"""

Andrew Burke, Katreina Carpenter, Cody Valle

HW3

Performs Naive Bayes algorithm on the 'auto-data.txt' dataset.

"""

import copy

import random

from decimal import Decimal

""" Reads a csv file and returns a table (list of lists) """

def read\_csv(filename):

import csv

the\_file = open(filename, 'r')

the\_reader = csv.reader(the\_file, dialect='excel')

table = []

for row in the\_reader:

if len(row) > 0:

table.append(row)

the\_file.close()

return table

""" Makes floats """

def get\_column\_as\_floats(table, index):

column = []

for row in table:

if row[index] != 'NA':

column.append(float(row[index]))

return column

"""

Gets a list of all different categorical values

"""

def get\_categories(table, index):

values = []

for row in table:

if row[index] not in values:

values.append(row[index])

return values

"""

Builds a Naive Bayes classifier

"""

def build\_naive\_bayes\_classifier(table, classes):

ret = {}

for c in classes:

# List possible class label values

label\_values = get\_categories(table, c)

# Fill dictionary with rows partitioned on class label values

nb\_column = {}

for value in label\_values:

nb\_column[value] = []

for row in table:

nb\_column[row[c]].append(row)

# Create return object

ret[c] = nb\_column

return copy.deepcopy(ret)

"""

Guesses a class label for the passed in instance based on the passed

in class index and attribute indices.

"""

def guess\_label\_categorical(classifier, instance, attrs, class\_index):

nb\_table = classifier[class\_index]

total\_rows = Decimal(sum([len(tb) for \_,tb in nb\_table.items()]))

guess = (Decimal(-1),'')

for class\_label,rows in nb\_table.items():

probability = Decimal(1)

denominator = Decimal(len(rows))

for attr in attrs:

numerator = Decimal(0)

for row in rows:

if row[attr] == instance[attr]:

numerator += 1

probability \*= numerator / denominator

probability \*= denominator / total\_rows

if probability > guess[0]:

guess = (probability,class\_label)

return guess[1]

"""

Tests the categorical Naive Bayes classifier from Step 1

old step 2: select random instrances from the dataset, predict

their corresponding mpg ranking, and then show their actual ranking

old step 3: compute the (multi-class) predictive accuracy and error

rate of the classifiers using separate training and testing sets

approch 1: random subsampling with k=10

approach 2: stratified k-fold cross validation with k\_10

old step 4: create confusion matrices for each classifer based on

the stratified 10-fold cross validation results

"""

def test1\_of\_naive\_bayes\_classifier(classifier, table):

from tabulate import tabulate

# Test using step2 approach from HW3

print '==========================================='

print 'TEST 1: Naive Bayes MPG Classifier'

print '==========================================='

total\_tests = 5

random\_instances = [random.randint(0, len(table) - 1) for \_ in range(total\_tests)]

correct = 0

for index in random\_instances:

predicted = guess\_label\_categorical(classifier, table[index], [WEIGHT,CYLINDERS,MODEL\_YEAR], MPG)

print ' predicted:', predicted, ' actual:', table[index][MPG]

if predicted == table[index][MPG]:

correct += 1

accuracy = float(correct) / total\_tests

print ''

print ' For k=5 Random Subsampling:'

print ' accuracy:', accuracy, ', error\_rate:', (1 - accuracy)

print ''

# Confusion matrix

matrix = [[0] \* 10 for \_ in range(10)]

for row in table:

predicted = guess\_label\_categorical(classifier, row, [WEIGHT,CYLINDERS,MODEL\_YEAR], MPG)

matrix[int(row[MPG]) - 1][int(predicted) - 1] += 1

tabbed\_table = []

for i in range(10):

total = sum(matrix[i]) \* 1.0

tabbed\_table.append([str(i + 1)] +

[matrix[i][j] for j in range(len(matrix[i]))] +

[total] +

[100. \* matrix[i][i] / (total if total != 0 else 1)])

print tabulate(tabbed\_table, headers = ['MPG', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Total', 'Recognition (%)'])

"""

Takes an instance and returns a label based on the MPG value.

"""

def convert\_mpg\_to\_categorical(table):

for row in table:

if float(row[MPG]) <= 13:

row[MPG] = '1'

elif float(row[MPG]) <= 15:

row[MPG] = '2'

elif float(row[MPG]) <= 17:

row[MPG] = '3'

elif float(row[MPG]) <= 20:

row[MPG] = '4'

elif float(row[MPG]) <= 24:

row[MPG] = '5'

elif float(row[MPG]) <= 27:

row[MPG] = '6'

elif float(row[MPG]) <= 31:

row[MPG] = '7'

elif float(row[MPG]) <= 37:

row[MPG] = '8'

elif float(row[MPG]) <= 45:

row[MPG] = '9'

else:

row[MPG] = '10'

"""

Converts the weight column to categories

"""

def convert\_weight\_to\_categorical(table):

for row in table:

if float(row[WEIGHT]) <= 1999:

row[WEIGHT] = '1'

elif float(row[WEIGHT]) <= 2499:

row[WEIGHT] = '2'

elif float(row[WEIGHT]) <= 2999:

row[WEIGHT] = '3'

elif float(row[WEIGHT]) <= 3499:

row[WEIGHT] = '4'

else:

row[WEIGHT] = '5'

"""

Step 1:

"""

def step1(table):

# Converts the weight attribute to categorical

convert\_weight\_to\_categorical(table)

# Converts the MPG attribute to categorical

convert\_mpg\_to\_categorical(table)

# Builds a Naive Bayes classifier

classifier = build\_naive\_bayes\_classifier(table, [MPG])

# Test the classifier

test1\_of\_naive\_bayes\_classifier(classifier, table)

"""

Guesses a class label for the passed in instance based on the passed

in class index and attribute indices. Can handle Gaussian distributions.

"""

def guess\_label\_with\_continuous(classifier, instance, cat\_attrs, cont\_attrs, class\_index):

import math

nb\_table = classifier[class\_index]

total\_rows = Decimal(sum([len(tb) for \_,tb in nb\_table.items()]))

guess = (Decimal(-1),'')

# Gaussian function

def gaussian\_probability(x, mean, sdev):

first, second = Decimal(0), Decimal(0)

if sdev > 0:

first = Decimal(1 / (math.sqrt(2 \* math.pi) \* sdev))

second = Decimal(math.e \*\* (-((x - mean) \*\* 2) / (2 \* (sdev \*\* 2))))

return first \* second

for class\_label,rows in nb\_table.items():

probability = Decimal(1)

denominator = Decimal(len(rows))

for attr in cat\_attrs:

numerator = Decimal(0)

for row in rows:

if row[attr] == instance[attr]:

numerator += 1

probability \*= numerator / denominator

for attr in cont\_attrs:

x = float(instance[attr['index']])

mean = attr['mean']

sdev = attr['sdev']

probability \*= gaussian\_probability(x, mean, sdev)

probability \*= denominator / total\_rows

if probability > guess[0]:

guess = (probability,class\_label)

return guess[1]

"""

Tests the categorical Naive Bayes classifier from Step 1

old step 2: select random instrances from the dataset, predict

their corresponding mpg ranking, and then show their actual ranking

old step 3: compute the (multi-class) predictive accuracy and error

rate of the classifiers using separate training and testing sets

approch 1: random subsampling with k=10

approach 2: stratified k-fold cross validation with k\_10

old step 4: create confusion matrices for each classifer based on

the stratified 10-fold cross validation results

"""

def test2\_of\_naive\_bayes\_classifier(classifier, table):

from tabulate import tabulate

import numpy

# Test using step2 approach from HW3

print '==========================================='

print 'TEST 2: Naive Bayes MPG Classifier'

print '==========================================='

total\_tests = 5

random\_instances = [random.randint(0, len(table) - 1) for \_ in range(total\_tests)]

correct = 0

weights = get\_column\_as\_floats(table, WEIGHT)

d = {'index' : WEIGHT,

'mean' : 1.0 \* sum(weights) / len(weights),

'sdev' : numpy.std(weights)}

for index in random\_instances:

predicted = guess\_label\_with\_continuous(classifier, table[index], [MODEL\_YEAR,CYLINDERS], [d], MPG)

print ' predicted:', predicted, ' actual:', table[index][MPG]

if predicted == table[index][MPG]:

correct += 1

# Print statistics from step3 of HW3

accuracy = float(correct) / total\_tests

print ''

print ' For k=5 Random Subsampling:'

print ' accuracy:', accuracy, ', error\_rate:', (1 - accuracy)

print ''

# Confusion matrix from step4 of HW3

matrix = [[0] \* 10 for \_ in range(10)]

for row in table:

predicted = guess\_label\_with\_continuous(classifier, row, [MODEL\_YEAR,CYLINDERS], [d], MPG)

matrix[int(row[MPG]) - 1][int(predicted) - 1] += 1

tabbed\_table = []

for i in range(10):

total = sum(matrix[i]) \* 1.0

tabbed\_table.append([str(i + 1)] +

[matrix[i][j] for j in range(len(matrix[i]))] +

[total] +

[100. \* matrix[i][i] / (total if total != 0 else 1)])

print tabulate(tabbed\_table, headers = ['MPG', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Total', 'Recognition (%)'])

"""

Step2:

"""

def step2(table):

# Converts the MPG attribute to categorical

convert\_mpg\_to\_categorical(table)

# Builds a Naive Bayes classifier

classifier = build\_naive\_bayes\_classifier(table, [MPG])

# Test the classifier

test2\_of\_naive\_bayes\_classifier(classifier, table)

"""

Takes a table and removes every row that has an 'NA' present

"""

def clean\_auto\_data(table):

ret = []

for row in table:

found = False

for value in row:

if value == 'NA':

found = True # There is an element in this row with 'NA'

break

if not found: # If there are no elements with 'NA'

ret.append(row) # Add the row to the return table

return ret

"""

Performs the Naive Bayes Classifier to the Titanic Dataset

"""

def naive\_bayes\_part(table):

from tabulate import tabulate

# Builds a Naive Bayes classifier

nb\_classifier = build\_naive\_bayes\_classifier(table, [SURVIVED])

# Test using step2 approach from HW3

print '==========================================='

print 'TEST 3: Naive Bayes Survival Classifier (Titanic)'

print '==========================================='

total\_tests = 10

random\_instances = [random.randint(0, len(table) - 1) for \_ in range(total\_tests)]

correct = 0

for index in random\_instances:

predicted = guess\_label\_categorical(nb\_classifier, table[index], [CLASS,AGE,SEX], SURVIVED)

print ' predicted:', predicted, ' actual:', table[index][SURVIVED]

if predicted == table[index][SURVIVED]:

correct += 1

# Print statistics from step3 of HW3

accuracy = float(correct) / total\_tests

print ''

print ' For k=10 Random Subsampling:'

print ' accuracy:', accuracy, ', error\_rate:', (1 - accuracy)

print ''

# Confusion matrix is a dictionary of dictionaries

matrix = {'yes' : {'yes' : 0, 'no' : 0},

'no' : {'yes' : 0, 'no' : 0}}

for row in table:

predicted = guess\_label\_categorical(nb\_classifier, row, [CLASS,AGE,SEX], SURVIVED)

matrix[row[SURVIVED]][predicted] += 1

tabbed\_table = []

for i in ['yes','no']:

total = sum([matrix[i][j] for j in ['yes','no']]) \* 1.0

tabbed\_table.append([i] +

[matrix[i][j] for j in ['yes','no']] +

[total] +

[100. \* matrix[i][i] / (total if total != 0 else 1)])

print tabulate(tabbed\_table, headers = ['SURVIVED', 'yes', 'no', 'Total', 'Recognition (%)'])

numerator = matrix['yes']['yes'] + matrix['no']['no']

total = numerator + matrix['yes']['no'] + matrix['no']['yes']

accuracy = float(numerator) / total

print ' accuracy:', accuracy, ' error\_rate:', (1 - accuracy)

"""

Partitions the passed in table into k folds.

"""

def partition\_into\_folds(table, k, class\_index):

folds = [[]] \* k

table.sort(key=lambda x: x[class\_index]) # Sort by class\_index

cur\_fold = 0

for row in table:

folds[cur\_fold].append(row)

cur\_fold = (cur\_fold + 1) % k

return folds

"""

Creates a testing set and training set from a list of folds, on the index of the test fold.

"""

def create\_test\_and\_train\_from\_folds(folds, index):

training\_set = []

# Build a new training set excluding the current fold (index)

for j in range(len(folds)):

if index != j:

training\_set += folds[j]

return (training\_set, folds[index])

"""

Takes a table and an index, calculates the minimum and

divisor used to normalize a value, then returns those.

"""

def get\_normalize\_factors\_from\_titanic(table, index):

d = {'crew' : 0., 'first' : 0.33, 'second' : 0.66, 'third' : 1.,

'child' : 0., 'adult' : 1.,

'female' : 0., 'male' : 1.}

mini = min([d[row[index]] for row in table])

maxi = max([d[row[index]] for row in table])

return (mini, (maxi - mini) \* 1.0)

"""

Takes a training set and an instance, and returns a label

"""

def guess\_label\_using\_stratified\_k\_fold(training, instance, k):

import operator

import math

neighbors = []

clsmin, clsminmax = get\_normalize\_factors\_from\_titanic(training, CLASS)

agemin, ageminmax = get\_normalize\_factors\_from\_titanic(training, AGE)

sexmin, sexminmax = get\_normalize\_factors\_from\_titanic(training, SEX)

def difference(row1, row2, index, mini, minmax):

d = {'crew' : 0., 'first' : 0.33, 'second' : 0.66, 'third' : 1.,

'child' : 0., 'adult' : 1.,

'female' : 0., 'male' : 1.}

return ((d[row1[index]] - mini) / minmax) - ((d[row2[index]] - mini) / minmax)

for row in training:

# Calculate Euclidean distance on normalized values

distance = 0.0

distance += difference(row, instance, CLASS, clsmin, clsminmax) \*\* 2

distance += difference(row, instance, AGE, agemin, ageminmax) \*\* 2

distance += difference(row, instance, SEX, sexmin, sexminmax) \*\* 2

root = math.sqrt(distance)

neighbors.append([root, row[SURVIVED]]) # Add the distance and label to neighbors

neighbors.sort(key=lambda x: x[0]) # Sort by root

# Determine the most occuring label

labels = {}

for i in range(k):

labels[neighbors[i][1]] = labels.get(neighbors[i][1], 0) + 1

""" max item from dictionary from: http://stackoverflow.com/questions/268272/getting-key-with-maximum-value-in-dictionary """

return max(labels.iteritems(), key=operator.itemgetter(1))[0]

"""

Performs the k-fold classifier on the Titanic Dataset

"""

def k\_fold\_part(table):

from tabulate import tabulate

print '==========================================='

print 'TEST 3: Stratified Survival Classifier (Titanic)'

print '==========================================='

folds = partition\_into\_folds(table, 10, SURVIVED) # Partitions the data into k folds

matrix = {'yes' : {'yes' : 0, 'no' : 0},

'no' : {'yes' : 0, 'no' : 0}}

for i in range(10):

training\_set, test\_set = create\_test\_and\_train\_from\_folds(folds, i)

# Test the current fold against the training set

for instance in test\_set:

predicted = guess\_label\_using\_stratified\_k\_fold(training\_set, instance, 10)

matrix[instance[SURVIVED]][predicted] += 1

# Print out the confusion matrix

tabbed\_table = []

for i in ['yes','no']:

total = sum([matrix[i][j] for j in ['yes','no']]) \* 1.0

tabbed\_table.append([i] +

[matrix[i][j] for j in ['yes','no']] +

[total] +

[100. \* matrix[i][i] / (total if total != 0 else 1)])

print tabulate(tabbed\_table, headers = ['SURVIVED', 'yes', 'no', 'Total', 'Recognition (%)'])

numerator = matrix['yes']['yes'] + matrix['no']['no']

total = numerator + matrix['yes']['no'] + matrix['no']['yes']

accuracy = float(numerator) / total

print ' accuracy:', accuracy, ' error\_rate:', (1 - accuracy)

"""

Step3:

"""

def step3(table):

naive\_bayes\_part(table)

k\_fold\_part(table)

""" Main function for this file """

def main():

# Some index references to use globally

global MPG

MPG = 0

global CYLINDERS

CYLINDERS = 1

global DISPLACEMENT

DISPLACEMENT = 2

global HORSEPOWER

HORSEPOWER = 3

global WEIGHT

WEIGHT = 4

global ACCELERATION

ACCELERATION = 5

global MODEL\_YEAR

MODEL\_YEAR = 6

global ORIGIN

ORIGIN = 7

global CAR\_NAME

CAR\_NAME = 8

global MSRP

MSRP = 9

global LABEL

LABEL = 10

global CLASS

CLASS = 0

global AGE

AGE = 1

global SEX

SEX = 2

global SURVIVED

SURVIVED = 3

auto\_table = read\_csv('auto-data.txt') # Read in the automotive data

auto\_table = clean\_auto\_data(auto\_table) # Cleans the table

step1(copy.deepcopy(auto\_table))

step2(copy.deepcopy(auto\_table))

titanic\_table = read\_csv('titanic.txt')[1:]

step3(copy.deepcopy(titanic\_table))

""" Entry point """

if \_\_name\_\_ == '\_\_main\_\_':

main()