Building on the Duckworth-Lewis Standard Edition method for ODI cricket using machine learning

MSc Data Science Report, Birkbeck College, University of London

Sophie Walker

25thj January 2024

Abstract

Since 1999 the Duckworth-Lewis Method (DL), in its various forms, has been used in One Day International (ODI) cricket matches to create a fair number of target runs for teams to reach in the event that the match is unable to be played in full due to poor weather. DL supposes that both teams have finite "resources" in the form of remaining wickets to fall and overs to play and prorates the target based on how many of the resources remain.

In this project it's shown that by extending the definition of resources to include the remaining types of players and the average scores at each cricket venue, and by using machine learning methods (Random Forest, Support Vector Machine and Feedforward Neural Network) the predictive ability of DL can be significantly improved.

NB. For the purposes of this project I'll be using the publicly available Duckworth-Lewis method from 2002 and will call it DL to distinguish it from later updates including the method now used in international matches, DLS, which is protected for commercial reasons.

Table of Contents

Abstract 2

1. Introduction

1.1 Outline 8

1.2 Aims 8

1.3 Introduction to Cricket 8

1.4 Cricket Formats 9

1.5 Duckworth-Lewis Rule 9

1.6 DL Standard Edition and Updates 12

1.7 Criticism of Duckworth-Lewis Methods

1.7.1 Complexity 14

1.7.2 Bias 14

1.7.3 Probabilistic approaches 15

1.7.4 Open to manipulation 15

1.7.5 Wickets 15

1.7.6 Parametric vs non-parametric 15

1.7.7 Powerplays/ fielding restrictions 16

1.7.8 T20 16

1.8 Other methods

1.8.1 Methods updating the model 16

1.8.2 Models that add or change the definition of “resources” 17

1.8.3 Machine and Deep Learning 18

1.8.4 Summary 18

2. Data

2.1 New “Resources” 19

2.2 Data Collection

2.2.1 Match data 19

2.2.2 Player data 20

2.2.3 Field data 20

2.2.4 Duckworth Lewis 20

2.3 Data Preparation

2.3.1 Match data 21

2.3.2 Cricketer data 22

2.3.3 DL tables 23

2.3.4 Grounds data 23

2.4 Train, Validation, Test split 23

2.5 Imbalanced data 24

2.6 Powerplays 26

3. Models

3.1 Model choice 27

3.2 Performance metrics 29

3.2.1 Multiclass classification 30

3.3 Bias Variance Trade-off, Learning curves and Cost functions 32

3.3.1 Cost/ Loss function 33

3.4 Random Forest 34

3.4.1 Training the model 35

3.4.2 Overall design of RF 38

3.5 Feature Scaling 39

3.6 Feature Selection

3.6.1 Pearson Correlation 42

3.6.2 Random Forest 43

3.7 Support Vector Machine

3.7.1 Training the model 44

3.7.2 Overall design of SVM 45

3.8 Neural Network

3.8.1 Training the model 46

3.8.2 Overall design of NN 49

4. Systems Architecture

4.1 Random Forest 50

4.2 Support Vector Machine 51

4.3 Neural Network 51

4.4 Evaluation 52

5. Results and Analysis

5.1 Accuracy 53

5.2 Confusion matrices, precision and recall 53

5.3 Distribution of classes 55

5.4 Other analysis 56

6. Conclusion

6.1 Looking back at the aims of this project 57

6.2 Overcoming criticisms 57

6.3 Limitations/ Further Study

6.3.1 Data 58

6.3.2 Resources 58

6.3.3 Classes 58

6.3.4 Computation 59

6.3.5 Models 59

7. References 60

8. Appendices

Appendix I - Python libraries 66

Appendix II – Github readme 67

Table of Figures

1. Labelled diagram of bowling in cricket
2. Illustrative DL graph
3. Excerpt from the DL tables
4. Screenshot of normalised JSON
5. Screenshot of match_df dataframe
6. Example of cricketer name formats
7. Code to calculate dl percentage
8. Data split code
9. Bar chart – class distribution
10. Table of class distribution %
11. Confusion matrix of DL predictions vs match outcomes
12. Performance metrics of DL predictive ability
13. Summary table of papers on DL
14. Table comparing RF, SVM and NN
15. Illustrative diagram of a confusion matrix
16. Example of performance metrics table
17. Example of a 3-class confusion matrix
18. Bias Variance graph
19. Example Accuracy and Loss learning curves
20. Example code showing loss term
21. Labelled Decision Tree diagram
22. Example Decision Tree
23. Diagram of a Random Forest model
24. Table of hyperparameter tuning methods
25. Table of hyperparameters tuned in different papers and websites
26. Table of hyperparameters and purpose
27. Accuracy learning curve n_estimators
28. Loss learning curve n_estimators
29. Hyperparameter importance – ANOVA scores
30. Accuracy learning curve max_samples
31. Loss learning curve max_samples
32. Table of hyperparameter choices for RF
33. Histogram plots for data features
34. Screenshot of code scrapped because of computational cost
35. Time taken for SVM to train using StandardScaled and SMOTE data
36. Time taken for SVM to train using MinMax and SMOTE data
37. Best accuracy for SVM trained using StandardScaled and SMOTE data and hyperparameters
38. Best accuracy for SVM trained using MinMax and SMOTE data and hyperparameters
39. Pearson correlation matrix of the data features
40. Feature importance diagram
41. Diagram of a hyperplane in increasing dimensions
42. Hinge loss diagram
43. Table of best accuracy scores using different scaling techniques and different features selected
44. Table of hyperparameter choices for SVM
45. Diagram of an FFN
46. Example graph of validation error against number of epochs
47. Table of activation functions and their purpose

48. Table of hyperparameter choices for NN
49. Snippet of code training RF
50. Snippet of code for ANOVA
51. Snippet of code training SVM
52. Snippet of code tuning SVM hyperparameters
53. Snippet of code – function to create NN
54. Snippet of code tuning NN hyperparameters
55. Snippet of confusion matrix and accuracy_score code
56. Table comparing accuracy of models
57. Confusion matrix Duckworth Lewis
58. Performance metrics DL
59. Confusion matrix RF
60. Performance metrics RF
61. Confusion matrix SVM
62. Performance metrics SVM
63. Confusion matrix NN
64. Performance metrics NN
65. Bar charts of class distribution for: Actual match outcome, DL, RF, SVM and NN predictions
66. Table showing the difference between the actual class distribution and the different models
67. Feature importance bar chart
68. Table of DL criticisms and whether the new models overcame them
69. Boxplot of Wickets taken against the outcome (0 for Team 1 win, 1 for Tie and 2 for Team 2 win.)

1. Introduction

1.1 Outline of this report

This report lays out:

- The aims of this project.
- The background to the Duckworth-Lewis method and why improvement is needed.
- The data used and why.
- How the models were chosen, and how they were trained.
- Results and analysis.
- Suggestions for future study.

1.2 Aims

- To train a Random Forest, Support Vector Machine and Neural Network to improve the predictive capability of the Duckworth-Lewis method.
- To extend the definition of resources to include scoring rates at cricket grounds and player "value" to improve upon predictive capability of Duckworth-Lewis method.

1.3 Introduction to cricket

Cricket is a popular bat-and-ball sport played between two teams of eleven players where each team takes turns to either bat or field. There are three main formats of cricket: Test, One Day or T20) but the rules below are universal.

Each round that one team bats and the other fields is called an "innings." During their batting innings the batting team will have two players on the field, one at each end of the pitch, standing in front of their wickets. Their task is to hit the ball and score "runs" by running the length of the pitch between the wickets.

Meanwhile, the opposition team will be fielding, with all eleven players on the field. Their task is to bowl the ball at the facing batter and try to dismiss them. A bowling member of the fielding team will deliver six balls aka "deliveries" to the batter in an "over."

When a batter is dismissed, they are replaced by another batter until there are no remaining batters, at which point the innings is over.

How the match ends depends on the format of the game but the aim of the game is to score the larger number of runs by the time the match is completed.

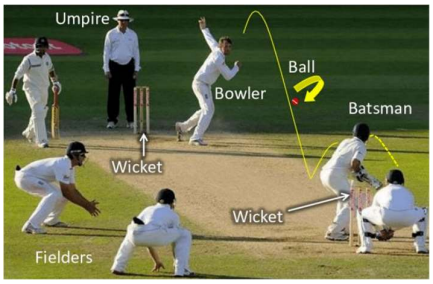


Figure 1. Labelled diagram of bowling in cricket (Egyir & Devendorf, 2020)

1.4 Cricket Formats

Introduced in the late 19th century Test is the oldest cricket format. It entails four innings over up to five days of play, and the winning team is the one who has dismissed the other team in full twice and has the higher score. If that's not achieved the match ends in a draw, and a tie is achieved if both teams have the same score, but is incredibly rare, only twice in international matches since the format began (ESPN, n.d.)

The challenge with test cricket is that it can often be slow and crowds can rarely watch the full match (Lavalette, 2019), and so in response to waning crowds, One Day cricket was introduced. This format was designed with spectators in mind, taking place over only one day consisting of 2 innings, 50 overs and a maximum of 9 hours of play. To increase the excitement of the match there's no option to draw, instead the team to win is the one who has scored the higher number of runs when the two innings are finished (either by the 50 overs ending or dismissal of all the batters). A tie is more common in ODIs, 43 since inception (ESPN, n.d.) but still rare.

Lastly T20 matches are similar to ODIs in that they are also time limited. These can be played in an evening and only consist of 20 overs each side and a typical match taking 2.5 hours (Findlater, 2021). T20s aren't a focus of this project but are relevant to future research.

1.4 Duckworth-Lewis

But what happens when there's bad weather? In test matches there are five possible days of play and an option to draw so there's time to reach a conclusion even when rain stops play. However, for ODIs there's a risk of not finishing the match at all, defeating the object of removing the option to draw. In cases where a match must be shortened if the delay happens before play begins it is simple to reduce the overs for both, but if there's an interruption after play has begun the situation becomes more complex.

Initially it was assumed that any team who was unable to complete their batting innings would have continued to bat at the same run rate until completion (Average Run Rate) and using that method you could extrapolate what target score was necessary to reach. However, scoring isn't linear as depending on how many overs are left to play and how many wickets the team has remaining, a team may bat more aggressively or defensively.

In 1998, statisticians Frank Duckworth and Tony Lewis addressed this problem in their paper *A fair method for resetting the target in interrupted one-day cricket matches*. They highlighted that a team's strategy will change depending on their individual situation, and set out five aims for an alternative to ARR including: Equal fairness to both sides so that the "relative positions of the two teams should be exactly the same after the interruption as they were before it", easy applicability and understandability. (Duckworth & Lewis, A Fair Method for Resetting the Target in Interrupted One-Day Cricket Matches, 1998).

Their method (DL) hypothesises that each team has a set of "resources" (the number of wickets taken and number of overs still to bat) and compares the proportion of resources both teams had remaining when the play was suspended to create a new target score to win. This is illustrated in the DL tables (Figure 2) and in the example below:

- First batting team (Team 1) faced all 50 overs so used 100% of their resources and scored 200 runs. If second batting team (Team 2) also faced all 50 overs then the target score to beat would be 200 runs as they both had the same opportunity to win.
- However, play stopped after 32 overs for Team 2 after they had lost 4 wickets.
- Looking at the tables for 18 overs remaining and 4 wickets lost shows 42% resources remaining.
- Therefore Team 2 used 100-42% = 58% of their resources, and the score to draw (the par score) would be 58%*200 = 116 runs, and the target score to win would be 117.
- If at the end of the match Team 2 scored more than 116 runs they would win the match, 116 exactly they would tie the match and less than 116 they would lose the match. (ICC, 2017)

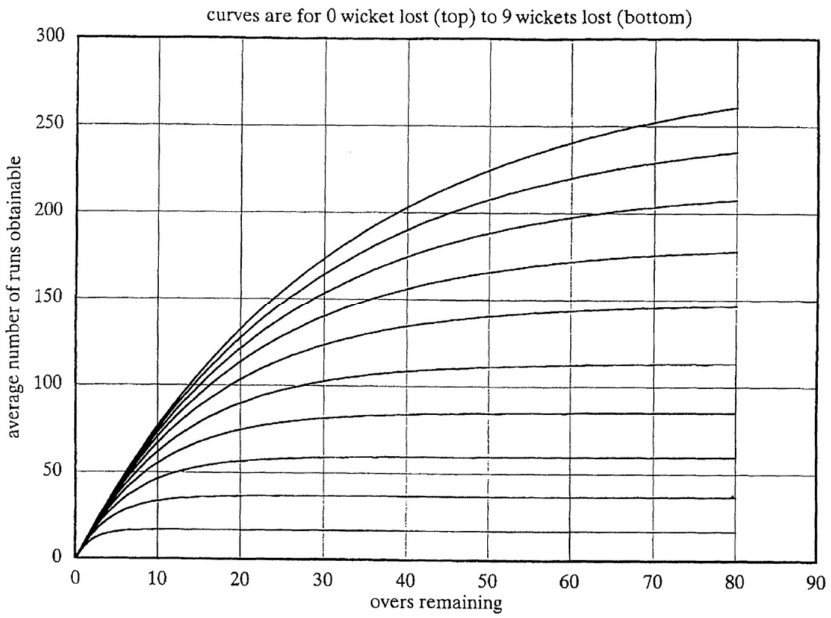


Figure 2. Illustrative DL graph showing the relationship between remaining overs, wickets and potential runs. (Duckworth & Lewis, A Fair Method for Resetting the Target in Interrupted One-Day Cricket Matches, 1998)

The D/L (Duckworth/Lewis) method of adjusting target scores in interrupted one-day cricket matches

Table of resource percentages remaining - over by over

2002 update

Overs left 50 to 0

overs left	wickets lost 0	1	2	3	4	5	6	7	8	9	overs left
50	100.0	93.4	85.1	74.9	62.7	49.0	34.9	22.0	11.9	4.7	50
49	99.1	92.6	84.5	74.4	62.5	48.9	34.9	22.0	11.9	4.7	49
48	98.1	91.7	83.8	74.0	62.2	48.8	34.9	22.0	11.9	4.7	48
47	97.1	90.9	83.2	73.5	61.9	48.6	34.9	22.0	11.9	4.7	47
46	96.1	90.0	82.5	73.0	61.6	48.5	34.8	22.0	11.9	4.7	46
45	95.0	89.1	81.8	72.5	61.3	48.4	34.8	22.0	11.9	4.7	45
44	93.9	88.2	81.0	72.0	61.0	48.3	34.8	22.0	11.9	4.7	44
43	92.8	87.3	80.3	71.4	60.7	48.1	34.7	22.0	11.9	4.7	43
42	91.7	86.3	79.5	70.9	60.3	47.9	34.7	22.0	11.9	4.7	42
41	90.5	85.3	78.7	70.3	59.9	47.8	34.6	22.0	11.9	4.7	41
40	89.3	84.2	77.8	69.6	59.5	47.6	34.6	22.0	11.9	4.7	40
39	88.0	83.1	76.9	69.0	59.1	47.4	34.5	22.0	11.9	4.7	39
38	86.7	82.0	76.0	68.3	58.7	47.1	34.5	21.9	11.9	4.7	38
37	85.4	80.9	75.0	67.6	58.2	46.9	34.4	21.9	11.9	4.7	37
36	84.1	79.7	74.1	66.8	57.7	46.6	34.3	21.9	11.9	4.7	36
35	82.7	78.5	73.0	66.0	57.2	46.4	34.2	21.9	11.9	4.7	35
34	81.3	77.2	72.0	65.2	56.6	46.1	34.1	21.9	11.9	4.7	34
33	79.8	75.9	70.9	64.4	56.0	45.8	34.0	21.9	11.9	4.7	33
32	78.3	74.6	69.7	63.5	55.4	45.4	33.9	21.9	11.9	4.7	32
31	76.7	73.2	68.6	62.5	54.8	45.1	33.7	21.9	11.9	4.7	31
30	75.1	71.8	67.3	61.6	54.1	44.7	33.6	21.8	11.9	4.7	30
29	73.5	70.3	66.1	60.5	53.4	44.2	33.4	21.8	11.9	4.7	29
28	71.8	68.8	64.8	59.5	52.6	43.8	33.2	21.8	11.9	4.7	28
27	70.1	67.2	63.4	58.4	51.8	43.3	33.0	21.7	11.9	4.7	27
26	68.3	65.6	62.0	57.2	50.9	42.8	32.8	21.7	11.9	4.7	26
25	66.5	63.9	60.5	56.0	50.0	42.2	32.6	21.6	11.9	4.7	25
24	64.6	62.2	59.0	54.7	49.0	41.6	32.3	21.6	11.9	4.7	24
23	62.7	60.4	57.4	53.4	48.0	40.9	32.0	21.5	11.9	4.7	23
22	60.7	58.6	55.8	52.0	47.0	40.2	31.6	21.4	11.9	4.7	22
21	58.7	56.7	54.1	50.6	45.8	39.4	31.2	21.3	11.9	4.7	21
20	56.6	54.8	52.4	49.1	44.6	38.6	30.8	21.2	11.9	4.7	20
19	54.4	52.8	50.5	47.5	43.4	37.7	30.3	21.1	11.9	4.7	19
18	52.2	50.7	48.6	45.9	42.0	36.8	29.8	20.9	11.9	4.7	18
17	49.9	48.5	46.7	44.1	40.6	35.8	29.2	20.7	11.9	4.7	17
16	47.6	46.3	44.7	42.3	39.1	34.7	28.5	20.5	11.8	4.7	16
15	45.2	44.1	42.6	40.5	37.6	33.5	27.8	20.2	11.8	4.7	15
14	42.7	41.7	40.4	38.5	35.9	32.2	27.0	19.9	11.8	4.7	14
13	40.2	39.3	38.1	36.5	34.2	30.8	26.1	19.5	11.7	4.7	13
12	37.6	36.8	35.8	34.3	32.3	29.4	25.1	19.0	11.6	4.7	12
11	34.9	34.2	33.4	32.1	30.4	27.8	24.0	18.5	11.5	4.7	11
10	32.1	31.6	30.8	29.8	28.3	26.1	22.8	17.9	11.4	4.7	10
9	29.3	28.9	28.2	27.4	26.1	24.2	21.4	17.1	11.2	4.7	9
8	26.4	26.0	25.5	24.8	23.8	22.3	19.9	16.2	10.9	4.7	8
7	23.4	23.1	22.7	22.2	21.4	20.1	18.2	15.2	10.5	4.7	7
6	20.3	20.1	19.8	19.4	18.8	17.8	16.4	13.9	10.1	4.6	6
5	17.2	17.0	16.8	16.5	16.1	15.4	14.3	12.5	9.4	4.6	5
4	13.9	13.8	13.7	13.5	13.2	12.7	12.0	10.7	8.4	4.5	4
3	10.6	10.5	10.4	10.3	10.2	9.9	9.5	8.7	7.2	4.2	3
2	7.2	7.1	7.1	7.0	7.0	6.8	6.6	6.2	5.5	3.7	2
1	3.6	3.6	3.6	3.6	3.6	3.5	3.5	3.4	3.2	2.5	1
0	0	0	0	0	0	0	0	0	0	0	0
overs left	0	wickets lost 1	2	3	4	5	6	7	8	9	overs left

Overs left 50 to 0

Figure 3. Excerpt from the DL tables

1.6 DL Standard Edition and Updates

In response to increasing scoring rates and additional data collected since 1998, DL has had two notable updates: 2004 Duckworth Lewis paper which updated the DL tables and created the DL Professional Edition (Duckworth & Lewis, A Successful Operational Research Intervention in One-Day Cricket, 2004), and 2016 Stern paper which created the DLS method (Stern, 2016).

In the former Duckworth and Lewis reviewed the performance of their own method and concluded that adjustments to the parameters of their formula was needed to "update the average contributions of early and later batsmen" and to "allow for the slightly higher average total runs that are being scored in ODIs." The DL tables were updated accordingly (Figure 2 - 2002 update) and these are the percentages used in this report.

One of the original aims of the DL method was to be simple enough for anyone to use pen and paper to calculate it, however the original model showed a weakness that was not possible to address using a static table. Teams making big scores appeared to score an even larger proportion of their runs early in their innings, but then behaved more like an average team towards the conclusion of their innings" (Jewson & French, The Duckworth-Lewis-Stern Method and County Cricket). And therefore under DL when Team 1's score was well-above average, the behaviour didn't match model's; it gave an advantage to Team 2 when play was stopped late in the second innings, and a disadvantage if stopped early in that innings (Stern, 2016).

In response the DL Professional Edition was created, which uses a "match factor" specific to the match being played to dampen the model and straighten it towards the average run rate, this "results in a team chasing a high score needing to get a larger proportion of their runs earlier in the innings, than a team chasing an average score" (Stern, 2016). This cannot be captured in the tables so is incorporated into a computer-based method using proprietary software.

In the 2016 paper, Steven Stern updated the DL Professional Edition and created the Duckworth Lewis Stern method. For high scoring innings the real pattern of runs straightens towards the average run rate which is not uniform across the innings and therefore Stern introduces an additional damping factor, which is a function of the number of overs remaining and thus acts differentially throughout the innings. (Stern, 2016).

For the purposes of this project, the following naming conventions will be used to distinguish between the methods and updates:

- 1998 paper and updated 2002 Duckworth-Lewis method and tables will be called the DL Standard Edition (DL)
- 2004 paper and corresponding computer-based method will be called the DL Professional Method (DLP)
- 2016 paper and update to DLP will be called the Duckworth-Lewis-Stern method (DLS). DLS is colloquially what the method is now called since Steven Stern became custodian of the method, but in recent papers the term DL is still used to refer to the older methods. (Hazra, Chatterjee, Mandal, Sarkar, & Mandal, 2023)

The DL tables are available to the public, but not the formulae and parameters of the model themselves, nor the computer-based models for DLP or DLS. Therefore, for this project the 2002 tables were used to calculate the target scores and winners of ODI matches. Despite not using the most modern model, in later updates the model has remained mostly the same, with the main change being the addition of a match factor, so this is still a meaningful

comparison, and is in line with ECB guidance which says that if access to DLS is not possible then the method should revert to DL (ECB, 2023).

1.7 Criticism of Duckworth-Lewis Methods

Since its adoption by the International Cricket Council (ICC) in 1999 there's been a mixed reception to DL, with criticism levied against the method including:

- Complexity
- Bias
- Not taking into consideration probability of winning
- Being open to manipulation
- Taking wickets too much into consideration
- Having a parametric rather than non-parametric model
- Not considering fielding restrictions
- Not applying to T20

1.7.1 Complexity

One of the key tenets of the DL method is to be easy enough to calculate with a pen and paper. However, DL has developed a reputation for confusing fans (Angel, 2013) (Fraser & Sheridan, 2022) as well as professionals.

In the 2015 England v New Zealand World Cup ODI match England required 54 runs to win from 37 deliveries, with 7 wickets lost. However, rain descended and when the team returned to play they required 34 runs from 13 deliveries. After the match, experienced Captain Eoin Morgan said "I don't understand Duckworth-Lewis. I don't think anybody does" (Lancaster, 2015).

It only takes a glance at the example scenarios on the ICC website to understand why this might be the case. But the concern over confusion goes deeper than struggling with the tables themselves, as some agree that whilst it's simple to work out the target scores, understanding the reason behind them is the challenge (Alves, 2008).

1.7.2 Bias

DL has also been criticised for being biased. In *Review of Duckworth Lewis Method* it was shown that historically DL was biased towards Team 1 (Shah, Sampat, Savla, & Bhowmick, 2015). However, in a 2008 FAQ Duckworth and Lewis both disagreed, explaining that on average in ODIs the 1st batting team wins c.52% of the time, whilst the 2nd batting team wins 48%, and that that ratio stayed the same over the matches where DL was used to decide the result (Duckworth & Lewis, D/L method: answers to frequently asked questions, 2008).

The data used in this project tells a different story; contrary to Shah et Al's supposition the method slightly favours Team 2 instead. This is explored further in the results and analysis section of this project.

Others have argued that DL is intrinsically biased towards the second team because "the chasing side has an idea of what it needs to do in the required time frame" (Akhtar, 2014). In their 2008 Q&A Duckworth and Lewis explained that in some situations it may be that the team batting first is more disadvantaged and therefore the method should favour them, even where the number of overs both team face is the same in total. For example, if Team 1's innings is cut short then Team 2 will play for the same number of overs as Team 1 had. However, Team 1 planned the pace of their play based on 50 overs to bat and Team 2 now has the advantage

as they know ahead of time how many overs they have and can adjust their tactics accordingly (Duckworth & Lewis, D/L method: answers to frequently asked questions, 2008).

1.7.3 Probabilistic approaches

'The D/L method maintains the margin of advantage. It does not maintain the probability of winning or losing' (Duckworth & Lewis, Your comprehensive guide to the Duckworth/Lewis method for re-setting targets in one-day cricket, 1999). Whilst Duckworth and Lewis made it clear that this method isn't for maintaining the same probability of winning, others argue that this is the fairest way to deal with an interruption. This is exemplified by the 2015 England v New Zealand match mentioned above. At the start of the interruption England had a larger chance of victory than after, despite them not playing a single ball in that time (Lancaster, 2015).

This is considered further in "Models that add or change the definition of resources" later in this report.

1.7.4 Open to manipulation

There's also a suggestion that teams can purposely affect the target number of runs if they know a match will be cut short (Bhogle, 2003). For example, if during Team 2's batting innings there's a likelihood of rain coming, the team may bat more conservatively to preserve their wickets, giving them a smaller target to beat when play restarts. However, when asked "How should teams adapt their tactics when rain is threatening, or when following a stoppage?" Duckworth and Lewis described their method as "strategically neutral." They suggested that perhaps where Team 2 are not on track to reach their DL target score they should increase the run rate but recognise the risk of losing wickets. The balancing act of scoring more runs versus losing wickets is intrinsic within cricket. And the concern of potential manipulation is tough to balance with the concerns over the complexity of DL in the first place; the rules themselves are easier to manipulate if understood well but if they're too complicated their validity is challenged.

1.7.5 Wickets

Further criticism has been levelled at the DL method for placing too much emphasis on the fall of wickets. *Review of Duckworth Lewis* said that DL placed more emphasis on wickets rather than net run rate and runs scored (Shah, Sampat, Savla, & Bhowmick, 2015) and Vedant Sahu in his analysis *The Issues With the Duckworth-Lewis-Stern Method* said each fall of a wicket is not made equal, as an on-form batter is much more valuable to the team than someone having a bad day at the crease (Sahu, 2019).

1.7.6 Parametric vs non-parametric

In *Duckworth-Lewis and Twenty20 Cricket*, Bhattacharya, Gill and Swartz identify that the DL model is a parametric one, i.e. the functional form of the model is decided before the model is trained, making assumptions as to the shape. In their paper they train a non-parametric model instead which Jack Jewson in his thesis on DL says could prove fairer. "Although the choice of model and model parameters is influenced by observed data ultimately the decision is a

judgement made by the model builders. A non-parametric model would be based solely on observed data and thus involves less judgements." (Jewson & French, An Analysis of the Duckworth Lewis method in the context of English county cricket, 2015)

1.7.7 Powerplays/ fielding restrictions

DL has also been criticised for failing to take into consideration fielding restrictions (Powerplays) (Ramachandran, 2002). Therefore, in the proposal for this paper data on Powerplays was to be used as part of the resources in the machine learning/ deep learning models trained. However, following further research it was found that whilst Powerplays give a small advantage to the batting team, it also increases the likelihood of a wicket being taken. (Manage, Silva, & Swartz, 2015). This is explained further when discussing the data used in this project under (Section 2.6.)

1.7.8 T20

Lastly there are multiple papers on the appropriateness of using the DL method in T20 matches, which are shorter and faster-paced, and the fall of a wicket may not have the same effect (Sahu, 2019) (Bhattacharya, Gill, & Swartz, 2011) (Perera, 2011). This isn't explored further here but is worth noting for future consideration of extending the proposed models.

1.8 Other methods

Following the criticism of DL, other methods have been proposed to improve upon the model. These are split below into three categories: Methods updating the model itself, methods that add to or amend the definition of resources and methods that use machine learning.

1.8.1 Methods updating the model

DL assumes that batters will bat faster as the innings goes on, and therefore later overs are worth more runs. However, two methods refute this, with their models showing different trends.

The VJD method (Jayadevan, 2002) models an innings as seven phases of play where batters may speed up or slow down. However, this was rebutted in Asif & McHale's paper, in which they gave examples of the VJD method producing contradictory changes to the target runs. In their own paper and model (DLSMA) they put forward a different method by estimating the parameters for DL and put forward an alternative functional form of their equation which slowed down the increasing run rate as overs go on, to account for situations where the batters are already going as fast as possible.

Steven Stern compared both methods to his DLS method arguing that other methods gave specific examples where "DL and DLS targets are claimed to be 'non-intuitive" but that the DL and DLS methods are more logically consistent. He laid out four conditions for logical consistency, including that a "chasing team level with par at the time interruption should remain level with par on resumption of their chase." Providing an example scenario using the VJD method Stern showed that given identical starts to a match, if one match suffered a rain interruption between the 30^{th} and 40^{th} overs but the other didn't, then despite the batting teams in both scenarios scoring in line with the par score at the 30^{th} over, there was a higher run

requirement for the rain interrupted team to win (Stern, 2016). And that whilst the DSMA method did satisfy the four conditions for consistency, the DL (Standard Edition, none of the others were compared) was still the most fair overall (Schall & Weatherall, 2013).

1.8.2 Models that add or change the definition of "resources"

The Player-aware Resource Compensation in Interrupted Cricket Matches (PRCICM) model takes each player as a weighted resource for their team and ranks players based on their host country average, average of last five matches and career average (Salam, Liagat, Zia, Kong, & Shamshad, 2022).

Predicting the Match Outcome in One Day International Cricket Matches, While the Game is in Progress is not a paper on rain-affected matches, but did use the DL concept of percentage of resources remaining to predict the outcome of ODI matches on an over-by-over basis using multiple linear regression. Their model extended the definition of resources to include: "home ground advantage, past performances, match experience, performance at the specific venue, performance against the specific opposition, experience at the specific venue and current form." (Bailey & Clarke, 2006).

Whilst both papers show good predictive potential it's not in the spirit of DL to address the relative skillsets of each side to such a degree. DL is designed to give equal opportunity to both teams to win on the day rather than considering who is the objectively better team with better advantages.

What if resources were probabilistic instead? In their paper *Rain rules for limited overs cricket and probabilities of victory* Preston and Thomas explore a probability-method instead. (Preston & Thomas, Rain rules for limited overs cricket and probabilities of victory, 2002). The method "acknowledges that loss of run-scoring resources needs to be compensated, but resources are valued in terms of what matters to teams, which is the probability of winning, rather than – as under Duckworth-Lewis – the expected run total in one of the innings." (Preston & Thomas, Duckworth-Lewis has had a good innings in one-day cricket. Is it time for it to retire?, 2015)

They highlight the advantages of their rule as including: Fairness because it preserves the teams' advantage, understandability of the underlying concept and being independent of potential manipulation because there's no incentive to change strategy as the probability of winning remains the same. (Preston & Thomas, Rain rules for limited overs cricket and probabilities of victory, 2002).

However, in practice Steven Stern says that probability methods can lead to inconsistencies. For example, imagine three matches with almost identical circumstances where Team 2 are batting but the innings is interrupted:

- 1st match – Team 2's likelihood of winning is 50% because their score is in line with the par score.
- 2nd match – Team 2 has a higher than 50% chance of winning because their score is higher than the par score. Here we say that they have scored r more runs than par.
- 3rd match – Team 2 has a less than 50% chance of winning because their score is lower than par. Here say that they have score r runs less than par.

The interruptions happen at the same time and both teams have the same resources remaining, but under the probability method:

- 1st match - Team 2 score at the par score, they will tie with Team 1.
- 2nd match - Team 2 scores par – r runs. Despite being ahead by r runs before the interruption, Team 1 wins.
- 3rd match - Team 2 score par score + r runs. Despite being ahead they were behind by r runs before the interruption, Team 2 wins. (Stern, 2016)

In other words, under the probability methods the overs that aren't played are ignored, and the match effectively just shortened. Whereas DL takes the potential play that was lost into consideration.

Preston and Thomas also self-identify another challenge in their method, explaining that there are difficulties in "applying it consistently to cases where play must be abandoned."

1.8.3 Machine and Deep learning

In Abbas and Haider's *Duckworth-Lewis-Stern Method Comparison with Machine Learning Approach* a range of classification models were trained: naïve bayes, neural networks, bagging with naïve bayes and random forest to compare accuracy of the prediction made by DL versus their models. They did this across ranges of overs, and found that for all but 20-30 overs bowled, their neural network model was the most accurate, with a consistent improvement over DL (Abbas & Haider, 2019).

Also, in *Cricket Score Forecasting using Neural Networks* a neural network was also chosen as the best model compared to a random forest and XGBoost as the others are convenient and readable but due to the high cardinality of some of the categorical features and lack of understanding of context that the other models have, a neural network was more accurate (Gupta, Joshi, Tahlan, Gupta, & Agarwal, 2021).

Improving Duckworth Lewis Method by using machine learning only showed an improvement of 2% over DL using a neural network but 13% and 28% for the support vector machine and random forest. Also trained were a logistic regression model and decision tree, but logistic regression had reduced accuracy compared to DL and the decision tree was outperformed by the random forest classifier (Jaipal, 2017).

1.8.4 Summary

Despite the criticisms mentioned above, there seems to be no consistent answer as to how best to improve upon or replace DL. When compared with previous methods used to reset target scores, as well as more novel methods, even as recently as July 2023 DL still comes out on top (Hazra, Chatterjee, Mandal, Sarkar, & Mandal, 2023) (Schall & Weatherall, 2013). However, if the DL method is built upon using additional resources (player value and ground scoring rate) and machine and deep learning methods are used, it can be improved upon.

2. Data

This section explains which data was collected and prepared and how.

2.1 New "resources"

In this project the definition of resources is extended using data on: ODI matches with a result obtained from 2002 – 2022 that didn't use DL to decide the result, player types and average scores at cricket venues. This addresses the criticism of DL that more emphasis is placed on wickets and reduces the likelihood of a team manipulating the target scores as there are too many variables to control. DL has also been criticised for being too complex, and extending the resources would seem to exacerbate this. For some alternate methods updated DL tables are suggested (Bhattacharya, Gill, & Swartz, 2011) but given the dimensionality of the data I've used, this isn't a viable option. However, personal technology has changed immensely since 1998, with 9 out of 10 adults in the UK owning a smartphone in 2019 (Deloitte, 2019) and many sites online have their own DL calculators to speed up target resetting so I would imagine this could be set up as an app or a similar calculator.

To assess the accuracy of the models a prediction is made as to the winner of the match for every delivery in the 2nd innings from the 20th over onwards. This is then compared to the prediction DL would have made, and to the real result of the match.

Throughout this project the terms predictor variables and target variables are used. In this case the predictor variables are the resources, whereas the target variables are the results of the match, with each result being called a class.

2.2 Data Collection

A mixture of structured and semi-structured data was obtained for use in this project. Typical structured data is highly organised, following a standard order, often in a database format. Whereas semi-structured has elements that make it easier to organise but won't necessarily be in a table or database (Sarker, 2021).

2.2.1 Match data

This was collected from 2346 semi-structured JSON files with ball-by-ball match data for men's ODIs played from 1st January 2002 to 31st December 2022 downloaded from cricsheet.org. This is freely-available detailed data per ball delivered and a full list of types of data provided can be viewed here. (Cricsheet, n.d.)

Only men's matches were chosen initially as for the same period there were only c. 350 women's matches and due to the different scoring rates of men's matches versus women's I felt the results would be skewed. However, in 2018 scoring patterns between men's and women's international matches were analysed to update the DLS system and found to have essentially identical wicket-adjusted resource utilisation rates. Due to the large scale of the dataset already obtained the women's data wasn't added in but in further research this would be explored alongside the T20 data (ICC, 2018).

2.2.2 Player data

This was collected from ESPN cricketers data from Kaggle, which is structured data in a tabular format. This contains data scraped from the ESPNcricinfo website on c.90,310 professional cricketers including columns on their playing role. (Gurung, 2018)

Some cricketers in the ODI matches were missing from the above data or their playing role wasn't clear, so the Uncover Cricket Legends: Cricketer's WikiData from Kaggle of 29,924 JSON files containing the infobox data from Wikipedia on professional cricketers was also collected. The infobox has a specific field for playing role, which meant that this data was very helpful for identifying those players who had mixed or missing information on this from the ESPN cricketers data (Spyz, 2023).

2.2.3 Field data

Average runs per over, runs per wicket statistics and ground aliases were collected from howstat.com for 218 grounds, copied with permission from the howstat webmaster (howstat, n.d.)

2.2.4 Duckworth Lewis

The DL standard edition tables (updated 2002) as seen on the ICC website. (ICC, 2002)

2.3 Data Preparation

The challenges in preparing the data for this project were that data was from a variety of sources, which meant that there were varied formats (JSONs, CSVs, tables on web pages), as well as no standardisation across the data e.g. cricketer names in different formats. There was also missing data, particularly in the cricketer data where role was often omitted and there was a lot of data that needed to be created from other data such as the Innings, over, ball number, running total of runs etc. This could sometimes be created from one source, but it was also necessary to create data from combinations from the different sources e.g. DL par score (Tie Score) was created using running total from the match JSONs and the DL table.

Code for this can be found under 1.Data.ipynb on the github for this project.

2.3.1 Match data

JSONs are comparable to a large dictionary of dictionaries, so if each key from the dictionary were used as a different column there would have been 2231 columns of data. Instead used the JSON normalise function to a depth of 1 (i.e the keys from the first layer of dictionary) to give 22 columns which made it easier to see where the relevant information was.

	innings	meta.data_version	meta.created	meta.revision	info.balls_per_over	info.city	info.dates	info.event	info.gender	info.match_type	...	info.player_of_match
0	[{'team': 'Australia', 'overs': [{'over': 0, '...	1.0.0	2017-01-14	2	6	Brisbane	[2017-01-13]	{'match_number': 1, 'name': 'Pakistan in Austr...	male	ODI	...	[MS Wade]
0	[{'team': 'Australia', 'overs': [{'over': 0, '...	1.0.0	2017-01-18	1	6	NaN	[2017-01-15]	{'match_number': 2, 'name': 'Pakistan in Austr...	male	ODI	...	[Mohammad Hafeez]
0	[{'team': 'Pakistan', 'overs': [{'over': 0, 'd...	1.0.0	2017-01-21	1	6	Perth	[2017-01-19]	{'match_number': 3, 'name': 'Pakistan in Austr...	male	ODI	...	[SPD Smith]

Figure 4. Screenshot of normalised JSON

Data was then removed if it added no information i.e. if any column had the same data in every row, or if it was irrelevant to the project. It was also filtered to remove any matches prior to 1st January 2002 and post 31st December 2022, leaving 2227 matches. Finally, 1st innings data and beginning of 2nd innings (first 20 overs) were dropped.

The JSONs also contained a field with a list of the information missing but none of the missing information was pertinent to this project.

Extracted the information from the remaining columns and created new columns of information for an overall match data dataframe. i.e. Each row is 1 ball delivery with information, new columns included remaining overs, balls etc as that information didn't exist in the JSONs.

	batting_team	bowling_team	innings_num	over_col	ball_col	Match ID	Start Date	Runs	Running Total	Batter out	...	Start Team	Remaining Team	Venue	Winner	First Team Innings	Remaining Overs	Remaining Balls	Remainder	Extras	Over and Balls Total
0	Australia	Pakistan	1	0	1	1000887	2017-01-13	0	0	NW	...	[DA Warner, TM Head, SPD Smith, CA Lynn, MR Ma...	[DA Warner, TM Head, SPD Smith, CA Lynn, MR Ma...	[Brisbane Cricket Ground, Woolloongabba]	Australia	NaN	49	5	49.5	N/E	NaN
1	Australia	Pakistan	1	0	2	1000887	2017-01-13	0	0	NW	...	[DA Warner, TM Head, SPD Smith, CA Lynn, MR Ma...	[DA Warner, TM Head, SPD Smith, CA Lynn, MR Ma...	[Brisbane Cricket Ground, Woolloongabba]	Australia	NaN	49	4	49.4	N/E	NaN
2	Australia	Pakistan	1	0	3	1000887	2017-01-13	0	0	NW	...	[DA Warner, TM Head, SPD Smith, CA Lynn, MR Ma...	[DA Warner, TM Head, SPD Smith, CA Lynn, MR Ma...	[Brisbane Cricket Ground, Woolloongabba]	Australia	NaN	49	3	49.3	N/E	NaN
3	Australia	Pakistan	1	0	4	1000887	2017-01-13	0	0	NW	...	[DA Warner, TM Head, SPD Smith, CA Lynn,	[DA Warner, TM Head, SPD Smith, CA Lynn,	[Brisbane Cricket Ground,	Australia	NaN	49	2	49.2	N/E	NaN

Figure 5. Screenshot of match_df dataframe

2.3.2 Cricketer data

Each cricketer who played in the relevant matches was given a value assigned to their playing role. For example: Ab de Villiers is a batter and this is assigned a value of 30. The challenge here was the format of the names in the match data as some have full names e.g. Azhar Ali and others are in the form of initials and surname e.g. PL Mommsen as well as missing or incorrect rows.

The name and role data from the ESPN CSV and Wikipedia JSONs were read in as dataframes (for ESPN both 'full name' and 'name' were included in case of aliases) and any missing data dropped because there were too many records to complete. For the Wikipedia JSONs only 8172 remained, but this isn't a problem as it was used primarily to bolster the larger ESPN data, and it can be assumed that the international cricketers would be more likely to have up-to-date Wikipedia pages and therefore the ones with missing information would be irrelevant to the project.

Both sources were then joined, duplicates removed and any players with irrelevant roles dropped e.g. umpire. Each playing type was then assigned a value: Batting 40, All-rounder or wicket keeper 30, Bowler 20 based on the rough ODI batting average for each role (Batting average (cricket), n.d.)

This left 33 rows without a player value assigned so these were manually amended by researching each one and either dropping the row if the player was irrelevant or adding their correct role.

A further column was added with full names converted to initial and surname so that either the initial and surname or full name version could be looked up.

```
1  cricketers_df[cricketers_df['Name'].str.contains("Codrington")]
```

	Name	Value
4342	A Codrington	20
4342	Austin Codrington	20

Figure 6. Example of cricketer name formats

A dictionary was created from the dataframe, with the keys being the full list of cricketers who played in the matches in the match data. This was then used by a program written using fuzzy principles that looked up the name of the player from the full cricket list, compared it to the dictionary of cricketers from the Wiki and ESPN data and created a new dictionary with their name and role "value."

Finally, this data was added as a column to the match data as well as columns for overall team value, average team value and the value of the team at the start of the match.

2.3.3 DL tables

The DL 2002 tables were read in as a string and turned into a dataframe. Wrote code to go through each line of the match data table, taking remaining overs and wickets and looking up the dl percentage (see figure 7.) In accordance with the DL method, calculated target score by taking the remaining dl percentage away from 100 for the percentage of resources used and multiplied it by Team 1's score, rounded down and added 1. Added DL percentage and tie score in as columns of the main match data.

```
dl_percentage_list = []
tie_list = []
count = 0
for delivery_index, delivery_row in match_df.iterrows():
    remaining_overs = delivery_row['Remainder']
    wickets_lost = str(delivery_row['Wickets taken'])
    row_index = np.where(dl_df['Overs Left'] == remaining_overs)[0]
    row_index = row_index[0]
    tie_percentage = dl_df.loc[row_index][wickets_lost]
    tie_percentage = 100-float(tie_percentage)
    dl_percentage_list.append(tie_percentage)
    #int rounds down by default
    tie_score = int(tie_percentage/100*delivery_row['First Team Innings'])
    tie_list.append(tie_score)

dl_percentage_df = pd.DataFrame(dl_percentage_list, columns=['DL %'])
tie_df = pd.DataFrame(tie_list, columns=['Tie Score'])
```

Figure 7. Code to calculate dl percentage.

2.3.4 Grounds data

Unable to scrape the data so manually copied the tables to a CSV file to read in. Included tables on each country's ODI cricket grounds with runs per over and runs per wicket. Also copied a table for the "aliases" of the cricket grounds as the names of the grounds are not standardised and were sometimes different in match data. Merged the original grounds list with the alias list on the official name of the ground. Manually added in any missing data for RPO or RPW and any missing grounds.

2.4 Train, validation, test split

Once the data was cleaned and preprocessed, and features such as the DL percentage were engineered, the data was split into three sets. This is done early to avoid "data snooping" bias, which is where you start spotting patterns in the test data yourself when looking at the data which affect decision making.

For the below split, sampling without replacement was used to ensure all the data was used and that validation and test data would be new to the models in order to test how well they generalise.

- Train – For training the model.
- Validation – For comparing the accuracy or loss of models using different value hyperparameters to know which model is best to choose.
- Test – For overall assessment of accuracy.

Where the dataset isn't as large as in this project (277819 rows x 7 columns) it's also possible to use techniques such as cross-validation instead of using a specific validation set. (Geron, 2019) For example: k-fold cross validation. This is a method where the dataset is split into k smaller sets, one set is kept as a "holdout." The model is trained on the remaining k-1 sets and evaluated on the test sets. This is repeated and so that each of the k sets is used as a holdout once. This gives the model a robustness because it has used the whole set to train, but also significantly increases the hyperparameter tuning time due to the additional number of models trained (Brownlee, A Gentle Introduction to k-fold Cross-Validation, 2023). However, given the computational cost of training some of the larger models in this project with a large amount of data, a validation set seemed most appropriate.

It's also possible to split the data randomly, setting a seed so that it's reproducible or saving the sets in a variable to prevent them from changing when running the code again. However, if the dataset is then updated, this will need to be run again. Instead the code on page 55 of Aurelion Geron's book *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow* was adapted to include a validation set. This method splits the code using a hash of each row's unique ID (Geron, 2019).

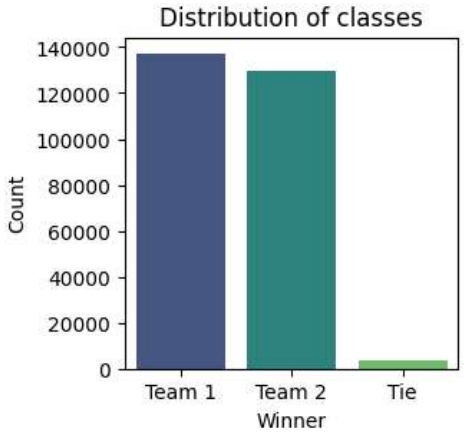
```
def is_id_in_set(identifier, set_ratio, hash_function):
    return hash_function(np.int64(identifier)) < set_ratio * 2**32

def split_data_with_id_hash(data, test_ratio, validation_ratio, id_column, random_state=None):
    np.random.seed(random_state)  # For reproducibility

    # Different hash functions for test and validation sets for indepdendence
    hash_for_test = lambda x: crc32(x) & 0xffffffff
    hash_for_validation = lambda x: crc32(x + 1) & 0xffffffff

    ids = data[id_column]

    in_test_set = ids.apply(lambda id_: is_id_in_set(id_, test_ratio, hash_for_test))
    in_validation_set = ids.apply(lambda id_: is_id_in_set(id_, validation_ratio, hash_for_validation))

    # Combine the conditions to get the training set
    in_train_set = ~in_test_set & ~in_validation_set

    return data.loc[in_train_set], data.loc[in_validation_set], data.loc[in_test_set]

train_set, validation_set, test_set = split_data_with_id_hash(data_df, test_ratio=0.1, validation_ratio=0.2, id_column='Row ID', random_state=7)
```

Figure 8. Data split code

The data is split in the ratio train 70: validation 20: test 10. Typically, it's suggested that this should be in the range 60-80: 10-20: 10-20, often with the lower % being better suited to a larger dataset (Acharya, 2023). However, with more time to work on the project this would have been explored further as this may be dependent on the classification problem itself, and data used.

2.5 Imbalanced data

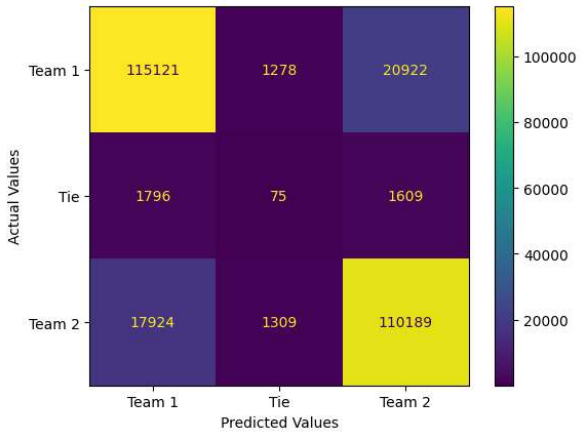
In the data collected only 43 ODIs ended in a tied score. Therefore, the amount of data in this category is extremely low (see below.) If the data were used as it is to train the models, then when making predictions they would be biased towards predicting Team 1 or Team 2.



	Winner	Percentage
0	Team 1	0.508177
1	Team 2	0.478945
2	Tie	0.0128783

Figure 9. Bar chart of class distribution, Figure 10. Table of class distribution %

This can also be seen if you compare the prediction of the winner made by DLS vs the real results of the matches:



	precision	recall	f1-score	support
0	0.85	0.84	0.85	137321
1	0.03	0.02	0.02	3480
2	0.83	0.85	0.84	129422
accuracy			0.83	270223
macro avg	0.57	0.57	0.57	270223
weighted avg	0.83	0.83	0.83	270223

Figure 11. Confusion matrix of DL predictions vs match outcomes, Figure 12. Performance metrics of DL predictive ability

On average 83% of the time the DL prediction is correct, but the precision and recall of 1, i.e. the “Tie” category is only 2 or 3%. This isn’t a criticism of the method as DL isn’t used specifically for prediction, but worth noting as a baseline for improvement with the new models.

To address the imbalance of the dataset, SMOTE methods were applied (Chalwa, Bowyer, Hall, & Kegelmeyer, 2002). SMOTE is an oversampling technique where additional samples from the minority class are created so that the dataset is better balanced when training. Unlike random oversampling SMOTE is more methodical; it initially chooses a random instance of the minority class, then finds k nearest neighbours from the minority class. It then interpolates another sample between one of the k nearest neighbours and the original sample (Brownlee, SMOTE for Imbalanced Classification with Python, 2021).NB. SMOTE is only used on the training set and not the validation or test set.

2.6 Powerplays

Fielding restrictions (renamed Powerplays in 2005) have been used in ODI matches since 1992 and have always been a variation on a rule restricting the number of fielding players who can stand outside the 30-yard line.

These rules were introduced to allow for higher scoring matches as the batters could hit the ball further with a lower risk of being caught. Initially it was thought that data on these restrictions would be a good addition to the batters' resources, however, the change in rules for powerplays over the years is complex and "we sometimes found it difficult to pin down details regarding the history of the powerplay" (Manage, Silva, & Swartz, 2015).

From research the following was found:

The only "mandatory" powerplays in each match from 2002 – 2015 were either the first 15 or first 10 overs. Given an ODI match cannot have a result unless both teams play at least 20 overs, this means that both teams would have had their mandatory powerplay outside of the scope of the data used in this project.

After 2015, the matches themselves were split into three "powerplay periods" with the first 10 overs mirroring the mandatory powerplays of the past, (max two fielders outside the 30-yard circle), and so again would be outside the remit of the data. The subsequent two periods restrict the number of fielders outside the circle to four and five respectively, but this was also the case for matches from 1992-2005 where after the mandatory powerplay only five fielders were able to stand outside the 30-yard line.

From 2005 – 2011 there were two powerplays of 5 overs each but the timing of these were chosen by the fielding team, and often they would choose to play them straight after the initial 10 over mandatory powerplay, again leaving all 20 overs of the powerplay outside of the remit of my data. This rule was updated in 2008 to allow the batting team to choose the timing of one of the powerplays but in 2011 the rule was amended to ensure that the two powerplays had to be played between the 16^{th} and 36^{th} over, again restricting their timings. (Powerplay rules in ODI cricket: History and application, 2022).

In *A Study of the Powerplay in One-Day Cricket* which analysed the powerplays in 597 matches between 2005 and 2013, it was found that: Different rules on powerplays over time had different effects, there was an advantage to the batting side but that that was to the tune of 6.5 runs on average and more wickets were taken in powerplay periods.

It is therefore very difficult to apply a method which would consistently identify the benefit to the batting team for the time period of the match data (2002 – 2022) and under DL the gain in runs would likely be outweighed by the risk of the loss of wicket, given the emphasis placed on wickets as mentioned in papers above (Shah, Sampat, Savla, & Bhowmick, 2015).

3. Models

3.1 Model choice

The problem at hand is a supervised one, because supervised learning "uses labelled training data and a collection of training examples to infer a function." (Sarker, 2021) And in this case the data is labelled with the outcome of the matches. It is also a classification problem, which is a "problem of predictive modelling… where a class label is predicted for a given example." I.e. For ODI matches where a result was obtained, there are three classes of outcome – Team 1 win, Team 2 win, Tie. Given there are three possible outcomes this is specifically a multiclass classification, not binary (Sarker, 2021). It's discussed later in this report whether that reflects the reality of the situation in ODIs or whether a Tie is really an absence of a class.

To decide which models to use to improve upon the predictive capability of DL, the below papers on supervised DL methods were considered to see which had previously performed the best and any limitations of these methods.

Models	Summary	Considerations
VJD and DLSMA	Tweak the functional form of the DL curves	Could reverse engineer parameters for the DL and try to improve formula. However, this is more statistical than data focused
PRCICM	Adds "values" of players based on previous performance as a resource	Improves predictive ability of DL-like model but an interesting part of the game is where the "underdog" team wins and this would reduce this.
Predicting the Match Outcome in One Day International Cricket Matches, While the Game is in Progress	Adds "home ground advantage, past performances, match experience, performance at the specific venue, performance against the specific opposition, experience at the specific venue and current form" to resources	As above - improves predictive ability but for a DL-like model cannot essentially predict that the better team will win. However, the average runs per over and wicket for each ground could be considered.
Rain rules for limited overs cricket and probabilities of victory	Values resources in terms of probability of winning instead of runs and wickets	Shown to be inconsistent, and effectively ignores the overs that aren't played, starting the match again with the same probabilities. A DL-like method aims to recreate what those overs would look like in order to better model a full match
Duckworth-Lewis-Stern Method Comparison with Machine Learning Approach	Trained naïve bayes, neural networks, bagging with naïve bayes and random forest	Neural Network most consistently beat DL with predictions from 40th over being 89% accurate
Cricket Forecasting using Neural Networks	Compared neural network, random forest and XGBoost but decided on neural network because of high cardinality of some of the categorical features and lack of context	Regression method. High cardinality a consideration depending on the data used. Trained an LSTM neural network, treats the data as a time series
Improving Duckworth Lewis Method by using machine learning	Trained SVM, Random Forest, Neural Network, Logistic Regression, Decision Tree	Had the best success with Random Forest (95% accuracy), SVM (88%) and logistic regression (88%) with Decision Tree (87%) close behind.
An Analysis of Duckworth-Lewis-Stern Method in the Context of Interrupted Limited over Cricket Matches, AND *Accuracy and fairness of rain rules for interrupted one-day cricket matches.*	Compares ARR, PARAB, WC1996, CLARK, MPO and DMPO to DL.	Found DL to be the most consistently fair and, with the DL tables, simple to understand how to apply the rule even if the maths is challenging.
Duckworth-Lewis and Twenty20 Cricket, AND *An Analysis of the Duckworth Lewis Method in the context of English County Cricket*	Used/ favoured the use of non-parametric methods for a DL-like model	DL is parametric and has been shown to be the best current option for resetting targets in interrupted matches, but it makes sense to allow the shape of the model to be dictated by the data itself

Figure 13. Summary table of papers on DL

Ultimately A Random Forest (RF), Support Vector Machine (SVM) and simple feedforward neural network (FFN) were chosen as they are all suitable for classification problems, are non-parametric methods utilising data, and were shown in multiple papers above to improve upon the predictive capability of DL.

The three methods are critically compared below:

Criteria	**Comparison**
Performance	Comparable accuracy for all three but NN has higher potential (Roßbach, 2018). "when the data set contains more noise, such as overlapping target classes, SVM does not perform well." (Sarker, 2021)
Robustness	All three can overfit to a dataset, however with careful choice of hyperparameters you can avoid this. (Cawley & Talbot, 2007). SVMs use quadratic programming, which provide global minima only. This is different from Neural Networks where it's possible to think you've found the best model with the lowest loss, but it's stuck in a local minima. (Pal, 2008)
Interpretability	NN and SVMs are black box models so not easy to interpret. RF are more interpretable but also pose a challenge but there are methods you can use to identify the most representative trees in the ensemble. (Banerjee, Ding, & Noone, 2012)
Cost and time expenditure	NNs take a lot of time and have a high computational cost. RFs need far less input preparation, are quick to train and optimise. SVMs are also computationally costly but less so than NNs. (Roßbach, 2018)

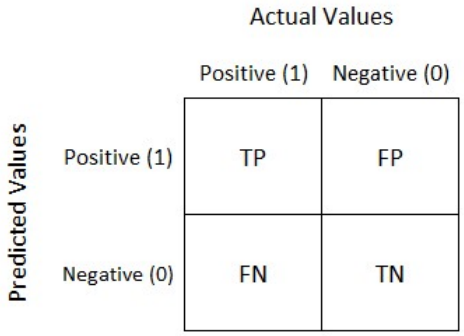
Figure 14. Table comparing RF, SVM and NN

3.2 Performance metrics

These are used to assess whether the new models are an improvement over DL, as well as which hyperparameters to choose.

For binary classification we can place the predictions the model has made into one of four categories (*True Positive (TP), True Negative(TN), False Positive(FN) and False Negative(FN)*.) Assuming that Positive is 1 and Negative is 0, True Positive is how many samples are correctly classified as 1, and True Negative is how many samples are correctly classified as 0.

False Positive on the other hand would be how many samples of class 1 were identified as 0, and the reverse for False Negative. This can be visualised in a Confusion Matrix (previously known as a Contingency Table) (Pearson, 1904):



(Narkhede, 2018)

Figure15. Illustrative diagram of a confusion matrix

The distinctions above can be very important depending on different types of classification problem, e.g. If a classification model were used as a hotel system, where positive meant that a room was occupied, and negative meant it was empty, if the model was very accurate overall but had a high number of False Positives then we'd risk double booking customers. If another model were as accurate but had a higher number of False Negatives, this may lose money but wouldn't be as disastrous.

We also consider four more metrics: Accuracy, recall, precision and F1 score which are defined as ratios of the above.

Accuracy

The proportion of samples has the model correctly predicted e.g. if out of 10 test samples 8 of them were correctly predicted then accuracy would be 0.8.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Recall

How many of positive samples were correctly identified.

$$Recall = \frac{TP}{(TP + FN)}$$

Precision

How many of the positive predictions are correctly identified.

$$Precision = \frac{TP}{(TP + FP)}$$

F1-Score

This combines Recall and Precision using harmonic mean.

$$F1\ Score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

3.2.1 Multiclass classification

For this project the situation is a little more complex as there are three categories: Team 1 wins (0), Tie(1) and Team 2 wins (2), but the above still applies.

In this project sklearn.metrics classification_report is used to give the Precision, Recall and F1-Scores for each class, alongside the "support" i.e. how many samples in each class. See example below from sklearn documentation:

	precision	recall	f1-score	support
class 0	0.50	1.00	0.67	1
class 1	0.00	0.00	0.00	1
class 2	1.00	0.67	0.80	3
accuracy			0.60	5
macro avg	0.50	0.56	0.49	5
weighted avg	0.70	0.60	0.61	5

Figure 16. Example of performance metrics table

It also provides the macro-average and micro-average for each metric. The macro-average is the sum of the chosen metric for each class, divided by the total number of classes.

$$Macro\ average\ Precision = \frac{(Precision\ class\ 0 + Precision\ class\ 1 + Precision\ class\ 2)}{3}$$

$$Macro\ average\ Recall = \frac{(Recall\ class\ 0 + Recall\ class\ 1 + Recall\ class\ 2)}{3}$$

Whereas for the micro-average, precision and recall are calculated the same way as for binary classification except TP will be the sum of TP class 0 + TP class 1 + TP class 2. This is the same for FP and FN. For example:

$$Macro\ average\ Recall = \frac{(TP\ class\ 0 + TP\ class\ 1 + TP\ class\ 2)}{(TP\ class\ 0 + FN\ class\ 0 + TP\ class\ 1 + FN\ class\ 1 + TP\ class\ 2 + FN\ class\ 2)}$$

(Evidently AI, n.d.)

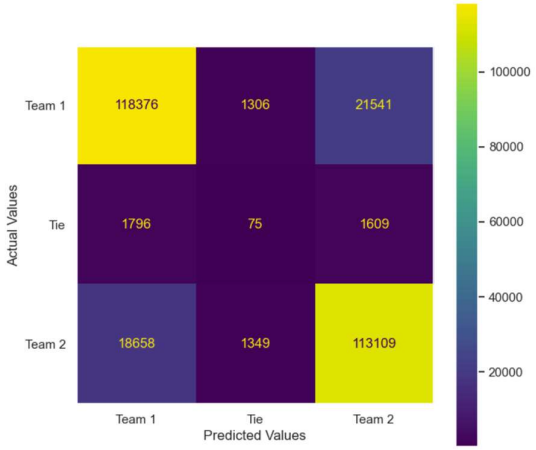


Figure 17. Example of a 3-class confusion matrix

3.3 Bias Variance Trade-off, Learning curves and Cost functions

When training a model it is key that its generalisation is considered, i.e. how well it predicts given new unseen data from the same dataset. Key to finding the model with the best generalisation are the concepts of bias and variance.

Bias is the error introduced when modelling a real-life scenario using a model which is likely to be much simpler than the reality. High bias means the model makes a lot of assumptions about the data. A high bias is also known as underfitting, as the model's assumptions mean the model doesn't fit close enough to the data to make good predictions, also known as good generalisation. In real life this can be seen where the model has low training accuracy.

Variance is a measure of dispersion, i.e. how far a set of numbers is spread from the average. This can be seen as error introduced when a model is too sensitive to noise in training data. High variance means the model is sensitive to small changes in the training data. High variance is also known as overfitting, as the model's sensitivity means it can learn the noise or randomness in the data, which prevents it from making good predictions. In real life this can be seen where the model has high training accuracy but low test or validation accuracy.

Below is a diagram showing both the bias and variance for increasingly complex models, alongside the total error. The aim when training a model is find the point at which error is lowest, i.e. Given the lowest point for bias is the highest for variance and vice versa, we find a compromise where they are simultaneously at their lowest. (Fortmann-Roe, Understanding the Bias Variance Tradeoff, 2012)

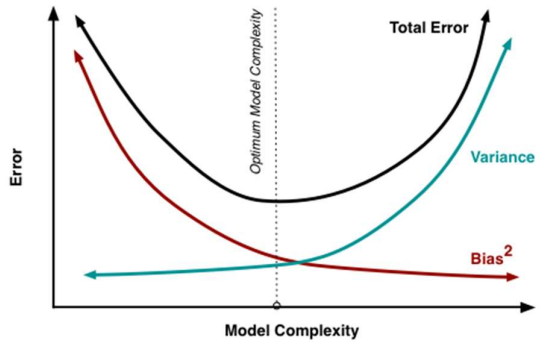


Figure 18. Bias Variance graph (Fortmann-Roe, Understanding the Bias Variance Tradeoff, 2012)

When tuning the hyperparameters for the models below I have used illustrative learning curves to show the validation loss and accuracy vs changing hyperparameters (see figure 19.) This helps to choose the best hyperparameter for the model to prevent overfitting and limit computational cost. All three of the models chosen are prone to overfitting (Cawley & Talbot, 2007), and due to the large dataset used there's also a risk of high computational cost.

Therefore, for some of the models it was better to choose a hyperparameter that didn't have the absolute lowest loss/ highest accuracy but was a good generalisation at a reasonable computational cost.

NB. As the data is imbalanced, but this balance is only addressed in the training set, not the validation or test set, there will be a gap between the training accuracy and validation/ test accuracy.

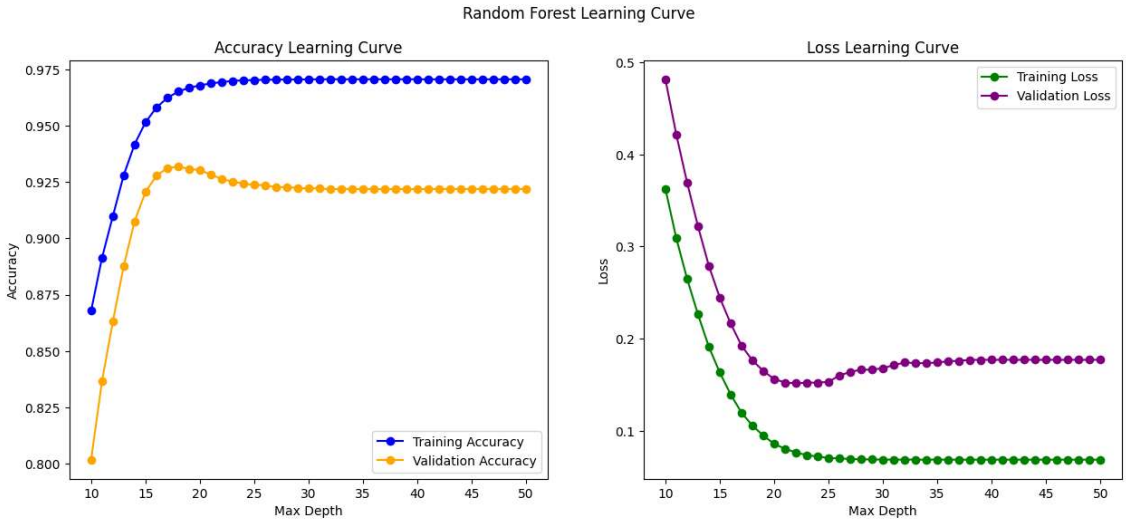


Figure 19. Example Accuracy and Loss learning curves

3.3.1 Cost/Loss Function

When training a model with supervised data we aim to minimise error between predictor variables and the target variables.

How we calculate that error is dependent on the cost function chosen. For each model trained below the cost function and justification for using that one is listed. Cost function is also referred to as loss function, and it is more accurate to say that loss function is for one training sample whereas the cost function gives average loss over the training set. However, the in-built terminology in Python refers to this as loss so where the code is shown, this will be called loss instead.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Figure 20. Example code showing loss term

3.4 Random Forest (RF)

Random Forest is an ensemble method of the decision tree; it fits multiple decision trees using smaller samples of the training data and a random number of the features, and then, in the case of classification, takes the majority vote of the trees as to the class for the outcome. A decision tree uses a hierarchical structure and will split on the features of data.

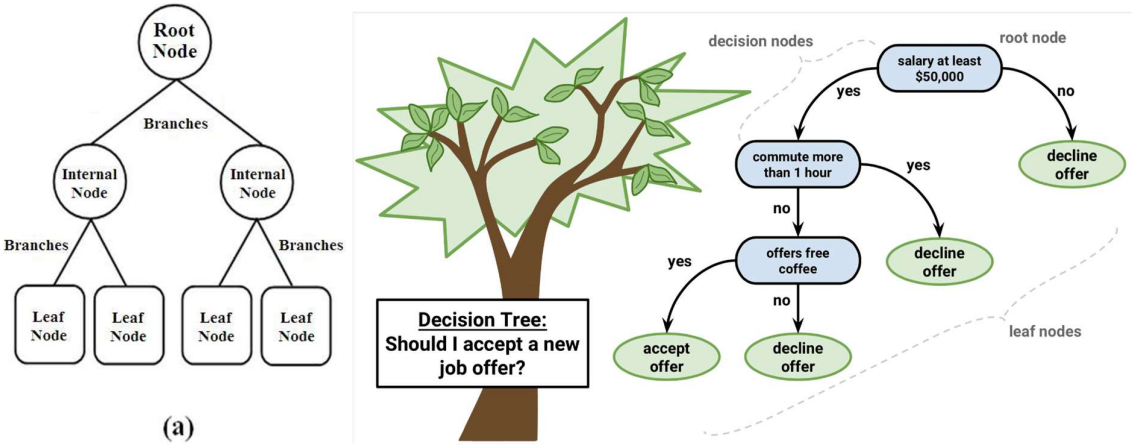


Figure 21. Labelled Decision Tree (Shah U. , 2010), Figure 22 Example Decision Tree (Thevapalan & Le, 2023)

How the tree is split depends on the algorithm used, e.g. ID3 which uses entropy theory (Quinlan, J;, 1986), CART which uses gini impurity (Breiman, Friedman, Olshen, & Stone, 1984). These splitting methods typically look at how homogeneous the data is once split on a feature, choosing to split on the feature that gives the class that's got the most similar datapoints.

The tree will continue to split until the criteria we choose is met, this can include options such as: The maximum depth of the tree or the number of data samples per leaf/ terminal node.

Decision trees therefore can become complex depending on the criteria chosen, and therefore can have high variance (i.e. The model fits too closely to the training data so that when new data such as test data is introduced it cannot generalise and predict well) but low bias.

However, when decision trees are used as part of a random forest, the variance (defined as the average of squared differences from the mean) diminishes as the forest grows. This is because of the collective effect of diversification; as the trees become less correlated (due to bootstrapped data and random feature selection) the overall variance declines.

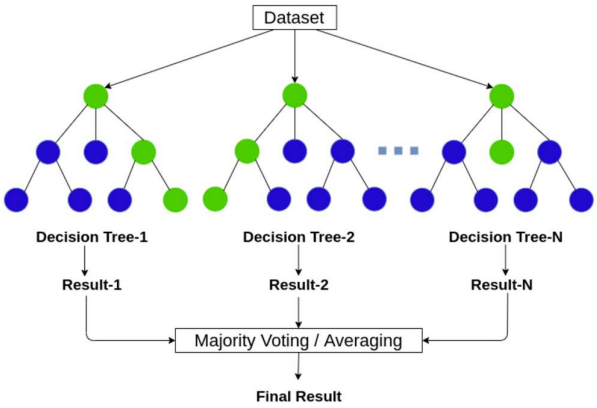


Figure 23. Diagram of a Random Forest model (Goyal, 2021)

3.4.1 Training the model

The hyperparameters of a model are variables that can be tuned to adapt how that model is trained. In the sklearn implementation of Random Forest Classifiers there are default values for these hyperparameters and the performance for the baseline Random Forest models in this case was very good (see 3.RF_model.ipynb on github.) And arguably these are the most suitable, as seen in *Influence of Hyperparameters on Random Forest Accuracy* where it was shown that the default option for max_features (the square root of the number of features) gave optimal accuracy (Bernard & Heutte, Influence of Hyperparameters on Random Forest Accuracy, 2009). However, *The parameter sensitivity of random forests* showed there's still "significant benefit to be gained by model tuning RFs away from their default parameter settings" as the tuning they performed enabled higher model performance across the board (Huang & Boutros, 2016).

As a Random Forest model is an ensemble method of decision trees, both the individual decision trees and the overall random forest have hyperparameters, which gives a lot of options to tune.

Below are some options for techniques on how to tune hyperparameters.

Method	Description	Limitations
Manual Search	Using trial and error (and sometimes domain knowledge) to train combinations of hyperparameters	Not thorough enough so risks missing best combinations
Grid Search	Creating a grid of combinations of ranges of hyperparameters to train the models	Very thorough but computationally expensive
Grid Search CV	As above but instead of using a holdout validation set to evaluate the models, uses cross validation.	Can multiply training time by number of times validation set is chosen so makes it more computationally expensive
Random Search CV	Chooses random combinations of the hyperparameters at random to train the models and evaluate using CV	Risk of missing good combinations through randomness

Figure 24. Table of hyperparameter tuning methods

For this Random Forest each hyperparameter was tuned sequentially in a variation of the manual search. A range of values were used for each hyperparameter in turn, evaluated using the validation set, computational cost and with reduction of overfitting in mind. Then when the best value for that hyperparameter was chosen, the next set of models was trained, focusing on another hyperparameter and incorporating the previous set's best value.

There is a risk with this approach of finding a good combination of the hyperparameters that isn't the best overall, however this was weighed up against the other options that were prohibitively computationally expensive. This method used alongside ANOVA to decide the order of training the hyperparameters (see below) was shown to be significantly faster (165 times in one study) than grid search (Zhu, Zhu, Zhou, Zhu, & Zhang, 2022).

The question therefore was how many of the hyperparameters should be trained, and in which order. There's limited literature on this as a Grid Search or Random Search method is typically preferred but below is a table of papers and webpages with their opinions on the most important hyperparameters to choose. Where the models were trained using R instead of

Python the equivalent hyperparameter has been listed too for easy comparison. The names given are how they are referred to in the sklearn documentation of Random Forest Classifier.

Source	Python	R
The parameter sensitivity of random forests (Huang & Boutros, 2016)	n_estimators max_features max_samples	ntree mtry samplesize
Hyperparameter tuning in random forests (Ellis, 2022)	n_estimators max_features max_depth	
Hyperparameters and Tuning Strategies for Random Forest (Boulesteix, Probst, & Wright, 2019)	n_estimators max_features max_samples min_samples_leaf criterion	ntree mtry samplesize replacement node size splitting rule
Optimising random forest hyperparameters (Vieira, 2023)	n_estimators max_features max_samples max_depth max_leaf_nodes criterion	
A Beginner's Guide to Random Forest Hyperparameter Tuning (Saxena, 2023)	n_estimators max_features max_samples max_depth min_samples_split max_leaf_nodes min_samples_leaf	

Figure 25. Table of hyperparameters tuned in different papers and websites

The most consistently mentioned hyperparameters were: n_estimators, max_features, max_samples, and max_depth. The below explains how these hyperparameters affect the model.

Hyperparameter name in Sklearn	**Purpose**
max_features	Maximum possible number of features the algorithm can choose from when growing each decision tree. This reduces correlation between the trees.
max_depth	Longest path between the root node and leaf node for each decision tree.
n_estimators	How many trees are grown.
max_samples	How much of the dataset each decision tree uses to train. Reduces training time.

Figure 26. Table of hyperparameters and purpose

Decision Trees are prone to overfitting so it makes sense that ¾ of the hyperparameters chosen are designed to reduce this risk by limiting the complexity of the model, which also reduces computational complexity. The last hyperparameter, n_estimators, also reduces this risk by reducing the variance, and therefore it would be tempting to making this value as large as possible. However as more models are created it increases computational cost, and in *How Many Trees in a Random Forest?* it was shown that when comparing AUC (area under the curve) whilst the best model had thousands of trees, there was only 1% difference when this was dropped to a few hundred (Baranauskas, Oshiro, & Perez, 2012).

We can see this in the learning curves for n_estimators for this project:

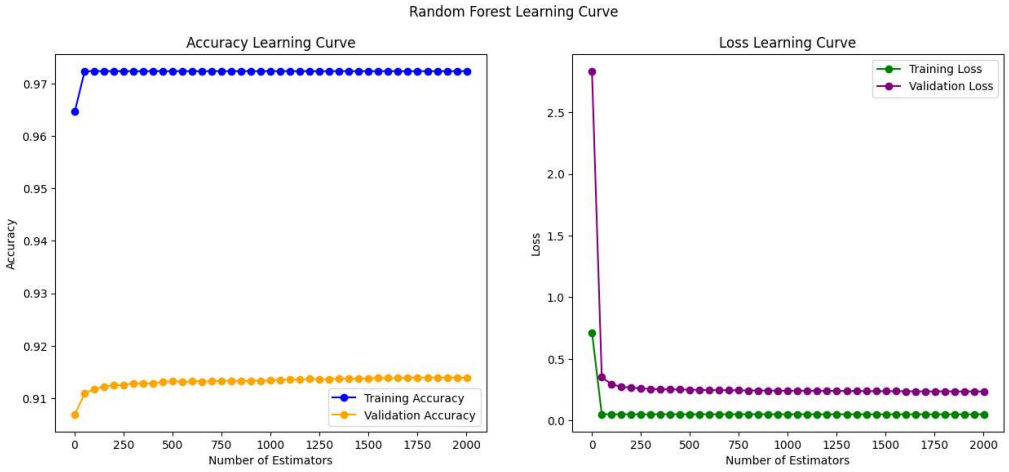


Figure 27. Example Accuracy learning curve n_estimators, Figure 28. Example Loss learning curve n_estimators

The above shows that as the number of estimators increases, the validation accuracy increases very slowly, and the validation loss decreases very slowly. If more estimators were used it's likely this would continue to improve, however it would take a very long time and as mentioned above, this would only improve the performance a very small amount.

The order of tuning the hyperparameters was decided using ANOVA (Analysis of Variance) as shown to be effective in An Efficient Approach for Assessing Hyperparameter Importance. (Hutter, Hoos, & Leyton-Brown, 2014). In this project ANOVA was used to show the variance change when different values of hyperparameters were chosen to see how important each hyperparameter change was to the model. The hyperparameters were then ordered from those which caused the largest variance change to the smallest, as below:

```
max_features: 179984926.41695178
n_estimators: 179612756.01284975
max_samples: 87856358.80683723
max_depth: 4120120.8915084153
```

Figure 29. Hyperparameter importance – ANOVA scores

Multi-class cross entropy (log loss in the code) is used as the cost function alongside accuracy score to plot learning curves. As mentioned earlier, these are used to choose the best value of the hyperparameter, where accuracy is high and loss is low but also the model isn't overfitting and the computational cost isn't prohibitive.

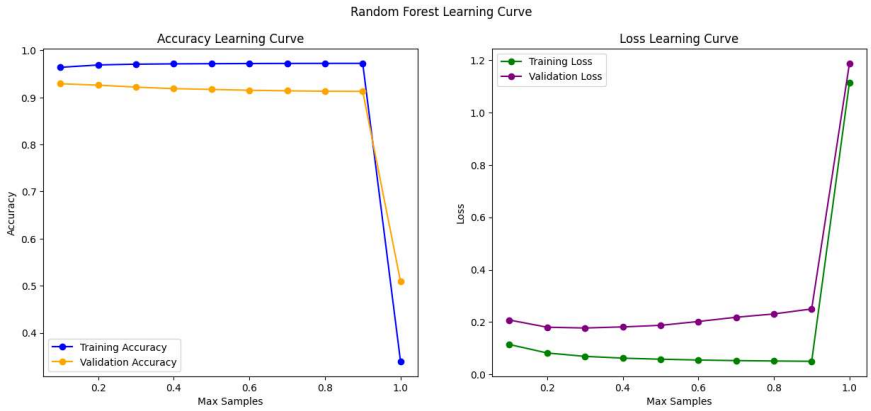


Figure 30. Accuracy learning curve max_samples, Figure 31. Loss learning curve max_samples

Validation and training accuracy are highest at 0.1 samples, however that means the random forests are training on a very small amount of the data and risks underfitting, so 0.3 was chosen instead. As this is where it starts to stabilise and where validation and training loss are lowest.

3.4.2 Overall design of RF

Design feature	**Choice**
n_estimators	192
max_features	3
max_samples	0.3
max_depth	18
Cost function	Crossentropy

Figure 32. Table of hyperparameter choices for RF

3.5 Feature scaling

Features in datasets may each be on very different scales (see below for histogram plots of the data in this project:

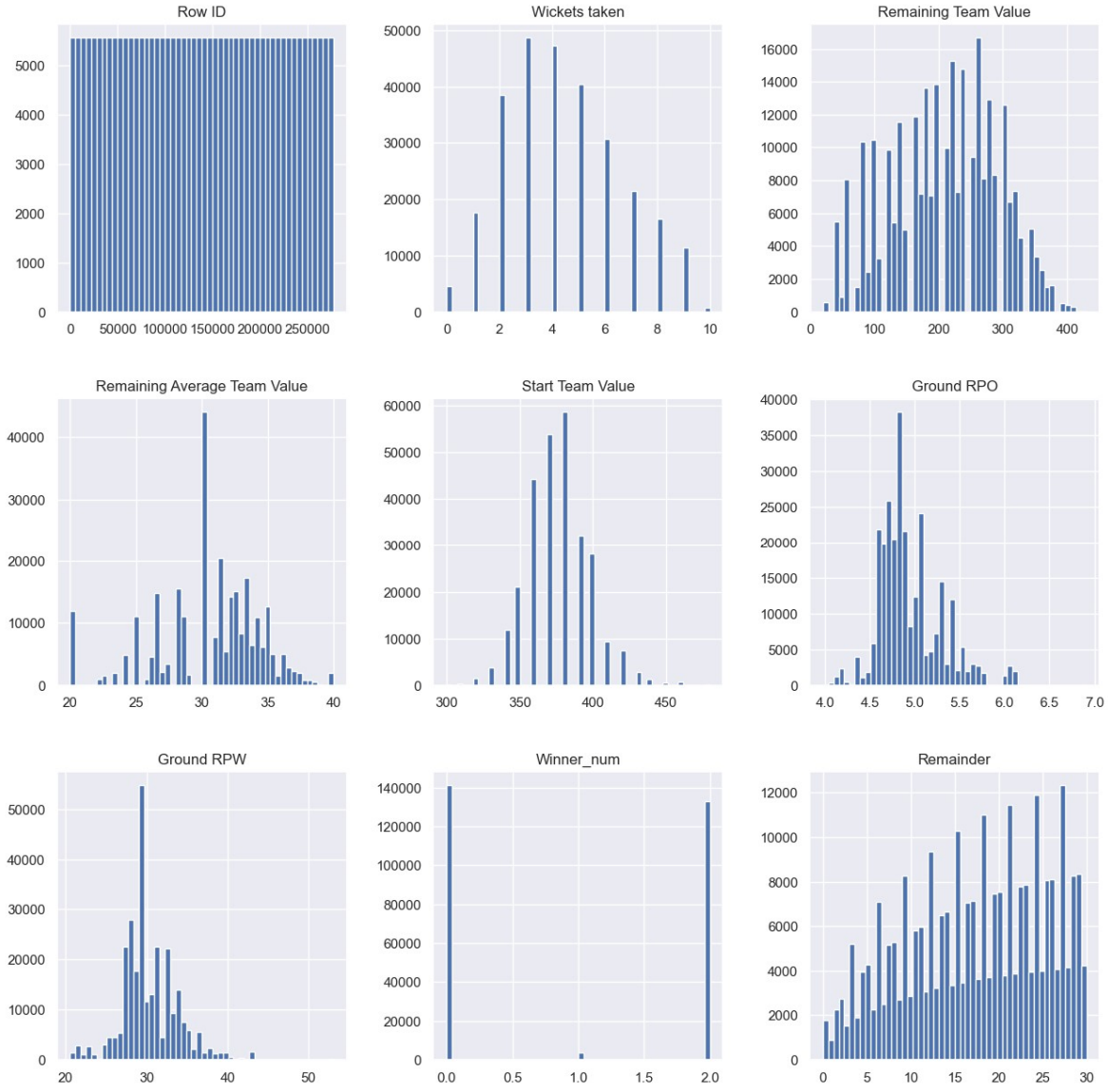


Figure 33. Histogram plots for data features

There is a risk when these features are on different scales that a model may learn that the one with larger values is more important. In the match data used the range of values goes from zero to a few hundred and therefore isn't a huge problem but the predictive accuracy did improve when scaling methods were used.

Features don't need to be scaled for Random Forest models as they are scale invariant (Mirjalili & Raschka, 2019). However, SVM models are sensitive to scaling as they choose a decision-boundaries based on distance. Neural Networks also perform better with feature

scaling as it improves the speed of the model convergence. Please see a snippet of code below for training a simple SVM using non-scaled data. This never completed as the model itself had taken 23 hours and counting to run:

```
#SMOTE data no scaling - took too long to run (was still running at nearly 24 hours for 1 model)

start_time = time.time()

model = SVC(kernel="linear", random_state=7)
model.fit(X_smin_train, y_smin_train)

model_train_score = model.score(X_smin_train, y_smin_train)
model_test_score = model.score(X_smin_test, y_smin_test)

end_time = time.time()
elapsed_time = end_time - start_time

print(f"Training score: {model_train_score}, Testing score: {model_test_score}, Time to train: {elapsed_time} seconds" )

y_pred = model.predict(X_smin_test)

cm = confusion_matrix(y_smin_test, y_pred, labels=model.classes_)
cmd = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels=labels)

fig, ax = plt.subplots(figsize=(7,7))
cmd.plot(ax=ax)
plt.grid(False)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()

print(classification_report(y_smin_test, y_pred, target_names=labels))
1395m 24.8s
```

Figure 34. Screenshot of code scrapped because of computational cost

There are two main types of scaling that were considered for this model:

- MinMax Scaler – values for each feature are rescaled and adjusted to end up in a pre-defined range. This is typically between 0 and 1 but can be adapted. MinMax scaling is calculated using the following: (Original value – minimum value)/ (maximum value – minimum value.) This can be helpful for models such as neural networks that work best with input values between 0 and 1.
- Standardisation – values are moved to give them a zero mean and unit variance: (Original value – mean value)/ standard deviation. This is more robust when the data has outliers because for MinMax the extreme values skew the scaling. (Geron, 2019)

SS and SMOTE

```
Training score: 0.5369614328696903, Testing score: 0.6011086314880139, Time to train: 2834.1844453811646 seconds
```

Figure 35. Time taken for SVM to train using StandardScaled and SMOTE data

MM and SMOTE

```
Training score: 0.5391969148997241, Testing score: 0.6042761500251962, Time to train: 4970.865523099899 seconds
```

Figure 36. Time taken for SVM to train using MinMax and SMOTE data

Python Machine Learning suggests that standardisation may be more generally practical for models such as SVM as it is easier for the model to learn weights i.e. the pattern in the data. This is because when training the SVM weights are initialised at 0 or close to, and as standardisation makes the values zero mean and unit variance these values will be closer. It also maintains useful information about outliers.

In practice when tuning the hyperparameters for the SVM I trained models first using standardisation and then using normalisation. There was a small difference in accuracy (the

standardised data performed better) and therefore given the above and the improvement I decided to use standardised data in training the final model. It was also significantly faster to train

```
Best accuracy is: 0.827933765298776 and best parameters are {'C': 200, 'gamma': 1, 'kernel': 'rbf'}
```

Figure 37. Best accuracy for SVM trained using StandardScaled and SMOTE data and hyperparameters.

```
Best accuracy is: 0.7847372210223182 and best parameters are {'C': 10, 'gamma': 100, 'kernel': 'rbf'}
```

Figure 38. Best accuracy for SVM trained using MinMax and SMOTE data and hyperparameters.

3.6 Feature Selection

Feature selection is where the number of features is reduced, which helps with computational cost, improve accuracy, interpretability and overfitting (Kaushik, 2023).

Once cleaned and oversampled the data for this project had 296,133 samples and 7 features. This isn't a huge number of features given the number of samples, but in line with the ethos of DL of being as simple and understandable as possible, as well as potentially improving accuracy two feature selection methods were used: Pearson Correlation and Random Forest.

For both techniques the features were used to reduce the dataset and then SVM and Neural Networks models were trained on both the original number of features and the new number of features to compare.

3.6.1 Pearson Correlation

Pearson correlation coefficients assigns a value to the relationship between two variables and ranges from -1 to 1. Where the correlation is positive it means that as the value of one of the variables increases, so does the other, and if negative it means that as the value of one goes up, the other goes down. If the correlation is 1 it means that that relationship is in a 1:1 ratio and if 0 then there's no relationship at all (Geron, 2019).

This can be used as a feature selection technique to remove duplication – e.g. Below the correlation coefficient between Wickets taken and Remaining Team Value is close to -1. This makes sense as when a wicket is taken, the Remaining Team Value is reduced as that player is removed.

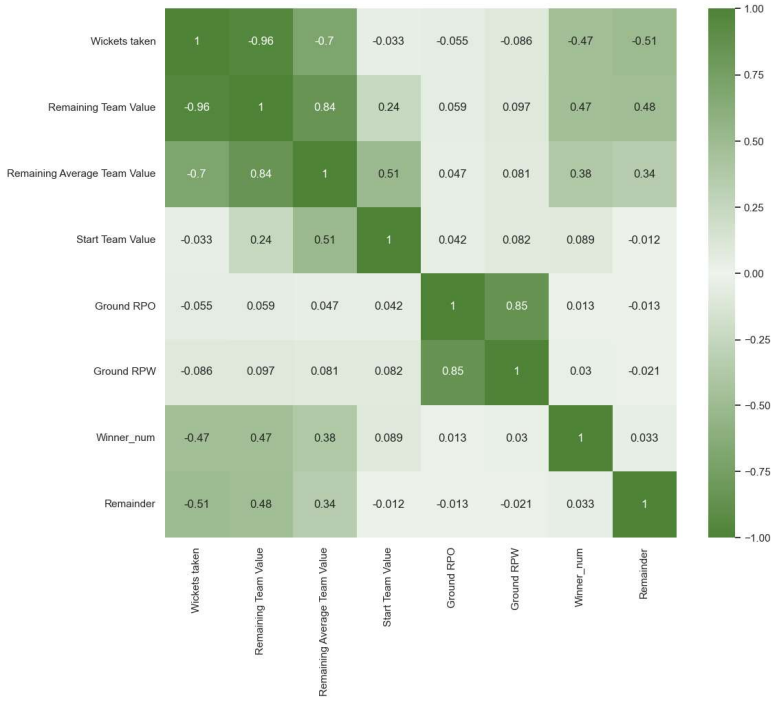


Figure 39. Pearson correlation matrix of the data features

Similarly, Ground RPO and Ground RPW have strong correlation as they are both measures of the speed with which runs can be scored at a specific cricket ground.

Based on the above I chose a threshold of 0.8 and removed: Remaining Team Value and Ground RPO.

3.6.2 Random Forest

Another method for feature selection is using a random forest. As seen above when choosing the criteria to split on when growing the decision trees, a random forest will consider a measure such as gini or entropy. This shows how "pure" or ordered the node will be if split on that criterion.

These measures of which criteria to use can also be used for feature importance and therefore selection. Below is a diagram of the importance of the 7 features used in the training set and their relative importance. As with the correlation measures you can see that two of the features are equally important (Ground RPW and Ground RPO) as well as (Remaining Average Team Value and Remaining Team Value).

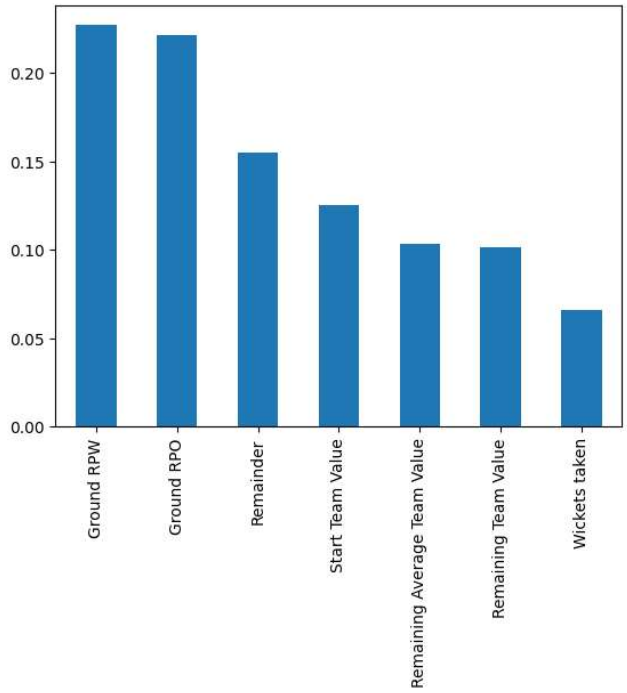


Figure 40. Feature importance diagram

3.7 Support Vector Machine (SVM)

SVMs are models which look to find the line or hyperplane that best separates the classes of data by projecting the data into additional dimensions. The hyperplane chosen is the one with the largest distance from the nearest training data in any class, i.e. largest margin. The greater the margin the lower the generalisation error (Sarker, 2021).

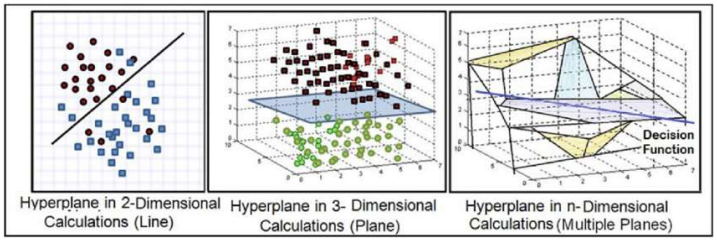


Figure 41. Diagram of a hyperplane in increasing dimensions (Datta & Singh, 2021)

SVM is designed for two-class-classification problems, however there are methods to use it for multi-class classification, where multiple classifiers are trained using different combinations of the classes. Predictions are then typically made using majority voting, i.e. The point is classified based on the majority decision of all the models.

There are a few different methods to do this and these were compared in *Multiclass Approaches for Support Vector Machine Based Land Cover Classification*. This paper concluded that on balance the One-vs-One method was best as whilst it didn't achieve the highest classification accuracy, the method that did (ECOC) required a very large training time (Pal, 2008). This is also the method supported by the SKlearn SVC according to s.1.4.1.1 of the documentation.

3.7.1 Training the model

Below are the key hyperparameters and choices for the model:

Cost(C) - This is the cost of violation to the margin. For most datasets the hyperplane or line will not perfectly separate the different classes and therefore we may want a model that allows some datapoints to sit the wrong side of the hyperplane/ line. Where the cost is set to a large number, the margin will be narrow but rarely violated. This means that the model will fit closely to the data and risks overfitting (low bias and high variance.) Where the cost is a small number the reverse is true and the model risks underfitting.

Gamma- Controls the shape of peaks on the hyperplane/ line. Small gamma gives a softer, broader bump, dependent on points that can be quite far from it. This gives high bias and low variance and can underfit the data. Large gamma gives more pointed bumps only dependent on points close to it. Has low bias and high variance, giving rise to overfitting.

Kernel - The kernel is the type of function the SVM uses to map the data from the original feature space into a higher dimensional space. The most common functions are: Linear, polynomial, RBF and Sigmoid.

Cost function - For SVMs the larger the margin the better because it allows more separation between the classes. Hinge Loss is often used as the cost function for SVMs because it penalises the model when it makes predictions that sit on the wrong side of the hyperplane and therefore are incorrectly classified, or that sit the right side of the hyperplane but close to the hyperplane, within the margin. (Alake, 2023)

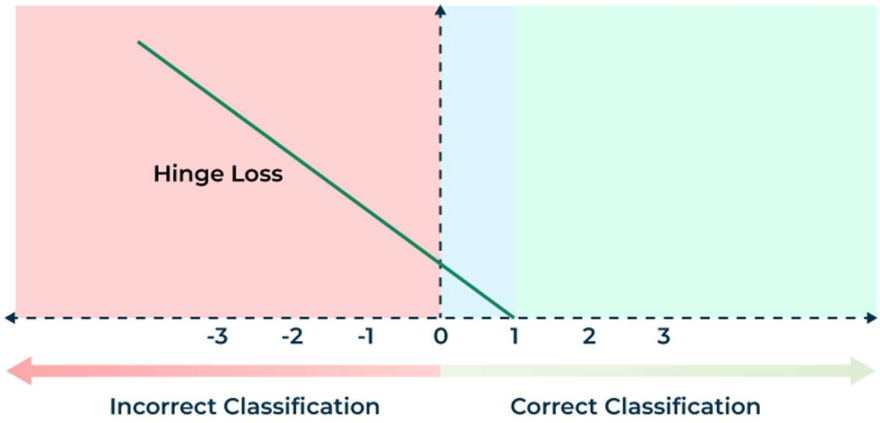


Figure 42. Hinge loss diagram (Geeks for Geeks, n.d.)

This can also be extended to multiclass problems and the Crammer-Singer method is the one used in the sklearn implementation of hinge_loss. (Crammer & Singer, 2001)

When training the SVM the challenge wasn't hyperparameters this time but the amount of data itself. Initially trained two baseline models, one each for StandardScaled data and MinMax normalised data, and they took c.90 minutes to train. Therefore, even using the method of tuning each hyperparameter in turn as we did for RF would not be computationally feasible. Instead, I used a grid search for the three hyperparameters (kernel, C and gamma) but reduced the size of the training set using feature selection and sampling without replacement. These were then compared to see which gave the best results. NB. The accuracy is lower than the final model trained because far less data was used (1/20th training set).

Data	**Accuracy**
MinMax	0.784737221
StandardScaler	0.827933765
SS with Pearson's Correlation for Feature Selection	0.783657307
SS with RF for feature selection without domain knowledge	0.806695464

Figure 43. Table of best accuracy scores using different scaling techniques and different features selected

3.7.2 Overall design of SVM

Design feature	**Choice**
Kernel	RBF
Cost	200
Gamma	1.6
Cost function	Hinge

Figure 44. Table of hyperparameter choices for SVM

3.8 Feedforward Neural Network (FNN)

A neural network is a model designed to imitate a human brain, designed as interconnected processors called neurons, connected by links with associated weights that can be adjusted to affect the output. (Negnevitsky, 2002/2011)

There are a range of different neural network types but for this project a FNN was trained as this has been shown to be successful when using tabular data in general (Damaraju & Ljubomir, n.d.) and more specifically when comparing machine learning techniques to improve upon DL (Abbas & Haider, 2019).

Also considered was an LSTM (Long Short Term Memory) neural network which takes time series data e.g. the ball by ball data in this project and retains some memory of the data passed through in order to make accurate predictions. This has also shown some promise in cricket forecasting (Gupta, Joshi, Tahlan, Gupta, & Agarwal, 2021), but this utilised player performance data to predict outcomes, rather than using the data that DL does. Training an LSTM is also computationally costly so may be worth further consideration in the future, to compare to the performance of the FNN.

It was also argued that in *The Unreasonable Effectiveness of Data* that "simple models and a lot of data trump more elaborate models based on less data."

FNNs consist of an input layer into which we pass the data, an output layer for the prediction from the model, hidden layer/s to transform the data and weights between the neurons to dictate the strength of the connection between the neurons. They may also contain bias terms which are constants added to the product of the weights * input to the neuron. Lastly each neuron will have an activation function, the product of the (weights * input) + bias will be passed through this function and will affect what the output looks like.

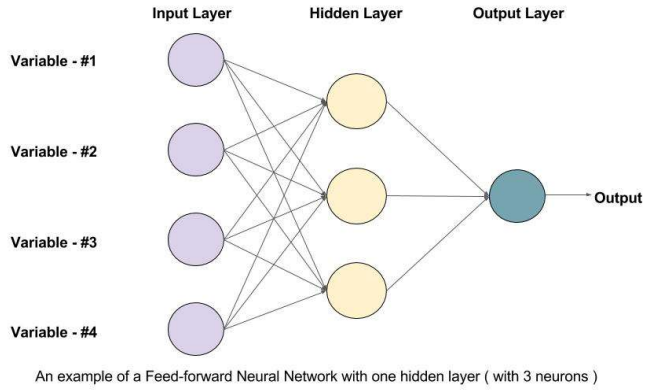


Figure 45. Diagram of an FFN (Mallick, n.d.)

3.8.1 Training the model

As with training the SVM, it was computationally prohibitive to train multiple neural networks to find the best values for the hyperparameters. Therefore, used random sampling with replacement to cut the dataset down by 1/20th for tuning the hyperparameters and then used the full training set to train the model once these were chosen.

Hidden layers – How many hidden layers are in the architecture. The more hidden layers there are, the more complex a situation the nn can model, which can become prone to overfitting and more computationally costly to train.

Neurons – How many neurons are in the architecture, as above this makes the network more complex and comes with the same advantages and disadvantages.

Epochs – An epoch is one run of the data through the neural network where it can potentially update the model's weights and biases. This can be optimised by using a method called "early stopping." With early stopping the accuracy and loss are calculated using the validation set after each epoch and when the model stops improving, and may be overfitting, the model training stops. A term called "patience" defines how sensitive early stopping is E.g. If patience = 2 then the model can stop improving for two epochs before the training stops. This patience term prevents the model from stopping too soon based on a small local minimum value for the loss/ error such as 50 epochs in the diagram below, when the real minimum is c. 200 instead. (Prechelt, 2012)

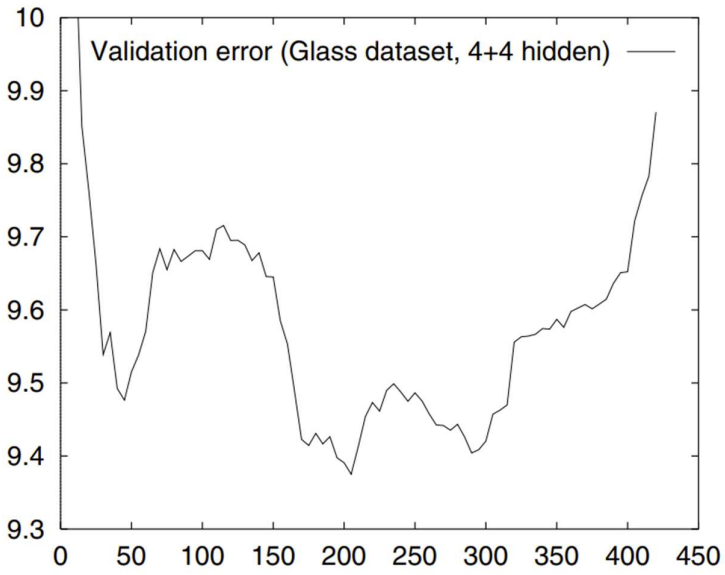


Figure 46. Example graph of validation error against number of epochs (Prechelt, 2012)

Dropout rate – A proportion of the neurons in the architecture will be "turned off" by setting them to 0 so that they don't impact training for that epoch. This is because the importance of some features in the data may be overstated if the neurons repeatedly extract the same features and if these features are specific to the training set then this can lead to overfitting and poor generalisation. (Hinton, Krizhevsky, Salakhutdinov, Srivastava, & Sutskever, 2014)

Batch size – Depending on the batch size it's possible to use the whole dataset to calculate the error gradient, and in turn therefore update the weights and biases of the model, or alternatively split these into "minibatches" and calculate the error gradient using these instead. Smaller batch sizes are faster to train but can have noise because of training the model on a smaller number of points, whereas larger batch sizes take much longer to train and give more accurate error gradient estimates. (Brownlee, A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size, 2019). These batch sizes can range from 1 to all the points in the data, but 32 is often the recommended size (Luschi & Masters, 2018), with

Yan LeCun making the tongue-in-cheek comment on Twitter (X) “Friends don’t let friends use minibatches larger than 32.” (LeCun, 2018)

Activation functions – The chosen form of the activation function creates different outputs from the neurons in a neural network. These can “activate” i.e. transform or keep the input to the neurons, or can “deactivate” them by setting the output to 0 so that that neuron has no effect. Below is a list of some of the main activation functions and their purpose.

Activation Function	Purpose
Binary Step	Output will either be 0 or 1, depending on a defined threshold
Linear	Output is the same as the input
Sigmoid	Output is in the range 0 to 1
Tanh	Output is in the range -1 to 1
ReLU	If the input is less than 0, the output will be 0. If it is above 0 then the output is the same as the input
Softmax	Can be used for multiclass classification as it returns a probability for a datapoint belonging to each class

Figure 47. Table of activation functions and their purpose

Because the problem isn’t linear or binary so Linear, Binary Step and Sigmoid were ruled out. For the hidden layers had the choice of using ReLU or Tanh – Tanh is typically only used in hidden layers because it centres the data but ReLU was more advantageous here because it is less computationally expensive and reduces the chance of overfitting because any negative neurons are turned off. For the output layer Softmax is best for multiclass.

Cost function - The typical cost function for a classification problem is crossentropy and because this is a multiclass classification problem there are two options here: categorical crossentropy or sparse categorical crossentropy. Categorical crossentropy requires one-hot-encoded target variables, whereas sparse categorical crossentropy requires a vector of integer labels e.g. [0,1,2.] One-hot-encoding the target variables turns them into a matrix of information [[1,0,0], [0,1,0], [0,0,1]] and therefore increases dimensionality and that can cause problems in itself. Therefore, chose to use sparse categorical crossentropy as the cost function here.

Optimisers - In training a neural network the aim is to find the weights and biases that correspond to a specific point on the cost function chosen, and optimisers are methods by which neural networks iteratively update the weights and biases to reach the optimal values.

There are a range of different types of optimiser e.g. Gradient descent. For gradient descent gradient at a point on the cost function is calculated, and then the point is moved a step along the slope of the cost function by taking the negative of the gradient.

Here Adam was used as it’s suitable for large datasets and shows improvement over previous optimisers such as RMSProp and Adagrad (Kingma & Ba, 2015).

Learning rate – The learning rate defines what size the step will be along the cost function by multiplying the magnitude of the gradient by the learning rate. The higher the learning rate the larger the step taken and faster the model trained, but there's a risk of missing the minimum values for the weights and biases. The lower the learning rate the smaller the step and it may make the model very slow to train.

The other benefit of using the Adam optimiser is it's possible to not tune the learning rate as it self-tunes during the training, which significantly speeds up the process, as shown in the code on git (5.NN_model.ipynb). Hyperparameter tuning including learning rate took c. 6 hours, whereas without took 1.5 hours and both gave optimal hyperparameters resulting in almost the same accuracy (93% for the model with learning rate and 92% for the model without.)

3.8.2 Overall design of NN

Design feature	**Choice**
Epochs	55
Early stopping patience	5
Layers	3
Neurons	25
Dropout rate	0.1
Learning rate	0.001
Hidden layers	ReLU
Output layer	Softmax
Cost function	Sparse categorical cross entropy
Optimiser	Adam

Figure 48. Table of hyperparameter choices for NN

4. Systems Architecture

Below is a brief overview of how each model was built using Python. Full code can be found at https://github.com/Birkbeck/msc-project-source-code-files-22-23-Swalke20.

4.1 Random Forest

The RF was built using the RandomForestClassifier from the sklearn library, tuned manually using for loops of options FOR each hyperparameter in turn.

```
model = RandomForestClassifier(n_estimators=192, max_features=3, max_samples=0.3, max_depth=18, n_jobs=-1, random_state=7, criterion='log_loss')
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)

y_val_pred = model.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)

y_test_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
```

Figure 49. Snippet of code training RF

The order of this was chosen using ANOVA scores to rank the importance of each of the hyperparameter.

```
# Create an empty dictionary to store the total ANOVA score for each hyperparameter
total_anova_scores = {hyperparameter: 0 for hyperparameter in param_grid}

# Create an empty array to store ANOVA scores for each hyperparameter
anova_scores = []

# Loop through all combinations of hyperparameter values
for hyperparameter_values in product(*param_grid.values()):
    hyperparameters = dict(zip(param_grid.keys(), hyperparameter_values))

    # Create a Random Forest model
    rf = RandomForestClassifier(random_state=7)

    # Create a feature selector using ANOVA
    selector = SelectKBest(f_classif, k='all')

    # Fit the selector on the training data with the specific hyperparameter values
    selector.fit(X_train, y_train)

    # Get the ANOVA scores for each feature
    anova_scores.extend(zip(hyperparameters, selector.scores_))

    # Accumulate the ANOVA scores for each hyperparameter
    for hyperparameter, score in zip(hyperparameters, selector.scores_):
        total_anova_scores[hyperparameter] += score

# Sort hyperparameters based on their total scores in descending order
sorted_hyperparameters = sorted(total_anova_scores.items(), key=lambda x: x[1], reverse=True)

# Print the hyperparameters in order of highest total score to lowest
for hyperparameter, total_score in sorted_hyperparameters:
    print(f"{hyperparameter}: {total_score}")
```

Figure 50. Snippet of code for ANOVA

4.2 Support Vector Machine

The SVM was built using the support vector classifier SVC function from the sklearn library.

```
final_model= SVC(kernel="rbf", random_state=7, C=200, gamma=1.6)
final_model.fit(X_ss_train, y_ss_train)

y_train_pred = final_model.predict(X_ss_train)
y_val_pred = final_model.predict(X_ss_val)
y_test_pred = final_model.predict(X_ss_test)
```

Figure 51. Snippet of code training SVM

The SVC was tuned manually using nested FOR loops of combinations of options of the hyperparameters.

```
best_accuracy = 0
best_loss = 100

for k in k_list:
    for gamma in gamma_list:
        for c in c_list:

            start_time = time.time()

            # for each combination of parameters, train an SVC
            model = SVC(kernel=k, gamma=gamma, C=c, random_state=7)
            model.fit(X_ss_train_sample, y_ss_train_sample)
            y_train_pred = model.predict(X_ss_train_sample)
            y_val_pred = model.predict(X_ss_val_sample)

            train_accuracy = accuracy_score(y_ss_train_sample, y_train_pred)
            val_accuracy = accuracy_score(y_ss_val_sample, y_val_pred)

            train_loss = hinge_loss(y_ss_train_sample, model.decision_function(X_ss_train_sample))
            val_loss = hinge_loss(y_ss_val_sample, model.decision_function(X_ss_val_sample))

            end_time = time.time()
            elapsed_time = end_time - start_time

            print(f"{k}, Gamma: {gamma}, C: {c}, Train Accuracy: {train_accuracy}, Validation Accuracy: {val_accuracy}, Train Loss: {train_loss}, Validation Loss: {val_loss}, Time to train: {elapsed_time} seconds")

            if val_accuracy > best_accuracy:
                best_accuracy = val_accuracy
                best_acc_parameters = {'C': c, 'gamma': gamma, 'kernel': k}

            if val_loss < best_loss:
                best_loss = val_loss
                best_loss_parameters = {'C': c, 'gamma': gamma, 'kernel': k}

print(f"Best accuracy is: {best_accuracy} and best parameters are {best_acc_parameters}")
print(f"Best loss is: {best_loss} and best parameters are {best_loss_parameters}")
```

Figure 52. Snippet of code tuning SVM hyperparameters

4.3 Neural Network

The Neural Network classifier was trained using Keras to build each layer of the neurons, then compiled.

```
def create_model(hidden_layers, neurons, dropout_rate, learning_rate):
    model = Sequential()
    model.add(Dense(neurons, input_dim=input_dim, activation='relu'))
    model.add(Dropout(dropout_rate))

    for _ in range(hidden_layers - 1):
        model.add(Dense(neurons, activation='relu'))
        model.add(Dropout(dropout_rate))

    model.add(Dense(3, activation='softmax'))
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

Figure 53. Snippet of code – function to create NN

This was also trained using nested FOR loops for the combinations of hyperparameters as well as EarlyStopping from the keras library to stop training the model it is no longer improving.

```
# Iterate over hyperparameter combinations
for hidden_layers in param_grid['hidden_layers']:
    for neurons in param_grid['neurons']:
        for dropout_rate in param_grid['dropout_rate']:
            for learning_rate in param_grid['learning_rate']:
                print(f"Training model with {hidden_layers} hidden layers, {neurons} neurons, "
                      f"{dropout_rate} dropout, and {learning_rate} learning rate.")

                # Create the model
                model = create_model(hidden_layers, neurons, dropout_rate, learning_rate)

                # Set up early stopping - Patience chosen so that there's less of a chance of getting caught in a local minima
                early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)


                # Train the model
                history = model.fit(X_ss_train_sample, y_ss_train_sample, epochs=200, batch_size=32,
                                    validation_data=(X_ss_val_sample, y_ss_val_sample), callbacks=[early_stopping])



                # Get the number of epochs it took for early stopping
                epochs = len(history.history['loss'])
                print(f"Early stopped after {epochs} epochs.")

                #Evaliud
                _, train_accuracy = model.evaluate(X_ss_train_sample, y_ss_train_sample)

                # Evaluate on validation set
                _, val_accuracy = model.evaluate(X_ss_val_sample, y_ss_val_sample)
                print(f"Validation accuracy: {val_accuracy}\n")

                hidden_layers_list.append(hidden_layers)
                neurons_list.append(neurons)
                dropout_rate_list.append(dropout_rate)
                learning_rate_list.append(learning_rate)
                epochs_list.append(epochs)
                train_accuracy_list.append(train_accuracy)
                val_accuracy_list.append(val_accuracy)
```

Figure 54. Snippet of code tuning NN hyperparameters

4.4 Evaluation

All three models were evaluated using confusion_matrix and accuracy_score from the sklearn library.

```
# Model Accuracy: how often is the classifier correct?
print(f"Train Accuracy: {accuracy_score(y_ss_train, y_train_pred): .2f}, Val Accuracy: {accuracy_score(y_ss_val, y_val_pred): .2f}, Test Accuracy: {accuracy_score(y_ss_test, y_test_pred): .2f}")

cm = confusion_matrix(y_ss_test, y_test_pred, labels=final_model.classes_)
cmd = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels=labels)

fig, ax = plt.subplots(figsize=(7,7))
cmd.plot(ax=ax)
plt.grid(False)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
ax.set_title('Confusion Matrix SVM')
plt.show()
```

Figure 55. Snippet of confusion matrix and accuracy_score code

A full list of the libraries used in this project can be found in Appendix I and an overview of the github repository in Appendix II.

5. Results and Analysis

5.1 Accuracy

The project was a success, with each method (RF, SVM and FNN) showing a 10% improvement over the predictive accuracy of the Duckworth-Lewis method for men's ODI cricket matches between 1st January 2002 and 31st December 2022.

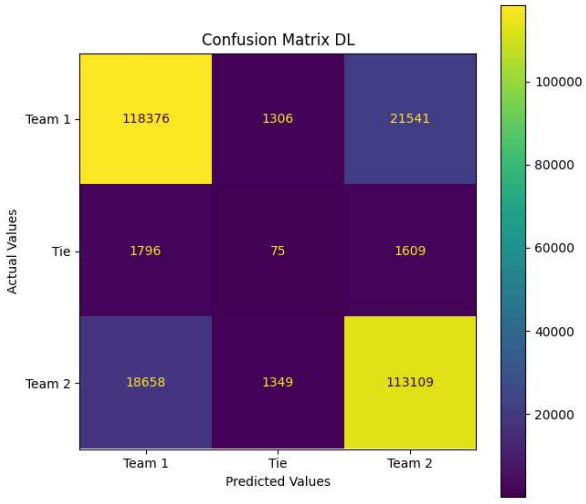
Method	Accuracy
DL	83
Random Forest	93
SVM	93
Feedforward Neural Network	93

Figure 56. Table comparing accuracy of models

Each model showed a very similar increase in accuracy over the DL method which suggests the additional features used (aka the resources) were more important to improving predictions than the model type itself.

5.2 Confusion Matrices, Precision and Recall

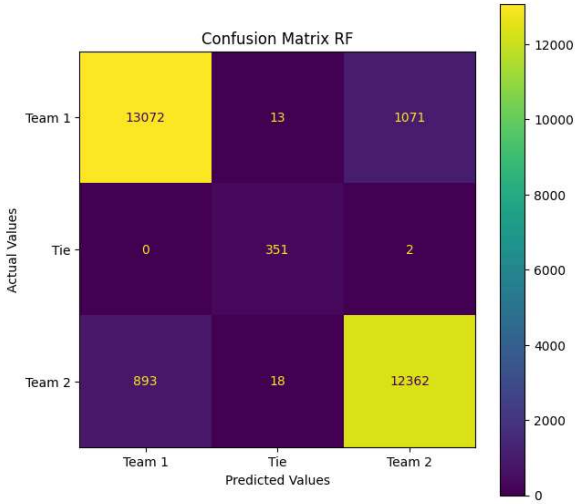
The models (RF, SVM and FNN) also all show improvement in precision and recall, though to different extents. Below is the confusion matrix for the Duckworth-Lewis method which suffers from very low precision, recall and therefore f1 score for predicting a Tie. As discussed earlier this is because the data is very imbalanced with just 1% being in the class "Tie."



	precision	recall	f1-score	support
Team 1	0.85	0.84	0.85	141223
Tie	0.03	0.02	0.02	3480
Team 2	0.83	0.85	0.84	133116
accuracy			0.83	277819
macro avg	0.57	0.57	0.57	277819
weighted avg	0.83	0.83	0.83	277819

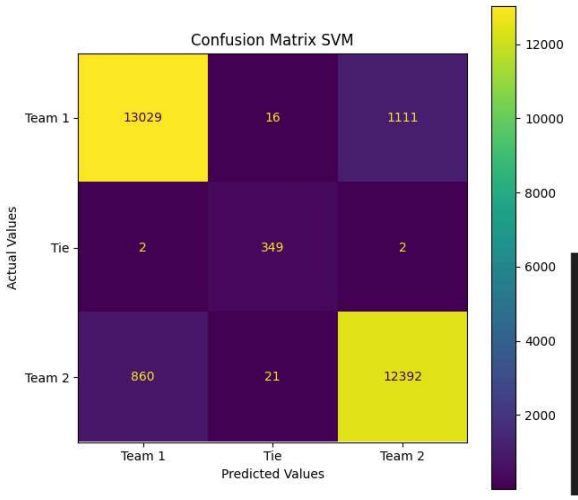
Figure 57. Confusion matrix Duckworth Lewis, Figure 58. Performance metrics DL

This is significantly improved upon with the Random Forest model, where the precision and recall for Tie overtook that of Team 1 and Team 2 as well. It also shows an almost equal precision and recall for each class, which means that most of the classes were correctly predicted but also not over-predicted.



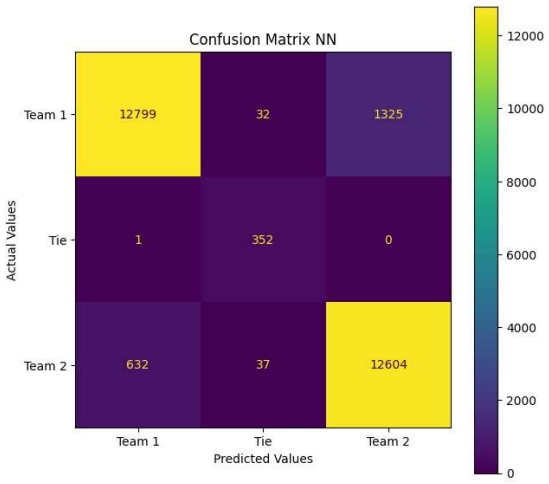
	precision	recall	f1-score	support
Team 1	0.94	0.92	0.93	14156
Tie	0.92	0.99	0.96	353
Team 2	0.92	0.93	0.93	13273
accuracy			0.93	27782
macro avg	0.93	0.95	0.94	27782
weighted avg	0.93	0.93	0.93	27782

Figure 59. Confusion matrix RF, Figure 60. Performance metrics RF



	precision	recall	f1-score	support
Team 1	0.94	0.92	0.93	14156
Tie	0.90	0.99	0.94	353
Team 2	0.92	0.93	0.93	13273
accuracy			0.93	27782
macro avg	0.92	0.95	0.93	27782
weighted avg	0.93	0.93	0.93	27782

Figure 61. Confusion matrix SVM, Figure 62. Performance metrics SVM



	precision	recall	f1-score	support
Team 1	0.95	0.90	0.93	14156
Tie	0.84	1.00	0.91	353
Team 2	0.90	0.95	0.93	13273
accuracy			0.93	27782
macro avg	0.90	0.95	0.92	27782
weighted avg	0.93	0.93	0.93	27782

Figure 63. Confusion matrix NN, Figure 64. Performance metrics NN

The SVM and NN also improve upon DL but show over prediction of the "Tie" class with almost 100% recall but lower precision, particularly for the NN. This is likely because whilst the training data was oversampled (SMOTE) to give a balanced set, the testing and validation sets are not. It's arguable that if a match outcome is predicted incorrectly that the safest incorrect prediction would be a Tie, rather than awarding the win to the wrong team. However, this would need to be balanced with the ethos of ODIs which is that an outcome needs to be reached.

5.3 Distribution of classes

The models also maintain the distribution of classes better than DL does. Below are plots of the distributions for: The actual winners, predicted winners under DL and predicted winners for each model. Under the diagrams is a table to show how different these predictions are from the actual results.

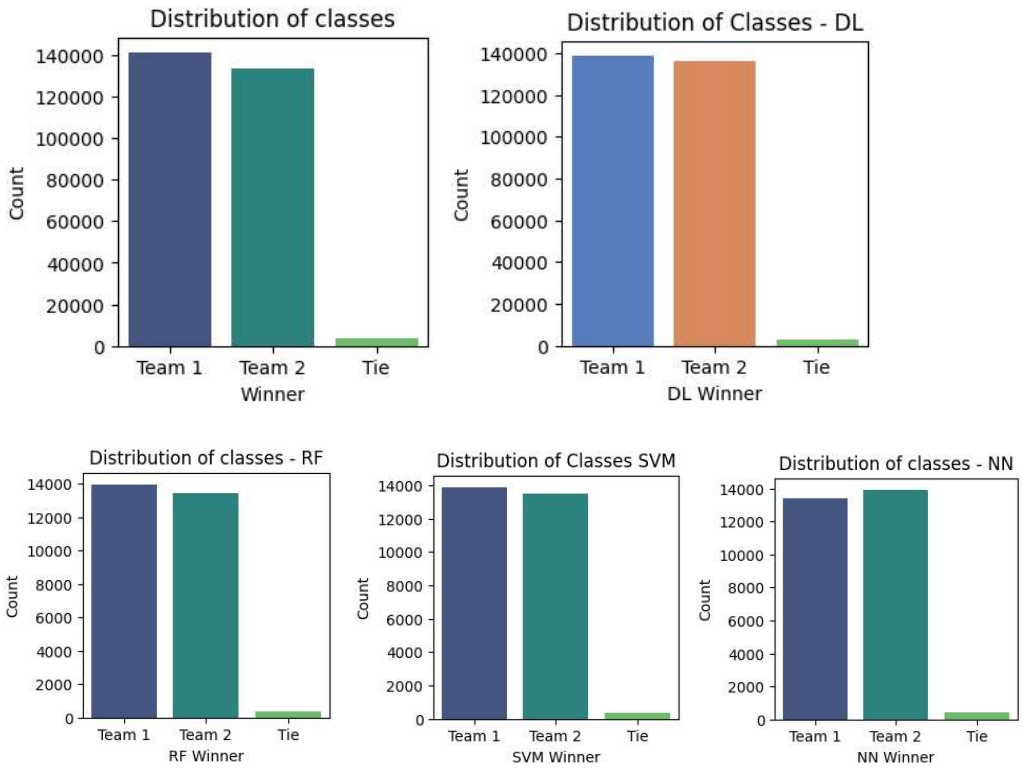


Figure 65. Bar charts of class distribution for: Actual match outcome, DL, RF, SVM and NN predictions

	Results	DL	% diff	RF	% diff	SVM	% diff	NN	% diff
Team 1	0.508327	0.499714	-0.86%	0.502664	-0.57%	0.5	-0.83%	0.501368	0.16%
Team 2	0.479146	0.49046	1.13%	0.483586	0.44%	0.486106	0.70%	0.483479	-0.37%
Tie	0.0125261	0.00982654	-0.27%	0.01375	0.12%	0.013894	0.14%	0.015154	0.21%

Figure 66. Table showing the difference between the actual class distribution and the different models

These show that DL is the most skewed, favouring Team 2 over Team 1. RF and SVM models also favours Team 2 but to a lesser extent, whereas the NN favours Tie and Team 1.

5.4 Other analysis

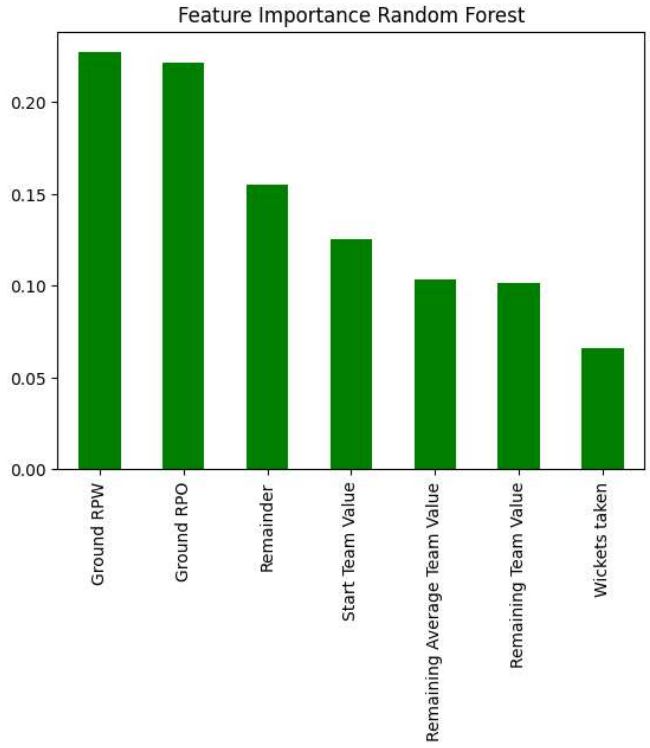


Figure 67. Feature importance bar chart

It's also interesting to discuss the feature importance for the RF model, which is the simplest to visualise. It's surprising to see that "Wickets taken" is the least important feature given it's one of the two features used by DL. "Remainder" is the other feature used by DL and whilst it's one of the more important features this is also overtaken by "Ground RPW" and "Ground RPO." This would be an area for further exploration to see what the causal link is, for example is it that over a shorter number of overs the capacity to score higher numbers of runs is vital?

6. Conclusion

This project was a success, with the initial aims both met as well as some of the criticisms of DL overcome. Below these are assessed and suggestions made for future study given more time and additional resources.

6.1 Looking back at the aims of this project

1. To train a Random Forest, Support Vector Machine and Neural Network to improve upon the predictive capability of the Duckworth-Lewis method.
2. To extend the definition of resources to include scoring rates at cricket grounds and player "value" to improve upon predictive capability of Duckworth-Lewis method.

All three models improved upon the predictive capability of DL by 10%, not just by using machine and deep learning but also by using the additional ground scoring rates and player values.

6.2 Overcoming criticisms

The project also goes some way to addressing the criticisms of DL mentioned in section 1.7 which could be seen as secondary aims. Below is an analysis of which criticisms were overcome and which weren't.

DL Criticisms	Success	Did the models improve this?
Complexity	No	With an app this would be simple to overcome the complexity of using the method. It is also less easy to explain than the DL method.
Biased towards Team 1 or Team 2	Yes	All three models improved upon this.
Not taking into consideration probability of winning	N/A	Not relevant as DL was improved upon rather than replaced
Being open to manipulation	Yes	Given the more important elements of the models were Ground RPO/ RPW and Team Value it would be difficult to artificially change the target score
Wickets too important	Yes	Wickets taken were actually the least important to the models
Parametric rather than non-parametric	Yes	All three models were non-parametric so didn't assume functional forms of the relationships in the data
No fielding restrictions	No	Too complex and seemed to have a negligible impact on matches
No T20	No	This would be addressed in future study

Figure 68. Table of DL criticisms and whether the new models overcame them

6.3 Limitations/ Further Study

Whilst overall the performance of the models was a success, there are some limitations of the project that may be worth considering for further study.

6.3.1 Data

RPW/ RPO averages are up-to-date and therefore higher than they would have been in 2002 when the data begins. In further study it would be interesting to see if there's a difference in predictive ability of the models over the time-period. Similarly, this could be considered for groups of overs e.g. How good the predictions are when only 30 overs are played, as opposed to 40.

According to Stern and the ICC the DLS method also works with T20 match data as well as women's ODI match data so in further study this information would be given to the model to see if this improves or reduces predictive capability.

6.3.2 Resources

The player values for batter, all-rounder and bowler were chosen based on an estimate of average career ODI run scores of 40, 30 and 20 respectively. However, there may be a difference in how this correlates with number of wickets lost if other numbers were chosen. For example, it would be expected that the correlation would be higher if these values were 3, 2 and 1 as with a loss of 1 wicket where a bowler was the facing batter, the relationship would be 1:1. Therefore, if the values were very different we'd expect lower correlation. This may be worth considering in future study.

6.3.3 Classes

In further work it would also be worth considering whether "Tie" is really a class, or an absence of a class. It happens very rarely, and ODIs are specifically designed to avoid ties so if there were a way to remove these altogether, the predictive capacity would be very high. This can be seen below in an example boxplot of the winner class against wickets taken where the 0 class (Team 1 wins) and 2 class (Team 2 wins) look very separable but the Tie class overlaps with both:

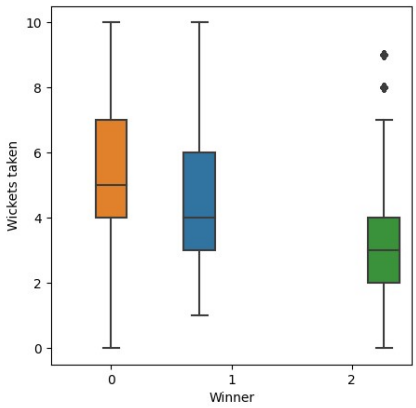


Figure 69. Boxplot of Wickets taken against the outcome (0 for Team 1 win, 1 for Tie and 2 for Team 2 win.)

In exploring this further one-hot-encoding could be considered instead of label encoding as used in this project as the model may have learned the relationship between the labels themselves (0,1,2.)

It's also worth adding that if any of the ML methods were to be used as a replacement for DL, this would need to be trained as a regression problem so that the runs needed to win could be predicted as the match continued.

6.3.4 Computation

Due to challenges with computational cost, it wasn't possible to tune the hyperparameters for the SVM or NN using the full set, or even using cross-validation. Therefore, only a small subset was used to improve these, rather than the larger training set. This inevitably means there may be some bias introduced. In future study it would be important to be able to take the time/ have the physical capacity to run tuning of the models in full to make them more robust.

It would also be useful to consider ensemble methods such as bagging to see if this improved the performance of the models. An example of this was run for the SVM model (please see 4.SVM_model.ipynb on git) but was also only run on the sample data. If possible to run on the whole dataset, I would consider a local learning strategy similar to that set out in *Efficient Machine Learning for Big Data: A Review* (Al-Jarrah, Yoo, Muhaidat, & Karagiannidis, 2015).

6.4.5 Models

The feature selection techniques chosen for SVM didn't out-perform using the original 7 features. This may be due to the size of the dataset as 7 features isn't a very high dimension for the number of samples but given more time it would be good to consider other possible feature selection techniques to see if these worked better. For example, Autoencoders weren't considered in my original proposal but have been used successfully for feature selection (Atashgahi, et al., 2022).

Lastly this project only considers the scenario where 1st innings is played in full and the 2nd finishes after 20+ overs and therefore does not look at situations where there is a halt in play but it subsequently restarts. This would be much more complex and therefore would only be worth pursuing if undertaking a large study.

7. References

Abbas, K., & Haider, S. (2019). Duckworth-Lewis-Stern Method Comparison with Machine Learning Approach. Islamabad: International Conference on Frontiers of Information Technology (FIT).

Acharya, A. (2023, June 13). *Training, Validation, Test Split for Machine Learning Datasets*. Retrieved from Encord: https://encord.com/blog/train-val-test-split/#:~:text=The%20rough%20standard%20for%20train,10%2D20%25%20test%20data.

Akhtar, A. (2014, April 2003). *Duckworth-Lewis: A mathematical conundrum?* Retrieved from Sportskeeda: https://www.sportskeeda.com/cricket/duckworth-lewis-method-a-mathematical-conundrum

Alake, R. (2023, November). *Loss Functions in Machine Learning Explained*. Retrieved from Data Camp: https://www.datacamp.com/tutorial/loss-function-in-machine-learning

Ali, J., Khan, R., Ahmad, N., & Maqsoo, I. (2012). Random Forests and Decision Trees. *International Journal of Computer Science Issues, 9*(5).

Al-Jarrah, O., Yoo, P., Muhaidat, S., & Karagiannidis, G. (2015). Efficient Machine Learning for Big Data: A Review. *Big Data Research*, 87-93.

Alves, M. (2008, November 25). *Duckworth-Lewis Method: Cricket Fair Play Or Another Complication?* Retrieved from Bleacher Report: https://bleacherreport.com/articles/85833-duckworth-lewis-method-cricket-fair-play-or-another-complication

Angel, G. (2013, March). *From Infamy to Ingenuity: The Evolution of Duckworth-Lewis Method*. Retrieved from Medium: https://medium.com/letters-from-a-sports-fan/dls-27c2ec93fc7c

Asif, M., & McHale, I. (2013). A modified Duckworth–Lewis method for adjusting targets in interrupted limited overs cricket. *European Journal of Operational Research, 225*(2), 353-362.

Atashgahi, Z., Sokar, G., van der Lee, T., Mocanu, E., Mocanu, D., Veldhuis, & Pechenizkiy, M. (2022). Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. *Machine Learning*, 377-414.

Bailey, M., & Clarke, S. (2006). Predicting the Match Outcome in One Day International Cricket Matches, While the Match is in Progress. *Journal of Sports Science and Medicine*, 480-487.

Banerjee, M., Ding, Y., & Noone, A. (2012). Identifying representative trees from ensembles. *Stat Med*.

Baranauskas, J., Oshiro, T., & Perez, P. (2012). How Many Trees in a Random Forest? In P. Perner, *Machine Learning and Data Mining in Pattern Recognition* (pp. 154-168). Berlin: Springer-Verlag.

Batting average (cricket). (n.d.). Retrieved December 22, 2023, from Wikipedia: https://en.wikipedia.org/wiki/Batting_average_(cricket)

Bernard, S., & Heutte, L. A. (2009). Influence of Hyperparameters on Random Forest Accuracy. *International Workshop on Multiple Classifier Systems (MCS)*, (pp. 171-180). Reykjavik.

Bernard, S., Heutte, L., & Adam, A. (2009). Influence of Hyperparamters on Random Forest Accuracy. In J. Benediktsson, J. Kittler, & F. Roli, *Multiple Classifier Systems* (pp. 171-181). Reykjavik: Springer.

Bhattacharya, R., Gill, P., & Swartz, T. (2011). Duckworth-Lewis and Twenty20 Cricket. *Journal of the Operational Research Society, 62*(11), 1951-1957.

Bhogle, S. (2003). *The Duckworth/Lewis Factor*. Retrieved May 16, 2023, from https://www.rediff.com/wc2003/2003/mar/06bhogle.htm

Boulesteix, A.-L., Probst, P., & Wright, M. (2019). Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery*.

Breiman, L., Friedman, J., Olshen, A., & Stone, C. (1984). Classification and Regression Trees. *Biometrics*, 874.

Brownlee, J. (2019, August 19). *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Brownlee, J. (2021, March 17). *SMOTE for Imbalanced Classification with Python*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Brownlee, J. (2023, October 4). *A Gentle Introduction to k-fold Cross-Validation*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/k-fold-cross-validation/

Cawley, G., & Talbot, N. (2007). Preventing over-fitting in model selection via Bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research,, 8*, 841-861.

Chalwa, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *JournalofArtificial IntelligenceResearch16*, 321-357.

Crammer, K., & Singer, Y. (2001). On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 265-292.

Cricsheet. (n.d.). *Available Match Data Downloads*. Retrieved July 2023, from Cricsheet: https://cricsheet.org/downloads/

Damaraju, N., & Ljubomir, B. (n.d.). *Searching for the Best Tabular Data Classifiers*. Retrieved May 19, 2023, from https://inflammatix.com/searching-for-the-best-tabular-data-classifiers-blog/

Datta, R., & Singh, S. (2021). Artificial inteligence in critical care: Its about time! *Medical Journal Armed Forces India, 77*, 266-275.

Deloitte. (2019). *Global Mobile Consumer Survey: UK cut.* Retrieved from Deloitte: https://www2.deloitte.com/content/dam/Deloitte/uk/Documents/technology-media-telecommunications/deloitte-uk-plateauing-at-the-peak-the-state-of-the-smartphone.pdf

Duckworth, F., & Lewis, A. (2004). A Successful Operational Research Intervention in One-Day Cricket. *The Journal of the Operational Research Society, 55*(7), 749-759.

Duckworth, F., & Lewis, T. (1998). A Fair Method for Resetting the Target in Interrupted One-Day Cricket Matches. *The Journal of the Operational Research Society, 49*(3), 220-227.

Duckworth, F., & Lewis, T. (1999). *Your comprehensive guide to the Duckworth/Lewis method for re-setting targets in one-day cricket.* Bristol: University of the West of England.

Duckworth, F., & Lewis, T. (2008). *D/L method: answers to frequently asked questions*. Retrieved May 17, 2023, from http://static.espncricinfo.com/db/ABOUT_CRICKET/RAIN_RULES/DL_FAQ.html

ECB. (2023, December 10). *Duckworth-Lewis-Stern Methodology of Recalulating The Target Score in an Interrupted Match*. Retrieved from ECB: https://resources.ecb.co.uk/ecb/document/2023/03/31/e382094c-98ab-4809-a4c6-18596a3a9c23/14-Duckworth-Lewis-Stern-Regulations-2023.pdf

Egyir, R., & Devendorf, R. (2020). *Design of a Cricket Game Using FPGAs.* Worcester: Worcester Polytechnic Institute.

Ellis, C. (2022, September 20). *Hyperparameter tuning in random forests*. Retrieved from Crunching the Data: https://crunchingthedata.com/hyperparameter-tuning-in-random-forests/

ESPN. (n.d.). *Australia v England First ODI*. Retrieved December 7, 2023, from ESPN cricinfo: https://www.espncricinfo.com/series/england-marylebone-cricket-club-tour-of-australia-1970-71-61725/australia-vs-england-only-odi-64148/full-scorecard

ESPN. (n.d.). *Records for Test Matches*. Retrieved December 6, 2023, from ESPN cricinfo: https://www.espncricinfo.com/records/tied-matches-283891

ESPN. (n.d.). *Tied Matches*. Retrieved December 7, 2023, from ESPN cricinfo: https://www.espncricinfo.com/records/tied-matches-283892

Evidently AI. (n.d.). *Accuracy, precision and recall in multi-class classification*. Retrieved from Evidently AI: https://www.evidentlyai.com/classification-metrics/multi-class-metrics#:~:text=To%20calculate%20the%20recall%2C%20divide,True%20Positives%20and%20False%20Negatives.

Findlater, J. (2021, October 7). *How Long is a T20 Match in Cricket?* Retrieved from Birmingham Mail: https://www.birminghammail.co.uk/sport/cricket/how-long-t20-match-last-21794047

Fortmann-Roe, S. (2012, June). *Understanding the Bias Variance Tradeoff*. Retrieved from Scott Fortmann-Roe: https://scott.fortmann-roe.com/docs/BiasVariance.html

Fortmann-Roe, S. (2012, June). *Understanding the Bias-Variance Tradeoff*. Retrieved from Scott.Fortmann-Roe: https://scott.fortmann-roe.com/docs/BiasVariance.html

Fraser, D., & Sheridan, J. (2022, October 26). *JUST NOT CRICKET What is the Duckworth-Lewis-Stern method, how is it calculated and when is it used in the T20 World Cup?* Retrieved from The Sun: https://www.thesun.co.uk/sport/cricket/3735644/duckworth-lewis-stern-new-method/

Geeks for Geeks. (n.d.). *Hinge-loss & relationship with Support Vector Machines*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/hinge-loss-relationship-with-support-vector-machines/

Geron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow* (2nd ed.). Sebastopol: O'Reilly Media.

Goyal, C. (2021). *Bagging- 25 Questions to Test Your Skills on Random Forest Algorithm*. Retrieved May 29, 2023, from https://www.analyticsvidhya.com/blog/2021/05/bagging-25-questions-to-test-your-skills-on-random-forest-algorithm/

Gupta, P., Joshi, N., Tahlan, R., Gupta, D., & Agarwal, S. (2021). Cricket Score Forecasting using Neural Networks. *International Journal of Engineering and Advanced Technology, 10*(5).

Gurung, R. (2018). ESPN Cricket Players Data. Retrieved August 2023, from https://www.kaggle.com/datasets/raghav333/espn-cricket-players-data

Hazra, S., Chatterjee, S., Mandal, A., Sarkar, M., & Mandal, B. (2023). An Analysis of Duckworth-Lewis-Stern Method in the Context of Interrupted Limited over Cricket Matches. In *Proceedings of International Conference on Data Analytics and Insights, ICDAI 2023.* Kolkata.

Hinton, G., Krizhevsky, A., Salakhutdinov, R., Srivastava, N., & Sutskever, I. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 1929-1958.

Hogan, S., & Brooker, S. (2012, November 22). *Cricket and the Wasp: Shameless self promotion (Wonkish).* Retrieved from Offsetting Behaviour: https://offsettingbehaviour.blogspot.com/2012/11/cricket-and-wasp-shameless-self.html

howstat. (n.d.). *One Day Internationals - Ground Records and Statistics*. Retrieved from howstat: http://howstat.com/cricket/Statistics/Grounds/GroundRunsPerWicket.asp?Mode=O

Huang, B., & Boutros, P. (2016). The parameter sensitivity of random forests. *BMC Bioinformatics*.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2014). An Efficient Approach for Assessing Hyperparameter Importance. *Proceedings of International Conference on Machine Learning* , (pp. 754-762).

ICC. (2002). *The D/L (Duckworth/ Lewis) method of adjusting target scores in interrupted one-day cricket matches.* Retrieved from ICC: https://icc-static-files.s3.amazonaws.com/ICC/document/2017/01/09/57272b5a-775e-4034-bb49-437052b5bee6/DuckworthLewis-Standard-Edition-Table.pdf

ICC. (2017). *Duckworth Lewis Stern*. Retrieved May 18, 2023, from ICC Cricket: https://www.icc-cricket.com/about/cricket/rules-and-regulations/duckworth-lewis-stern

ICC. (2018, September 29). *ICC releases latest version of DLS system*. Retrieved from ICC Cricket: https://www.icc-cricket.com/media-releases/864006

Jaipal, M. (2017). *Improving Duckworth Lewis method by using machine learning.* Dublin.

Jayadevan, V. (2002). A new method for the computation fo target scores in interrupted, limited-over cricket matches. *Current Science, 83*(5), 577-586.

Jewson, J., & French, S. (2015). *An Analysis of the Duckworth Lewis method in the context of English county cricket.* Warwick: University of Warwick.

Jewson, J., & French, S. (n.d.). The Duckworth-Lewis-Stern Method and County Cricket. Retrieved December 10, 2023, from https://warwick.ac.uk/fac/sci/statistics/staff/research_students/jewson/dl_poster_asru_nocricket.pdf

Kanstren, T. (2020, September 11). *A Look at Precision, Recall and F1-Score*. Retrieved from Medium: https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

Kaushik, S. (2023, Feb 23). *Introduction to feature selection methods with an example*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. *ICLR.*

Lancaster, R. (2015, June 25). *Why Cricket's Duckworth-Lewis Method is Outdated and Needs a Complete Overhaul*. Retrieved from Bleacher Report: https://bleacherreport.com/articles/2505332-why-crickets-duckworth-lewis-method-is-outdated-and-needs-a-complete-overhaul

Lavalette, T. (2019, February 19). *Crickets problem with being too slow*. Retrieved December 8, 2023, from Forbes: https://www.forbes.com/sites/tristanlavalette/2019/02/13/crickets-problem-with-being-too-slow/

LeCun, Y. (2018, April 26). *Twitter*. Retrieved from Twitter: https://twitter.com/ylecun/status/989610208497360896

Luschi, C., & Masters, D. (2018). *Revisiting Small Batch Training for Deep Neural Networks.* Graphcore Research.

Mallick, S. (n.d.). *Understanding Feedforward Neural Networks*. Retrieved May 29, 2023, from https://learnopencv.com/understanding-feedforward-neural-networks/

Manage, A., Silva, R., & Swartz, T. (2015). A Study of the Powerplay in One-Day Cricket. *European Journal of Operational Research 244(3)*.

Mirjalili, V., & Raschka, S. (2019). *Python Machine Learning* (3rd ed.). Packt Publishing.

Narkhede, S. (2018, May 9). *Understanding Confusion Matrix*. Retrieved from Medium: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Negnevitsky, M. (2002/2011). What is a Neural Network? In *Artificial Intelligence A Guide to Intelligent Systems Third Edition* (pp. 166-167). Pearson.

Pal, M. (2008). *Multiclass Approaches for Support Vector Machine Based Land Cover Classification.* Kurukshetra: arXiv.org.

Pearson, K. (1904). *On the Theory of Contingency and its Relation to Association and Normal Correlation.* London: Dulau and Co.

Perera, H. (2011). *A Second Look at Duckworth-Lewis in Twenty20 Cricket.* Burnaby: Simon Fraser University.

Powerplay rules in ODI cricket: History and application. (2022, June 22). Retrieved from Sports Adda: https://www.sportsadda.com/cricket/features/powerplay-rules-odi-cricket

Prechelt, L. (2012). Early Stopping-But When? In G. Montavon, G. Orr, & K. Muller, *Neural Networks: Tricks of the Trade 2nd Ed* (pp. 53-67). Springer.

Preston, I., & Thomas, J. (2002). Rain rules for limited overs cricket and probabilities of victory. *Journal of the Royal Statistical Society: Series D (The Statistician) Volume 51, Issue 2*, 189-202.

Preston, I., & Thomas, J. (2015, June 17). *Duckworth-Lewis has had a good innings in one-day cricket. Is it time for it to retire?* Retrieved from The Conversation: https://theconversation.com/duckworth-lewis-has-had-a-good-innings-in-one-day-cricket-is-it-time-for-it-to-retire-43278

Quinlan, J;. (1986). Induction of decision trees. In Springer, *Machine Learning* (pp. 81-106). Boston: Kluwer Academic Publishers.

Ramachandran, R. (2002). *For a fair formula*. Retrieved May 16, 2023, from https://web.archive.org/web/20070322191327/http://www.hinduonnet.com/fline/fl1924/stories/20021206004410400.htm

Roßbach, P. (2018). *Neural Networks vs. Random Forests – Does it always have to be Deep Learning?* Retrieved May 20, 2023, from https://blog.frankfurt-school.de/neural-networks-vs-random-forests-does-it-always-have-to-be-deep-learning/

Ruislip Middlesex Cricket Club. (n.d.). *10 ways of getting out*. Retrieved December 8, 2023, from Ruislip Middlesex Cricket Club: https://ruislipmiddx.cricketclubwebsite.co.uk/pages/page_4503/10-ways-of-getting-out.aspx

Sahu, V. (2019, May 28). *The Issues with the Duckworth-Lewis-Stern Method*. Retrieved from Bruin Sport Analytics: https://www.bruinsportsanalytics.com/post/duckworthlewissternmethod

Salam, Z., Liagat, H., Zia, H., Kong, X., & Shamshad, S. (2022). Player-aware resource compensation in interrupted cricket matches. *PeerJ Computer Science, 8*(e917).

Sarker, I. (2021). Machine Learning: Algorithms, Real-World Applications and Research. *SN Computer Science*.

Saxena, S. (2023, August 25). *A Beginner's Guide to Random Forest Hyperparameter Tuning*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Schall, R., & Weatherall, D. (2013). Accuracy and fairness of rain rules for interrupted one-day cricket matches. *Journal of Applied Statistics 40(11):* , 2462-2479.

Selvey, M. (2009). *England have last laugh as first one-dayer ends in farce*. Retrieved May 16, 2023, from https://www.theguardian.com/sport/2009/mar/20/west-indies-england-cricket-farce

Shah, H., Sampat, J., Savla, R., & Bhowmick, K. (2015). Review of Duckworth Lewis Method. *Procedia Computer Science*(45).

Shah, U. (2010, September 10). *The step-by-step approach using Decision Tree Modeling using Python*. Retrieved from Medium: https://medium.com/analytics-vidhya/the-step-by-step-approach-using-decision-tree-modeling-using-python-4ef1b4a3fe3

Sood, G., & Willis, D. (2016). *How much does the toss really matter?* Retrieved May 17, 2023, from https://www.espncricinfo.com/blogs/content/story/997931.html

Spyz, V. (2023). Uncover Cricket Legends: Cricketer's WikiData. Retrieved August 2023, from https://www.kaggle.com/datasets/varunnagpalspyz/uncover-cricket-legends-cricketers-wikidata

Stern, S. (2016). The Duckworth-Lewis-Stern method: extending the Duckworth-Lewis methodology to deal with modern scoring rates. *The Journal of the Operational Research Society, 6*7(12), 1469-1480.

Thevapalan, A., & Le, J. (2023, June). *Decision Tres in Machine Learning Using R*. Retrieved from Datacamp: https://www.datacamp.com/tutorial/decision-trees-R

Varma, A. (2004). *Simple and subjective? Or complex and objective?* Retrieved May 16, 2023, from https://www.espncricinfo.com/story/simple-and-subjective-or-complex-and-objective-141952

Vieira, R. (2023, September 1). *Optimising random forest hyperparameters*. Retrieved from Rui Vieira: https://ruivieira.dev/optimising-random-forest-hyperparamaters.html

Wilde, F. (2020). *The unstoppable rise of T20 cricket: from 'bit of fun' outsider to global juggernaut*. Retrieved May 18, 2023, from https://inews.co.uk/sport/cricket/t20-cricket-history-evolution-417072

Zhu, N., Zhu, C., Zhou, L., Zhu, Y., & Zhang, X. (2022). Optimization of the Random Forest Hyperparameters for Power Industrial Control Systems Intrusion Detection Using an Improved Grid Search Algorithm. *Appl. Sci*.

Appendix I Python Libraries

Below is a list of the main Python libraries used in this project.

Use	Library name	Description
General	numpy	To set seed and random sampling
	pandas	To create the dataframes
	json	Reading in JSONs
Data processing	sklearn.preprocessing(MinMaxScaler, StandardScaler)	Scaling data
	imblearn.over_sampling	SMOTE
Performance metrics	sklearn.metrics(classification_report, confusion_matrix, accuracy_score, hinge_loss, log_loss)	Assessing accuracy, confusion matrices
Visualisation	seaborn	Visualising box plots, correlation matrices
	matplotlib	Visualising confusion matrices and bar charts
Training models		
	sklearn.ensemble.RandomForestClassifier	Training RF model
	sklearn.feature_selection	ANOVA for hyperparameter importance for RF
	tensorflow.keras(Model, optimizers, input)	To train the Neural Network
	sklearn.svm.SVC	To train the SVC

Appendix II – Github Readme

DLS Project

Project to improve upon the predictive capability of DL method (2002 tables) using machine and deep learning methods and extending the definition of "resources" to include cricketer "value" as well as grounds data.

Data used

ODI cricket mens matches from Jan 2002 - Dec 2022

Cricketer data

Grounds data

Code

1.Data.ipynb Importing and formatting the data for the project (Match data, cricketer data and grounds data)

1a.Cricketer_Wiki.ipynb Code to import the cricketer wiki data

2.Visualisation_preprocessing Visualisation of the data, splitting into train val and test, processing target variable into categories and scaling and SMOTE data

data_formatting.py Code to take in train, val and test datasets and return X and y values. Also can perform sampling.

3.RF_model.ipynb Random Forest model

4.SVM_model.ipynb Support Vector Machine model

5.NN_mode.ipynb Feedforward Neural Network model