



PROJECT REPORT

CSCI-527 Advanced Databases and Data Mining (League of Legends – Win/Loss Classification/Prediction)

By

Mohammed Abdul Aziz Syed	50143871
Paul Mandrumaka	50137144
Siva Prasad Bhushetty	50145947
Shoebuddin Ahmed Fnu	50144855

Professor

Dr. Truong Huy Nguyen (Ph.D.)

**(Department of Computer Science and Information System,
Texas A & M University –Commerce)**

Introduction:

Data mining is a relatively young and interdisciplinary field of computer science, is the process that attempts to discover patterns in large data sets. It utilizes methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use (“data mining”, Wikipedia).

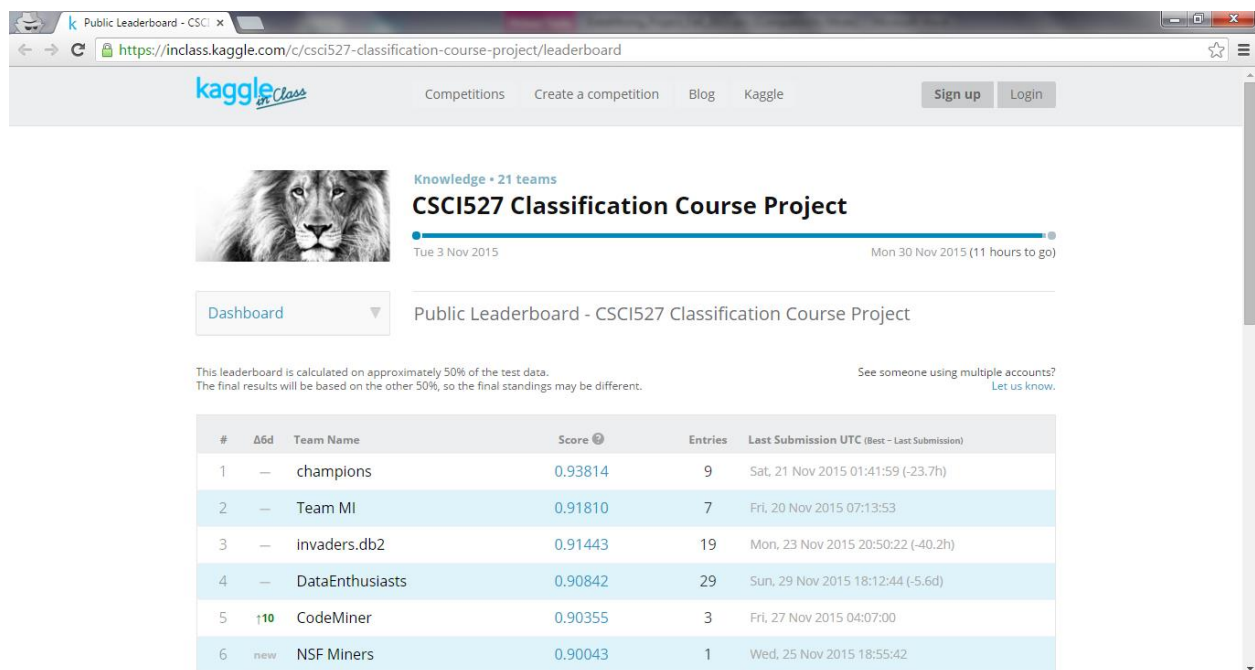
League of Legends (LoL) by Riot Games is one of the most played eSports in the world at the moment. Arguably, the fun aspect of games or sport lies within the uncertainty of their outcomes. There is nothing more boring than playing or watching a game, be it soccer, basketball, or any video games, with a predictable ending. This is why La Liga (the premier soccer league in Spain), despite having two of the best soccer players of the modern time Lionel Messi and Cristiano Ronaldo, is widely considered to be less entertaining than her counterpart Premier League in England. The reason is because Barcelona and Real Madrid are usually classes above the rest of the league. To make La Liga more fun to watch, there must be a way to close the skill gaps among teams. Similarly, the creators of LoL have tried their best to match players with teammates and opponents of as similar skill level as possible.

In this project, we have addressed a problem that Riot is highly interested in, i.e., predicting win/loss outcomes, by examining 7 Million match records of solo gamers. Each record comprises of all publicly available game statistics of a match played by some gamer, including an important field called "winner". If "winner" is true, the gamer in question won the match, otherwise, he/she lost. Our task is to create a classifier that takes as input one such match record, with the "winner" field left out, and labels this record as a "win" or a "loss". The test set comprises of ~770K such records.

This is the final project report. Earlier, the predictions made by DataEnthusiasts team were roughly 90.043% accurate.

However we fine-tuned our prediction techniques, which are described below. These refinements made us achieve 90.842% accuracy this time. Thereby we beat our own record on kaggle public leaderboard. Results can be viewed at <https://inclass.kaggle.com/c/csci527-classification-course-project/leaderboard>.

Here is the proof of submission and improvement



#	Δ6d	Team Name	Score 🏆	Entries	Last Submission UTC (Best - Last Submission)
1	—	champions	0.93814	9	Sat, 21 Nov 2015 01:41:59 (-23.7h)
2	—	Team MI	0.91810	7	Fri, 20 Nov 2015 07:13:53
3	—	invaders.db2	0.91443	19	Mon, 23 Nov 2015 20:50:22 (-40.2h)
4	—	DataEnthusiasts	0.90842	29	Sun, 29 Nov 2015 18:12:44 (-5.6d)
5	↑10	CodeMiner	0.90355	3	Fri, 27 Nov 2015 04:07:00
6	new	NSF Miners	0.90043	1	Wed, 25 Nov 2015 18:55:42

Our Approach towards Classification of data mining algorithms:

The data sizes gathered from different fields are increasing exponentially, data mining techniques that extract information from huge amount of unprocessed data have become more demanding in commercial and scientific domains, including marketing, customer relationship management, forecasting and

astronomy etc. During the evaluation, the input datasets and the number of classifiers used are varied to measure the performance of Data Mining algorithm.

Before we begin discussion about classification of data mining algorithms. We have to know what data mining process is about, **Data Mining** is an analytic process designed to explore **data** (usually large amounts of **data** - typically business or market related - also known as "big data") in search of consistent patterns and/or systematic relationships between variables, and then to validate the findings by applying the detected patterns.

As DataEnthusiasts used **WEKA** (acronym as Waikato Environment for Knowledge Analysis) is an open source libraries written in java for machine learning algorithms for data mining tasks. So we have used to classify the League of Legends Data. We used Java Weka library to build model from training data set. The detailed approach would be described in final project report. However, to outline the usage, **weka.jar** is added to java class path, so that all predefined classes are available to our java program.

Weka classes expect to read data from a specific file format called arff (Attribute Relation File Format). Here is where pre-processing was needed.

As the main function of data mining is extract knowledge discovery from data that is transform the raw data to meaningful patterns which can be easily understood. The three phases in this are

1. Pre-processing
2. Data Mining (DM)
3. Post-processing

Pre-processing Step:

The main purpose of pre-processing stage is to eliminate the incomplete data or

erroneous information.

We applied three techniques for preprocessing that is

1. Eliminate missing data.
2. Attributes Selection.
3. Discretization.

The supplied training data set is in csv (Comma Separated Values) file.

Certain fields have missing values, which are empty strings. Ex: ...,xyz,,

That is, two adjacent commas. This is not supported by weka.

Hence we have to pre-process the records and replace blank/empty strings with question marks (indicating missing values) '?'.
So, ...,xyz,,... → ...,?,xyz?,....

To answer certain specific questions

What is not clean about the data?

- Missing values just left blank in supplied csv training data set.

What did we need to do to prepare data?

- We came with a file processing program in java that reads csv training data set and creates a new arff file after performing data cleansing task.

Attributes Selection:

The attributes used to identify each record uniquely was clearly a noise. So we removed the queue_type, Season, Version etc from training data set, in arff file (in the pre-processing step itself).

During the week ending Project Milestone 2, we found the class in Weka library, which ranks the attributes based on information gain. It's a Ranker Algorithm for subset attribute selection. Corresponding class is

weka.attributeSelection.RankSearch. It uses gain ratio to identify the top attributes contributing towards class prediction.

Results:

=== Attribute Selection on all input data ===

Search Method:

RankSearch :

Attribute evaluator : weka.attributeSelection.GainRatioAttributeEval

Attribute ranking :

90 firstInhibitorAssist
99 inhibitorKills
75 firstInhibitorKill
17 towerKills
5 unrealKills
19 deaths
89 pentaKills
14 largestKillingSpree
15 quadraKills
77 tripleKills
88 killingSprees
13 largestMultiKill
69 doubleKills
43 timeline.goldPerMinDeltas.tenToTwenty
47 timeline.xpDiffPerMinDeltas.tenToTwenty
93 goldEarned
78 assists
67 neutralMinionsKilledEnemyJungle
25 firstTowerKill
73 kills
21 firstTowerAssist
34 champLevel
70 goldSpent
55 timeline.xpPerMinDeltas.tenToTwenty
46 timeline.xpDiffPerMinDeltas.zeroToTen
42 timeline.goldPerMinDeltas.zeroToTen
45 timeline.xpDiffPerMinDeltas.twentyToThirty
39 timeline.csDiffPerMinDeltas.tenToTwenty
41 timeline.goldPerMinDeltas.twentyToThirty
66 firstBloodKill
1 physicalDamageDealt
68 totalDamageDealtToChampions
8 totalDamageDealt
85 largestCriticalStrike
60 timeline.damageTakenDiffPerMinDeltas.tenToTwenty
16 magicDamageTaken
38 timeline.csDiffPerMinDeltas.zeroToTen
33 item5
100 totalDamageTaken
54 timeline.xpPerMinDeltas.zeroToTen
53 timeline.xpPerMinDeltas.twentyToThirty
72 map_id
32 item4
30 item6
74 season

84 physicalDamageDealtToChampions
 51 timeline.creepsPerMinDeltas.tenToTwenty
 50 timeline.creepsPerMinDeltas.zeroToTen
 18 totalTimeCrowdControlDealt
 27 item3
 82 visionWardsBoughtInGame
 59 timeline.damageTakenDiffPerMinDeltas.zeroToTen
 26 item2
 87 trueDamageDealtToChampions
 44 timeline.goldPerMinDeltas.thirtyToEnd
 2 item1
 102 summoner_id
 48 timeline.xpDiffPerMinDeltas.thirtyToEnd
 86 trueDamageDealt
 64 timeline.damageTakenPerMinDeltas.tenToTwenty
 28 item0
 12 teamId
 35 minionsKilled
 95 create_time
 91 totalHeal
 101 physicalDamageTaken
 10 match_id
 58 timeline.damageTakenDiffPerMinDeltas.twentyToThirty
 80 neutralMinionsKilled
 9 magicDamageDealtToChampions
 56 timeline.xpPerMinDeltas.thirtyToEnd
 24 neutralMinionsKilledTeamJungle
 63 timeline.damageTakenPerMinDeltas.zeroToTen
 76 trueDamageTaken
 71 magicDamageDealt
 6 highestAchievedSeasonTier
 4 wardsKilled
 37 timeline.csDiffPerMinDeltas.twentyToThirty
 49 timeline.creepsPerMinDeltas.twentyToThirty
 62 timeline.damageTakenPerMinDeltas.twentyToThirty
 98 totalUnitsHealed
 61 timeline.damageTakenDiffPerMinDeltas.thirtyToEnd
 31 wardsPlaced
 11 duration
 96 sightWardsBoughtInGame
 57 timeline.role
 22 championId
 65 timeline.damageTakenPerMinDeltas.thirtyToEnd
 92 spell2Id
 83 spell1Id
 40 timeline.csDiffPerMinDeltas.thirtyToEnd
 36 timeline.lane
 97 mode
 94 queue_type
 3 match_type
 7 objectivePlayerScore
 79 totalScoreRank
 81 combatPlayerScore
 20 totalPlayerScore
 23 firstBloodAssist
 52 timeline.creepsPerMinDeltas.thirtyToEnd
 Merit of best subset found : 0.186

Attribute Subset Evaluator (supervised, Class (nominal): 29 winner):

Classifier Subset Evaluator
Learning scheme: weka.classifiers.bayes.NaiveBayes
Scheme options:
Hold out/test set: Training data
Accuracy estimation: classification error

Selected attributes: 5,14,15,17,19,75,89,90,99 : 9

unrealKills
largestKillingSpree
quadraKills
towerKills
deaths
firstInhibitorKill
pentaKills
firstInhibitorAssist
inhibitorKills

Discretization

Discretization on numeric or continuous attributes. In this case, we have opted for keeping all of these values in the data. This means we can simply discretize by removing the "numeric" as the type for the "timeline.goldPerMinDeltas.zeroToTen", "timeline.goldPerMinDeltas.tenToTwenty", "timeline.goldPerMinDeltas.thirtyToEnd" etc attributes in the ARFF file

Data mining techniques used:

As mentioned earlier, we are using weka libraries in our java program to build model and classify the results.

Naïve Bayesian:

Naïve Bayesian can be explained with an example

Example 1: To predict the chance or the probability of raining, we usually use some evidences such as the amount of dark cloud in the area. Let H be the event of raining and E be the evidence of dark cloud, then we have

$$P(\text{raining} | \text{dark cloud}) = P(\text{dark cloud} | \text{raining}) \times P(\text{raining}) / P(\text{dark cloud})$$

– $P(\text{dark cloud} | \text{raining})$ is the probability that there is dark cloud when it rains. Of

course, “dark cloud” could occur in many other events such as overcast day or forest fire, but we only consider “dark cloud” in the context of event “raining”. This probability can be obtained from historical data recorded by some meteorologists.

– $P(\text{raining})$ is the priori probability of raining. This probability can be obtained from statistical record, for example, the number of rainy days throughout a year. – $P(\text{dark cloud})$ is the probability of the evidence “dark cloud” occurring. Again, this can be obtained from the statistical records, but the evidence is not usually well recorded compared to the main event. Therefore, sometimes the full evidence, i.e., $P(\text{dark cloud})$, is hard to obtain.

It estimates the probability of the hypothesis (**winner**) when we have a certain evidence (**physicalDamageDealt**)

Ex: $P(\text{winner}|\text{physicalDamageDealt}) = P(\text{physicalDamageDealt}|\text{winner}) * P(\text{winner})/P(\text{physicalDamageDealt})$

It does so, on all those attributes present in training data set, against the attribute which we want to classify (i.e. **winner**)

However, we need not bother about this code because weka.jar has this already implemented. But, the usage in the project is shown below

Here is the internal implementation of

NaiveBayesUpdateable.buildClassifier(instances) method, which does this task

```
public void buildClassifier(Instances instances) throws Exception {  
    // can classifier handle the data?  
    getCapabilities().testWithFail(instances);  
  
    // remove instances with missing class  
    instances = new Instances(instances);  
    instances.deleteWithMissingClass();  
}
```

```

m_NumClasses = instances.numClasses();

// Copy the instances
m_Instances = new Instances(instances);

// Discretize instances if required
if (m_UseDiscretization) {
    m_Disc = new weka.filters.supervised.attribute.Discretize();
    m_Disc.setInputFormat(m_Instances);
    m_Instances = weka.filters.Filter.useFilter(m_Instances, m_Disc);
} else {
    m_Disc = null;
}

// Reserve space for the distributions
m_Distributions = new Estimator[m_Instances.numAttributes() - 1]
    [m_Instances.numClasses()];
m_ClassDistribution = new DiscreteEstimator(m_Instances.numClasses(),
    true);

int attIndex = 0;
Enumeration enu = m_Instances.enumerateAttributes();
while (enu.hasMoreElements()) {
    Attribute attribute = (Attribute) enu.nextElement();

    // If the attribute is numeric, determine the estimator
    // numeric precision from differences between adjacent values
    double numPrecision = DEFAULT_NUM_PRECISION;
    if (attribute.type() == Attribute.NUMERIC) {
        m_Instances.sort(attribute);
        if ((m_Instances.numInstances() > 0)
            && !m_Instances.instance(0).isMissing(attribute)) {
            double lastVal = m_Instances.instance(0).value(attribute);
            double currentVal, deltaSum = 0;
            int distinct = 0;
            for (int i = 1; i < m_Instances.numInstances(); i++) {
                Instance currentInst = m_Instances.instance(i);
                if (currentInst.isMissing(attribute)) {
                    break;
                }
                currentVal = currentInst.value(attribute);
                if (currentVal != lastVal) {
                    deltaSum += currentVal - lastVal;
                    lastVal = currentVal;
                    distinct++;
                }
            }
            if (distinct > 0) {
                numPrecision = deltaSum / distinct;
            }
        }
    }

    for (int j = 0; j < m_Instances.numClasses(); j++) {
        switch (attribute.type()) {

```

```

    case Attribute.NUMERIC:
        if (m_UseKernelEstimator) {
            m_Distributions[attIndex][j] =
                new KernelEstimator(numPrecision);
        } else {
            m_Distributions[attIndex][j] =
                new NormalEstimator(numPrecision);
        }
        break;
    case Attribute.NOMINAL:
        m_Distributions[attIndex][j] =
            new DiscreteEstimator(attribute.numValues(), true);
        break;
    default:
        throw new Exception("Attribute type unknown to NaiveBayes");
    }
}
attIndex++;
}

// Compute counts
Enumeration enumInsts = m_Instances.enumerateInstances();
while (enumInsts.hasMoreElements()) {
    Instance instance =
        (Instance) enumInsts.nextElement();
    updateClassifier(instance);
}

// Save space
m_Instances = new Instances(m_Instances, 0);
}

```

However, we need not bother about this code because **weka.jar** has this already implemented. But, the usage in the project is shown below

```

//Naive Bayes Classifier
ArffLoader loader = new ArffLoader();
loader.setFile(new File("F:\\MS\\TAMUC\\Fall-
2015\\DataMining\\Assignment\\training_set\\game.arff"));
Instances structure = loader.getStructure();
//specifies the index of winner attribute in arff file
structure.setClassIndex(28);
//instantiate Naive Bayes classifier
NaiveBayesUpdateable classifier = new NaiveBayesUpdateable();
//build the classifier model
classifier.buildClassifier(structure);
Instance current = null;
//get the model trained on every instance by iteratively updating
while((current = loader.getNextInstance(structure)) != null)
    classifier.updateClassifier(current);
//once the model is built, persist it for applying it on test data set
SerializationHelper.write("game.model", classifier);

```

```
//print the summary of model  
System.out.println(classifier);
```

The details of Naïve Bayesian classifier implementation in WEKA can be found at <http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayesUpdateable.html>

weka.classifiers.bayes.NaiveBayesUpdateable is the implementation class for this.

Click <https://svn.cms.waikato.ac.nz/svn/weka/branches/stable-3-6/weka/src/main/java/weka/classifiers/bayes/NaiveBayes.java> to view complete source code (internal implementation provided by WEKA)

With use of **Naïve Bayesian** technique to classify the result earlier which give the accuracy around 75% only.

Learning:

- 1) We have tried implementing winner prediction for LOL game we are succeed in getting the accuracy about 75%, but this not competitive accuracy so we have to go for another algorithm approach for getting more accuracy, as we did not have neither much knowledge nor assistance.

Improvement: 1

We tried out a different and popular algorithm C4.5 (which is discussed in class as well) to build decision tree, since its efficient. Weka has it implemented in **weka.classifiers.trees.J48** class

C4.5 Algorithm:

This is divide-and-conquer algorithm. Builds decision tree by recursively

selecting attributes to test. It splits the dataset into subsets according to the outcome of the test. Ultimately, the goal is to obtain a subset that contains instances of only one class. In the decision tree, each leaf is associated with one class. Branches represent the outcomes of each test. Only one test is used for each branching node.

The tree gets built up by choosing the attribute with the largest *information gain*. Use this attribute as the next test to split the dataset. Thus, it guarantees that the tree built is the shortest.

Ex:

In the **LoL** training data set, two classes for winner {**true**, **false**}

C1 = **true**

C2 = **false**

If one instance is selected at random from set S

The probability of it belonging to class $C_j = \text{Freq}(C_j, S)/|S|$

The information that it conveys $-\log_2 [\text{Freq}(C_j, S)/|S|]$ bits.

$$\text{Info}(S) = -\sum_{j=1}^k \left(\frac{\text{Freq}(C_j, S)}{|S|} \times \log_2 \left[\frac{\text{Freq}(C_j, S)}{|S|} \right] \right)$$

The expected information measurement of the test on the attribute X is the total weighted sum

$$\text{Info}_x(T) = \sum_{i=1}^n \left(\frac{|T_i|}{|T|} \times \text{Info}(T_i) \right)$$

$$\text{Gain}_x(T) = \text{Info}(T) - \text{Info}_x(T)$$

This is how Information Gain is captured on each attribute and the attribute with highest information gain is selected and data set is split accordingly.

Improvement 2:

We experimented the effect of pruning in decision trees. Weka's '**weka.classifiers.trees.J48**' lets us generate pruned as well as unpruned trees. In general, if you increase pruning, the accuracy on the training set will be lower. WEKA documentation mentions however various things to estimate the accuracy better, namely training/test split or cross-validation. If you use cross-validation for example, we discovered the importance of pruning confidence factor, so we started pruning somewhere where it prunes enough to make the learned decision tree sufficiently accurate on test data, but doesn't sacrifice too much accuracy on the training data.

As we know pruning is a way of reducing the size of the decision tree. This will reduce the accuracy on the training data, but (in general) increase the accuracy on unseen data. It is used to mitigate [overfitting](#), where we would achieve perfect accuracy on training data, but the model (i.e. the decision tree) we learn is so specific that it doesn't apply to anything but that training data.

Details about is:

The J48 classifier provides a methods for pruning a decision tree. If you use values > 0.5 for the confidence parameter in J48 then "pruning" will be done based on unmodified classification error on the training data. This is effectively equivalent to turning pruning off. This was we realized after the submitting the report to kaggle account, our accuracy had less value.

So, we have started pruning the decision tree other way that it smaller confidence factor incurs less pruning of the tree. That is we run J48 with the confidence factor **0.50, 0.25, 0.15 down**

Confidence factor 0.50

#	Δ3d	Team Name	Score [?]	Entries	Last Submission UTC (Best - Last Submission)
1	↑1	Team MI	0.89617	4	Sun, 15 Nov 2015 17:49:38
2	↓1	DataEnthusiasts	0.89151	4	Tue, 17 Nov 2015 16:56:55
Your Best Entry ↑ Top Ten! You made the top ten by improving your score by 0.00149. Tweet this!					
3	↑6	Data Explorers	0.85418	2	Mon, 16 Nov 2015 02:16:26
4	↓1	invaders.db2	0.83244	4	Sun, 15 Nov 2015 00:54:09 (-1.6h)
5	↑5	Lionesses	0.82924	2	Mon, 16 Nov 2015 03:18:08
6	new	champions	0.81687	2	Tue, 17 Nov 2015 01:38:51
7	↑1	Rangers	0.81439	2	Mon, 16 Nov 2015 03:27:41
8	↑3	TamucDataminers	0.81433	5	Sun, 15 Nov 2015 19:48:56
9	↓1	M... ..	0.81011	2	Mon, 16 Nov 2015 01:16:55

Confidence factor 0.25

#	Δ3d	Team Name	Score [?]	Entries	Last Submission UTC (Best - Last Submission)
1	↑1	Team MI	0.89617	4	Sun, 15 Nov 2015 17:49:38
2	↓1	DataEnthusiasts	0.89195	5	Wed, 18 Nov 2015 07:03:07
Your Best Entry ↑ Top Ten! You made the top ten by improving your score by 0.00044. Tweet this!					
3	↑6	Data Explorers	0.88805	3	Tue, 17 Nov 2015 22:59:50
4	↑2	DataMiners	0.87753	2	Tue, 17 Nov 2015 23:03:34
5	new	champions	0.87345	3	Wed, 18 Nov 2015 05:46:57
6	↓3	invaders.db2	0.83244	4	Sun, 15 Nov 2015 00:54:09 (-1.6h)
7	↑3	Lionesses	0.82924	2	Mon, 16 Nov 2015 03:18:08

Confidence factor 0.15

#	Δ3d	Team Name	Score ?	Entries	Last Submission UTC (Best - Last Submission)
1	new	champions	0.91047	5	Wed, 18 Nov 2015 07:31:21
2	↓1	Team MI	0.89617	4	Sun, 15 Nov 2015 17:49:38
3	↓1	DataEnthusiasts	0.89587	10	Thu, 19 Nov 2015 06:41:25
Your Best Entry ↑ Top Ten! You made the top ten by improving your score by 0.00392. Tweet this!					
4	↓1	Data Explorers	0.89164	4	Wed, 18 Nov 2015 14:04:10
5	↑5	DataMiners	0.87753	2	Tue, 17 Nov 2015 23:03:34
6	↓2	invaders.db2	0.83244	6	Thu, 19 Nov 2015 03:57:46 (-4.2d)
7	↓2	Lionesses	0.82924	2	Mon, 16 Nov 2015 03:18:08

Reference to pruning. <https://en.wikipedia.org/wiki/Pruning> (algorithm)

Improvement 3.1:

As part of our final efforts, we tried using top 30 attributes, as shown in previous pages.

Data Pre Processing:

- Wrote a program to select only top 30 attributes from training data set
- Same program is used to select top 30 attributes from test data set (The same attributes that were chosen for training)

Data Mining Step:

- Now, since the selected attributes are few in number, the size of training data set file reduced.
- Therefore we were able to feed larger data set to our learning algorithm
- Built the model and persisted it
- Applied the saved model on test data set, and saved the predictions to **.arff** file

Data Post Processing:

- Ran the same old Kaggle Output file formatter, which converts to **.csv** file with **_id, winner** attributes

Result: Unfortunately, we didn't gain much out of this attempt. In fact, the accuracy went down to 87% which is not an improvement over current result.

Failure Analysis:

- When we observed the test data set closely, we figured out that most of the attributes among those top 30 selected ones had missing values (i.e ?)
- This resulted in wrong predictions by the model

Improvement 3.2:

- We identified those attributes which had more than 50% of missing values in training data set and removed them from both training and test data set
- Selected top 50 attributes from these newly attributes subset
- This reduced the size of training data set from nearly 3.73 GB to 1.83 GB

Data Pre Processing:

- Wrote a program to select only top 50 attributes from training data set (after missing valued attributes are eliminated)
- Same program is used to select top 50 attributes from test data set (The same attributes that were chosen for training)

Data Mining Step:

- Now, since the selected attributes are few in number, the size of training data set file reduced.
- Therefore we were able to feed larger data set to our learning algorithm
- Built the model and persisted it

- Applied the saved model on test data set, and saved the predictions to **.arff** file

Data Post Processing:

- Ran the same old Kaggle Output file formatter, which converts to **.csv** file with **_id, winner** attributes

Result: This improved our predictions. The accuracy went up to **90.842%** which is an improvement over previous result.

Here is the internal implementation of J48.buildClassifier (instances) method,

```
/**
 * Generates the classifier.
 *
 * @param instances the data to train the classifier with
 * @throws Exception if classifier can't be built successfully
 */
public void buildClassifier(Instances instances)
    throws Exception {

    ModelSelection modSelection;

    if (m_binarySplits)
        modSelection = new BinC45ModelSelection(m_minNumObj, instances);
    else
        modSelection = new C45ModelSelection(m_minNumObj, instances);
    if (!m_reducedErrorPruning)
        m_root = new C45PruneableClassifierTree(modSelection, !m_unpruned, m_CF,
            m_subtreeRaising, !m_noCleanup);
    else
        m_root = new PruneableClassifierTree(modSelection, !m_unpruned, m_numFolds,
            !m_noCleanup, m_Seed);
    m_root.buildClassifier(instances);
    if (m_binarySplits) {
        ((BinC45ModelSelection)modSelection).cleanup();
    } else {
        ((C45ModelSelection)modSelection).cleanup();
    }
}
```

We need not bother about this code because weka.jar has this already implemented. But, the usage in the project is shown below

//J48 Classifier

```

J48 classifier = new J48();
//ensure that there is no pruning
classifier.setUnpruned(true);
//turn on debugging feature, so that we can debug in case of any issues
classifier.setDebug(true);
//build the model for classification
classifier.buildClassifier(data);
//once the model is built, save it for applying it on test data set
//in future because building model is a time consuming process
SerializationHelper.write("game_lol_pca_1.model", classifier);

```

The Java Docs for this class can be found at

<http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html>

Click <https://svn.cms.waikato.ac.nz/svn/weka/branches/stable-3-6/weka/src/main/java/weka/classifiers/trees/J48.java> to view complete source code (internal implementation provided by WEKA)

Post pruning:

Starts after the completely over-fitted decision tree is created. Works upward from the leaves to the root, tries to replace internal nodes with lower-level nodes.

Replacement criterion, by estimate node and its children error rate this has to be less than the combined weighted children estimated error this decides further portioning.

Error rate

Let E be observed number of error instances, N is total number of instances, f is observed error rate.

- Then $\Pr [(f - q)/\sigma > z] = e$ is the one-tailed random probability above.
- f (the random variable) subtracted by the mean q (the estimated error rate of the population) is divided by (the standard deviation), and e is the upper-tailed confidence level.

Consider one confidence level e , we can get the error rate q value by getting the z

value first and then solving the inequality in the expression above,

Since C4.5 used $e = 25\%$ as the default confidence level, its corresponding z becomes 0.69.

Same we implemented using in J48 classifier.

Post-processing

In the case of machine learning algorithms such as trees or decision rules trained with noisy data, the results are generated covering few training data. This is because the induction algorithms try to subdivide the training data set. To overcome this problem the decision trees or rules should be shrunk, by either post-pruning (decision trees) or truncation (decision rules). After obtaining new knowledge, this can be either implemented in an expert system or used by an end user. In this last case, the knowledge results should be documented for the end user interpretation. Another possibility is to display the knowledge and transform it into a form understandable to the end user. We can also check the new knowledge for potential conflicts with previously induced knowledge. In this step, we can also summarize the rules and combine them with a domain-specific knowledge provided for the given task. Once a learning system has induced concept hypotheses (models) from the training set, their evaluation (or testing) should take place. There are several criteria used for this purpose: classification accuracy, understanding, computational complexity, and so on.

As the kaggle competition of league of legend required to have file in specific format. After the data mining algorithm applied on the test data the generated comma separated value file will have **_id** and **winner** attributes.

File format in diagram representation given below.

```
1 _id, winner
2 55855f6bdc3be924b392240d, 1
3 55855f6bdc3be924b3922413, 1
4 55855f6bdc3be924b3922422, 1
5 55855f70dc3be924b392242b, 1
6 55855f73dc3be924b392243d, 1
7 55855f73dc3be924b3922444, 1
8 55855f74dc3be924b392244c, 1
9 55855f74dc3be924b392244d, 0
10 55855f74dc3be924b392244e, 0
11 55855f74dc3be924b3922456, 1
12 55855f74dc3be924b392245b, 1
13 55855fdddc3be924b392290d, 0
14 55855fdddc3be924b3922919, 1
15 55855fdddc3be924b3922920, 1
16 55855fe2dc3be924b3922929, 1
17 55855fe2dc3be924b392292e, 0
18 55855fe2dc3be924b3922933, 0
19 55855fe3dc3be924b3922954, 0
20 55856083dc3be924b3923047, 1
21 55856083dc3be924b3923053, 0
22 55856084dc3be924b392305d, 1
23 55856084dc3be924b392305f, 0
24 55856084dc3be924b3923063, 1
25 55856084dc3be924b392306b, 1
26 55856085dc3be924b392307a, 1
27 55856085dc3be924b3923080, 1
28 55856086dc3be924b392309a, 0
29 55856087dc3be924b392309f, 1
30 55856087dc3be924b39230a7, 0
31 55856087dc3be924b39230a8, 1
32 558560f3dc3be924b3923542, 0
33 558560f4dc3be924b392355c, 1
34 558560f4dc3be924b392356a, 1
```

Results of Data Mining:

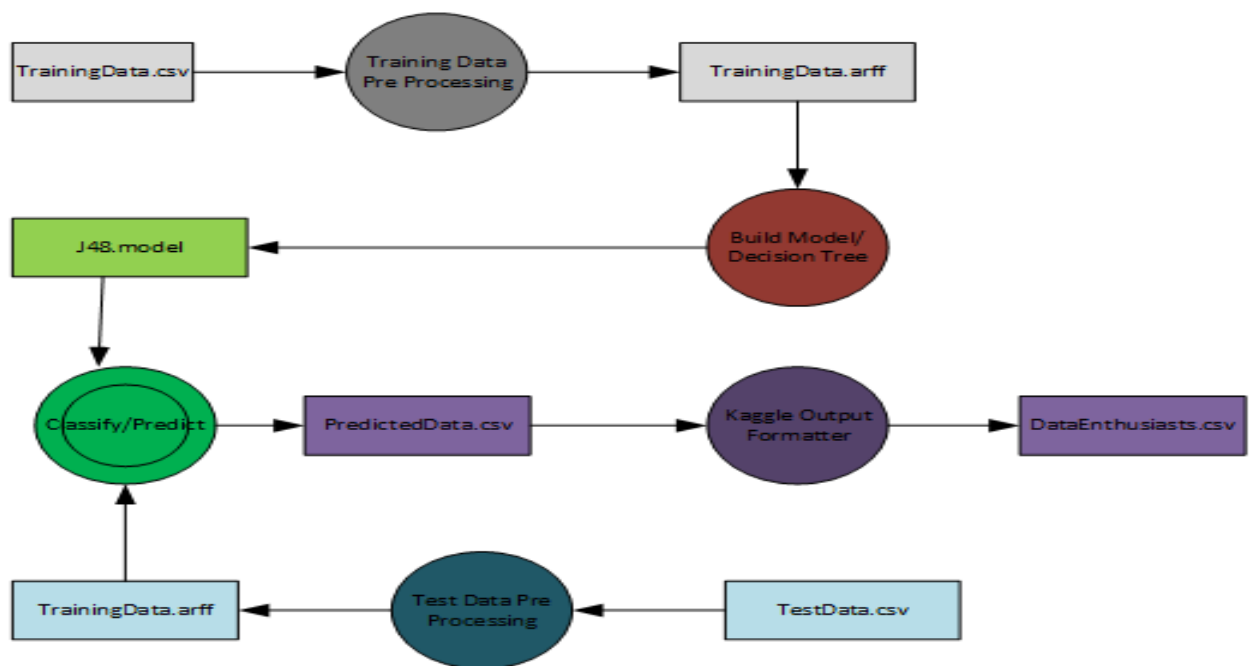
1. If the player is first inhibitor to assist the game, having previous experience more than 30 and plays for a duration of more than 30 minutes, there is a high probability that he will win the game. (Confidence: 57%)
2. If the player is first inhibitor assist, having more than 30 experience level and plays for duration more than 30 minutes without earning inhibitor kills (in other words being killed by inhibitor), he will lose the game. (Confidence: 55%)
3. If the player is the first inhibitor to assist the game and plays for less than 30 minutes and has less than 4 deaths within that duration, he will win the game (Confidence: 70%)
4. If the player has more than 4 deaths in first 30 minutes of the game and his champion level less than 14, he loses the game (Confidence: 61%)
5. If the player plays for more than 30 minutes, spending 75% of time by

controlling the crowd by possessing less than 2000 item5's, he will lose the game. (Confidence: 55%)

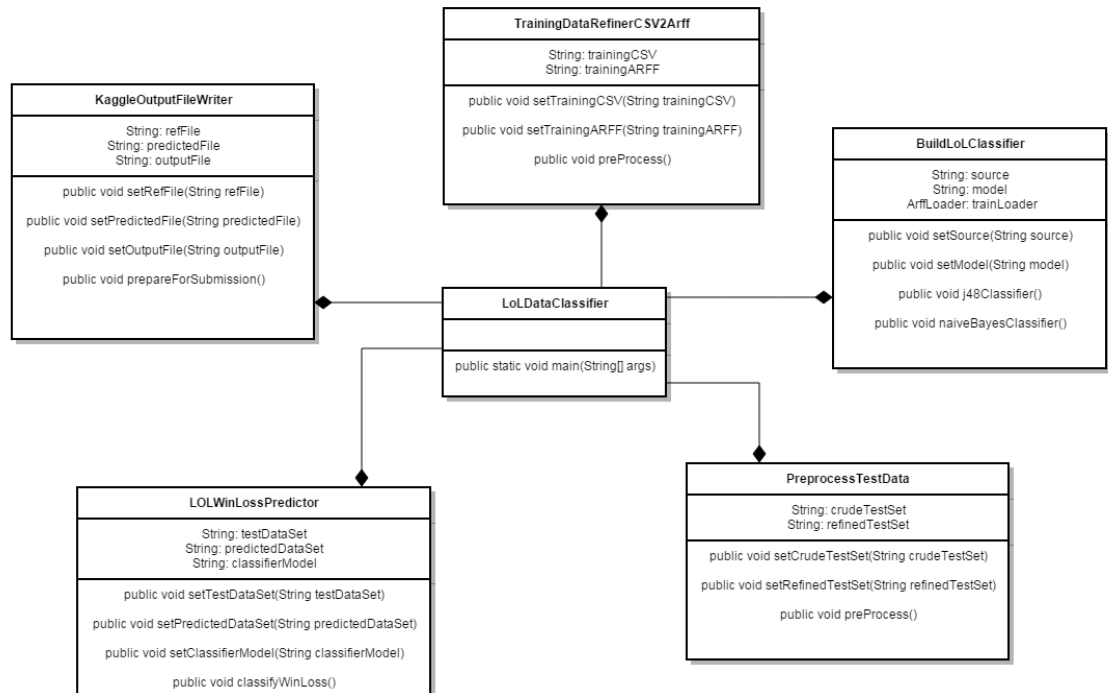
The Top 5 most important attributes, our algorithm in use depends on are

1. firstInhibitorAssist
2. deaths
3. towerKills
4. largestKillingSpree
5. firstInhibitorKill

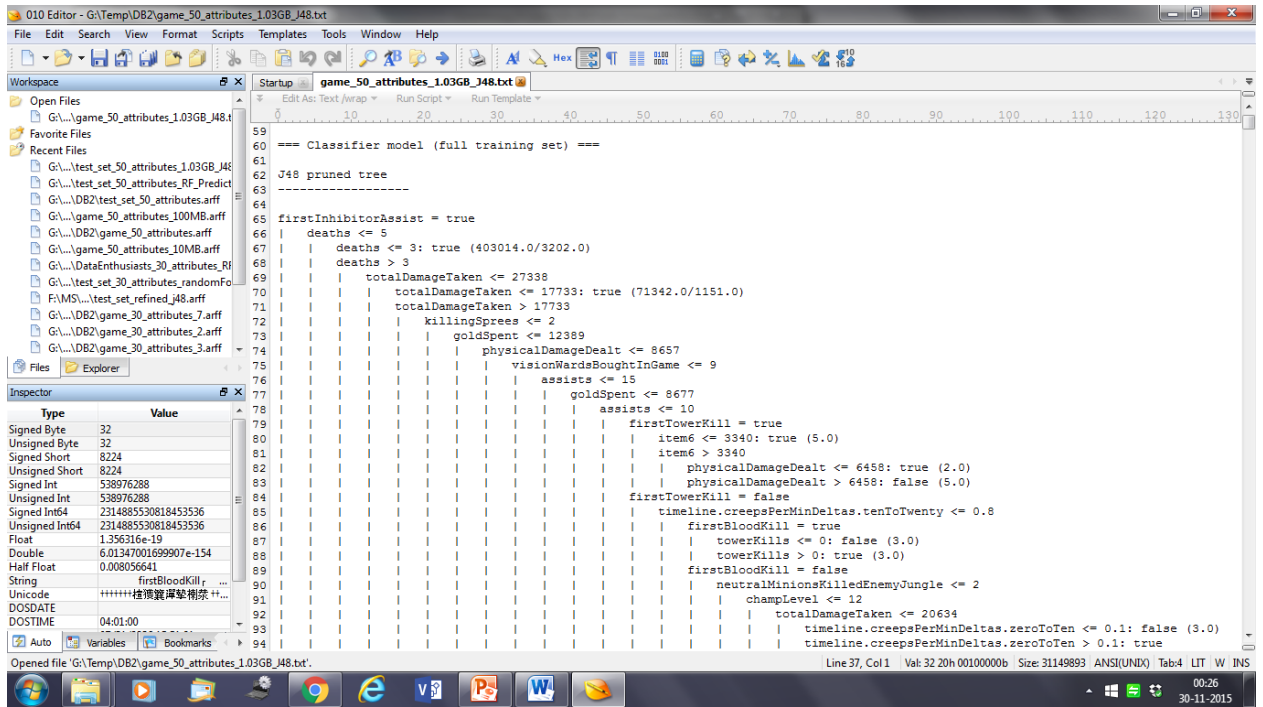
ARCHITECTURE DIAGRAM



CLASS DIAGRAM



GENERATED DECISION TREE



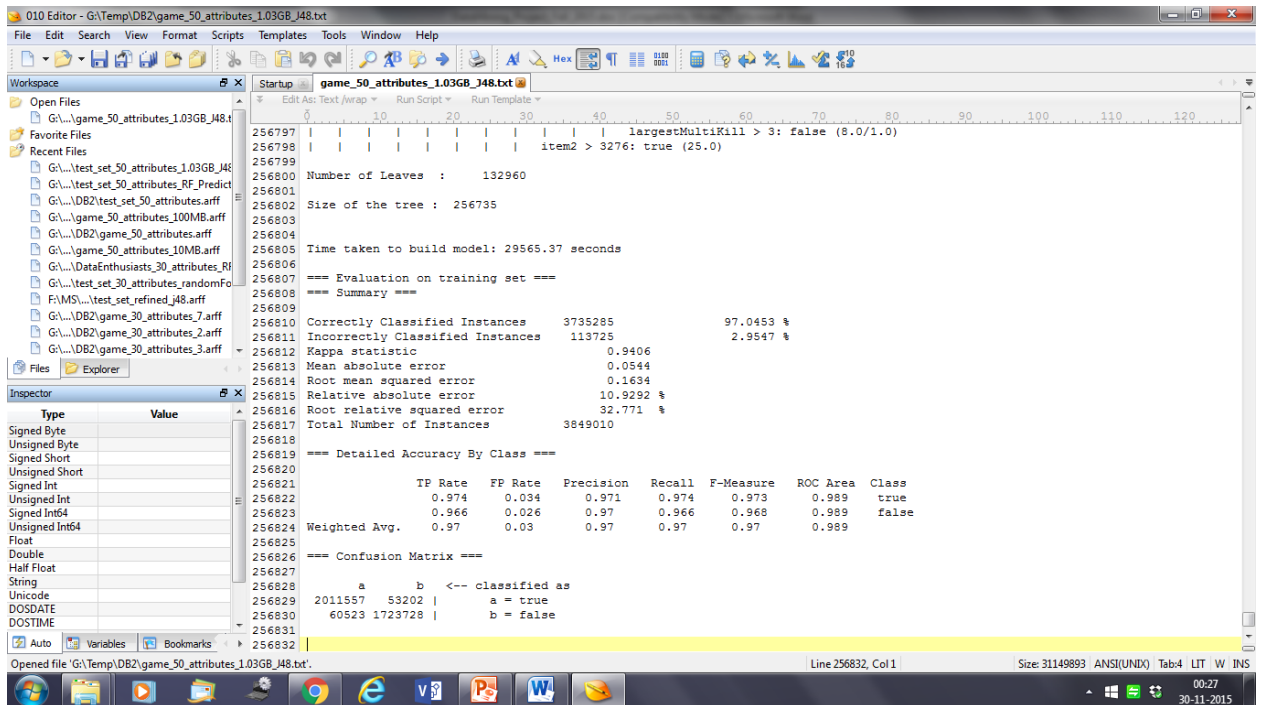
```
59 === Classifier model (full training set) ===
60
61 J48 pruned tree
62 -----
63
64 firstInhibitorAssist = true
65 | deaths <= 5
66 | | deaths <= 3: true (403014.0/3202.0)
67 | | deaths > 3
68 | | | totalDamageTaken <= 27338
69 | | | | totalDamageTaken <= 17733: true (71342.0/1151.0)
70 | | | | totalDamageTaken > 17733
71 | | | | | killingSpree <= 2
72 | | | | | goldSpent <= 12389
73 | | | | | physicalDamageDealt <= 8657
74 | | | | | visionWardsBoughtInGame <= 9
75 | | | | | assists <= 15
76 | | | | | goldSpent <= 8677
77 | | | | | assists <= 10
78 | | | | | firstTowerKill = true
79 | | | | | item6 <= 3340: true (5.0)
80 | | | | | item6 > 3340
81 | | | | | | physicalDamageDealt <= 6458: true (2.0)
82 | | | | | | physicalDamageDealt > 6458: false (5.0)
83 | | | | | firstTowerKill = false
84 | | | | | | timeline.creepsPerMinDeltas.tenToTwenty <= 0.8
85 | | | | | | firstBloodKill = true
86 | | | | | | towerKills <= 0: false (3.0)
87 | | | | | | towerKills > 0: true (3.0)
88 | | | | | firstBloodKill = false
89 | | | | | | neutralMinionsKilledEnemyJungle <= 2
90 | | | | | | champLevel <= 12
91 | | | | | | totalDamageTaken <= 20634
92 | | | | | | timeline.creepsPerMinDeltas.zeroToTen <= 0.1: false (3.0)
93 | | | | | | timeline.creepsPerMinDeltas.zeroToTen > 0.1: true
94 | | | | |
```

Inspector

Type	Value
Signed Byte	32
Unsigned Byte	32
Signed Short	8224
Unsigned Short	8224
Signed Int	538976288
Unsigned Int	538976288
Signed Int64	2314885530818453536
Unsigned Int64	2314885530818453536
Float	1.356316e-19
Double	6.01347001699907e-154
Half Float	0.008056641
String	firstBloodKill = true
Unicode	+++++检测资源消耗
DOSDATE	04-01-00
DOSTIME	04-01-00

Opened file: G:\Temp\DB2\game_50_attributes_1.03GB_J48.txt. Line 37, Col 1 Val: 32.20h 00100000b Size: 31149893 ANSI(UND) Tab: 4 LIT W INS 00:26 30-11-2015

GENERATED DECISION TREE SUMMARY



```
256797 | | | | | largestMultiKill > 3: false (8.0/1.0)
256798 | | | | | item2 > 3276: true (25.0)
256799
256800 Number of Leaves : 132960
256801
256802 Size of the tree : 256735
256803
256804 Time taken to build model: 29565.37 seconds
256805
256806 === Evaluation on training set ===
256807 === Summary ===
256808
256809 Correctly Classified Instances 3735285 97.0453 %
256810 Incorrectly Classified Instances 113725 2.9547 %
256811 Kappa statistic 0.9406
256812 Mean absolute error 0.0544
256813 Root mean squared error 0.1634
256814 Relative absolute error 10.9292 %
256815 Root relative squared error 32.771 %
256816 Total Number of Instances 3849010
256817
256818 === Detailed Accuracy By Class ===
256819
256820 TP Rate FP Rate Precision Recall F-Measure ROC Area Class
256821 0.974 0.034 0.971 0.974 0.973 0.989 true
256822 0.966 0.026 0.97 0.966 0.968 0.989 false
256823 Weighted Avg. 0.97 0.03 0.97 0.97 0.97 0.989
256824
256825 === Confusion Matrix ===
256826
256827 a b <-- classified as
256828 2011557 53202 | a = true
256829 60523 1723728 | b = false
256830
256831
256832
```

Inspector

Type	Value
Signed Byte	32
Unsigned Byte	32
Signed Short	8224
Unsigned Short	8224
Signed Int	538976288
Unsigned Int	538976288
Signed Int64	2314885530818453536
Unsigned Int64	2314885530818453536
Float	1.356316e-19
Double	6.01347001699907e-154
Half Float	0.008056641
String	firstBloodKill = true
Unicode	+++++检测资源消耗
DOSDATE	04-01-00
DOSTIME	04-01-00

Opened file: G:\Temp\DB2\game_50_attributes_1.03GB_J48.txt. Line 256832, Col 1 Size: 31149893 ANSI(UND) Tab: 4 LIT W INS 00:27 30-11-2015

Conclusion: The large amounts of data that is nowadays virtually stored along with the certain prediction games like League of legend have to be analyzed and should be used to their full extent. Various decision tree algorithms can be used for prediction of winner of League of legend. Our studies showed that J48 gives 90.043% accuracy; hence it can be used as a base learner. With the help of other meta-algorithms like missing attribute removal and attribute selection, J48 gives accuracy of 90.043% which makes a good predictive model as because we were able to find the various attributes, those all are critical in predict the game winner in the game of League of Legends.