Tarika Jain

Google playstore app

SECTION 1

I got the Google Play Store dataset which had the attributes App, Category, Rating, Reviews, Size, Installs, Type, Price, Content rating, Genres, Last updated, Current ver, Android ver. I found quite a deep breath of work on this dataset and therefore was intrigued to use that as my inspiration.

	A	B	C	D	E	F	G
1	Category	Rating	Reviews	Type	Price	Install range	
2	MEDICAL	4.19	6	0	0	0-1000	
3	FAMILY	4.19	2	0	0	0-1000	
4	FAMILY	3.5	38824	0	0	1000000-5000000	
5	GAME	4.5	5849	0	0	50000-500000	
6	GAME	4.6	102107	0	0	1000000-5000000	
7	FAMILY	4	29708	0	0	1000000-5000000	
8	TRAVEL_ANI	4.4	17878	0	0	1000000-5000000	
9	MEDICAL	4.3	214	1	2.99	5000-10000	
10	FAMILY	4.1	321	0	0	50000-500000	
11	DATING	4.1	11633	0	0	50000-500000	
12	SOCIAL	3.6	58	0	0	5000-10000	
13	GAME	4.7	8038	0	0	50000-500000	
14	TOOLS	4.7	11018	0	0	50000-500000	
15	MEDICAL	3	2	0	0	0-1000	
16	FINANCE	5	12	0	0	0-1000	
17	BUSINESS	4	624	0	0	50000-500000	
18	COMMUNIC/	3.3	78	0	0	5000-10000	
19	GAME	3.8	35572	0	0	1000000-5000000	
20	FINANCE	4.19	2	0	0	0-1000	
21	FAMILY	4.4	12	0	0	0-1000	
22	PERSONALIZ	3.2	114	0	0	5000-10000	
23	TOOLS	4.5	60571	0	0	1000000-5000000	
24	DATING	3.4	5	0	0	0-1000	
25	FAMILY	5	2	0	0	0-1000	
26	MEDICAL	4.7	11	1	15.99	0-1000	
27	BUSINESS	3.9	45964	0	0	1000000-5000000	
28	TOOLS	4.3	2158	0	0	50000-500000	
29	PERSONALIZ	4.2	18280	0	0	1000000-5000000	
30	TOOLS	4.4	55	0	0	5000-10000	
31	LIBRARIES_A	4.19	2221	0	0	50000-500000	
32	DATING	2.5	5377	0	0	50000-500000	
33	DATING	4.1	825	0	0	50000-500000	
34	FAMILY	4.1	265	1	2.99	5000-10000	
35	FAMILY	4.8	38	0	0	0-1000	

Tarika Jain

Aim:
1. To predict the number of installs for each category based on ratings, reviews and price

The following are the two precedents that used the same dataset:

1. https://www.kaggle.com/shikhabains/google-reviews-prediction/notebook
2. https://www.kaggle.com/lava18/all-that-you-need-to-know-about-the-android-market
(The second analysis gives a good understanding of how I can approach this data going forward.)

SECTION 2

Raw features provided: App, Category, Rating, Reviews, Size, Installs, Type, Price, Content rating, Genres, Last updated, Current ver, Android ver

Features kept: Category, Rating, Reviews, Installs, Type, Price

Features modified: Installs to Number of installs range, 'Type' to numerical from a nominal attribute.

The following pre-formatting was performed on the data:

1. **Size:** Initially converted all app sizes into MB. After performing basic analysis, I found out that it wasn't contributing much to the aim of my analysis and therefore deleted the 'Size' attribute.
2. **Number of installs:** Took out '+' from the number of installs' values. Further, since the range of the number of installs was really big i.e. 0-5000k, I took ranges of the number of installs such as 0-1000 and so on and assigned instances to each based on that in such a way that each bucket contained almost the same amount of instances. This was done to ease out the analysis process such that it takes up less computational power and hopefully get more accurate results.
3. I deleted some of the repetitive attributes and the ones that didn't contribute a lot to my aim for this project.
4. **Type:** Converted this attribute to numerical i.e. 0 and 1 from a nominal one which had 'TRUE' and 'FALSE' as its class values.

Class value chosen for prediction: Number of Installs
This data was gathered directly from the Google Play Store.

Initial data exploration:
Correlation matrix

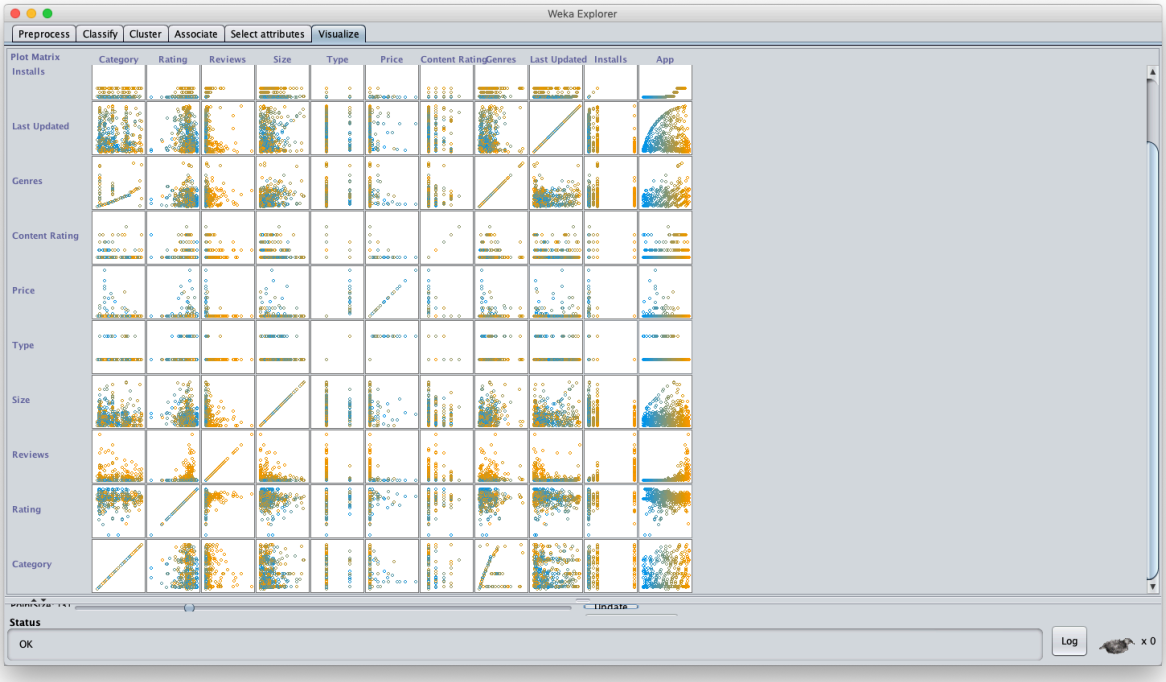
	A	B	C	D	E
1		*Rating*	*Reviews*	*Type*	*Price*
2	Rating	1			
3	Reviews	0.17046952	1		
4	Type	0.03843579	-0.0861424	1	
5	Price	-0.007881	-0.0229944	0.2281156	1

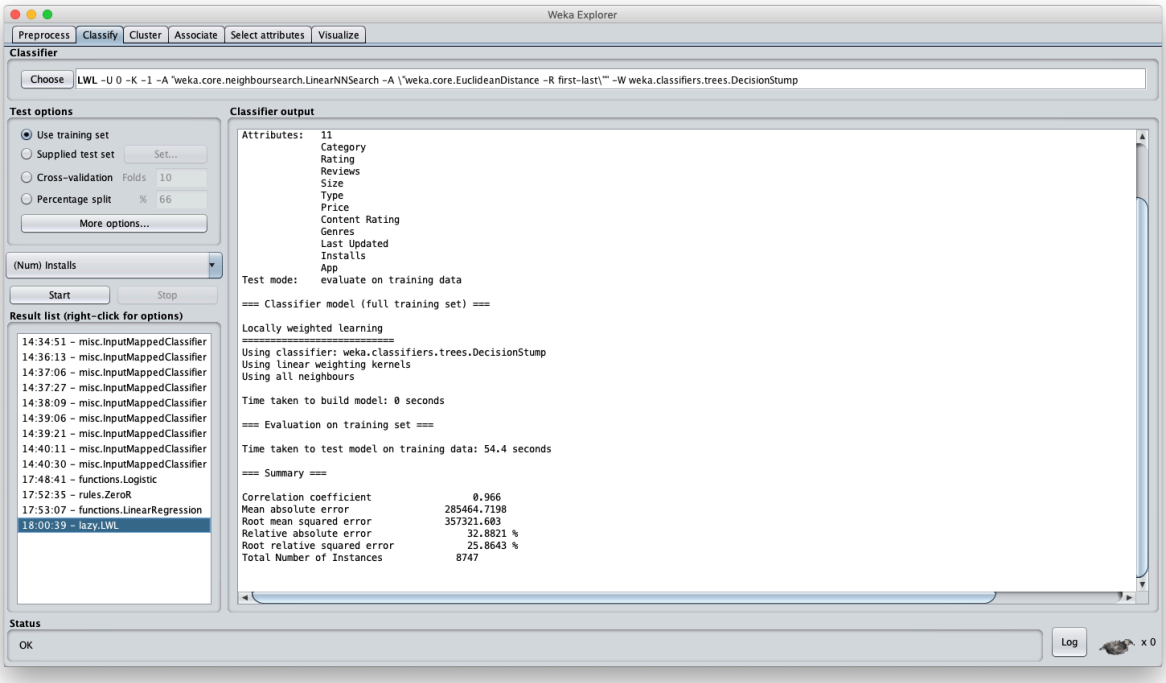
Tarika Jain

SECTION 3

1. Basic Error Data Analysis:

When I saw the dataset, I knew I had to re-format the ‘installs’ attribute and therefore my first step was to normalize the attribute values using the formula [(Value- Min)/(Max-Min)]. As observed in the Visualize section, it immediately showed that it was still creating problems.



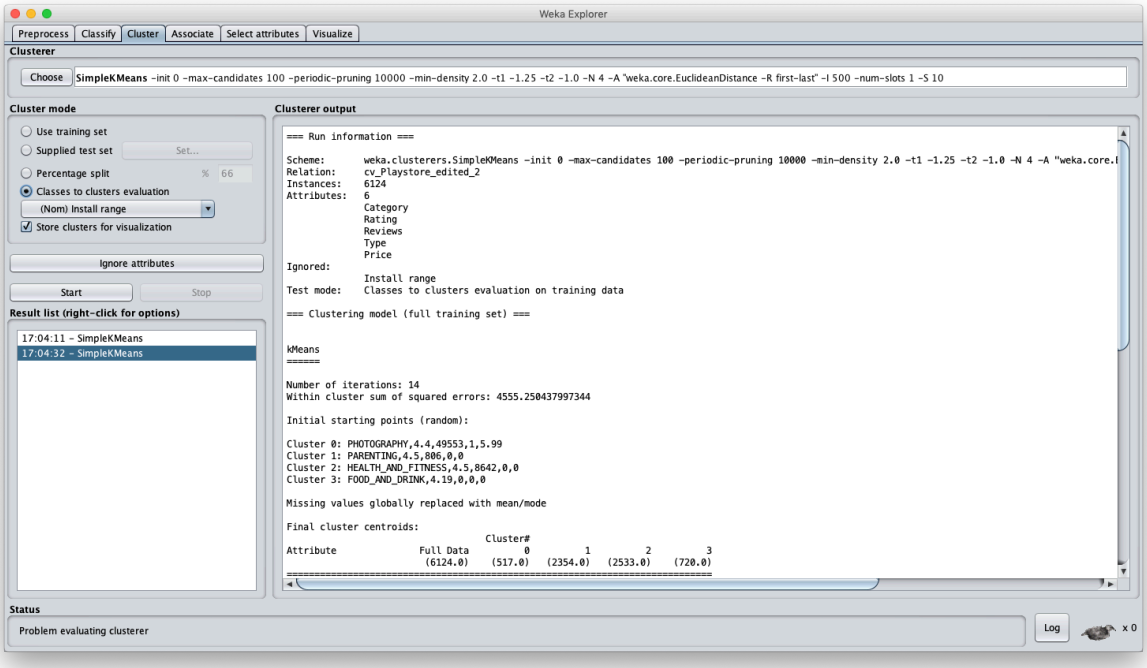


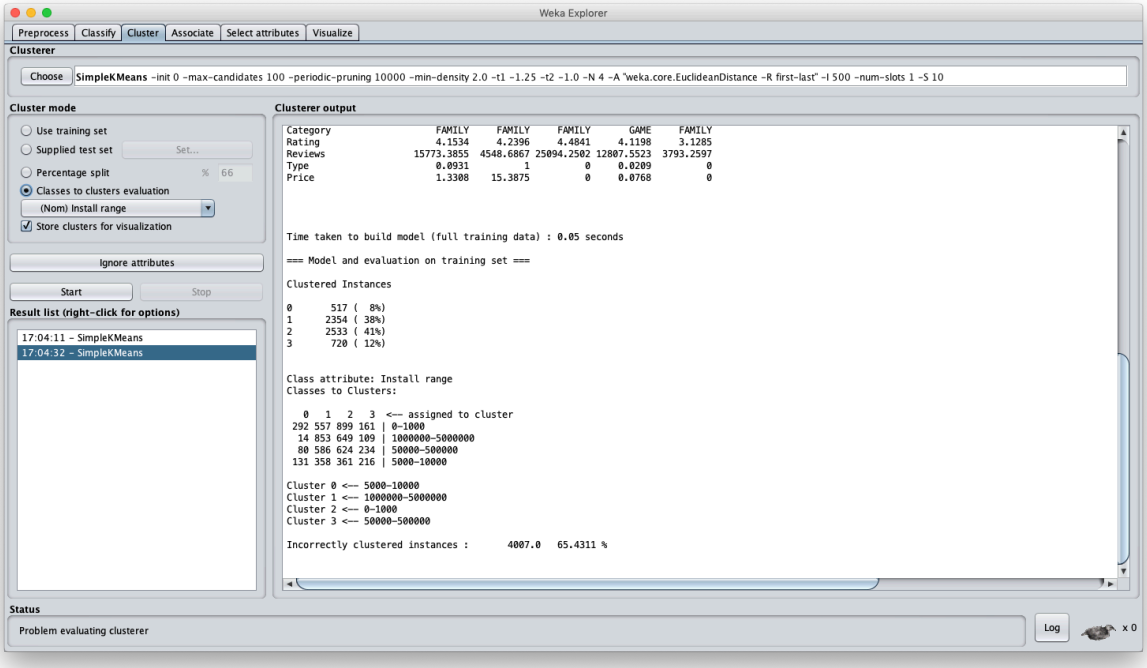
Tarika Jain

I wasn't able to perform even simple linear regression on this data with the class value as 'installs' as the range was too big and therefore I tried to analyze the data using Lazy weight learners which gave me an RRSE of 25%.

Clustering

I also decided to cluster my initial data to get a sense of how categories get clustered together to understand certain patterns in the dataset.





Tarika Jain

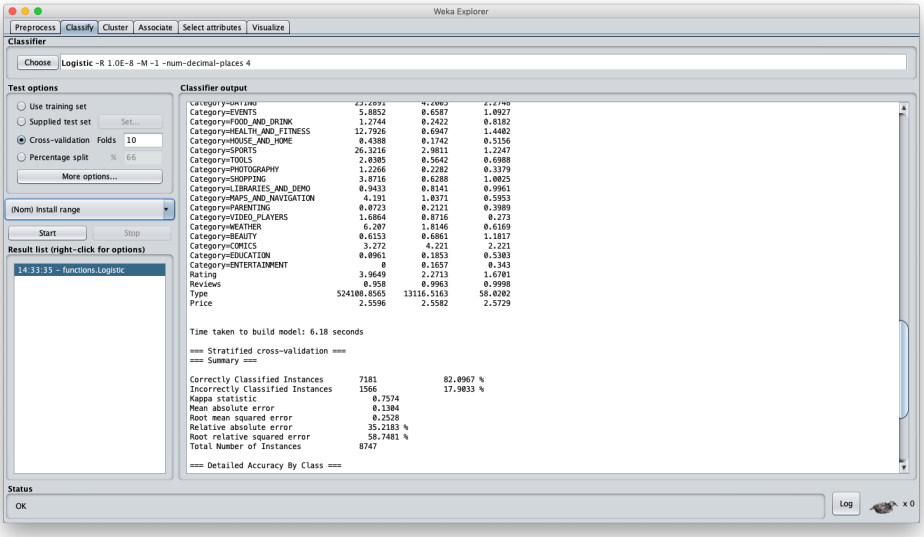
Using the classes to clusters evaluation where class value is ‘Install range’, it can be seen that the majority of Photography as a category with a paid type and above 4.4 rating belong to Cluster 0. The Cluster 1 majority contains Parenting, Cluster 2 contains Health and fitness and finally Cluster 3 majorly contains Food and drink. All contain majorly Free apps i.e. 0 and have a rating above 4. This categorization is not completely reliable as there are about 65% incorrectly classified instances.

2. **Problematic Features:** ‘Installs’ attribute as it had a huge range 0-5000K and the ‘Size’ of the app’ as its class values were in different units. Another problem encountered was that the attribute ‘Genres’ was very similar to ‘Category’ but just a little more detailed and therefore I found it being repetitive and not really contributing to the analysis but just taking more computational power.

3. **Ideas for improvement:**

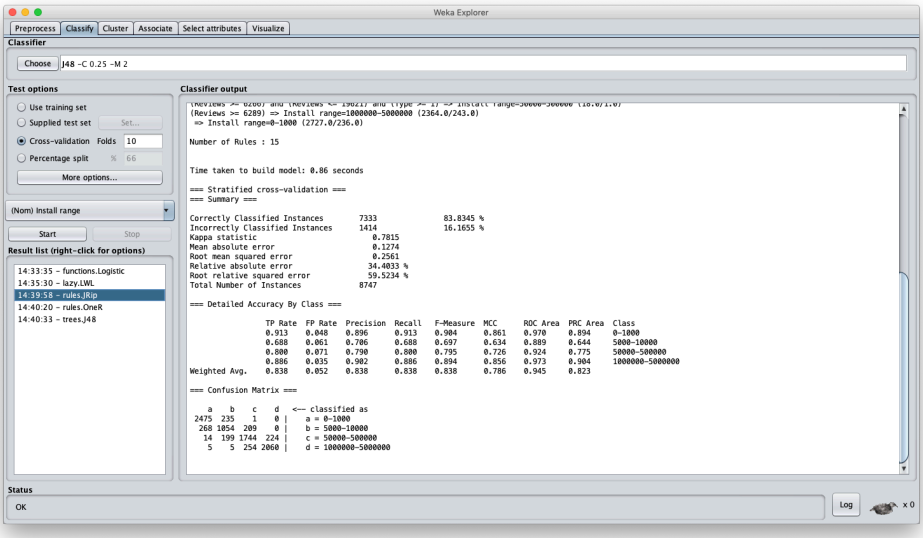
- **For Installs:** I initially normalized the attribute values which didn’t work as it was giving me an even larger Root relative squared error (RRSE) value which is bad. I finally decided to make the attribute values nominal i.e. bucketed class values such that distribution of instances was almost same.This helped simplify the analysis and reduce the amount of computational power required.
- **For Size:** I didn’t see too much contribution of size of the app towards the aim of my analysis.
- **Other improvement:** Removed multiple attributes such as Last updated, Current ver, Android ver and Content rating to make the analysis process more streamlined.
- **Tests:** After reconfiguring my data, I tried the following analysis:

1. Logistic regression on cv set | Accuracy: 82.0967% | RRSE: 58.7481%

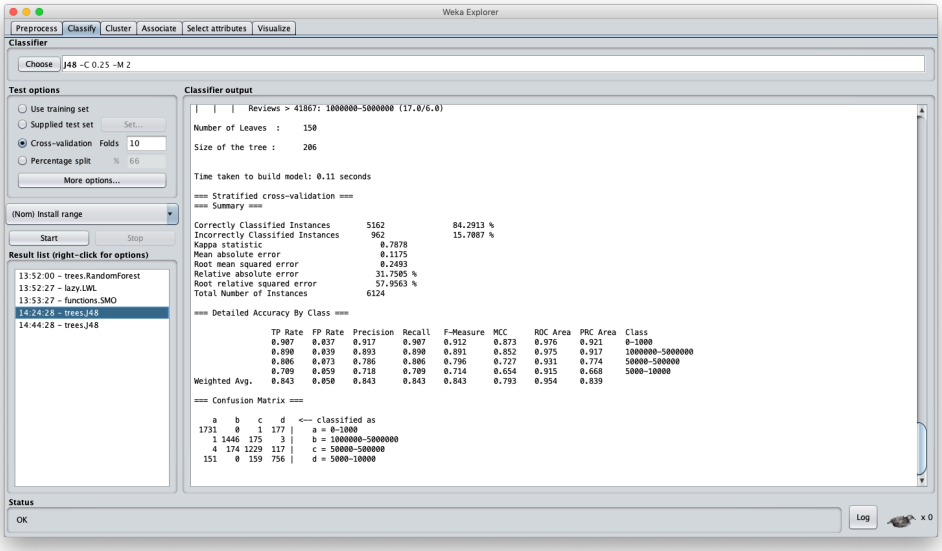


Tarika Jain

2. JRip rule on cv set | Accuracy: 83.8345% | RRSE: 59.5234%

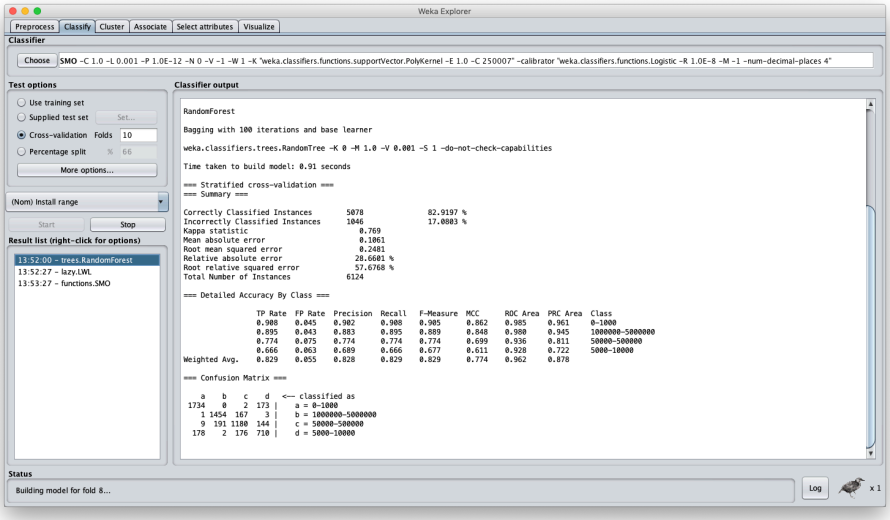


3. J48 decision trees on cv set | Accuracy: 84.2913% | RRSE: 57.9563%

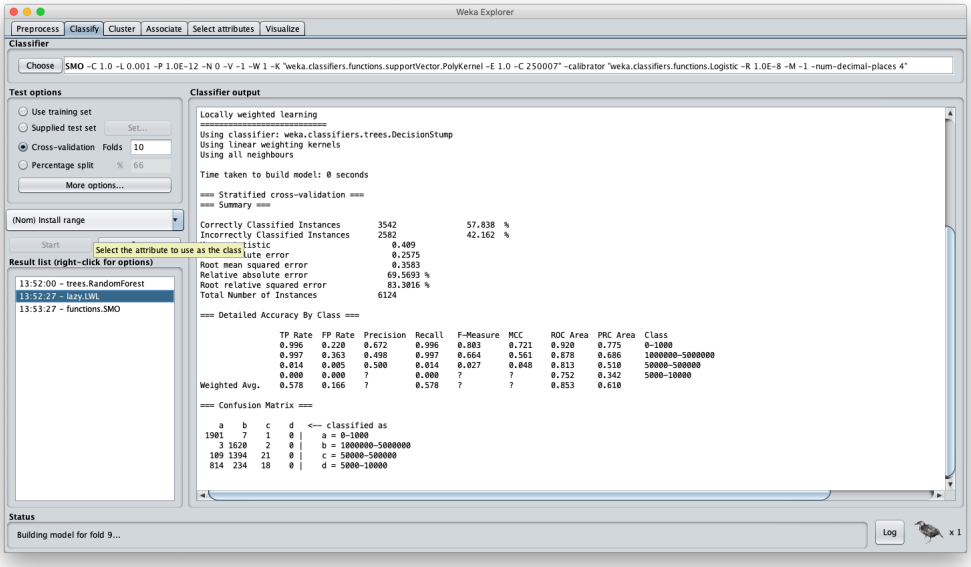


4. Random forests on cv set | Accuracy: 82.9197% | RRSE: 57.6768%

Tarika Jain



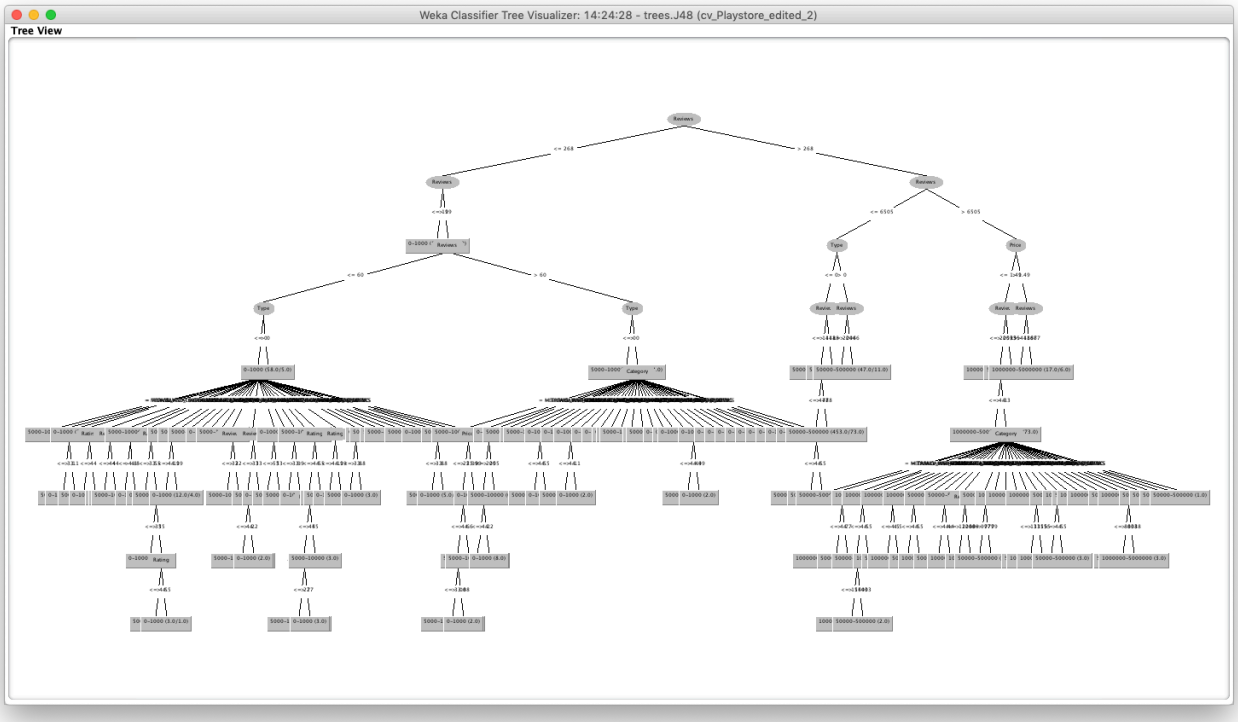
5. LWL on cv set I Accuracy: 57.838% I RRSE: 83.3016%



As J48 turned out be the best performing algorithm on my cross-validation dataset with highest accuracy rate and lowest RRSE comparatively, I decided to go ahead with this. The RRSE is still very high which means that the 15% of the incorrectly classified instances will be quite far away from the actual instance values.

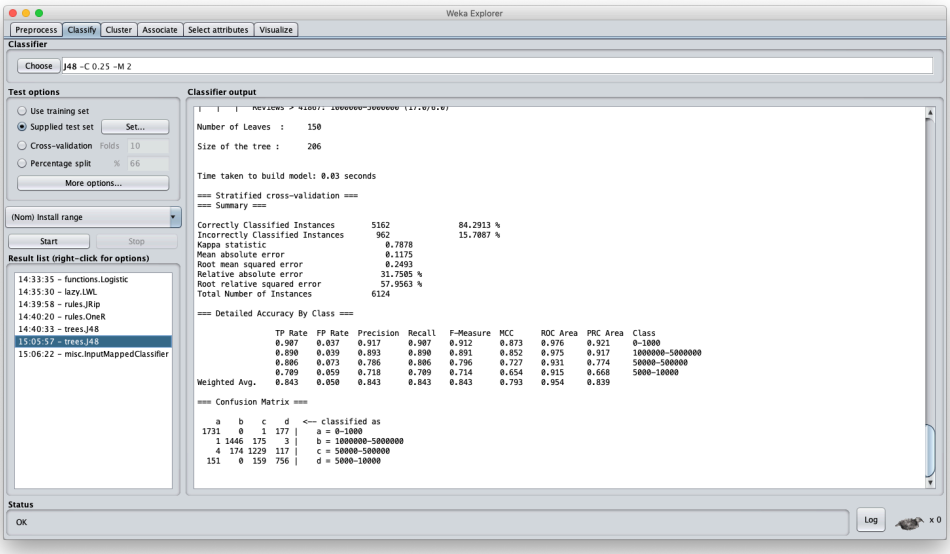
Reason: I think the reason J48 worked the best in this scenario was because my class value is to predict the number of installs based on the categories, number of ratings and reviews, type and price which can be understood very well based on setting rules and taking a top-down approach whereby the decision tree is first splitting the training data through the number of reviews as the node (<=268, >268) and then further dividing based on type and price. Further, it again divides using the number of reviews and then finally by category.

Tarika Jain



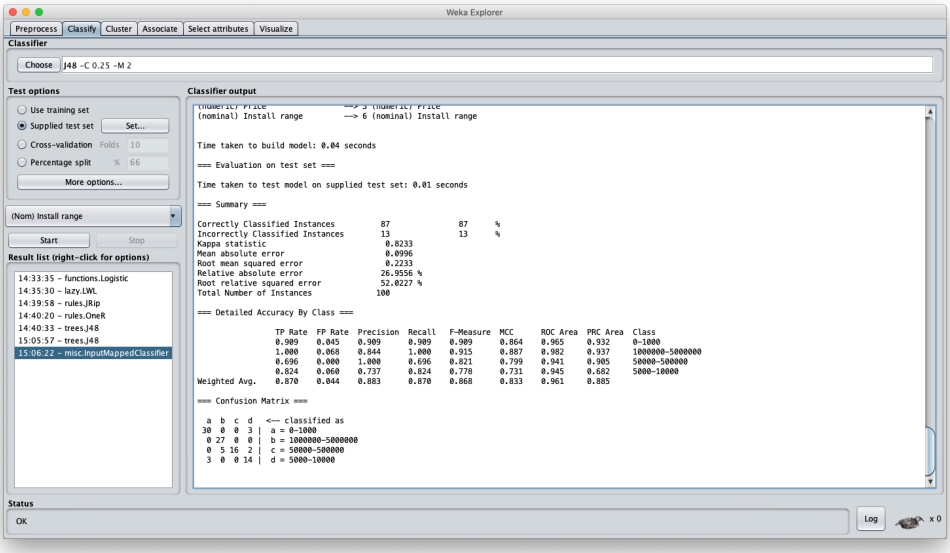
- After looking at this distribution, I thought it might be a good idea to remove the attribute of 'Type' to see if the distribution gets more simplified and maybe provide for a better accuracy but it was a little less so I decided not to go with that idea.

Training the model using J48 decision tree algorithm on cv set | Accuracy: 84.2913%

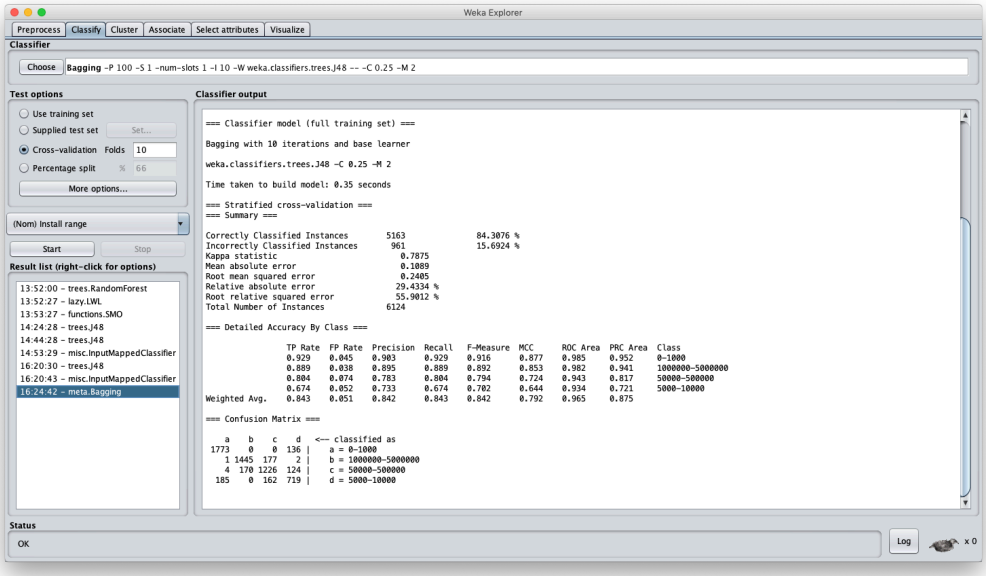


Testing the model using J48 decision tree algorithm on dev set | Accuracy: 87%

Tarika Jain

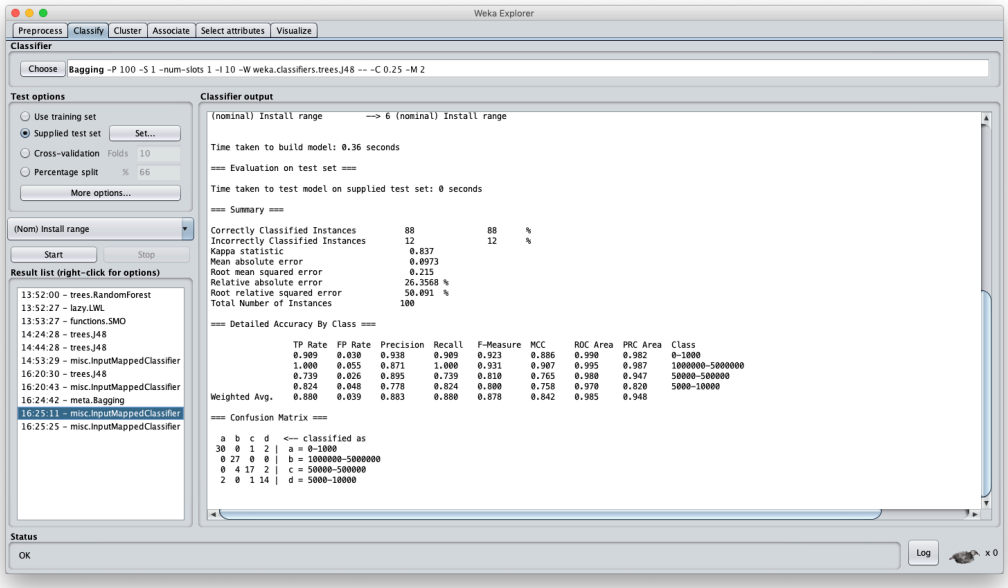


4. Evaluation:



A. Based on the J48 results on the dev set, I also tried bagging to train the data for better results and then test with the dev set.
The accuracy improved a little to **84.3876%** and on the dev set the accuracy was **88%** which was really good.

Tarika Jain



B. Finally, I performed tuning in the next section on the best performing model to see if I can improve the accuracy of the model by tinkering the minNumObj.

SECTION 4

Based on my initial experimentation with multiple algorithms on the complete cross-validation set and further using that training results for prediction evaluation on my dev set, I decided to take the best performing algorithm which was the J48 decision tree algorithm on my dataset for my tuning process for classification. I decided to tune the minNumObj parameter. Range tuned is 1-3.

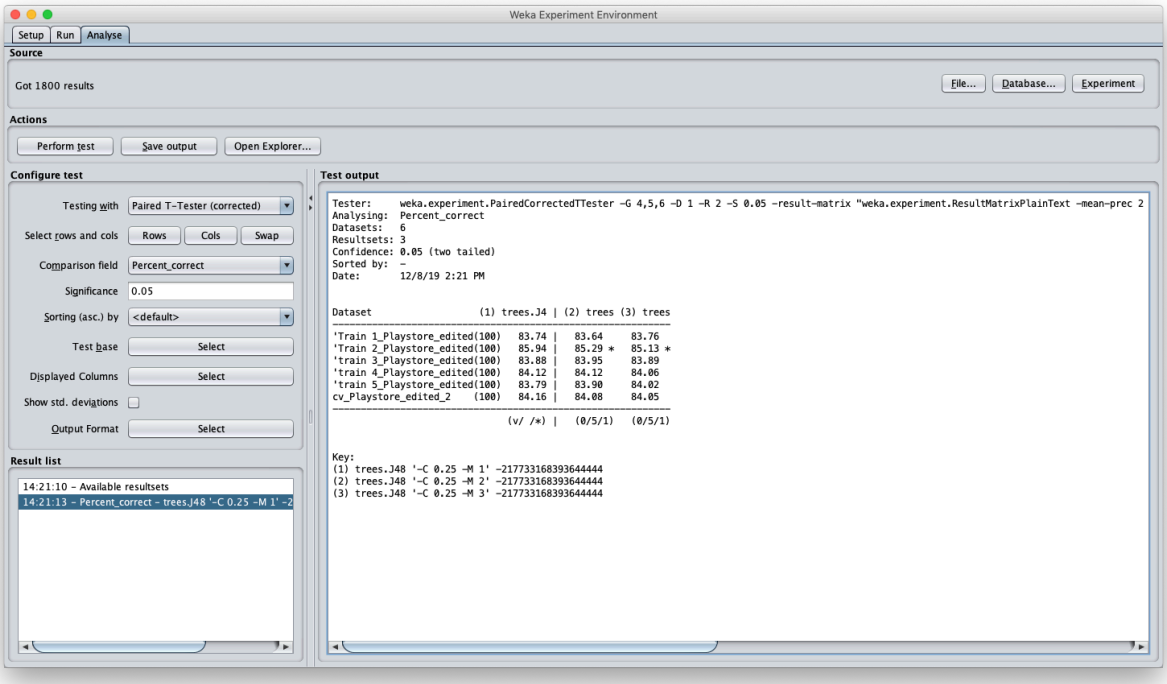
Fold	Train set performance for exponent = 1	Train set performance for exponent = 2	Train set performance for exponent = 3	Optimal Setting	Test set performance -OS	Default Setting	Test set performance - DS
1	83.74	83.64	83.76	3	84.5714	2	84.4082
2	85.94	85.29	85.13	1	83.1837	2	82.5306
3	83.88	83.95	83.89	2	84.0	2	84.0
4	84.12	84.12	84.06	1	84.898	2	84.6531
5	83.79	83.90	84.02	3	83.9052	2	83.7418
Average	84.294	84.18	84.172		84.11166		83.86674

Average of each performance based on percent correct:

Tarika Jain

Performance of exponent 1 = 84.294
Performance of exponent 2 = 84.18
Performance of exponent 3 = 84.172

By taking the difference between the Optimal and default setting on the test set performance average and performing a t-test on it, I found out that the P value is 0.08899883 which is more than 0.05 and that means that the model with the new settings shows insignificant improvement from its default settings i.e. 2 as the minNumObj; therefore tuning was not worth it.



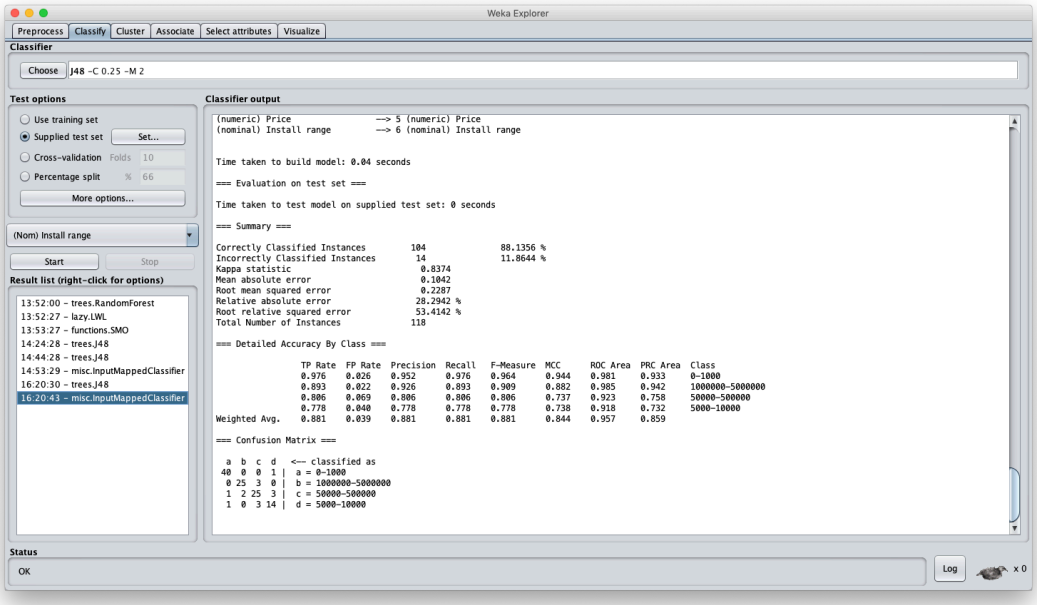
Paired T-test

	A	B
1	84.5714	84.4082
2	83.1837	82.5306
3	84	84
4	84.898	84.6531
5	83.9052	83.7418
6		
7	0.08899883	

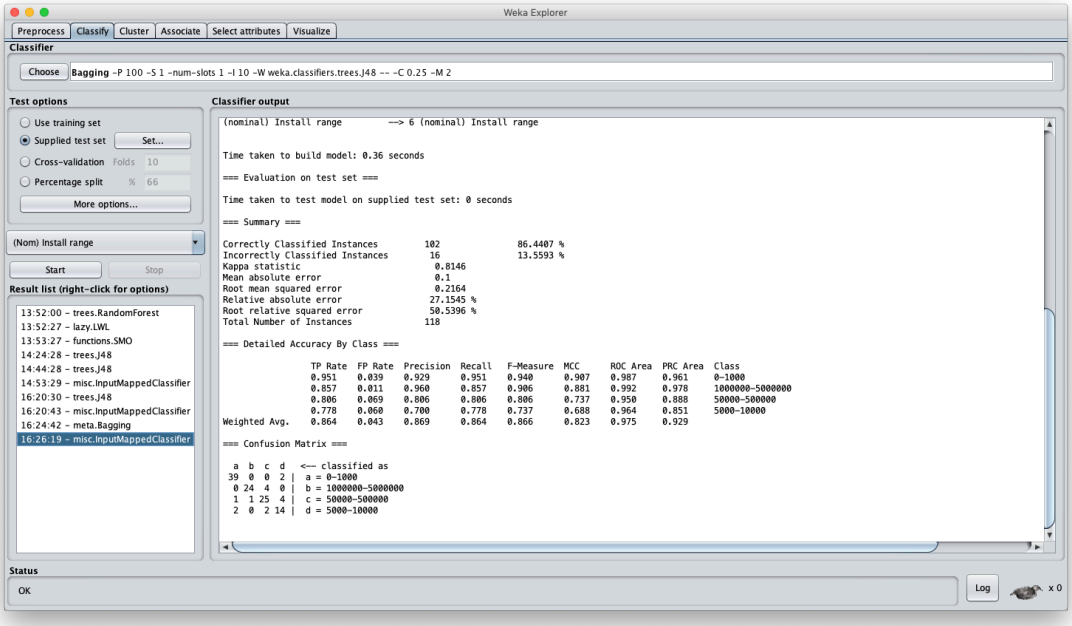
SECTION 5

Step 1: Testing on the trained model using the J48 decision tree algorithm with default settings as the best found settings.
Accuracy: 88.1356% | RRSE: 53.4142%

Tarika Jain



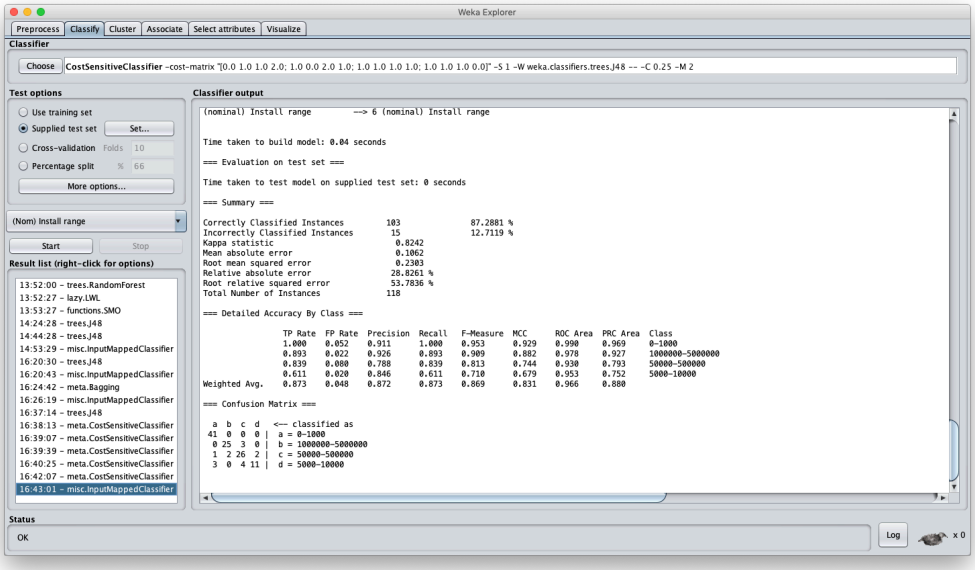
Step 2: Testing on the trained model using the J48 decision tree algorithm with default settings as the best found settings and adding an ensemble method of bagging to it.



Accuracy: 86.4407% | RRSE: 50.5396%
While the accuracy increased while testing on the dev set using bagging, the accuracy decreased with the test set using bagging. I therefore decided to not consider this method in my analyzation further.

Tarika Jain

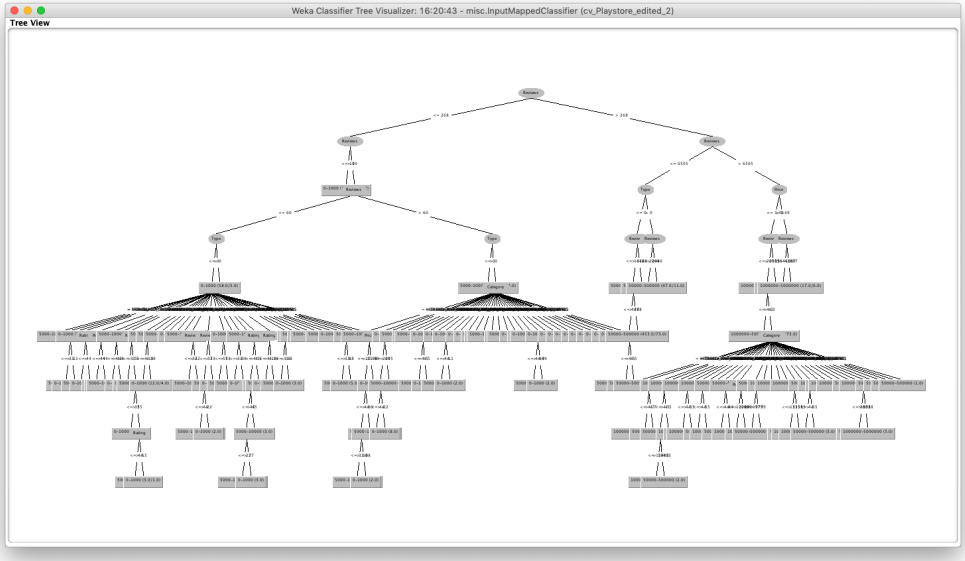
Step 3: Testing on the trained model using the J48 decision tree algorithm with default settings as the best found settings and performing cost- sensitive analysis on it to try to improve performance.



The performance didn't improve but actually got a little worse.

I therefore finally went with the first model without the bagging or the cost-sensitive analysis as it gave the best results on the final test set.

SECTION 6



Tarika Jain

Based on the best performing model above which has an accuracy of 88.1356%, it can be concluded that majority of the instances from a test set would be correctly predicted into one of the 4 ranges for the number of installs thereby suggesting that given a particular category, rating, number of reviews, type and price of app, it is an 88% chance that number of installs for that app category will be correctly predicted.

Through this project I learnt a lot about the benefits and disadvantages of various algorithms which helped in understanding how to build the best prediction model.