CS 6375 HW 4:

Sample_3_MLC_2022

Parker Whitehead

Part-1: Build solver

We will load the data and use it for building our solver

Download data

```
!wget https://www.ics.uci.edu/~dechter/uaicompetition/2022/TuningBenchmarks/MLC.zip
!unzip /content/MLC.zip
```

```
--2022-12-05 08:41:04--  https://www.ics.uci.edu/~dechter/uaicompetition/2022/TuningBenchmarks/MLC.zip
Resolving www.ics.uci.edu (www.ics.uci.edu)... 128.195.1.88
Connecting to www.ics.uci.edu (www.ics.uci.edu)|128.195.1.88|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 51653514 (49M) [application/zip]
Saving to: ‘MLC.zip’

MLC.zip             100%[===================>]  49.26M  60.8MB/s    in 0.8s

2022-12-05 08:41:05 (60.8 MB/s) - ‘MLC.zip’ saved [51653514/51653514]

Archive:  /content/MLC.zip
  inflating: MLC/Sample_1_MLC_2022.data
  inflating: MLC/Sample_1_MLC_2022.uai
  inflating: MLC/Sample_2_MLC_2022.data
  inflating: MLC/Sample_2_MLC_2022.uai
  inflating: MLC/Sample_3_MLC_2022.data
  inflating: MLC/Sample_3_MLC_2022.uai
  inflating: MLC/Sample_4_MLC_2022.data
  inflating: MLC/Sample_4_MLC_2022.uai
```

Import libraries

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

Load data

The data consists of three sets of variables:

1. Evidence (observed) variables - X
2. Hidden variables - H
3. Query variables - Y

The data loader class *Data* reads the data and partitions it accordingly.

Helper functions:

`convertToXY()` : This function returns (X, Y) from the .data file

```
class Data:
  #fpath: File path of the .data file

  #self.evid_var_ids: Contains the indices of the observed variables
  #self.query_var_ids: Contains the indices of the query variables
  #self.hidden_var_ids: Contains the indices of the hidden variables

  #self.evid_assignments: Assignments to evid variables
  #self.query_assignments: Assignments to query variables
```

```
    #self.weights: Pr(e, q)
    def __init__(self, fpath):

      f = open(fpath, "r")

      self.nvars = int(f.readline()) #1

      line = np.asarray(f.readline().split(), dtype=np.int32)#2
      self.evid_var_ids = line[1:]
      evid_indices = range(1, self.evid_var_ids.shape[0]*2, 2)

      line = np.asarray(f.readline().split(), dtype=np.int32) #3
      self.query_var_ids = line[1:]
      query_indices = range(self.evid_var_ids.shape[0]*2+1, (self.evid_var_ids.shape[0]+self.query_var_ids.shape[0])*2, 2)

      line = np.asarray(f.readline().split(), dtype=np.int32)#4
      self.hidden_var_ids = line[1:]

      line = f.readline()#5
      self.nproblems = int(f.readline())#6

      self.evid_assignments = []
      self.query_assignments = []
      self.weights = []
      for i in range(self.nproblems):
        line = np.asarray(f.readline().split(), dtype=float)
        self.evid_assignments.append(np.asarray(line[evid_indices], dtype=np.int32))
        self.query_assignments.append(np.asarray(line[query_indices], dtype=np.int32))
        self.weights.append(line[-1])
      self.evid_assignments = np.asarray(self.evid_assignments)
      self.query_assignments = np.asarray(self.query_assignments)
      self.weights = np.asarray(self.weights)

    def convertToXY(self):
      return (self.evid_assignments, self.query_assignments)

    def convertResults(self, query_predictions):
      out = np.zeros((query_predictions.shape[0], 1+2*self.query_var_ids.shape[0]), dtype=int)
      out[:, 2::2] = query_predictions[:, :]
      out[:, 1::2] = self.query_var_ids
      out[:, 0] = self.query_var_ids.shape[0]
      return out

data_directory = '/content/MLC/'
dname = 'Sample_3_MLC_2022'


f =open(data_directory+dname+'.data','r')
nvars = int(f.readline())
line = np.asarray(f.readline().split(), dtype=np.int32)


f =open(data_directory+dname+'.data','r')
x=f.readlines()


len(x)
```

```
10006
```

```
evid_var_ids = line[1:]
evid_var_ids.shape
```

```
(358,)
```

```
data = Data(data_directory+dname+'.data')


#Getting Evidence and Query data into X, y

X, y = data.convertToXY()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

Defining UAI Helper Classes

These classes utilize the fact that we are aware of the Markov Network that generates our dataset. As such, we can calculate the probability of any given q based on all of our known X values, using the Markov Network.

```
import numpy as np
from dataclasses import dataclass

@dataclass
class UAIFunction:
    # Number of variables participating in the function
    size: int
    # List of variables s.t. that the index is the order of the variables.
    indexes: list()
    # List of values for each permutation of variable values.
    values: list()

class UAIHelper:
    def __init__(self, fpath):
        with open(fpath,'r') as f:
            # Ignore type; will always be markov, for our case.
            f.readline()
            self.num_vars = int(f.readline())
            # according to prof, variables will always be binary, so we can ignore 3rd line.
            f.readline()

            # Ignore whitespace line.
            f.readline()
            self.num_functions = int(f.readline())
            # Each index contains a set!
            self.var_function_participation = np.array([set() for _ in range(self.num_vars)])
            self.functions = np.array([UAIFunction(0,[],[]) for _ in range(self.num_functions)]).astype(type(UAIFunction(0,[],[])))

            for i in range(self.num_functions):
                line = f.readline().replace('\n',' ').split(' ')
                vars = int(line[0])
                indexes = []
                for j in range(vars):
                    indexes.append(int(line[1+j]))
                    self.var_function_participation[indexes[j]].add(i)
                self.functions[i].size = vars
                self.functions[i].indexes = indexes

            f.readline()

            for i in range(self.num_functions):
                line = f.readline().replace('\n',' ').split(' ')
                num_values = int(line[0])
                values = []
                for j in range(num_values):
                    values.append(float(line[1+j]))
                self.functions[i].values = values

    def createDataMatrix(self, X, Q_size, X_indexes, Q_indexes):

        # create set of all included X indexes.
        all_x_indexes = set()
        for x_i in X_indexes:
            all_x_indexes.add(x_i)

        # create array of var name to index
        name_to_index = np.zeros(self.num_vars).astype(int)
        for index, value in enumerate(X_indexes):
            name_to_index[value] = index

        # create new matrix of size (x.shape[0], x.shape[1]+q.shape[1])
        new_X = np.zeros((X.shape[0],X.shape[1]+Q_size),dtype=float)
        # for each row:
        for i in range(X.shape[0]):
            # fill in all begining entries with x.
            for j in range(X.shape[1]):
                new_X[i,j] = X[i,j]

            # for each q in Q:
            for j in range(Q_size):
                q_index = Q_indexes[j]
                # for every function q is a part of:
                weight_true = 1
```

```
                weight_false = 1
                for function_index in self.var_function_participation[q_index]:
                    pass
                    # calculate weight multiplication sum of true & false
                    current_function = self.functions[function_index]

                    variable_table_index = 0
                    q_index_modifier = 0

                    # Find the index of the false state & the modifier to get the true state.
                    for counter, f_variable_index in enumerate(current_function.indexes):
                        # if all variable participants present, use; otherwise, ignore function.
                        if f_variable_index not in all_x_indexes and f_variable_index != q_index:
                            continue # if you don't have sufficient metrix to calculate, ignore function
                        if f_variable_index != q_index and X[i,name_to_index[int(f_variable_index)]] == 1:
                            variable_table_index += len(current_function.values) / 2**(counter+1)
                        elif f_variable_index == q_index:
                            q_index_modifier = len(current_function.values) / 2**(counter+1)

                    # then calculate the true and false weights for q.
                    weight_true*=current_function.values[int(variable_table_index + q_index_modifier)]
                    weight_false*=current_function.values[int(variable_table_index)]
                # Calculate weight_true / (weight_true + weight_false)
                weight_final = weight_true / (weight_true + weight_false)
                # Add this to the matrix.
                new_X[i,j + X.shape[1]] = weight_final

        # return new matrix and use!
        return new_X
```

Load UAI Helper and Create new input matrix for LogReg

```
uai_helper = UAIHelper(data_directory+dname+'.uai')

new_X_train = uai_helper.createDataMatrix(X_train, len(data.query_var_ids), data.evid_var_ids, data.query_var_ids)
```

Train solver: Logistic Regression

```
clf = MultiOutputClassifier(LogisticRegression(max_iter=1000)).fit(new_X_train, y_train)
```

Create new_X_test

```
new_X_test = uai_helper.createDataMatrix(X_test, len(data.query_var_ids), data.evid_var_ids, data.query_var_ids)
```

Predict Query Assignments

```
y_pred = clf.predict(new_X_test)
```

Store the query assignments in file - **Note this is the file to submit as the result**

```
results_in_format = data.convertResults(y_pred)
np.savetxt(X=results_in_format, delimiter=' ', fmt='%d', fname=data_directory+dname+'.pred')
```

Part-2: Test solver

Once we have trained the solver, we want to test how good it is.

For a given evidence $E = e$, let $Q = \hat{q}$ denote the solver prediction and $Q = q$ denote the ground truth value.

$$Err = log\frac{\prod_{i \in Data} Pr(e^{(i)}, q^{(i)})}{\prod_{i \in Data} Pr(e^{(i)}, \hat{q}^{(i)})}$$

Let $MaxErr$ denote the Err for a trivial solver. Then,

$$Score = max(0, 100(1 - \frac{Err}{MaxErr}))$$

Using Random Forests as the trivial solver

```
clf = MultiOutputClassifier(RandomForestClassifier(n_estimators = 10, max_depth=2)).fit(X_train, y_train)
```

```
y_trivial = clf.predict(X_test)
```

Load Variable Elimination Code

```
!git clone https://github.com/vkomaragiri/VEC.git
```

```
Cloning into 'VEC'...
remote: Enumerating objects: 111, done.
remote: Counting objects: 100% (72/72), done.
remote: Compressing objects: 100% (67/67), done.
^C
```

```
cd /content/VEC/
```

```
/content/VEC
```

```
!pip install igraph
!pip install Cython
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: igraph in /usr/local/lib/python3.8/dist-packages (0.10.2)
Requirement already satisfied: texttable>=1.6.2 in /usr/local/lib/python3.8/dist-packages (from igraph) (1.6.7)
ERROR: Operation cancelled by user
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: Cython in /usr/local/lib/python3.8/dist-packages (0.29.32)
ERROR: Operation cancelled by user
```

```
!python setup.py build_ext --inplace
```

```
running build_ext
```

Read the Markov network

[ ] ↳ *5 cells hidden*

Compute $log_{10}Pr(X, y)$

[ ] ↳ *1 cell hidden*

Compute error and score

```
def computeErr(true_ll, pred_ll):
  return np.sum(true_ll)-np.sum(pred_ll)
```

```
def computeScore(err, max_err):
  return np.max((0, 100*(1.0-err/max_err)))
```

```
y_pred = np.loadtxt(data_directory+dname+'.pred', dtype=int, delimiter=' ')[:, 1:][:, 1::2]
ntest = 10
lprob_true = computeLogProb(X_test[:ntest, :], y_test[:ntest, :])
lprob_pred = computeLogProb(X_test[:ntest, :], y_pred[:ntest, :])
lprob_trivial = computeLogProb(X_test[:ntest, :], y_trivial[:ntest, :])
```

```
err = computeErr(lprob_true, lprob_pred)
maxErr = computeErr(lprob_true, lprob_trivial)
```

```
print(err, maxErr)
```

```
0.03938934279949535 39.53018077858064
```

```
print("Score:", computeScore(err, maxErr))
```

```
Score: 99.9003562796231
```

```
print(err, maxErr)
```

```
0.03938934279949535 39.53018077858064
```

```
!pip freeze
```

```
requests-oauthlib==1.3.1
resampy==0.4.2
rpy2==3.5.5
rsa==4.9
scikit-image==0.18.3
scikit-learn==1.0.2
scipy==1.7.3
screen-resolution-extra==0.0.0
scs==3.2.2
seaborn==0.11.2
Send2Trash==1.8.0
setuptools-git==1.2
Shapely==1.8.5.post1
six==1.15.0
sklearn-pandas==1.8.0
smart-open==5.2.1
snowballstemmer==2.2.0
sortedcontainers==2.4.0
soundfile==0.11.0
spacy==3.4.3
spacy-legacy==3.0.10
spacy-loggers==1.0.3
Sphinx==1.8.6
sphinxcontrib-serializinghtml==1.1.5
sphinxcontrib-websupport==1.2.4
SQLAlchemy==1.4.44
sqlparse==0.4.3
srsly==2.4.5
statsmodels==0.12.2
sympy==1.7.1
tables==3.7.0
tabulate==0.8.10
tblib==1.7.0
tenacity==8.1.0
tensorboard==2.9.1
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.9.2
tensorflow-datasets==4.6.0
tensorflow-estimator==2.9.0
tensorflow-gcs-config==2.9.1
tensorflow-hub==0.12.0
tensorflow-io-gcs-filesystem==0.28.0
tensorflow-metadata==1.11.0
tensorflow-probability==0.17.0
termcolor==2.1.1
terminado==0.13.3
testpath==0.6.0
text-unidecode==1.3
textblob==0.15.3
texttable==1.6.7
thinc==8.1.5
threadpoolctl==3.1.0
tifffile==2021.11.2
toml==0.10.2
tomli==2.0.1
toolz==0.12.0
torch @ https://download.pytorch.org/whl/cu113/torch-1.12.1%2Bcu113-cp38-cp38-linux_x86_64.whl
```

✓ 0s completed at 3:15 AM