

2019BITE072 ANAS MURTAZA

2019BITE051 KAPIL KUMAR

2019BITE027 ADITYA VIBHUTE

Perform the following steps as discussed in the class:

Step1: Choose a data set from the following websites (at least 5 variables) and define a problem statement (Last date : 03/06/2022).

<https://www.kaggle.com/datasets> <https://archive.ics.uci.edu/ml/datasets.php>

Step2: Preprocess the data set

Step3: Define a neural network architecture for your problem statement (with proper reasons)

Step4: Train the neural network (with 70% dataset)

Step5: Test the neural network (with 30% dataset)

PROBLEM STATEMENT

Bob has started his own mobile company. He wants to give tough fight to big companies like Apple,Samsung etc.

He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.

Bob wants to find out some relation between features of a mobile phone(eg:- RAM,Internal Memory etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.

In this problem you do not have to predict actual price but a price range indicating how high the price is

This kind of prediction will help companies estimate price of mobiles to give tough competition to other mobile manufacturer

Also it will be usefull for Consumers to verify that they are paying best price for a mobile

Weka Explorer

Preprocess Classify Cluster Associate **Select attributes** Visualize

Attribute Evaluator
Choose **CfsSubsetEval -P 1 -E 1**

Search Method
Choose **BestFirst -D 1 -N 5**

Attribute Selection Mode
☒ Use full training set
☐ Cross-validation Folds: 10 Seed: 1

No class

Start Stop

Result list (right-click for options)
18:21:26 - BestFirst + CfsSubsetEval

Attribute selection output

```

=====
price_range
Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
  Best first.
  Start set: no attributes
  Search direction: forward
  Stale search after 5 node expansions
  Total number of subsets evaluated: 145
  Merit of best subset found: 0.926

Attribute Subset Evaluator (supervised, Class (numeric): 21 price_range):
  CFS Subset Evaluator
  Including locally predictive attributes

Selected attributes: 1,7,9,11,13,14,16 : 7
  battery_power
  int_memory
  mobile_wt
  pc
  px_width
  ram
  sc_w
  
```

FEATURE SELECTION:

Search Method: Best first. Start set: no attributes Search direction: forward Stale search after 5 node expansions Total number of subsets evaluated: 145 Merit of best subset found: 0.926

Attribute Subset Evaluator (supervised, Class (numeric): 21 price_range): CFS Subset Evaluator Including locally predictive attributes

Selected attributes: 1,7,9,11,13,14,16 : 7

```

battery_power
int_memory
mobile_wt
pc
px_width
ram
sc_w

```

```

from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
#we have used only three libraries RANDOM, MATH and CSV

# Loading a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Convert string column to integer if required
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

```

```

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(neuron['output'] - expected[j])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] -= l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] -= l_rate * neuron['delta']

# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        for row in train:
            outputs = forward_propagate(network, row)

```

```

        expected = [0 for i in range(n_outputs)]
        expected[row[-1]] = 1
        backward_propagate_error(network, expected)
        update_weights(network, row, l_rate)

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{ 'weights': [random() for i in range(n_inputs + 1)] } for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{ 'weights': [random() for i in range(n_hidden + 1)] } for i in range(n_outputs)]
    network.append(output_layer)
    return network

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = len(set([row[-1] for row in train]))
    network = initialize_network(n_inputs, n_hidden, n_outputs)
    train_network(network, train, l_rate, n_epoch, n_outputs)
    predictions = list()
    for row in test:
        prediction = predict(network, row)
        predictions.append(prediction)
    return(predictions)

# TEST BACPROP ON MOBILE CLASSIFICATION DATASET
seed(1) #used in random number generation
# load and prepare data
filename = '/content/f_selected.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
#feature selection

# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# normalize input variables
minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
n_folds = 10
l_rate = 0.3
n_epoch = 500
n_hidden = 6
scores = evaluate_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
Scores: [93.0, 92.0, 88.0, 90.0, 90.5, 90.0, 91.5, 90.5, 90.0, 88.5]
Mean Accuracy: 90.400%

```

In this data:

id:ID

battery_power:Total energy a battery can store in one time measured in mAh

blue:Has bluetooth or not

clock_speed:speed at which microprocessor executes instructions

dual_sim:Has dual sim support or not

fc:Front Camera mega pixels

four_g:Has 4G or not

int_memory:Internal Memory in Gigabytes

m_dep:Mobile Depth in cm

mobile_wt:Weight of mobile phone

n_cores:Number of cores of processor

pc:Primary Camera mega pixels

px_height:Pixel Resolution Height

px_width:Pixel Resolution Width

ram:Random Access Memory in Megabytes

sc_h:Screen Height of mobile in cm

sc_w:Screen Width of mobile in cm

talk_time:longest time that a single battery charge will last when you are

three_g:Has 3G or not

touch_screen:Has touch screen or not

wifi:Has wifi or not

[Colab paid products](#) - [Cancel contracts here](#)

