# Data mining with WEKA, Part 2: Classification and clustering

Michael Abernethy May 11, 2010

Data mining is a collective term for dozens of techniques to glean information from data and turn it into meaningful trends and rules to improve your understanding of the data. In this second article of the "Data mining with WEKA" series, we'll discuss two common data mining methods — classification and clustering — which can be used to do more powerful analysis on your data.

View more content in this series

## Introduction

In Part 1, I introduced the concept of data mining and to the free and open source software Waikato Environment for Knowledge Analysis (WEKA), which allows you to mine your own data for trends and patterns. I also talked about the first method of data mining — regression — which allows you to predict a numerical value for a given set of input values. This method of analysis is the easiest to perform and the least powerful method of data mining, but it served a good purpose as an introduction to WEKA and provided a good example of how raw data can be transformed into meaningful information.



In this article, I will take you through two additional data mining methods that are slightly more complex than a regression model, but more powerful in their respective goals. Where a regression model could only give you a numerical output with specific inputs, these additional models allow you to interpret your data differently. As I said in Part 1, data mining is about applying the right model to your data. You could have the best data about your customers (whatever that even means), but if you don't apply the right models to it, it will just be garbage. Think of this another way: If you only used regression models, which produce a numerical output, how would Amazon be able to tell you "Other Customers Who Bought X Also Bought Y?" There's no numerical function that could give you this type of information. So let's delve into the two additional models you can use with your data.

In this article, I will also make repeated references to the data mining method called "nearest neighbor," though I won't actually delve into the details until Part 3. However, I included it in the comparisons and descriptions for this article to make the discussions complete.

## Classification vs. clustering vs. nearest neighbor

Before we get into the specific details of each method and run them through WEKA, I think we should understand what each model strives to accomplish — what type of data and what goals each model attempts to accomplish. Let's also throw into that discussion our existing model — the regression model — so you can see how the three new models compare to the one we already know. I'll use a real-world example to show how each model can be used and how they differ. The real-world examples all revolve around a local BMW dealership and how it can increase sales. The dealership has stored all its past sales information and information about each person who purchased a BMW, looked at a BMW, and browsed the BMW showroom floor. The dealership wants to increase future sales and employ data mining to accomplish this.

### Regression

Question: "How much should we charge for the new BMW M5?" Regression models can answer a question with a numerical answer. A regression model would use past sales data on BMWs and M5s to determine how much people paid for previous cars from the dealership, based on the attributes and selling features of the cars sold. The model would then allow the BMW dealership to plug in the new car's attributes to determine the price.

Example: `Selling Price = $25,000 + ($2900 * Liters in Engine) + ($9000 * isSedan) + ($11,000 * isConvertible) + ($100 * inches of car) + ($22,000 * isM)`.

### Classification

Question: "How likely is person X to buy the newest BMW M5?" By creating a classification tree (a decision tree), the data can be mined to determine the likelihood of this person to buy a new M5. Possible nodes on the tree would be age, income level, current number of cars, marital status, kids, homeowner, or renter. The attributes of this person can be used against the decision tree to determine the likelihood of him purchasing the M5.

### Clustering

Question: "What age groups like the silver BMW M5?" The data can be mined to compare the age of the purchaser of past cars and the colors bought in the past. From this data, it could be found whether certain age groups (22-30 year olds, for example) have a higher propensity to order a certain color of BMW M5s (75 percent buy blue). Similarly, it can be shown that a different age group (55-62, for example) tend to order silver BMWs (65 percent buy silver, 20 percent buy gray). The data, when mined, will tend to cluster around certain age groups and certain colors, allowing the user to quickly determine patterns in the data.

### Nearest neighbor

Question: "When people purchase the BMW M5, what other options do they tend to buy at the same time?" The data can be mined to show that when people come and purchase a BMW M5, they also tend to purchase the matching luggage. (This is also known as basket analysis). Using

this data, the car dealership can move the promotions for the matching luggage to the front of the dealership, or even offer a newspaper ad for free/discounted matching luggage when they buy the M5, in an effort to increase sales.

## Classification

*Classification* (also known as classification trees or decision trees) is a data mining algorithm that creates a step-by-step guide for how to determine the output of a new data instance. The tree it creates is exactly that: a tree whereby each node in the tree represents a spot where a decision must be made based on the input, and you move to the next node and the next until you reach a leaf that tells you the predicted output. Sounds confusing, but it's really quite straightforward. Let's look at an example.

### Listing 1. Simple classification tree

```
    [ Will You Read This Section? ]
          /              \
        Yes              No
        /                 \
[Will You Understand It?]  [Won't Learn It]
    /         \
  Yes          No
  /             \
  [Will Learn It]  [Won't Learn It]
```

This simple classification tree seeks to answer the question "Will you understand classification trees?" At each node, you answer the question and move on that branch, until you reach a leaf that answers yes or no. This model can be used for any unknown data instance, and you are able to predict whether this unknown data instance will learn classification trees by asking them only two simple questions. That's seemingly the big advantage of a classification tree — it doesn't require a lot of information about the data to create a tree that could be very accurate and very informative.

One important concept of the classification tree is similar to what we saw in the regression model from Part 1: the concept of using a "training set" to produce the model. This takes a data set with known output values and uses this data set to build our model. Then, whenever we have a new data point, with an unknown output value, we put it through the model and produce our expected output. This is all the same as we saw in the regression model. However, this type of model takes it one step further, and it is common practice to take an entire training set and divide it into two parts: take about 60-80 percent of the data and put it into our training set, which we will use to create the model; then take the remaining data and put it into a test set, which we'll use immediately after creating the model to test the accuracy of our model.

Why is this extra step important in this model? The problem is called *overfitting:* If we supply *too much* data into our model creation, the model will actually be created perfectly, but just for that data. Remember: We want to use the model to predict future unknowns; we don't want the model to perfectly predict values we already know. This is why we create a test set. After we create the model, we check to ensure that the accuracy of the model we built doesn't decrease with the test set. This ensures that our model will accurately predict future unknown values. We'll see this in action using WEKA.

This brings up another one of the important concepts of classification trees: the notion of pruning. *Pruning,* like the name implies, involves removing branches of the classification tree. Why would someone want to remove information from the tree? Again, this is due to the concept of overfitting. As the data set grows larger and the number of attributes grows larger, we can create trees that become increasingly complex. Theoretically, there could be a tree with `leaves = (rows * attributes)`. But what good would that do? That won't help us at all in predicting future unknowns, since it's perfectly suited only for our existing training data. We want to create a balance. We want our tree to be as simple as possible, with as few nodes and leaves as possible. But we also want it to be as accurate as possible. This is a trade-off, which we will see.

Finally, the last point I want to raise about classification before using WEKA is that of false positive and false negative. Basically, a false positive is a data instance where the model we've created predicts it should be positive, but instead, the actual value is negative. Conversely, a false negative is a data instance where the model predicts it should be negative, but the actual value is positive.

These errors indicate we have problems in our model, as the model is incorrectly classifying some of the data. While some incorrect classifications can be expected, it's up to the model creator to determine what is an acceptable percentage of errors. For example, if the test were for heart monitors in a hospital, obviously, you would require an extremely low error percentage. On the other hand, if you are simply mining some made-up data in an article about data mining, your acceptable error percentage can be much higher. To take this even one step further, you need to decide what percent of false negative vs. false positive is acceptable. The example that immediately comes to mind is a spam model: A false positive (a real e-mail that gets labeled as spam) is probably much more damaging than a false negative (a spam message getting labeled as not spam). In an example like this, you may judge a minimum of 100:1 false negative:positive ratio to be acceptable.

OK — enough about the background and technical mumbo jumbo of the classification trees. Let's get some real data and take it through its paces with WEKA.

## WEKA data set

The data set we'll use for our classification example will focus on our fictional BMW dealership. The dealership is starting a promotional campaign, whereby it is trying to push a two-year extended warranty to its past customers. The dealership has done this before and has gathered 4,500 data points from past sales of extended warranties. The attributes in the data set are:

- Income bracket [0=$0-$30k, 1=$31k-$40k, 2=$41k-$60k, 3=$61k-$75k, 4=$76k-$100k, 5=$101k-$150k, 6=$151k-$500k, 7=$501k+]
- Year/month first BMW bought
- Year/month most recent BMW bought
- Whether they responded to the extended warranty offer in the past

Let's take a look at the Attribute-Relation File Format (ARFF) we'll use in this example.

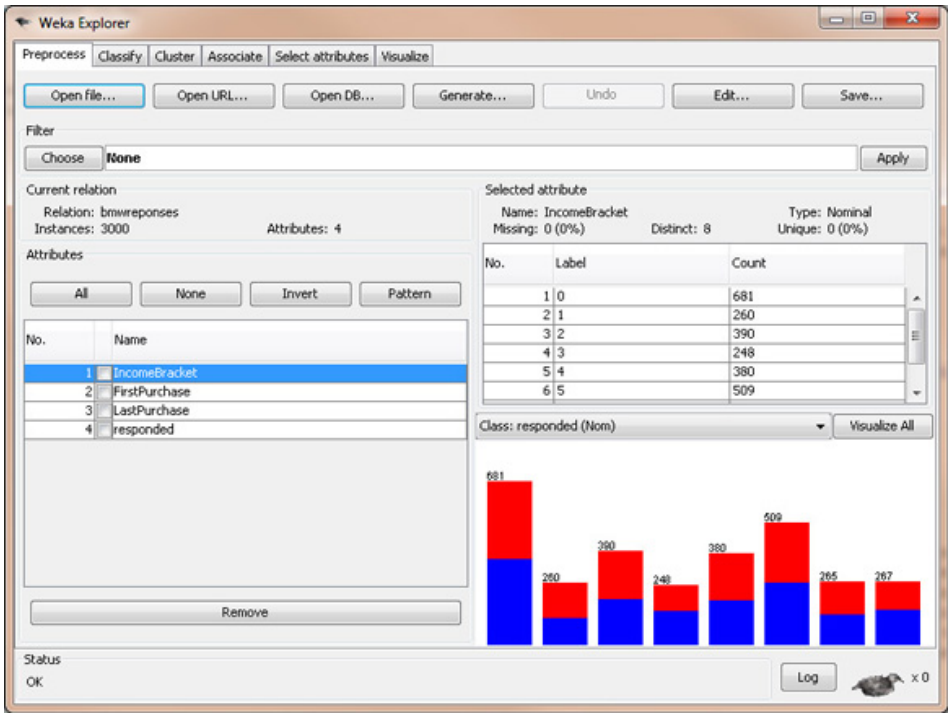
## Listing 2. Classification WEKA data

```
@attribute IncomeBracket {0,1,2,3,4,5,6,7}
@attribute FirstPurchase numeric
@attribute LastPurchase numeric
@attribute responded {1,0}

@data

4,200210,200601,0
5,200301,200601,1
...
```

## Classification in WEKA

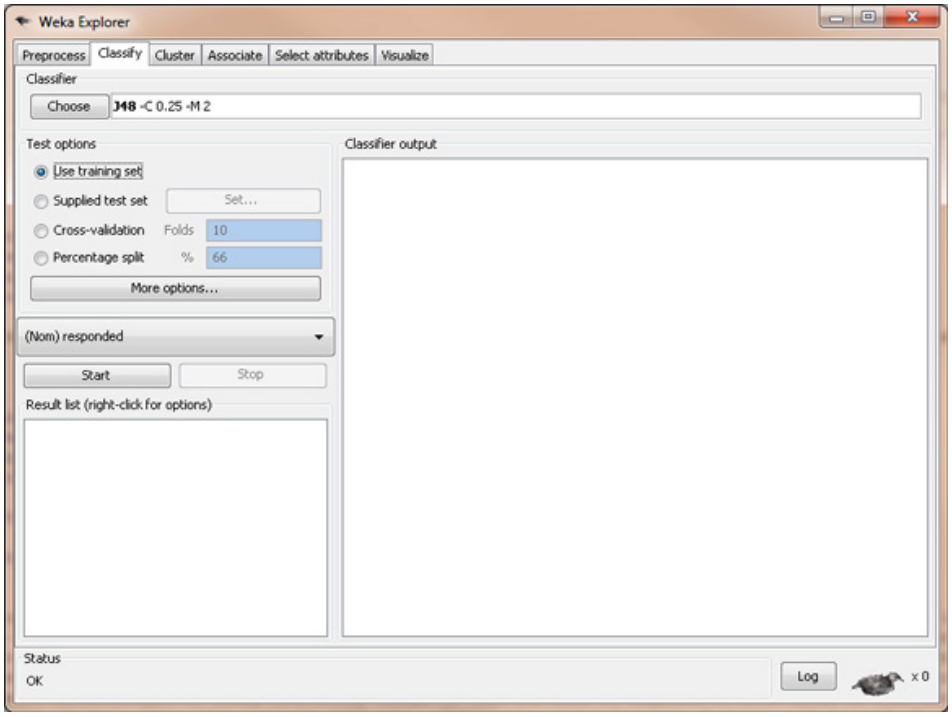
Load the data file bmw-training.arff (see Download) into WEKA using the same steps we've used up to this point. Note: This file contains only 3,000 of the 4,500 records that the dealership has in its records. We need to divide up our records so some data instances are used to create the model, and some are used to test the model to ensure that we didn't overfit it. Your screen should look like Figure 1 after loading the data.

## Figure 1. BMW classification data in WEKA




Like we did with the regression model in Part 1, we select the **Classify** tab, then we select the **trees** node, then the **J48** leaf (I don't know why this is the official name, but go with it).

## Figure 2. BMW classification algorithm




At this point, we are ready to create our model in WEKA. Ensure that **Use training set** is selected so we use the data set we just loaded to create our model. Click **Start** and let WEKA run. The output from this model should look like the results in Listing 3.

## Listing 3. Output from WEKA's classification model

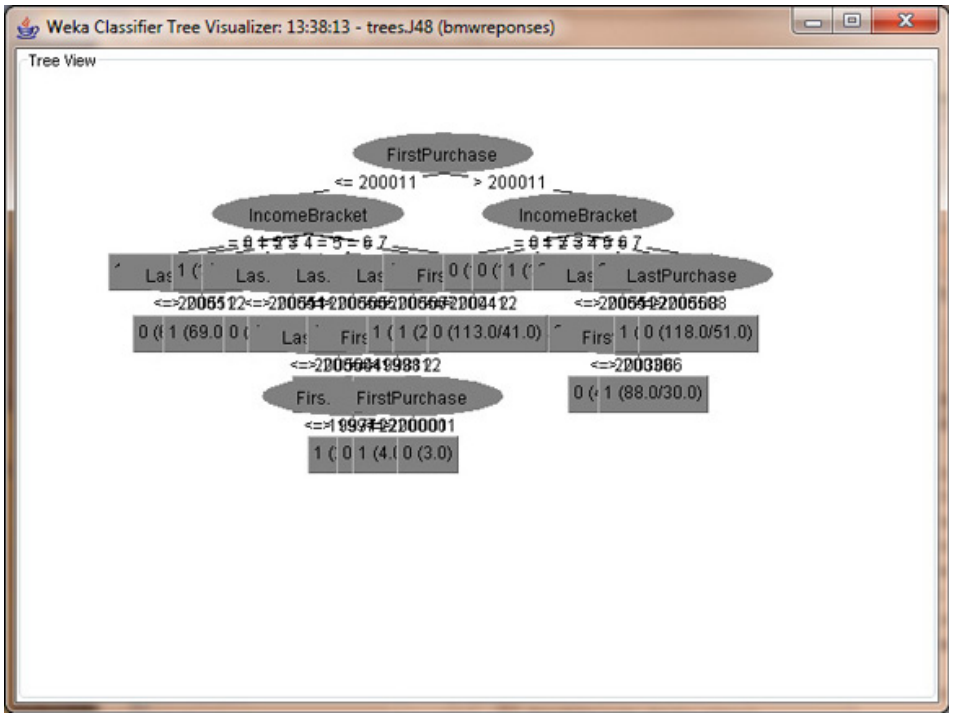
```
Number of Leaves  :  28

Size of the tree :  43


Time taken to build model: 0.18 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances        1774               59.1333 %
Incorrectly Classified Instances      1226               40.8667 %
Kappa statistic                          0.1807
Mean absolute error                      0.4773
Root mean squared error                  0.4885
Relative absolute error                 95.4768 %
Root relative squared error             97.7122 %
Total Number of Instances             3000

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.662     0.481      0.587     0.662     0.622      0.616    1
                 0.519     0.338      0.597     0.519     0.555      0.616    0
Weighted Avg.    0.591     0.411      0.592     0.591     0.589      0.616

=== Confusion Matrix ===

    a    b   <-- classified as
 1009  516 |    a = 1
```

```
710   765 |    b = 0
```

What do all these numbers mean? How do we know if this is a good model? Where is this so-called "tree" I'm supposed to be looking for? All good questions. Let's answer them one at a time:

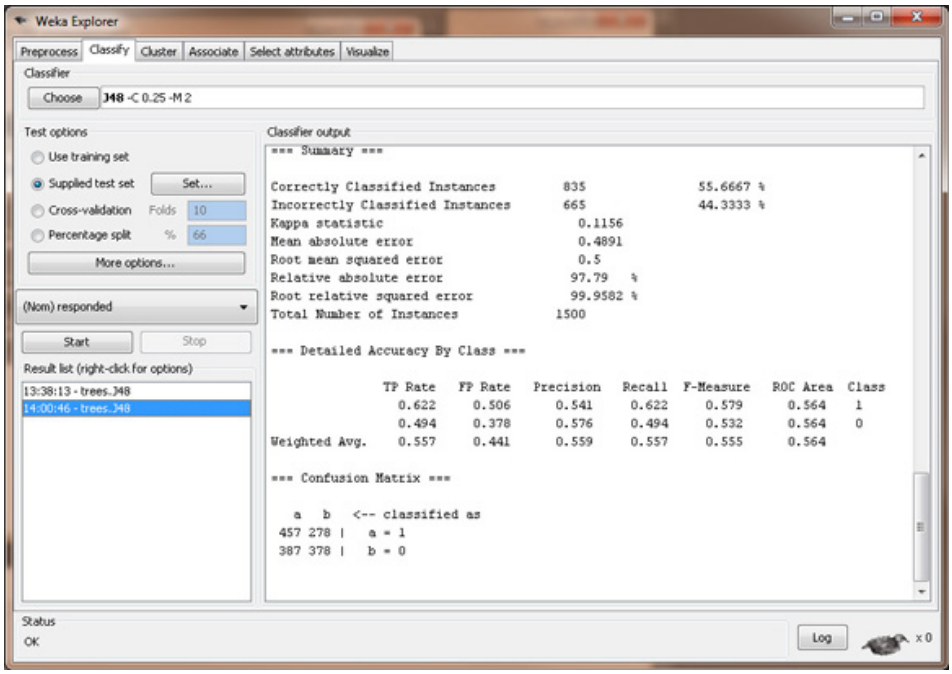
- **What do all these numbers mean?** The important numbers to focus on here are the numbers next to the "Correctly Classified Instances" (59.1 percent) and the "Incorrectly Classified Instances" (40.9 percent). Other important numbers are in the "ROC Area" column, in the first row (the 0.616); I'll explain this number later, but keep it in mind. Finally, in the "Confusion Matrix," it shows you the number of false positives and false negatives. The false positives are 516, and the false negatives are 710 in this matrix.
- **How do we know if this is a good model?** Well, based on our accuracy rate of only 59.1 percent, I'd have to say that upon initial analysis, this is not a very good model.
- **Where is this so-called tree?** You can see the tree by right-clicking on the model you just created, in the result list. On the pop-up menu, select **Visualize tree**. You'll see the classification tree we just created, although in this example, the visual tree doesn't offer much help. Our tree is pictured in Figure 3. The other way to see the tree is to look higher in the Classifier Output, where the text output shows the entire tree, with nodes and leaves.

## Figure 3. Classification tree visualization




There's one final step to validating our classification tree, which is to run our test set through the model and ensure that accuracy of the model when evaluating the test set isn't too different from the training set. To do this, in **Test options**, select the **Supplied test set** radio button and click **Set**. Choose the file bmw-test.arff, which contains 1,500 records that were not in the training set we used to create the model. When we click **Start** this time, WEKA will run this test data set through the model we already created and let us know how the model did. Let's do that, by clicking **Start**. Below is the output.

## Figure 4. Classification tree test




Comparing the "Correctly Classified Instances" from this test set (55.7 percent) with the "Correctly Classified Instances" from the training set (59.1 percent), we see that the accuracy of the model is pretty close, which indicates that the model will not break down with unknown data, or when future data is applied to it.

However, because the accuracy of the model is so bad, only classifying 60 perent of the data records correctly, we could take a step back and say, "Wow. This model isn't very good at all. It's barely above 50 percent, which I could get just by randomly guessing values." That's entirely true. That takes us to an important point that I wanted to secretly and slyly get across to everyone: Sometimes applying a data mining algorithm to your data will produce a bad model. This is especially true here, and it was on purpose.

I wanted to take you through the steps to producing a classification tree model with data that seems to be ideal for a classification model. Yet, the results we get from WEKA indicate that we were wrong. A classification tree is *not* the model we should have chosen here. The model we created tells us absolutely nothing, and if we used it, we might make bad decisions and waste money.

Does that mean this data can't be mined? The answer is another important point to data mining: the nearest-neighbor model, discussed in a future article, will use this same data set, but will create a model that's over 88 percent accurate. It aims to drive home the point that you have to choose the right model for the right data to get good, meaningful information.

**Further reading**: If you're interested in learning more about classification trees, here are some keywords to look up that I didn't have space to detail in this article: ROC curves, AUC, false

positives, false negatives, learning curves, Naive Bayes, information gain, overfitting, pruning, chi-square test

## Clustering

*Clustering* allows a user to make groups of data to determine patterns from the data. Clustering has its advantages when the data set is defined and a general pattern needs to be determined from the data. You can create a specific number of groups, depending on your business needs. One defining benefit of clustering over classification is that every attribute in the data set will be used to analyze the data. (If you remember from the classification method, only a subset of the attributes are used in the model.) A major disadvantage of using clustering is that the user is required to know ahead of time how many groups he wants to create. For a user without any real knowledge of his data, this might be difficult. Should you create three groups? Five groups? Ten groups? It might take several steps of trial and error to determine the ideal number of groups to create.

However, for the average user, clustering can be the most useful data mining method you can use. It can quickly take your entire set of data and turn it into groups, from which you can quickly make some conclusions. The math behind the method is somewhat complex and involved, which is why we take full advantage of the WEKA.

### Overview of the math

This should be considered a quick and non-detailed overview of the math and algorithm used in the clustering method:

1. Every attribute in the data set should be normalized, whereby each value is divided by the difference between the high value and the low value in the data set for that attribute. For example, if the attribute is age, and the highest value is 72, and the lowest value is 16, then an age of 32 would be normalized to 0.5714.
2. Given the number of desired clusters, randomly select that number of samples from the data set to serve as our initial test cluster centers. For example, if you want to have three clusters, you would randomly select three rows of data from the data set.
3. Compute the distance from each data sample to the cluster center (our randomly selected data row), using the least-squares method of distance calculation.
4. Assign each data row into a cluster, based on the minimum distance to each cluster center.
5. Compute the *centroid,* which is the average of each column of data using only the members of each cluster.
6. Calculate the distance from each data sample to the centroids you just created. If the clusters and cluster members don't change, you are complete and your clusters are created. If they change, you need to start over by going back to step 3, and continuing again and again until they don't change clusters.

Obviously, that doesn't look very fun at all. With a data set of 10 rows and three clusters, that could take 30 minutes to work out using a spreadsheet. Imagine how long it would take to do by hand if you had 100,000 rows of data and wanted 10 clusters. Luckily, a computer can do this kind of computing in a few seconds.

## Data set for WEKA

The data set we'll use for our clustering example will focus on our fictional BMW dealership again. The dealership has kept track of how people walk through the dealership and the showroom, what cars they look at, and how often they ultimately make purchases. They are hoping to mine this data by finding patterns in the data and by using clusters to determine if certain behaviors in their customers emerge. There are 100 rows of data in this sample, and each column describes the steps that the customers reached in their BMW experience, with a column having a 1 (they made it to this step or looked at this car), or 0 (they didn't reach this step). Listing 4 shows the ARFF data we'll be using with WEKA.

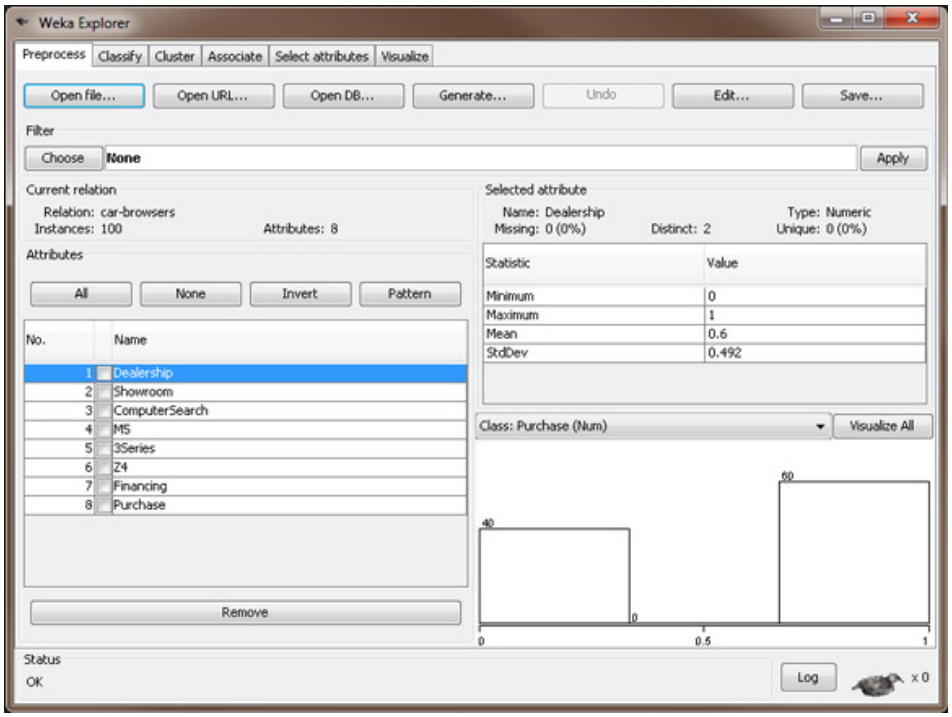
## Listing 4. Clustering WEKA data

```
@attribute Dealership numeric
@attribute Showroom numeric
@attribute ComputerSearch numeric
@attribute M5 numeric
@attribute 3Series numeric
@attribute Z4 numeric
@attribute Financing numeric
@attribute Purchase numeric

@data

1,0,0,0,0,0,0,0
1,1,1,0,0,0,1,0
...
```

## Clustering in WEKA

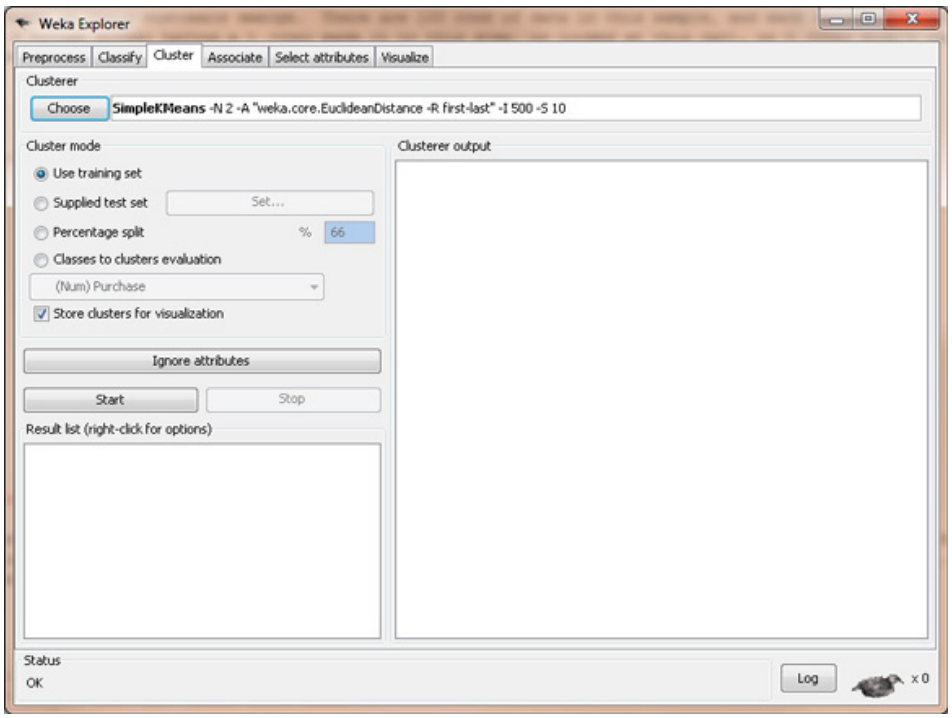
Load the data file bmw-browsers.arff into WEKA using the same steps we used to load data into the **Preprocess** tab. Take a few minutes to look around the data in this tab. Look at the columns, the attribute data, the distribution of the columns, etc. Your screen should look like Figure 5 after loading the data.

## Figure 5. BMW cluster data in WEKA



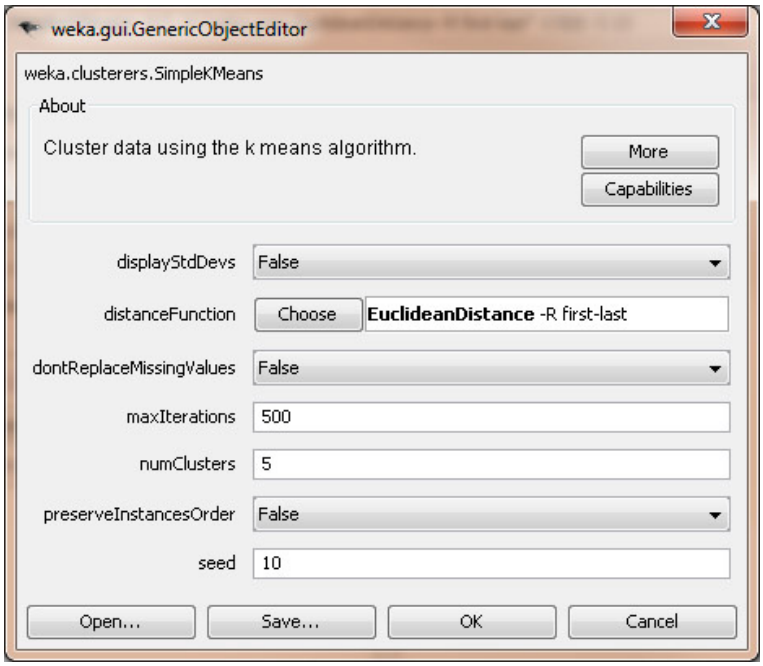

With this data set, we are looking to create clusters, so instead of clicking on the **Classify** tab, click on the **Cluster** tab. Click **Choose** and select **SimpleKMeans** from the choices that appear (this will be our preferred method of clustering for this article). Your WEKA Explorer window should look like Figure 6 at this point.

## Figure 6. BMW cluster algorithm



Finally, we want to adjust the attributes of our cluster algorithm by clicking **SimpleKMeans** (not the best UI design here, but go with it). The only attribute of the algorithm we are interested in adjusting here is the **numClusters** field, which tells us how many clusters we want to create. (Remember, you need to know this before you start.) Let's change the default value of 2 to 5 for now, but keep these steps in mind later if you want to adjust the number of clusters created. Your WEKA Explorer should look like Figure 7 at this point. Click **OK** to accept these values.

## Figure 7. Cluster attributes




At this point, we are ready to run the clustering algorithm. Remember that 100 rows of data with five data clusters would likely take a few hours of computation with a spreadsheet, but WEKA can spit out the answer in less than a second. Your output should look like Listing 5.

## Listing 5. Cluster output

```
                           Cluster#
Attribute            Full Data      0          1          2          3          4
                       (100)       (26)       (27)        (5)       (14)       (28)
=============================================================================
Dealership              0.6     0.9615     0.6667          1     0.8571          0
Showroom               0.72     0.6923     0.6667          0     0.5714          1
ComputerSearch         0.43     0.6538          0          1     0.8571     0.3214
M5                     0.53     0.4615      0.963          1     0.7143          0
3Series                0.55     0.3846     0.4444        0.8     0.0714          1
Z4                     0.45     0.5385          0        0.8     0.5714     0.6786
Financing              0.61     0.4615     0.6296        0.8          1        0.5
Purchase               0.39          0     0.5185        0.4          1     0.3214


Clustered Instances

0       26 ( 26%)
1       27 ( 27%)
2        5 (  5%)
3       14 ( 14%)
4       28 ( 28%)
```

How do we interpret these results? Well, the output is telling us how each cluster comes together, with a "1" meaning everyone in that cluster shares the same value of one, and a "0" meaning everyone in that cluster has a value of zero for that attribute. Numbers are the average value of everyone in the cluster. Each cluster shows us a type of behavior in our customers, from which we can begin to draw some conclusions:

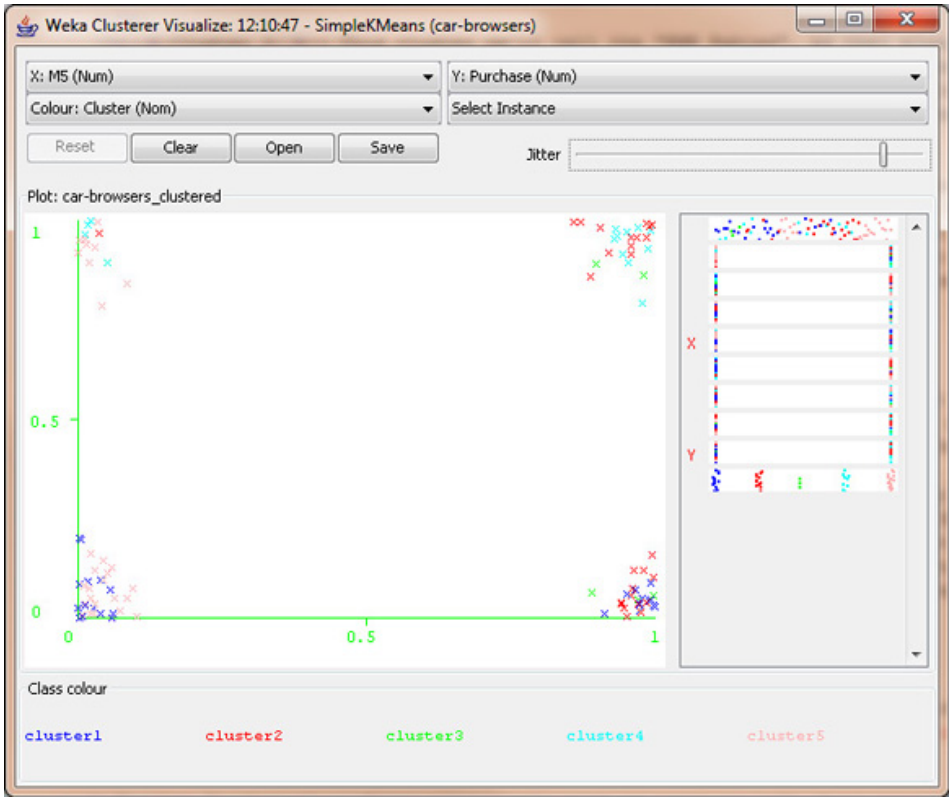
- **Cluster 0**— This group we can call the "Dreamers," as they appear to wander around the dealership, looking at cars parked outside on the lots, but trail off when it comes to coming into the dealership, and worst of all, they don't purchase anything.
- **Cluster 1**— We'll call this group the "M5 Lovers" because they tend to walk straight to the M5s, ignoring the 3-series cars and the Z4. However, they don't have a high purchase rate — only 52 percent. This is a potential problem and could be a focus for improvement for the dealership, perhaps by sending more salespeople to the M5 section.
- **Cluster 2**— This group is so small we can call them the "Throw-Aways" because they aren't statistically relevent, and we can't draw any good conclusions from their behavior. (This happens sometimes with clusters and may indicate that you should reduce the number of clusters you've created).
- **Cluster 3**— This group we'll call the "BMW Babies" because they always end up purchasing a car and always end up financing it. Here's where the data shows us some interesting things: It appears they walk around the lot looking at cars, then turn to the computer search available at the dealership. Ultimately, they tend to buy M5s or Z4s (but never 3-series). This cluster tells the dealership that it should consider making its search computers more prominent around the lots (outdoor search computers?), and perhaps making the M5 or Z4 much more prominent in the search results. Once the customer has made up his mind to purchase the vehicle, he always qualifies for financing and completes the purchase.
- **Cluster 4**— This group we'll call the "Starting Out With BMW" because they always look at the 3-series and never look at the much more expensive M5. They walk right into the showroom, choosing not to walk around the lot and tend to ignore the computer search terminals. While 50 percent get to the financing stage, only 32 percent ultimately finish the transaction. The dealership could draw the conclusion that these customers looking to buy their first BMWs know exactly what kind of car they want (the 3-series entry-level model) and are hoping to qualify for financing to be able to afford it. The dealership could possibly increase sales to this group by relaxing their financing standards or by reducing the 3-series prices.

One other interesting way to examine the data in these clusters is to inspect it visually. To do this, you should right-click on the **Result List** section of the **Cluster** tab (again, not the best-designed UI). One of the options from this pop-up menu is **Visualize Cluster Assignments**. A window will pop up that lets you play with the results and see them visually. For this example, change the X axis to be `M5 (Num)`, the Y axis to `Purchase (Num)`, and the Color to `Cluster (Nom)`. This will show us in a chart how the clusters are grouped in terms of who looked at the M5 and who purchased one. Also, turn up the "Jitter" to about three-fourths of the way maxed out, which will artificially scatter the plot points to allow us to see them more easily.

Do the visual results match the conclusions we drew from the results in Listing 5? Well, we can see in the X=1, Y=1 point (those who looked at M5s and made a purchase) that the only clusters

represented here are 1 and 3. We also see that the only clusters at point X=0, Y=0 are 4 and 0. Does that match our conclusions from above? Yes, it does. Clusters 1 and 3 were buying the M5s, while cluster 0 wasn't buying anything, and cluster 4 was only looking at the 3-series. Figure 8 shows the visual cluster layout for our example. Feel free to play around with the X and Y axes to try to identify other trends and patterns.

### Figure 8. Cluster visual inspection




**Further reading**: If you're interested in pursuing this further, you should read up on the following terms: Euclidean distance, Lloyd's algorithm, Manhattan Distance, Chebyshev Distance, sum of squared errors, cluster centroids.

## Conclusion

This article discussed two data mining algorithms: the classification tree and clustering. These algorithms differ from the regression model algorithm explained in Part 1 in that we aren't constrained to a numerical output from our model. These two models allow us more flexibility with our output and can be more powerful weapons in our data mining arsenal.

The classification tree literally creates a tree with branches, nodes, and leaves that lets us take an unknown data point and move down the tree, applying the attributes of the data point to the tree until a leaf is reached and the unknown output of the data point can be determined. We learned that in order to create a good classification tree model, we need to have an existing data set with known output from which we can build our model. We also saw that we need to divide our data set into two parts: a training set, which is used to create the model, and a test set, which is used

to verify that the model is accurate and not overfitted. As a final point in this section, I showed that sometimes, even when you create a data model you think will be correct, it isn't, and you have to scrap the entire model and algorithm looking for something better.

The clustering algorithm takes a data set and sorts them into groups, so you can make conclusions based on what trends you see within these groups. Clustering differs from classification and regression by not producing a single output variable, which leads to easy conclusions, but instead requires that you observe the output and attempt to draw your own conclusions. As we saw in the example, the model produced five clusters, but it was up to us to interpret the data within the clusters and draw conclusions from this information. In this respect, it can be difficult to get your clustering model correct (think what would happen if we created too many or too few clusters), but conversely, we were able to carve out some interesting information from the results — things we would have never been able to notice by using the other models we've discussed so far.

Part 3 will bring the "Data mining with WEKA" series to a close by finishing up our discussion of models with the nearest-neighbor model. We'll also take a look at WEKA by using it as a third-party Java™ library, instead of as a stand-alone application, allowing us to embed it directly into our server-side code. This will let us mine the data on our servers directly, without having to manipulate it into an ARFF file or run it by hand.

## Downloadable resources

| Description | Name | Size |
|---|---|---|
| Sample code | os-weka2-Examples.zip | 17KB |

## Related topics

- SIGKDD Explorations
- Weka 3: Data Mining Software in Java
- Downloading and installing Weka