University of New Brunswick

Machine Learning and Data Mining
(CS 6735)

Programming Project

Professor:
Huajie Zhang
Computer Science

Author:
Saeed Kazemi
(3713280)

April 20, 2021

Contents

Introduction

This project has focused on comparing and discussing five various machine learning algorithms based on the experimental results on the five different datasets. Furthermore, this research will be implemented in Python, and the source codes are available on the GitHub repository.

The rest of this assignment is organized as follows. Section 1 provides some information about the dataset along with techniques that are used for pre-processing. Then, in Section 2, the machine learning algorithms will be reviewed. Finally, the results are discussed in Section 3.

The Questions

Conduct an experimental study on the following machine learning algorithms: (1) ID3; (2) Adaboost on ID3; (3) Random Forest; (4) Naïve Bayes; (5) K-nearest neighbors (kNN).

1. Implement the five algorithms using Java or Python.
2. Evaluate your implementation on the datasets in data.zip (downloadable from course website) using 10 times 5-fold cross-validation, and report the average accuracy and standard deviation. All datasets are for UCI machine learning repository. You can check the detailed descriptions from this link.
   - For breast cancer data see this link.
   - For car data see this link.
   - For ecoli data see this link.
   - For letter recognition data see this link.
   - For mushroom data see this link.

   For each data set, there is a target variable, the one your model predicts. The following are the target variable for each data set.
   - Mushroom: first column (e, p)
   - Letter: first column (A, B, ...)
   - Ecoli: last column (cp, im, ..)
   - Car: last column (acc, uacc, ..)
   - Breast-cancer: last column (2, 4)
3. Compare and discuss your algorithms (implementations) based on your experimental results.

   Submission:

   (a) Hand in a report of your experimental study via Desire2Learning, including:
      i. Description of the learning algorithms you implement.
      ii. Description of the datasets you use (number of examples, number of attribute, number of classes, type of attributes, etc.).
      iii. Technical details of your implementation: pre-processing of data sets (discretization, etc.), parameter setting, etc.
      iv. Design of your programming implementation (data structures, overall program structure).
      v. Report and analysis of your experimental results.

   (b) Submit your code via Desire2Learning.

1 Datasets and pre-processing

In this section, we will review the datasets as well as the pre-processing techniques. Dataset can be downloaded from the Desire2Learning website. Before describing the dataset, let look at the discretization algorithm.

1.1 Discretization algorithm

Since some of algorithms accept only discreated values, I have used the equal width binning algorithm to discretize the continuous data to categorical version. This method to partition the range of features into k equal-width distances. The interval width (w) is calculated by the below equation.

$$w = \frac{[max(feature) - min(feature)]}{k}$$

Then, the i^{th} interval range can be found by using this equation.

$$[min(feature) + (i-1)w, \quad min(feature) + iw] \quad \text{where} \quad i = 1, 2, 3, \ldots, k$$

Now, we can scan each features values and if it is in the i^{th} interval range, we replace it with i value. As a result, the new feature set has only i different value.

Breast cancer, Ecoli and letter dataset have the continuous attributes, and I have discretized them for using ID3, Random Forests, and Adaboost classifier.

Table 1: The equal width binning function for discretizing the continious data to categorical version.

```
def discr(x, k=5):
    w = (np.max(x) - np.min(x)) / k
    bins = [np.min(x) + (i) * w for i in range(k)]
    return np.digitize(x, bins=bins, right=False)
```

1.2 Categorical to numerical values

Car and Mushroom dataset are two data set that have categorical values. To converting these two datasets to a contentious data, I have replaced the categorical values with a numerical values.

1.3 Breast Cancer Wisconsin (Diagnostic) Dataset

This dataset was included real data about breast cancer. In the CSV file, there are 11 columns. The first column, which is a seven-digit number, indicates the patient ID. Also, the last column is our target. Other columns were real-valued features that are computed from each cell nucleus.

For reading this dataset, we used a Pandas DataFrame. Since some values in the dataset were a question mark, we replaced those with a NaN value. After that, we drop all rows that had NaN values. The final size of the dataset is 682 rows $\times$ 10 columns. Based on the Question, this dataset has two class values, 2 and 4.

1.4 Car Evaluation Dataset

Car Evaluation Database was derived from a simple hierarchical decision model. This dataset has four class values named unacc, acc, good, vgood, and it is located in the final column. Other features are buying, maint, doors, persons, safety and lug_boot. These features have categorical values. The table below shows the attribute information.

Table 2: The attribute information of Car Evaluation dataset.

Column Names	their values
buying	vhigh, high, med, low
maint	vhigh, high, med, low
doors	2, 3, 4, 5more
persons	2, 4, more
lug_boot	small, med, big
safety	low, med, high
Classes	unacc, acc, good, vgood

In addition, there is no missing value in the dataset. Therefore, the final size of this dataset is 6 $\times$ 1728. Furthermore, in order to use some algorithms like kNN, we changed the categorical values to numerical values by means of the replace method in Pandas DataFrame.

1.5 Ecoli Dataset

This data contains protein localization sites. The size of the dataset is 336 × 8, and there is no missing value. Table 3 indicates the attribute information of this dataset. Furthermore, All features have a real value.

Table 3: The attribute information of Ecoli dataset.

#	Attribute Names	description
1	Sequence Name	Accession number for the SWISS-PROT database
2	mcg	McGeoch's method for signal sequence recognition.
3	gvh	von Heijne's method for signal sequence recognition.
4	lip	von Heijne's Signal Peptidase II consensus sequence score.
5	chg	Presence of charge on N-terminus of predicted lipoproteins.
6	aac	score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.
7	alm1	score of the ALOM membrane spanning region prediction program.
8	alm2	score of ALOM program after excluding putative cleavable signal regions from the sequence.

The last column shows the classes of this dataset which takes eight different values.

1.6 Letter Recognition Dataset

The objective of this dataset is to classify each of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts, and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique samples. Each sample was converted into 16 numerical attributes. These values are an integer number between 0 and 15.

Table 4: The attribute information of letter recognition dataset.

#	Attribute description
1	lettr capital letter (26 values from A to Z)
2	x-box horizontal position of box (integer)
3	y-box vertical position of box (integer)
4	width width of box (integer)
5	high height of box (integer)
6	onpix total # on pixels (integer)
7	x-bar mean x of on pixels in box (integer)
8	y-bar mean y of on pixels in box (integer)
9	x2bar mean x variance (integer)
10	y2bar mean y variance (integer)
11	xybar mean x y correlation (integer)
12	x2ybr mean of x * x * y (integer)
13	xy2br mean of x * y * y (integer)
14	x-ege mean edge count left to right (integer)
15	xegvy correlation of x-ege with y (integer)
16	y-ege mean edge count bottom to top (integer)
17	yegvx correlation of y-ege with x (integer)

The size of the dataset is 20000 $\times$ 16. The first column also shows the class of sample that is a capital letter. This dataset has no missing values. In addition, all classes are replaced with a numerical value as a preprocessing step.

1.7 Mushroom Dataset

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible (e) or poisonous (p). There are about 22 categorical features for each sample in the dataset. Some samples have a character "?" which indicates the dataset has some missing value. Also, all categorical values are replaced with numerical values. The size of the dataset is 8124 $\times$ 16.

2 Implemented algorithms

In this section, we will review the algorithms as well as implemented techniques. Furthermore, to implementing k-fold cross-validation, we implemented a classifier class which the Parent's class of other algorithms is. In the classifier class, we have three methods, kfold_split(), accuracy(), and fit(). Table 5 indicates the methods and attributes of this class.

Table 5: The methods and attributes of classifier class.

Class Name: classifiers	
parent: -	Child: KNN, NB, KNN, RF

Attributes	
n_folds:	the number of folds

Methods		
kfold_split(dataset, n_folds)		split the dataset to n_folds
accuracy(y_true, y_pred)		compute the accuracy
accuracy = fit()		train the algorithm
Parameters	dataset:	array-like of shape (n_samples, n_features+target)
	n_folds	int, default=5
	y_true	list-like of shape (n_samples,)
	y_pred	list-like of shape (n_samples,)
Returns	accuracy	accuracy score of each k_fold in percentage

I have implemented the k_fold algorithm as follows:

1. Find the length of each fold by dividing the dataset length into k_fold.
2. Select an index randomly, and then copy the selected index to the new set by means of the pop method in Pandas.
3. Check the length of the new set. If it is less than part (1), repeat part (2), otherwise make another set.

2.1 K-nearest neighbors algorithm (KNN)

For using this algorithm, we need to make an instance of the KNN class. This instance calls the __init__ method to set some attributes for the algorithm. After that, with the fit method, we train the algorithm. The table below shows a piece of code used for making an instance of the KNN class. Moreover, table 7 illustrates the methods and attributes of this class. Furthermore, the k parameter must be an odd number.

Table 6: Making an instance of KNN class.

```
for _ in range(10):
    knn = KNN(n_folds=5, dataset=dataset, k_neighbor=3)
    accuracy = knn.fit()
    acc.append(sum(accuracy) / len(accuracy))
```

Table 7: The methods and attributes of KNN class.

Class Name: KNN	
parent: classifiers	Child: -

Attributes	
n_folds:	the number of folds
dataset:	the train dataset (the last column is the target)
k_neighbor:	the number of folds

Methods		
__init__(dataset, n_folds, k_neighbor)		constructor to initialise the attributes of the class
pred = knn(tr_set, te_set)		compute the distants
find_response(k_neighbor, classes)		find the nearest neighbors
Parameters	dataset:	array-like of shape (n_samples, n_features+target)
	n_folds:	int, default=5
	k_neighbor:	an odd integer, default=3
	tr_set:	array-like of shape (n_samples, n_features+target)
	te_set:	array-like of shape (n_samples, n_features+target)
Returns	pred:	predicted classes

I have implemented the kNN algorithm as follows:

1. For each sample in the test data, calculate the distances between the current sample and all samples in the training set (with knn() method)
2. Sort the distances list in ascending order and find the class of the top k_neighbor (with find_response() method).
3. Pick and return the first and most repeated label from the previous collection as a label of test example (with find_response() method).

2.2 Naive Bayes algorithm

For implementing this algorithm, we wrote the NB class. Also, we consider normal distribution for implementation. Table 8 illustrates the methods and attributes of this class.

Table 8: The methods and attributes of NB class.

Class Name: NB	
parent: classifiers	Child: -

Attributes	
n_folds:	the number of folds
dataset:	the train dataset (the last column is the target)

Methods		
__init__(dataset, n_folds, k_neighbor)		Constructor to initialise the attributes of the class
naivebayes(trainset, testset)		Main loop of class
predict(models, test_row)		Return the prediction value
model_classes(dataset)		Set a Gaussian model for each class
find_pdf(x, mean, stdev)		Calculate probability
Parameters	dataset:	array-like of shape (n_samples, n_features+target)
	n_folds	int, default=5
	k_neighbor	an odd integer, default=3
	tr_set:	array-like of shape (n_samples, n_features+target)
	te_set:	array-like of shape (n_samples, n_features+target)
Returns	pred:	predicted classes

The table below shows a piece of code used for making an instance of the NB class.

Table 9: Making an instance of NB class.

```
for _ in range(10):
    nb = NB(n_folds=5, dataset=dataset)
    accuracy = nb.fit()
    acc.append(sum(accuracy) / len(accuracy))
```

I have implemented the Naive Bayes algorithm as follows:

1. Split training set by class value (model_classes())
2. Find the mean and std of each attribute in each split dataset by their class (model_classes()).
3. Calculate the class probability for each test sample (find_pdf()). Combine probability of each feature (predict()).
4. Compare probability for each class. Return the class label which has max probability (predict()).

2.3 Iterative Dichotomiser 3 algorithm (ID3)

We wrote the ID3 class to implement this algorithm. Table 10 illustrates the methods and attributes of this class. This algorithm calls yourself like a recursive function. Moreover, the dataset must be a categorical data. Therefore, I have used 5-bin discretization for all attributes in Ecoli, Breast Cancer, and Letter datasets.

Table 10: The methods and attributes of ID3 class.

Class Name: ID3	
parent: classifiers	Child: RF

Attributes	
n_folds:	the number of folds
dataset:	the train dataset (the last column is the target)

Methods		
__init__(dataset, n_folds, attr_names)		constructor to initialise the attributes of the class
id3(trainset, testset)		Build the ID3 algorithm
decision_tree(trainset, attr_names)		Build the decision tree
split(trainset, arc, val)		Split the dataset based on an attribute
attr_selection(trainset, attr_names)		Find the best attribute for the split
entropy(data)		Compute the entropy
test(test_set, tree, attr_names)		Calculate the prediction accuracy
predict(models, test_row)		Predict an unseen instance
Parameters	dataset:	array-like of shape (n_samples, n_features+target)
	n_folds:	int, default=5
	names:	a list of feature names
Returns	pred:	predicted classes

To use this algorithm, you need to make an instance first. The table below shows a piece of code used for doing it.

Table 11: Making an instance of ID3 class.

```
for _ in range(10):
    id3 = ID3(n_folds=5, dataset=dataset, names=names)
    accuracy = id3.fit()
    acc.append(sum(accuracy) / len(accuracy))
```

I have implemented the ID3 algorithm as follows:

1. Create a node. If all samples belong to the same class, the node is marked as a leaf node and returns the class.

2. For each feature in the data set, calculate the Information Gain. Then split feature that has the maximum information gain.
3. For each value in the split attribute, extend the corresponding branch, and divide samples according to the feature value.
4. Use the same procedure, recursion from the top down until one of the following three conditions is met and the recursion stops. If there are still samples belonging to different categories in the leaf node, the category containing the most samples is selected as the classification of the leaf node.
    - All samples belong to the same class.
    - All samples in the training set were classified.
    - All features were executed once as split features.

2.4 Random Forests algorithm

We used the ID3 class to implement the random forests algorithm. Table 12 illustrates the methods and attributes of this class. To converting continuous values to discreated version, I have used 5-bin discretization for all attributes in Ecoli, Breast Cancer, and Letter datasets.

Table 12: The methods and attributes of Random Forests class.

Class Name: RF	
parent: ID3	Child: -

Attributes	
n_folds:	the number of folds
num_trees:	the number of trees
dataset:	the train dataset (the last column is the target)

Methods		
__init__(data, n_folds, attrNames, numTrees)		constructor to initialise the attributes of the class
pred = RF(trainset, testset)		Build the Random forest algorithm
voting(prediction)		Find the most common result from trees
bootstrapping(data)		Make a new dataset randomly from data
Parameters	dataset:	array-like of shape (n_samples, n_features+target)
	n_folds:	int, default=5
	numTrees:	int, default=5
	attrNames:	a list of feature names
Returns	pred:	predicted classes

I have implemented the Random Forests algorithm as follows:

1. For Each tree do:
   (a) Create a bagging dataset.
   (b) Train a ID3 algorithm based on the new dataset.
   (c) Test the tree with the test set.
2. Vote between the results of algorithms to find the final result.

The table below shows a piece of code that is used for making an instance.

Table 13: Making an instance of Random Forests class.

```
for _ in range(10):
    rf = RF(n_folds=5, dataset=dataset, names=names)
    accuracy = rf.fit()
    acc.append(sum(accuracy) / len(accuracy))
```

2.5 Adaboost algorithm

We used the binary classification (stump) to implement the Adaboost algorithm. Table 14 illustrates the methods and attributes of this class. Moreover, the implemented algorithm could do binary classification. Also, the dataset must be a categorical data. So, I have used 5-bin discretization for all attributes in Ecoli, Breast Cancer, and Letter datasets.

Table 14: The methods and attributes of Adaboost class.

Class Name: AB	
parent: classifiers	Child: -

Attributes	
n_folds:	the number of folds
n_clfs:	the number of weak classifiers
dataset:	the train dataset (the last column is the target)

Methods		
__init__(data, n_folds, n_clfs)		constructor to initialise the attributes of the class
pred = AB(trainset, testset)		Build the Adaboost algorithm
ABoost(X, y)		Find the weak lassifiers
predict(data)		Predict the classes
Parameters	dataset:	array-like of shape (n_samples, n_features+target)
	n_folds:	int, default=5
	n_clfs:	int, default=5
Returns	pred:	predicted classes

We used this Adaboost algorithm flow for this project:

1. Initially set uniform example weights.
2. For Each weak classifier do:
   (a) Greedy search to find best threshold and feature.
   (b) Find the lowest error as sum of weights of misclassified samples
   (c) Store the best configuration that has lowest error as a first weak classifier
   (d) Calculate predictions and update weights

(e) Normalize to weights

The table below shows a piece of code that is used for making an instance.

Table 15: Making an instance of Adaboost class.

```
for _ in range(10):
    adaboost = AB(n_folds=5, dataset=dataset, n_clf=5)
    accuracy = adaboost.fit()
    acc.append(sum(accuracy) / len(accuracy))
```

3 Results and Discussion

In this section, we reviewed the result of implemented algorithms on datasets.

3.1 K-nearest neighbors algorithm (KNN)

This algorithm is very simple and easy to implement. It has only one hyper-parameter (k_neighbor), and we considered it as 3 for all datasets. The algorithm got significantly slower for the letter and mushroom dataset because the volume of data increased. Furthermore, this algorithm takes the numerical values and since Car and Mushroom dataset have the categorical attributes, I have converted them to numerical version by the replace method in Pandas library.

Table 16: The accuracy of KNN on cancer dataset.

	Accuracy KNN on cancer dataset
The 1st run	97.65
The 2nd run	97.21
The 3rd run	97.21
The 4th run	97.06
The 5th run	97.65
The 6th run	97.35
The 7th run	97.50
The 8th run	97.06
The 9th run	96.62
The 10th run	97.06
Average accuracy:	97.24
Standard deviation:	0.30

Table 17: The accuracy of KNN on cars dataset.

	Accuracy KNN on cars dataset
The 1st run	91.77
The 2nd run	92.29
The 3rd run	92.06
The 4th run	92.00
The 5th run	92.17
The 6th run	91.19
The 7th run	91.01
The 8th run	92.06
The 9th run	92.46
The 10th run	91.42
Average accuracy:	91.84
Standard deviation:	0.46

Table 18: The accuracy of KNN on ecol dataset.

	Accuracy KNN on ecol dataset
The 1st run	82.99
The 2nd run	83.58
The 3rd run	85.37
The 4th run	83.88
The 5th run	84.78
The 6th run	85.07
The 7th run	84.18
The 8th run	86.87
The 9th run	84.18
The 10th run	84.18
Average accuracy:	84.51
Standard deviation:	1.03

Table 19: The accuracy of KNN on letter dataset.

	Accuracy KNN on letter dataset
The 1st run	95.44
The 2nd run	95.61
The 3rd run	95.70
The 4th run	95.39
The 5th run	95.54
The 6th run	95.46
The 7th run	95.51
The 8th run	95.59
The 9th run	95.42
The 10th run	95.56
Average accuracy:	95.52
Standard deviation:	0.09

Table 20: The accuracy of KNN on mushroom dataset.

	Accuracy KNN on mushroom dataset
The 1st run	99.88
The 2nd run	99.88
The 3rd run	99.89
The 4th run	99.88
The 5th run	99.86
The 6th run	99.91
The 7th run	99.86
The 8th run	99.89
The 9th run	99.86
The 10th run	99.93
Average accuracy:	99.88
Standard deviation:	0.02

3.2 Naive Bayes algorithm

It is a fast algorithm. In this algorithm, we must assume that the features are independent with a Gaussian distribution. As you can see, the algorithm had a great result on cancer and mushroom dataset, while for other datasets, the results were not good. The reason could be neither features are independent nor their distribution

is Gaussian. This algorithm does not have any hyperparameters. Furthermore, this algorithm takes the numerical values and since Car and Mushroom dataset have the categorical attributes, I have converted them to numerical version by the replace method in Pandas library.

Table 21: The accuracy of Naive Bayes on cancer dataset.

	Accuracy Naive Bayes on cancer dataset
The 1st run	96.32
The 2nd run	96.03
The 3rd run	96.47
The 4th run	95.88
The 5th run	96.18
The 6th run	96.47
The 7th run	96.47
The 8th run	96.32
The 9th run	96.18
The 10th run	96.32
Average accuracy:	96.26
Standard deviation:	0.19

Table 22: The accuracy of Naive Bayes on cars dataset.

	Accuracy Naive Bayes on cars dataset
The 1st run	82.09
The 2nd run	81.80
The 3rd run	82.20
The 4th run	82.38
The 5th run	82.26
The 6th run	82.14
The 7th run	82.49
The 8th run	81.39
The 9th run	81.91
The 10th run	82.61
Average accuracy:	82.13
Standard deviation:	0.34

Table 23: The accuracy of Naive Bayes on ecol dataset.

	Accuracy Naive Bayes on ecol dataset
The 1st run	57.61
The 2nd run	61.49
The 3rd run	51.64
The 4th run	57.01
The 5th run	58.51
The 6th run	54.33
The 7th run	67.46
The 8th run	61.49
The 9th run	57.91
The 10th run	49.25
Average accuracy:	57.67
Standard deviation:	4.95

Table 24: The accuracy of Naive Bayes on letter dataset.

	Accuracy Naive Bayes on letter dataset
The 1st run	64.99
The 2nd run	65.01
The 3rd run	64.87
The 4th run	64.92
The 5th run	64.83
The 6th run	64.94
The 7th run	64.95
The 8th run	64.89
The 9th run	64.90
The 10th run	64.80
Average accuracy:	64.91
Standard deviation:	0.06

Table 25: The accuracy of Naive Bayes on mushroom dataset.

	Accuracy Naive Bayes on mushroom dataset
The 1st run	91.86
The 2nd run	91.91
The 3rd run	91.65
The 4th run	91.79
The 5th run	91.72
The 6th run	91.84
The 7th run	91.86
The 8th run	91.88
The 9th run	91.44
The 10th run	91.76
Average accuracy:	91.77
Standard deviation:	0.14

3.3 Iterative Dichotomiser 3 algorithm (ID3)

The speed of building decision tree is relatively fast, the algorithm is simple, and the generated rules are easy to understand. Also it could not support continuous values. Therefore, all datasets were discretized before using this algorithm (see session 1.1 for more detail).

Table 26: The accuracy of ID3 on mushroom dataset.

	Accuracy ID3 on mushroom dataset
The 1st run	100.0
The 2nd run	100.0
The 3rd run	100.0
The 4th run	100.0
The 5th run	100.0
The 6th run	100.0
The 7th run	100.0
The 8th run	100.0
The 9th run	100.0
The 10th run	100.0
Average accuracy:	100.0
Standard deviation:	0.0

Table 27: The accuracy of ID3 on cancer dataset.

	Accuracy ID3 on cancer dataset
The 1st run	93.68
The 2nd run	93.09
The 3rd run	94.71
The 4th run	94.41
The 5th run	95.15
The 6th run	93.53
The 7th run	94.85
The 8th run	93.97
The 9th run	93.68
The 10th run	94.26
Average accuracy:	94.13
Standard deviation:	0.62

Table 28: The accuracy of ID3 on cars dataset.

	Accuracy ID3 on cars dataset
The 1st run	90.61
The 2nd run	90.09
The 3rd run	90.32
The 4th run	91.13
The 5th run	90.38
The 6th run	91.25
The 7th run	90.32
The 8th run	90.38
The 9th run	89.80
The 10th run	90.49
Average accuracy:	90.48
Standard deviation:	0.41

Table 29: The accuracy of ID3 on ecol dataset.

	Accuracy ID3 on ecol dataset
The 1st run	62.39
The 2nd run	66.57
The 3rd run	65.97
The 4th run	65.37
The 5th run	62.99
The 6th run	63.28
The 7th run	65.37
The 8th run	64.78
The 9th run	66.27
The 10th run	65.97
Average accuracy:	64.90
Standard deviation:	1.41

Table 30: The accuracy of ID3 on letter dataset.

	Accuracy ID3 on letter dataset
The 1st run	75.51
The 2nd run	75.26
The 3rd run	75.56
The 4th run	75.64
The 5th run	75.90
The 6th run	75.54
The 7th run	75.43
The 8th run	75.38
The 9th run	75.54
The 10th run	75.50
Average accuracy:	75.53
Standard deviation:	0.16

3.4 Random Forest algorithm

Random Forest is less impacted by noise, because we train several decision trees. For this project, the number of trees was set to 5. Due to the fact that this algorithm is using ID3, we need to discretize data.

Table 31: The accuracy of Random Forest on cancer dataset.

	Accuracy Random Forest on cancer dataset
The 1st run	96.47
The 2nd run	95.44
The 3rd run	94.85
The 4th run	95.29
The 5th run	95.15
The 6th run	94.41
The 7th run	94.85
The 8th run	95.74
The 9th run	95.44
The 10th run	95.00
Average accuracy:	95.26
Standard deviation:	0.54

Table 32: The accuracy of Random Forest on cars dataset.

	Accuracy Random Forest on cars dataset
The 1st run	90.20
The 2nd run	90.09
The 3rd run	89.04
The 4th run	90.61
The 5th run	91.19
The 6th run	90.49
The 7th run	90.55
The 8th run	89.74
The 9th run	90.03
The 10th run	90.72
Average accuracy:	90.27
Standard deviation:	0.56

Table 33: The accuracy of Random Forest on ecol dataset.

	Accuracy Random Forest on ecol dataset
The 1st run	67.16
The 2nd run	67.46
The 3rd run	70.45
The 4th run	68.06
The 5th run	64.78
The 6th run	67.76
The 7th run	68.66
The 8th run	65.37
The 9th run	64.18
The 10th run	66.87
Average accuracy:	67.07
Standard deviation:	1.79

Table 34: The accuracy of Random Forest on letter dataset.

	Accuracy Random Forest on letter dataset
The 1st run	76.29
The 2nd run	76.21
The 3rd run	76.21
The 4th run	76.48
The 5th run	76.64
The 6th run	76.22
The 7th run	76.32
The 8th run	76.36
The 9th run	76.19
The 10th run	76.36
Average accuracy:	76.33
Standard deviation:	0.14

Table 35: The accuracy of Random Forest on mushroom dataset.

	Accuracy Random Forest on mushroom dataset
The 1st run	100.0
The 2nd run	100.0
The 3rd run	100.0
The 4th run	100.0
The 5th run	100.0
The 6th run	100.0
The 7th run	100.0
The 8th run	100.0
The 9th run	100.0
The 10th run	100.0
Average accuracy:	100.0
Standard deviation:	0.0

3.5 Adaboost algorithm

Adaboost is easy to implement. Also, it could not overtrain since each node is divided into two branches. It can be implemented with other classifications. I have implemented it with a binary stump classifier. As a result, I could classify only cancer and mushroom datasets because these are only two classes. For this project, the number of weak classifiers considered 5. Also this algorithm could only support symbolizing values. Therefore, all datasets were discretized before using this algorithm (see session 1.1 for more detail).

Table 36: The accuracy of Adaboost on cancer dataset.

	Accuracy Adaboost on cancer dataset
The 1st run	95.15
The 2nd run	95.00
The 3rd run	95.15
The 4th run	95.88
The 5th run	95.74
The 6th run	94.12
The 7th run	95.44
The 8th run	94.56
The 9th run	95.15
The 10th run	94.26
Average accuracy:	95.04
Standard deviation:	0.55

Table 37: The accuracy of Adaboost on mushroom dataset.

	Accuracy Adaboost on mushroom dataset
The 1st run	98.87
The 2nd run	98.87
The 3rd run	98.87
The 4th run	98.87
The 5th run	98.87
The 6th run	98.87
The 7th run	98.87
The 8th run	98.87
The 9th run	98.87
The 10th run	98.87
Average accuracy:	98.87
Standard deviation:	0.00