Carbon

Jiayu Wang

1. Load data

```
#load module data
import data
#create a dataset object and read data from file sample.txt
sample = data.DataSet()
#To read nominal data, you have to add argument 'nominal',
default is 'numeric'
#Read data from 'sample.txt'
sample.read('sample.txt', 'numeric')
#view the data
sample.x
array([[  5.,   1.,   1., ...,   3.,   1.,   1.],
       [  5.,   4.,   4., ...,   3.,   2.,   1.],
       [  3.,   1.,   1., ...,   3.,   1.,   1.],
       ...,
       [  5.,  10.,  10., ...,   8.,  10.,   2.],
       [  4.,   8.,   6., ...,  10.,   6.,   1.],
       [  4.,   8.,   8., ...,  10.,   4.,   1.]])
#view class labels
sample.y[:10]
['2', '2', '2', '2', '2', '4', '2', '2', '2', '2']
#view dataset dimension
sample.dim()
(699, 9)
#view features
sample.label
['Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of
Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size',
'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses']
#view subject ids
sample.key[:5]
['1000025', '1002945', '1015425', '1016277', '1017023']
#create train dataset and test dataset using 1:10 hold out
train,test = data.holdOut(sample,0.1)
```

2. kNN

This is the k nearest neighbor algorithm.

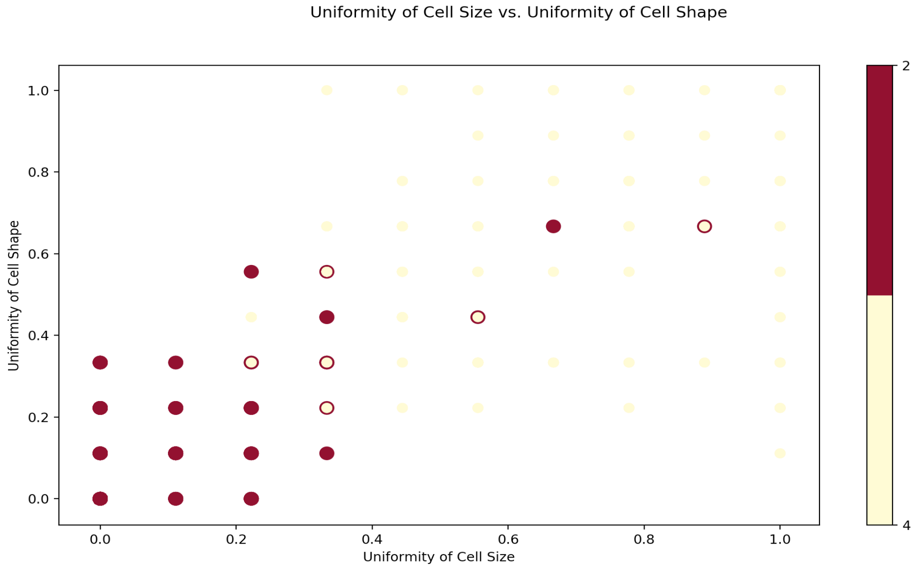
The algorithm only works for: numerical data and nominal class

Parameters:

name	type	default	values
k	int	4	1 to inf
distance	str	'euclidean'	'euclidean' , 'correlation'

```
#load kNN module
from supervised import kNN
#create a classifier
clf = kNN.build()
```

```
#train the classifier with train data and k=4
clf.train(train,4,dist='euclidean')
#to view the model by plot 2 features with class label
clf.view('Uniformity of Cell Size','Uniformity of Cell Shape')
```



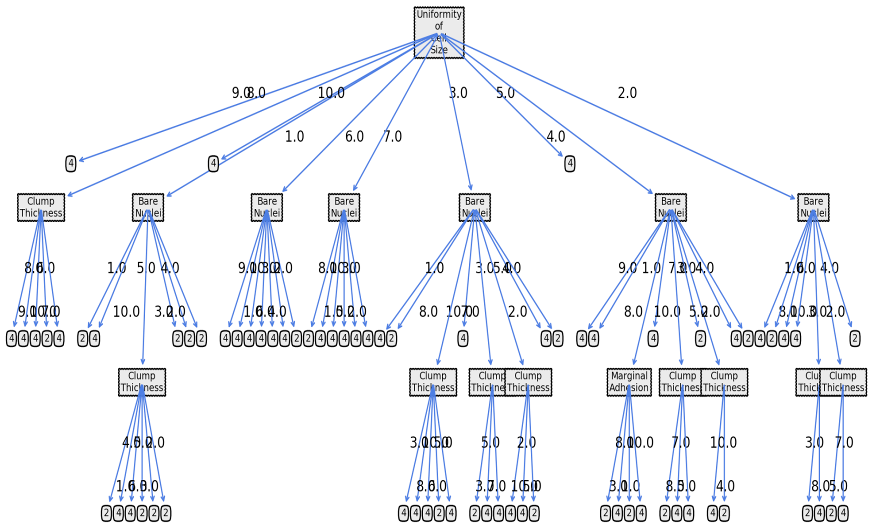

```
#to classify a new subject, return class label and prediction
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
('4', 0.0)
#test the classifier with test data
#and save the predicted labels in result
result = clf.test(test)
the total error rate is: 0.159420
result
(['2', '4', '2', '4', '2', '4', '2', '2', '2', '2', '2', '2',
'2', '2', '4', '2', '2', '2', '4', '2', '4', '4', '2', '2', '2',
'2', '2', '2', '2', '2', '2', '2', '4', '2', '2', '2', '4', '2',
'4', '4', '4', '4', '4', '4', '4', '2', '4', '2', '2', '4', '4',
'2', '4', '4', '4', '4', '4', '4', '4', '2', '2', '2', '4', '2',
'2', '4', '2', '4', '4'], [1.0, 0.0, 1.0, 0.25, 1.0, 0.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 0.75, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0,
0.0, 0.25, 1.0, 0.75, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.25, 0.25, 0.25, 0.0, 0.25,
0.0, 1.0, 0.0, 1.0, 1.0, 0.25, 0.25, 0.75, 0.25, 0.0, 0.0, 0.0,
0.0, 0.25, 0.0, 0.75, 0.5, 1.0, 0.0, 0.75, 1.0, 0.0, 1.0, 0.0,
0.0])
#to save the classifier in folder models/ as model.knn
clf.save('model')
#to load saved classifier
clf2 = kNN.load('model')
```

3. ID3

ID3 is an algorithm that generate classification tree based on information gains.
The algorithm only works for: nominal data and nominal class

```
#convert continuous data to nominal data
train.num2nom()
```

```
test.num2nom()
train.type
'nominal'
#load ID3 module
from supervised import ID3
#create a classifier
clf = ID3.build()
#train the classifier with train data to build the tree
clf.train(train)
#to view the model
clf.view()
```




```
#to classify a new subject
clf.classify(['5.0', '4.0', '4.0', '5.0', '7.0', '10.0', '3.0',
'2.0', '1.0'])
'4'
#test the classifier with test data
#and save the predicted labels in result
result = clf.test(test)
the total error rate is: 0.115942
#to save the classifier in folder models/ as model.id3
clf.save('model')
#to load saved classifier
clf3 = ID3.load('model')
```

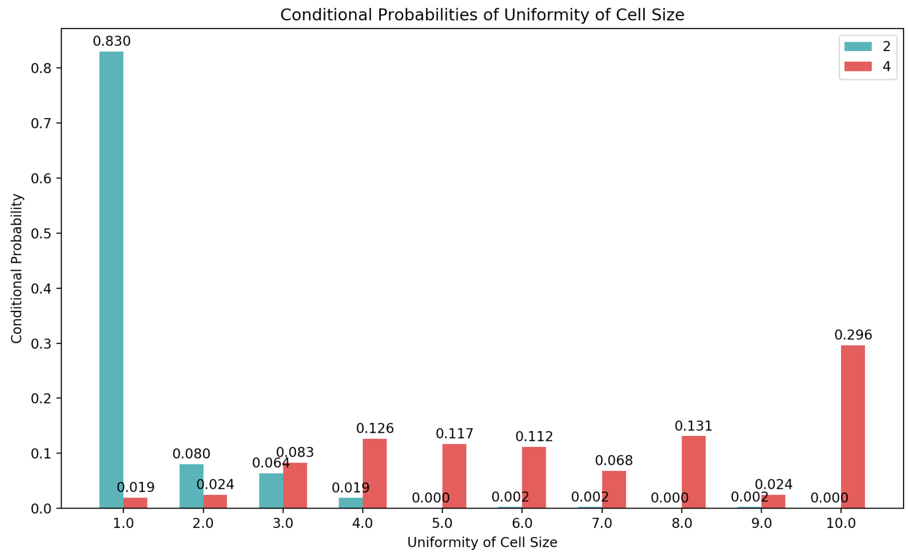
4. NB

This is the Naïve Bayes algorithm.

The algorithm only works for: nominal data and nominal class

```
#load NB module
from supervised import NB
#create a classifier
clf = NB.build()
#train the classifier with train data
clf.train(train)
#to view the model
```

```
#we can view the conditional prob of a certain feature
clf.view('Uniformity of Cell Size')
```



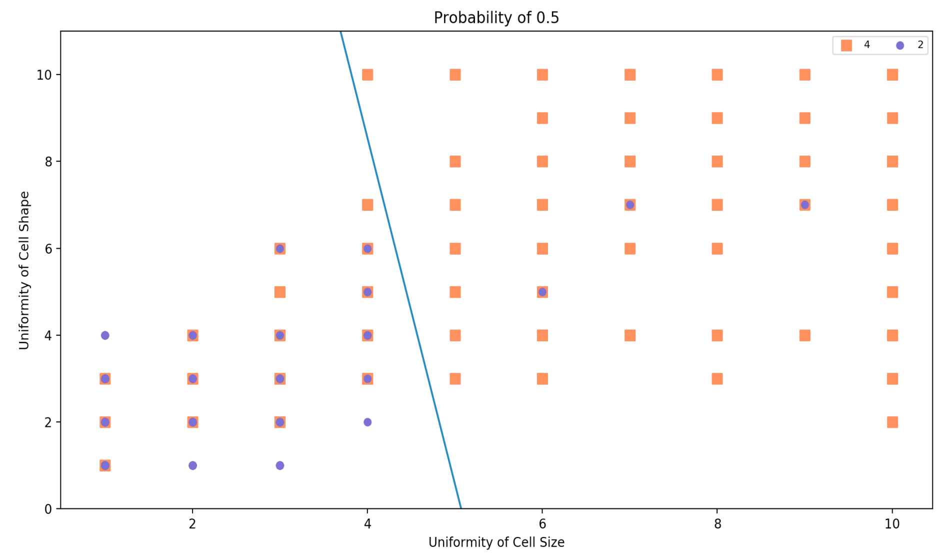

```
#to classify a new subject,
#the first returned value is the class,
#the second is the prediction
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
['4', -0.39598636415362054]
#test the classifier with test data
#and save the predicted labels in result
result = clf.test(test)
the total error rate is: 0.043478
#to save the classifier in folder models/ as model.nb
clf.save('model')
#to load saved classifier
clf4 = NB.load('model')
```

5. logReg

This is the logistic regression.

The algorithm only works for: nominal data and nominal bi-class

```
#load logReg module
from supervised import logReg
#create a classifier
clf = logReg.build()
#train the classifier with train data
clf.train(train)
#to view the model
#we can view the model with 2 features over train dataset
#The line in the middle is the possibility of 0.5
clf.view('Uniformity of Cell Size','Uniformity of Cell
Shape',train)
```




```
#to classify a new subject
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
['4', 0.9953286647482673]
#test the classifier with test data
#and save the predicted labels and values in result
result = clf.test(test)
the total error rate is: 0.086957
#to save the classifier in folder models/ as model.logreg
clf.save('model')
#to load saved classifier
clf5 = logReg.load('model')
```

6. SVM

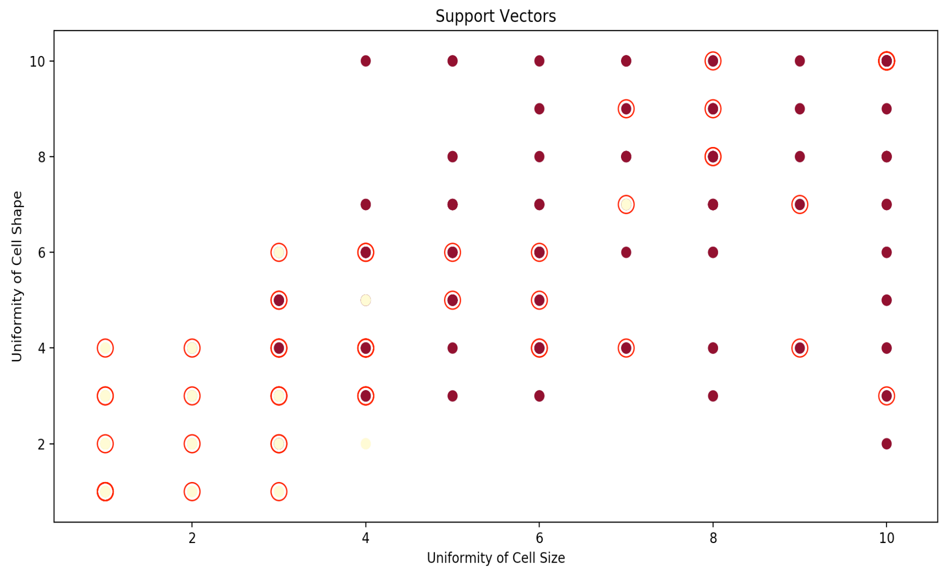
This is the support vector machine.

The algorithm only works for: nominal data and nominal bi-class

Parameters:

- C: float
- toler: float
- maxIter: +int
- kTup:('lin',0), ('rbf',1)

```
#load SVM module
from supervised import SVM
#create a classifier
clf = SVM.build()
#train the classifier with train data
#Here we set C to 100, tolerance to 0.001 and max iteration
number to 400. And use linear kernel.
clf.train(train,C=100,toler=0.001,maxIter=100,kTup=('lin',0))
#to view the model
#we can view the model with 2 features over train dataset
#circled points are support vectors
clf.view('Uniformity of Cell Size','Uniformity of Cell
Shape',train)
```




```
#to classify a new subject
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
'4'
#test the classifier with test data
#and save the predicted labels in result
result = clf.test(test)
the total error rate is: 0.043478
#to save the classifier in folder models/ as model.svm
clf.save('model')
#to load saved classifier
clf6 = SVM.load('model')
```

7. AdaBoost

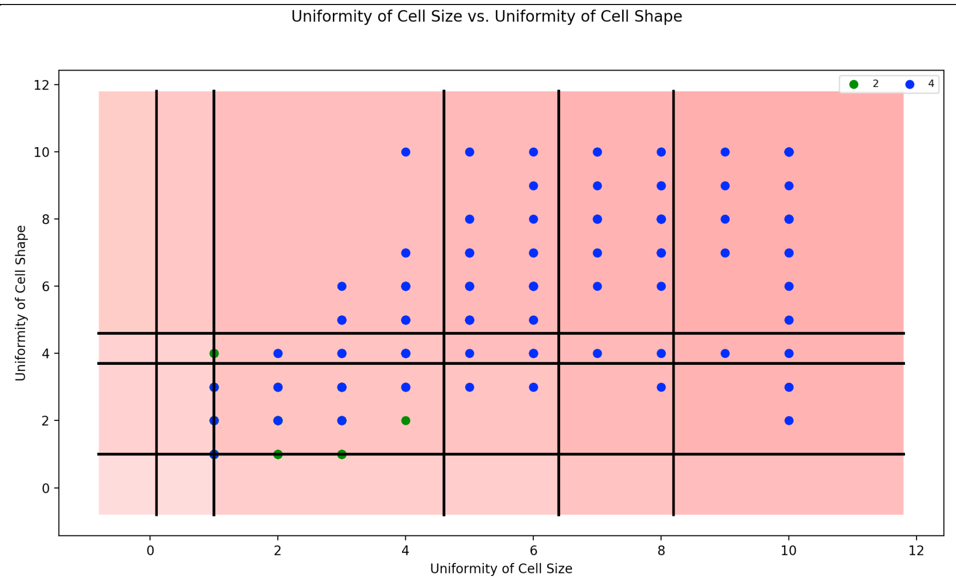
This is the Adaptive Boosting.

The algorithm only works for: nominal data and nominal bi-class

Parameters:

numIt: +int

```
#load Ada module
from supervised import Ada
#create a classifier
clf = Ada.build()
#train the classifier with train data
#maximum number of iteration is 50
clf.train(train, numIt=50)
#to view the model
#we can view the model with 2 features over train dataset
#circled points are support vectors
clf.view('Uniformity of Cell Size','Uniformity of Cell
Shape',train)
```




```
#to classify a new subject
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
'4'
#test the classifier with test data
#and save the predicted labels in result
result = clf.test(test)
the total error rate is: 0.130435
#to save the classifier in folder models/ as model.ada
clf.save('model')
#to load saved classifier
clf7 = Ada.load('model')
```

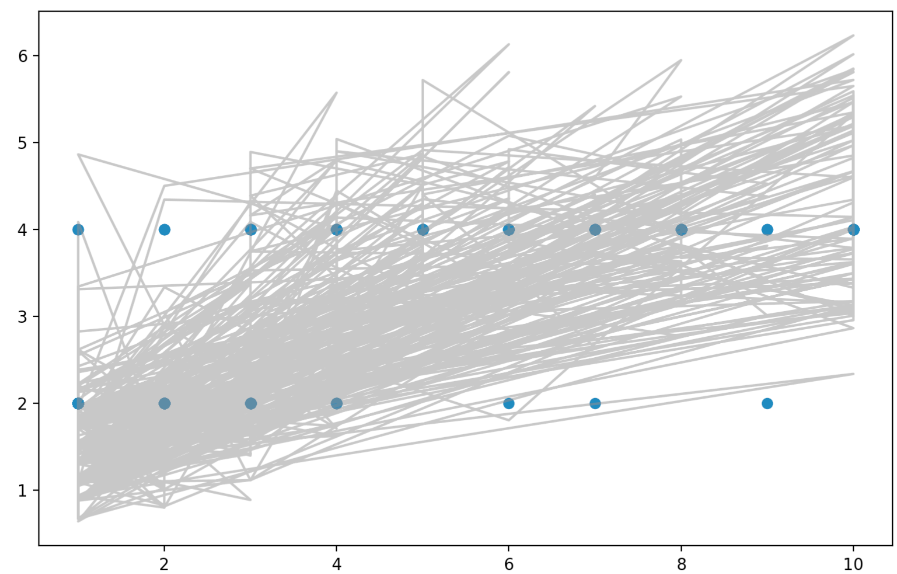
8. Regression

There are some regression algorithms based on linear regression. We have 'linear' for regular linear regression, 'lwlr' for locally weighted linear regression.
The algorithm only works for: numerical data and numerical classes
Parameters:

- method: 'linear', 'lwlr', 'ridge', 'lasso'
- k: float
- lam: float(0-1)
- eps: float
- numIt: int(1-inf)

```
#load reg module
from supervised import reg
#create a classifier
clf =reg.build()
#train the classifier with train data, and choose a method
#1.linear regression
clf.train(train,method='linear')
0.869617690359
#2.locally weighted linear regression
clf.train(train,method='lwlr',k=10)
#3.ridge linear regression
```

```
clf.train(train,method='ridge',l=100)
#4.lasso linear regression
clf.train(train,method='lasso',eps=0.01,numIt=1000)
#to view the model
#we can view the model with 1 features and class
clf.view('Uniformity of Cell Size',train)
```



```
#to classify a new subject
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
'4'
#test the classifier with test data
#save the predicted labels in result
result = clf.test(test)
the total error rate is: 0.144928
#to save the classifier in folder models/ as model.reg
clf.save('model')
#to load saved classifier
clf8 = reg.load('model')
```

9. CART

Regression binary tree building algorithm: CART.

The algorithm works for: numerical data and nominal and numeric classes

Parameters:

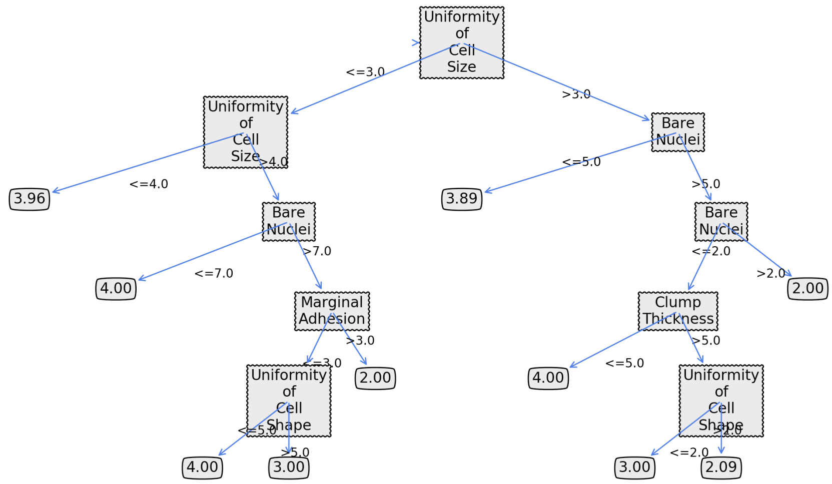
model = True,False

tolS = +int

tolN = +int

```
#load CART module
from supervised import CART
#create a classifier
clf = CART.build()
#train the classifier with train data, minimal step 1, minimal
leaf size 4, model tree method
clf.train(train,tolS=1,tolN=4,model=False)
```

```
#to view the model as a binary tree
clf.view()
```




```
#to classify a new subject, get the class label and prediction
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
('4', 4.0)
#test the classifier with test data
#save the predicted labels in result
result,values = clf.test(test)
the total error rate is: 0.173913
#to save the classifier in folder models/ as model.cart
clf.save('model')
#to load saved classifier
clf8 = reg.load('model')
```

10. Summary

A summary function to summary the results

Parameters:

- yHat: list of str
- yVal: list of float
- y: list of str
- binary: bool

```
#load summary module
import summary
#get results
result = clf.test(test)
#run summary
summary.summary(result[0],result[1],test.y,binary=True)
**** Confusion Matrix ****
                Predict
        class    2         4
True     2      31         3
         4       2        33

******* pos & neg ********
```

```
condition positive(P):  34
condition negative(N):  35
true positive(TP):  31
true negative(TN):  33
false positive(FP):  2
false negative(FN):  3
sensitivity, recall, hit rate or true positive rate(TPR):
        TPR = TP/P = 0.911765
specificity, or true negative rate(TNR):
        TNR = TN/N = 0.942857
```

```
#fast start
import data
sample = data.DataSet()
sample.read('sample.txt')
train,test = data.holdOut(sample,0.1)
from imp import reload

reload(CART)
clf = CART.build()
clf.train(train, tolS=1,tolN=4,model=False)
clf.classify([5, 4, 4, 5, 7, 10, 3, 2, 1])
clf.view('Uniformity of Cell Size','Uniformity of Cell
Shape',train)
```