Cơ sở trí tuệ nhân tạo:

Đồ Án 3: Máy Học

Nguyễn Sĩ Hùng (1612226) - Đoàn Minh Hiếu (1612198)

Ngày 20 tháng 12 năm 2018

Mục lục

1 Tìm hiểu công cụ Weka

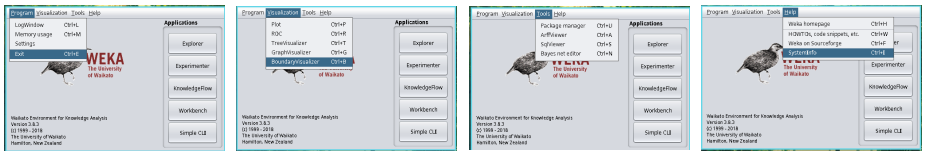
1.1 Weka Interface

Khi sử dụng giao diện đồ hoạ, chương trình sẽ có giao diện như sau:

Các nút có các chức năng khác nhau:

- Explorer: Giao diện đồ hoạ chính dùng để khai thác dữ liệu trong Weka. Phần báo cáo này chủ yếu sẽ tập trung nói về phần này.
- Experimenter: Giao diện đồ hoạ dùng để thực hiện các so sánh giữa các thuật toán học với các dữ liệu test.
- KnowledgeFlow: Giao diện đồ hoạ khá giống với Explorer nhưng hỗ trợ kéo thả.
- Workbench: Giao diện đầy đủ nhất bao gồm nhiều chức năng
- SimpleCLI: Giao diện hỗ trợ thực hiện các câu lệnh trên commandline thay vì dùng giao diện commandline của người dùng.

Trên menu gồm 4 danh mục, mỗi danh mục sẽ có các tuỳ chọn khác nhau như:

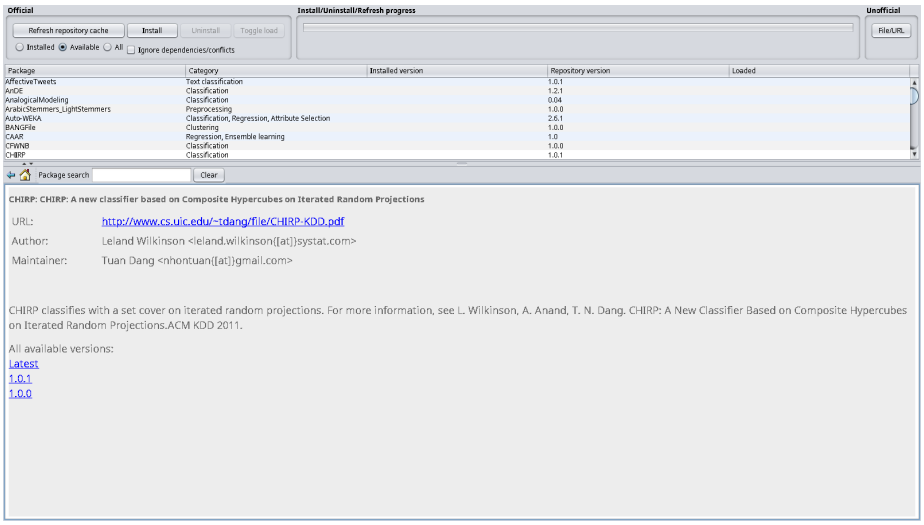


Trong đó, các tuỳ chọn nổi bật là:

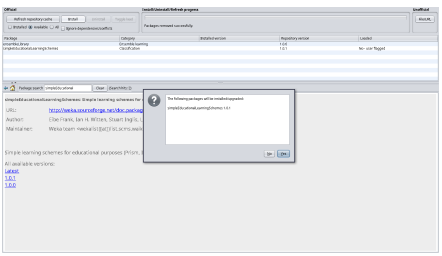
- Program: Log Window dùng để ghi lại log ra stdout hay stderr. Hữu dụng cho hệ điều hành Windows.
- Visualization: Plot dùng để vẽ 2D, TreeVisualize để vẽ cây quyết định.
- Tools: Package manager dùng để quản lý cài đặt các gói dữ liệu cài thêm.
- Help: HOWTOs, code snippets, etc dùng để trợ giúp các ví dụ về cách sử dụng Weka

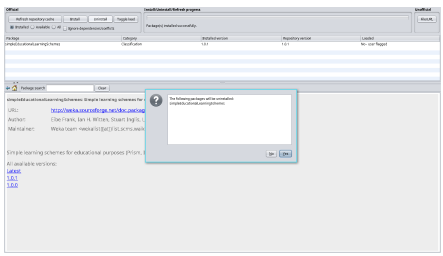
1.2 Package Manager

Dùng để mở tải thêm các gói dữ liệu, chương trình khác (VD: Giải thuật ID3), ta có thể tải thông qua package manager:



Trong đó, để cài 1 gói dữ liệu mới, ta có thể tìm kiếm trong thanh tìm kiếm package search hoặc duyệt qua các gói được liệt kê sẵn. Sau khi đã chọn được gói cần tải, ta bấm nút install để cài đặt. Các gói đã được cài sẽ hiện trong mục installed. Để gỡ các gói ta bấm uninstalled trong mục installed.





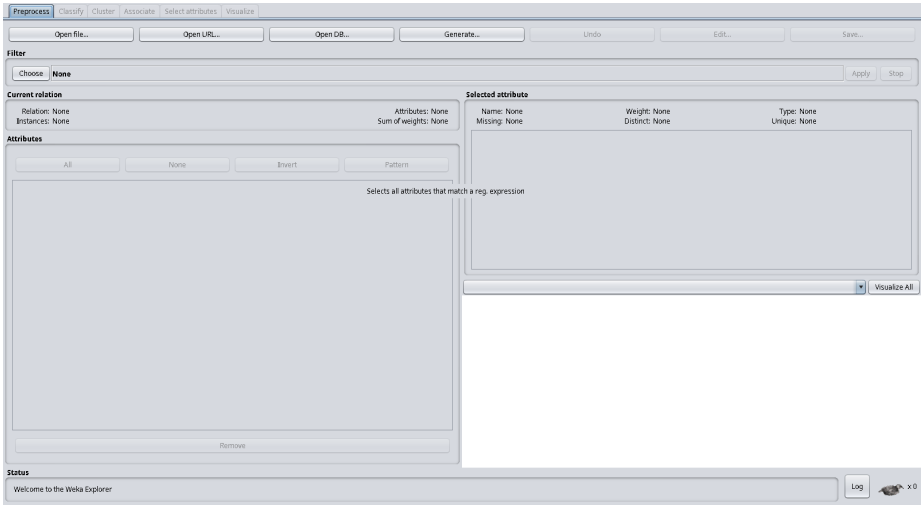
Các câu lệnh thông dụng thường sử dụng là:

- java <Tên lớp> <Các tham số>: Dùng để gọi lớp java tương ứng với các tham số.
- break: Tạm dừng thread hiện tại
- kill: Buộc dừng thread hiện tại
- cls: Xoá output
- capabilities <Tên lớp> <Các tham số>: Liệt kê sự tương thích của một lớp với các tham số.
- exit: xoá khu vực output.
- help <Câu lệnh>: Hướng dẫn cách sử dụng 1 câu lệnh cụ thể.

1.3 Explorer

1.3.1 Giao diện

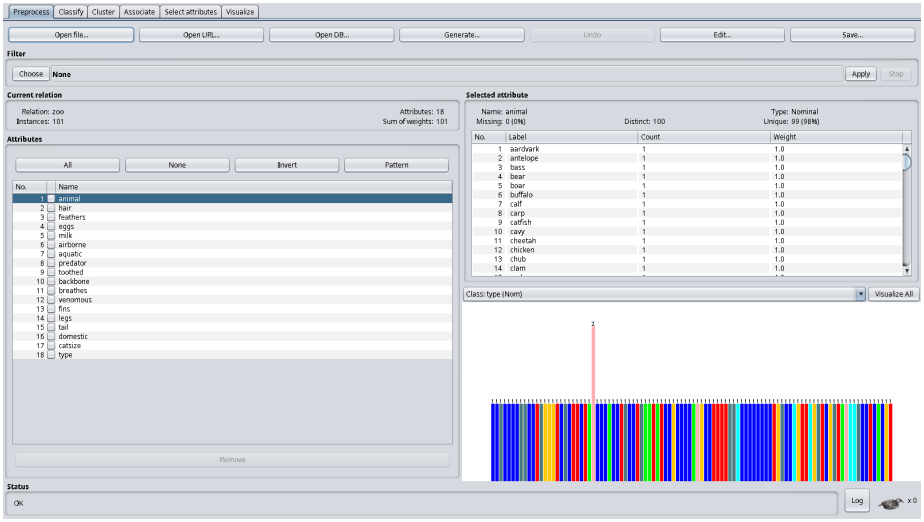
Giao diện gồm nhiều tab:



- Preprocess: Chọn dữ liệu và chỉnh sửa dữ liệu để thực hiện phân loại.
- Classify: Train và test các thuật toán học và hồi quy.
- Cluster: Học cụm đối với dữ liệu.
- Associate: Liên kết dữ liệu.
- Select attribute: Chọn dữ liệu phù hợp.
- Visualize: Mô phỏng dữ liệu 2D.

1.3.2 Preprocess

Trước tiên, ta sẽ load dữ liệu vào Weka thông qua các cách sau:

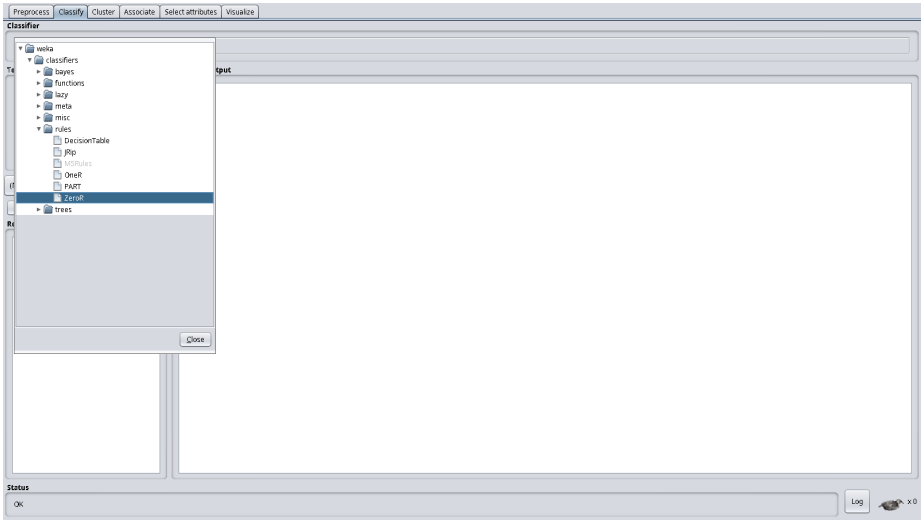


- Open file: Mở file trong máy tính
- Open URL: Mở file từ địa chỉ URL
- Open DB: Mở file từ cơ sở dữ liệu
- Generate: Tạo ra cơ sở dữ liệu mới.

Sau khi đã load được dữ liệu, ta có thể thực hiện lọc dữ liệu như xoá thuộc tính, lọc các thuộc tính,... Sau khi đã chọn được lọc phù hợp, ta có thể áp dụng nó để dữ liệu trước khi train.

1.3.3 Classification

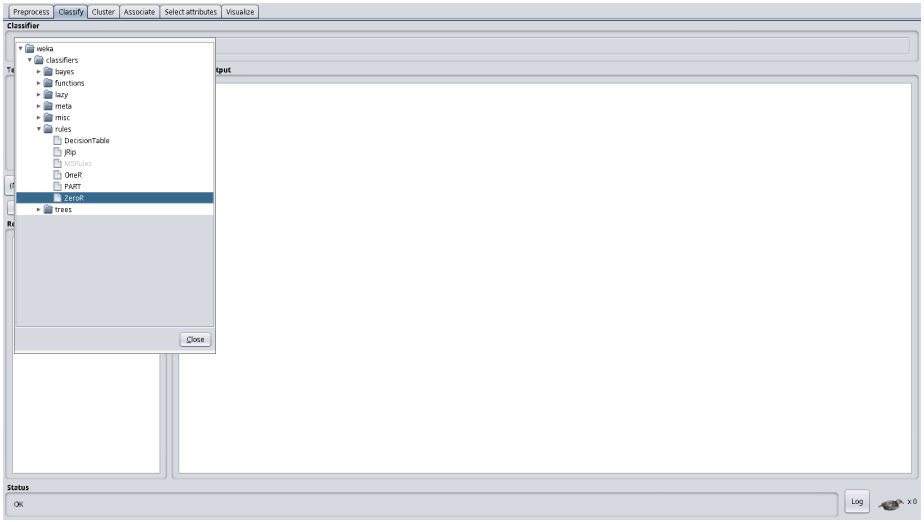
Ta sẽ chọn 1 cách classify cho phù hợp với từng yêu cầu của bài toán:



Sau khi đã chọn xong giải thuật phù hợp, bấm nút Start để bắt đầu. Weka sẽ hiện thị kết quả:

- Run information: Thông tin về tập dữ liệu như tên quan hệ, số thuộc tính, test mode, số dữ liệu.
- Classify model: Tuỳ thuộc vào chọn thuật toán mà thông tin hiển thị ra sẽ khác nhau.
- Summary: Một danh sách các thống kê về độ chính xác về tập dữ liệu test
- Detailed Accuracy By Class: Thông tin chi tiết hơn về độ chính xác được mở rộng ra từng lớp.
- Confusion Matrix: Số lượng dòng dữ liệu được gắn cho từng lớp.

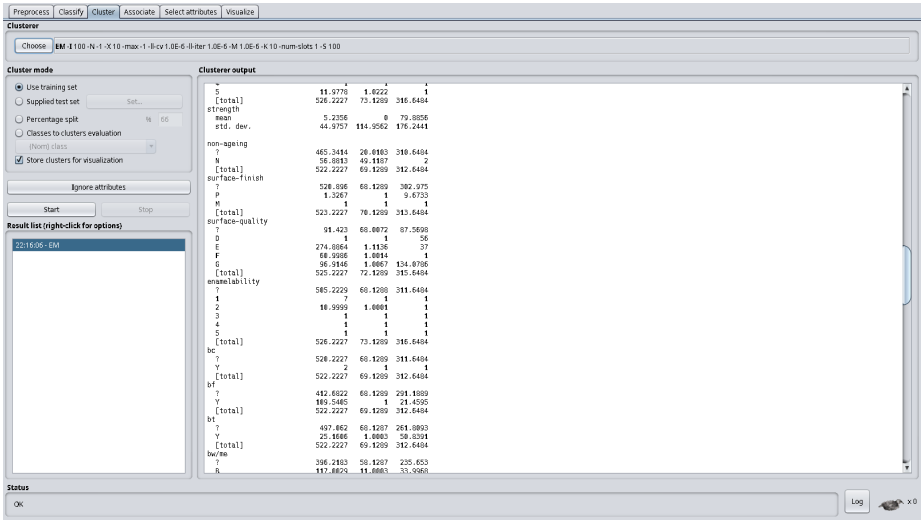
Sau khi chạy thì danh sách các kết quả được hiển thị trong ô result list, ta có thể visualize, xoá, lưu,... trên đó. Hình bên dưới là dùng visualize 1 cây được tạo từ tập dữ liệu zoo với thuật toán J48.



- Use training set: Sẽ dùng bộ dữ liệu để train để test
- Supplied test set: Sẽ thực hiện test trên bộ dữ liệu được chọn.
- Cross-validation: Thực hiện phân chia bộ training thành n tập con. Sẽ thực hiện lấy kết quả trung bình của n lần, mỗi lần train n-1 tập và test 1 tập tương ứng.
- Percentage split: Phân chia ra số phần trăm dùng để test và số phần trăm dùng để train.

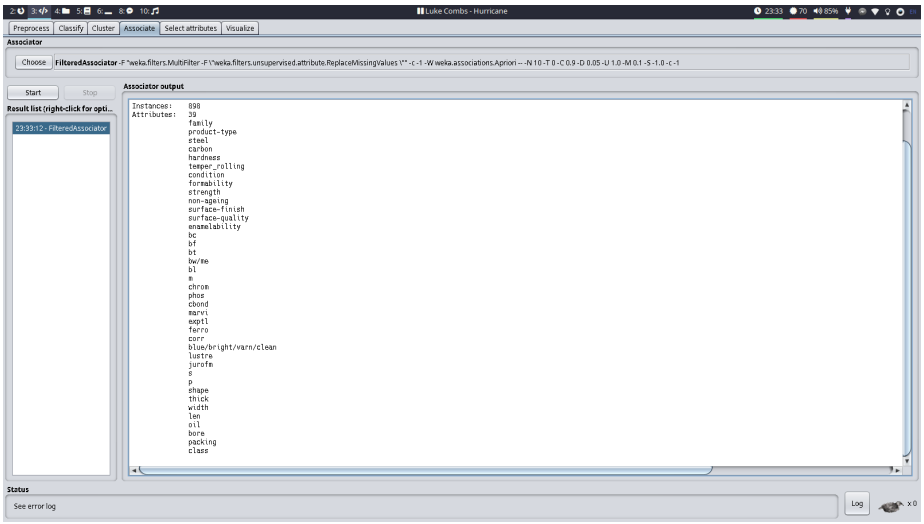
1.3.4 Clustering

Khá giống như chế độ Classification, chế độ clustering cũng có các chức năng tương tự như: visualize tree hoặc graph, bỏ 1 attribute, lọc bỏ bớt các thuật tính không cần thiết, chọn thuật toán để học...



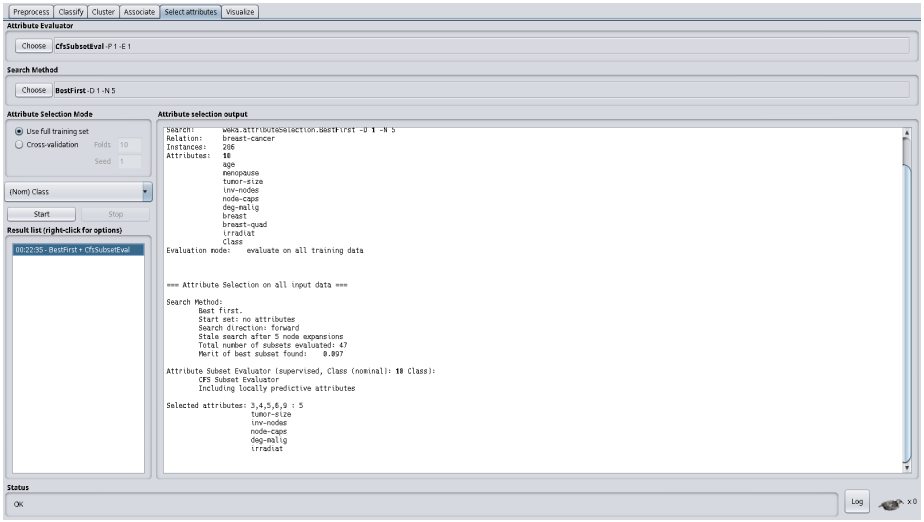
1.3.5 Associating

Cũng giống như cách sử dụng đối với clusterting hay classification. Associating cũng có các chức năng như chọn thuật toán học, lọc bớt các thuật tính, visualize tree hoặc graph...



1.3.6 Selecting Attributes

Tìm ra tập hợp các thuộc tính mang tính quyết đinh trong một tập hợp các thuộc tính đưa ra kết quả với khả năng cao nhất

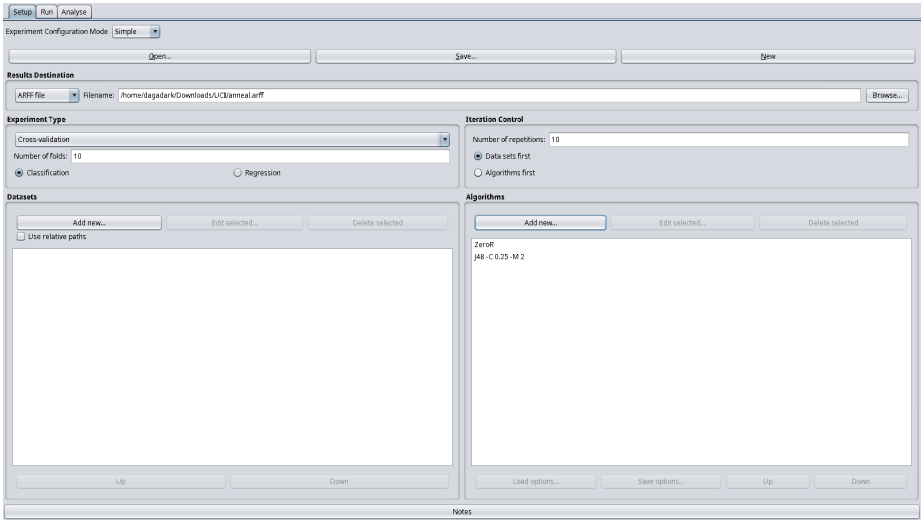


1.4 Experimenter

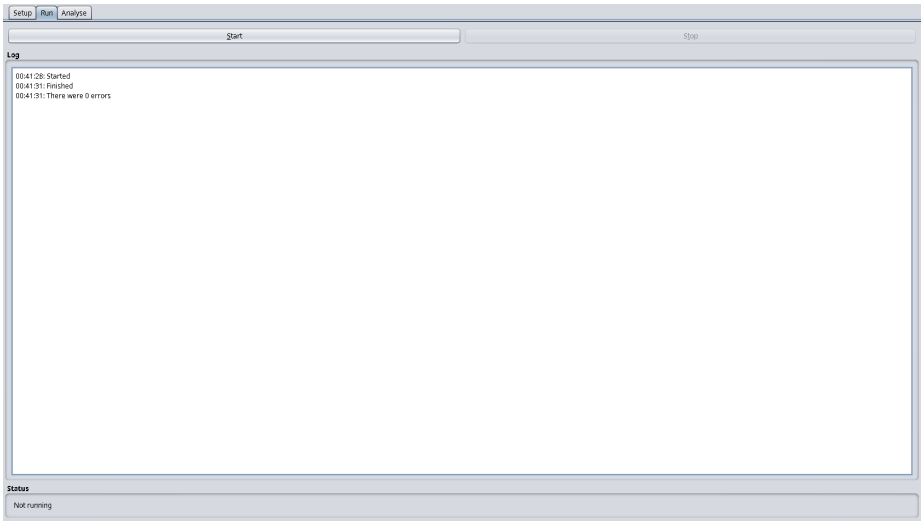
Để so sánh hai hay nhiều giải thuật khác nhau, ta sử dụng experimenter để so sánh các kết quả chạy đối với các bộ dữ liệu tương ứng. Kết quả của việc này sẽ được đưa ra file ARFF, CSV hay JDBC. Có 3 cách thử nghiệm chính: Cross-validation, train/test percentage split(random hay order preserved) Có 2 loại iteration control:

- Number of repetition: Số lần lặp lại để đạt được kết quả thống kê hợp lý.
- Data sets first/Algorithm first: Lưu dữ liệu trong database rồi mới thực hiện thuật toán hay là thực hiện thuật toán sớm nhất có thể.

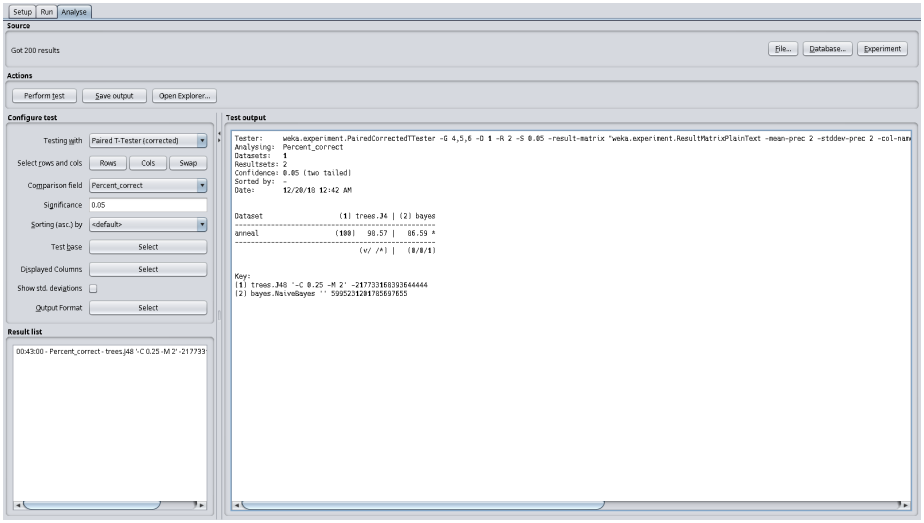
Ta có thể chọn các thuật toán khác nhau trong phần algorithm để thực hiện so sánh. Sau đó, ta có thể chọn file lưu ra.



Sau khi đã thực hiện được các thiết lập, ta sẽ sang phần chạy. Bấm nút start để bắt đầu và ta sẽ có kết quả như hình sau:



Sau khi đã chạy, ta có thể phân tích kết quả chạy dựa analyse

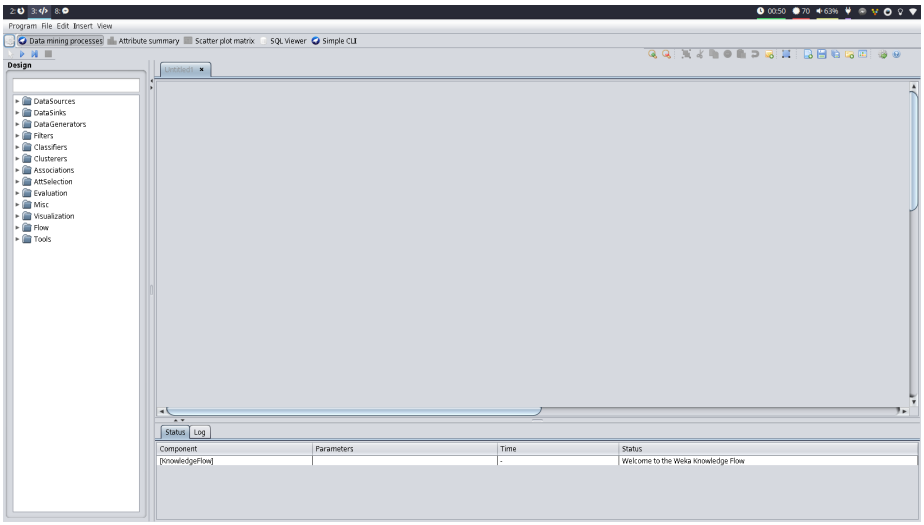


1.5 KnowledgeFlow

Cung cấp một công cụ thay thế cho giao diện explorer. Nó cung cấp các đặc điểm sau đây:

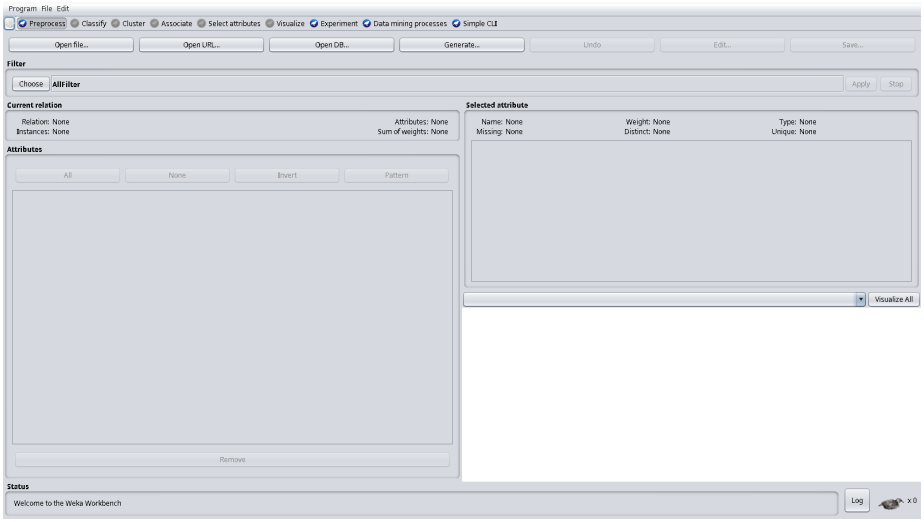
- dữ liệu dễ hình dung
- chạy được nhiều điểm xuất phát song song.
- kết hợp nhiều filter
- Hỗ trợ multithread
- Mô phỏng
- Hỗ trợ plugin
- ...

Các công cụ chủ yếu được mô tả như hình bên dưới:



1.6 Workbench

Là tổng hợp tất cả các chức năng của Weka được đưa vào một giao diện.



1.7 File ARFF

1.8 Tổng quát

ARFF gồm có 2 phần: Header và Data. Header gồm:

- Tên của quan hệ

- Danh sách các thuộc tính và loại thuộc tính

Data gồm các dữ liệu được thu thập theo dạng các dòng.

1.8.1 Header

Khai báo quan hệ: @relation <tên quan hệ> với tên quan hệ không được bắt đầu bằng các ký tự $\backslash \backslash u0021, \%$, và phải có đóng mở nháy đơn nếu có khoảng trắng. Khai báo thuộc tính: @attribute <tên thuộc tính> <kiểu dữ liệu> với kiểu dữ liệu phải thuộc trong các kiểu sau:

- numeric (gồm integer và real)
- string
- nominal có dạng {<nominal-name1>, <nominal-name2>...}
- date
- relation

1.8.2 Data

Được khai báo: @data, bên dưới sẽ khai báo các dữ liệu

2 Sử dụng Weka để chạy thuật toán ID3

2.1 Tạo tập tin Zoo.arff chứa dữ liệu Zoo

File ARFF (Attribute-Relation File Format) là một file ascii text mô tả danh sách các đối tượng cùng chia sẻ một tập các thuộc tính. Cấu trúc của file ARFF gồm 2 phần: Header và Data

- Header: Tên của quan hệ, danh sách các thuộc tính:

```
% 1. Title:  Zoo Database
% 2. Sources: http://archive.ics.uci.edu/ml/datasets/Zoo
@RELATION zoo
@ATTRIBUTE animal {aardvark,antelope,bass,bear,boar,buffalo,calf,
carp,catfish,cavy,cheetah,chicken,chub,clam,crab,crayfish,crow,deer,
dogfish,dolphin,dove,duck,elephant,flamingo,flea,
frog,fruitbat,giraffe,girl,gnat,goat,gorilla,gull,haddock,hamster,
hare,hawk,herring,honeybee,housefly,kiwi,ladybird,lark,
leopard,lion,lobster,lynx,mink,mole,mongoose,moth,newt,octopus,
opossum,oryx,ostrich,parakeet,penguin,pheasant,pike,piranha,
pitviper,platypus,polecat,pony,porpoise,puma,pussycat,
raccoon,reindeer,rhea,scorpion,seahorse,seal,sealion,seasnake,
seawasp,skimmer,skua,slowworm,slug,sole,sparrow,squirrel,
starfish,stingray,swan,termite,toad,tortoise,
tuatara,tuna,vampire,vole,vulture,wallaby,wasp,wolf,worm,wren}
@ATTRIBUTE hair {0, 1}
@ATTRIBUTE feathers {0, 1}
@ATTRIBUTE eggs {0, 1}
@ATTRIBUTE milk {0, 1}
@ATTRIBUTE airborne {0, 1}
@ATTRIBUTE aquatic {0, 1}
@ATTRIBUTE predator {0, 1}
@ATTRIBUTE toothed {0, 1}
@ATTRIBUTE backbone {0, 1}
@ATTRIBUTE breathes {0, 1}
@ATTRIBUTE venomous {0, 1}
@ATTRIBUTE fins {0, 1}
@ATTRIBUTE legs {0, 2, 4, 5, 6, 8}
@ATTRIBUTE tail {0, 1}
@ATTRIBUTE domestic {0, 1}
@ATTRIBUTE catsize {0, 1}
@ATTRIBUTE type { mammal, bird, reptile, fish, amphibian, insect, invertebrate }
```

- Data: Chưa dữ liệu của các đối tượng:

```
@DATA
% Instances (101):
```

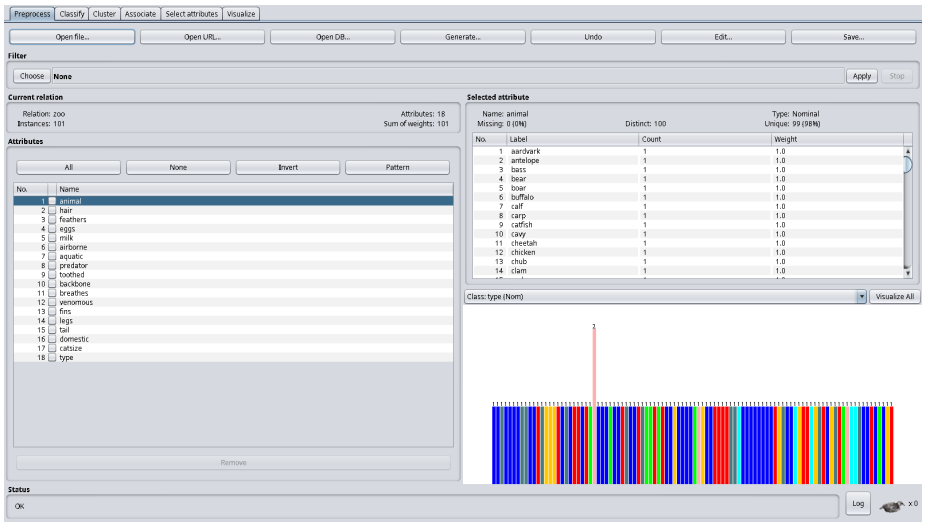
```
aardvark,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mammal
antelope,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mammal
bass,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
bear,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mammal
boar,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,mammal
buffalo,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mammal
calf,1,0,0,1,0,0,0,1,1,1,0,0,4,1,1,1,mammal
carp,0,0,1,0,0,1,0,1,1,0,0,1,0,1,1,0,fish
catfish,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
cavy,1,0,0,1,0,0,0,1,1,1,0,0,4,0,1,0,mammal
cheetah,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,mammal
chicken,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0,bird
```

2.2 Mô tả tổng quát về dữ liệu Zoo

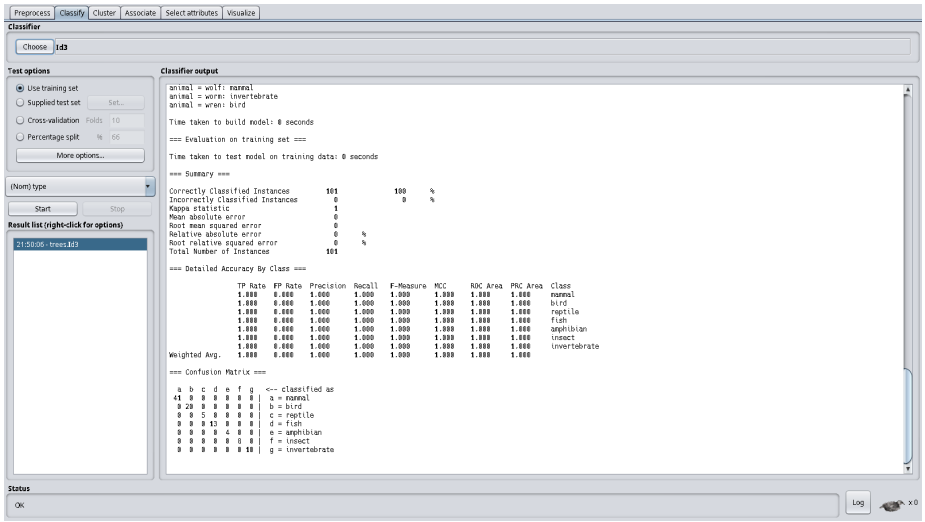
- Số mẫu: 101
- Tên và ý nghĩa của các thuộc tính:
  - animal: tên các loài động vật (gồm 100 tên)
  - hair: 0: Không có tóc, 1: có tóc
  - eggs: 0: không đẻ trứng, 1: đẻ trứng
  - milk: 0: không có sữa, 1: có sữa
  - airborne 0: không bay được, 1: bay được
  - aquatic: 0: không sống ở nước, 1: sống ở nước
  - predator: 0: không ăn thịt, 1: ăn thịt
  - toothed: 0: không có răng, 1: có răng
  - backbone: 0: không có xướng sống, 1: có sướng sống
  - breathes: 0: không thở trên cạn, 1: có thở trên cạn
  - vonomous: 0: không độc, 1: có độc
  - fins: 0: không có vây, 1: có vây
  - legs: số lượng chi: 0, 2, 4, 5, 6, 8
  - tail: 0: không có đuôi, 1: có đuôi
  - domestic: 0: không nuôi trong nhà, 1: có nuôi trong nhà
  - catsize: 0: không thuộc họ mèo, 1: họ mèo
  - type: 7 lớp động vật: mammal (động vật có vú), bird(chim), reptile (bò sát), fish (cá), amphibian (lưỡng cư), insect (côn trùng), invertebrate (động vật không xương sống)

2.3 Thuật toán ID3

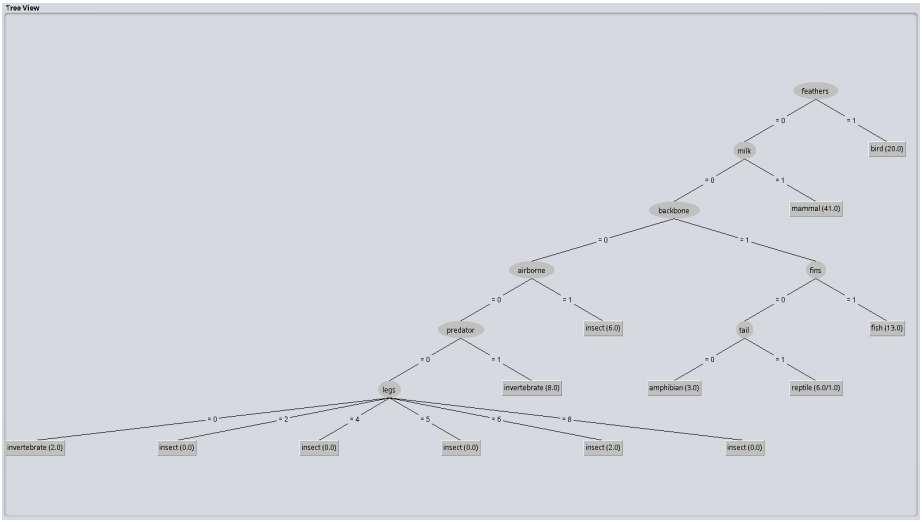
Chọn Open file và đưa tập dữ liệu Zoo.arff vào:



Chọn thẻ Classify, chọn thuật toán ID3, chọn tập dữ liệu test (ở đây chọn lại chính nó), rồi bấm start. Ta thu được kết quả:



Từ model, ta sinh ra cây quyết định:



2.4 Prediction từ Decision Tree

Dựa vào cây quyết định ở trên ta có thể đoán được kết quả của 5 mẫu như sau:

1. NameIsSecret,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1, 'mammal'
2. NameIsSecret,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0, 'bird'
3. NameIsSecret,0,0,1,0,0,0,1,1,1,1,1,0,0,1,0,0, 'reptile'
4. NameIsSecret,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0, 'fish'
5. NameIsSecret,0,0,1,0,0,1,1,1,1,1,0,0,4,1,0,0, 'reptile'

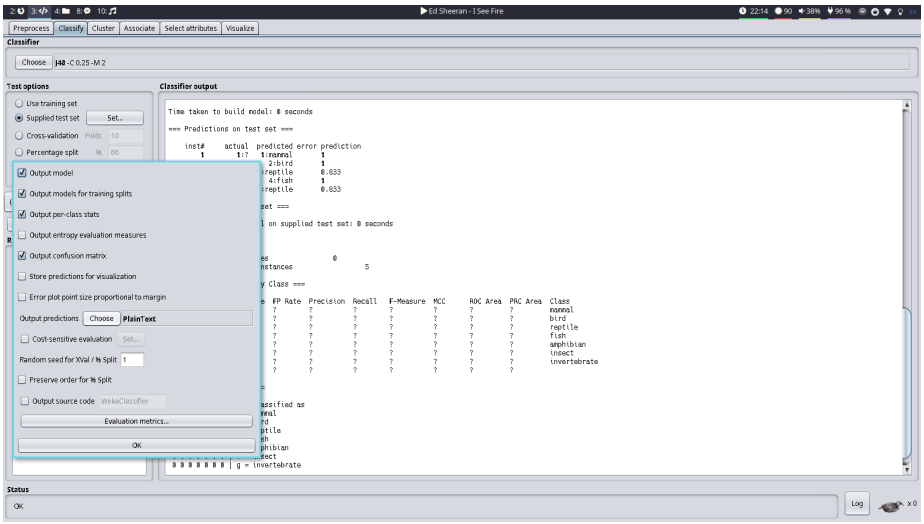
Kiểm chứng với Weka, ta tạo 1 file data_set_test.arff:

```
@RELATION zoo
@ATTRIBUTE animal {aardvark,antelope,bass,bear,boar,buffalo,calf,carp,catfish,cavy,cheetah,c
@ATTRIBUTE hair {0, 1}
@ATTRIBUTE feathers {0, 1}
@ATTRIBUTE eggs {0, 1}
@ATTRIBUTE milk {0, 1}
@ATTRIBUTE airborne {0, 1}
@ATTRIBUTE aquatic {0, 1}
@ATTRIBUTE predator {0, 1}
@ATTRIBUTE toothed {0, 1}
@ATTRIBUTE backbone {0, 1}
@ATTRIBUTE breathes {0, 1}
@ATTRIBUTE venomous {0, 1}
@ATTRIBUTE fins {0, 1}
@ATTRIBUTE legs {0, 2, 4, 5, 6, 8}
@ATTRIBUTE tail {0, 1}
```

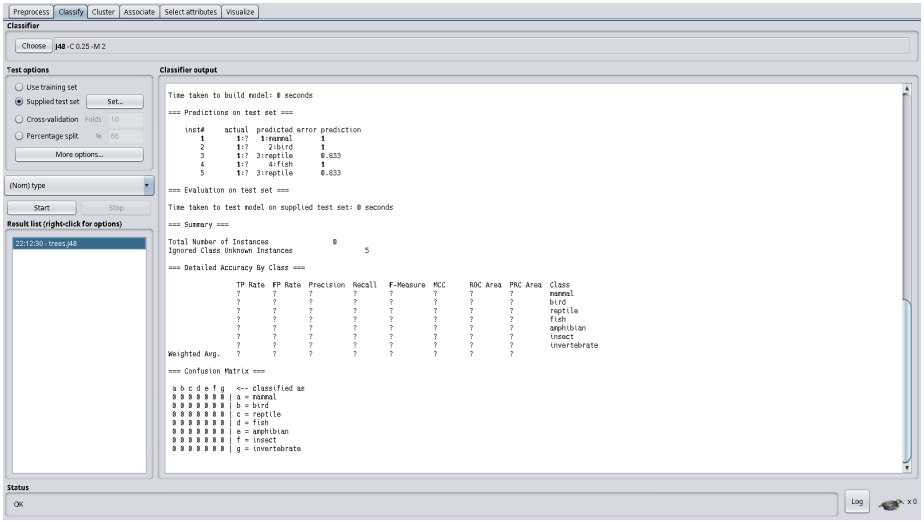
```
@ATTRIBUTE domestic {0, 1}
@ATTRIBUTE catsize {0, 1}
@ATTRIBUTE type { mammal, bird, reptile, fish, amphibian, insect, invertebrate}

@DATA
% Instances (101):
?,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1, ?
?,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0, ?
?,0,0,1,0,0,0,1,1,1,1,1,0,0,1,0,0, ?
?,0,0,1,0,0,1,1,1,1,1,0,0,1,0,1,0,0, ?
?,0,0,1,0,0,1,1,1,1,1,0,0,4,1,0,0, ?
```

Trong mục Supplied test set ta chọn file test_data_set đã tạo ở trên. Chọn More Option và chọn Output predictions.



Click start để thu được kết quả:



3 Chạy các thuật toán khác

3.1 Chương trình python cho giải thuật Naive Bayes

Giả sử ta có 1 tập dữ liệu training set gồm n thuộc tính $(f_1, f_2, ...f_n)$ ứng với m lable $(L_1, L_2, \ldots, L_m)$ Thuật toán Naive Bayes chia làm 2 giai đoạn
+ Giai đoạn học: Tính xác suất cho từng thuộc tính với từng nhãn

$$P(fi|Li) = \frac{count(fi, Li)}{count(Li)} = \frac{numberOfInstances(Li, fi)}{numberofInstances(Li)}$$

Tính xác suất của từng nhãn trong toàn bộ dữ liệu

$$P(Li) = \frac{numberOfInstances(Li)}{numberOfInstances}$$

+ Giai đoạn dự đoán input để dự đoán: Tập các thuộc tính $(f1, f2, \ldots, fn)$ Tiến hành tính xác suất xảy ra nhãn Li ứng với input và chọn nhãn có xác suất lớn nhất

$$P(L = Li|f1, f2 \ldots, fn) = P(f1|Li) * P(f2|Li) * ... * P(fn|Li) * P(Li)$$

3.1.1 Cài đặt

- Hàm đọc file zoo.arff

```
def loadArff(filename):
        dataset, meta = arff.loadarff('zoo.arff')
        dataset2 = []
        for i in range(len(dataset)):
```

```
                new =[]
                for j in range(len(dataset[i])):
                        if j!=13:
                                new.append(dataset[i][j].decode('utf-8'))
                        else:
                                new.append(int(dataset[i][j]))
                dataset2.append(new)
        return dataset2
```

- Hàm tách tập dữ liệu thành training set và test set theo tỷ lệ splitRatio

```
def splitDataset(dataset, splitRatio):
        trainSize = int(len(dataset) * splitRatio)
        trainSet = []
        copy = list(dataset)
        while len(trainSet) < trainSize:
                index = random.randrange(len(copy))
                trainSet.append(copy.pop(index))
        return [trainSet, copy]
```

- Hàm tính xác suất P(fi|Li) và P(Li)

```
def dem(trainingSet):
        cnt={}
        #init all element in dictionary cnt = 0
        for i in range(len(trainingSet)):
                for j in range(len(trainingSet[i]) - 1): #-label
                        cnt[(j,trainingSet[i][j],
                        trainingSet[i][len(trainingSet[i]) - 1])]=0
        for i in range(len(trainingSet)):
                for j in range(len(trainingSet[i]) - 1): #-label
                        cnt[(j,trainingSet[i][j],
                        trainingSet[i][len(trainingSet[i])-1])]+=1
        return cnt

def prob(trainingSet):
        cnt = dem(trainingSet)
        cntLabel={}
        for i in range(len(trainingSet)):
                cntLabel[trainingSet[i][len(trainingSet[i]) - 1]]=0
        for i in range(len(trainingSet)):
                cntLabel[trainingSet[i][len(trainingSet[i]) - 1]]+=1
        prob={}
        for key,value in cnt.items():
                prob[key]=value / cntLabel[key[2]]
        for key,value in cntLabel.items():
                prob[key]=value / len(trainingSet)
        return prob, cntLabel
```

- Hàm đoán cho mẫu truy vấn

```
#probTable: bảng xác suất đã tính, query: câu truy vấn
def predict(trainingSet, query):
        labels=['mammal', 'bird', 'reptile', 'fish',
        'amphibian', 'insect', 'invertebrate']
        probTable, cntLabel = prob(trainingSet)
        maxP = 0;
        bestLabel=''
        for i in range(len(labels)):
                product=1
                for j in range(len(query)-1): #attribute from 0 to 16
                        if (j, query[j], labels[i]) not in probTable:
                                probTable[(j, query[j], labels[i])] =
                                1/(cntLabel[labels[i]] + len(trainingSet))
                        if query[j]!='?':
                                product *= probTable[(j, query[j],
                                labels[i])]
                if product >= maxP:
                        maxP = product
                        bestLabel = labels[i]
        return bestLabel
```

- Hàm lấy kết quả dự đoán cho cả tập testSet và lấy độ chính xác

```
def getPredictions(trainingSet,testSet): #cntLabel: đếm số lượng các label
        predictions = []
        for i in range(len(testSet)):
                result = predict(trainingSet, testSet[i])
                predictions.append(result)
        return predictions

def getAccuracy(testSet, predictions):
        correct = 0
        for i in range(len(testSet)):
                if testSet[i][-1] == predictions[i]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0
```

- Hàm main() thực hiện các bước trên

```
def main():
        dataset = loadArff('zoo.arff');
        splitRatio = 0.67
        trainingSet, testSet = splitDataset(dataset,splitRatio)
        predictions= getPredictions(trainingSet,testSet)
        print(predictions);
        accuracy = getAccuracy(testSet,predictions)
```

```
            print('Accuracy = ',accuracy,'%')
main()
```

3.1.2 Hàm làm trơn: smoothing function

Ở đây, nhóm dùng hàm làm trơn Laplace Smoothing với hệ số alpha = 1 (Laplace add 1 Smoothing Function).
Đối với những thuộc tính gắn với nhãn mới, chưa gặp qua trong training set. Lúc đó, theo nguyên tắc toán học thì: $P(f'|L') = 0$. Nhưng như vậy sẽ dẫn trường hợp tính xác suất cho các nhãn Li ứng với tập thuộc tính input đều bằng 0, ta không thể chọn ra nhãn có xác suất lớn nhất được. Để khắc phục tình trạng này, cần áp dụng hàm làm trơn cho những thuộc tính ứng với nhãn mới như vậy để cho $P(f'|L')$ đủ nhỏ và vẫn đảm bảo phân biệt được giữa các nhãn trong lúc tính cho mẫu input dự đoán.

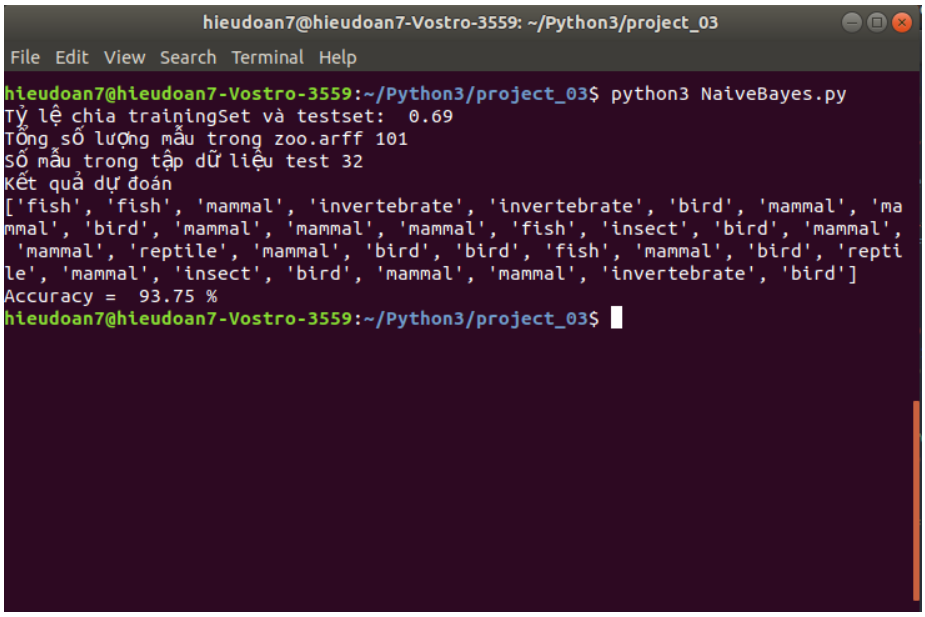
$$P(f'|L') = \frac{1}{(count(L') + |V|)}$$

Trong đó:

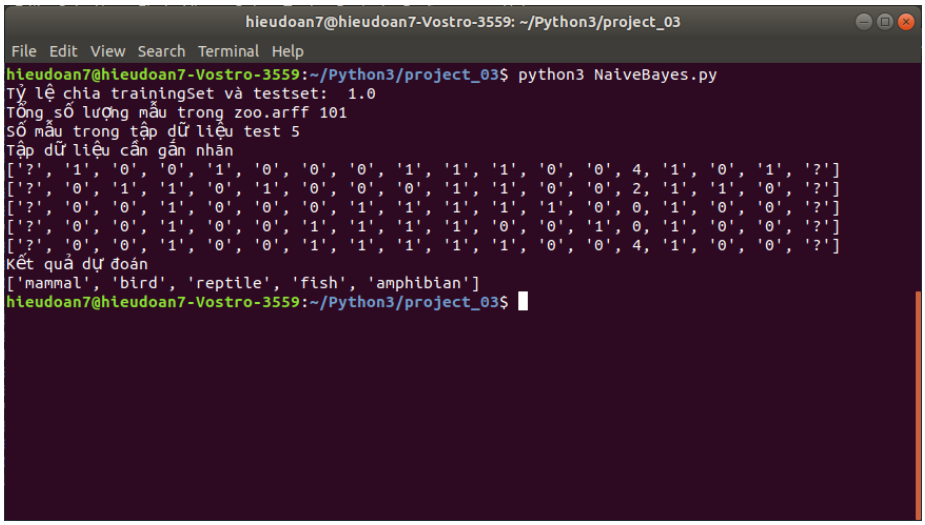
- count(L'): số lượng mẫu có nhãn L' trong tập training set
- |V|: Tổng số lượng mẫu trong tập training set

3.1.3 Chạy chương trình python cho giải thuật Naive Bayes

Chạy giải thuật Naive Bayes với tỷ lệ splitDataset=0.69 trên tập dữ liệu zoo.arff thu được kết quả.



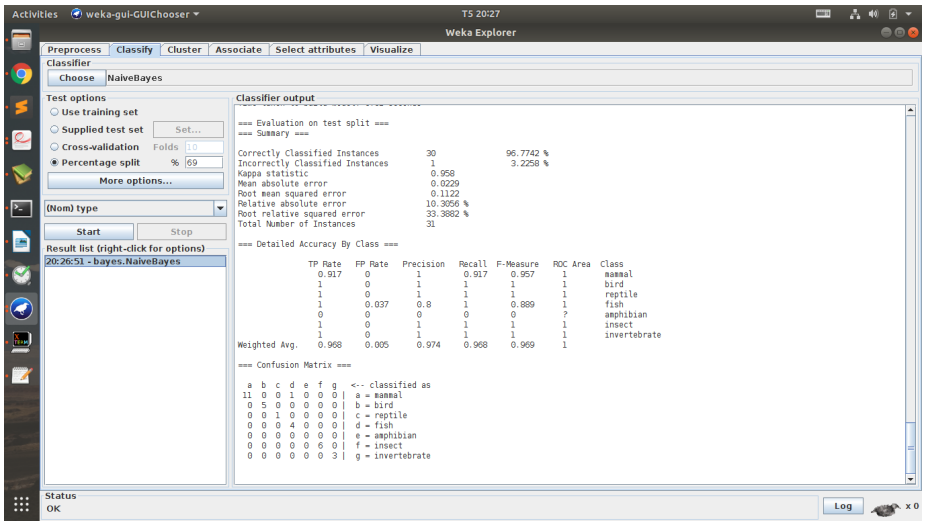
Chạy với 5 mẫu test.



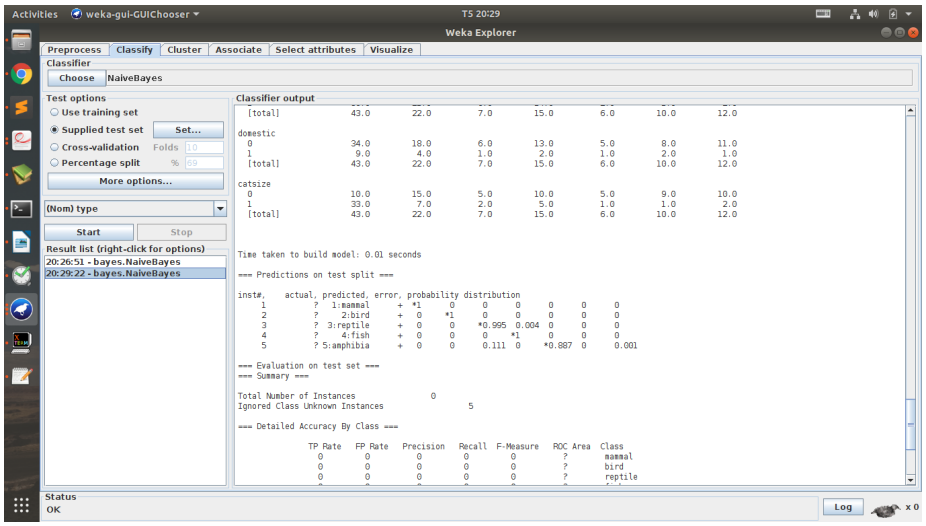
3.2 Các thuật toán khác trên Weka

3.2.1 Naive Bayes

Chạy thuật toán Naive Bayes trên Weka với tập dữ liệu zoo.arff với tỷ lệ 0.69.



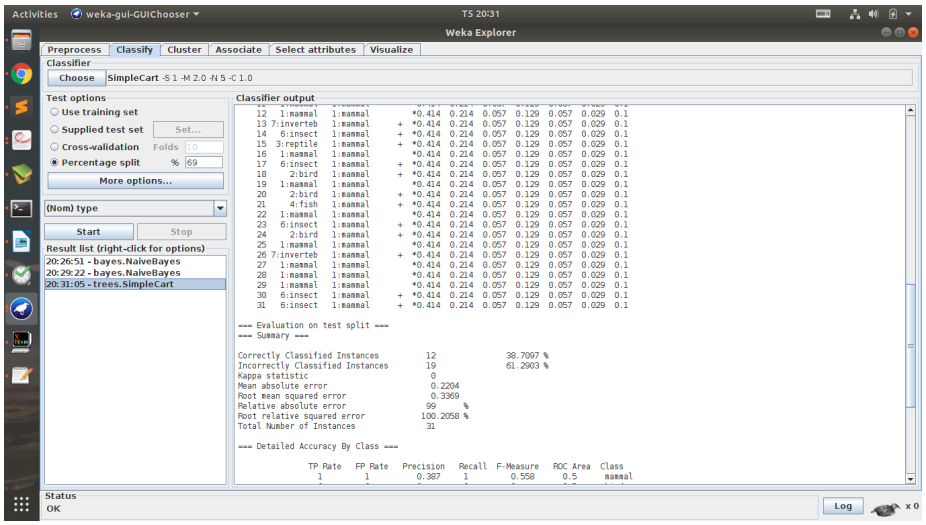
Test trên bộ dữ liệu gồm 5 mẫu ở trên (test_data_set.arff)



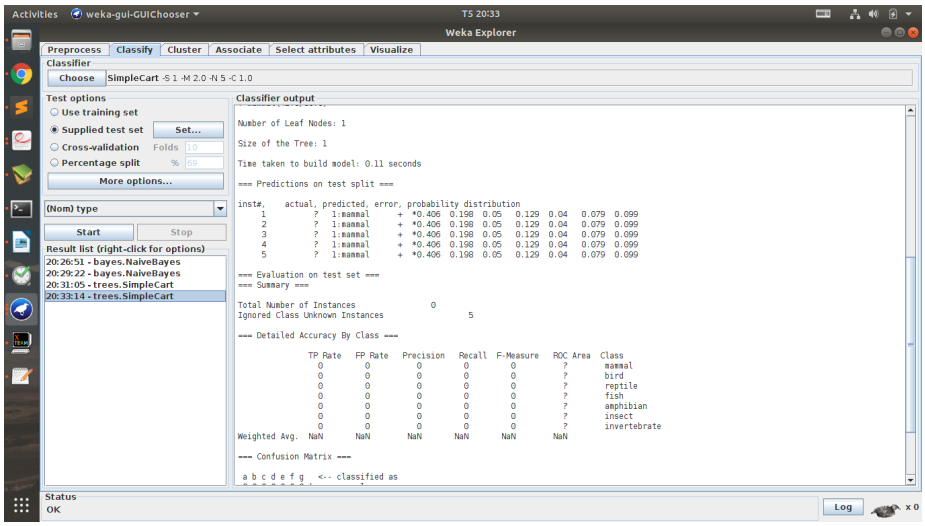
Ta thấy kết quả là: ‘mammal’, ‘bird’, ‘reptile’, ‘fish’, ‘amphibia’ giống như chạy khi cài đặt bằng Python ở trên.

3.2.2 SimpleCart

Chạy thuật toán SimpleCart trên Weka với tập dữ liệu zoo.arff với tỷ lệ 0.69.

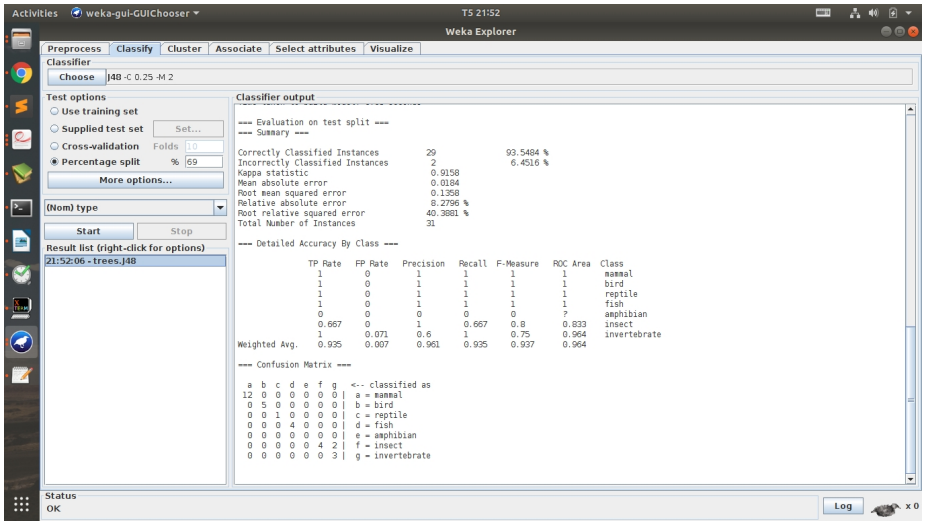


Test trên bộ dữ liệu gồm 5 mẫu ở trên (test_data_set.arff)

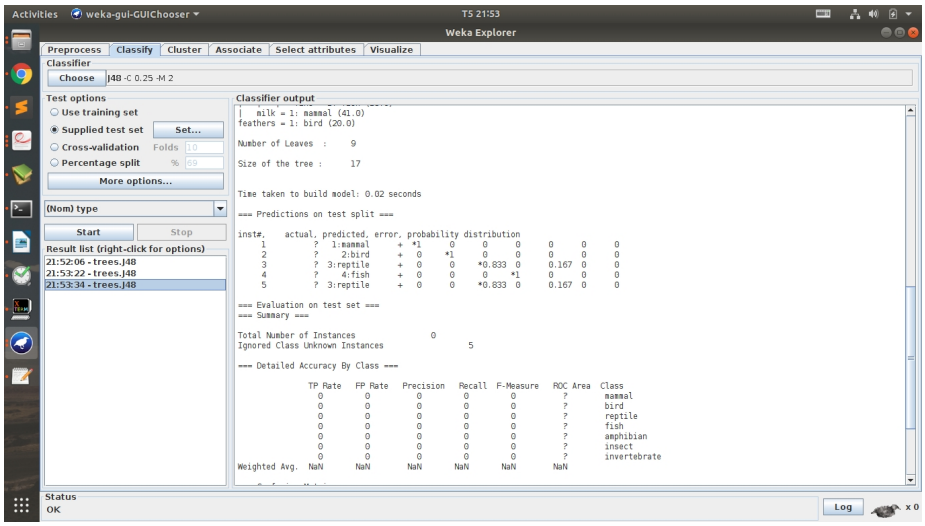


3.2.3 J48

Chạy thuật toán J48 trên Weka với tập dữ liệu zoo.arff với tỷ lệ 0.69.



Test trên bộ dữ liệu gồm 5 mẫu ở trên (test_data_set.arff)


Activities
weka-gui-GUIChooser
T5 21:53
Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
J48 -C 0.25 -M 2
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
Percentage split
More options...
(Nom) type
Start
Stop
Result list (right-click for options)
21:52:06 - trees.J48
21:53:22 - trees.J48
21:53:34 - trees.J48
Classifier output
| milk = 1: mammal (41.0)
feathers = 1: bird (20.0)
Number of Leaves : 9
Size of the tree : 17
Time taken to build model: 0.02 seconds
=== Predictions on test split ===
inst#, actual, predicted, error, probability distribution
1 ? 1:mammal + *1 0 0 0 0 0 0
2 ? 2:bird + 0 *1 0 0 0 0 0
3 ? 3:reptile + 0 0 *0.833 0 0.167 0 0
4 ? 4:fish + 0 0 0 *1 0 0 0
5 ? 3:reptile + 0 0 *0.833 0 0.167 0 0
=== Evaluation on test set ===
=== Summary ===
Total Number of Instances 0
Ignored Class Unknown Instances 5
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
0 0 0 0 0 ? mammal
0 0 0 0 0 ? bird
0 0 0 0 0 ? reptile
0 0 0 0 0 ? fish
0 0 0 0 0 ? amphibian
0 0 0 0 0 ? insect
0 0 0 0 0 ? invertebrate
Weighted Avg. NaN NaN NaN NaN NaN NaN
Status
OK
Log
x 0