child_weight, the tree splitting stops.
   - In simple words, it blocks the potential feature interactions to prevent overfitting. Should be tuned using CV.
6. **subsample[default=1] [range: (0,1)]**
   - It controls the number of samples (observations) supplied to a tree.
   - Typically, its values lie between (0.5-0.8)
7. **colsample_bytree[default=1] [range: (0,1)]**
   - It controls the number of features (variables) supplied to a tree
   - Typically, its values lie between (0.5,0.9)

After having best parameters from CV, we use them to train them on our model and able to get the below plot (Fig: 9). Below plot is the importance of variables listed in descending order which is much like Random forest. From this, we can conclude that Yardline, Temperature, Humidity, Homescorebeforeplay, Visitorscorebeforeplay are the most common predictors effecting the Yards. The test rmse is 4.54339 which best when compared to all other linear models which performed on Cross Validation.

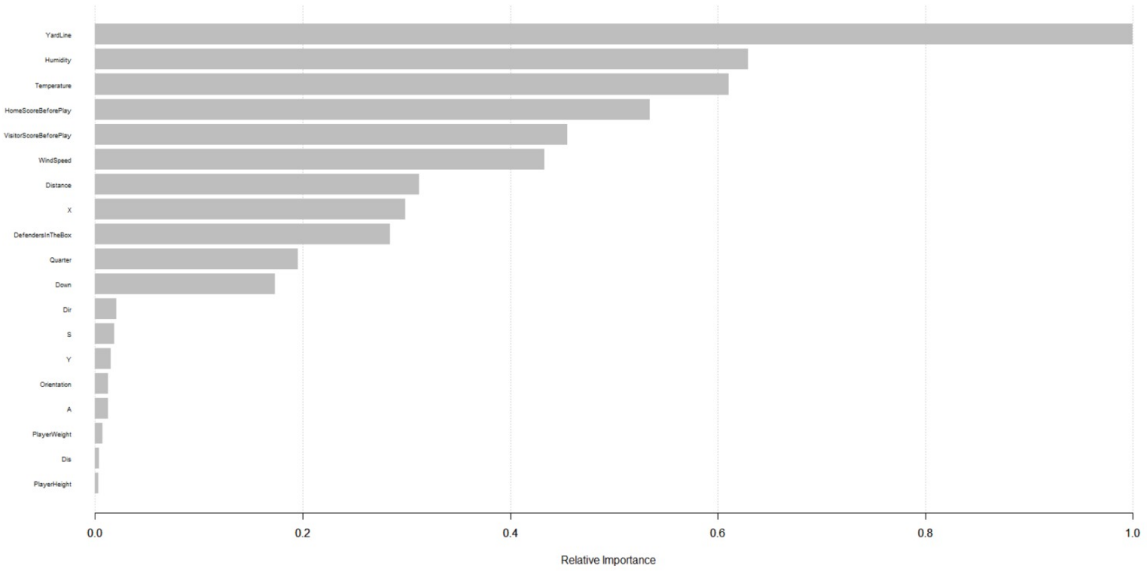

Fig: 9 – Variable Importance plot of XGBOOST

NFL Big Data Bowl
How many yards will an NFL player gain after receiving a handoff?

PREDICTIVE MODELS (ALL WITH CV'S)	**TEST RMSE VALUES**
LASSO (W/o Tuning)	6.458017
LASSO (with Tuning)	6.451709
RIDGE (W/o Tuning)	6.470857
RIDGE (with Tuning)	6.451737
PLS (with Tuning)	6.504466
PCR (with Tuning)	6.4517377
RANDOM FOREST (with Tuning) (Without CV)	1.257535
XGBOOST (with Tuning)	4.54339

Table: 1 – Predictive Models with Test RMSE Values

	XGBOOST	RANDOM FOREST
Yard Line	1	1
Temperature	2	2
Humidity	3	3
Humidity score before play	4	4
Visitor score before play	5	5
Windspeed	6	7
X	9	6
Defenders in the Box	10	11
Quarter	11	10
Down	12	12
Season2017	13	27
Dir	14	16
Orientation	15	23
S	16	20

Table: 2 – Ranking of Important variables with XGBOOST & Random Forest

Rankings from the above table gives enough information that in both the models top 5 has the same rankings. At which Yard line play happens makes an important role in predicting the yards. After that

NFL Big Data Bowl
How many yards will an NFL player gain after receiving a handoff?

Temperature, Humidity, Homescorebeforeplay, Visitorscorebeforeplay are the other factors that effects in gaining yards. In detail explanation was given during presentation. Because of the page constraint we are not able to include importance of each variables with the plots.

Conclusion:

In predicting our response variable Yards, Non-Linear models performed better than Linear models. We can say that by looking into the Test RMSE values of each model from the above table. Even though Random Forest has low RMSE, we expect that it might overfit our data than the xgboost.

Data Execution Steps: While running Random Forest model it takes around 12hrs of execution time because of 275,000 rows in train set. We used Python for data preprocessing and included them in submission. Train.csv is the rawdata file which was imported into Python and used for preprocessing. Final.csv file is the file after preprocessing and used for training and testing of our models in R.

Appendix

- NFL Data set Link from Kaggle. https://www.kaggle.com/c/nfl-big-data-bowl-2020/data
- More Info about NFL. https://www.nfl.com/