Name	Jordan Healy - 13379226
	Ryan McDyer - 13431038
Programme	CASE
Module Code	CA4010
Assignment Title	Data Mining Report
Submission Date	Sunday 20th November 2016

DECLARATION

We declare that this material, which we now submit for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work. We understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion, or copying. We have read and understood the Assignment Regulations set out in the module documentation. We have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study

1 CONTENTS

2 INTRODUCTION

The following report is a detailed explanation of our research, findings and our reasoning with regards to our dataset for the Data Warehousing & Data Mining module (CA4010). Our aim is to predict the number of views an image from Flickr has based on location and date & time it was taken.

2.1 TOOLS USED

Tool	Used for
Bash	Automating the downloading data from Flickr API
Java	Combining our JSON files into one large JSON Array
Json2csv	Converting our JSON file to comma separated values
R	Calculating the standardised values for skewed attributes
Knime	Generating decision tree

3 IDEA AND DATASET DESCRIPTION

3.1 DATASET

We decided to create our own dataset for this project. Initially, we wanted to predict the tags you could assign to an image. We looked at a number of image-hosting websites, including Imgur, Flickr, Instagram, and Photobucket. Photobucket and Imgur don't have tags so we had to discount these websites straight away.

We looked at the APIs of Flickr and Instagram, and concluded that Flickr had the better API. It was robust enough to give us the data we needed, while being intuitive enough to allow for complex searches, and there was a wealth of API documentation on the Flickr website.

To access the API you must have an API key, and to get an API key you need to associate your "app" that will access the API with a Flickr account. For the purposes of this project we set up an account and registered our "app" with Flickr. We experimented with a number of different API calls before we found the one we needed: "flickr.photos.search". We tried the following API call:

```
curl -X GET
"https://api.flickr.com/services/rest/?method=flickr.photos.search
        &has_geo=true
        &safe_search=2
        &media=photos
        &api_key=1234567890abcdef
        &extras=tags%2Cgeo%2Cdate_taken%2Cviews
        &lat=53.345786
        &lon=-6.260172
        &radius=3.0
        &format=json
        &nojsoncallback=1"
```

Code 1

This query returned the data for roughly 330,000 images - that's the number of images that have ever been uploaded to Flickr with a geo-tag inside our 3 kilometre radius. At first this seemed ideal, but Flickr limits the number of images you can access with one query to 8,000. As a result we had

the data for 8,000 unique images, and 322,000 duplicates of one image's data. This wasn't satisfactory.

We tweaked the bash script to instead query for up to 100 results per day between 2012-09-30 and 2016-09-30. This returned less images, but we had no duplicate data. So we tweaked it again to query for a second "page" of data if there was more than 100 results on a given day, and to query for a third page if there was still more, and so on. This gave us 1,688 JSON files (1 file per "page"), which contained 109,593 unique images. The Java class we wrote for this can be seen in the appendix.

These 109,593 images were taken within 3km radius of a spot on Fleet St., Dublin 2.

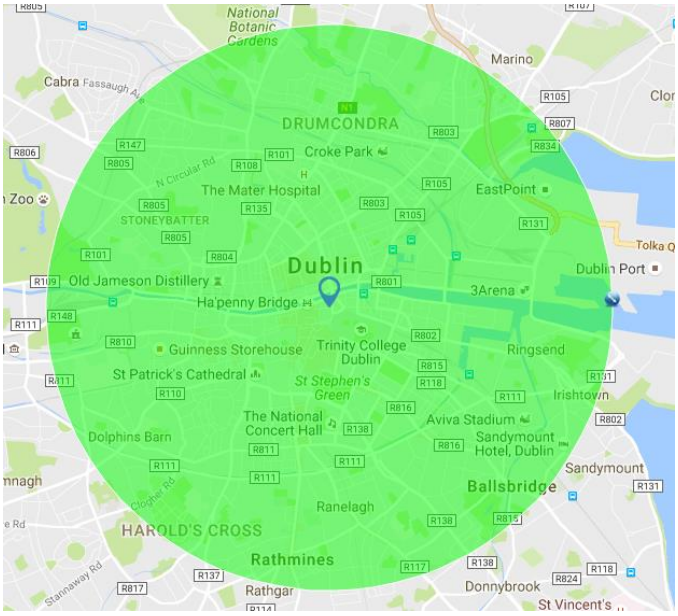

Image 1

Our reasoning for choosing this location is that, within this area we have included several different popular tourist locations such as;

1. Croke Park
2. The Old Jameson Distillery
3. Ha'penny Bridge
4. The Spire
5. 3Arena
6. Guinness Storehouse
7. Trinity College
8. St. Stephens Green
9. Aviva Stadium

Our dataset consisted of 27 unique attributes, all details relating to one specific image and some metadata describing certain attributes.

The values for these metadata attributes aren't inherently understood until they are read from the Flickr API documentation (https://www.flickr.com/services/api/), in which they are further described. For instance;

- “datetakengranularity” describes the accuracy of the “datetaken” attribute, using the values 0, 4, 6 or 8 with the following representations.

Value	“datetaken” represented accuracy
0	Y-m-d H:i:s
4	Y-m
6	Y
8	Circa

Table 1

- “datetakenunknown” describes whether the “datetaken” value is known or unknown with the values 0 or 1 respectively.
- “accuracy” describes how accurate the “latitude” & “longitude” values are using the values 0 to 16, representing the least and most accurate respectively.
- “context” describes whether the image was taken outdoor or indoor using the values 0 or 1 respectively.

4 DATA PREPARATION

4.1 PRE-PROCESSING

So now that we had a satisfactory amount of data, our next problem was that we needed it all in a single CSV file.

We wrote our own Java program (CombineJSON.java) to combine the JSON files into one huge JSON file, stripping away the useless meta-data (e.g. page number), headers and footers in each JSON file and concatenating the results together. We then used a tool called json2csv (https://www.npmjs.com/package/json2csv) to convert our new JSON file to CSV. From this we chose 54,593 of our 109,593 tuples at random to use as our training data. The following data preparation is applied solely to this training dataset and has been written entirely in Java.

The next step in our pre-processing was to isolate and eliminate attributes that we deemed useless with regards to our prediction. These attributes were as follows;

- secret - part of the image URL that uniquely identifies images
- server - which Flickr server the image has been stored on
- farm - which group of Flickr servers the image is stored in
- ispublic - whether the image is publicly accessible
- isfriend - whether the owner of the image is a Flickr friend to me (the user scraping the API)
- isfamily - whether the owner of the image is part of the my Flickr family
- geo_is_public
- geo_is_friend
- geo_is_family
- geo_is_contact

We cannot find a precise definition for the final four attributes stated above, but our reason for removing them is that we believe that we can say they are in the same vein as the “ispublic”, “isfriend” and “isfamily” attributes.

We then removed some tuples that we thought would cause outliers when it comes to making our prediction. We removed a tuple if it satisfied any one of the following conditions;

Attribute	When value is	Reason
datetakengranularity	not 0	tuples that have any other value for this attribute will have an inaccurate “datetaken” value
datetakenunknown	not 0	tuples that have any other value for this attribute will not have a “datetaken” value
accuracy	not 16 or 15	tuples that have any other value for this attribute will have inaccurate “latitude” and “longitude” values

Table 2

Now that these attributes have been used, we can remove them from our dataset as we deem the “datetaken”, “latitude” and “longitude” values to be accurate enough for our prediction.

4.2 Creating Classifications

Classifications are an integral part of making a data mining prediction and as our dataset did not have any, we had to create it. We looked at the five number summary for the “views” attribute (since this is what we are basing our prediction on).

Min	0
Q1	17
Median	63
Q3	229
Max	98001

Table 3

This creates the following classifications;

Low	0 -> 17
Medium	18 -> 63
High	64 -> 229
Very High	230 -> 98001

Table 4

We did consider using standard number ranges for our classification, by splitting the max value into 4 equally weighted ranges. This would give us the following classes;

Low	0 -> 24498
Medium	24499 -> 48999
High	49000 -> 73500
Very High	73501 -> 98001

Table 5

As you can see from table 3, that our data is positively skewed. The problem with this is that if we were to use the classifications in table 5, before ever looking at another attribute, we could still make a lazy prediction that any image would be classed as having a low number of views. This is simply because the majority of images in our dataset would be classed as low. Once we decided on a set of classifications, we added the appropriate classes to each tuple under the attribute header of “views_classification”.

4.3 Discrepancy Detection

There was only one attribute that can be accredited to instances of field overloading in our dataset. This was in every value for the "datetaken" attribute as the value was saved in the format YYYY-MM-DD hh:mm:ss, when really this attribute should only contain the date of the image when it was taken and not its time. Since we know that all the values for "datetaken" in this dataset are in the same format, we split the values by separating them by a space character. We saved exclusively the date value into "datetaken" and created a new attribute called "timetaken" to save exclusively the time value.

There was metadata left by Flickr in some of the values for the "tags" attribute. Flickr adds tags to images when they are uploaded to Instagram, Foursquare or by the Flickr mobile app. For each tuple, we removed these metadata tags from the "tags" attribute then added a new attributes "flickr_app", "instagram_app" and "foursquare_venue" with values indicating whether these tags occurred. We did this because we feel that if an image is linked on another site, this could directly affect the views it receives on Flickr as it has a wider audience that other images do not have, thus creating a higher view count.

4.4 Missing Values

There were only two attributes in our dataset that contained missing values, namely "tags" and "title". We decided not to fill in this attribute with a global constant as this is not a mistake in our dataset, but merely an intention by the user when uploading their image. Although, we did create two new metadata attributes for each tuple based on the length of the images title and the number of tags that it had.

- "count_tags" was calculated by counting the number of words in the "tags" attribute that are separated by a space character.
- "title_length" was calculated by simply counting the number of characters in the "title" attribute

We did this as it is logical that an image with a low number of tags (or a short title) would get considerably less views compared to an image that has a high number of tags (or a long title).

4.5 Workshop Findings and Results

We learned from preparing our data that, no matter what source you receive your dataset from, it isn't always suitable to be analysed immediately. Each step we took to clean our data, we feel, is integral in creating an accurate prediction. When a dataset has a large number of attributes it doesn't necessarily mean that it is clean and with that, the inverse is also true. We removed attributes we thought were useless and added attributes we felt were mandatory for our analysis. We also removed tuples were believed would cause outliers in our prediction. We felt that our dataset was moderately clean but wasn't quite to our liking. We attempted to transform our data to make our prediction easier, and we did this without changing any values.

5 Algorithm Description

In this section we will explain, in order, our thought process and reasoning with regards to our chosen prediction algorithm and its implementation.

5.1 CLUSTERING

The reason for deciding to use clustering is that our aim is cluster images with similar attribute values together. This will give us clusters of images with similar image photography behaviour. We are looking to gain insight into the correlation of image location, time, the number of tags and the length of title relating to the number of views.

After much discussion and analysis of our dataset we originally believed that the best suited algorithm and method for constructing a model for our dataset would be to use the k-means algorithm. This algorithm is an exclusive clustering algorithm meaning that each data object can only exist in one cluster. It groups a specific attribute (data object) into a k number of groupings (clusters) based on a k number of centroids (i.e. each cluster contains only one centroid). We choose these k centroid positions initially (typically ones that are a far distance apart), but we recalculate the centroids of these clusters and repeat the clustering algorithm (assigning each object to the cluster with the nearest centroid) until there is no difference in iterations.

Our attempt at clustering title_length against views falters, this because there is a heavily positive skew in title_length, so it seems as if there is a high correlation between using a shorter title_length and a large number of views. This is counter-intuitive to our original thinking as, logically, it would make more sense to have a large number of views if your image had a longer title.

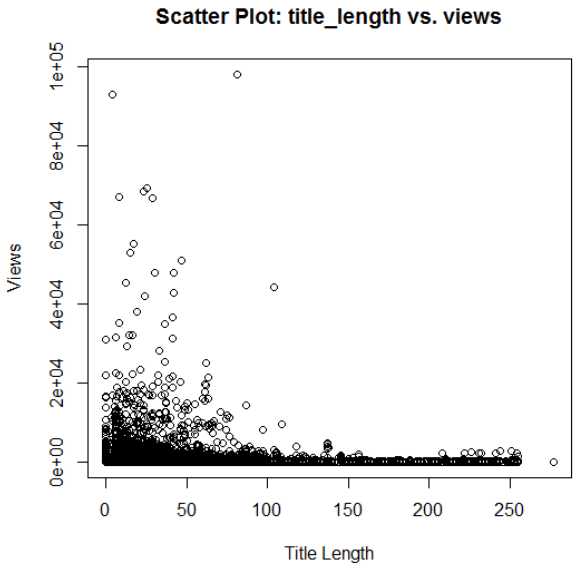

Image 2

5.1.1 Classification for title_length

We attempted to create classifications for title_length (similar to that of views) to group these views, we thought that maybe this would help in visualising the correlation better. We created the classes based on its five number summary.

Min	0
Q1	0
Median	4
Q3	10
Max	75

Table 6

Even by viewing this we can see its positive skew. This gave us the following classes;

Low	0 -> 4
Medium	5 -> 10
High	11 -> 75

Table 7

When it came to create a scatter plot of title_length_classification vs views we see a graph of 3 lines equidistant apart. This is not very useful (even worse, in fact), as the class values manipulate the raw values so that they are no longer continuous.

5.1.2 Standardisation of title_length

For confirmation that title_length is skewed we create a histogram of its values and their corresponding frequencies.

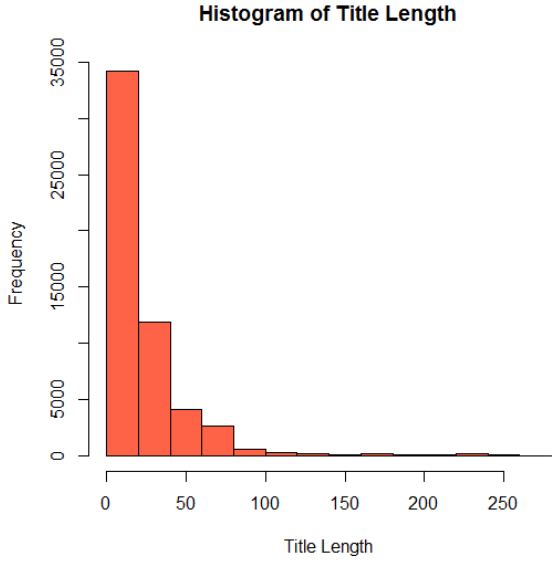

Image 3

Just by looking at its shape we can see it's positive skew. To minimise skewness, we used a log transform. However, when there are zero values (or even negative values) in the data, this will lead to some of the values in the transformed variable being undefined. For this reason, a constant c, is sometimes added to the variable before taking a log transform. We used $f(x_i) = \log(x_i + c)$, where x_i is the i^{th} observed value and f is the transformation function. Typically c is 1, but with a very positively skewed distribution, it will be a higher value. Next, we had to try and find a value for c that brings the skewness as close to zero as possible. Our definition of skewness is;

$$\gamma_1 = \mathrm{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu_3}{\sigma^3} = \frac{\mathrm{E}\left[(X-\mu)^3\right]}{(\mathrm{E}[(X-\mu)^2])^{3/2}} = \frac{\kappa_3}{\kappa_2^{3/2}}$$

where μ is the mean, σ is the standard deviation, E is the expectation value (the long-run average value of repetitions of the experiment it represents), μ_3 is the third central moment (the quantitative measure of shape) and k_t are the t^{th} cumulants (a set of quantities that provide an alternative to the moments of the distribution). Calling this function on our attribute values gives us

the value 3.401073. To find the optimum value for *c* we created a graph that plots *c* against the skewness for *c*.

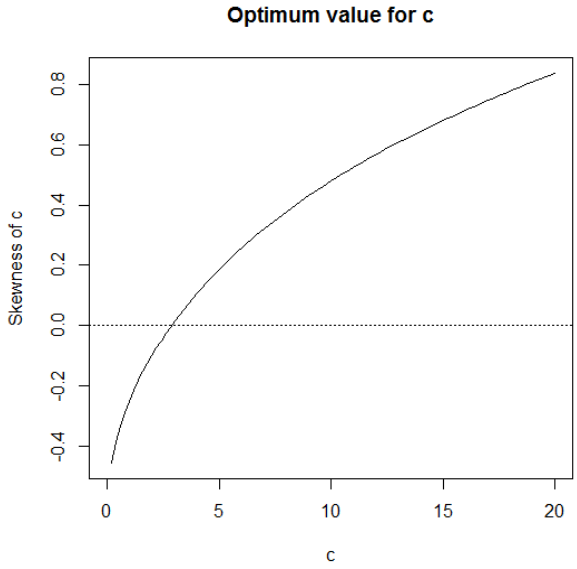

Image 4

From this we can see that the optimum value for c is approximately 3. So applying our log transform with the value we got for c, our histogram for displaying its values and their corresponding frequencies we looks as follows.

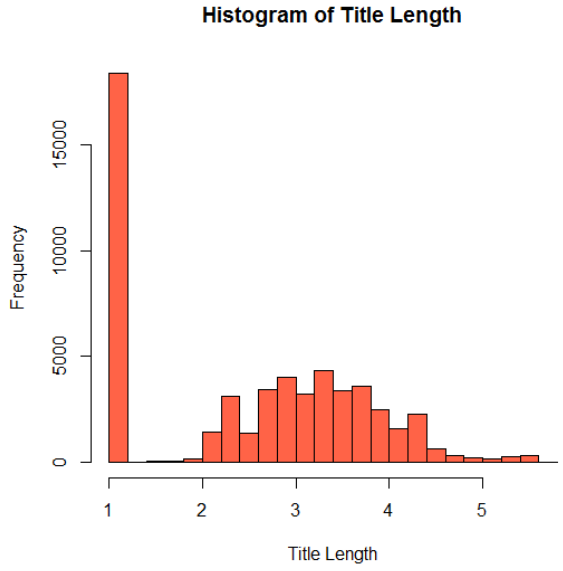

Image 5

As we can see we've removed our skewness, but haven't removed our outliers. Our reasons for removing skewness is that since the Euclidean distance is used, the clusters will be influenced strongly by the magnitudes of the variables, especially by outliers. So we will need to reduce large distances between values since data is positively skewed (i.e. images can have title length of 45 or even 175).

5.1.3 Difference between normalised and standardised

They are both used to bring all of the variables into proportion with one another. Normalisation is the adjusting of values that are measured on different scales to a notionally common scale. Standardisation is used to help to maximize compatibility and interoperability. Standardisation, in our case, is used to remove skewness, but that process does not remove outliers.

5.2 Analysis of Clustering and Reason For Choosing Decision Tree

After looking at several scatter plots for our k-means output, we concluded that this method may not be the best algorithm for our prediction as we cannot decipher anything meaningful from our output. Our clustering was hard to understanding as it was difficult to interpret. We could not see any initial correlation and there are no completely satisfactory methods for determining the appropriate number of clusters. No matter what number of clusters we chose we still didn't see a strong correlation. With this, there is a difficulty in comparing quality of the clusters produced as the results seem to be extremely similar.

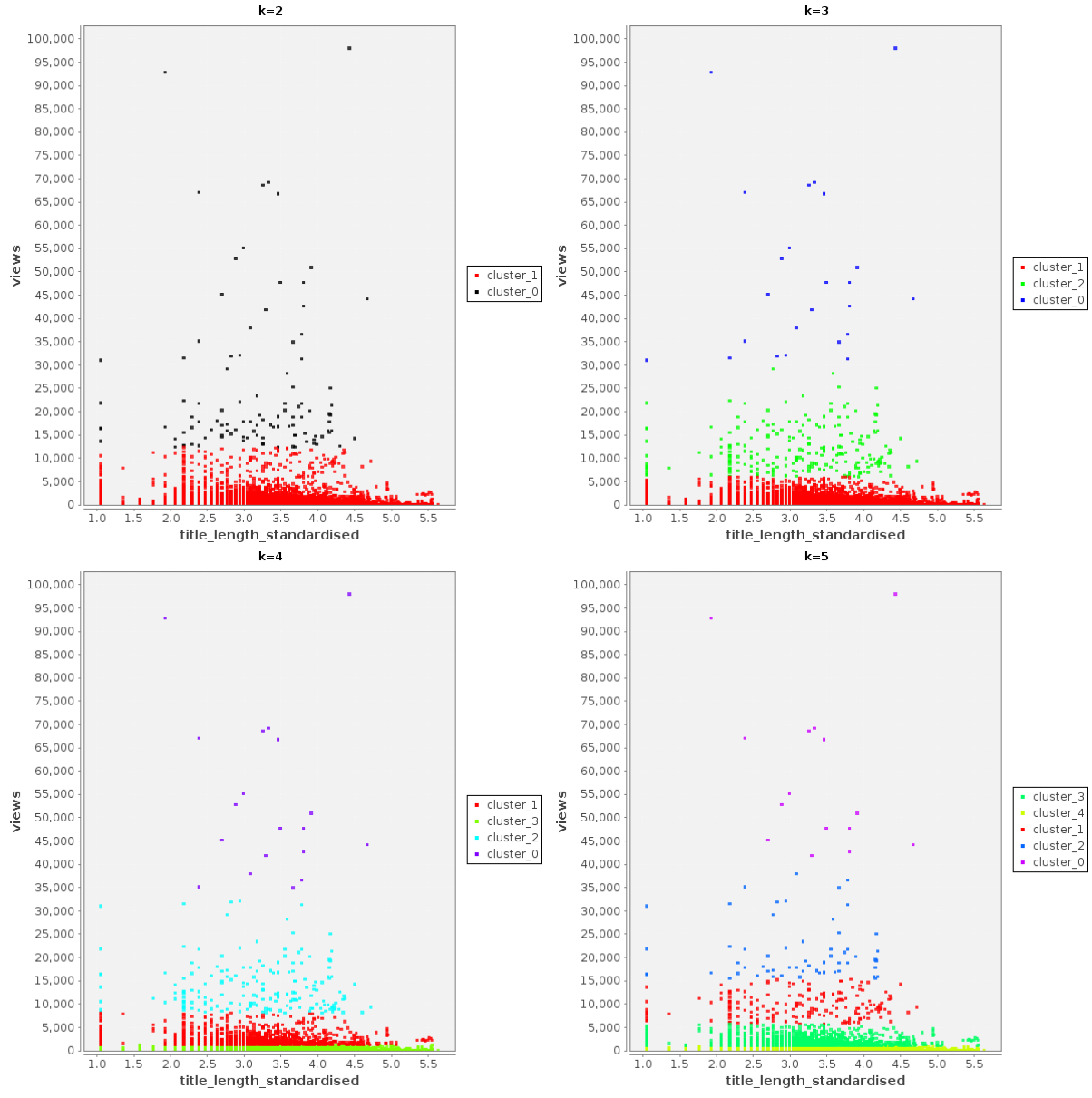

Image 6

However, this is not for nothing as we can use our small clustering analysis to remove outliers, thusly creating a better prediction in our decision tree.

5.3 Decision Tree

The reason for deciding to use classification is that our aim is to create a model or classifier that will be constructed to predict categorical labels. After our analysis from our clustering attempt, we agreed that classification would be the more suited in making our prediction. A decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. Each tree has both decision nodes and leaf nodes. Each decision tree has the following components;

1. Each internal node represents a "test" on an attribute (e.g. whether an image was uploaded to Instagram, what season it was taken in, etc.)
2. Each branch represents the outcome (e.g. true or false, summer/winter, etc.)
3. Each leaf node represents a class label (a decision taken after computing all attributes)

Each individual path from root node to leaf node represents a classification rule.

When creating our decision tree we decided that we created classifications for several attributes this is so that it is easier for us to make a tree (i.e. easier in the sense that each decision node will now have viewer branches). Therefore, we created the following classes;

Original Attribute	**New Attribute**	**Value**	**Classes**
timetaken	taken_in_am	hh value in hh:mm:ss	True – 07->18 False – 00->06 || 19->23
datetaken	season	MM value in YYYY-MM-DD	winter - 11, 12, 01 spring - 02, 03, 04 summer - 05, 06, 07 autumn - 08, 09, 10
title_length	title_length_classification	Raw Value	Low - 0 Medium - 1->13 High - 14->30 Very high - 31+
count_tags	count_tags_classification	Raw Value	Low - 0 Medium - 1->4 High - 5->10 Very high - 11+

Table 8

From there we created a tree to classify images from each classification of views we created earlier (i.e. low, medium, high, very high). These are the attributes we chose to use in our decision tree;

- title_length_classification
- count_tags _classification
- season
- taken_in_am
- flickr_app
- instagram_app
- foursquare_venue

Using Knime we constructed a decision tree based on the above attributes. Knime implements the C4.5 algorithm for creating decision trees. This algorithm has three base cases;

1. All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
2. None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
3. Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

And works in the following steps:

1. Check for the above base cases.
2. For each attribute a, find the normalized information gain ratio from splitting on a.
3. Let a_best be the attribute with the highest normalized information gain.
4. Create a decision node that splits on a_best.
5. Recur on the sublists obtained by splitting on a_best, and add those nodes as children of node.

For our decision tree count_tags_classification is our root node, as this is the first attribute we will evaluate. Intuitively, this makes sense, that an image with a high number of tags will have a more chance of having a high number of views. This tree is better described in the image attached (named "decision-tree.png"). As we can see in this diagram, each traversal will result in a deciding classification of views and will also generate it's own unique rule.

This is how to read the tree diagram, from the training data set, 50% of the images have a very high class of views if it was taken in winter, not taken during the day and has a very high number of tags. Also, 66.7% of the images have a low class of views if it was taken on an autumn morning and uploaded using the Flickr app using a long title.

The classification of the number of views of an image is decided upon by taking into account different attribute scenarios. After we have constructed the decision tree, we can start classifying some images. This is done by starting at the root node and traversing through the tree by checking if your data has the feature from the node.

5.3.1 Process of Running and Generating Our Prediction

After creating the decision tree from Knime we are given a set of approximately 120 unique rules. These rules are used like a test, to create a predicted classification for each tuple in our test dataset. The prediction is generated when a tuple satisfies all the conditions in a specific rule. We did this by iterating through each tuple in our test dataset, and adding a field called "predicted_classificaton". We then iterated through this dataset again, comparing the predicted_classification value to the views_classification value. We added up all the correct predictions and divided that into the total number of tuples to create our prediction accuracy.

6 FINAL RESULTS AND ANALYSIS

The following section provides a detailed description of the results and findings from our implemented algorithm on our test dataset.

6.1 Analysis of Decision Tree

The first thing we notice is that the high percentage of certainties for predicting our classification, only occur when the number of patterns covered by the node is below 50. Meaning that these high certainties only occur when the rules are applied to a small number of test tuples. For example, if we look at the conditions where;

- flickr_app = true
- instagram_app = true
- season = winter
- title_length_classification = high
- count_tags_classification = medium

We can say with 75% certainty that an image that satisfies those conditions will have a high number of views. But the problem with that is only 20 images were analysed.

Inversely, a node which has a high number of patterns (typically around 150) doesn't give a strong prediction of what views class that image lays in. For example, if we look at the conditions where;

- season = spring
- instagram_app = true
- title_length_classification = very high
- count_tags_classification = medium

After analysing 402 images, we can only say with 59% certainty that that image will have a medium number of views.

Sometimes we had to look for negative certainties in our tree (that is, we look for things that are not likely to happen), for example if the following criteria were met;

- instagram_app = true
- season = winter
- title_length_classification = medium
- count_tags_classification = medium

Then we have 100% confidence that a photo will not receive a very high number of views, because all of the images with those attributes have either a low, medium or high view classification.

We find it interesting that images that have a high number of tags and were uploaded to Instagram are most likely to have anywhere between 18 and 63 views (medium classification). This is because, in this scenario, the user uploads the image to their Instagram account with which they have their Flickr account connected. Meaning that Flickr is seen as an afterthought in this situation. But the medium number of views is understandably seen as the tags which the user applies to their Instagram image will also be applied to their Flickr image (i.e. more ways that image can be searched). We found some interesting predictions in some nodes in our tree that back this up. For example, of the 62 photos that had a prediction of a very high views classification that met the following criteria;

- title_length_classification = medium
- count_tags_classification = medium
- season = winter

In the child nodes that were generated based on the instagram_app value, the values for very high views prediction were as follows:

- instagram_app = false: 62
- instagram_app = true: 0

Similarly, of the 102 images that had a prediction of a very high views classification that met the following criteria;

- title_length_classification = medium
- count_tags_classification = medium
- flickr_app = false
- Photo taken in morning = false

In the child nodes that were generated based on the instagram_app value, the values for very high views prediction were as follows:

- instagram_app = false: 102
- instagram_app = true: 0

This is even visible at a higher level in our tree. Of the 385 photos that were predicted to have a very high views classification that met the following criteria;

- title_length_classification = very high
- count_tags_classification = medium

In the child nodes that were generated based on the instagram_app value, the values for very high views prediction were as follows:

- instagram_app = false: 356
- instagram_app = true: 29

Evidently, a node which classifies an image as a certain class (in this case let's choose medium) its corresponding child nodes are extremely less likely to have a higher classification (here it would be, high or very high). Although this anomaly does occur in our tree, but only a select few times.

The foursquare_venue attribute doesn't seem to differ in its classifying too much. For each occurrence of that splitting node, the classification for the attribute value being true or false usually only differs from medium to low. Although, there is only one occurrence of the classification jumping more than one class (low for false and high for true), but this node was only analysing 17 images.

As we would expect, there is a direct correlation between the classification of the number of tags and the classification of the number of views. When there is a very high occurrence of tags, with the most certainty, we can say that there will be a very high number of images. This inherently makes sense to us as the number of tags on an image creates a wider audience for your image (casts a broader net).

Were this tree to be duplicated with a different yet similar dataset, then you would most probably find different results because your training data set would be different (even ever so slightly). In theory, if content based filtering is appropriate for view classification of images, then the decision tree becomes more accurate as the size of the training data set gets larger.

6.2 KNIME SCORER NODE

Knime has a scorer node which creates a confusion matrix i.e. how many rows of which attribute and their classification match. This is used to describe the performance of a classification model, in our case, our decision tree.

	Very High	High	Medium	Low	Sum
Very High	6494	2540	3250	1512	13796
High	4009	3957	3353	2419	13738
Medium	604	474	10609	2132	13819
Low	1513	1638	5882	4451	13484
Sum	12620	8609	23094	10514	

Table 9

There are four possible predicted: very high, high, medium and low. Our classifier made a total of 54837 predictions (e.g. 54837 images were being tested for each classification of its views). Out of those 54837 images, the classifier predicted very high 12620 times, high 8609 times, medium 23094 times and low 10514 times. In reality, 13796 images have a very high number of views, 13738 images have a high number of views, 13819 images have a medium number of views and 13484 images have a low number of views.

We looked at the accuracy statistics table generated in the Scorer node. The precision of our predictions was as follows:

- Very high was predicted accurately 51.5% of the time
- High was predicted accurately 46% of the time
- Medium was predicted accurately 45.9% of the time
- Low was predicted accurately 42.3% of the time

6.3 ANALYSIS OF SCORER

Our scorer only got the value of 46.5% accuracy. This value can be improved by including the place_id attribute into the decision tree. Our accuracy value jumps considerably to 66%, we originally wanted to analyse our tuples by their latitude and longitude values, but we were unsure how to correlate them. It turns out that the place_id attribute does that for us. We took this attribute out because we were unsure how to describe its values in relation to each other attribute (i.e. we didn't believe it was a complementary attribute).

There are some things that can be done to improve this;

1. A more accurate prediction would be made on a larger dataset
2. Treating missing values and removing outliers would help, although we have done that. (Not applicable in our case).
3. Similarly with, feature transformation. We created new metadata attributes.
4. Maybe we could have done a better Feature Selection, choosing attributes that better complement themselves to create a subset.

APPENDIX

COMBINEJSON.JAVA

```
import java.io.*;
import java.time.*;
import java.util.*;

public class CombineJSON {
    public static void main(String[] args) {
        try {
          StringBuilder sb = new StringBuilder();
            String tmpJson;
            final File folder = new File ("/home/ryan/CA4010/flickrJSON/");
            for(File f : folder.listFiles()) {
                BufferedReader br = new BufferedReader(new
FileReader(f.getAbsolutePath()));
                tmpJson = br.readLine();
                tmpJson = tmpJson.substring(tmpJson.indexOf("[") + 1);
                tmpJson = tmpJson.substring(0, tmpJson.lastIndexOf("]"));
                tmpJson = tmpJson.replace("},{", "},\n{");
                tmpJson = tmpJson.replace("#", "");
                if(tmpJson.length() != 0) { //dont include empty files
                    sb.append(tmpJson);
                    sb.append(",");
                    sb.append(System.lineSeparator());
                }
                br.close();
            }
            sb.setLength(sb.length() - 2); //Remove final comma and final
"\n"
            PrintWriter writer = new PrintWriter("all.json", "UTF-8");
            writer.print("["); //Write the JSON as a large array
            writer.print(sb.toString());
            writer.print("]");
            writer.flush();
            writer.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```