MCS507 Final Project

Applications of Bayesian Classifiers and the Orange Software System

Prepared By: Adam McElhinney December 4, 2012

1 Assignment One: Overview and Illustrative Example

Our first objective is to identify the main problem this technique aims to solve. We then provide an overview of the techniques and their implementation. We conclude with an illustrative example that displays the utility of this software.

1.1 Main Problem this Technique Aims to Solve

Bayesian Classifiers broadly refers to a class of tools that rely on Bayes rule to classify objects into various categories. Classification problems are found in nearly every aspect of academic and industrial researching. In particular, Bayesian techniques have proven to be extremely versatile. They have many broad applications in industry and academia. Applications of Bayesian classifiers are found in:

1. Spam detection
2. Speech Recognition (Merhav)
3. Diagnosis of Dental Pain (Chattopadhyay 2010)
4. Plant Identification

1.2 Overview of the Theory

As stated, Bayesian Classifiers rely on Bayes theorem, which states for two events A and B:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{1}$$

We can extend this theorem to any partition of the event space as:

$$P(A_i|B) = \frac{P(B|A_i) * P(A_i)}{\sum_j P(B|A_j) * P(A_j)} \tag{2}$$

Based on this simple rule, we can address problems of classification by taking a group of observations whose features are known. Then upon finding a suitable probability distribution, we can use Bayes Theorem to calculate the probability that another observation belongs to a certain class, conditional on its features (StatSoft, 2012).

1.3 Illustrative Example

We can illustrate this concept with an example. Suppose that all football teams are either winners or losers. Further, supposed that there are only two football teams on tv that day: the Chicago Bears and the Green Bay Packers. Since the Chicago Bears are such a superior team, they are winners 80% of the time and therefore losers 20% of the time. Whereas the Green Bay packers, being inferior in every way, are winners a mere 10% of the time and therefore losers 90% of the time. Now suppose upon turning on ESPN to catch the game scores, you hear them refer to a winning team, but cannot make out the name of the team. You would like to know which team they were discussing, but you know that they were either discussing the Chicago Bears or the Green Bay Packers with equal chance. Thus, we can use Bayes Theorem to represent this problem as:

$$P(Bears|Winner) = \frac{P(Winner|Bears) * P(Bears)}{P(Winner)} \tag{3}$$

$$P(Winner) = P(Winner|Bears) * P(Bears) + P(Winner|Packers) * P(Packers) \tag{4}$$

Using the known values, $P(Winner|Bears) = .9$ and $P(Bears) = .5$, we can compute that $P(Winner) = .8 * .5 + .1 * .5 = .45$. Finally, this implies

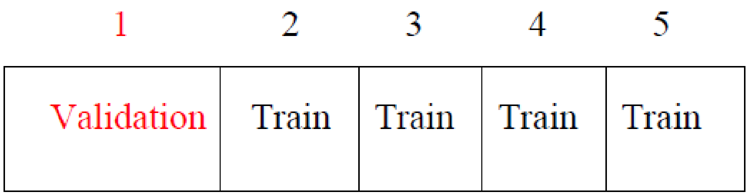
$$P(Bears|Winner) = \frac{.8 * .5}{.45} = 89\% \tag{5}$$

We can thus conclude with nearly 89% certainty that they were discussing the Chicago Bears and not the Green Bay Packers.

1.4 Overview of the Methodology

Now that we have an understanding of Bayes Theorem, we can further discuss the implementation of it for purposes of a Bayesian Classifier. As previously stated, a Bayesian Classifier builds off Bayes theorem to predict membership to a class. If the classifier uses strict assumptions about the independence of the variables, it is referred to as a Naive Bayes Classifier (How to Build a Naive Bayes Classifier, 2012).

These classifiers are typically implemented using k-fold cross validation. k-fold cross validation is a commonly used machine learning technique where the data is divided into roughly k equal parts (k being specified by the user at the outset). The model is then fit using the observations in k-1 parts. Then the model is then used to score the k-th part. The accuracy of the model is then computed based on its predictive power for this k-th part. This process is repeated k-times, using a different partition each time. The selection of the value k is somewhat arbitrary, but for large data sets, k=10 is generally accepted as a reasonable choice.



1	2	3	4	5
Validation	Train	Train	Train	Train

Fig. 1: Example of k-fold cross-validation, k=5. (Hastie & Tibshirani 2009)

2 Assignment Two: Overview of the *Orange* Software Package

2.1 About *Orange*

Bayesian Classifiers may be used in Python via the *Orange* modules. However, *Orange* is actually a comprehensive data mining suite with capabilities far beyond just one classifier.

The stated goal of *Orange* is

> ...not to cover just about any method and aspect in machine learning and data mining (although through years of development quite a few have been build up), but to cover those that are implemented deeply and thoroughly, building them from reusable components that expert users can change or replace with the newly prototyped ones.

To accomplish these goals, *Orange* developers refer to a "component-based framework". The three components of this framework are:

1. A library of C++ core objects and routines
2. A collection of Python modules that site over the core library
3. A collection of graphical widgets that are implemented via a visual programming framework

Orange is developed by the Artificial Intelligence Laboratory in the Department of Computer and Information Science in the University of Ljubljana, Slovenia (*Orange* Documentation, 10/20/2012). The suite was the realization that several researchers at the laboratory were each independently writing their own code for scoring, model building etc. *Orange* was accepted into the Google Summer of Code Project in 2012 which resulted in the creation of the following projects:

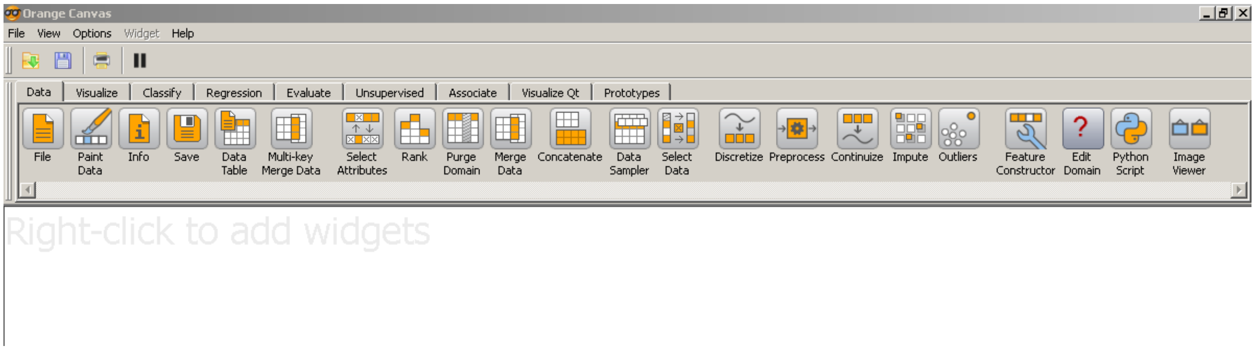
1. Widgets for statistics
2. Computer vision add-on for *Orange*
3. Multi-Target Learning for *Orange*

Orange features implementations of the majority of cutting-edge machine learning and data mining techniques, including techniques from the following categories:

1. Summary Statistics
2. Classification
3. Regression
4. Ensemble Algorithms
5. Clustering
6. Network Analysis

2.2 Overview of the *Orange* GUI

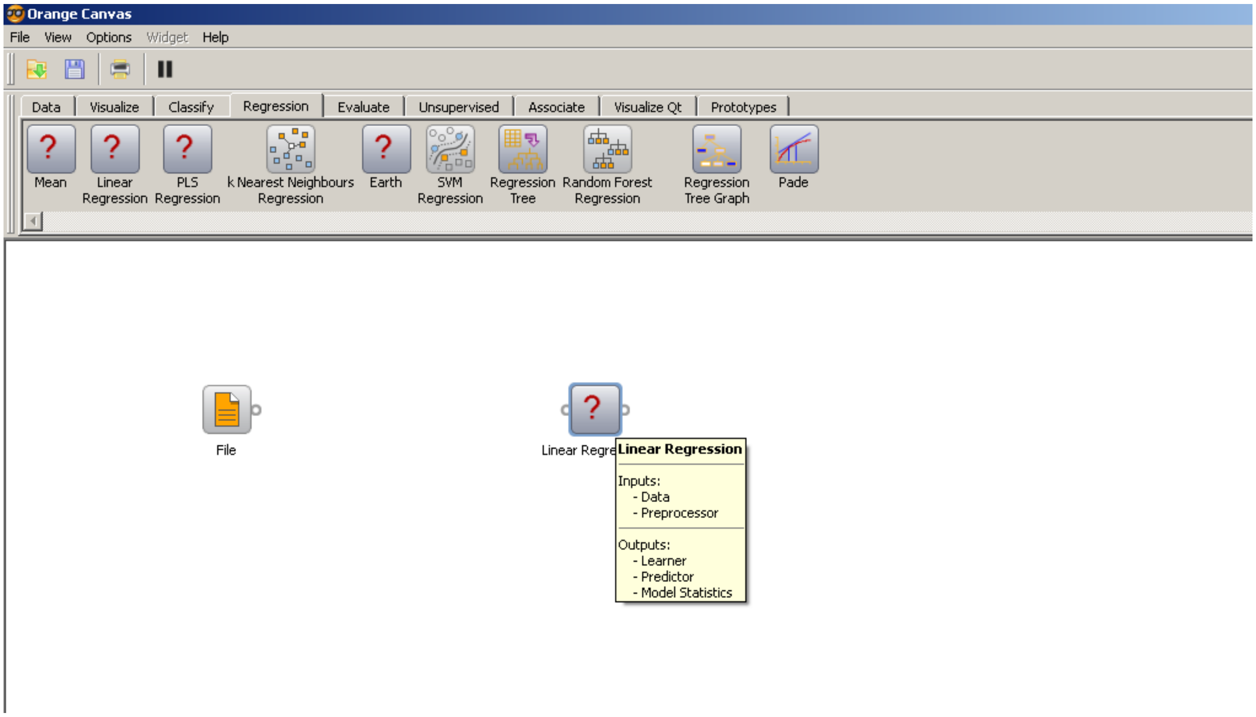
Orange features a GUI with a unique visual programming paradigm. Visual programming refers to the collection of widgets which are implemented on a visual canvas. These widgets are modular pieces of code that are manually dragged by the user onto the canvas.



Right-click to add widgets

Fig. 2: Image of the *Orange* Canvas with Widgets Displayed at Top

Each widget performs a discrete action and contains a set of inputs and outputs. Hovering over a widget reveals its inputs and outputs.



Linear Regression

Inputs:
- Data
- Preprocessor

Outputs:
- Learner
- Predictor
- Model Statistics

Fig. 3: A Widget Displaying its Inputs and Outputs

Then a widget with an output equal to the input of another widget can be combined by the user dragging a line from the one widget to the other. In this way, the user is “programming” without writing any code. Additional options for each widget can be found by clicking on the widget while it is placed on the canvas.

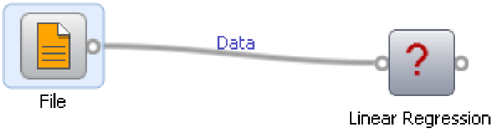


Fig. 4: Connecting Widgets

Orange claims that the user should be able to complete a sophisticated data mining project entirely using the canvas. However, in my experience it seems like supplemental Python knowledge would definitely be required. Particularly if one wanted to complete more sophisticated data cleaning or graphing than are supported by the canvas. However, the canvas definitely can be used to supplement the code.

3 Assignment Three: A More Substantial Example

3.1 Overview of Example

We know consider the application of Bayesian Classifiers to a simple problem; determining whether an object is likely to be blue, green, or yellow based upon its position (Note that the shapes of the objects were added to provide additional visual distinction and do not represent another dimension of the data). The dots were created manually using the *Orange* module's data painting feature. The groups were specifically chose such that they have unequal size and that there exists substantial overlap between the groups.

Tab. 1: Overview of the Data Set

Category	Count
Blue	900
Green	630
Yellow	760

We begin our implementation by writing functions to split the data into separate lists for each class. This will be used later for graphing and scoring purposes.

```python
########################
# Functions to split the data
########################


def unique(data, class_col):
    """
    Creates a list of unique observations from data
    class_col=the column number containing the group
    """
    groups=[]
    for i in range(len(data)):
        # Check to see if this group has been added to groups
        if data[i][class_col] not in groups:
            groups.append(data[i][class_col])
    return groups


# Split the data into unique groups
def unique_split(data,class_col):
    # Get the list of unique groups
    r=unique(data,class_col)
    p=[]
    for q in range(len(r)):
        p.append([])
    for i in range(len(data)):
        #print i
        for j in range(len(r)):
            #print j
            if data[i][class_col]==r[j]:
                p[j].append(data[i])

    return p
```

Listing 1: Partition the Data

Next, we import the data using the $C4.5$ data format. This format is common in machine learning as it provides definitions of the variables, as well as explicit type declarations that are understood by the majority of machine learning software tools. We then split the data into the respective groups.

```
data = orange.ExampleTable(”3_groups”)
print data.domain.attributes
print data[:4]

# Get a small amount of data
index=Orange.data.sample.SubsetIndices2(p0=0.10)
ind=index(data)
#data_test=data.select(ind,0)
data_test=data


########################
# Split the data
########################

X, Y = data_test.to_numpy(”A/C”)
data_2=[]
for i in range(len(Y)):
    data_2.append([X[i][0],X[i][1],Y[i]])
p=unique_split(data_2,2)

# Group 1
X11=[p[0][i][0] for i in range(len(p[0]))]
X12=[p[0][i][1] for i in range(len(p[0]))]

# Group 2
X21=[p[1][i][0] for i in range(len(p[1]))]
X22=[p[1][i][1] for i in range(len(p[1]))]

# Group 3
X31=[p[2][i][0] for i in range(len(p[2]))]
X32=[p[2][i][1] for i in range(len(p[2]))]

# Obtain the counts
len(X11);len(X21);len(X31)
```

Listing 2: Import the Data and Split

Next, we prepare the data into the form required by *Matplotlib* and plot the data.

```
########################
# Plot the data
########################


import matplotlib.pyplot as plt
import matplotlib
fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(X11, X12, s=10, c='b', marker=”+”)
```

```
ax1.scatter(X21, X22, s=10, c=’c’, marker=”o”)
ax1.scatter(X31, X32, s=10, c=’y’, marker=”x”)
plt.title(’Plot of Three Classes of Data’)
plt.show()
```

Listing 3: Prepare the Data for Plotting; Plot the Data

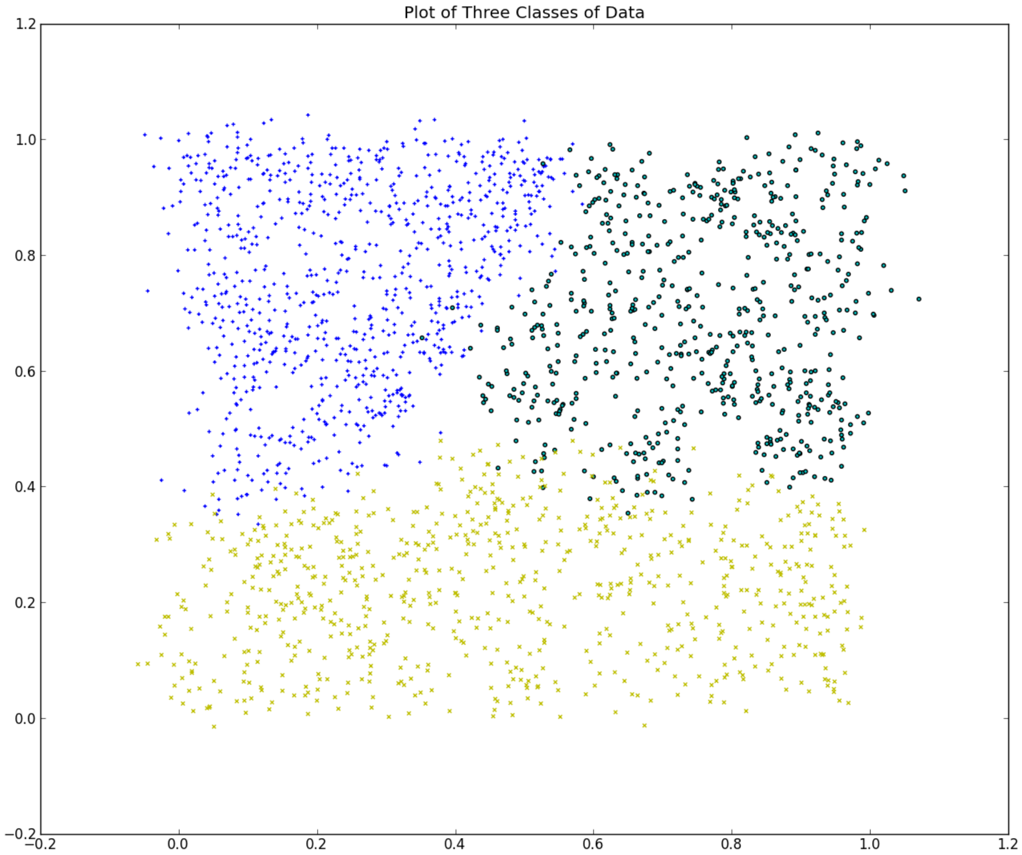


Fig. 5: Classification of 3 colors in 2-dimensional space

Using the Bayesian methods outlined in the first section, a Naive Bayes Classifier was trained and validated using k-fold cross validation.

```
#########################
# Build Classifier
#########################

import orange, orngTest, orngStat, orngTree
classifier = orange.BayesLearner(data)
bayes = orange.BayesLearner()
```

```
bayes.name = "bayes"
learners = [bayes]

results = orngTest.crossValidation(learners, data_test, folds=10)
```

Listing 4: Construct the Classifier

Once the classifier is constructed, we wish to compute the misclassified operations. The package *Orange* provides a convenient way of doing this, however it cannot interface directly with *Matplotlib*. Thus, the data was converted to *NumPy* arrays.

```
########################
# Compute the misclassified observations
########################

X, Y = data_test.to_numpy("A/C")
data_scored=[]
for i in range(len(results.results)):
    if results.results[i].classes[0]==results.results[i].actual_class:
        data_scored.append(1)
    else:
        data_scored.append(0)

import matplotlib.pyplot as plt
import matplotlib

X1w=[];X2w=[]
for i in range(len(X)):
    if data_scored[i]==0:
        X1w.append(X[i][0])
        X2w.append(X[i][1])
```

Listing 5: Compute the Misclassified Observations

We now overlay the misclassified observations onto plots of the data points to give a visual approximation of how successful is our classifier.

```
########################
# Plot the misclassified data
########################
fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(X11, X12, s=10, c='b', marker="+")
ax1.scatter(X21, X22, s=10, c='c', marker="o")
ax1.scatter(X31, X32, s=10, c='y', marker="x")
ax1.scatter(X1w, X2w, s=10, c='m', marker="^")
plt.title('Plot of Three Classes of Data, Showing the Misclassified Elements')
plt.show()
```

Listing 6: Compute the Misclassified Observations

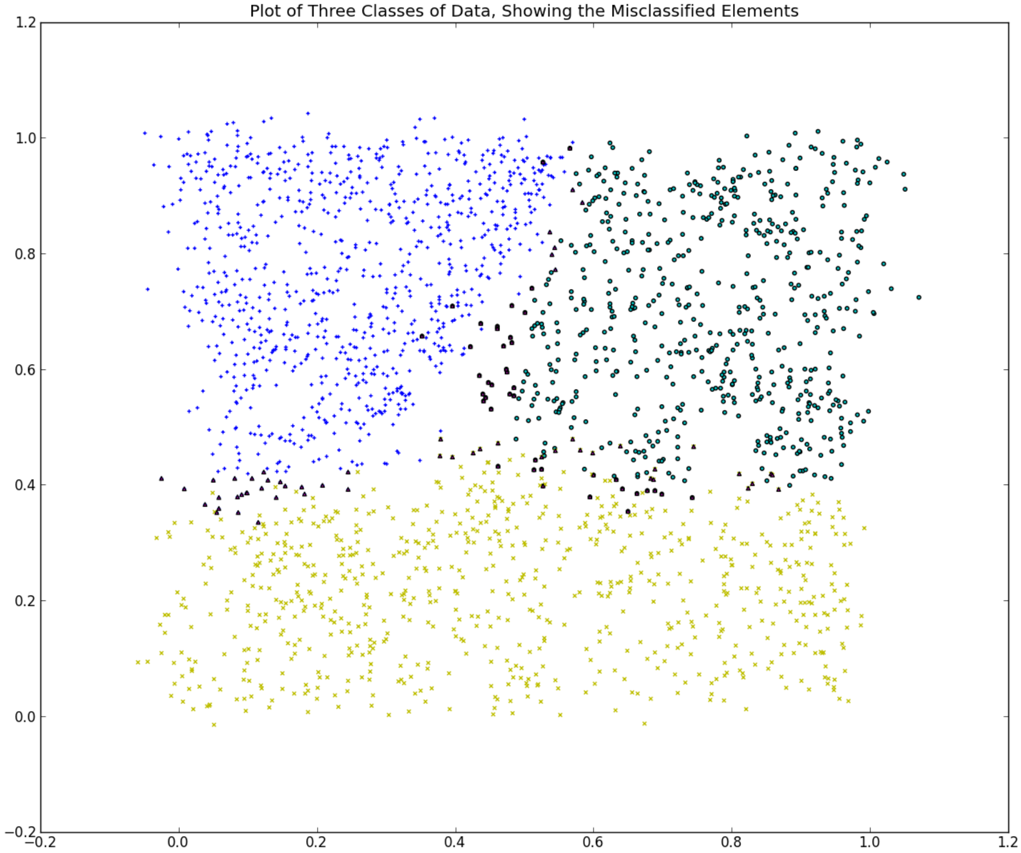


Fig. 6: Classification of 3 colors in 2-dimensional space; Misclassifications displayed

As the above figure shows, our classifier is quite successful. We see a small number of misclassified observations (represented as dark triangles) that predictably lie in the boundaries between the three groups. However, in addition to a visual representation of the accuracy of the classifier, the *Orange* package provides us with a full range of summary statistics to assess the model performance.

```
# output the results
print "Learner CA IS Brier AUC"
for i in range(len(learners)):
    print "%-8s %5.3f %5.3f %5.3f %5.3f" % (learners[i].name, \
    orngStat.CA(results)[i], orngStat.IS(results)[i], orngStat.BrierScore(results)[i], orngStat.AUC(results)[i])
```

Listing 7: Compute the Misclassified Observations

Tab. 2: Model Performance Statistics

Statistics	Value
Classification Accuracy	96%
Information Score	1.301
Brier Score	.093
Area Under ROC	.998

The exact definitions of all of these metrics is beyond the scope of this paper. However, one metric to call out is the Classification Accuracy. This is simply the percentage of observations that are classified correctly. The value of 96% represents an excellent classifier. This intuitively matches our observations based on the graph of misclassifications.

4 Conclusion

We have shown how the *Orange* software package implements the Bayesian Classifiers and how it can be very useful for quickly solving real world problems. Some areas of consideration for additional explanation for the final presentation are:

1. Evaluation on another ”real-world” data set
2. High dimensional data (data sets where there are a much larger amount of explanatory variables as compared with the number of observations)
3. Comparison with other classifiers (Support vector machines, Neural Networks, etc)

In summation, Bayesian Classifiers are a versatile tool with applications to real world data sets.

5 Bibliography

About. (2012). Orange. Retrieved 10/20/2012 from http://orange.biolab.si/about/

Chattopadhyay S, Davis RM, Menezes DD, Singh G, Acharya RU, Tamura T. ”Application of Bayesian classifier for the diagnosis of dental pain.” J Med Syst. 2012 Jun;36(3):1425-39. Epub 2010 Oct 13.

Cross-Validation. (2012). Wikipedia. Retrieved 11/18/2012 from https://en.wikipedia.org/wiki/Cross-validation_%28statistics%29

Data Mining Fruitful and Fun (2004). Orange. Retrieved 12/01/2012 from http://orange.biolab.si/wp/orange-leaflet.pdf

Hastie & Tibshirani.”Cross-validation and bootstrap”. Retrieved from www-stat.stanford.edu/t̃ibs/sta306b/cvwrong.p November 27, 2012.

How To Build a Naive Bayes Classifier. Retrieved 11/20/2012 from http://bionicspirit.com/blog/2012/02/09/howto-build-naive-bayes-classifier.html/

Naive Bayes classifier. (2012). Wikipedia. Retrieved 11/18/2012 from https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Orange and Visual Programming.(2004). Orange. Retried 12/02/2012 from http://orange.biolab.si/wp/orange-leaflet-visual.pdf

StatSoft Electronic Statistics Textbook. Naive Bayes Classifier. StatSoft. Retrieved 11/15/2012 from http://www.statsoft.com/textbook/