6601 – Assignment 4: Machine Learning and Decision Trees

Instructor: Thad Starner TA: Daniel Kohlsdorf, Titilayo Craig

Weiren Wang(903076444) weirenwang@gatech.edu

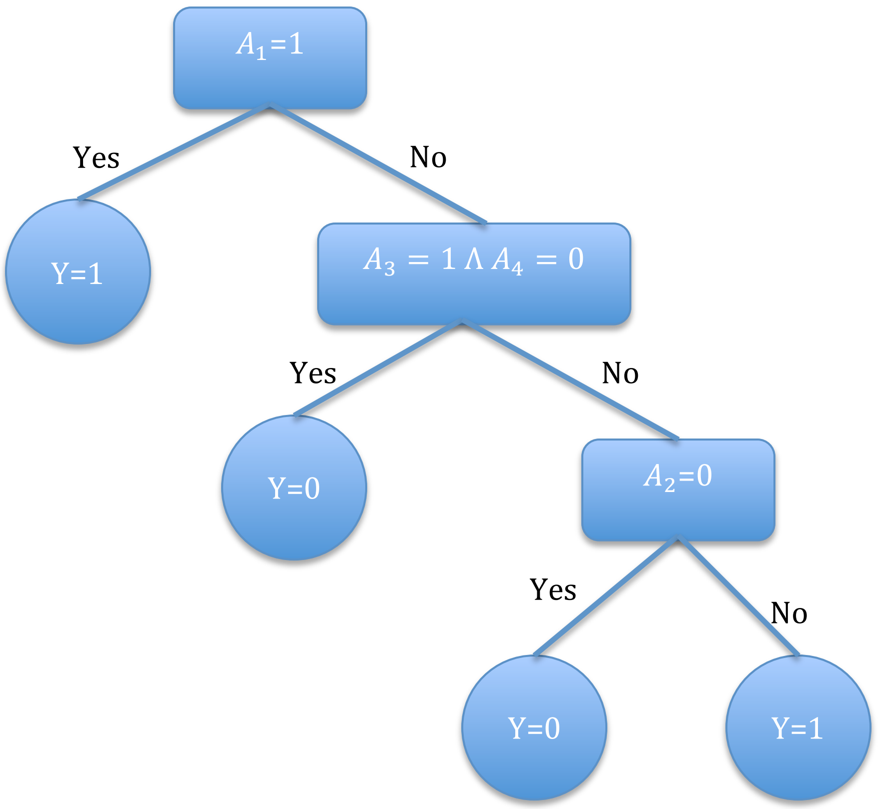
https://www.linkedin.com/in/weirenwang

https://github.com/JeffreyWeirenWang

Warm Up (20%)

Construct a decision list to classify the data below. Select tests to be as small as possible (in terms of attributes), breaking ties among tests with the same number of attributes by selecting the one that classifies the greatest number of examples correctly, If multiple tests have the same number of attributes and classify the same number of examples, then break the tie using attributes with lower index numbers (e.g., select An over A2).

Solution:

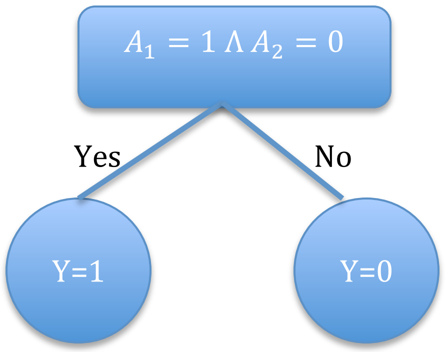


Practice (20%)

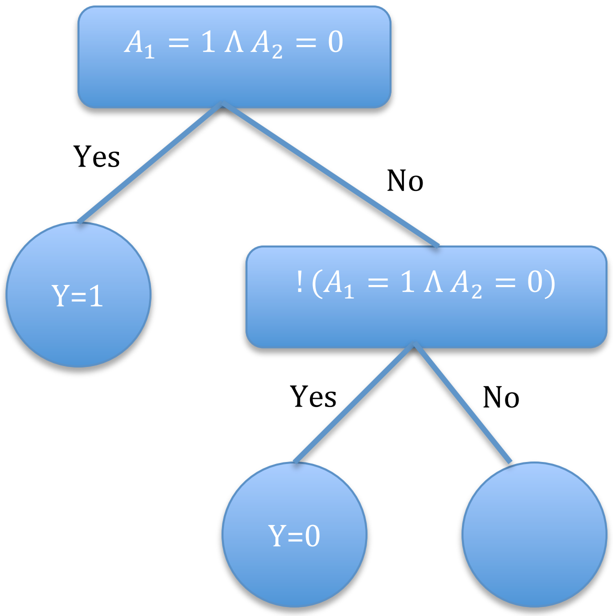
Prove that a decision list can representthesamefunctionasadecisiontreewhile using at most as many rules as there are leaves in the decision tree for that function. Give an example of a function represented by a decision list using strictly fewer rules than the number of leaves in a minimal-sized decision tree for that same function.

Solution: In decision tree, the path from root node to leaf node represents a classification by multiple conditions and logic conjunction. In decision list, it could have only one rule to correspond to path through the decision tree where the rule in the decision list has the test given by the logical conjunction. The output for the rule is the corresponding classification at the leaf. In this way, we could construct a decision list that captures the same function represented in the decision tree.

Examples: $A_1 = 1 \wedge A_2 = 0 \Longrightarrow \; y = 1$



Decision List



Decision Tree

From the comparison above, we could find that the Decision List need only two leaves while the Decision Tree need three leaves.

Implementation (20%)

Decision Tree

A decision tree is a flowchart like structure in which internal node represents a "test" on an attribute, each branch represents the outcome of the test and each leaf node represents a class label. The paths from root to leaf represents a classification rules.

In decision tree implementation, I designed several important functions:

1. Information Gain
   The function measures the ranking of entropy results from splitting a set of examples based on attributes.
   $$B(q) = -(q log_2 q + (1-q) log_2 (1-q))$$
   $$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} * B(\frac{p_k}{p_k + n_k})$$
   $$Gain(A) = B\left(\frac{p}{p+n}\right) - Remainder(A)$$ 2.The
2. Decision-Tree-Learning

```
1.ID3 (Examples, Target_Attribute, Attributes)
2.        Create a root node for the tree
3.        If all examples are positive, Return the single-node tree Root, with label = +.
4.        If all examples are negative, Return the single-node tree Root, with label = -.
5.         If number of predicting attributes is empty, then Return the single node tree
          Root,with label = most common value of the target attribute in the examples.
6.        Otherwise Begin
7.                 A ← The Attribute that best classifies examples.
8.                 Decision Tree attribute for Root = A.
9.                  For each possible value, v_i, of A,
10.                         Add a new tree branch below Root, corresponding to the test A = v_i.
11.                                  Let Examples(v_i) be the subset of examples that have the value v_i for A
12.                                  If Examples(v_i) is empty
13.                                  Then below this new branch add a leaf node with label = most
                                     common target value in the examples
14.                                  Else below this new branch add the subtree ID3 (Examples(v_i),
                                     Target_Attribute, Attributes – {A})
15.       End
16.       Return Root
```

Random Forest

Random Forests are an combined learning method for classification that operate by constructing a multiple of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The path from leaf to node represent the classification.

```
1. Classify(node,V)
2.          Input: node from the decision tree, if node.attribute = j then the split is done on the j'th attribute
3.          Input: V a vector of M columns where Vj = the value of the j'th attribute.
4.          Output: label of V
5.          If node is a Leaf then
6.                      Return the value predicted by node
7.          Else
8.                      Let j = node.attribute
9.                      If j is categorical then
10 .                               Let v = Vj
11.                                Let childv = child node corresponding to the attribute's value v
12.                                Return Classify(childv,V)
13.                     Else j is real-valued
14.                                Let t = node.threshold (split threshold)
15.                                If Vj < t then
16.                                          Let childLO = child node corresponding to (<t)
17.                                          Return Classify(childLO,V)
18.                                Else
19.                                          Let childHI = child node corresponding to (>=t)
20.                                          Return Classify(childHI,V)
```

Decision Tree (Missing Attribute)

In the read world, sometimes the dataset will miss some attribute values. One way to handle this is to pretend that the example has all possible values for the attribute, but to weight each value according to it's frequencies among all of the examples that reach that node in the decision tree. The miss_attr_decision_tree function is based on decision tree. In implementation, we calculate the possibilities of attributes. Then, we assign the missing attribute value with the attribute with largest possibility.

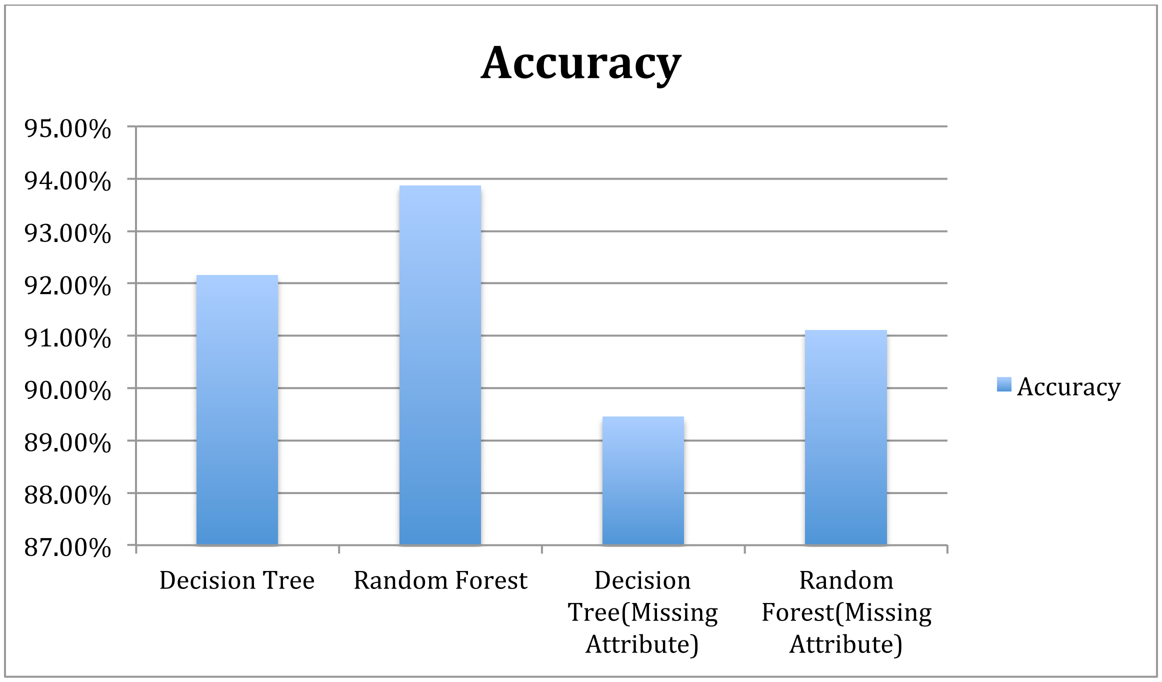
Random Forest (Missing Attribute)

In the missing attribute random forest version, instead of call decision tree function, we call miss_attr_decision tree function. This is the only revisement.

Cross Validation

In this test, we are going to use k-fold cross validation (k=10). We randomly separate the data into 10 parts. Each time, we select one part as the test data, another 9 parts as the training data. In the end, we calculate the average test error.

Result



Appendix

```
"""Learn to estimate functions from examples. (Chapters 18-20)"""

from utils import *
import copy, heapq, math, random
from collections import defaultdict, Counter


class DataSet:

  def __init__(self, examples=None, attrs=None, attrnames=None, target=-1,
          inputs=None, values=None, distance=mean_boolean_error,
          name='', source='', exclude=()):
    update(self, name=name, source=source, values=values, distance=distance)
    # Initialize .examples from string or list or data directory
    if isinstance(examples, str):
      self.examples = parse_csv(examples)
    elif examples is None:
      self.examples = parse_csv(DataFile(name+'.csv').read())
    else:
      self.examples = examples
    # Attrs are the indices of examples, unless otherwise stated.
```

```
        if not attrs and self.examples:
            attrs = range(len(self.examples[0]))
        self.attrs = attrs
        # Initialize .attrnames from string, list, or by default
        if isinstance(attrnames, str):
            self.attrnames = attrnames.split()
        else:
            self.attrnames = attrnames or attrs
        self.setproblem(target, inputs=inputs, exclude=exclude)

    def setproblem(self, target, inputs=None, exclude=()):
        self.target = self.attrnum(target)
        exclude = map(self.attrnum, exclude)
        if inputs:
            self.inputs = removeall(self.target, inputs)
        else:
            self.inputs = [a for a in self.attrs
                           if a != self.target and a not in exclude]
        if not self.values:
            self.values = map(unique, zip(*self.examples))
        self.check_me()

    def check_me(self):
        "Check that my fields make sense."
        assert len(self.attrnames) == len(self.attrs)
        assert self.target in self.attrs
        assert self.target not in self.inputs
        assert set(self.inputs).issubset(set(self.attrs))
        map(self.check_example, self.examples)

    def add_example(self, example):
        "Add an example to the list of examples, checking it first."
        self.check_example(example)
        self.examples.append(example)

    def check_example(self, example):
        "Raise ValueError if example has any invalid values."
        if self.values:
            for a in self.attrs:
                if example[a] not in self.values[a]:
                    raise ValueError('Bad value %s for attribute %s in %s' %
                                     (example[a], self.attrnames[a], example))

    def attrnum(self, attr):
        "Returns the number used for attr, which can be a name, or -n .. n-1."
        if attr < 0:
```

```
        return len(self.attrs) + attr
      elif isinstance(attr, str):
        return self.attrnames.index(attr)
      else:
        return attr

   def sanitize(self, example):
     "Return a copy of example, with non-input attributes replaced by None."
     return [attr_i if i in self.inputs else None
          for i, attr_i in enumerate(example)]

   def __repr__(self):
      return '<DataSet(%s): %d examples, %d attributes>' % (
        self.name, len(self.examples), len(self.attrs))

#______________________________________________________________________________

def parse_csv(input, delim=','):
   r"""Input is a string consisting of lines, each line has comma-delimited
   fields.  Convert this into a list of lists.  Blank lines are skipped.
   Fields that look like numbers are converted to numbers.
   The delim defaults to ',' but '\t' and None are also reasonable values.
   >>> parse_csv('1, 2, 3 \n 0, 2, na')
   [[1, 2, 3], [0, 2, 'na']]
   """
   lines = [line for line in input.splitlines() if line.strip()]
   return [map(num_or_str, line.split(delim)) for line in lines]

class DecisionFork:
   """A fork of a decision tree holds an attribute to test, and a dict
   of branches, one for each of the attribute's values."""

   def __init__(self, attr, attrname=None, branches=None):
      "Initialize by saying what attribute this node tests."
      update(self, attr=attr, attrname=attrname or attr,
          branches=branches or {})

   def __call__(self, example):
      "Given an example, classify it using the attribute and the branches."
      attrvalue = example[self.attr]
      return self.branches[attrvalue](example)

   def add(self, val, subtree):
      "Add a branch.  If self.attr = val, go to the given subtree."
      self.branches[val] = subtree
```

```
    def display(self, indent=0):
        name = self.attrname
        print 'Test', name
        for (val, subtree) in self.branches.items():
            print ' '*4*indent, name, '=', val, '==>',
            subtree.display(indent+1)

    def __repr__(self):
        return ('DecisionFork(%r, %r, %r)'
                % (self.attr, self.attrname, self.branches))

class DecisionLeaf:
    "A leaf of a decision tree holds just a result."

    def __init__(self, result):
        self.result = result

    def __call__(self, example):
        return self.result

    def display(self, indent=0):
        print 'RESULT =', self.result

    def __repr__(self):
        return repr(self.result)

#______________________________________________________________________________

def DecisionTreeLearner(dataset):
    "[Fig. 18.5]"

    target, values = dataset.target, dataset.values

    def decition_tree_missattr_learning(examples, attrs, parent_examples=()):
            #compute the majority attr
            popular_attrs = dict()
            for a in attrs:
                    the_list = []
                    for e in examples:
                            "The majority couldn't be empty, the pretend attr wouldn't
be count"
                            the_list.append(e[a])
                    majority = Counter(the_list).most_common(1)
                    #??? mapping object a or value a
                    popular_attrs[a] = majority
        "assign attr for missing data"
        examples2 = copy.deepcopy(examples)
        for e in examples2:
            for a in attrs:
                    # empty means the attr is missing
                    if examples2[a] == 'empty':
                         exammples2[a] = popular_attrs[a]

        if len(examples) == 0:
             return plurality_value(parent_examples)
        #one possiblility is that the attr is empty or major.
        elif all_same_class(examples2):
             return DecisionLeaf(examples[0][target])
        elif len(attrs) == 0:
             return plurality_value(examples)
        else:
             #choose attr
             A = choose_attribute(attrs, examples2)
             #reassign attr on raw data
             for e in examples:
                  if examples[A] == 'empty':
                       examples[A] = popular_attrs[A]
             tree = DecisionFork(A, dataset.attrnames[A])
             for(v_k, exs) in split_by(A, examples):
                  subtree = decision_tree_learning(exs, removeall(A, attrs),
examples)
                  tree.add(v_k, subtree)
             return tree



  def decision_tree_learning(examples, attrs, parent_examples=()):
    if len(examples) == 0:
      return plurality_value(parent_examples)
    elif all_same_class(examples):
      return DecisionLeaf(examples[0][target])
    elif len(attrs) == 0:
      return plurality_value(examples)
    else:
      A = choose_attribute(attrs, examples)
      tree = DecisionFork(A, dataset.attrnames[A])
      for (v_k, exs) in split_by(A, examples):
        subtree = decision_tree_learning(
          exs, removeall(A, attrs), examples)
```

```
            tree.add(v_k, subtree)
        return tree

    def plurality_value(examples):
        """Return the most popular target value for this set of examples.
        (If target is binary, this is the majority; otherwise plurality.)"""
        popular = argmax_random_tie(values[target],
                              lambda v: count(target, v, examples))
        return DecisionLeaf(popular)

    def count(attr, val, examples):
        return count_if(lambda e: e[attr] == val, examples)

    def all_same_class(examples):
        "Are all these examples in the same target class?"
        class0 = examples[0][target]
        return all(e[target] == class0 for e in examples)

    def choose_attribute(attrs, examples):
        "Choose the attribute with the highest information gain."
        return argmax_random_tie(attrs,
                             lambda a: information_gain(a, examples))

    def information_gain(attr, examples):
        "Return the expected reduction in entropy from splitting by attr."
        def I(examples):
            return information_content([count(target, v, examples)
                               for v in values[target]])
        N = float(len(examples))
        remainder = sum((len(examples_i) / N) * I(examples_i)
                    for (v, examples_i) in split_by(attr, examples))
        return I(examples) - remainder

    def split_by(attr, examples):
        "Return a list of (val, examples) pairs for each val of attr."
        return [(v, [e for e in examples if e[attr] == v])
            for v in values[attr]]
    #return decition_tree_missattr_learning(dataset.examples, dataset.inputs)
    return decision_tree_learning(dataset.examples, dataset.inputs)

def information_content(values):
    "Number of bits to represent the probability distribution in values."
    print "values"
    print values
    if not removeall(0, values):
            return 0
    probabilities = normalize(removeall(0, values))
    return sum(-p * math.log(p,2) for p in probabilities)
    """
    result = 0;
    for p in probabilities:
            if p!=1 or  p!=0:
                    result += -p * log2(p)
    return result
def removeall(val, the_list):
            return[value for value in the_list if value != val]
    """
#________________________________________________________________________

### A decision list is implemented as a list of (test, value) pairs.

def DecisionListLearner(dataset):
    """[Fig. 18.11]"""

    def decision_list_learning(examples):
        if not examples:
            return [(True, False)]
        t, o, examples_t = find_examples(examples)
        if not t:
            raise Failure
        return [(t, o)] + decision_list_learning(examples - examples_t)

    def find_examples(examples):
        """Find a set of examples that all have the same outcome under
        some test. Return a tuple of the test, outcome, and examples."""
        unimplemented()

    def passes(example, test):
        "Does the example pass the test?"
        unimplemented()

    def predict(example):
        "Predict the outcome for the first passing test."
        for test, outcome in predict.decision_list:
            if passes(example, test):
                return outcome
    predict.decision_list = decision_list_learning(set(dataset.examples))

    return predict
```

```
def random_foresti_train(learner, dataset, examples=None,tree_num = 1):
        "majority vote"
        "might choose training data instead of dataset examples"
        raw_data = dataset
        trees = []
        for i in range(tree_num):

        traindata=random.choice(raw_data.examples,n=len(rawdata.examples),repla
ce=true)
                dataset.examples = traindata
                trees.append(learner(dataset))

        return trees


def random_forest_test(trees, example):
        results = []
        for t in trees:
                results.append(t(dataset.sanitize(example)))
                "t(example) is using tree and example and get result"
        majority = Counter(results).most_common(1)
        return majority




def test(predict, dataset, examples=None, verbose=0):
    "training:dataset    testing:examples"
    "Return the proportion of the examples that are correctly predicted."
    if examples is None: examples = dataset.examples
    if len(examples) == 0: return 0.0
    right = 0.0
    for example in examples:
        desired = example[dataset.target]
        output = predict(dataset.sanitize(example))
        if output == desired:
            right += 1
            if verbose >= 2:
               print '   OK: got %s for %s' % (desired, example)
        elif verbose:
            print 'WRONG: got %s, expected %s for %s' % (
               output, desired, example)
    return right / len(examples)

def train_and_test(learner, dataset, start, end):
    """Reserve dataset.examples[start:end] for test; train on the remainder.
    Return the proportion of examples correct on the test examples."""
    examples = dataset.examples
    try:
        dataset.examples = examples[:start] + examples[end:]
        return test(learner(dataset), dataset, examples[start:end])
    finally:
        dataset.examples = examples

def cross_validation(learner, dataset, k=10, trials=1):
    """Do k-fold cross_validate and return their mean.
    That is, keep out 1/k of the examples for testing on each of k runs.
    Shuffle the examples first; If trials>1, average over several shuffles."""
    if k is None:
        k = len(dataset.examples)
    if trials > 1:
        return mean([cross_validation(learner, dataset, k, trials=1)
                    for t in range(trials)])
    else:
        n = len(dataset.examples)
        random.shuffle(dataset.examples)
        return mean([train_and_test(learner, dataset, i*(n/k), (i+1)*(n/k))
                    for i in range(k)])

def leave1out(learner, dataset):
    "Leave one out cross-validation over the dataset."
    return cross_validation(learner, dataset, k=len(dataset.examples))

def learningcurve(learner, dataset, trials=10, sizes=None):
    if sizes is None:
        sizes = range(2, len(dataset.examples)-10, 2)
    def score(learner, size):
        random.shuffle(dataset.examples)
        return train_and_test(learner, dataset, 0, size)
    return [(size, mean([score(learner, size) for t in range(trials)]))
            for size in sizes]

#______________________________________________________________________________
# The rest of this file gives datasets for machine learning problems.

orings = DataSet(name='orings', target='Distressed',
                attrnames="Rings Distressed Temp Pressure Flightnum")


zoo = DataSet(name='zoo', target='type', exclude=['name'],
              attrnames="name hair feathers eggs milk airborne aquatic " +
              "predator toothed backbone breathes venomous fins legs tail " +
            "domestic catsize type")


iris = DataSet(name="iris", target="class",
            attrnames="sepal-len sepal-width petal-len petal-width class")

#______________________________________________________________________________
# The Restaurant example from Fig. 18.2

def RestaurantDataSet(examples=None):
   "Build a DataSet of Restaurant waiting examples. [Fig. 18.3]"
   return DataSet(name='restaurant', target='Wait', examples=examples,
              attrnames='Alternate Bar Fri/Sat Hungry Patrons Price '
               + 'Raining Reservation Type WaitEstimate Wait')

restaurant = RestaurantDataSet()

def T(attrname, branches):
   branches = dict((value, (child if isinstance(child, DecisionFork)
                     else DecisionLeaf(child)))
               for value, child in branches.items())
   return DecisionFork(restaurant.attrnum(attrname), attrname, branches)


def Majority(k, n):
   """Return a DataSet with n k-bit examples of the majority problem:
   k random bits followed by a 1 if more than half the bits are 1, else 0."""
   examples = []
   for i in range(n):
      bits = [random.choice([0, 1]) for i in range(k)]
      bits.append(int(sum(bits) > k/2))
      examples.append(bits)
   return DataSet(name="majority", examples=examples)

def Parity(k, n, name="parity"):
   """Return a DataSet with n k-bit examples of the parity problem:
   k random bits followed by a 1 if an odd number of bits are 1, else 0."""
   examples = []
   for i in range(n):
      bits = [random.choice([0, 1]) for i in range(k)]
      bits.append(sum(bits) % 2)
      examples.append(bits)
   return DataSet(name=name, examples=examples)

def Xor(n):
   """Return a DataSet with n examples of 2-input xor."""
```

```
    return Parity(2, n, name="xor")

def ContinuousXor(n):
    "2 inputs are chosen uniformly from (0.0 .. 2.0]; output is xor of ints."
    examples = []
    for i in range(n):
        x, y = [random.uniform(0.0, 2.0) for i in '12']
        examples.append([x, y, int(x) != int(y)])
    return DataSet(name="continuous xor", examples=examples)

#______________________________________________________________________________

def compare(algorithms=[PluralityLearner, NaiveBayesLearner,
                        NearestNeighborLearner, DecisionTreeLearner],
            datasets=[iris, orings, zoo, restaurant, SyntheticRestaurant(20),
                      Majority(7, 100), Parity(7, 100), Xor(100)],
            k=10, trials=1):
    """Compare various learners on various datasets using cross-validation.
    Print results as a table."""
    print_table([[a.__name__.replace('Learner','')] +
                 [cross_validation(a, d, k, trials) for d in datasets]
                 for a in algorithms],
                header=[''] + [d.name[0:7] for d in datasets], numfmt='%.2f')

def BankDataSet(examples=None):
        return DataSet(name='bank', target = 'class',
                       attrnames='variance skewness curtosis entropy class')


bank = BankDataSet()
if __name__=='__main__':
        random.seed(437)
        bank_tree = DecisionTreeLearner(bank)
        bank_tree.display()
        restaurant_tree = DecisionTreeLearner(restaurant)
        #restaurant_tree.display()
        print cross_validation(DecisionTreeLearner, bank, k=10, trials=10)
```

References:

AIMA-python code: https://code.google.com/p/aima-python/
Decision Tree wiki: http://en.wikipedia.org/wiki/Decision_tree
Random forest wiki: http://en.wikipedia.org/wiki/Random_forest