**Classifying the Users as Spenders and Non-Spenders**

**AIM**:

To classify whether the Users while gaming, will potentially spend money or not based on their activities in the game.

**ABSTRACT**:

We perform classification of Users as Spending Users vs Non Spending Users using 3 machine learning techniques.

1. Logistic regression. [1]

2. Recursive Partition Tree. [2]

3. Support Vector Machine.[3]

We chose the **RPART Partition tree Classifier as the best Classifier** among the three using the following metrics.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Classifier | Description | Precision | Recall | F1 Score | Area under  Curve | Mutual Info  Prediction Vs Truth |
| Recursive Partition Tree | CART - Decision Tree.[4] | **85.33%** | **80.86%** | **83.034%** | **0.7844852** | **0.1619491** |
| Logistic Regression | Sigmoid classifier | 85.962% | 77.44% | 81.085% | 0.8433 | 0.1528608 |
| Support Vector Machine | Max margin classifier. | 81.45% | 89.06% | 85.085% | 0.7347998 | 0.7347998 |

***Precision***

It is the number of Spenders correctly classified to the total number of Users Classified as “Spenders” by the Classifier. In other words, For Logistic Regression, Out of all the Users who are classified as “Spenders”, ***85.9% of them are actual Spenders***.

***Recall***

It is the number of Spenders correctly Classified to the total number of actual Spenders. Given a 100 actual spenders in the Data, the Recursive Partition Tree can capture up to 80 of them.

***F1Score***

It is a measure of classifier’s Accuracy. F1 score is a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

**Area Under Curve**

**I**t isequal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.[5]

**Mutual Info between the Predicted Values vs Truth:**

It showshow close they both are and how much info they share between each other. The higher is the better.[6]

**PURPOSE**:

1. The Gamers who spend money on games are our treasures.
2. We need to identify those potential spenders from our gamers to capture the future VIP customers.
3. We have to make sure these potential Spenders are having an enhanced and enriched gaming experience.
4. We need to know if we had made them feel the worth of the money they spent and they are encouraged to pay more. We need to understand their impressions, ideas, likes and dislikes of our game when they play.
5. The interaction logs obtained from the alliance chat rooms give a peek at what the players are communicating to each other.
6. By analyzing the relationship between the User expenditure and the User social interaction gives us a notion of what they feel about our games. Do they want to sustain the relationship/business with us? Do they find their time and money are worth spent? Do they feel excited for our new games /levels/changes/. Are they encouraged to spend more and buy more games from us?

**THEORY:**

**SUPERVISED LEARNING:**

Given is a set of input data, X (set of Independent variables) and its corresponding Output data, Y (Dependent variable). In our problem X corresponds to gamer’s attributes and Y corresponds to whether he is a spender or not. This set of input data with known output is called ***Training set***. [7] Our goal is, given a training set, to learn a function so that is a good predictor for the corresponding value of Y. The function **H** is called a ***hypothesis***. Supervised learning is the task of creating a mapping function from labeled input ***training*** data to Output. This mapping function can be used to predict Y’ for the new input X’.

**X**

**XX**

**LINEAR REGRESSION**:

To perform supervised learning, we need to represent the hypothesis H which maps the input to the output. Let us start designing the hypothesis as a linear function of input X. X is given by a set of independent variables.[8]

Here, ’s are the parameters (also called weights) parameterizing the space of linear functions mapping from X to Y. From the training set of data, we obtain the values of these weights. Using the training data, the ’s are updated from an arbitrary initial value till convergence based on the ***Euclidean distance*** between true value(Y) and the predicted value . Θ, X represents the corresponding vectors forms of parameters and the input features. We predict the values of Y using The regression assumes that the error between the hypothesis and the actual output follows *Normal Distribution*.

**CLASSIFICATION:**

When **Y** can take on only a small number of discrete values, we call it a Classification problem.[9] In other words, given a training set of input and output data, an inference function is created to map the input to the discrete output values using a learning algorithm, called the ***Classification algorithm.*** The inference function is called as a ***Classifier***. More formally, In a Classification problem, we have a training sample of ***N observations*** on a **class variable** that takes values , and  ***predictor variables****,* ,. Our goal is to find a model for predicting the values of Y from new values. In theory the solution is simply a partition of the X space into disjoint sets, ,, such that

For our problem, we will look into classification task for binary output variable (k =2 above).

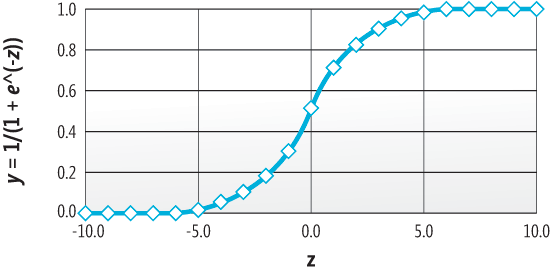
We will look at three learning algorithms.

**LOGISTIC REGRESSION:**

Logistic regression, also called a logit model, is used to model binary outcome variables (Spender or not) for the given set of input variables or predictor variables[1]. In the logit model, the ***log hypothesis of the outcome*** is modeled as a ***linear combination*** of the predictor variables. Let P(Y) is the probability that Y will occur, where Y is a binary object (Y/N, 1/0, true/false). Then

We define the hypothesis through a ***sigmoid function*** instead of a simple linear combination of input vectors.

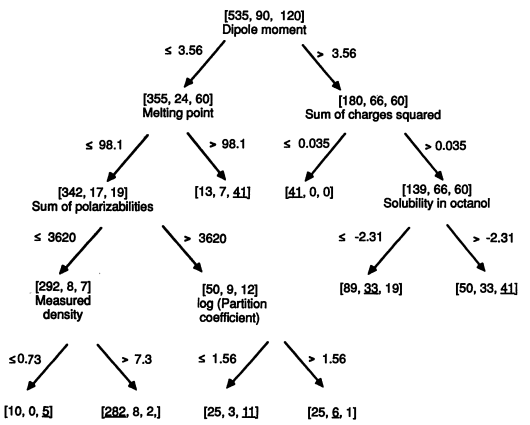
The sigmoid function simply separates the data into two halves the positive and the negative sides of the outcome variable. Using the training data, the is obtained by maximize the log likelihood of the outcome. The error between the hypothesis and the actual output is assumed to follow a ***Bernoulli distribution***. We minimize the error to maximize the likelihood of the outcome. Below is the sigmoid curve which transcends from 0 to 1 for a given integer input.



Please refer wiki for more details. Let us see the next type of classification Algorithm.

**CLASSIFICATION AND REGRESSION TREE (CART) & RECURSIVE PARTITION (RPART):**

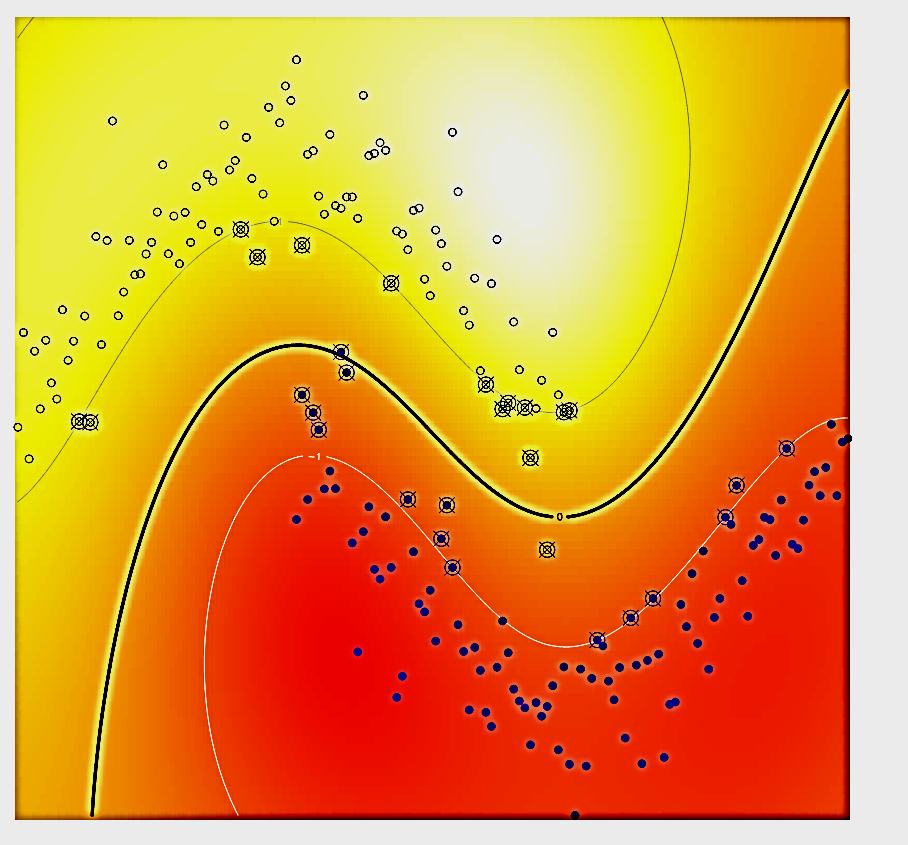
Classification and regression trees are machine-learning methods for constructing prediction models from data.[4] The models are obtained by recursively partitioning the data space and fitting a simple prediction within each partition. It resembles more like a decision tree. Classification trees are designed for dependent variables that take continuous or ordered discrete values, with prediction error typically measured by the squared difference between the observed and predicted values. Classification tree methods yield rectangular sets by recursively partitioning the data set one variable at a time. CART is implemented in R using a **Recursive Partition (RPART)** module. Below is a picture showing how decision trees look like. The ***rpart*** programs build classification or regression models of a very general structure using a two stage procedure; the resulting models can be represented as binary trees. **Rpart** recursively perform univariate splits of the dependent variable based on values on a set of covariates. Rpart employ information measures (such as the ***Gini coefficient***) for selecting the current covariate. **Decision trees assume** that our decision boundaries are parallel to the axes



Let us look at the third Classification Algorithm, Support Vector Machines.

**SUPPORT VECTOR MACHINE (SVM):**

Support Vector Machines (SVM’s) are a relatively new learning method used for binary classification[3]. The basic idea is to find a ***hyperplane*** which separates the ***d-dimensional data perfectly into its two classes.***  The notion is to separate the data using a decision surface (Hyperplane) with as maximum “***Margin***”{the distance between the nearest data point to the decision boundary”}.However, since example data is often not linearly separable, SVM’s introduce the notion of a “***kernel induced feature space***” which casts the data into a higher dimensional space where the data is separable. This method of construction necessarily means that the decision function for an SVM is fully specified by a (usually small) subset of the data which defines the position of the separator. SVMs don’t rely on ALL of the underlying data to train the model. These points are referred to as the ***support vectors*** (in a vector space, a point can be thought of as a vector between the origin and that point). SVMs are black-box models; It’s difficult to learn anything about a problem by looking at the parameters from a fitted SVM Model. However, SVM’s are intuitive, theoretically well- founded, and have shown to be practically successful. SVM’s have also been extended to solve regression tasks (where the system is trained to output a numerical value, rather than “yes/no” classification). Below Is a picture of Support Vector machine with highlighted **non-linear decision surface**(Hyperplane) and rounded Support vector which aid the decision surface.



**DECISION BOUNDARY**

**SUPPORT VECTORS**

**DATA POINTS**

**PRIMARY DATA :** User\_data.txt, attacks.txt and alliance\_chat.txt.

**DERIVED DATA:** They are secondary data derived from the primary sources.

* User Performance Metrics: User scores, Match Results, Alliance association, match frequencies.
* Alliance Performance Metrics: Alliance Scores, Match results.
* Knight Performance Metrics: Knight Scores, Match results, match frequencies.
* User Social Metrics: User Sentiments, User subjectivity, chat counts.
* User Delight Metrics: Loots, Boosts, Mights, Wall Strengths & XPs.

**DATA DISTRIBUTION:**

* Out of **10215693** User Ids who are registered in User\_data, only **506,707** are the number of User id correspond to Users who spend money on games.
* Very few of the User’s attack activity are given by the two months data of attacks.txt namely **41621** Defending users and **2261** attacking Users**. 2168** users are both Defenders and attackers.
* The alliance\_chat.txt holds only messages from **6069** distinct Users.
* We compiled a list of **5387** Users from this bunch and used it for Classification of Users.

**TRAINING AND TESTING DATA**:

**Training Data:** Feature Set of 4000 Users from total of 5387 User data.

**Testing Data:** Feature Set of remaining 1387 Users.

**FEATURE SET**: The following are the feature set we choose the data in order to determine whether the User is potential Spender or Not.

1. *UserID*: Userid is an increasing number. Usually IDs are not used as feature set. But surprisingly there is a strong correlation between this number and the user’s behavior for this data set.
2. *LoginCount:* the number of logins made by the user shows how interested he is towards gaming.
3. *UserWinCount*: The number of matches won by the User.
4. *UserLostCount:* The number of matches lost by the User.
5. *UserDrawCount:* the number of matches drawn by the User.
6. *UserDefenseMight*: the scaled might of the User while defending in matches.
7. *UserAttackMight*: The scaled might of the user while he is attacking.
8. *UserDefenseScore*: The Score indicating the defense strength of the user.
9. *UserAttackScore*: The Score indicating the attack strength of the user.
10. *UserAvgDefenceScore*: The average defense score of the User across the matches.
11. *UserAvgAttackScore*: the average attacks score of the User across the matches.
12. *UserKnightWin*: Number of matches won by the user as knight.
13. *UserKnightLost*: Number of matches lost by the user as a knight.
14. *UserKnightDrawCount*: Number of matches drawn by the user as Knight.
15. *UserKnightDef*: The defense score of the user as a knight.
16. *UserKnightAtk*: The attack score of the user as knight.
17. *UserScore* : DefenceScore+AttackScore+WinCount+LostCount+ 0.5\*B.DrawCount+ KnightDefScore+ KnightAtkScore
18. *num\_alliances*: Number of alliances associated by the user.
19. *AllianceAvgWin*: Average number of matches won by the alliance per user.
20. *AllianceTotalWin*: Total number of matches won by the alliances.
21. *AllianceAvgLost*: Average number of matches lost by the alliances of the user.
22. *AllianceTotalLost*:Total number of matches lost by the alliances of the user.
23. *AllianceTotalDraw*: Total number of matches drawn by the alliances of the user.
24. *AllianceAvgDraw:* Average number of matches drawn by the alliances of the user.
25. *AllianceSumDefense:* Sum of Defense scores of the users in the alliance.
26. *AllianceSumAttack*: Sum of Attack scores of the users in the alliance.
27. *AvgAtkPerDay*: Average number of attacks done by the user per day.
28. *AvgDefPerDay*:Average number of defense done by the user per day.
29. *TotalSentiment*: Sum of all sentiments of the user’s chat messages.
30. *AvgSentiment*: Avearage of all sentiments of the user’s chat messages.
31. *TotalSubjectivity*: Sum of all subjectivity scores of the user’s chat messages.
32. AvgSubjectivity:Average of all subjectivity scores of the user’s chat messages.
33. *Msgcount*: Number of chat messages per day by a user.
34. *DailyAvgWordCount*: Daily number of words given by the user in the chat messages.
35. *SumAtkBoost*:Sum of all attack boosts taken by the user.
36. *AvgAtkBoost*: Average of all attack boosts taken by the user.
37. *SumDefBoost*: sum of all defense boosts taken by the user.
38. *AvgDefBoost*: Average of all defense boosts taken by the user.
39. *TotalWall:* Sum of all wall strengths of the defending user during matches.
40. *AvgWall*: Average of all wall strengths of the defending user during matches.
41. *SumXP*:Sum of all XP earned by the user in matches.
42. *AvgLoot*: Average of all the Loot of resources{ Gold , Food, Wood, Ore, Stone}
43. *SumLoot*: Sum of all the loot of the resources{ gold, Food, Wood, Ore, Stone}
44. **Spender : Truth data which consists of ‘1’ if the user has spent money on game and ‘0’ if he has not.**

Machine Learning Language used**: R**

**DATA ANALYSIS:** In order to find the most important features to consider to predict whether the user is a spender or not, we used Correlation Coefficient and Mutual Information between the features and the Spender Truth data.

|  |  |  |
| --- | --- | --- |
| FEATURE | CORRELATION COEFF | MUTUAL INFORMATION |
| userID | -0.4306 | 0.520697 |
| UserScore | 0.008784 | 0.520697 |
| LoginCount | 0.220188 | 0.389121 |
| AvgSubjectivity | 0.264362 | 0.380692 |
| AvgSentiment | 0.200167 | 0.376659 |
| TotalSubjetivity | 0.161757 | 0.351221 |
| TotalSentiment | 0.181493 | 0.34914 |
| AllianceSumDefense | 0.101966 | 0.259243 |
| AllianceSumAttack | -0.16439 | 0.230302 |
| AllianceSumDraw | 0.255756 | 0.229438 |
| AllianceSumLost | 0.258181 | 0.216288 |
| AllianceTotalWin | 0.256607 | 0.207764 |
| AllianceAvgDraw | 0.245934 | 0.20639 |
| AllianceAvgWin | 0.242546 | 0.193197 |
| UserAttackScore | -0.08127 | 0.181405 |
| AllianceAvgLost | 0.246871 | 0.179748 |
| UserDrawCount | 0.164403 | 0.14226 |
| Msgcount | 0.143144 | 0.124195 |
| UserDefenseScore | 0.142411 | 0.118032 |
| SumLoot | 0.099987 | 0.117324 |
| UserWinCount | 0.116691 | 0.098211 |
| TotalWall | 0.093854 | 0.093327 |
| UserLostCount | 0.215442 | 0.091884 |
| SumXP | 0.087611 | 0.06869 |
| AvgAtkPerDay | 0.209129 | 0.053682 |
| AvgDefPerDay | -0.01227 | 0.040723 |
| DailyAvgTalkCount | 0.141644 | 0.030281 |
| num\_alliances | 0.138909 | 0.025928 |
| AvgWall | 0.105515 | 0.020193 |
| UserAvgAttackScore | -0.04221 | 0.016185 |
| SumAtkBoost | 0.049099 | 0.013255 |
| SumDefBoost | 0.044089 | 0.011894 |
| AvgLoot | 0.081408 | 0.007526 |
| UserAvgDefenceScore | 0.065963 | 0.004224 |
| UserKnightWin | 0.015817 | 0.003782 |
| UserKnightAtk | -0.02465 | 0.00371 |
| UserDefenseMight | 0.007638 | 0.002634 |
| UserKnightDrawCount | 0.030976 | 0.002316 |
| UserKnightLost | 0.015288 | 0.00067 |
| UserKnightDef | 0.010531 | 0.00039 |
| UserAttackMight | 0.196328 | 0 |
| AvgAtkBoost | 0.078871 | 0 |
| AvgDefBoost | 0.083525 | 0 |

We added quadratic features by hardcoding combinations of features to improve the accuracy of models.

**LOGISTIC REGRESSION:**

We implemented Logistic Regression using R by using a subset of the above feature set[10]. Below is the summary of the Classifier:

***Deviance:***

It is a measure of how different is the predicted output compared to the true output.

***Deviance Residuals***

It show the distribution of the residual error among the data points.

***Columns Estimate, Std.Error, Z value and Pr(>|z|)***

It specifies how appropriate the coefficients are for each of the feature we have chosen. Stars show how significant are the coefficients using Z test.

***The Null Deviance***

It specifies how much deviance the predicted data produce from the output if only the Intercept is used for predicting the output.

***Residual Deviance***

It specifies how much deviance the predicted data produce from the output if features and the Intercept are used for predicting the output. The better the difference between the Residual Deviance and Null Deviance, the better the classifier.

**Degrees of Freedom**

In statistics, the number of degrees of freedom is the number of values in the final calculation of a statistic that are free to vary.

**AIC Akaike Information Criterion**

The Akaike information criterion (AIC) is a measure of the relative quality of a statistical model for a given set of data. The higher the better.

Call:

glm (formula = Spender ~ userID + LoginCount + UserAttackScore \* UserDrawCount +

UserAttackScore \* UserLostCount + UserLostCount + UserDrawCount +

UserAttackMight + UserDefenseScore + UserAttackScore +

UserAvgDefenceScore +UserAvgAttackScore + UserKnightWin +

UserKnightLost + UserKnightDrawCount + UserKnightDef + UserScore +

AvgSubjectivity \* AvgSubjectivity +AllianceTotalWin +

AllianceAvgWin + AllianceAvgLost +AllianceSumDraw +

AllianceAvgDraw + AllianceSumDefense +

UserAttackScore \* UserAttackScore + AllianceSumAttack +

AvgAtkPerDay + AvgAtkPerDay \* AvgAtkPerDay +AvgSubjectivity +

msgcount + UserAttackScore \* UserKnightLost + AvgAtkBoost +

SumXP + SumDefBoost + AvgDefBoost + TotalWall + AvgWall \*AvgWall +

AvgWall + AvgLoot + SumLoot,

family = binomial,data = train)

Deviance Residuals: Min 1Q Median 3Q Max

-3.6322 0.0930 0.3525 0.5973 1.6205

Coefficients of the features

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Coefficients: |  |  |  |  |  |
| Features | Estimate | Std.Error | z value | Pr(>|z|) | Significance |
| (Intercept) | 7.42E-01 | 2.35E-01 | 3.166 | 0.00155 | \*\* |
| userID | -1.97E-07 | 1.76E-08 | -11.219 | <2.E-16 | \*\*\* |
| LoginCount | 8.67E-05 | 1.59E-05 | 5.446 | 5.15E-08 | \*\*\* |
| UserAttackScore | -3.84E-04 | 1.51E-04 | -2.544 | 0.01095 | \* |
| UserDrawCount | -1.80E-03 | 4.24E-04 | -4.254 | 2.10E-05 | \*\*\* |
| UserLostCount | 5.63E-03 | 1.87E-03 | 3.014 | 0.00258 | \*\* |
| UserAttackMight | -1.51E-01 | 2.15E-01 | -0.701 | 0.48342 |  |
| UserDefenseScore | 5.20E-04 | 4.18E-04 | 1.245 | 0.21306 |  |
| UserAvgDefenceScore | 2.23E-02 | 4.07E-02 | 0.547 | 0.58416 |  |
| UserAvgAttackScore | -7.14E-03 | 2.52E-02 | -0.283 | 0.77716 |  |
| UserKnightWin | 1.94E-02 | 2.60E-02 | 0.747 | 0.455 |  |
| UserKnightLost | -2.63E-01 | 1.61E-01 | -1.63 | 0.10311 |  |
| UserKnightDrawCount | 4.60E-02 | 4.59E-02 | 1.001 | 0.31686 |  |
| UserKnightDef | 2.66E-02 | 6.13E-02 | 0.433 | 0.66485 |  |
| UserScore | -1.56E-05 | 3.47E-05 | -0.45 | 0.65283 |  |
| AvgSubjectivity | 3.08E-04 | 4.07E-05 | 7.567 | 3.83E-14 | \*\*\* |
| AllianceTotalWin | 9.45E-06 | 1.23E-05 | 0.771 | 0.44085 |  |
| AllianceAvgWin | -1.14E-05 | 2.82E-05 | -0.404 | 0.68648 |  |
| AllianceAvgLost | 1.21E-04 | 7.35E-05 | 1.645 | 0.10002 |  |
| AllianceSumDraw | -1.08E-05 | 7.96E-06 | -1.354 | 0.17579 |  |
| AllianceAvgDraw | 1.68E-05 | 1.98E-05 | 0.849 | 0.39567 |  |
| AllianceSumDefense | 6.73E-05 | 8.94E-05 | 0.753 | 0.45146 |  |
| AllianceSumAttack | -1.00E-04 | 1.20E-04 | -0.834 | 0.4042 |  |
| AvgAtkPerDay | 4.73E-02 | 8.57E-03 | 5.515 | 3.49E-08 | \*\*\* |
| msgcount | 2.95E-04 | 1.38E-04 | 2.141 | 0.03228 | \* |
| AvgAtkBoost | 1.39E+01 | 9.43E+00 | 1.474 | 0.14056 |  |
| SumXP | -3.46E-05 | 3.02E-05 | -1.146 | 0.25174 |  |
| SumDefBoost | 1.21E-02 | 2.49E-02 | 0.485 | 0.62797 |  |
| AvgDefBoost | 2.12E+01 | 2.19E+01 | 0.971 | 0.33161 |  |
| TotalWall | -1.14E-05 | 1.79E-05 | -0.636 | 0.52449 |  |
| AvgWall | 3.16E-03 | 2.21E-03 | 1.429 | 0.15292 |  |
| AvgLoot | -1.69E-02 | 8.62E-03 | -1.955 | 0.05059 | . |
| SumLoot | 2.86E-05 | 3.41E-05 | 0.84 | 0.40075 |  |
| UserAttackScore:UserDrawCount | -5.12E-08 | 5.75E-08 | -0.89 | 0.37339 |  |
| UserAttackScore:UserLostCount | -1.79E-07 | 8.41E-07 | -0.213 | 0.83138 |  |
| UserAttackScore:UserKnightLost | 3.74E-04 | 2.30E-04 | 1.627 | 0.10384 |  |

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Null deviance: 3550.3 on 3999 degrees of freedom

Residual deviance: 2784.8 on 3964 degrees of freedom

AIC: 2856.8 Number of Fisher Scoring iterations: 10

**Analysis of Deviance Table**

The function will show the change in deviance obtained by adding each of the terms in the order listed in the model formula. Every step shows an improvement in the Residual Deviance and the total degrees of Freedom.

Analysis of Deviance Table Model: binomial, link: logit

Response: Spender Terms added sequentially (first to last)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Features | Df | Deviance | Residual Df | Residual Dev |
| NULL |  |  | 3999 | 3550.3 |
| userID | 1 | 312.943 | 3998 | 3237.4 |
| LoginCount | 1 | 66.042 | 3997 | 3171.4 |
| UserAttackScore | 1 | 7.316 | 3996 | 3164 |
| UserDrawCount | 1 | 68.478 | 3995 | 3095.6 |
| UserLostCount | 1 | 46.239 | 3994 | 3049.3 |
| UserAttackMight | 1 | 3.705 | 3993 | 3045.6 |
| UserDefenseScore | 1 | 9.3 | 3992 | 3036.3 |
| UserAvgDefenceScore | 1 | 1.403 | 3991 | 3034.9 |
| UserAvgAttackScore | 1 | 2.21 | 3990 | 3032.7 |
| UserKnightWin | 1 | 0.222 | 3989 | 3032.5 |
| UserKnightLost | 1 | 1.444 | 3988 | 3031 |
| UserKnightDrawCount | 1 | 3.717 | 3987 | 3027.3 |
| UserKnightDef | 1 | 0.567 | 3986 | 3026.8 |
| UserScore | 1 | 1.268 | 3985 | 3025.5 |
| AvgSubjectivity | 1 | 115.897 | 3984 | 2909.6 |
| AllianceTotalWin | 1 | 12.567 | 3983 | 2897 |
| AllianceAvgWin | 1 | 25.576 | 3982 | 2871.4 |
| AllianceAvgLost | 1 | 2.894 | 3981 | 2868.6 |
| AllianceSumDraw | 1 | 0.372 | 3980 | 2868.2 |
| AllianceAvgDraw | 1 | 0.884 | 3979 | 2867.3 |
| AllianceSumDefense | 1 | 0.549 | 3978 | 2866.7 |
| AllianceSumAttack | 1 | 0.504 | 3977 | 2866.2 |
| AvgAtkPerDay | 1 | 45.804 | 3976 | 2820.4 |
| msgcount | 1 | 7.88 | 3975 | 2812.6 |
| AvgAtkBoost | 1 | 9.996 | 3974 | 2802.6 |
| SumXP | 1 | 3.232 | 3973 | 2799.3 |
| SumDefBoost | 1 | 4.749 | 3972 | 2794.6 |
| AvgDefBoost | 1 | 2.224 | 3971 | 2792.4 |
| TotalWall | 1 | 0.195 | 3970 | 2792.2 |
| AvgWall | 1 | 0.489 | 3969 | 2791.7 |
| AvgLoot | 1 | 3.271 | 3968 | 2788.4 |
| SumLoot | 1 | 0.795 | 3967 | 2787.6 |
| UserAttackScore:UserDrawCount | 1 | 0.778 | 3966 | 2786.8 |
| UserAttackScore:UserLostCount | 1 | 0.054 | 3965 | 2786.8 |
| UserAttackScore:UserKnightLost | 1 | 1.941 | 3964 | 2784.8 |

**METRICS FOR MEASURING ACCURACY**

**Confusion Matrix**

It is a contingency table or an error matrix that allows visualization of the performance of an algorithm

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | TESTING\_TRUTH | |
|  |  | Non-Spender | Spender |
| TEST\_PREDICTION | Non-Spender | 398(TN) | 198(FN) |
| Spender | 111 (FP) | 680 (TP) |

TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives.

**Precision**

It is defined as the ratio of number of positive values correctly classified to the total number of positive values. In our problem, it is the ratio of number of Users correctly classified as Spenders to the total number of Users classified as Spenders.**True Positive Rate or Recall**

It is the ratio of number of positive values correctly classified vs the total number of positive values. In our problem, it is the ratio of number of Users correctly classified as Spenders vs the total number of Spenders.

**Accuracy**

It is the ratio of total number of correctly classified items to the total number of items classified. In our problem, it corresponds to number of Users correctly classified as Spenders and Non-Spenders to the total number of users classified.

**F1 Score**

It is a measure of classifier’s Accuracy. F1 score is a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

**Specificity**

It is the proportional of correctly classified negative values to the total number of negative items.

**False Positive Rate or Fall-out**

It is the proportion of incorrectly classified negative values to the total number of negative items. It is simply 1-specificity.

**Mutual Information**

Apparently the mutual information shared by the predicted test results and the actual testresults is good measure of classifier’ performance.

**ROC Curve**

In signal detection theory, a ***Receiver Operating Characteristic*** (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the total actual positives (**Recall**) vs. the fraction of false positives out of the total actual negatives (**Fall Out**), at various threshold settings.

**Area Under Curve** (**AUC**)

The area under the ROC curve is a measure of the performance of the Classifier. The higher, the better the classifier. Area Under curve for this classifier = 0.8433

|  |  |  |
| --- | --- | --- |
| **Metric** | **Formula** | **Value** |
| Precision | TP/(TP+FP) | 85.94% |
| True Positive Rate or Recall | TP/(TP+FN) | 77.44% |
| Accuracy |  | 77.72% |
| F1 Score |  | 81.015% |
| Specificity |  | 71.19% |
| False Positive Rate or Fall-Out |  | 21.80% |
| Mutual Information  (Test Prediction , Test truth) | I(X;Y) = \sum_{y \in Y} \sum_{x \in X}                   p(x,y) \log{ \left(\frac{p(x,y)}{p(x)\,p(y)}                               \right) }, \,\! | 0.1528608 |
| Area Under the Curve | Area under the ROC curve | 0.8433 |

**ROC Curve for Logistic Regression**

(FPR =0.218, TPR= 0.774)

The peak performance of the classifier is obtained when the False positive rate is around **0.2180** and the True positive rate is **0.7745**.

**CLASSIFICATION USING RPART TREE:**

We performed decision tree classification using ***rpart*** package from R. [2].

rpart(formula = Spender ~ UserWinCount + UserLostCount + UserDrawCount +

UserAttackMight + UserDefenseScore + UserAttackScore +

UserAvgDefenceScore + UserKnightLost + UserKnightDrawCount +

UserKnightDef + UserKnightAtk + num\_alliances + AllianceTotalWin +

AllianceAvgLost + AllianceSumDraw + AllianceAvgDraw +

AllianceSumDefense + AllianceSumAttack + AvgAtkPerDay +

TotalSubjetivity + AvgSubjectivity + msgcount +

SumAtkBoost + AvgAtkBoost + SumXP + userID + LoginCount +

AvgDefBoost + AvgWall + SumXP + AvgLoot + SumLoot,

data = train,

control = rpart.control(xval = 10, minbucket = 35, cp = 0.00015))

***Cross Validation (Xval)***

It is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

**Minbucket**

The minimum number of observations in any leaf node in the decision tree.

**Complexity Parameter (CP)**

Any split that does not decrease the overall lack of fit by a factor of CP is not attempted.

CP nsplit rel error xerror xstd

1 0.087692308 0 1.0000000 1.0000000 0.03589515

2 0.049230769 1 0.9123077 0.9538462 0.03521363

3 0.024615385 2 0.8630769 0.8953846 0.03430869

4 0.004615385 3 0.8384615 0.8676923 0.03386279

5 0.001923077 5 0.8292308 0.8907692 0.03423517

6 0.001538462 9 0.8215385 0.8953846 0.03430869

7 0.000150000 14 0.8138462 0.9030769 0.03443054

**Xerror**

It is Cross validation error. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds. It is computed using a 10-fold cross-validation. If your misclassification costs are uniform, an xerror value of 0.9 means that  the misclassification rate is 0.9 times the misclassification rate of the trivial tree with no splits.  It should be easy to calculate the rate of the trivial tree, because it assigns all cases to the same class -- the class that minimizes the rate. In general, xerror is computed from the misclassification \*risk\*, which takes into account the loss matrix.

**Xstd**

The standard deviation of the Cross validation error.

**Nsplit**

Number of splits resulting from that feature**.**

The following are the Predictive variables and their importance

Variable importance

LoginCount 45

userID 9

AvgSubjectivity 8

UserLostCount 6

TotalSubjetivity 4

AllianceAvgLost 3

AllianceAvgDraw 3

UserDefenseScore 3

AllianceSumDraw 3

UserDrawCount 3

SumXP 2

msgcount 2

SumLoot 2

AvgAtkPerDay 2

AllianceTotalWin 2

UserWinCount 1

AllianceSumAttack 1

UserAttackScore 1

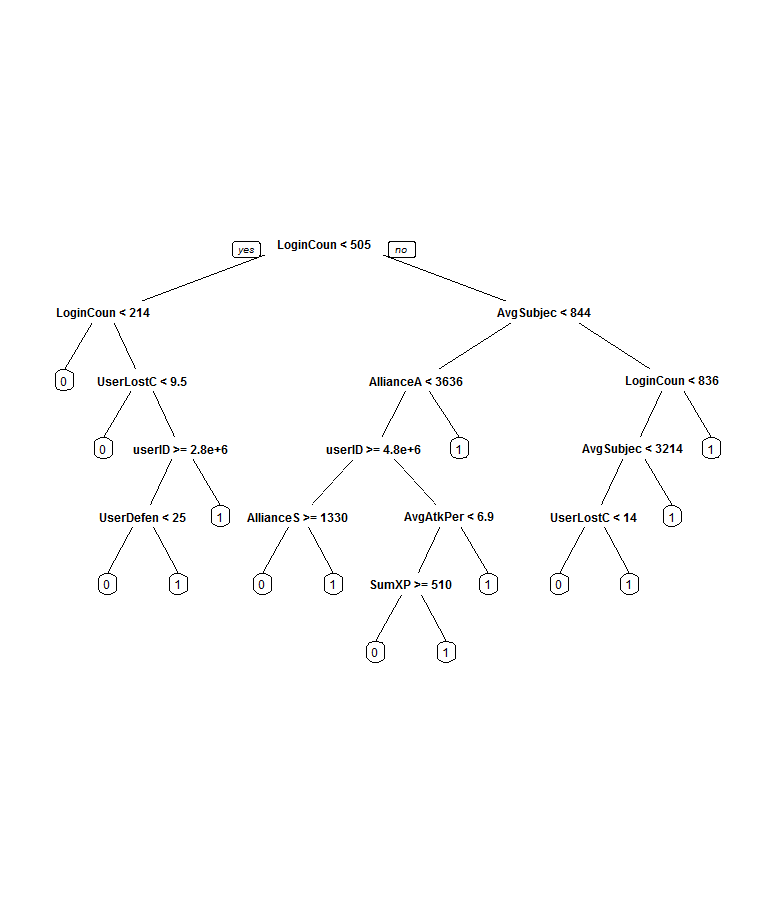
UserAttackMight 1

AvgLoot 1

Below is the Decision tree which shows the division performed by the model to classify the users into **Spender (1) and**

**Non-Spender (0)**

**DECISION TREE FOR RECURSIVE PARTITION TREE**



**Metrics of Performance for RPART Tree Classifier:**

**Confusion Matrix**:  Visualization of the performance of an algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | TESTING\_TRUTH | |
|  |  | Non-Spender | Spender |
| TEST\_PREDICTION | Non-Spender | 387(TN) | 168(FN) |
| Spender | 122 (FP) | 710 (TP) |

TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives.

|  |  |  |
| --- | --- | --- |
| **Metric** | **Formula** | **Value** |
| Precision | TP/(TP+FP) | 85.33% |
| True Positive Rate or Recall | TP/(TP+FN) | 80.86% |
| Accuracy |  | 79.09% |
| F1 Score |  | 83.034% |
| Specificity |  | 76.03% |
| False Positive Rate or Fall-Out |  | 23.96% |
| Mutual Information  (Test Prediction , Test truth) | I(X;Y) = \sum_{y \in Y} \sum_{x \in X}                   p(x,y) \log{ \left(\frac{p(x,y)}{p(x)\,p(y)}                               \right) }, \,\! | 0.1619491 |
| Area Under the Curve | Area under the ROC curve | 0.7844852 |

**ROC Curve for Recursive Partition Tree on Test Data:**

The RPART Classifier gives best performance where the false positive rate is around **24%** and the true positive rate is **80%**.

**Classification Using Support Vector Machines:**

We used **kernlab** module in R to perform svm classification. [11]

ksvm(Spender ~ userID + LoginCount + UserAttackScore\*UserDrawCount + UserAttackScore\*UserLostCount +

UserLostCount + UserDrawCount + UserAttackMight + UserDefenseScore + UserAttackScore +

UserAvgDefenceScore + UserAvgAttackScore + UserKnightWin + UserKnightLost +

UserKnightDrawCount + UserKnightDef + UserScore + AvgSubjectivity\*AvgSubjectivity +

AllianceTotalWin + AllianceAvgWin + AllianceAvgLost + AllianceSumDraw + AllianceAvgDraw +

AllianceSumDefense + UserAttackScore\*UserAttackScore+ AllianceSumAttack + AvgAtkPerDay +

AvgAtkPerDay\*AvgAtkPerDay + AvgSubjectivity + msgcount + UserAttackScore\*UserKnightLost +

AvgAtkBoost + SumXP + SumDefBoost + AvgDefBoost + TotalWall + AvgWall\*AvgWall+ AvgWall +

AvgLoot + SumLoot,

data=train,

kernel='rbfdot',

kpar='automatic',

C=320,

cross=15,

prob.model=TRUE,

type='nu-svc')

***Kernel***

The higher dimensional nonlinear surface into which the data points are mapped in order to perform maximum margin classification. Here the Kernel value is “rbfdot”. Which refer to Gaussian ***Radial Basis Function***.

**C**

Cost of the constraint parameter for ***Regularization*** { a way to simplify the feature set if the number of features blows up} term used in ***Lagrange’s formulation {***An technique used to solve mathematical constrained optimization problems)

**Cross (** Cross Validation parameter)

It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

**Type**

SVM can also be used for Regression and Outlier detection in addition to classification. This value indicates it.

**Kpar**

The value’automatic’ assumes the parameters {sigma or the ***inverse kernel width*** , which affects nonlinearity of the Gaussian Radial basis function}. For small values of sigma the decision boundary is nearly linear. As sigma increases the flexibility of the decision boundary increases. Large values of γ lead to **over fitting** {the phenomenon where the model fits the training data too tightly that it fail to fit the testing data}.

Support Vector Machine object of class "ksvm"

SV type: nu-svc (classification)

parameter : nu = 0.2

Gaussian Radial Basis kernel function.

Hyperparameter : sigma = 0.0347176951870791

Number of Support Vectors : 1588

Objective Function Value : 38871.75

Training error : 0.04075

Cross validation error : 0.194005

Probability model included.

**Nu**

The nu parameter sets the upper bound on the training error and the lower bound on the fraction of data points to become Support Vectors

**Training Error**

The amount of deviation shown by the predicted value for the training feature to the actual training output.

**Number of Support Vectors**

This value shows how divisible or indivisible close/crowded are the data from two different classes are located. The higher this value the crowded is the data from two classes and more difficult it is to cleanly classify them apart.

**Metrics to Evaluate Performance of the classifier**

**Confusion Matrix**:

Contingency table or an error matrix, that allows visualization of the performance of an algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | TESTING\_TRUTH | |
|  |  | Non-Spender | Spender |
| TEST\_PREDICTION | Non-Spender | 331(TN) | 96(FN) |
| Spender | 178 (FP) | 782 (TP) |

TP: True Positive TN:True Negative: FP:False Positive, FN: False Negative.

|  |  |  |
| --- | --- | --- |
| **Metric** | **Formula** | **Value** |
| Precision | TP/(TP+FP) | 81.45% |
| True Positive Rate or Recall | TP/(TP+FN) | 89.06% |
| Accuracy |  | 80.24% |
| F1 Score |  | 85.085% |
| Specificity |  | 65.03% |
| False Positive Rate or Fall-Out |  | 34.97% |
| Mutual Information  ( Test Prediction , Test truth ) | I(X;Y) = \sum_{y \in Y} \sum_{x \in X}                   p(x,y) \log{ \left(\frac{p(x,y)}{p(x)\,p(y)}                               \right) }, \,\! | 0.1113471 |
| Area Under the Curve | Area under the curve | 0.7347998 |

**ROC Curve for Support Vector Machine for test Data :**

The above graph gives the notion of a ROC curve with peak performance at **33.7%** False positive rate and **80%** True Positive Rate. Plotting all the Classifier ROC graph into a single graph we have:

Logistic regression Classifier has best amount of area under the curve.

Among the Classifiers the **best option would be to choose RPART classifier**. RPART Partition tree Classifier is

* Great with Precision(reasonably) , Specificity, Mutual Information between the Predicted/Truth.
* Good with Recall, F1 score,Area Under Curve.

**CONCLUSION:**

We perform classification of Users as Spending Users vs Non Spending Users using 3 machine learning techniques.

1. Logistic regression.

2. Recursive Partition Tree.

3. Support Vector Machine.

We chose the **RPART Partition tree Classifier as the best Classifier** among the three using the following metrics.

RPART Partition tree Classifier is

* Great with Precision(reasonably) , Specificity, Mutual Information between the Predicted/Truth.
* Good with Recall, F1 score,Area Under Curve.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Classifier | Description | Precision | Recall | F1 Score | Area under  Curve | Mutual Info  Prediction Vs Truth |
| Recursive Partition Tree | CART - Decision Tree. | **85.33%** | **80.86%** | **83.034%** | **0.7844852** | **0.1619491** |
| Logistic Regression | Sigmoid classifier | 85.962% | 77.44% | 81.015% | 0.8433 | 0.1528608 |
| Support Vector Machine | Max margin classifier. | 81.45% | 89.06% | 85.085% | 0.7347998 | 0.7347998 |

**EXTRA TRIALS:**

We also tried with Conditional Inference Trees using R package[12].

* Conditional Inference trees (CTree) is non-parametric class of regression trees embedding tree-structured regression models into a well defined theory of conditional inference procedures.
* The goodness of a split is evaluated by two-sample linear statistics which are special cases of the linear statistic.
* Compare that to the Gini coefficient used by the RPART trees for the next split.

**Result:**

The Precision we got is upto to **84.7%.** But it could not be completed in time.

Ctree were not giving consistent for the same parameters as it is parameter less which is making it less reliable.

**REFERENCES:** Wikipedia is a good resource for most of the topics. In addition to that we have

1. <http://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf>
2. http://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf
3. http://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html
4. <http://www.stat.wisc.edu/~loh/treeprogs/guide/wires11.pdf>
5. http://gim.unmc.edu/dxtests/roc3.htm
6. <http://www.scholarpedia.org/article/Mutual_information>
7. http://www.mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html
8. http://onlinestatbook.com/2/regression/intro.html
9. <http://www.cs.princeton.edu/~schapire/talks/picasso-minicourse.pdf>
10. <http://data.princeton.edu/R/glms.html>
11. <http://www.jstatsoft.org/v15/i09/paper>
12. http://cran.r-project.org/web/packages/partykit/vignettes/ctree.pdf