Plant Identification by Leaf Shape Using Multiple Implement Methods/Algorithms

Supervisor: Dr. Reza Zafarani

Author: Wei Liu

Data Mining Course Project

December 2016

Syracuse University

1. Introduction

There are estimated to be nearly half a million species of plant in the world. Considering such huge number of species, sometimes, identifying and classifying different plants is a challenging work even for the plant specialists. Among the characteristics of the plants, for example, size, leaf shape, flower color, odor, form, etc., leaf shape is considered a good one for the classification and data mining. Figure 1 is a typical leaf image. From this kinds of image, we can extract the useful data for further classification. There are several topics in Kaggle discussed how to convert the leaf image to numbers as attributes. Now let's suppose we already get all the quantities numbers for each leaf class, which is provided in Kaggle too.

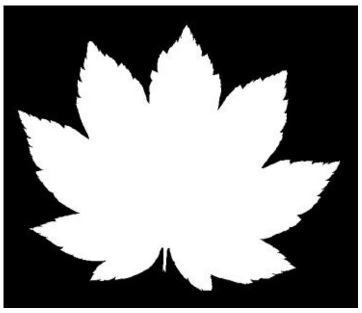

Figure 1: A leaf shape image for classification

The training data and test data I will use can be obtained from Kaggle. There are in total 192 attributes for each instance. In the training data, there are 1584 instances. In this project, different classifiers for this plants species classification/identification problem are built by plain python code, python sklearn package and also WEKA respectively. Also, the classifiers will be applied on the test dataset, and the performance of different classifiers will be evaluated. We will use the Holdout to estimate the overall accuracy for each classifier. The training set is splited by a fixed ratio. We use the larger portion as the training dataset and the smaller dataset for testing. Through this project, my goal is 1) deepen understanding to the algorithms naive Bayes and decision tree; 2) practice using the data mining tool WEKA; and 3) practice my coding skills with python.

2. Model/Algorithm/Method

The hardest part is to build the naive Bayes and decision tree using plain code with python. Following part is a brief recap of those two algorithms.

$$\underbrace{P(c\,|\,x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\,|\,c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\,|\,\mathrm{X}) = P(x_1\,|\,c)\times P(x_2\,|\,c)\times\cdots\times P(x_n\,|\,c)\times P(c)$$

Figure 2: Naive Bayes formula and interpretations

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \boxed{\text{Mean}}$$

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu)^2\right]^{0.5} \qquad \boxed{\text{Standard deviation}}$$

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad \boxed{\text{Normal distribution}}$$

Figure 3: Probability density function for the normal distribution

Figure 2 is the formula of naive Bayes classifiers. Since in this project, all the data for attributes are continuous numbers, so we calculate the possibility assuming the numbers follows normal distribution, as shown is figure 3.

For the decision tree algorithm, we assume that each internal node has two branches. Since all the instance data for one attribute are the numerical type, so we split the instances at internal node by simple >= or < comparison. The entropy was used as the measure criteria of heterogeneity, which can be calculated as in figure 4. For each node, we choose the attribute which can generate largest information gain by comparing the entropy before and after the split.

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_b p(x_i)$$

Figure 4: Calculation of the entropy at nodes

For the classifiers test with WEKA, we have to 1) converting .csv file to .arff file, and 2) modifying the data type of column attributes (from string to nominal), otherwise the "incompatible data type" problem will happen for further modeling.

3. Results and Conclusions

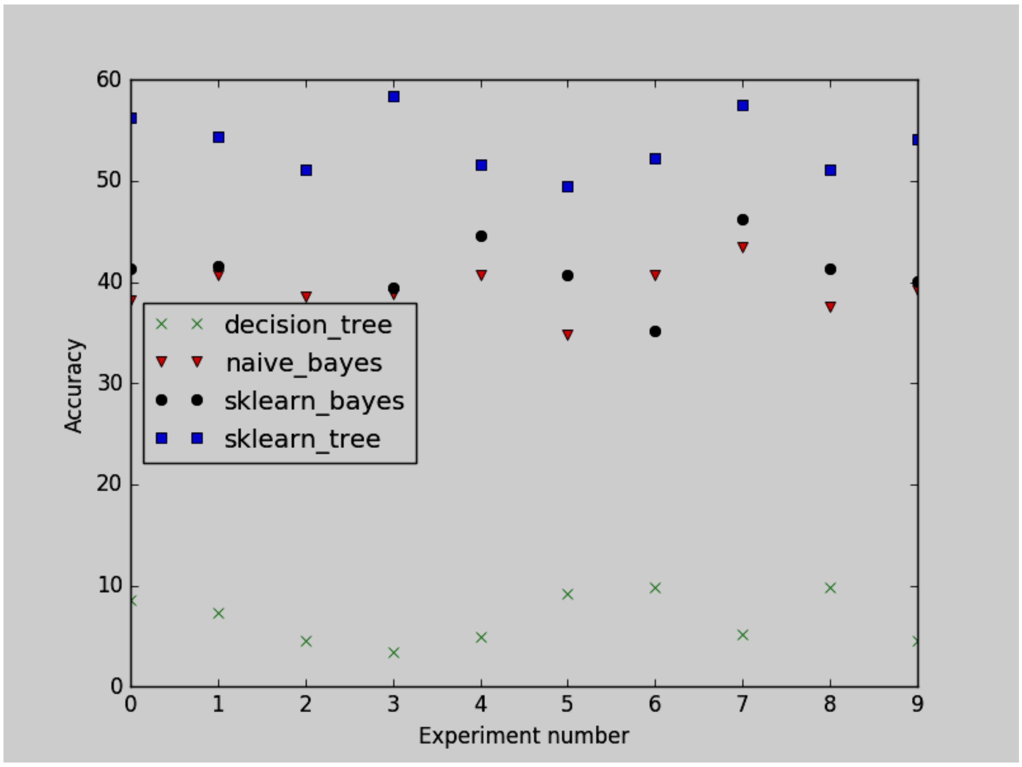

Figure 5: Comparison of accuracy of different algorithms/methods

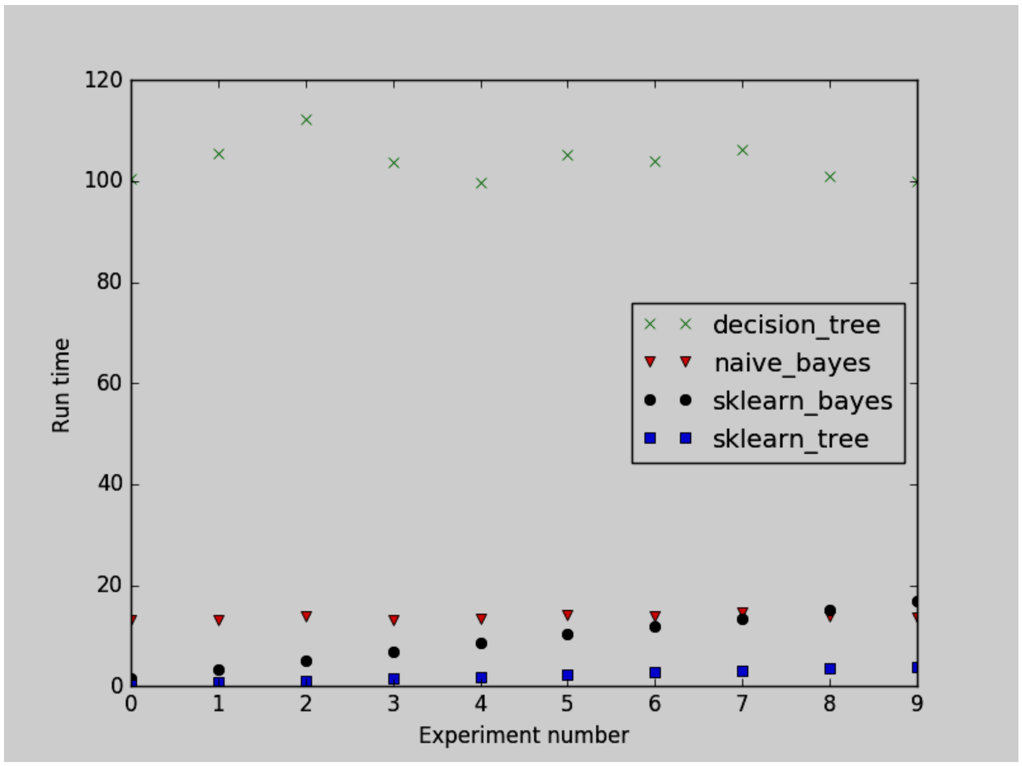

Figure 6: Comparison of runtime of different algorithms/methods

The implemented of naive Bayes and decision tree by sklearn tool seems very concise and simple. After the training dataset and testing dataset get prepared, you can call those methods like the following:

```
from sklearn.naive_bayes import GaussianNB
Y = np.array([instance[0] for instance in trainingSet])
X = np.array([instance[1:] for instance in trainingSet])
clf1 = GaussianNB()
clf1.fit(X, Y)


from sklearn import tree
Y = np.array([instance[0] for instance in trainingSet])
X = np.array([instance[1:] for instance in trainingSet])
clf1 = tree.DecisionTreeClassifier()
clf1.fit(X, Y)
```

The average runtime and average accuracy is demonstrated like following:

```
decision tree average time consumed: 103.801600051
decision tree average accuracy: 6.72782874618
naive Bayes average time consumed: 13.60589993
naive Bayes average accuracy: 39.2660550459
sklearn Bayes average time consumed: 9.31220002174
sklearn Bayes average accuracy: 40.4892966361
sklearn decision tree average time consumed: 2.13099994659
sklearn decision tree average accuracy: 53.6391437309
```

From the figure 5, we can see that the rank for the accuracy is: sklearn_decision_tree > naive Bayes ≈ sklearn_Bayes >> decision tree. It is surprise to see the naive Bayes algorithm implemented with plain code has comparable performance with the sklearn_Bayes. We can see that the highest accuracy of sklearn_decision_tree method is around 53%, which is still not quite satisfactory.

The most critical observation is that there is an over-fitting problem for the decision tree implemented with plain code, which means the result can be affected by noise instance. While in the decision tree implemented using sklearn tool, the tree may has been post-pruned. For all those four methods, moderate fluctuation is observed, which should be related with the random splited dataset.

From the figure 6, we can see that the rank for runtime is the following: sklearn_decision_tree sklearn_Bayes < naive Bayes << decision tree. It proves that the algorithm implemented with sklearn indeed runs faster, especially for the decision tree method. It is also observed that there is a strange trend for sklearn_Bayes method: the runtime is gradually increasing with the experiment number.

Building the code requires further attention to the details of those algorithms. From my experience of this project, I have learnt several other things that need to be keep in mind for data mining.

First, it is always important to preprocess the dataset as the first step. Some useless rows/columns in the dataset, which is provided as csv file, were deleted (e.g., the header in the first row and the IDs in the first column). Sometimes, the original dataset needs to be carefully examined before further steps were taken. Although this part is not in the core part in the algorithm, cleaning the data is always necessary to keep the following code concise, neat and easier for understanding.

Secondly, checking the code on a smaller partial dataset before applying it to the whole dataset is important too. This trick will help you check the correctness of the code easier and more efficiently.

Using the same dataset, the naive Bayes and decision tree algorithms was implemented using WEKA with 10 fold cross-validation. A zoom-in part of the decision tree is shown in figure7.

Table 1 Summary for naive Bayes classifier in WEKA

```
Time taken to build model: 0.23 seconds

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        957               96.6667 %
Incorrectly Classified Instances       33                3.3333 %
Kappa statistic                       0.9663
Mean absolute error                    0.0007
Root mean squared error               0.026
Relative absolute error                3.4332 %
```

Root relative squared error	26.0347 %
Total Number of Instances	990

Table 2 Summary for decision tree classifier (J48) in WEKA

Correctly Classified Instances	675	68.1818 %
Incorrectly Classified Instances	315	31.8182 %
Kappa statistic	0.6786	
Mean absolute error	0.0067	
Root mean squared error	0.0784	
Relative absolute error	33.48 %	
Root relative squared error	78.4306 %	
Total Number of Instances	990	

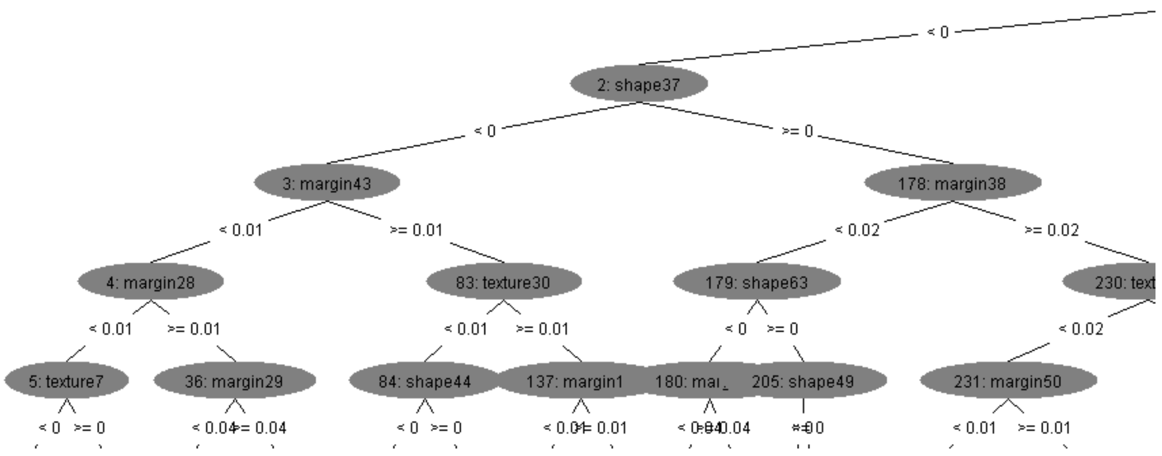

Figure 7: Partial zoom-in of the visualized decision tree

We can see that the naive Bayes classifier in WEKA has a very high accuracy (96.7%), while the decision tree has only 68.2% accuracy. Also, it is obvious that the model was built and validated (10 fold cross validation) in a short time compares with those implemented with python.