Analyzing Tackles using Oversampling and PCA against different ML Models in PySpark

## Overview

For this project, we intend to explore the extend of the PySpark library to answer our Research Questions below:

1. Print the Top 3 teams that are best at tackling.  
2. Print the Top 3 teams that missed the most tackles.  
3. Print the Top 3 players that are the best at tackling.  
4. Print the Top 3 players that missed the most tackles.

5. What are the Top 3 features to identify tackles?   
6. What are the Bottom 3 features to identify tackles?   
7. Which model is the best to predict a successful Tackle?  
8. Use precision, confusion matrix, TPR, TNR, F1-Score, AUC, ROC, and Accuracy of each model using PCA.

For our project, we will use previous season NFL SuperBowl data to analyze features contributing to a successful tackle. The original dataset contains 13 csv files containing information of the different games within the season, along with game highlights, players information, and plays tracking for each week of the first 9 weeks within the season. The original data can be found at: <https://www.kaggle.com/competitions/nfl-big-data-bowl-2024/overview>.

Looking at first glance, we could already predict that our challenge will be in terms of data cleaning and preprocessing due to the number of files that we must merge. Knowing that we are dealing with real data, there is a change that we will have to take in account class imbalance along with dimension reduction. Having experience the process using R and non PySpark-Python method of data cleaning and preprocessing, we wanted to explore the extend of the PySpark’s MLLib and ML libraries. This proved to be challenging because a lot of our favorite techniques were not available. Due to this, there are some techniques that we had to implement manually and others that are replaced to another method available to the libraries.

For this project, we decided to retain the structure of the dataset as a DataFrame mostly because we wanted to be able to save the output and treated datasets into csv files so that we can debug the results. We initially hoped that everything could be done in the local environment, however as we kept going, we realized that it is impossible. And we did not want to reduce the dimensionality of the dataset too much.

Although there are many shortcomings in our project, from the lack of optimization of the hyperparameters and model parameters to the lack in complexity in our project, our objective was to explore the PySpark abilities and libraries and become more comfortable with the syntax and program.

Before we continue with our methodology, I’d like to add a disclaimer that that I don’t watch nor follow the NFL Football games and league. I don’t understand American football well either (but if you ask me about the ‘real’ football, also known as soccer, that is a whole different ball game) so if that is any mistake in the application or analysis of the project, it is purely due to my ignorance in the topic. I had to rely on ChatGPT to teach me about the Football terms and gameplay.

## Methodology

Upon looking at our raw data, we found that the NFL Season 2022 Week 1-9 dataset was separated to 13 different files. Upon merging them together, we found that the data had many missing values. Additionally, the formatting of the data was all over the place, meaning that we need to reformat or drop complicated columns. Upon analyzing the number of rows and columns, we found that there were 12,188,407 rows and 67 columns in the newly merged dataset. To test our file merging progress, we created research questions 1 to 4. We hoped that by comparing it to the NFL statics website: <https://www.nfl.com/scores/2022/REG1>, we can ensure that our merging of the 13 files into our dataset was a success.

After merging the files, we also to identify and define the target variable as ‘Class’. In order words, data cleaning was necessary. We created the target variable ‘Class’ and assigned those with the value ‘1’ in the tackle column as ‘1’ and assign those with the value ‘1’ in the pff\_missedTackle column as ‘0’. After, we removed columns that were unrelated to analyzing the Tackles manually, resulting to 11,858,889 rows and 18 columns. Lastly, we had to deal with the missing values. In this step, we did not want to manipulate the data, so we decided to not impute the missing values with mean or median. When we tried imputing the missing values with missForest, we found that that the method is unavailable in the PySpark library. However, the method using kNN was available. Upon attempting to use it, we countered many errors and many attempts, we decided to comment the code out and remove all missing values – also due to our need to simplify our dataset. In hindsight, I would not have done it if I was comfortable with PySpark due to the high risk in losing too much information gain. However, after dealing with, it for 72 hours, I just didn’t want to deal with missing values imputation anymore. Also in this step, we also converted some values that were strings data types into floats and eliminated other strings data type because we did not want to deal with categorical features while getting used to PySpark. The whole process created the newly cleaned dataset called ‘cleaned\_df2’ where it has 166,324 rows and 18 columns.

After analyzing this dataset, we found that the Class was heavily imbalanced and thought that 18 columns is still too complex. Due to this we implemented oversampling and PCA for dimension reduction and set it to 10 components because we hoped to stay on our local environment to run our program. Optimizing the PCA components would have been too heavy, and we did not want to test our device too close to the Finals. In this process, although PCA is available in the PySpark library, SMOTE and oversampling was not. Before oversampling, we wanted to implement SMOTE, however, due to the complexity in implementing it manually, we decided to compensate with oversampling. Due to this, we predict that there is a high chance that we will have an abnormally high TPR or TNR as the result. But at this rate, it’s been a week into the Project and a few days away from the deadline, so we must move on.

Upon applying oversampling and PCA, our trials had only started. Despite the code not having any mistakes, errors keep coming up in the oversampling state. Occasionally succeeding and other times failing, due to this, we decided to split our versioning keeping one for the local environment (main.py) and creating another one of the Google Cluster environments (main\_cluster.py). Here, we found that the Google Cluster version was more stable, so we were able to continue, while splitting the main.py file into smaller chunks (Preliminary Research Question.py, Preprocessing\_cleaneddata.py, and Preprocessing\_TrainTestBalancedData.py), hoping to lessen the load in the local environment testing.

Initially, we flirted with the idea of using Boruta or other feature selection techniques, however by using the other techniques we will still have to take measure to prevent overfitting. Due to this, we compromised and choose to use the PCA feature selection method to reduce the dimensionality of the dataset. We could have use the Boruta method and choose a regularization technique, but after the horrors from Assignment 3 and 4, we did not want to touch learning rates and regularization parameters for the temporary duration. And so, we proceeded with PCA feature selection without PCA optimization. From this, we knew already that we had to forego high accuracy results and attempt to push through towards our modelling stage. From this stage, we created four new datasets and csv folders, named train\_df, test\_df, train\_df\_pca, and test\_df\_pca.

After discounting a lot of the methods and techniques that we wanted to try, we decided to compensate by using multiple models from the many different types of models: regression, classification, unsupervised learning, clusters, and neural network. But upon execution, we backtracked and decided to keep our tail between our legs and stay with supervised learning models only. At this point, we have completely abandoned our initial goal from our project proposal in using gradient boosting, regularization, and k-means modelling. But with the effort that we’ve put in and the ticking seconds towards the project deadline, it’s too late to change paths.

So, at this point we are attempting to model with: logistic regression, random forest, decision tree, naïve Bayesian, and SVM (Linear SVC), however in one last attempt to save our project, we attempted to implement cross-validation while modelling. But it failed.

Upon running our program on Google Cluster, we came upon these errors:

Caused by: java.lang.IllegalArgumentException: requirement failed: Naive Bayes requires nonnegative feature values but found [206.87050929764,174.85110972377376,-18.229419076694327,9.795812148909326,-6.834812201978687,2.8509015046237387,3.299649516754692,10.346281952714746,-4.8736663521734584,2.5145104815408694].

at scala.Predef$.require(Predef.scala:281)

at org.apache.spark.ml.classification.NaiveBayes$.requireNonnegativeValues(NaiveBayes.scala:359)

at org.apache.spark.ml.classification.NaiveBayes.$anonfun$trainDiscreteImpl$1(NaiveBayes.scala:178)

at org.apache.spark.sql.catalyst.expressions.ScalaUDF.$anonfun$f$2(ScalaUDF.scala:210)

at org.apache.spark.sql.catalyst.expressions.ScalaUDF.eval(ScalaUDF.scala:1192)

... 29 more

Upon realizing that we need to customize our dataset for the Naïve Bayesian model, we decided to remove it from the study leaving us with the models: logistic regression, random forest, decision tree, and Linear SVC (SVM). Because four is still more than the proposed 3 models, we will still count this as our mini victory over PySpark as we proceeded to the performance metric section of the project.

Here, we initially attempted to use the PySpark libraries to calculate our target metrics as requested in our research question. However, due to the approaching deadline and the frustration of failed attempts, we decided to proceed with manual calculation of metrics since we couldn’t use the Binary Classification Evaluator. Additionally, our code in manually calculating the AUC and ROC curve kept failing, so we decided not to include them in the metrics.

Together with the metrics, we also included calculating the correlations of the features in relations with the Class target Variable to answer research question 5 and 6 to find the top 3 best and worst performing features to identify tackles. Using the newly produced output metrics, we can analyze the results among the models to answer research question 7 – which model is the best in predicting successful tackles.

## Results

Bringing back our Research Questions into the picture:

Research Questions:

1. Print the Top 3 teams that are best at tackling.  
2. Print the Top 3 teams that missed the most tackles.  
3. Print the Top 3 players that are the best at tackling.  
4. Print the Top 3 players that missed the most tackles.

5. What are the Top 3 features to identify tackles?   
6. What are the Bottom 3 features to identify tackles?   
7. Which model is the best to predict a successful Tackle?  
8. Use precision, confusion matrix, TPR, TNR, F1-Score, and Accuracy of each model using PCA.

We will start by providing the results for the first four Research questions, where we found that:

Top 3 teams best at tackling:

+----+-----+

|club|count|

+----+-----+

| SEA|15196|

| ARI|14485|

| CAR|14360|

+----+-----+

Seattle Seahawks at first place with a total of 15,196 successful tackles within the first 9 weeks of the 2022 NFL Season.

 Arizona Cardinals at second place with a total of 14,485 successful tackles within the first 9 weeks of the 2022 NFL Season.

Carolina Panthers at third place with a total of 14,360 successful tackles within the first 9 weeks of the 2022 NFL Season.

Top 3 teams that missed the most tackles:

+----+-----+

|club|count|

+----+-----+

| HOU| 5857|

| JAX| 5660|

| CAR| 4872|

+----+-----+

Houston Texans at first place with the total of 5,857 missed tackles for the first 9 weeks of the 2022 NFL Season.

Jacksonville Jaguars at second place with the total of 5,660 missed tackles for the first 9 weeks of the 2022 NFL Season.

Carolina Panthers at third place with a total of 4,872 missed tackles for the first 9 weeks of the 2022 NFL Season; telling us sometimes all you need to do is try and you might win third place in both the top successful tackles category and missed tackles category, earning yourself a valuation of USD 3.6B.

When comparing the results for the clubs with the NFL official website, we realized that they don’t make statistics just for the first 9 weeks of the season. They are also not as cruel as us in highlighting the missed tackles category. Despite this observation, we are continued further to analyze the players.

Top 3 players best at tackling:

+-----+----------------+-----+

|nflId| displayName|count|

+-----+----------------+-----+

|52435| Jordyn Brooks| 2603|

|46269|Foyesade Oluokun| 2502|

|41243| C.J. Mosley| 2323|

+-----+----------------+-----+

 Jordyn Brooks from the Seattle Seahawks takes first place with 2,603 successful tackles for the first 9 weeks of the 2022 NFL Season, comprising more than 17 percent of the Seattle Seahawks total successful tackles count. When compared to the NFL statistics, it was surprising that he was not mentioned in any of the defensive player award from week 1 to 9. This tells us either that tackles are not highly rated as a contribution to the defensive player award, or that our dataset was altered.

 Foyesade Oluokun from the Jacksonville Jaguars takes second place with 2,503 successful tackles for the first 9 weeks of the 2022 NFL Season, showing a promising career start after his transfer from the Atlanta Falcons to the Jacksonville Jaguars.

 C.J. Mosley from New York Jets wins third place with 2,323 successful tackles for the first 9 weeks of the 2022 NFL Season. At this point, none of the three players were mentioned in the first 9 weeks defensive player award.

Top 3 players that missed the most tackles:

+-----+----------------+-----+

|nflId| displayName|count|

+-----+----------------+-----+

|54502| Jalen Pitre| 1280|

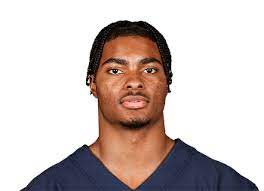
|44926|Rayshawn Jenkins| 862|

|54513| Jaquan Brisker| 723|

+-----+----------------+-----+

 Jalen Pitre, this The Weekend lookalike wins first place with 1,280 missed tackles for the first 9 weeks of the 2022 NFL Season. Unlike his singer doppelganger, this defensive back is off to an unpromising start for Houston Texans on the 2022 NFL Season according to our dataset. Pitre taking responsibility a little below 22 percent of the Houston Texans’ missed tackles.

 Rayshawn Jenkins, continuing our trend in matching lookalike, this Bob Marley Jacksonville Jaguar safety takes second place with 862 missed tackles for the first 9 weeks of the 2022 NFL Season. When comparing his statistics with the Season highlights, we saw that Jenkins did a comeback winning the title as Defensive Player of the Week on Week 15.

 Jaquan Brisker, this Chris Rock lookalike, wins third place with 723 missed tackles for the first 9 weeks of the 2022 NFL Season. This Chicago Bears safety did not appear in the Players of the Week award.

Upon cleaning the dataset, as mentioned in the methodology, we were left with 166,324 rows and 18 columns consisting of the features:

Final DataFrame Schema:

root

|-- playDirection: integer (nullable = false)

|-- dis: double (nullable = true)

|-- o: float (nullable = true)

|-- dir: float (nullable = true)

|-- quarter: integer (nullable = true)

|-- down: integer (nullable = true)

|-- yardsToGo: integer (nullable = true)

|-- yardlineNumber: integer (nullable = true)

|-- preSnapHomeScore: integer (nullable = true)

|-- preSnapVisitorScore: integer (nullable = true)

|-- passLength: float (nullable = true)

|-- absoluteYardlineNumber: integer (nullable = true)

|-- defendersInTheBox: float (nullable = true)

|-- preSnapHomeTeamWinProbability: double (nullable = true)

|-- preSnapVisitorTeamWinProbability: double (nullable = true)

|-- assist: integer (nullable = true)

|-- forcedFumble: integer (nullable = true)

|-- Class: integer (nullable = true)

After modeling, we saved our results in a variable called ‘Predictions’ where we have a dataframe with a schema of:

root

|-- pca\_features: vector (nullable = true)

|-- Class: integer (nullable = true)

|-- rawPrediction: vector (nullable = true)

|-- probability: vector (nullable = true)

|-- prediction: integer (nullable = true)

Following our clubs and players’ appreciations and roasts, we continue onwards to Research Question 5 to 8

5. What are the Top 3 features to identify tackles?   
6. What are the Bottom 3 features to identify tackles?

7. Which model is the best to predict a successful Tackle?  
8. Use precision, confusion matrix, TPR, TNR, F1-Score, and Accuracy of each model using PCA.

Here, we received the results:

Evaluation Metrics for Logistic Regression with PCA with PCA

Precision: 0.5771468087702885

Recall: 0.5409073579870377

Accuracy: 0.5719518476830036

F1 Score: 0.5584397693503631

True Positive Rate (TPR): 0.5409073579870377

True Negative Rate (TNR): 0.603047313552526

Top 3 features for Logistic Regression with PCA to identify tackles:

preSnapVisitorScore : 0.06087124170796891

down : 0.040100724543633046

yardsToGo : 0.0010036590158827245

Bottom 3 features for Logistic Regression with PCA to identify tackles:

quarter : -0.008910427589369623

yardlineNumber : -0.01707633774375571

preSnapHomeScore : -0.16825554432033443

Evaluation Metrics for Random Forest with PCA with PCA

Precision: 0.6656332839106798

Recall: 0.5648112847884103

Accuracy: 0.6394937580056207

F1 Score: 0.6110916328087942

True Positive Rate (TPR): 0.5648112847884103

True Negative Rate (TNR): 0.7146144111669287

Top 3 features for Random Forest with PCA to identify tackles:

down : 0.3445750112139378

preSnapHomeScore : 0.1517539959037057

yardlineNumber : 0.10885487531262934

Bottom 3 features for Random Forest with PCA to identify tackles:

yardsToGo : 0.044529749937913136

playDirection : 0.035674826754119114

dis : 0.0035580890682999147

Evaluation Metrics for Decision Tree with PCA with PCA

Precision: 0.6890147938306579

Recall: 0.4172703011818528

Accuracy: 0.6139235673653266

F1 Score: 0.5197673038109937

True Positive Rate (TPR): 0.4172703011818528

True Negative Rate (TNR): 0.8111332007952287

Top 3 features for Decision Tree with PCA to identify tackles:

down : 0.27758872782074495

preSnapHomeScore : 0.18157560019193836

o : 0.1618287969533754

Bottom 3 features for Decision Tree with PCA to identify tackles:

playDirection : 0.0

dis : 0.0

preSnapVisitorScore : 0.0

Evaluation Metrics for Linear SVC with PCA with PCA

Precision: 0.5908700219654284

Recall: 0.47174990468928707

Accuracy: 0.5709474973212919

F1 Score: 0.5246332570168745

True Positive Rate (TPR): 0.47174990468928707

True Negative Rate (TNR): 0.6708919105784743

Top 3 features for Linear SVC with PCA to identify tackles:

preSnapVisitorScore : 0.137157356584864

down : 0.07118510422685244

playDirection : 0.0005403406919617207

Bottom 3 features for Linear SVC with PCA to identify tackles:

quarter : -0.014288984415270394

yardlineNumber : -0.022925480646499283

preSnapHomeScore : -0.27325044429299383

A screenshot of a computer

Description automatically generated

Now, to clean up the result for the presentation, we converted our result outputs into a table format starting with our performance metrics to answer research question 5 and 6.

|  |  |  |  |
| --- | --- | --- | --- |
| Models | Best Performing Features | Worst Performing Features | Notes |
| Logistic Regression | 1. preSnapVisitorScore 2. Down 3. yardsToGo | 1. quarter 2. yardlineNumber 3. preSnapHomeScore | Here, the worst performing features have negative correlation. Meaning that the most bottom feature, preSnapHomeScore is the best performing feature. Just that it has a negative relationship with the Class. |
| Random Forest | 1. Down 2. preSnapHomeScore 3. yardlineNumber | 1. yardsToGo 2. playDirection 3. distance (dis) |  |
| Decision Tree | 1. Down 2. preSnapHomeScore 3. orientation (o) | 1. playDirection 2. distance (dis) 3. preSnapVisitorScore |  |
| Linear SVC | 1. preSnapVisitorScore 2. down 3. playDirection | 1. quarter 2. yardlineNumber 3. preSnapHomeScore | Here, the worst performing features have negative correlation. Meaning that the most bottom feature, preSnapHomeScore is the best performing feature. Just that it has a negative relationship with the Class. |
| Final Answer | 1. Down 2. preSnapHomeScore 3. preSnapVisitorScore | 1. quarter 2. playDirection 3. yardlineNumber 4. distance(dis) |  |

We missed the fact that there are negative correlations between the features. Therefore, when deciding the final determining best and worst features in predicting successful tackles, we must assume that for logistic regression and linear SVC models, the worst feature, preSnapHomeScore is the best. From looking at the pattern/rank between the models, we can conclude that the best features to predict successful tackles are downs, preSnapHomeScore, and preSnapVisitorScore. This makes sense because competition and time crunch cause the pressure to perform more tackles to defend the other opponent from scoring. Especially in the last downs, you are more pressured to make good tackles to change the side of offense, and based on the score, if you’re losing you want to make sure the opponents don’t get more points while if you’re winning you want to prevent your opponent from catching up. For the worst features to predict successful tackles, we have the features: quarter, playDirection, yardlineNumber, and distance (dis). This makes sense because the game is very long, depending momentum more based on downs rather than quarters. Winning earlier on in the game does not necessarily mean that you will have a big advantage. This goes for playDirection as well, since NFL players are professional, they should perform well whether then are playing towards the right or the left. Like the reasoning of the quarter, the yardlineNumber and distance is not as important since the pressure comes from the downs rather than the distance of the ball and offense player to the defender and the yardlineNumber the offensive player is located. The only time yardlineNumber would cause pressure to tackle is when it is close to goal post, explaining why it is still more important as a feature compared to the other features mentioned in the worst performing feature list.

But after finishing the report paper, we decided to update this error as a final hail mary to get a final win against PySpark, so we updated the code to get this result for research question 5 and 6, resulting to the table and results below:

Top 3 features for Logistic Regression with PCA to identify tackles:

preSnapHomeScore : 0.16615916157424823

preSnapVisitorScore : 0.06409464198317431

down : 0.0401605247924456

Bottom 3 features for Logistic Regression with PCA to identify tackles:

yardsToGo : 0.0006936973396539032

playDirection : 0.00021455613120887717

dis : 0.0001585845594853777

Top 3 features for Random Forest with PCA to identify tackles:

down : 0.32823610303892775

preSnapHomeScore : 0.17149082401219182

o : 0.11516751123739088

Bottom 3 features for Random Forest with PCA to identify tackles:

preSnapVisitorScore : 0.04933282269491672

playDirection : 0.02982608053557904

dis : 0.0063791563857208845

Top 3 features for Decision Tree with PCA to identify tackles:

down : 0.28608019713818844

o : 0.14317124836888628

yardlineNumber : 0.12481276113734988

Bottom 3 features for Decision Tree with PCA to identify tackles:

playDirection : 0.010013966562221768

dis : 0.00380486388277418

preSnapVisitorScore : 0.0

Top 3 features for Linear SVC with PCA to identify tackles:

preSnapHomeScore : 0.2834551530722658

preSnapVisitorScore : 0.11731813471121813

down : 0.0734825550429532

Bottom 3 features for Linear SVC with PCA to identify tackles:

yardsToGo : 0.003887376303905881

playDirection : 0.000507380640696493

dis : 0.00019458710417393624

|  |  |  |
| --- | --- | --- |
| Models | Best Performing Features | Worst Performing Features |
| Logistic Regression | 1. preSnapHomeScore 2. preSnapVisitorScore 3. Down | 1. yardsToGo 2. playDirection 3. distance (dis) |
| Random Forest | 1. Down 2. preSnapHomeScore 3. orientation (o) | 1. preSnapVisitorScore 2. playDirection 3. distance (dis) |
| Decision Tree | 1. Down 2. orientation (o) 3. yardlineNumber | 1. playDirection 2. distance (dis) 3. preSnapVisitorScore |
| Linear SVC | 1. preSnapHomeScore 2. preSnapVisitorScore 3. Down | 1. yardsToGo 2. playDirection 3. distance (dis) |
| Final Answer | 1. Down 2. preSnapHomeScore 3. orientation (o) | 1. playDirection 2. yardToGo 3. distance(dis) |

Comparing the table with the previous table, we can see that there are some features that changed. Our analysis for preSnapHomeScore can be visualized as seen in the newly updated table. Despite the changes our logic for the features importance is still somewhat true.

Next, to answer the research question 7 and 8, we made a table compromising of the performance metrics of the different models: Logistic Regression, Random Forest, Decision Tree, and Linear SVC (SVM).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Logistic Regression | Random Forest | Decision Tree | Linear SVC (SVM) |
| Precision | 57.715% | 66.563% | 68.901% | 59.087% |
| Accuracy | 57.195% | 63.949% | 61.392% | 57.095% |
| F1-Score | 55.844% | 61.109% | 51.977% | 52.463% |
| Recall/TPR | 54.091% | 56.481% | 41.727% | 47.175% |
| TNR | 60.305% | 71.461% | 81.113% | 67.089% |
| Rank of Performance | 3 | 1 Best | 2 | 4 |

From the table above, we can see that the decision tree model had the best performance in terms of precision with logistic regression being the worst. This makes sense because we did not use gradient boosting measures or regularization for the logistic regression. Additionally, the method of decision tree splitting through information gain gives it an advantage over the logistic regression in terms of identifying TP. In terms of Recall (or TPR), we can see that all models gave low percentage, we had predicted this earlier in the methodology section that we suspect was caused by using oversampling to solve our Class imbalance problem. In this case, there were less Class 0 (missed tackle data) compared to Class 1 (successful tackle data), hence why we had to replicate more of the Class 0. By replicating more of Class 0, this causes the models to predict Class 0 easier due to less variance within the data causing the TNR to be higher in all models. Despite this bias of which we suspect was caused by the oversampling method, we can see that the TPR is highest for the Random Forest model and lowest for the Decision Tree model. This is weird since both models are similar with the exception that random forest is an ensembled therefore harder to explain with the features compared to the decision tree. And for the TNR, the highest was the decision tree model and the lowest was the logistic regression model. In terms of accuracy and f1-score (which is another method to calculate accuracy), random forest model wins the best score. This makes sense because the model foregoes explainability to attain better accuracy; while the other three methods might not provide the best accuracy, instead provides better explainability of the features using information gain or weights. In terms of the lower accuracy, the linear SVC method performs the worse and very close to it is the logistic regression model. This pattern might signal to us that the relationship between the features with the Class target variable might not be linear, therefore we should test the dataset again with models such as: QDA and higher order or gaussian SVM models. In terms of f1-score, the lowest percentage goes to decision tree, this is highly contributed by the low recall score which we believe was caused by the high number of false negative. It will not be caused by a low true positive because the decision tree also has the highest precision score.

Lastly, we concluded that random forest has the best rank in terms of predictive ability to classify successful tackles because of its high rank in accuracy and f1-score, additionally its precision score is also quite high. Comparing the reasonings of the four models, it is obvious that random forest will produce the highest accuracy as an ensemble model. Decision tree takes second place in accuracy due to its high precision in predicting true positives and the logistic regression takes third place due to the overall higher accuracy and f1-scoring compared to the last place, Linear SVC. This last place position for the Linear SVC might be caused due to the underfitting of the model to the dataset since the dataset might need models with higher polynomial degrees.

However, if we change our research question to which model can help us explain our features better to understand contributions to successful tackles better, then we would say either the decision tree and logistic regression models due to its ability to detect correlations for feature importance well yet still having a somewhat decent accuracy.

## Discussion

As mentioned, multiple times through this report, our expectation from our project proposal did not materialize upon execution. However, if I had more time and more experience with PySpark, here is what I would have done instead.

Starting with merging the file, I would have converted some of the columns differently. Instead of ‘birthDate,’ I would have converted the column into age, and attempt longer to convert the height and weight into BMI as a new column. I would have also turned the values into ordinal values after the removal of outliers and remove the categorical features. After I would have removed columns with more than 60% missing values to reduce the risk of hallucinations and analyze the distribution of the columns individually after manually cleaning the data based on logic. If the distribution looks normal, then I would use mean or median to impute the missing values; else, I will have done the kNN imputing manually using the Euclidean distance.

After missing values imputation, I would have dropped columns and rows that does not contribute to team tackles based on logic along with columns consisting of zero variance through manual calculation and collinearity based on the high correlation in pairplot analysis. I would have also implemented Boruta as the first layer of feature selection with the tackle and pff\_missedtackle variables still there for pre-processing. And if the target variables got removed, I’ll just backtrack and continue to analyze the Class distribution.

After converting the tackle and pff\_missedtackle columns into the target variable, Class, I will analyze the counts of the binary Class outcomes, dropped the rows that does not explain tackles or missed tackles, and check if the Class outcomes were somewhat balance. If it is still imbalanced, I would try the SMOTE technique manually; if it doesn’t work, I will try with oversampling again and undersampling. I predict that oversampling will provide better results than undersampling.

After balancing the treated dataset, I will train-test split the dataset allocating 80% to the training dataset and 20% to the testing dataset. Here, I will implement our chosen models along with others representing the unsupervised learning and neural network models in four ways with another attempt in using cross validation. First without any additional techniques, second with PCA feature selection, third with gradient boosting (possibly manually if there are no supporting libraries in PySpark), and lastly with learning rate and regularization for relevant models. Acknowledging the errors that I found in this project, I will also alter my dataset and its features based on the modeling. One example would be to conduct a Chi-square Test before conducting the Naïve Bayesian model to make sure that all features are independent to each other along with removing features that consist of negative values or convert them into ordinal non-negative values.

We will use the same metrics in this project to analyze the results of the project but this time, try again using the PySpark libraries. And this time try to include the AUC and ROC curve once again. And for the correlation section to find the best and worst performing features, we must add absolute values on the weights so that we can judge the features strength correctly despite the relationship type (inverse or direct relationship) between the features and the Class target variable.

Although, this project is subpar in terms of industry-standard of conducting ML modeling, through the trial and errors along with the tedious data cleaning and preprocessing, we achieved our goal in understanding PySpark better and understand the limitations and strength that it has in working with Big Data. There is no doubt that the PySpark programming is crucial in dealing with Big Data and cloud computing. However, it would be interesting to see the difference it has to other similar languages such as SparkR, Hadoop, and many more within the Spark grouping.

## Conclusion

Despite having slightly more than 2 weeks to work on this project, this bittersweet journey in conquering PySpark has finally come to an end. We found out that despite its pros in dealing with Big Data, the cons in the PySpark library of being incomplete compared to libraries such as Scikit-learn and more signals to us that there are still many opportunities in building libraries within the Spark environment. We can see PySpark being very useful soon as the AI and ML industry matures and companies/organizations that to fully adopt the use of data as their operation standard.

Through the ups of finally getting the code to work and downs of compromising on the methods and techniques to use, we thoroughly enjoyed the adventure in tackling PySpark. Although it won’t be the last time that we will use the programming language, we will definitely take a decently long hiatus after the finals to recuperate from the morally low and trauma that language had gave us this first quarter of the year.