Lecture#3

Text Classification

Document Classification

- A very common machine learning problem is to classify a document based on its text contents
- We use the text contents as input to a machine learning algorithm, and outputs which category the document (most likely) belongs to
- The (probably) earliest and most common application is spam detection
- We want to classify if an email is spam or not based on the text contents of the email
- Spam filtering can also be used for entries and comments on blogs and social media

Spam Filtering

- The early systems for spam detection were based on rules
- Programmers manually designed a set of rules that could detect spam
- Rules could for example detect:
  - Overuse of capital letters
  - Words related to pharmaceutical products or Rolex watches
  - Overuse of eye-catching colors
  - …
- The biggest problem with this approach was that spammers learned about the rules and created spam that could circumvent them

Spam Filtering

- Another problem is that what is spam or not depends on the user
- Advertising of drugs could be normal for a pharmacist, but unwanted for a programmer
- Therefore, we need a spam filter that can learn!
- Modern spam filters also take into consideration the sender of an email, but that is out of scope for us
- We are only interested in the text contents

Learning to classify text

- There are several algorithms that can be used for text classification:
  - Artificial Neural Networks
  - Support Vector Machines
  - K-nearest Neighbor algorithms
  - Decision Trees
  - Approaches based on natural language processing
  - …
- We will focus on a simple, powerful and very common algorithm: *Naïve Bayes*

Bayes' theorem

- First, we need to learn about Bayes' theorem
- It describes the probability of an event, based on prior knowledge of conditions that might be related to the event
- Bayes' theorem is stated using the following formula:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- … where P(A|B) shall be interpreted as ”probability that A occurs given B”
- It is best explained using an example:

Example

- We are interested in
  ”is a stiff neck a good sign of being a good FIFA player?”
- To answer this using Bayes’ theorem we need to know the prior probabilities:
  - 50% of the good FIFA players have a stiff neck:
    $P(stiff \mid good) = 0.5$
  - One in 50000 players is good at FIFA:
    $P(good) = 1/50000$
  - One in 20 players suffer from a stiff neck:
    $P(stiff) = 1/20$

Example

- We can now use the prior probabilities:
  - P(stiff | good) = 0.5   P(good) = 1/50000   P(stiff) = 1/20
- … to calculate the probability of being a good FIFA player if you have a stiff neck:

$$P(good|stiff) = \frac{P(stiff|good) \cdot P(good)}{P(stiff)}$$

$$P(good|stiff) = \frac{0.5 \cdot 1/50000}{1/20} = 0.0002$$

Example

- Given the prior probabilities:
  - P(stiff | good) = 0.5   P(good) = 1/50000   P(stiff) = 1/20
- … we can use Bayes' theorem to say that it is a probability of 0.0002 that a player is good at FIFA if he has a stiff neck
- Or that one in 5000 players with stiff necks is good at FIFA

Naïve Bayes

- Bayes' theorem only takes on attribute into consideration (stiff neck) when calculating the probability of belonging to a specific category (good FIFA player)
- In most real-world applications we have more than one attribute:
  - stiff neck
  - good gamepad
  - large TV
  - …
- We need a way of combining several inputs to get a probability of belonging to a specific category
- This is handled by the Naïve Bayes classifier

Naïve Bayes

- The classifier is called *naïve* because it assumes that the attributes are *independent* of each other
- It means that the probability of one attribute belonging to a specific category is completely unrelated to the probability of other attributes belonging to that category
- There are no relation between attributes
- This is actually a false assumption
- Example: ”money” is a better spam indicator if in combination with ”casino” than with ”programming”

Naïve Bayes

- The independence between attributes means that the actual probability calculated by the Naïve Bayes classifier is inaccurate
- You cannot say that the resulting probability is the *actual* probability that a document belongs to a category
- We can however *compare* the results of the document belong to different categories, and see which has the highest probability
- This works surprisingly well for real-world document classification problems

Naïve Bayes classification

1. Calculate the frequencies of each attribute belonging to each category
2. For each category:
    1. Multiply the conditional probability of each attribute into a product
    2. Multiply the product with the category probability
3. Classify the document as belonging to the category with the highest probability

Let’s look at an example!

Example dataset

Game pad?	Stiff neck?	Player skill
Great	Yes	Good
Average	Yes	Good
Junk	Yes	Good
Average	No	Good
Junk	No	Bad
Average	No	Bad
Great	Yes	Bad
Average	No	Bad
Average	No	Bad

Frequency table

- First step is to generate a frequency table:

Game Pad?			Stiff neck?			Player skill?	
	Good	Bad		Good	Bad	Good	Bad
Great	1	1	Yes	3	1	4	5
Average	2	3	No	1	4		
Junk	1	1					

Game pad?	Stiff neck?	Player skill
Great	Yes	Good
Average	Yes	Good
Junk	Yes	Good
Average	No	Good
Junk	No	Bad
Average	No	Bad
Great	Yes	Bad
Average	No	Bad
Average	No	Bad

Prior probabilities

- We continue filling the table with prior probabilities:

Game Pad?			Stiff neck?			Player skill?	
	Good	Bad		Good	Bad	Good	Bad
Great	1	1	Yes	3	1	4	5
Average	2	3	No	1	4		
Junk	1	1					
P(Great | x)	1/4	1/5	P(Yes | x)	3/4	1/5	4/9	5/9
P(Avg | x)	2/4	3/5	P(No | x)	1/4	4/5		
P(Junk | x)	1/4	1/5					

Classification

- The table is all we need for classification
- Now we can answer questions like:
  - ”A player has an average game pad and a stiff neck. Is he a good or bad player?”
  - ”A player has a great game pad and not a stiff neck. Is he a good or bad player?”
- We have two possible categories, the player being Good or Bad
- Let’s calculate the probabilities of the above mentioned player belonging to the two categories:

Classification

- Classify the player:
  - {game pad = average, stiff neck = yes}
- Probability that the player is Good:

  P(Good) * P(average | Good) * P(yes | Good) = 4/9 * 2/4 * 3/4 = **0.1667**

- Probability that the player is Bad:

  P(Bad) * P(average | Bad) * P(yes | Bad) = 5/9 * 3/5 * 1/5 = **0.0667**

- It is a higher probability for good than bad, so we classify the player as <u>good</u>!

Classification

- Classify the player:
  - {game pad = great, stiff neck = no}
- Probability that the player is Good:

  P(Good) * P(great | Good) * P(no | Good) = 4/9 * 1/4 * 1/4 = **0.0278**

- Probability that the player is Bad:

  P(Bad) * P(great | Bad) * P(no | Bad) = 5/9 * 1/5 * 4/5 = **0.0889**

- It is a higher probability for bad than good, so we classify the player as <u>bad</u>!

Threshold

- In many applications it is better to return a ”don’t know” than a misclassified document
- In spam filtering, it is more important to avoid having legitimate email classified as spam than to catch every single spam message
- This can be solved by using a *threshold*
- If we use a threshold of 3 means that the probability for the highest category must be at least 3 times higher than the probability of the other category, otherwise the classifier is unsure
- In our examples we used a threshold of 1, meaning that we always classify as the highest category regardless of the difference in probabilities

The examples using threshold

{game pad = average, stiff neck = yes}				
P(Good)	P(Bad)	Ratio	Threshold	Classified as
0.1667	0.0667	2.499	1	Good
0.1667	0.0667	2.499	3	Don't know

{game pad = great, stiff neck = no}				
P(Good)	P(Bad)	Ratio	Threshold	Classified as
0.0278	0.0889	3.198	1	Bad
0.0278	0.0889	3.198	3	Bad

Variants of Naïve Bayes

- The approach described here is called *Multinomial Naïve Bayes*
- There are a number of other variants of Naïve Bayes, mainly *Gaussian* and *Bernoulli*
- In *Bernoulli*, we don't count the actual frequency of an attribute in a category
- Instead we use 1 if the attribute appears in any document belonging to the category, and 0 otherwise
- In *Gaussian*, we assume that attributes are numeric and follow a normal distribution

Text classification

- In the examples we have seen so far we have had two attributes:
  - Game pad: {great, average, junk}
  - Stiff neck: {yes, no}
- In document classification, we have to classify texts of different length
- To do this we first have to convert the text contents of each document to a *bag-of-words*
- Then we have to count the frequency and calculate the probability that each word belongs to each category
- Let's look at an example:

Example dataset

Text	Spam?
Buy cheap Rolex?	Yes
You want Viagra?	Yes
Can you buy milk?	No
I want candy tonight	No
Want to go to the gym?	No

Example dataset

Text	Spam?
Buy cheap Rolex?	Yes
You want cheap Viagra?	Yes
Can you buy milk?	No
Want candy tonight?	No
Gym tonight?	No

- The unique words are (special characters removed):
    - buy, cheap, rolex, you, want, viagra, can, milk, candy, tonight, gym
- Next step is to create a frequency matrix

Example dataset

						Spam?	
	Yes	No		Yes	No	Yes	No
buy	1	1	can	0	1	2	3
cheap	2	0	milk	0	1		
rolex	1	0	candy	0	1		
you	1	1	tonight	0	1		
want	1	1	gym	0	1		
viagra	1	0					

Example dataset

						Spam?	
	Yes	No		Yes	No	Yes	No
buy	1	1	can	0	1	2	3
cheap	2	0	milk	0	1		
rolex	1	0	candy	0	1		
you	1	1	tonight	0	1		
want	1	1	gym	0	1		
viagra	1	0				2/5	3/5
P(buy | x)	1/2	1/3	P(can | x)	0/2	1/3		
P(cheap | x)	2/2	0/3	P(milk | x)	0/2	1/3		
P(rolex | x)	1/2	0/3	P(candy | x)	0/2	1/3		
P(you | x)	1/2	1/3	P(tonight | x)	0/2	1/3		
P(want | x)	1/2	1/3	P(gym | x)	0/2	1/3		
P(viagra | x)	1/2	0/3					

Classification

- Now you want to classify the text
  ”buy cheap candy”
- As in the previous examples, we calculate the probability for being spam or not being spam:

  P(yes) * P(buy | yes) * P(cheap | yes) * P(candy | yes) = 2/5 * 1/2 * 2/2 * 0/2 = 0

- Here we can see a problem: if a word has never showed up in a category, we multiply with a 0 and the result will always be 0…
- To solve this we can apply Laplace correction:

Laplace correction

- In Laplace correction we always add some constant value to each probability to avoid 0 probabilities
- If we use 1/3 as Laplace correction the probability for being spam looks like:

  P(yes) * P(buy | yes) * P(cheap | yes) * P(candy | yes) =
  = 2/5 * (1/2+1/3) * (2/2+1/3) * (0/2+1/3) =
  = 0.4 * 0.833 * 1.333 * 0.333 = **2.9**

- And for not being spam:

  P(no) * P(buy | no) * P(cheap | no) * P(candy | no) =
  = 3/5 * (1/3+1/3) * (0/3+1/3) * (1/3+1/3) =
  = 0.6 * 0.667 * 0.333 * 0.667 = **0.089**

- This message is classified as spam!

Spam or not?

- The text ”buy cheap candy” was clearly classified as spam
- Is this correct?
- The word that is most prominent in the result is ”cheap”, which exists in 2 of 2 spam and 0 of 3 legitimate messages
- If ”cheap” is not a good indicator for spam, we need more training data where cheap appears in legitimate message
- We need quite large amounts of data for text classification to be accurate

Weka

Weka

- Weka is a collection of machine learning algorithms
- It consists of a GUI tool and an API library
- It has been around for several years and has been used in numerous projects
- Weka can be downloaded at https://www.cs.waikato.ac.nz/ml/weka/
- We will now take a look at how we can use Weka to classify the Wikipedia dataset

Wikipedia dataset

- We will use a subset of the Wikipedia dataset consisting of 35 articles about programming, and 35 about video games.
- All tags and code has been removed and the text has been converted to a *bag-of-words*
- The data is stored in an *arff* file
- Arff (attribute-relation file format) is the format Weka uses for data files

Arff file

- The first part in the arff files defines a name for the classification task, and each attribute with their respective type:

```
@relation Wikipedia

@attribute text string
@attribute articletype {games,programming}
```

- This is followed by the actual data:

```
@data
'perl from wikipedia free ...',programming
'console game from wikipedia free ...',games
'declarative programming from wikipedia ...',programming
…
```

Attribute types

- Attributes can be:
  - Strings:
    @attribute text string
  - Nominal (fixed set of values):
    @attribute articletype {games,programming}
  - Numeric:
    @attribute temperature numeric

Data section

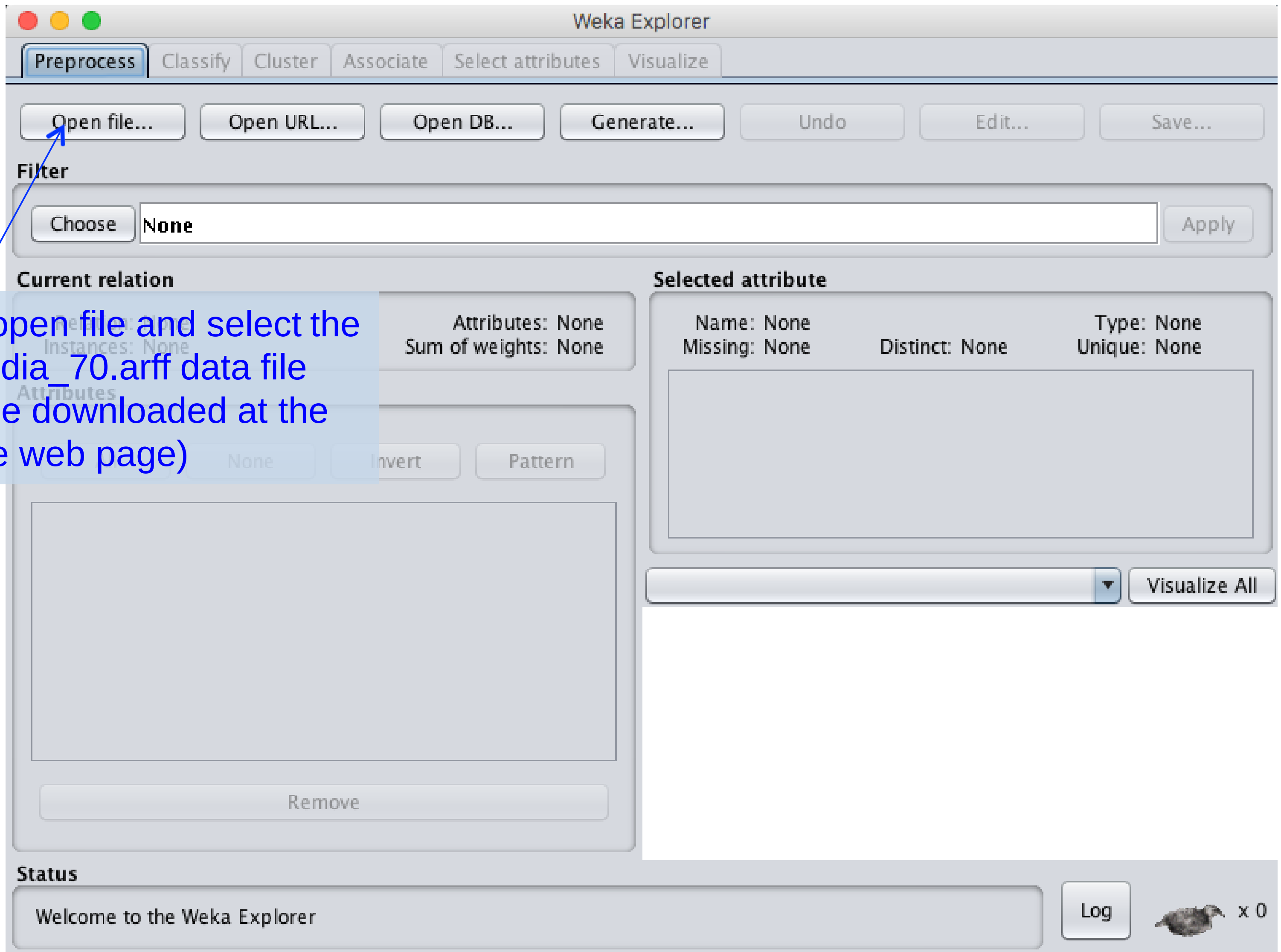
- Each attribute is separated with a comma
- In our case we only have one attribute (the text) and a category
- Text must be between '...'
- Weka is very strict with which ' character you use, so make sure you use the correct one
- Each row in the file is called an *instance* (or *example*)

Using Weka

- Double-clicking weka.jar should open the Weka application
- You should now see the following window:

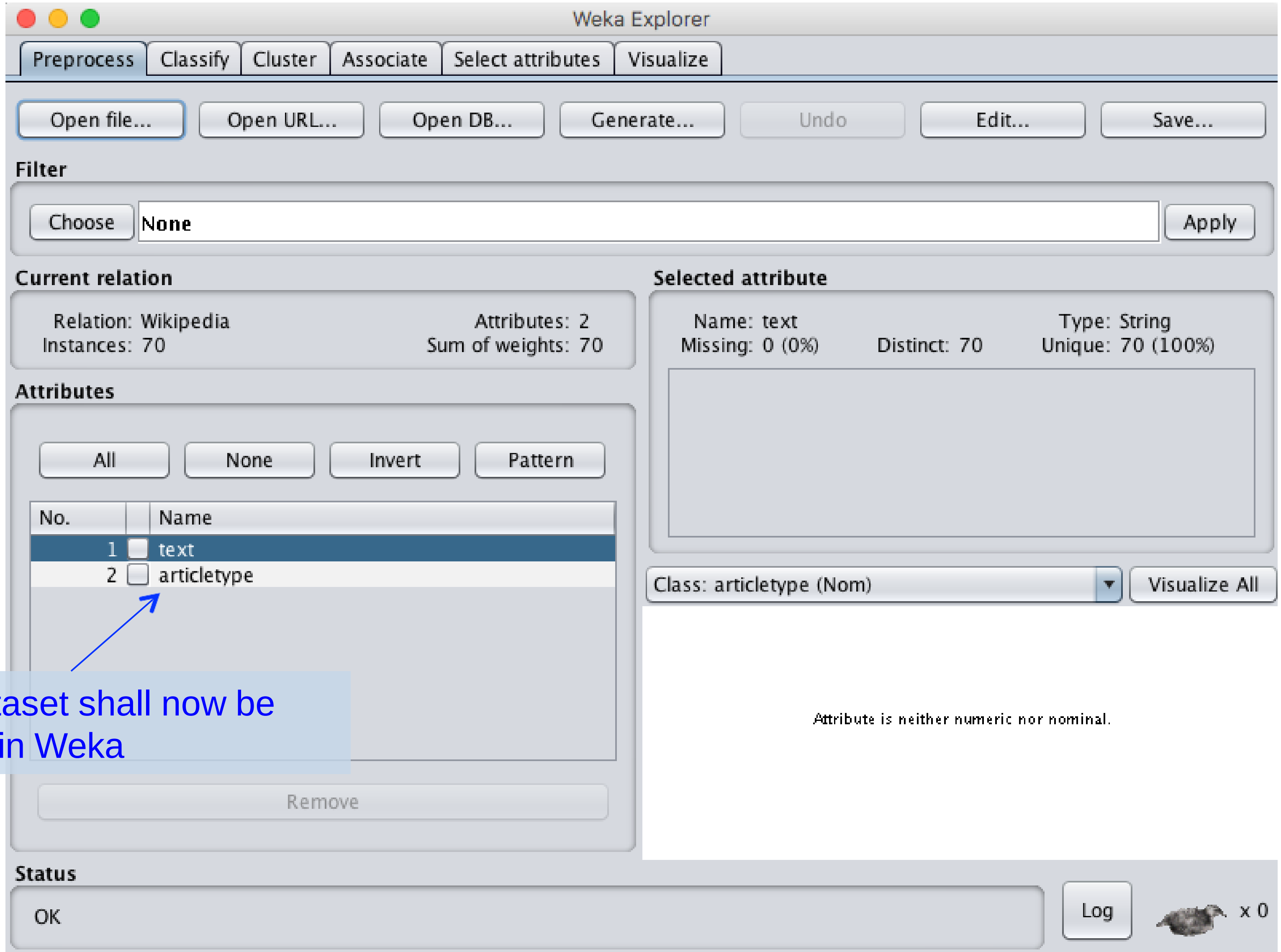
Open the Explorer application

Using Weka



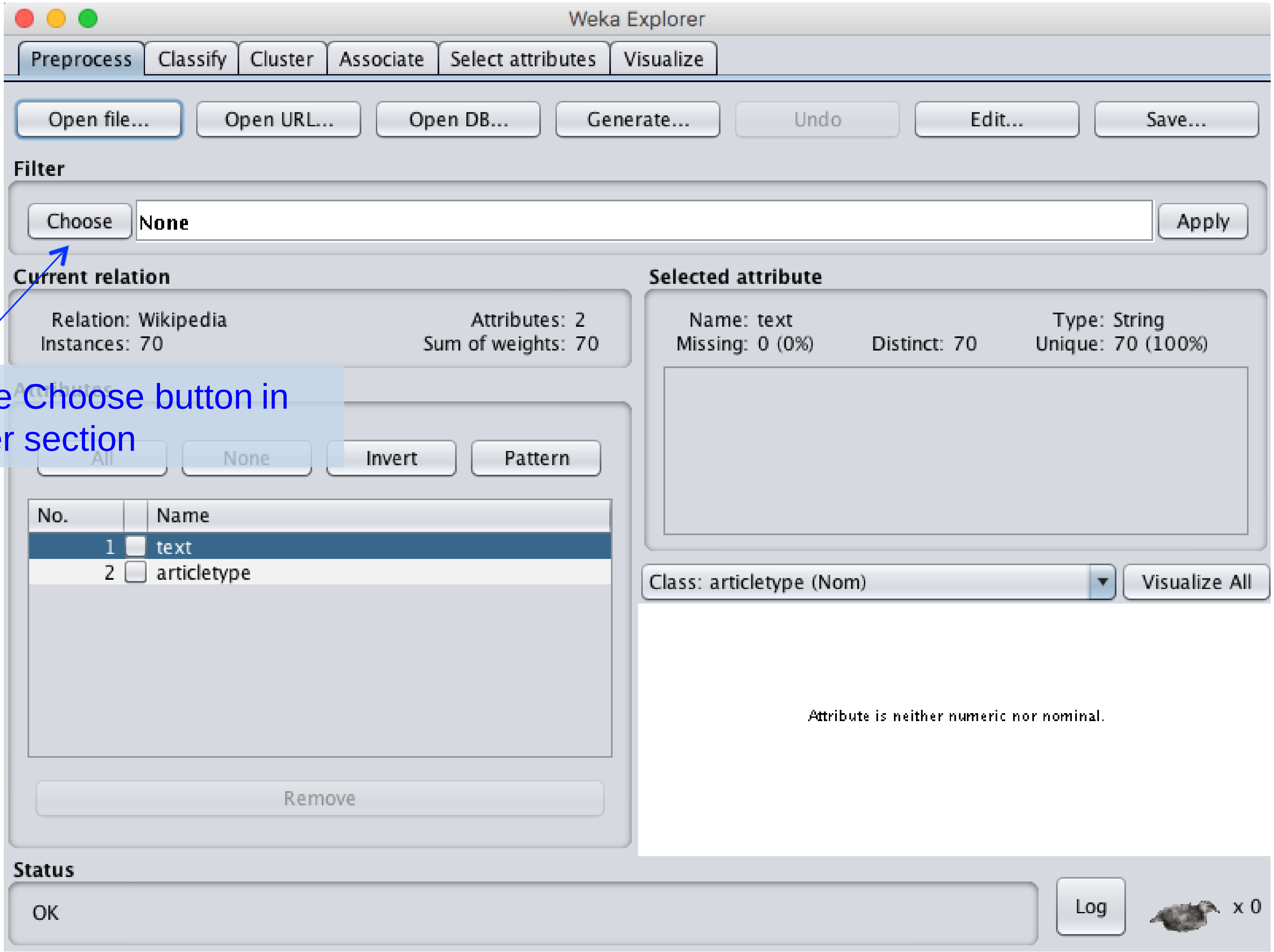
Click open file and select the wikipedia_70.arff data file (can be downloaded at the course web page)

Using Weka



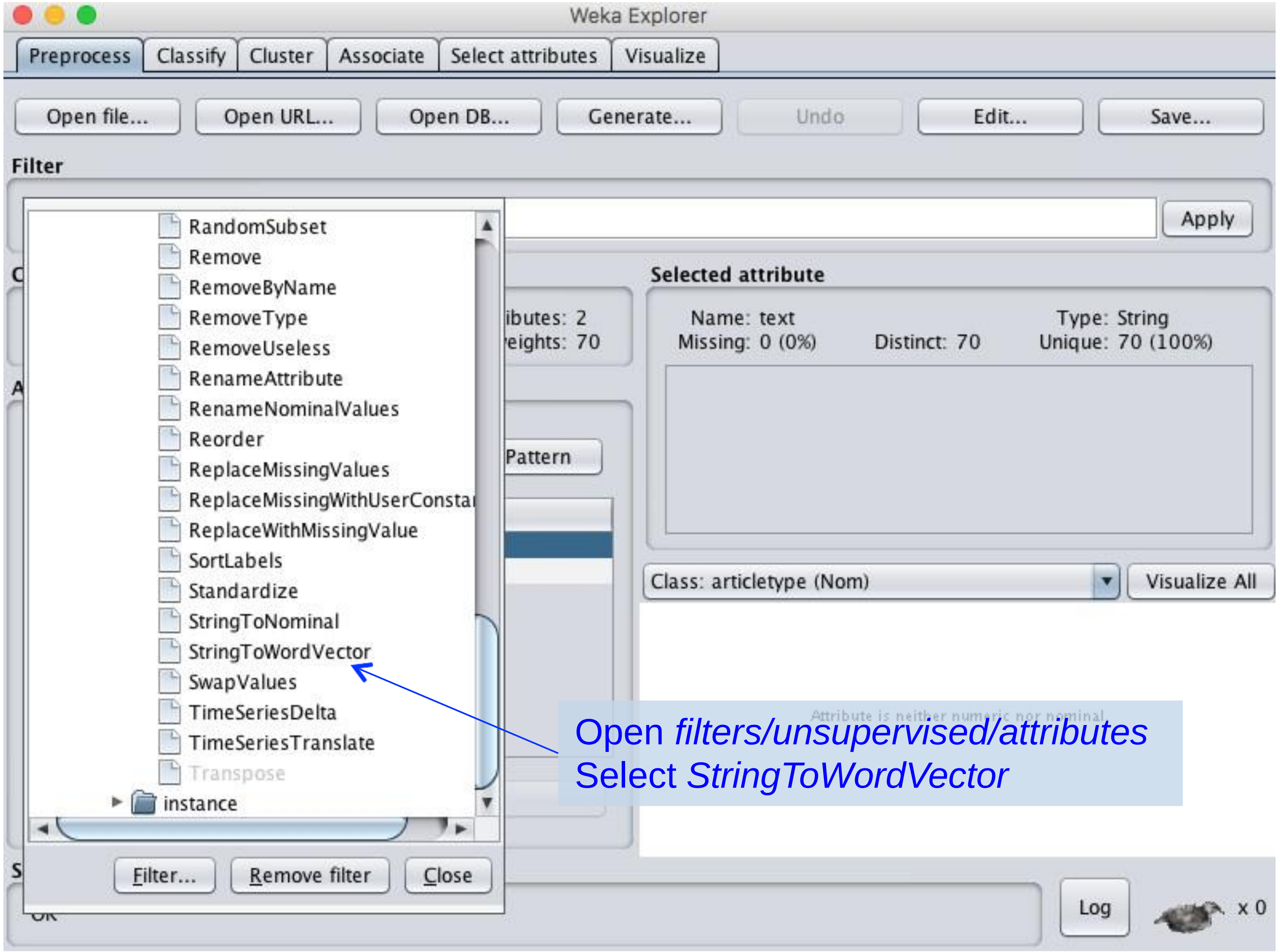
The dataset shall now be loaded in Weka

Using Weka

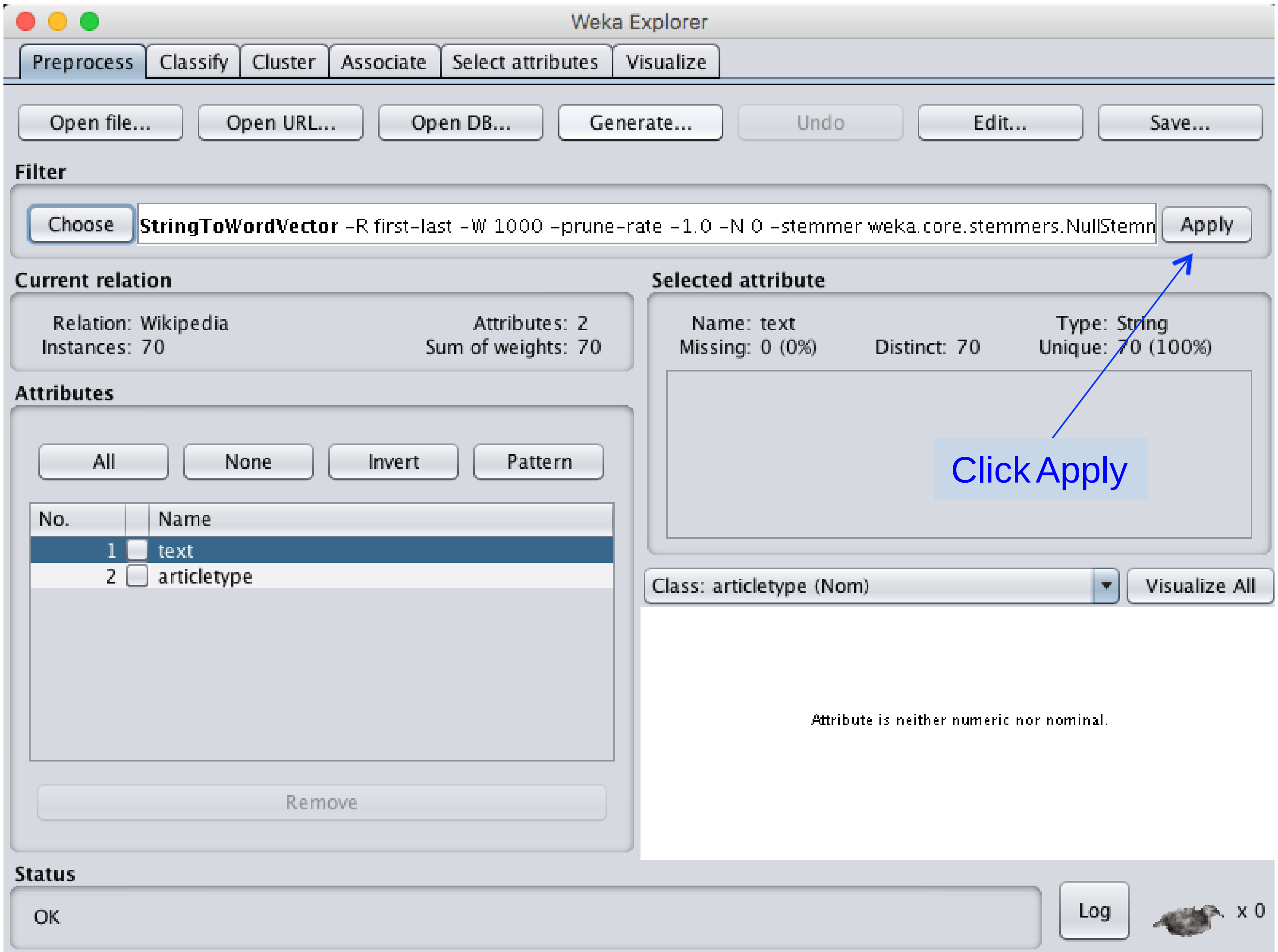


Click the Choose button in the Filter section

Using Weka

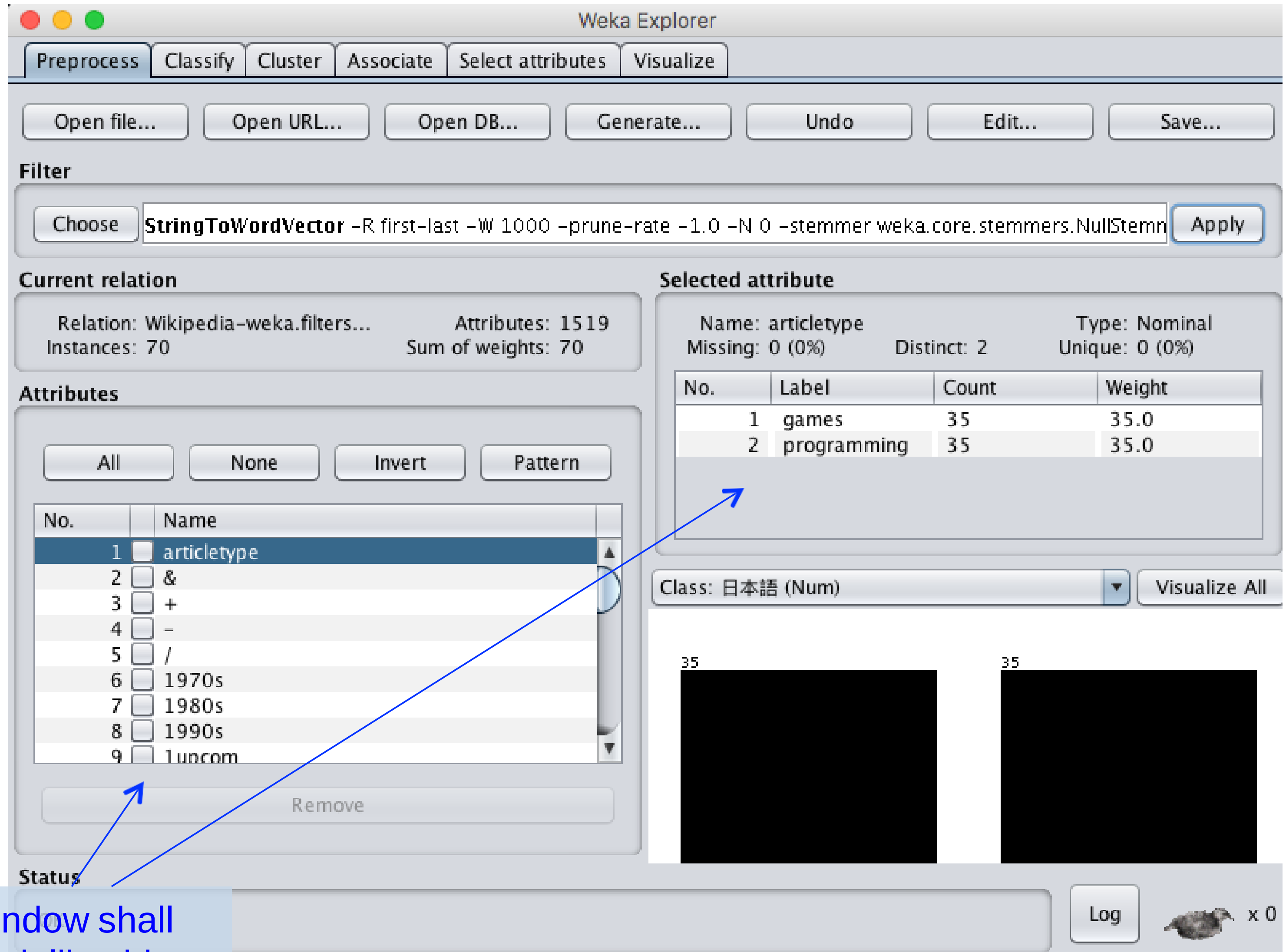

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Apply
RandomSubset
Remove
RemoveByName
RemoveType
RemoveUseless
RenameAttribute
RenameNominalValues
Reorder
ReplaceMissingValues
ReplaceMissingWithUserConsta
ReplaceWithMissingValue
SortLabels
Standardize
StringToNominal
StringToWordVector
SwapValues
TimeSeriesDelta
TimeSeriesTranslate
Transpose
instance
Filter...
Remove filter
Close
Selected attribute
Name: text
Type: String
Missing: 0 (0%)
Distinct: 70
Unique: 70 (100%)
ibutes: 2
eights: 70
Pattern
Class: articletype (Nom)
Visualize All
Open filters/unsupervised/attributes
Select StringToWordVector
Log
x 0

Using Weka



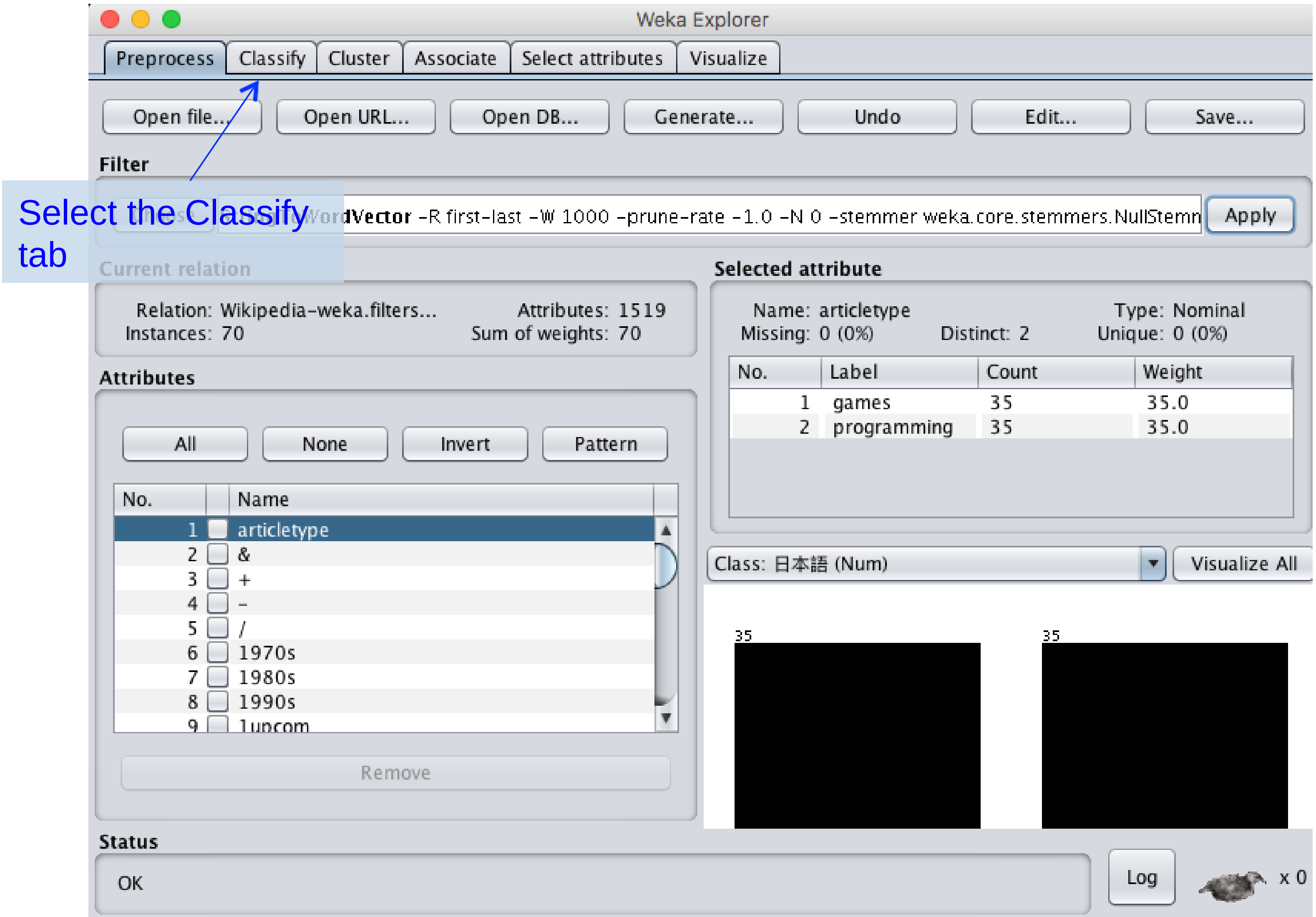
Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemn
Apply
Current relation
Relation: Wikipedia
Instances: 70
Attributes: 2
Sum of weights: 70
Selected attribute
Name: text
Type: String
Missing: 0 (0%)
Distinct: 70
Unique: 70 (100%)
Click Apply
Attributes
All
None
Invert
Pattern
No.
Name
1 text
2 articletype
Remove
Class: articletype (Nom)
Visualize All
Attribute is neither numeric nor nominal.
Status
OK
Log
x 0

Using Weka

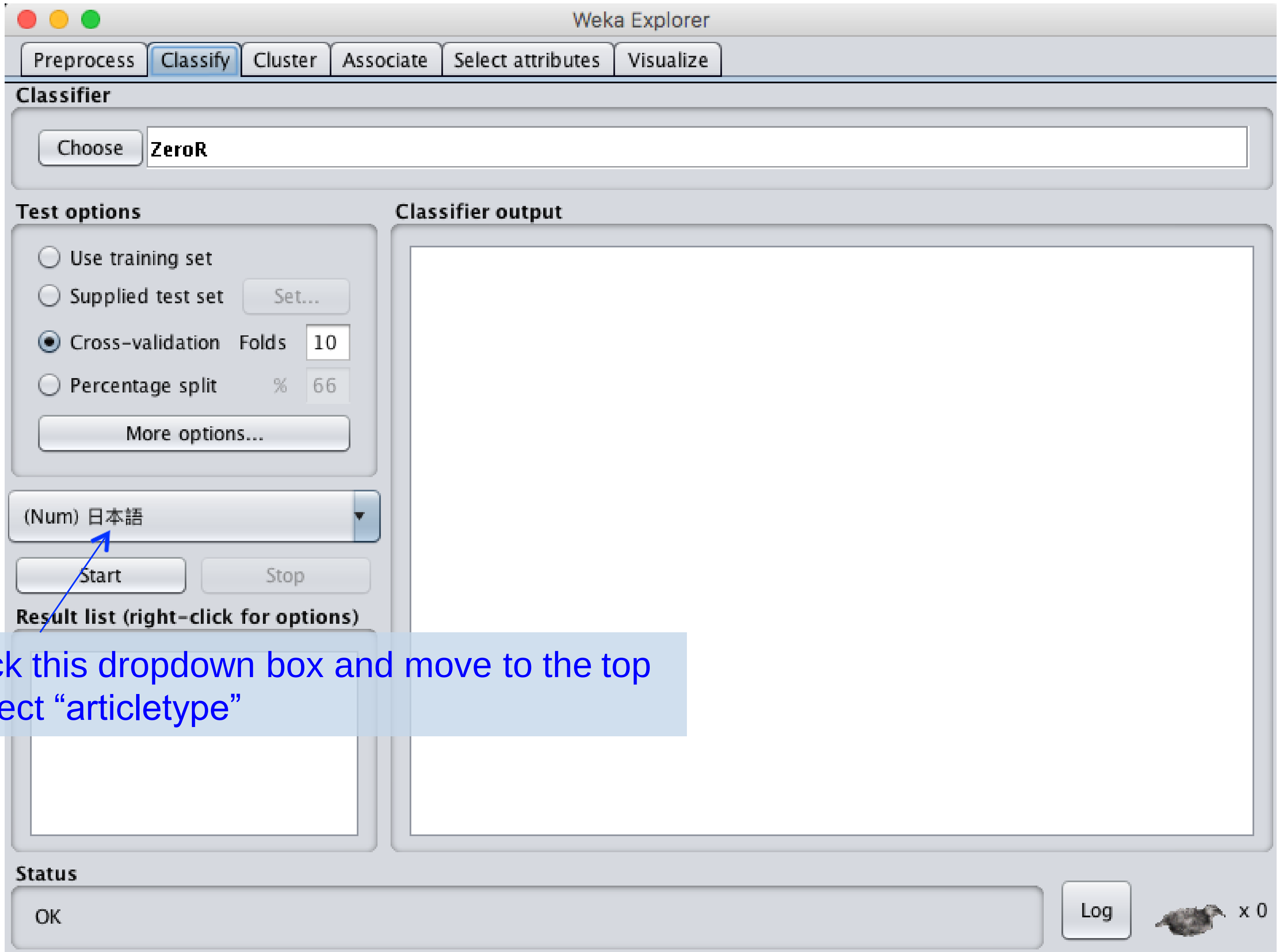


The window shall now look like this

Using Weka

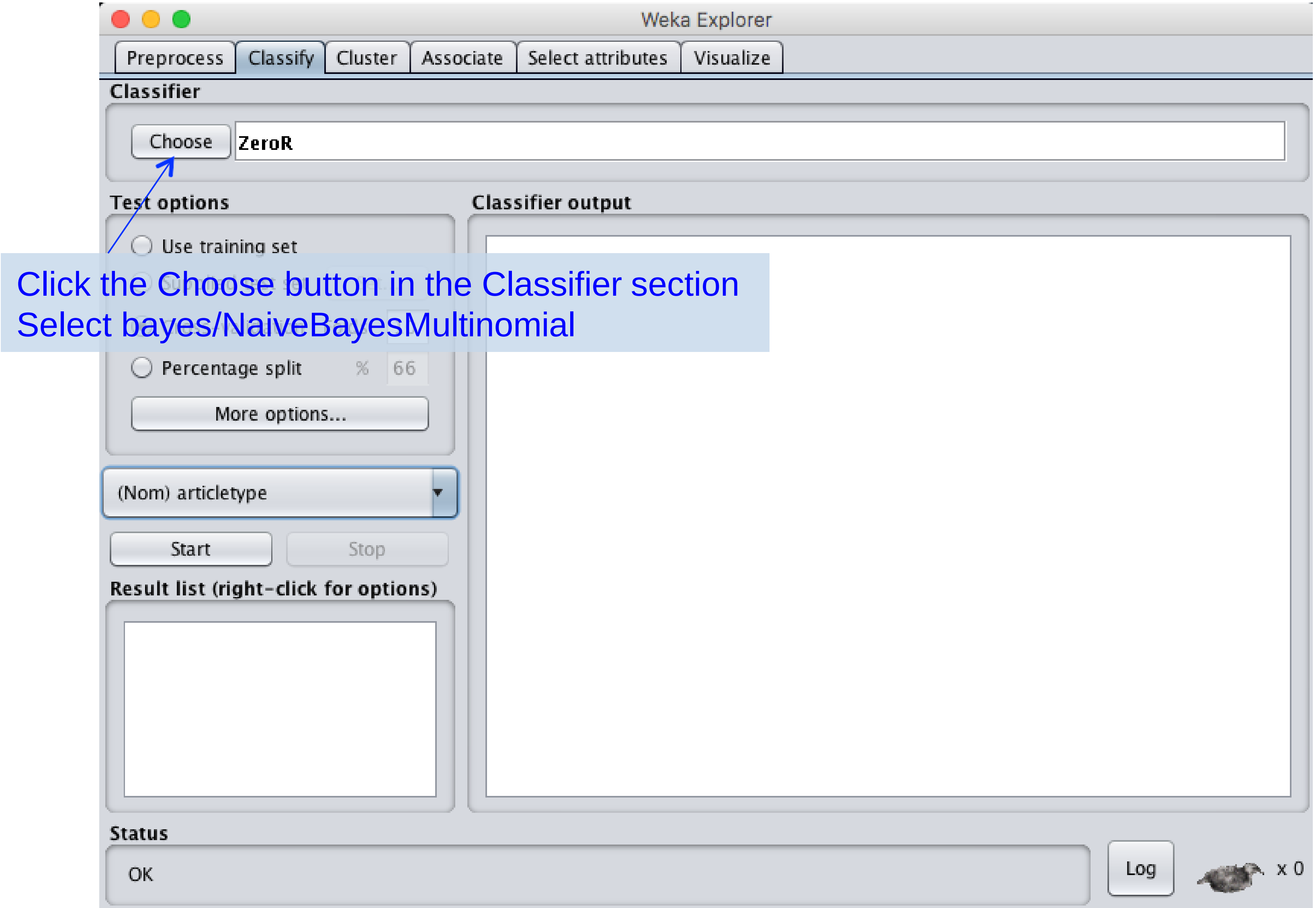

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Select the Classify tab
dVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemn
Apply
Current relation
Relation: Wikipedia-weka.filters...
Attributes: 1519
Instances: 70
Sum of weights: 70
Selected attribute
Name: articletype
Type: Nominal
Missing: 0 (0%)
Distinct: 2
Unique: 0 (0%)
No. Label Count Weight
1 games 35 35.0
2 programming 35 35.0
Attributes
All
None
Invert
Pattern
No. Name
1 articletype
2 &
3 +
4 -
5 /
6 1970s
7 1980s
8 1990s
9 1upcom
Remove
Class: 日本語 (Num)
Visualize All
35
35
Status
OK
Log
x 0

Using Weka



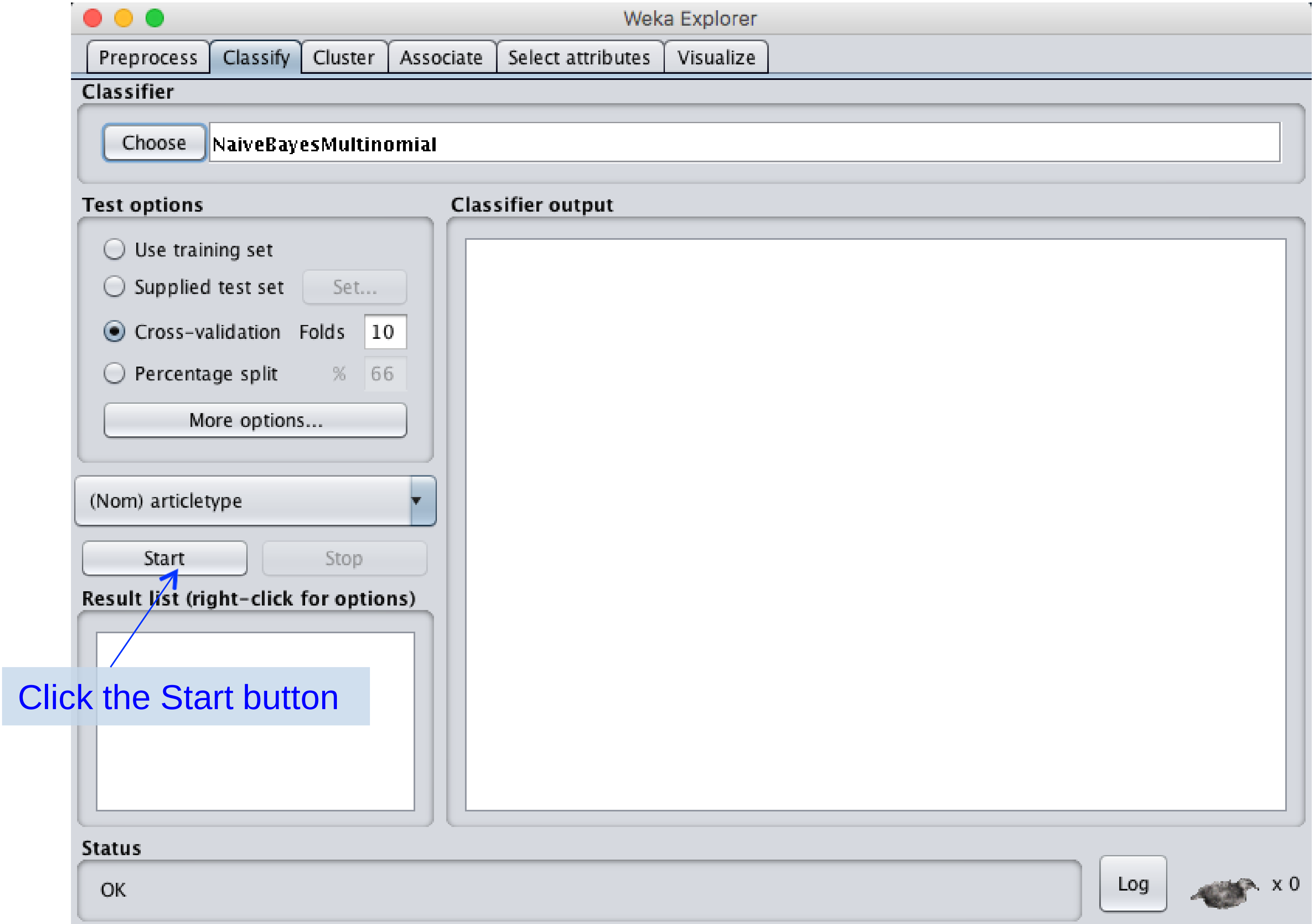
Click this dropdown box and move to the top
Select “articletype”

Using Weka

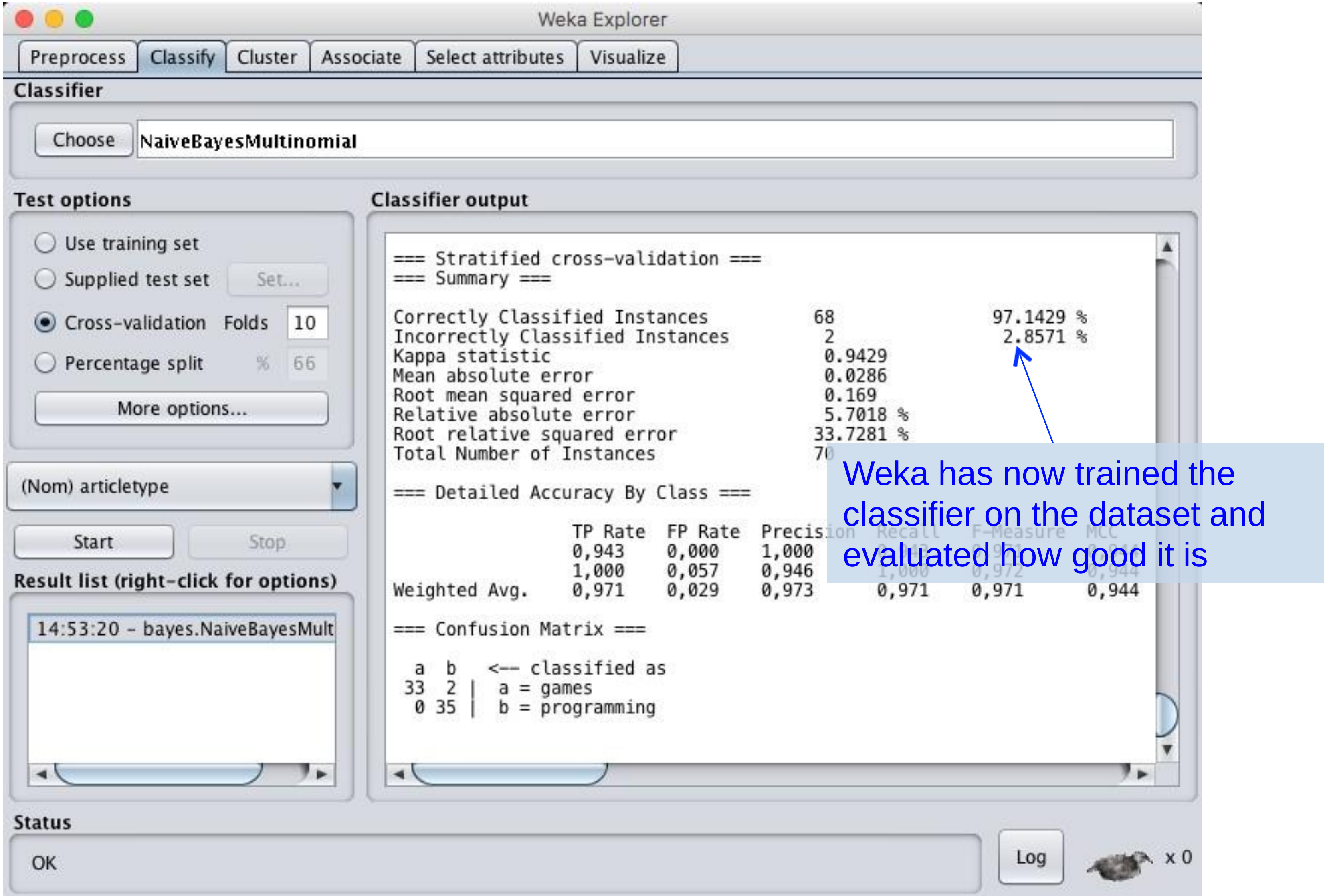


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
ZeroR
Test options
Use training set
Percentage split
%
66
More options...
(Nom) articletype
Start
Stop
Result list (right-click for options)
Classifier output
Status
OK
Log
x 0
Click the Choose button in the Classifier section
Select bayes/NaiveBayesMultinomial

Using Weka


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayesMultinomial
Test options
Classifier output
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) articletype
Start
Stop
Result list (right-click for options)
Click the Start button
Status
OK
Log
x 0

Using Weka



Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayesMultinomial
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) articletype
Start
Stop
Result list (right-click for options)
14:53:20 - bayes.NaiveBayesMult
Classifier output
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 68 97.1429 %
Incorrectly Classified Instances 2 2.8571 %
Kappa statistic 0.9429
Mean absolute error 0.0286
Root mean squared error 0.169
Relative absolute error 5.7018 %
Root relative squared error 33.7281 %
Total Number of Instances 70
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC
0,943 0,000 1,000
1,000 0,057 0,946
Weighted Avg. 0,971 0,029 0,973 0,971 0,971 0,944
=== Confusion Matrix ===
a b <-- classified as
33 2 | a = games
0 35 | b = programming
Weka has now trained the classifier on the dataset and evaluated how good it is
Status
OK
Log
x 0

Interpreting the results

- Weka produced the following result:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          68               97.1429 %
Incorrectly Classified Instances         2                2.8571 %
Kappa statistic                          0.9429
Mean absolute error                      0.0286
Root mean squared error                  0.169
Relative absolute error                  5.7018 %
Root relative squared error             33.7281 %
Total Number of Instances               70

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
                 0,943    0,000    1,000      0,943    0,971      0,944
                 1,000    0,057    0,946      1,000    0,972      0,944
Weighted Avg.    0,971    0,029    0,973      0,971    0,971      0,944

=== Confusion Matrix ===

  a  b   <-- classified as
 33  2 |  a = games
  0 35 |  b = programming
```

- What does this tell us?

Interpreting the results

- The line ”Correctly Classified Instances” tells us the accuracy:
  - 68 of 70 articles were correctly classified
- The Confusion Matrix is often very interesting:

Interpreting the results

- We have two categories *games* and *programming*
- We can see that 33 articles in category *a* (games) were correctly classified
- 2 articles about games were however incorrectly classified
- All articles about programming were correctly classified

```
=== Confusion Matrix ===

  a  b   <-- classified as
 33  2 |  a = games
  0 35 |  b = programming
```

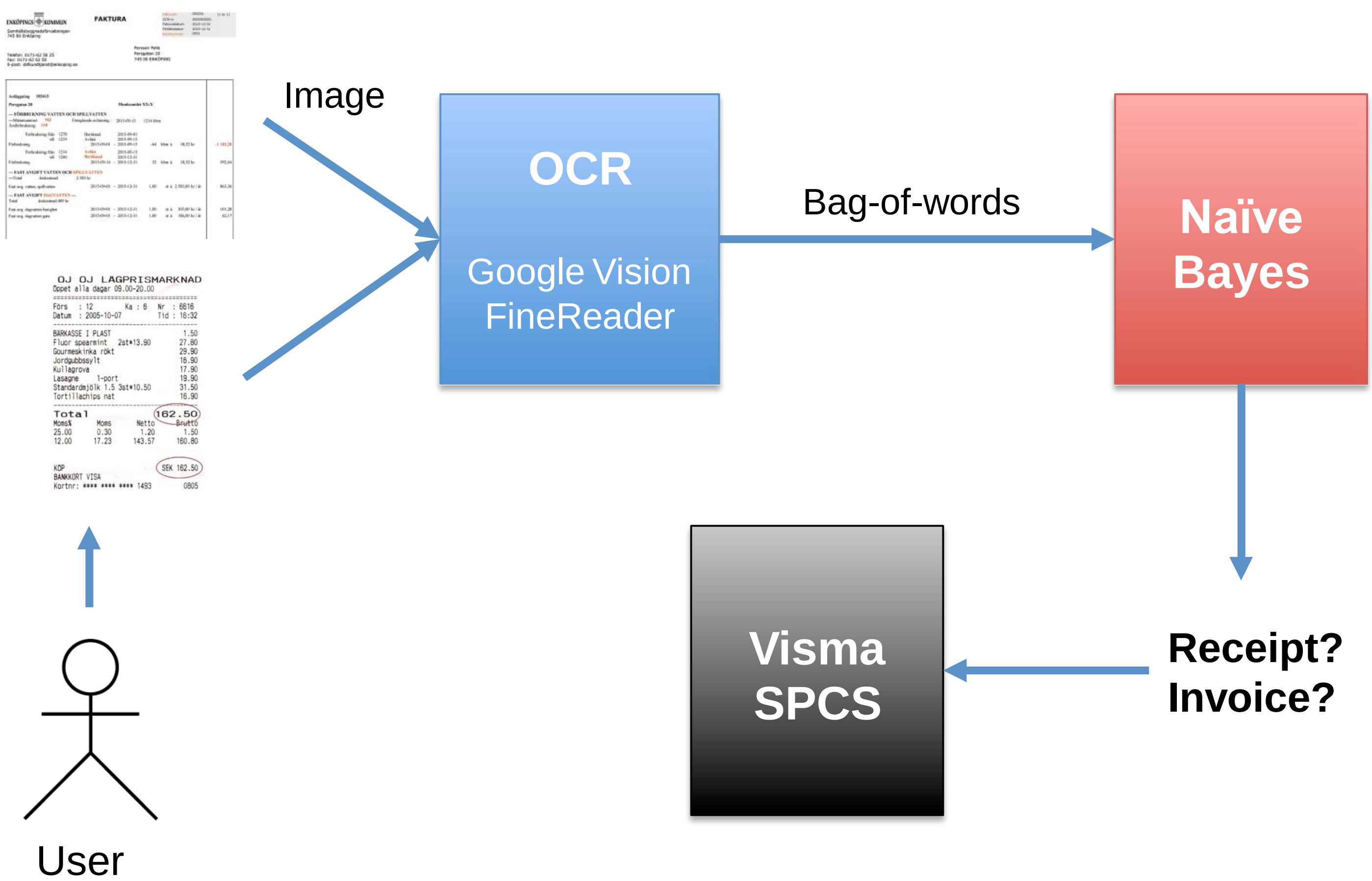
A more realistic case:
Receipt or Invoice?

Background

- A user in Visma’s accounting system uploads a picture of a receipt or invoice
- The user must then select if it is a receipt or invoice in a dropdown menu
- It is quite easy to misclick or forgot this step

- Student project at Visma:
  - Automatize this with machine learning

System Overview



Result

- Dataset of 37 receipts and 38 invoices (75 images)
- 97.3% correctly classified images
- Enough for Visma to use it in production

- So far no continuation on the project