Practical Data Mining

COMP-321B

Tutorial 5: Article Identification

Shevaun Ryan
Mark Hall

August 15, 2006

1 Introduction

This tutorial will focus on text mining, using text documents instead of attribute tables.

Before you start, please note that the data sets are large and take some time to process. You may also find that you run out of memory on the Java heap; if this happens WEKA closes with an error message. To avoid this, run the following command line to start WEKA:

java -Xmx1024m weka.gui.explorer.Explorer

This increases the Java heap to 1024m. You can also avoid this problem by deleting old classification models that you no longer need. Just select the name of the test in the 'results list' and press the delete key.

In the tasks we will be using a couple of data sets made up of hundreds of newspaper articles. Each article has been classified with a 1 if the article is related to the title of the document, and 0 if the article is unrelated. The goal of this tutorial is to see if WEKA can correctly classify articles based solely on the words in the article. In order to do this we must turn each word into an attribute; either a binary attribute indicating whether the word is found in a document, or a whole number indicating the number of times the word appears within a document (the word count).

Another difference with this tutorial is the section of the output that we are interested in. We will be recording and comparing the True Positive and False Positive Rates, as well as the ROC area. This data is more useful than the accuracy percentage, as we will see later in the tutorial.

The classifiers we will be using in this tutorial are:

Naive Bayes (weka.classifiers.bayes.NaiveBayes), Naive Bayes Multinomial (weka.classifier.bayes.NaiveBayesMultinomial) and Support Vector Classifier (weka.classifiers.functions.SMO).

We will also be using the Attribute Selected Classifier (weka.classifers.meta.AttributeSelectedClassifier) with InfoGainAttributeEval (weka.attributeSelection.InfoGainAttributeEval) as the evaluator and Ranker (weka.attributeSelection.Ranker) as the search method.

Use Cross-Validation with 10 folds for each classification.

NB: The class attribute for these data sets is the *first* attribute in the list, instead of the last one (which is the WEKA default). Before running any classifier, make sure to select the correct class attribute from the list just above the **Start** button.

1.1 Introduction to the data sets used

Both of these data sets contain a collection of newspaper articles on various topics. The files have been processed into ARFF format, so they are made up of a number of instances - each instance consists of a string attribute (the article

itself) and a binary class attribute indicating whether the article is related to the title of the data set, or not.

The data sets can be found in /home/ml/321/Tutorial5/.

ReutersCorn-train.arff This data set contains 1554 articles, 45 of which are about corn.

ReutersGrain-train.arff This data set contains 1554 articles, 103 of which are about grain.

2 Using Binary Attributes

In order to use the words of each article as attributes, we will be using a filter called StringToWordVector
(weka.filters.unsupervised.attribute.StringToWordVector).

2.1 Exercise A: Classify articles based on the words that appear in the articles

Load the **ReutersCorn-train.arff** data set.

Task A1: Select the 'StringToWordVector' filter and change the following settings: lowerCaseTokens = True, onlyAlphabeticTokens = True, wordsToKeep = 2500. Apply the filter and browse the attribute list. What did this filter do? What percentage of instances are positive (have a class label of 1)?

Task A2: Go to the **Classify** panel. Classify the data using Naive Bayes and then SMO. Record the TP and FP rate for the positive instances and the ROC area.

Task A3: Compare the results from the above task. Which model was better at classifying the articles and why (include your knowledge of TP/FP rates and ROC area in your answer)?

The relevance of the TP/FP rates can change, depending on the goal of the classification and the data being used. For example, if you were classifying email spam from real email one of the most important goals would be to keep the FP rate as close to zero as possible, because otherwise the model is throwing away real emails. Having a high TP rate is good also, but in this situation it's better to have a few spam emails make it through the filter as long as the recipient is not losing important emails.

Which do you think is more important for classifying newspaper articles, the TP rate or the FP rate, and why?

Task A4: Reclassify the data using the Attribute Selected Classifier with the same classifiers used in Task A2, using InfoGain as the attribute evaluator and Ranker as the search method. In the Ranker settings, change the

numToSelect to 100. This means that the algorithm will rank all the attributes but only retain the top 100 and classify the data based on those attributes. Record the TP and FP rate and ROC area of each model.

Task A5: Scroll up in the output window to view the Ranker results (from Task A4). Browse the words (attributes) that have been retained. How do you think the evaluator performed? Are the words relevant to articles about corn? Inspect the top 20 words and write down any that you think are unimportant (or not specific to corn articles). Explain why you think they made it on the list.

Task A6: Compare the results from Tasks A2 and A4 and comment on your findings (did ranking the attributes give a better or worse result? Why?).

3 Word Count Attributes

3.1 Exercise B: Classify articles using the frequency of words appearing in the articles

Load the **ReutersGrain-train.arff** data set.

Task B1: Select the StringToWordVector filter and change the following settings: lowerCaseTokens = True, onlyAlphabeticTokens = True, wordCount = True, wordsToKeep = 2500. Apply the filter and browse the attribute list. What is different about the attribute values and what do the numbers represent? (It may help to look at the data in the **Edit** window.)

Task B2: Go to the **Classify** panel. Classify the data using Naive Bayes Multinomial and then SMO. Record the TP and FP rate for the positive instances and the ROC area.

Task B3: Compare the results from the above task. Which model was better at classifying the articles and why (include your knowledge of TP/FP rates and ROC area in your answer)?

Task B4: Go back to the **Preprocess** panel and undo the StringToWordVector. Redo it, but change the wordCount setting to **false**. Return to the **Classify** panel and reclassify the data using the Attribute Selected Classifier with Naive Bayes (not Naive Bayes MultiNominal) classifier, InfoGain as the attribute evaluator and Ranker as the search method. Run the test three times, the first time set the numToSelect (in the Ranker settings) to 100, the second time change it to 50 and the third time change it to 25. Run the tests again with the AttributeSelectedClassifer and SMO and the settings 100, 50 and 25. Record the TP/FP rate and ROC area for each test.

Task B5: Compare the results from the above task and comment on your findings. (e.g. which test performed the best and why?)

Task B6: Compare the results from Task B2 with the results from Task A2. Did SMO perform better with binary attributes or with word count? Which performed better out of Naive Bayes and Naive Bayes Multinomial? Give an explanation for each outcome.

4 Working with Unknown Instances

In this section of the tutorial you will get to have a go at classifying a couple of instances that have their class attribute missing. The aim of this task is to build models using the techniques you learned in the above exercises, and to use the models to successfully classify the unknown instances. Before you start there are a few things to note about this exercise.

Because we want to use two different data files on the same model (one to train and one to test), the headers and attributes of both files must be identical. The test files have already been set up for you (so that you can't see where they came from), so the only thing you need to do is to apply the exact same filter to any training files you use.

The filter you will need to use is:

weka.filters.unsupervised.attribute.StringToWordVector

and you will need to change the following options: doNotOperateOnPerClassBasis = True, lowerCaseTokens = True, onlyAlphabeticTokens = True, outputWordCounts = True, wordsToKeep = 1000.

Hint: If you click 'Save' after running the filter on your training data, you can save the filtered data as a new .arff file (i.e. FilteredGrain.arff), and then you can reload this file each time you need to use it, instead of having to load the original file and filter it again and again.

The basic procedure for creating a model and then using it to classify an unknown instance is as follows:

Run the classifier on the training data (use training set for testing as it doesn't matter at this stage). Now that you have a model, change the test options to 'Supplied test set' and select the file you want to test (in this exercise it will be **Mystery1.arff** or **Mystery2.arff**).

Click the 'More options...' button, tick the 'Output predictions' box and hit 'OK'. Now **right-click** on the model you just built in the 'Result list' (generally the last item in the list) and select 'Re-evaluate model on current test set'. This will run the test instance through the classification model and the prediction results will appear under 'Predictions on test set'.

Here's a short explanation of the following columns:

inst#: The instance number - in this case it will always be 1.
actual: The actual classification of the instance - in this case it will be a question mark.
predicted: The predicted classification. The first number is the index of the class value, the second number is the predicted class - in this case 0 or 1.

probability distribution: these two numbers represent the model's predicted probability of each class value being correct. The first number applies to the first class value in the Confusion Matrix (in this case, 0) and the second number applies to the second class value (1).
Consequently, the higher the probability a value has, the surer the model is of its classification.

4.1 Classifying Unknown Instances

In this task we will be using both the **ReutersCorn-train.arff** data set and the **ReutersGrain-train.arff** data set as training sets and **Mystery1.arff** and **Mystery2.arff** as test sets.

Mystery1 and Mystery2 are files that contain one instance each. They have been modified so that the class value of both instances are missing. The instances have come from the Reuters test files, but the text has been replaced by attribute indices and word counts so they will appear unreadable.

Task C1: Use the above knowledge to help you classify the instance in the file **Mystery1.arff**. Use the Naive Bayes Multinomial classifier to build your models (for both corn and grain), and record the predicted outcome and the probability distribution for each model.

Task C2: Analyse your results from the above task. Is this instance an instance of the Corn data set, the Grain data set, or neither? Explain your answer.

Task C3: Now repeat Task C1, but use the SMO classifier instead of the Naive Bayes Multinomial classifier. Record the predicted outcome and the probability distribution for each model.

Task C4: Compare the above findings to your findings in Task C2. Do you still think your previous classification was correct or have you changed your mind (and to what)? Why/why not?

Task C5: Use Naive Bayes Multinomial classifier to classify the instance in **Mystery2.arff**. Does it belong to the Corn data set, the Grain data set, or neither? Record the predicted outcome and the probability distribution for each model, and comment on your findings (i.e. give an explanation for the results you got, and explain why you chose the category that you did).

5 Answers

Answer A1:

Answer A2: Naive Bayes: TP Rate ..... FP Rate .....
ROC Area .....
SMO: TP Rate ..... FP Rate ..... ROC Area .....

Answer A3:

Answer A4: Ranked Naive Bayes: TP Rate ..... FP Rate .....
ROC Area .....
Ranked SMO: TP Rate ..... FP Rate ..... ROC Area .....

Answer A5:

Answer A6:

Answer B1:

Answer B2: Naive Bayes Multinomial: TP Rate ..... FP Rate .....
ROC Area .....
SMO: TP Rate ..... FP Rate ..... ROC Area .....

Answer B3:

Answer B4: Ranked (100):
Naive Bayes: TP Rate ..... FP Rate .....
ROC Area .....
SMO: TP Rate ..... FP Rate ..... ROC Area .....

Ranked (50):
Naive Bayes: TP Rate ..... FP Rate .....
ROC Area .....
SMO: TP Rate ..... FP Rate ..... ROC Area .....

Ranked (25):
Naive Bayes: TP Rate ..... FP Rate .....
ROC Area .....
SMO: TP Rate ..... FP Rate ..... ROC Area .....

Answer B5:

Answer B6:

Answer C1: NBM:
Corn Training Data: Predicted Outcome ..... Probability of '0' .....
Probability of '1' .....
Grain Training Data: Predicted Outcome ..... Probability of '0' .....
Probability of '1' .....

Answer C2:

Answer C3: SMO:
Corn Training Data: Predicted Outcome ..... Probability of '0' .....
Probability of '1' .....
Grain Training Data: Predicted Outcome ..... Probability of '0' .....
Probability of '1' .....

Answer C4:

Answer C5: NBM:
Corn Training Data: Predicted Outcome ..... Probability of '0' .....
Probability of '1' .....
Grain Training Data: Predicted Outcome ..... Probability of '0' .....
Probability of '1' .....

Comments: