Kaggle Competition Report

NBA Career Prediction

Written by Group 3

Amy Yang ([redacted]), Chanthru Vimalasri, Yatindra Vegunta ([redacted])

Contents

Introduction

The NBA (National Basketball Association) is a professional basketball league in North America that plays a significant role in driving the growth of the game around the world. In the NBA, a rookie is any player who has never played a game in the NBA until that year. Being able to accurately predict the career success of a player from their current statistics and performance in their rookie season is vital. This information allows management to make better draft, trade and personnel decisions to give their team the best chance of winning a championship. The success of such teams and players will in turn strengthen the prestige and influence of the NBA not only in sport but art, culture and economics as well.

Following the CRISP-DM methodology, this project built several classification models with the historical statistics of rookie players and determined the best two classifiers using performance metrics including ROC-AUC score and Confusion matrix. In the sections below, each phase of CRISP-DM will be discussed in detail followed by our reflection on this project and the description of member contributions. All our files in this project were saved under the Github repository https://github.com/amy-panda/NBA_Career_Prediction.

Business understanding

To have a better understanding of the business needs, the objectives and requirements of the project are described below.

Business Objectives

To predict if a rookie player will last at least 5 years in the NBA league based on their historical statistics.

Success Criteria

The performance of a model at distinguishing if a rookie player will last at least 5 years or less than 5 years in the NBA league.

Risk and Benefit

A high performing model with accurate results will help the NBA league direct their resources towards players with the greatest talent and potential, encouraging viewership, sponsorship deals, merchandise sales and contributing to higher business profit. On the other hand, inaccurate results predicted by a model could have an adverse impact on players' career paths and the success of teams, devalue the brand name of the NBA league and cause financial losses in business.

Data understanding

The data used in this project is the historical statistics for rookie players including number of games they played, minutes they played in games, points per game and other metrics of their performance during a game. The data dictionary is included in Appendix A and data summary at Appendix B. The data, made up of training and test sets, was downloaded from Kaggle and saved into two files - train.csv and test.csv.

EDA. Exploratory data analysis on the training set shows:

- No null values
- No duplicate records
- All columns have numeric values
- Negative values in columns of ‘GP’, ‘3P’, ‘3PA’,’3P%’ , ‘FT%’ and ‘BLK’
- Unrealistically high values for FT% which are over 100%
- Imbalanced data issue for two classes (Records for target=1 and target=0 is around 5:1 as displayed in Figure 1)

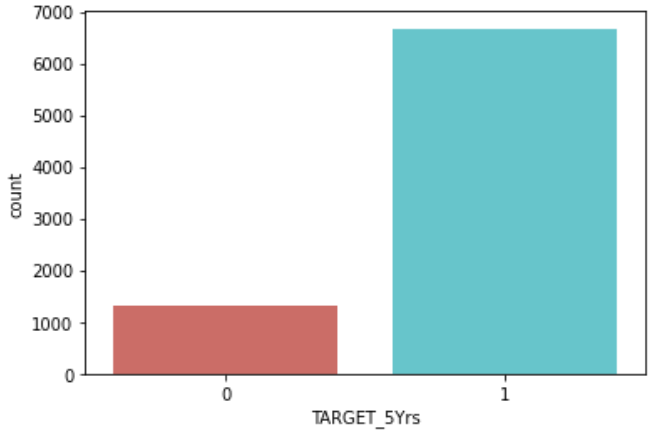


Figure 1 Imbalance class for target variable

Data preparation

The steps below were taken for data preparation:

Removing unreasonable values

Removing the unreasonable values minimises the noise within the data and ensures a higher quality data is used to build a model. As discussed in the Data understanding section, there are negative and unrealistically high values within the data. To treat these unreasonable values, several options were explored in our experiments including removing the unreasonable values, applying imputation methods (such as single imputation by using mean/median, multiple imputation and Nearest neighbour imputation) and keeping these data points. Based on the exploration, removing unreasonable values/outliers contributed to better model performance compared to keeping this data (doing nothing) or using imputation approaches. Thus this method was chosen to perform the data preparation.

Feature engineering

Based on the EDA, five columns were removed from the original dataset including ‘Id’, ‘3P Made’, ‘3PA’, ‘3P%’ and ‘BLK’. By removing these columns, the evaluation metrics (accuracy, ROC AUC and F1 score) were improved and the computing time was shortened in comparison with other options such as keeping all these features or applying the imputation methods. The rationale for removing these features is described in detail under Appendix C. An absolute function was run as an alternative to convert the negative values into positive. This allowed us to retain these features and whilst results did improve slightly for some models, it ultimately reduced the AUROC of our best performing model and was thus discarded.

Three new features were extracted from existing variables of raw data including total points players gained (TOTAL_PTS), total minutes they played (TOTAL_MIN) and the ratio of Field Goals Percent against Free Throw Percent (FG/FT). The evaluation metrics were slightly improved for most classifiers in comparison with the metric values when new features were excluded.

Table 1 New Features

Features	Metric	Description
TOTAL_PTS	TOTAL_PTS = GP * PTS	Total points earned by a rookie player
TOTAL_MIN	TOTAL_MIN = GP * MIN	Total minutes played by a rookie player
FG/FT	FG/FT = FG% / FT%	The ratio of Field Goals % against Free Throw %

Scaling the data

Scaling is also an important step of data preparation which brings data points that are far from each other closer to increase the algorithm's effectiveness and speed up the Machine Learning processing. For example, GP and MIN have much larger scales than the other variables. By scaling the data, the computation time was shorter for most classifiers as shown in Appendix D.

Splitting training and validation sets

The data were split into 80% training and 20% validation sets before modelling. This allows us to evaluate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model and helps to prevent our model from overfitting. We have also tried splitting the data into 3 parts, 80% training, 10% validation and 10% testing. This split however has not resulted with a higher ROC, hence was not chosen as the main model.

Modelling

Treating imbalanced data

In a dataset with class imbalance, balancing the class before modelling is a key step that defines the modelling process. Several methods were explored to treat the imbalanced data in the train set including oversampling, undersampling, STOME and hyperparameter setting (such as class_weight in Logistic Regression and scale_pos_weight in XGBoost). For an XGBoost model, the hyperparameter scale_pos_weight controls the balance of positive and negative weights and is used to treat unbalanced classes. The typical value for scale_pos_weight is sum(negative instances)/sum(positive instances). Since the negative class (target=0) is around 1/5 of the positive class (target=1) in our project , the scale_pos_weight was estimated to be 0.2 in the XGBoost model.

Training models with hyperparameter tuning

A baseline model was established with the cleaned data to set a benchmark for testing. Multiple algorithms were examined to build classification models including Logistics Regression, Random Forest, SVC, KNN (Euclidean and Manhattan) and XGBoost. Several hyperparameter tuning techniques were applied to optimise the models under each algorithm including manual search, random search, grid search and automatic search with the Hyperopt package. Table 2 summarises models with different algorithms and hyperparameter tuning approaches. The performance metric (ROC-AUC) and Kaggle score are also included in the table for evaluating models in the next section.

Table 2: Models and Results

Model	Hyperparameter Tuning /Imbalance Treatment	ROC-AUC	Kaggle Score
Baseline	N/A	0.5	N/A
XGBoost	Default setting	Train=0.9992, Val=0.6629	0.70282
XGBoost	Manual search	Train=0.7185, Val=0.7171	0.70941
XGBoost	Grid search	Train= 0.7153, Val= 0.7163	Not submitted
XGBoost	Hyperopt package	Train = 0.7338, Val=0.7200	0.7049
XGBoost	All Parameters, Hyperparameter tuning with grid search	Train = 0.9991, Val=0 .6319	0.68873
SVM	Default setting	Train = 0.8131, Val = 0.6251	Not submitted
SVM	Manual Search with tuned C, Gamma and Kernel Type.	Train = 0.7801, Val =0.6498	Not submitted
Random Forest	Default setting	Train = 0.7770, Val=0.7121	0.69707
Random Forest	Default setting with random search	Train = 0.7812, Val=0.7067	0.70419
Random Forest	Default setting with Grid Search	Train = 0.7885, Val = 0.7083	Not submitted
Random Forest	Random Search with tuned settings	Train = 0.8030, Val = 0.7097	0.6969
Logistics Regression	Default setting	Train= 0.7002, Val= 0.7263	0.70854

Logistic Regression	Default setting with SMOTE	Train= 0.7049, Val= 0.7265	Not submitted
Logistic Regression	Default setting with Tomek Links method	Train = 0.7125, Val=0.7262	0.70866
Logistic Regression	Hyperparameter class_weight = 'balanced'	Train= 0.7000, Val= 0.7259	Not submitted
Logistic Regression	All Parameters	Train=0.7062, Val=0.6996	Not Submitted
AdaBoost	Default setting	Train = 0.7226, Val=0.7084	0.70282
Knn Manhattan	All Parameters, Hyperparameter tuning with grid search	Train=0.8565, Val=0.5881	Not Submitted
Knn Manhattan	Removed parameters 'Id', '3P Made', '3PA', '3P%' and 'BLK' and Hyperparameter tuning with grid search	Train= 0.8567 Val= 0.5927	Not Submitted
KNN Euclidean	All Parameters, Hyperparameter tuning with grid search	Train = 0.8532 Val= 0.5966	Not Submitted
KNN Euclidean	Removed parameters 'Id', '3P Made', '3PA', '3P%' and 'BLK' and Hyperparameter tuning with grid search	Train = 0.8500, Val=0.5893	Not Submitted

Evaluation

The metric ROC-AUC score is used to assess the model performance and identify the two best models based on the experiments discussed in the Modelling section. Table 3 lists the best models with the setting of hyperparameters.

Table 3: Best Models

Model	ROC-AUC	Kaggle Score	Hyperparameters
XGBoost	Train = 0.7185, Val=0.7171	0.70941	n_estimators=100, eta=0.02, max_depth=3, subsample=0.8, scale_pos_weight=0.2, min_child_weight=1.5, gamma=5
Logistic Regression	Train = 0.7125, Val=0.7262	0.70866	default (Treating imbalanced data with undersampling method Tomek Links)

Considering AUROC as our metric, all classification models performed better than their counterpart baseline models. By exploring various lines of experimentation, the models yielding the best performance were found to be XGBoost with Manual Hyperparameter tuning and Logistic Regression with Tomek Links undersampling. Their scores in combination with the lack of overfitting highlight their ability to predict if a rookie player will last at least 5 years in the NBA league based on their historical statistics.

Figure 2 presents the confusion matrix for the best model - XGBoost. It can be seen that the sensitivity and specificity are 26% and 92% respectively. This indicates the model has a high accuracy in predicting whether a rookie player will last for over 5 years in the NBA leagues while suffers from a relatively low accuracy in identifying the rookie player who played less than 5 years. Also export the confusion matrix plot for Logistic Regression model and include it in Appendix E.

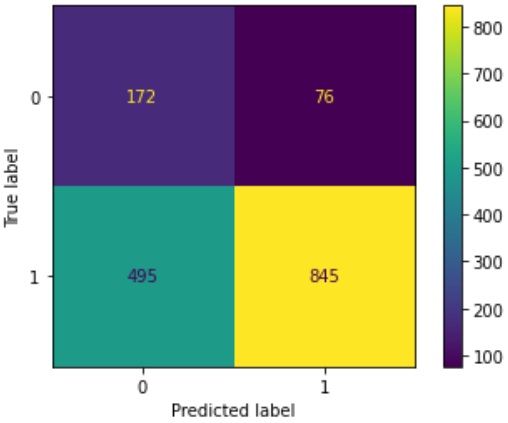


Figure 2 Confusion Matrix Plot (XGBoost)

The great accuracy in predicting a rookie player who will play for over 5 years in the NBA league brings more business value than a good prediction for players who will last for less than 5 years. Selecting a good player accurately will bring a continuous financial benefit to the business via television, merchandising, sponsorships and tickets and build a stronger brand name in the long run. The business might be exposed to risks of signing up some player who will not stay long in the NBA league but the business suffers less from this compared to missing a good player. From the business perspective, the best two classifiers above achieve the success criteria.

Additional review process is needed before the model is deployed in production which takes into account of the aspects below:

- Was anything missing?
- Were all phases properly executed?
- Need more time?
- Any other questions relevant to the review process

By summarising our findings and correcting anything if required, we could make a better decision if the model is ready for deployment, needs more iteration, or just creates a new project.

Deployment

The deployment of our XGBoost model will involve the following:

1. Deployment via Batch inference. This method will run periodically and generate the results as and when the new data is updated, providing insights. Whilst it generates answers with more latency, it has the ability to deploy more complex models such as XGBoost. Given the predictions will be performed in the historic data related to the players at the end of each season, live predictions will not be a viable option.
2. Creating a web app with streamlit. Streamlit helps to deploy the ML model in the form of web applications and increases the accessibility of the real-time prediction for NBA rookie players' career with friendly user interface and compelling insight through visualisation.
3. Automatic deployment and testing. As new players progress through their rookie season, new data can be implemented into the model to ensure predictions align with current statistics. Automation will create a user-friendly experience in generating these predictions and reduce the risk of manual errors.
4. Continuous monitoring and improvement. ML models can degrade over time and continuous monitoring will allow us to preemptively identify potential issues before they cause material risk whilst maintaining the models' performance.

Reflection on issues faced

Imputation methods

To apply the imputation methods to outliers, several issues were faced such as how to implement different imputation methods on outliers as these methods are normally applied to missing values and how to code these methods in Python and apply them to our dataset. To resolve the issue, the outliers were first identified using specific criteria and then replaced with np.nan. After this, different imputation methods (including single, multiple and knn) were implemented with the sklearn package in Python. The solution was based on research from several articles in Medium and the documentation page of sklearn.

Importing self-defined functions into the notebook

After creating the functions in the python file under the folder of 'src/data' such as the function split_train_test() in set.py file, we came across an error message when importing this function into the notebook with the code 'from src.data.sets import split_train_test'. By doing research online, the problem was solved by adding the code "import sys" and "sys.path.insert(1, '..')" before the function is imported. The additional code helps to add the current folder path to the system folder path so that the function can be located and imported successfully.

Managing the performance metrics of the test dataset

We found it getting harder to track all the evaluation metrics for each model and the score received after each submission in Kaggle. To track this, a database table was created on the Notion page to record the model information such as hyperparameters and their evaluation results. The Kaggle score was added to the database immediately after each submission. This is a temporary solution and better approaches will be explored in future experiments.

Using Github

We had to go through a steep learning curve to understand how github works and how we can move our local repository to a Github Branch. As a team, we collaborated and transferred the knowledge to help each other with moving the model and documents to Github without over-writing each other's work.

Member Contribution

Member Name	Contribution
Amy Yang	• Prepared the data preprocessing file including trying different methods for data cleaning (removing rows and imputation methods), feature engineering, splitting training and test sets, scaling the data and treating imbalanced data • Explored different models with hyperparameter tuning process including Logistic Regression, Random Forest, XGBoost, AdaBoost • Made Kaggle submissions and tracked metrics results • Drafted, edittied and finalised the final report • Set up Github repo with branches and maintained both the master branch and ‘amy’ branch • Organised meetings and recorded meeting minutes
Chanthru Vimalasri	• Prepared the data preprocessing files, specifically looking at feature engineering and data cleaning with a focus on increasing the dimensionality of the dataset by converting negative values to absolute rather than removing columns. • Explore models with a focus on hyperparameter tuning for Random Forest, Support Vector Machine (SVM) and XGBoost • Planned, organised and distributed lines of experimentation between team members to ensure the most effective use of time and expertise. • Made Kaggle submissions an tracked metric results • Drafted, edited and finalised individual experiment reports and final Kaggle group report • Maintained ‘chanthru’ branch within Github and pushed relevant updates to the master branch
	• Prepared the data preprocessing files which included data

Yatindra Vegunta	cleansing, duplicate data checks, null value checks and feature engineering ● Explored models with hyperparameter tuning including Random Forest, KNN Manhattan, KNN euclidean, Logistic regression and XGBoost. ● Editing and contributing to the final report with my experiments documented. ● Made Kaggle submissions an tracked metric results ● Maintained ‘yatin’ branch within Github and pushed relevant updates

Conclusion

We have initially faced issues in understanding what the parameters mean and how we should be utilising them in the models. As we collaborated, we have decided to use the datasets based on the common understanding within the team. We have individually carried out experiments by building various models and later comparing them to each other's models to decide which model is a viable option for deployment. We would like to conclude our findings by selecting the best fit model based on the various experiments carried out in previous weeks. We have chosen the XGBoost with hyperparameters mentioned in the evaluation section above. This model was chosen based on the Kaggle score it has generated, when compared to all other models listed in table2.

Given the results and the various lines of experimentation explored, our recommendation is not to devote further resources to this project. Through multiple iterations it is clear that an AUROC score of approximately 0.71 is the highest that is feasible right now. If a greater variety of data can be introduced through more features, we can resume the project and begin testing again to see if we can develop a model with better predictive ability.

Appendix

Appendix A - Data Dictionary

Data Fields	Description
Id	Player Identifier
GP	Games Played
MIN	Minutes Played
PTS	Points Per Game
FGM	Field Goals Made
FGA	Field Goals Attempts
FG%	Field Goals Percent
3P Made	3-Points Made
3PA	3-Points Attempts
3P%	3-Points Percent
FTM	Free Throw Made
FTA	Free Throw Attempts
FT%	Free Throw Percent
OREB	Offensive Rebounds
DREB	Defensive Rebounds
REB	Rebounds
AST	Assists
STL	Steals
BLK	Blocks
TOV	Turnovers
TARGET_5Yrs	Outcome:1 if career length >= 5 years, 0 otherwise

Appendix B - Data Summary

	Id	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	3P%
count	8000.00000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000
mean	7798.50000	62.777875	18.576662	7.267088	2.807037	6.231212	44.608900	0.264525	0.816562	19.583700
std	2309.54541	17.118774	8.935263	4.318732	1.693373	3.584559	6.155453	0.384093	1.060964	16.003155
min	3799.00000	-8.000000	2.900000	0.800000	0.300000	0.800000	21.300000	-1.100000	-3.100000	-38.500000
25%	5798.75000	51.000000	12.000000	4.100000	1.600000	3.600000	40.400000	0.000000	0.100000	8.400000
50%	7798.50000	63.000000	16.800000	6.300000	2.400000	5.400000	44.400000	0.300000	0.800000	19.500000
75%	9798.25000	74.000000	23.500000	9.500000	3.700000	8.100000	48.700000	0.500000	1.500000	30.600000
max	11798.00000	123.000000	73.800000	34.200000	13.100000	28.900000	67.200000	1.700000	4.700000	82.100000

	FTM	FTA	FT%	OREB	DREB	REB	AST	STL	BLK	TOV	TARGET_5Yrs
count	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000	8000.000000
mean	1.392525	1.947788	71.365825	1.077838	2.168500	3.245300	1.624513	0.648687	0.245212	1.257763	0.833625
std	0.926153	1.252352	10.430447	0.785670	1.392224	2.085154	1.355986	0.407626	0.821037	0.723270	0.372440
min	0.000000	0.000000	-13.300000	0.000000	0.200000	0.300000	0.000000	0.000000	-17.900000	0.100000	0.000000
25%	0.700000	1.000000	65.000000	0.500000	1.100000	1.700000	0.700000	0.300000	0.100000	0.700000	1.000000
50%	1.200000	1.700000	71.400000	0.900000	1.900000	2.800000	1.300000	0.600000	0.200000	1.100000	1.000000
75%	1.900000	2.600000	77.500000	1.500000	2.900000	4.300000	2.200000	0.900000	0.400000	1.600000	1.000000
max	8.100000	11.100000	168.900000	5.500000	11.000000	15.900000	12.800000	3.600000	18.900000	5.300000	1.000000

Appendix C - Rationale for Removed Features

The reasons for the removal of these features are summarised below:

- The Id column is only used to identify the order of the dataset and does not have any relevance to the target variable, removing this column ensures that only relevant features are used to fit the model.

- Removing the other features (3P Made, 3PA, 3P% and BLK) is based on the rationale that the unreasonable values in these columns are significant compared to the total training data size. For example, there are 1,628 rows with negative values in the column of '3P Made'. If we remove all these records from the dataset, it means around 20% of the data size will be lost and can not be used to build the model. Instead of reducing the data size, removing the whole column would be a better choice under this circumstance. Similar cases with other features (3PA, 3P% and BLK).

- Alternative options were explored and considered for the treatment of these features, such as imputation methods, and keeping the features. Based on evaluation metrics, removing these features enables better performance of different classifiers than applying imputation methods.

Appendix D - Scaling the data

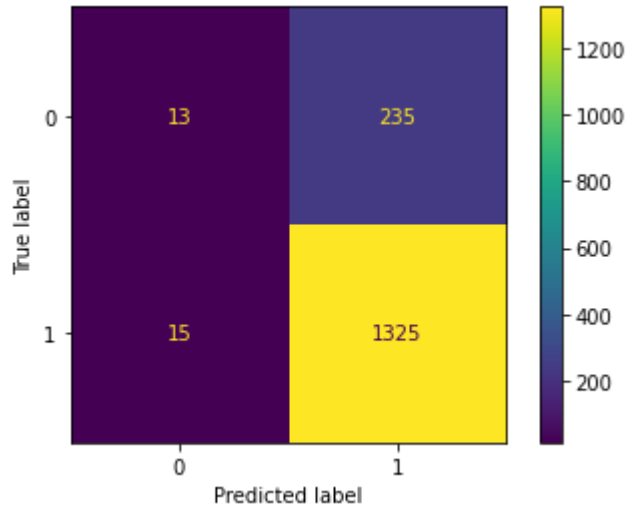
Classifiers without scaling the data

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
QuadraticDiscriminantAnalysis	0.61	0.63	0.69	0.66	0.03
BernoulliNB	0.63	0.63	0.68	0.68	0.03
GaussianNB	0.54	0.61	0.69	0.60	0.02
LabelPropagation	0.76	0.54	NaN	0.76	3.98
KNeighborsClassifier	0.82	0.54	0.59	0.79	0.35
LabelSpreading	0.76	0.54	NaN	0.76	3.67
ExtraTreeClassifier	0.74	0.54	0.54	0.75	0.03
DecisionTreeClassifier	0.73	0.54	0.54	0.74	0.13
XGBClassifier	0.83	0.53	0.66	0.79	0.67
LinearDiscriminantAnalysis	0.84	0.53	0.73	0.79	0.06
BaggingClassifier	0.80	0.53	0.62	0.78	0.57
RandomForestClassifier	0.84	0.52	0.67	0.79	1.43
AdaBoostClassifier	0.84	0.51	0.70	0.78	0.40
CalibratedClassifierCV	0.84	0.51	0.73	0.78	1.86
LGBMClassifier	0.83	0.51	0.69	0.78	0.20
ExtraTreesClassifier	0.84	0.51	0.67	0.78	0.83
LogisticRegression	0.84	0.50	0.73	0.77	0.09
DummyClassifier	0.84	0.50	0.50	0.77	0.02

Classifiers with scaling the data

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
QuadraticDiscriminantAnalysis	0.61	0.63	0.69	0.66	0.04
BernoulliNB	0.63	0.63	0.68	0.68	0.02
GaussianNB	0.54	0.61	0.69	0.60	0.02
LabelPropagation	0.76	0.54	NaN	0.76	2.22
KNeighborsClassifier	0.82	0.54	0.59	0.79	0.17
LabelSpreading	0.76	0.54	NaN	0.76	3.40
ExtraTreeClassifier	0.74	0.54	0.54	0.75	0.02
DecisionTreeClassifier	0.73	0.54	0.54	0.74	0.10
XGBClassifier	0.83	0.53	0.66	0.79	0.79
LinearDiscriminantAnalysis	0.84	0.53	0.73	0.79	0.05
BaggingClassifier	0.80	0.53	0.62	0.78	0.56
RandomForestClassifier	0.84	0.52	0.67	0.79	1.40
AdaBoostClassifier	0.84	0.51	0.70	0.78	0.41
CalibratedClassifierCV	0.84	0.51	0.73	0.78	1.74
LGBMClassifier	0.83	0.51	0.69	0.78	0.27
ExtraTreesClassifier	0.84	0.51	0.67	0.78	0.77
LogisticRegression	0.84	0.50	0.73	0.77	0.08
DummyClassifier	0.84	0.50	0.50	0.77	0.01

Appendix E - Confusion Matrix Plot (Logistic Regression)



Sensitivity: 46%
Specificity: 85%