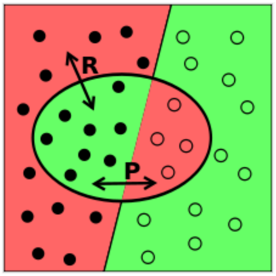
Code Challenging Assignment for Snapsort Company

By: Negar Koochakzadeh (MITACS Internship Applicant) *June 18, 2012*

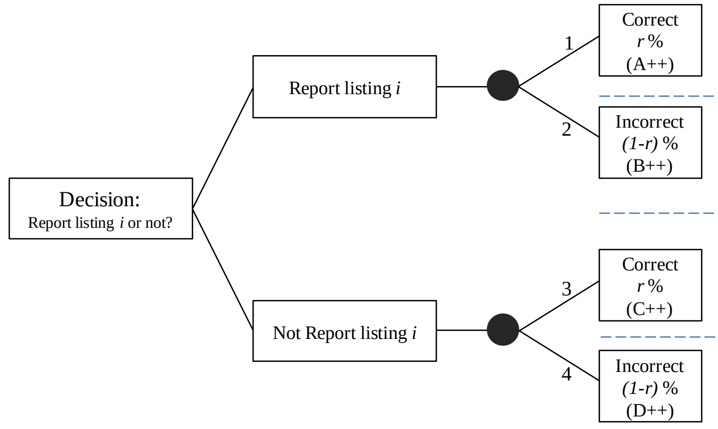
In this assignment, I have implemented a program that reads lists of products and price listings from two text files (provided by Snapsort) and matches each price listing with a correct product based on the objective mentioned by Snapsort in which the emphasis is more on the precision than the recall.

For this purpose, I assign a real value (*r*) in range of [0,1] to each price listing related to each product in order to consider probable relevancy of a listing to a product. In the current version, I have defined static rules based on my own manual investigations and observations on the provided dataset. However, more accurate rules could be extracted if a training set would be provided.

The assigned relevancy is then considered for making decision, whether to report a price listing for a product or not. For this purpose, I consider the main objective (higher precision) in which decision is made based on the risk value (*Expected Monetary Value (EMV)*) of how an incorrect listing may influence the precision value. Figure below shows the concept of precision and recall and how decision can be made based on EMV.



A 🟩● Relevant - Reported
B 🟥○ Not Relevant - Reported (False Positive)
C 🟥● Relevant - Not Reported (False Negative)
D 🟩○ Not Relevant - Not Reported



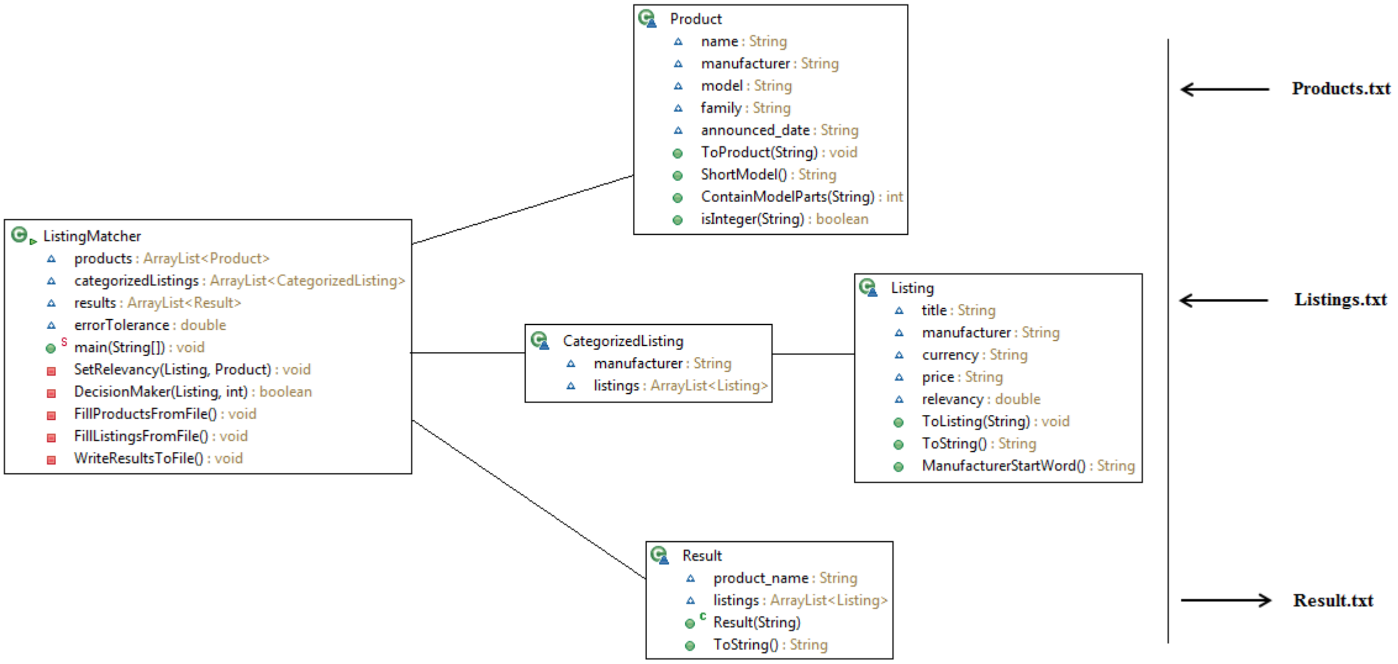
Decision Tree Analysis	Precision	Recall
Prior to making decision	$P = \frac{A}{A+B}$	$R = \frac{A}{A+C}$
Correct *r* % (A++)	$P_1 = \frac{A+1}{A+B+1}$	$R_1 = \frac{A+1}{A+C+1}$
Incorrect *(1-r)* % (B++)	$P_2 = \frac{A}{A+B+1}$	No change
Correct *r* % (C++)	No change	$R_3 = \frac{A}{A+C+1}$
Incorrect *(1-r)* % (D++)	No change	No change

$$Expected\ Monetary\ Value = (1-r)*(P_2-P_1) = (1-r)*(\frac{1}{A+B+1})$$

Branch 1 is the idealistic case in which we decide to report listing *i* and it is true. In this case both precision and recall will increase. Branch 2 causes precision reduction which needs to be avoided. Thus the difference between branch 1 and 2 is measured as the risk of precision reduction. If this risk is less than a threshold, listing *i* is reported, otherwise it is not reported. $A+B$ is equal to the number of reported items so far. Required information about calculating recall is not available; therefore it is not possible to calculate recall risk. However, branch 1 increases recall while branch 3 decreases recall. Thus branch 1 is preferable to all other branches (in case of having tolerable precision risk). Branch 4 does influence neither precision nor recall.

Starting from the beginning of the sorted list of price listings based on their relevancy to the current product all the listings with $r=1$ are reported and all the items with $r=0$ are not reported. For the other items (with $0 < r < 1$) EMV is calculated and compared with the threshold. For instance by setting the threshold to 0.01, items with relevancy equal to 0.2 and 0.7 are respectively allowed to be reported if it appears after 80 (for 0.2) and 30 (for 0.7) reported items so far, otherwise it is not reported.

The following figure illustrates the five classes implemented in this coding assignment as well as input/output files. ListingMatcher is the start class containing main function. Other classes are the data structure for input and output data elements. This program is coded in Java and can be executed in Linux by compiling and executing *ListingMatcher.java* using JavaC. Two files of *Products.txt* and *Listings.txt* should exist in the same path as *ListingMatcher.java*. The output will be reported in *result.txt* located in the same path.


Product
name : String
manufacturer : String
model : String
family : String
announced_date : String
ToProduct(String) : void
ShortModel() : String
ContainModelParts(String) : int
isInteger(String) : boolean
ListingMatcher
products : ArrayList<Product>
categorizedListings : ArrayList<CategorizedListing>
results : ArrayList<Result>
errorTolerance : double
S main(String[]) : void
SetRelevancy(Listing, Product) : void
DecisionMaker(Listing, int) : boolean
FillProductsFromFile() : void
FillListingsFromFile() : void
WriteResultsToFile() : void
CategorizedListing
manufacturer : String
listings : ArrayList<Listing>
Listing
title : String
manufacturer : String
currency : String
price : String
relevancy : double
ToListing(String) : void
ToString() : String
ManufacturerStartWord() : String
Result
product_name : String
listings : ArrayList<Listing>
C Result(String)
ToString() : String
Products.txt
Listings.txt
Result.txt