CS235 Data Mining Final Project Report

KUAI YU, University of California, Riverside
I-HSIEN HUANG, University of California, Riverside
YIFEI LAI, University of California, Riverside

In our project, we selected a dataset (hereafter referred to as the Mushroom dataset which categorizes mushroom types as either poisonous or edible based on various attributes such as gill-color, cap-shape and odor) from UCI's machine learning repository: https://archive.ics.uci.edu/ml/datasets/mushroom. For our KDD techniques, we have decided to use Clustering, Random Forests and Bayesian classifiers.

KEYWORDS

K-Medoids, Random forests, Bayesian

1. INTRODUCTION

Through our project, we aim to employ three different binary classification methods to be used to label a mushroom as either edible or poisonous based on its 22 features. The three algorithms are: Naïve Bayes Classifier, Random Forest Classifier, and K-Medoids Clustering Analysis. In this report, we analyze, compare and contrast these algorithms based on their accuracy and f1 performance measures in order to determine the most effective and cost-efficient classification method.

2. DATA PREPROCESSING

Prior to the effective training of our machine learning models, it is necessary to ensure the training set is cleaned and free of error or missing values. This is done via OpenRefine where we were able to locate a potential source for error: namely that one of the attributes (stalk-root) contained 2480 empty values denoted by '?'. This is an understandable error seeing that it is likely that the stalk was damaged or severed during the mushroom data collection process. We opted to pad these empty cells with the mode value for this particular feature. This decision was made because first, we believe the type of the mushroom gill could provide us with important insight as to the toxicity of the mushroom and was therefore too valuable an attribute to simply discard. Secondly, after small-scale testing of our classifiers before and after data padding, it was deemed safe to replace the null values by the most common category of 'bulbous' with a count of 3776 out of 8124. This was done in OpenRefine by first clustering the null-valued rows of the dataset and then replacing these values by 'b' (fig. 1).

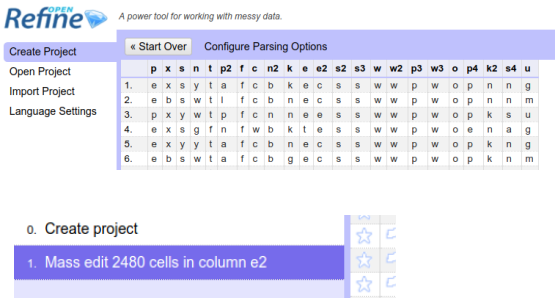


fig. 1, Creating an OpenRefine Project to clean null-data in column e2.

```
[208]: ['poisonous',
        'cap-shape',
        'cap-surface',
        'cap-color',
        'bruises?',
        'odor',
        'gill-attachment',
        'gill-spacing',
        'gill-size',
        'gill-color',
        'stalk-shape',
        'stalk-root',
        'stalk-surface-above-ring',
        'stalk-surface-below-ring',
        'stalk-color-above-ring',
        'stalk-color-below-ring',
        'veil-type',
        'veil-color',
        'ring-number',
        'ring-type',
        'spore-print-color',
        'population',
        'habitat']
```

Out[209]:

	poisonous	cap-shape	cap-surface	cap-color	bruises?	odor	gill-attachment	gill-spacing	gill-size	gill-color	...
0	e	x	s	y	t	a	f	c	b	k	...
1	e	b	s	w	t	l	f	c	b	n	...
2	p	x	y	w	t	p	f	c	n	n	...
3	e	x	s	g	f	n	f	w	b	k	...

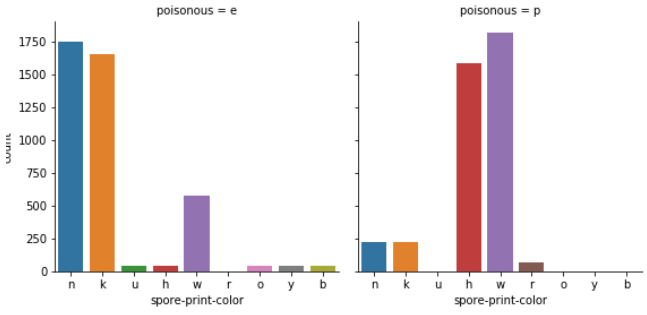
Out[211]:

	poisonous_e	poisonous_p	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	...
0	1	0	0	0	0	0	0	1	0	0	...
1	1	0	1	0	0	0	0	0	0	0	...
2	0	1	0	0	0	0	0	1	0	0	...
3	1	0	0	0	0	0	0	1	0	0	...

fig. 2, Mushroom Datasets attributes after renaming and the dataframe printed out in Pandas before and after one-hot-encoding. (Note that one-hot-encoding was only used for the Naïve Bayes classifier approach while the other approaches used label-encoding or ascii-encoding)

Before the raw Mushroom dataset can be used, it must first undergo data transformation whereby the nominal data values for each feature (e.g. 'e' and 'p') are changed into numerical representations (i.e. '1' and '0'). This is because none of the proposed classifiers are able to interpret non-numeric data and some classifiers (such as Naïve Bayes) are better off with binary representations of an occurrence within a feature (fig. 2). For this reason, one-hot-encoding was achieved using Pandas' get_dummies function call for Naïve Bayes, whereas direct ascii-translation was used for the Random Forest Classifier and label-encoding used for the Clustering Analysis. The pros and cons of using these encoding schemes will be discussed in later parts.

In order to get a feel for the distribution and frequency of various categorical values for each feature and how these values are correlated to the toxicity of a given mushroom, we used seaborn's factorplot to plot the frequency of each nominal category from each feature for both labels of poisonous = 'e' (edible) and poisonous = 'p' (poisonous). The results reveal a lack of obvious correlation between features, which means Naïve Bayes' feature independence assumption holds. However there are certain features in which we see a clear correlation between the high frequency of certain nominal values (or lack thereof) for edible and poisonous mushrooms. For instance, a plot of the spore-print-color frequencies show that there is a high chance that chocolate or white spore prints indicate poisonous mushrooms. Similarly, if a mushroom exhibits no odor, then it is likely to be edible. N.B. the converse is not true judging from the plot, since not all edible mushrooms exhibit no odor, and only a few poisonous mushrooms exhibit no odor.



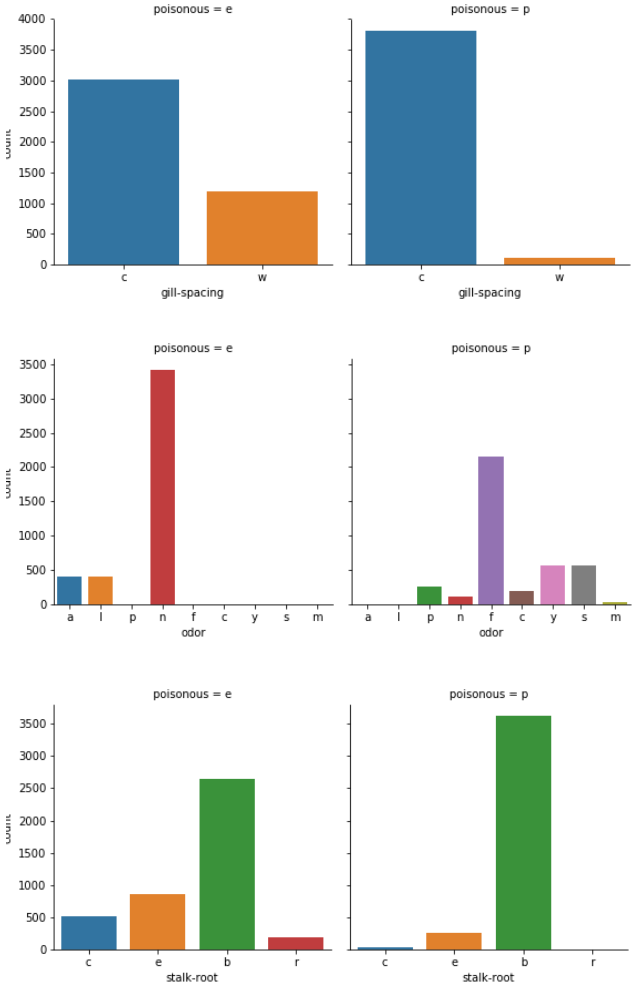


fig. 3, factor plots of two attributes (top) which contribute significantly to the toxicity of a mushroom and two attributes (bottom) which do not seem to significantly affect toxicity. Note that the Mushroom Dataset is a balanced dataset with 4208(51.8%) edible and 3916(48.2%) poisonous mushrooms.

3. KDD TECHNIQUES AND EXPERIMENTAL DETAILS

3.1 Naïve Bayes Classifier

3.1.1 Introduction

The reason why naïve Bayes is considered for our project is because of its ability to produce accurate classification for complex datasets with a minimal amount of training data. In some instances, naïve Bayes can even outperform more complex models such as logistic regression and decision trees for nominal input values. However, one critical disadvantage of using naïve Bayes is that it maintains the strong assumption that a dataset's features are independent, which is seldom the case

in the real world where data features are often closely related. In our implementation we used three extensions of the vanilla Naïve Bayes classifier to predict the toxicity of given mushrooms.

3.1.2 Implementation

The Naïve Bayes classifier is developed in Python on Jupyter Notebook for its fast prototyping and modular approach to development and ability to plot graphs and charts selectively which are useful for debugging and data analytics. Modules such as scikit-learn, Matplotlib, Pandas, Seaborn and Numpy are used to speed up the development process.

The first step in training the Naïve Bayes classifier is to divide the input data to be stored in X and Y variables, in which X stands for the feature values and Y stands for the target label. We then perform a training/testing dataset split using scikit-learn's train_test_split function, in which we allocate 80% of the available data to be used as training data and the rest as testing data. For the Naïve Bayes Classifier, we will forgo the additional allocation of validation data unless a bad classification accuracy score demands adjustments to our hyperparameters. After the split, we receive 5686 training and 2437 testing mushroom items.

The classification model we use is the Naïve Bayes classifier from scikit-learn. There are three implementations of Naïve Bayes from which we could choose: Bernoulli Naïve Bayes, Multinomial Naïve Bayes and Gaussian Naïve Bayes.

The Gaussian Naïve Bayes classifier is an extension of the Naïve Bayes classifier in which we assume an underlying gaussian distribution for our mushroom dataset. This is beneficial because many real world datasets do possess a gaussian distribution and it is relatively easy to implement, only requiring a mean and variance inputs. The Gaussian distribution applied to likelihood has following form where μ_i & σ_i are the mean and variance for class i:

$$p(X|C_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(\Theta-\mu_i)^2}{2\sigma_i^2}}$$

The trained Gaussian NB produced the following accuracy and f1 scores:

```
Misclassified samples: 127 out of 2437
Accuracy: 0.9479 F1: 0.9487
```

The Multinomial Naïve Bayes classifier is an extension of the Naïve Bayes classifier which uses a multinomial distribution for each of the features. Namely, when we estimate the likelihood distribution of a pair, $P(x_i|C_j)$, we assume the (i, j) pair possess a multinomial distribution. This works well for datasets which contain occurrences of categorical or nominal data (i.e. our dataset). The Multinomial NB classifier additionally takes into account the number of occurrences of each feature category. The Multinomial NB classifier has the following likelihood where xi is the number of times a categorical data has occurred for k output labels:

$$p(X|C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki^{xi}}$$

The trained Multinomial NB produced the following accuracy and f1 scores:

```
Misclassified samples: 115 out of 2437
Accuracy: 0.9528 F1: 0.9575
```

The Bernoulli Naïve Bayes classifier is an extension of the Naïve Bayes classifier which assumes a Bernoulli distribution for each of the features. Bernoulli NB is typically used for document and text classification, and features are assumed to be independent binary variables. This fits our one-hot encoded dataset. Furthermore, Bernoulli NB has the advantage of taking into account the non-occuring feature categories, which are ignored in the Multinomial NB classifier. The equation for Bernoulli NB likelihood is as follows:

$$p(X|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}(1 - p_{ki})^{(1-x_i)}$$

The trained Bernoulli NB produced the following accuracy and f1 scores:

```
Misclassified samples: 153 out of 2437
Accuracy: 0.9372 F1: 0.9436
```

3.1.3 Evaluation

Judging by the accuracy and f1 scores (which is a more comprehensive performance measure that takes precision and recall into account), the various Naïve Bayes implementations are ranked in the following order from most effective to least effective: MultinomialNB → BernoulliNB → GaussianNB. We can also visualize the learning curves of all three classifiers and see how their accuracy increase with the number of training data:

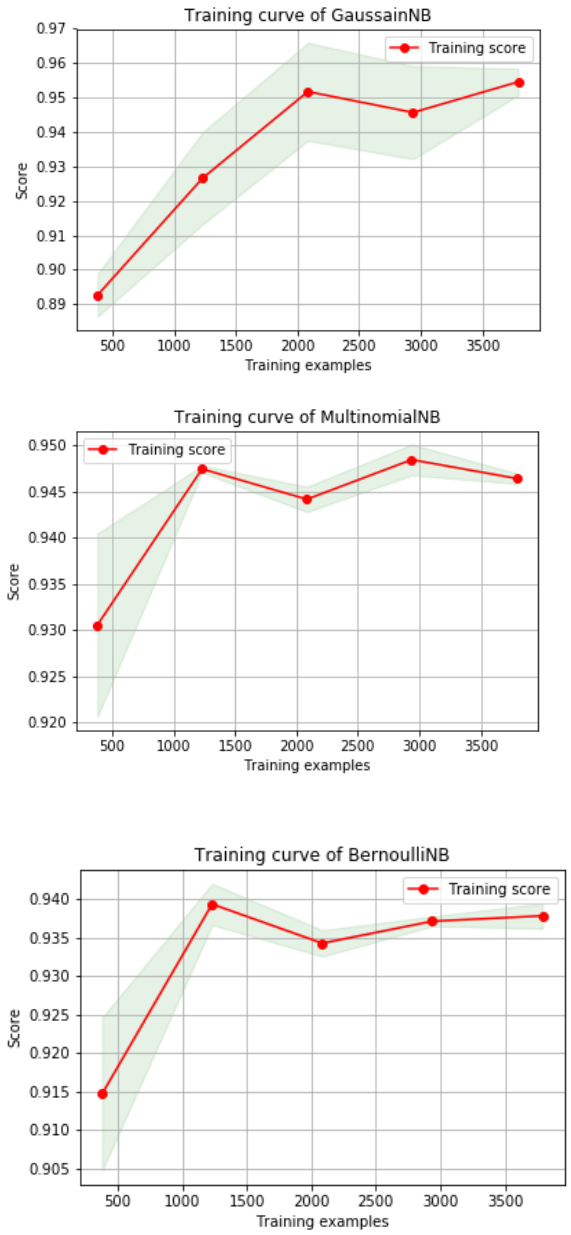


fig. 4, Learning (training) curves of the GaussianNB, MultinomialNB and BernoulliNB classifiers, where the red curve is the mean of accuracy scores with the green shaded area being the error boundaries.

Although an f1 score of 0.9575 may seem quite accurate, but given the relatively small input dataset, we should be able to define a classifier with much better performance. In the case of the mushroom dataset, attributes such as cap-color, gill-color and stalk-color are typically not conditionally independent, which could be the reason why the Naïve Bayes approach ended up producing non-optimal classification results.

3.2 Random Forest Classifier

3.2.1 Introduction

The Random Forest classifier is a form of supervised learning. And as the name suggests, it will build a lot of decision trees with random features. Normally, the accuracy increases as the number of the trees increase. However, when the number of trees increases, it will take more time to build the decision trees. Each node is a feature, and choosing random features to build the tree. After training, each tree will vote for testing data to determine what kind of label it is. Therefore, it will be less possibility to over-fitting.

3.2.2 Implementation

There are several reasons why we choose to use the Random Forest Algorithm. Firstly, there are not too many kinds in each feature. That means Random Forest's Decision Trees may contain most parts of the condition, and the cost of computation will not be high at the same time. Besides, there are only 22 features and 1 label. Therefore, there does not exist a big overhead for building random forest trees.

In the implementation, we decided to code with python, because it comprises several useful and practical functions. Before we start learning the data, we translate the data from character to integer, for the reason that in python, sklearn functions for decision trees and random forests only receives non-string arguments. The following processes help us learn the data accurately. Firstly, split the data to two different sets, one for training(70%) and the other for testing(30%). And then pass the features and label into the function. After spilting the data, we set the arguments (hyperparameters) on how we use the random forest decision trees. For example, max_depth, min_nodes, how many percents for training, max_features, and so on. After setting the training model, we call the model and pass the training set into it. The prediction is then produced. After which, we use the function accuracy_score (which is number of correctly classified samples/total samples) to compare the prediction and real data to count the accuracy for training data and testing data.

When it comes to visualization, we visualize the importance of the features, for the reason that it helps us to analyze the priority of the features in determining the label of these mushrooms.

Therefore, even if the dataset volume grows sharply, we still can utilize these significant features to reduce the complexity in Random Forest Learning to prevent overfitting, and it will decrease the learning time by producing a higher convergence rate. Besides, we visualize the curve of the accuracy related to the number of training data. It will allow us to figure out how many training data we should train to get the real model or pattern of these poisonous mushrooms.

3.2.3 Evaluation

Upon training the model on the training set, it achieved a training accuracy of 1.00 and a testing accuracy of 0.996 for 10 decision trees within the random forest. It is therefore no longer necessary to perform cross validation of hyperparameters as a good result is already attained. After we analyze the data and the model of Random Forest, we figure out why the accuracy was really high. Firstly, the data is not complex, hence it was easy for the decision tree to classify the feature and in the process contain all possible of combinations of features that would lead to an accurate outcome without overfitting. Furthermore, because there exists a few features which were very much indicative of whether or not mushrooms were poisonous, that once these features were used, the accuracy could be easily ascertained. To test this theory, we used different subsets of data and selectively chose combinations of attributes to test the algorithm, and the results show that the algorithm is indeed sound and useful.

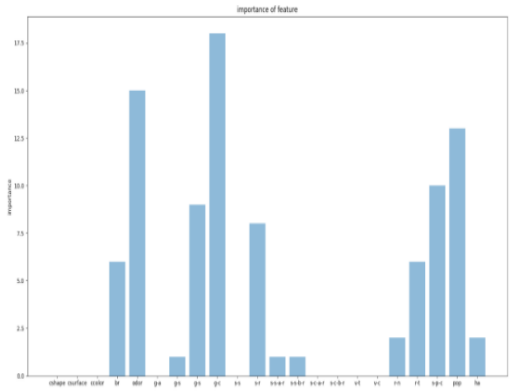


fig. 5, the importance of each figure in determining the output label for the classifier. As shown, some of the most important features were already understood during data preprocessing.

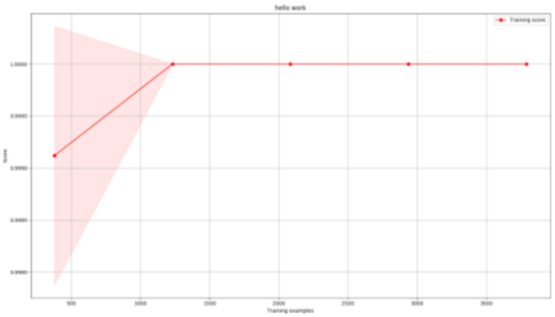


fig. 6, the learning curve of the random forest classifier. Note that after around 1250 sets of training data, the training accuracy has already been increased to 1.00, meaning that the random forest classifier is efficient and converges very quickly.

3.3 K-Medoids Clustering Analysis

3.3.1 Introduction

We choose to use the k-medoids algorithm to perform the clustering analysis. The Mushroom dataset can be classified into two classes, poisonous and edible. Those objects which represent the mushrooms in one class should be close to each other in the multi-dimensional feature space. We can pinpoint the position of each object according to its location, which would allow us to calculate the distance between mushroom samples (and thereby their similarity) and finally obtain two partitions of those objects in the space.

The k-medoids algorithm is similar to k-means algorithm, with the k-medoids algorithm choosing one data point as the center of one cluster as opposed to calculating a theoretical average. Each time an object is injected into the dataset, it will be assigned to a cluster which ensures the total distance between all points and the center of their corresponding clusters is minimized, and the central data point of each cluster may change. In this project, we should change the format of our dataset and select an encoding scheme to make sure it can be input to the k-medoids algorithm.

3.3.2 Implementation

The dataset has 23 columns. The first column is the labels of mushrooms. The rest of them are the attributes of mushrooms. We should use the columns except first column to do the clustering. In addition, we use the first 8000 rows as the input objects. For each attribute, the original dataset uses one letter as a label of one class. For example, the

odor has nine labels like ‘a’ as almond, ‘l’ as anise and ‘y’ as fishy. The clustering algorithm needs numbers instead of letters to calculate the distance between different objects in the n-dimensional space, so we should transform those letters to digits, and uses the digits as the coordinates of one object. Because this dataset is stored as csv format. We can use the Pandas library to read the csv file and choose one specific column by setting the parameter usecols. For each column, we use the LabelEncoder function in sklearn library to do the transform. It encodes labels with value between 0 and the number of classes. Then each entry of each column can be transformed into digit, and we output each column as a row into a new csv file. The columns of the new file are different objects. During clustering, we need to read each column instead of a row of the new file. Similarly, we transform the format of label of each mushroom, and output them into another csv file, if the mushroom is poison, the label is 1, otherwise, the label is 0.

After transforming the data format, we can start clustering. We choose to calculate distance between each object by using the Euclidean distance. The pairwise_distances function can calculate the distance by this metric and return a matrix which contains all the distance between all the objects. The program firstly initializes an array of k cluster centers randomly. We set the k as 2, because the classes of mushroom is 2. Because we know all the distance, when we insert a new object, we can check this object against the two centers of clusters, then the program will set the object to its nearest center to make sure the total distance D which is the sum of the distance of all the points to their corresponding centers is minimized. Next, the program will update cluster medoids which are the centers of these clusters. Repeating above operations until all the objects are inserted. Finally, this program will return the two medoids of two clustering, and the clustering number of each object.

Furthermore, we should draw those objects in a plot, and calculate the accuracy of this clustering algorithm. Because there are 23 attributes, we can’t draw those points in a plot with 23 dimensions. We decide to reduce the dimension of each object by using the PCA. The sklearn library has the PCA function. After reducing dimension, we can draw each object in a 2-dimensional plot. To calculate the accuracy, we need to compare the label of each mushroom to the number of its cluster, if a match is found, we will record this object.

3.3.3 Evaluation

There are two different results of clustering which are determined by the first two centers of the two clusters the program produces randomly. The first one is correct. There are two different correct results, one is the true correct and the other is the false correct. The true correct is the poison mushrooms are really put into the cluster
containing poison mushrooms. The false one is adverse. However, the plots of these two cases are same. Because we know the number of poison mushrooms is larger than the non-poison, we can know the plot is the true correct one or the other. The accuracy is 69% for the true positive set and 31% for the false positive set. The
second case is incorrect. The first two centers of two clusters are totally wrong. We show the figures of the correct (figure 7) and incorrect (figure 8) results below:

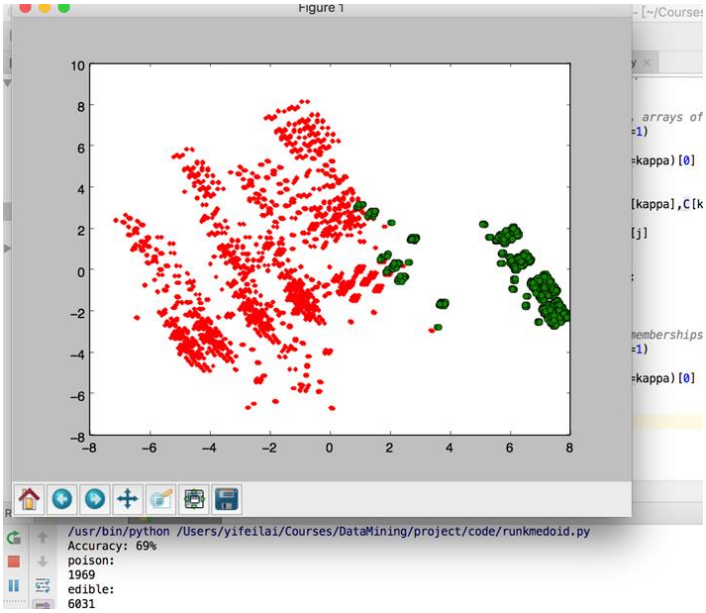


fig. 7, Output clusters of the result clustering 1 of 2. As shown, there are two clear groups which predict the difference between mushrooms and poisonous mushrooms with accuracy of close to 70%.

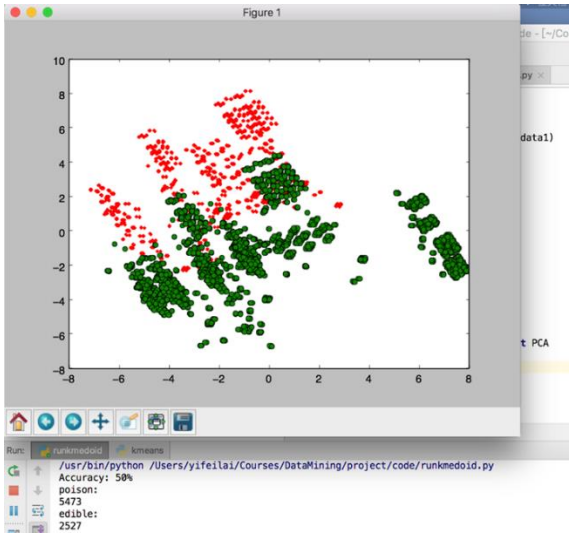


fig. 8, Output clusters of the result clustering 2 of 2. As shown, there are two overlapping groups which are difficult to separate and the accuracy is 50%, which is no better than a random guess.

4. RELATED WORKS

There are several related works to our data mining classifiers from which we referenced and some of which also served as guidelines to our implementations. These works have been cited and fully accredited in the references section. Some of these are were also discussed in detail in our project midterm report.

5. CONCLUSION

Throughout this project, we have learned the effective use of three classifiers with which to perform supervised learning tasks on the UCI Mushroom Dataset. Upon training and testing, we have determined that the most effective model is Random Forest for its ability to capitalize on the relatively low complexity of the training set as well as a few key features which are closely tied with the output labels. The second most effective model is the Naïve Bayes model with Multinomial extension for its high efficiency and ability to work with discrete, categorical data. Finally, we have concluded that the K-Medoids Clustering algorithm is not as effective for working with this type of dataset, since there is no guarantee that the biggest difference between mushrooms is its toxicity, allowing for a clean grouping of poisonous and edible sets (perhaps poisonous and non-poisonous mushrooms are more similar in their toxicity than their difference in cap shape for example). Hence, in order for clustering to work perfectly, feature preselection would be required at the risk of losing information that could be helpful in determining the output labels.

References & Works Cited

Brownlee, Jason. “Naive Bayes for Machine Learning.” Machine Learning Mastery, 11 Apr. 2016, machinelearningmastery.com/naive-bayes-for-machine-learning/.

The Bernoulli model.” Stanford NLP, Cambridge University Press, 17 Apr. 2009, nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html.

Vryniotis, Vasilis. “Machine Learning Tutorial: The Naive Bayes Text Classifier.” DatumBox, 13 Oct. 2013, blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/.

Taheri, Sona, et al. “Improving Naive Bayes Classifier Using Conditional Probabilities.” 9-Th Australasian Data Mining Conference, http://crpit.com/confpapers/CRPITV121Taheri.pdf.

Kohavi, Ron. “Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid.” KDD-96, 1996, http://www.aaai.org/Papers/KDD/1996/KDD96-033.pdf.

Brownlee, Jason. “How to Implement Random Forest From Scratch in Python.” Machine Learning Mastery, 2016, machinelearningmastery.com/implement-random-forest-scratch-python/.

Polamuri, Saimadhu. “HOW THE RANDOM FOREST ALGORITHM WORKS.” Dataaspirant, 22 May 2017, dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/.

Ali, Jehad, et al. “Random Forests and Decision Trees.” IJCSI International Journal of Computer Sciences, vol. 9, no. 5, ser. 3, Sept. 2012. 3, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.3863&rep=rep1&type=pdf.

Breiman, Leo. "Random Forests." Machine Learning 45.1(2001):5-32. http://math.univ-toulouse.fr/~agarivie/Telecom/apprentissage/articles/randomforest2001.pdf.

Bauckhage C. Numpy/scipy Recipes for Data Science: k-Medoids Clustering[R]. Technical Report, University of Bonn, 2015.

CSDN.NET Blog tutorial: http://blog.csdn.net/eastmount/article/details/53285192

Guha, Sudipto, R. Rastogi, and K. Shim. "CURE: an efficient clustering algorithm for large databases." Information Systems 26.1(2001):35-58. https://www-users.cs.umn.edu/~hanxx023/dmclass/cure.pdf.

Karypis, George, et al. “CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic
Modeling.” IEEE Computer, http://pdfs.semanticscholar.org/19a5/95de8a54c89270048512b2b5404e60119ce8.pdf.