BayesExpert

Crowdsourcing the Community of Science

Easy for Scientists and AIs to express Scientific Facts

- Bayesian models are popular in Medicine but hard to make by hand
  - There are no Easy to Use Python models
- Represents the Medical Statistics of Systematic Reviews and Meta Analyses
- Experts can accept common assumptions or add exceptions and nonlinearities through rules
- Modular for crowdsourcing : Compilation time separating out of risk so a scientist doesn't have to worry about double counting and only has to worry about the subject of their own distributions
  - RR, Sensitivity and Specificity are easy to mine, and usually in abstracts.
- The same hand built nets can learn probabilities from data
  - Facilitates automatic finding of nonlinearities/exceptions to rules in automatic mining of scientific papers.
  - Experts can make small models that our software puts together
  - Our Generative Cooperative Network (GCN) can automatically compose models that help each other and exclude non consonant models.
  - BayesExpert is just one of the kinds of Generative Model that can be crowdsourced into our GCN
    - Neural generative models and Simulations are others
    - A good starting point for AI-DSL but non-generative models can be added as it and GCN mature

Rule Types

- The two basic Bayesian Rule types are discrete distribution and dependency, for the leaf nodes and the conditional probability tables of Bayesian nets.
- The dependency node expresses dependencies in the form of Relative Risk and Sensitivity / Specificity statistics from the medical literature.
- There are also probabilistic logic nodes that can be used to convert variable indicators into what is needed for studies
    - any_of (or), all_of (and), avg (finds the mean), and if_then_else can model any concept. (not, or the absence of a condition, is already in the Bayesian Rule types.
    - The rules use a convention that variable states are written negative to positive or left to right
    - Rules can express nonlinearities , create new indicators, or apply soft constraints

Rule Types (from https://github.com/Rejuve/bayesnet)

```
discreteDistribution = bayesianNetwork.discreteDistributions.add()
        discreteDistribution.name = "cough"
        variable = discreteDistribution.variables.add()
        variable.name = "cough_up_blood"
        variable.probability = 0.01
        variable = discreteDistribution.variables.add()
        variable.name = "cough_with_green_phlegm"
        variable.probability = 0.04
        variable = discreteDistribution.variables.add()
        variable.name = "cough_with_clear_spitum"
        variable.probability = 0.05
        variable = discreteDistribution.variables.add()
        variable.name = "no_cough"
        variable.probability = 0.90
```

```
cpt["testing_compliance"] = all_of(bayesianNetwork,cpt,
    {
    "tested":{"not_tested"},
    "covid_risk":{"high_covid_risk"}
    },
    ["poor_testing_compliance","testing_compliance"]
    )
```

```
cpt["self_care"] = any_of(bayesianNetwork,cpt,
    {
    "isolation_space":{"no_isolation_space"},
    "testing_compliance":{"poor_testing_compliance"},
    "quarantine_compliance":{"poor_quarantine_compliance"},
    "own_thermometer":{"dont_own_thermometer"}
    },
    ["poor_self_care","self_care"]
    )
```

```
cpt["covid_risk"] = if_then_else(bayesianNetwork,cpt,
          {
          "high_covid":{"high_covid"},
          "medium_exposure":{"medium_exposure"},
          "covid_environment":{"high_risk_covid_environment","medium_risk_covid_environment,low_risk_cov
          },
          ["high_covid_risk", "medium_covid_risk","low_covid_risk","no_covid_risk"]
          )
```

```
cpt["cardiovascular_disease"] = dependency(bayesianNetwork,cpt,
[
({"age":["elderly"]},{"relative_risk":7}),
({"diabetes":["diabetes"]},{"relative_risk":3}),
({"obesity":["obesity"]},{"relative_risk":2}),
({"heart_disorder_indicators":["heart_disorder_indicators"]},{"sensitivity":0.4, "specificity":0.3}),
({"hypertension":["hypertension"]},{"relative_risk":3.15})

],
{"cardiovascular_disease":0.09,"no_cardiovascular_disease":0.91}
)
```

```
cpt["cold_symptoms"] = avg(bayesianNetwork,cpt,
        [
        "fatigue",
        "congestion",
        "feeling_well",
        "inflammation_symptoms",
        "breathing_problems_at_night"
        ],
        [ "significant_cold_symptoms","mild_cold_symptoms","no_cold_symptoms"]
        )
```

```
cpt["disorders_of_lipid_metabolism"] = dependency(bayesianNetwork8,cpt,
[
    ({"bmi":["bmi_over_40_high_risk"]},{"relative_risk":2.428, "plus_minus":0.4, "ci": 95}),
    ({"bmi":["bmi_35_to_39_moderate_risk","bmi_30_to_34_low_risk"]},{"relative_risk":1.857, "plus_minus": 0.6, "ci": 99}),
    ({"hypertension":["hypertension"]},{"relative_risk":2.052, "plus_minus": 1.5, "ci": 90})],
{"disorders_of_lipid_metabolism":0.2175,"no_disorders_of_lipid_metabolism":0.7825}
)
```

Methodology

- Rules must be in DAG order , such that all dependencies are previously defined
- Rules are converted to a Bayesian Network in order, starting from the leaves of the DAG
- Each Rule is either a leaf or a Conditional Probability Table (CPT) in the Bayesian Network
- The generated CPT of rules is encoded as in crisp logic, but when probabilities are put in the results have the partial membership of fuzzy sets
    - all_of, the OR rule, is 100 % true iff any of the variables are true, otherwise it is 0% true
    - any_of, the AND rule is 100% true iff all of the variables are true, and otherwise it is 0% true
    - When probabilities go through the rules, the result is partial memberships

Fuzzy XOR problem (with made up relationships)

```
discreteDistribution = bayesianNetwork11.discreteDistributions.add()
discreteDistribution.name = "hypertension"
variable = discreteDistribution.variables.add()
variable.name = "hypertension"
variable.probability = 0.50
variable = discreteDistribution.variables.add()
variable.name = "no_hypertension"
variable.probability = 0.50


cpt["neither_hypertension_nor_bmi_over_40"] = all_of(bayesianNetwork11,cpt,
{
 'hypertension':{ "no_hypertension"},
 "bmi":{"bmi_35_to_39_moderate_risk","bmi_30_to_34_low_risk","bmi_25_to_29_overweight","bmi_under_25_healthy"}
 },
 ["neither_hypertension_nor_bmi_over_40","not_neither_hypertension_nor_bmi_over_40"]

 )


cpt["both_hypertension_and_bmi_over_40"] = all_of(bayesianNetwork11,cpt,
{
 'hypertension':{ "hypertension"},
 "bmi":{"bmi_over_40_high_risk"}
 },
 ["both_hypertension_and_bmi_over_40","not_both_hypertension_and_bmi_over_40"]

 )


outstr = outstr + addCpt(bayesianNetwork11,cpt)
cpt = {}

cpt["disorders_of_lipid_metabolism"] = dependency(bayesianNetwork11,cpt,
[
    ({"neither_hypertension_nor_bmi_over_40":["neither_hypertension_nor_bmi_over_40"]},{"relative_risk":2}),
    ({"both_hypertension_and_bmi_over_40":["both_hypertension_and_bmi_over_40"]},{"relative_risk":2}),
],
{"disorders_of_lipid_metabolism":0.2175,"no_disorders_of_lipid_metabolism":0.7825},
        #adjust=True
)
```

```
import sn_bayes
from sn_bayes.utils import query
evidence = {"bmi":"bmi_over_40_high_risk","hypertension":"hypertension"}
outvars = ["disorders_of_lipid_metabolism"]
results = query(longevity11,bayesianNetwork11,evidence,outvars)
results
```

```
{'disorders_of_lipid_metabolism': {'disorders_of_lipid_metabolism': 0.6094439211322045,
  'no_disorders_of_lipid_metabolism': 0.3905560788679553}}
```

```
import sn_bayes
from sn_bayes.utils import query
evidence = {"bmi":"bmi_35_to_39_moderate_risk","hypertension":"hypertension"}
outvars = ["disorders_of_lipid_metabolism"]
results = query(longevity11,bayesianNetwork11,evidence,outvars)
results
```

```
{'disorders_of_lipid_metabolism': {'disorders_of_lipid_metabolism': 0.0686735372047655,
  'no_disorders_of_lipid_metabolism': 0.9313264627952346}}
```

```
import sn_bayes
from sn_bayes.utils import query
evidence = {"bmi":"bmi_over_40_high_risk","hypertension":"no_hypertension"}
outvars = ["disorders_of_lipid_metabolism"]
results = query(longevity11,bayesianNetwork11,evidence,outvars)
results
```

```
{'disorders_of_lipid_metabolism': {'disorders_of_lipid_metabolism': 0.0686735372047655,
  'no_disorders_of_lipid_metabolism': 0.9313264627952346}}
```

```
import sn_bayes
from sn_bayes.utils import query
evidence = {"bmi":"bmi_under_25_healthy","hypertension":"no_hypertension"}
outvars = ["disorders_of_lipid_metabolism"]
results = query(longevity11,bayesianNetwork11,evidence,outvars)
results
```

```
{'disorders_of_lipid_metabolism': {'disorders_of_lipid_metabolism': 0.32777982950210577,
  'no_disorders_of_lipid_metabolism': 0.6722201704978943}}
```

Methodology for calculating Conditional Probability Tables in the Dependency Function

- Relative Risk is converted to Direct Relative Risk in a calibration function
- Relative Risk and Sensitivity/Specificity are converted to conditional probabilities on an individual level using priors from the subnet built so far
- Those individual risk relations are converted to the combined risk relations in conditional probability tables using quadratic programming on constraints from the rules of probability and preferences for smoothness and linearity in the absence of other information.
- Quadratic programming equalities include that prob a + ~prob a = 1 and that prob a given b * prob b + prob a given ~b * ~b = prob a.
- Quadratic programming uses an upper and lower bound in inequalities, below and above the linear combination
- Confidence interval derived boundaries, so that all variables bounds are raised or lowered in proportion to their confidence interval size
- Studies that are feasible within a 95% confidence interval are consonant
- Notebook shows no more than 4% error when done this way on prepared scenarios

The rest of the net is brought into the calculation of CPTs

- To ensure there is no double counting of variable's effects we calibrate the dependency rule.
  - For each variable with a RR in the dependency rule, we test with the variable and without it, to see the natural RR that occurs without including the dependency directly. This amount is subtracted off the RR before it is run, replacing the RR due to all causes with the direct RR due to the variable, not affecting through another.
  - If after calibration the direct RR is close enough to 1, the variable is eliminated as a dependency
- To ensure each part of the net fits in with the results of the previous part of the net, we calculate the priors from the net rather than from a dataset.
  - To convert the RRs and sensitivity/specificity to individual priors are needed from the previous CPTs along the DAG for every variable, and these are calculated from the subnet so far.
- These features allow the net to be modularized , and for model contributors to not have to worry about how other variables were constructed.

Convert Relative Risk to individual conditional probabilities

#Solution to these equations:
#rri=probagivenbi / probagivengood
#probagivengood= sum(probagivengoodbi*priorgoodbi)/sum(priorgoodbi)
#probagivenbi*priorbi +probagivennotbi * (1-prior bi) = priora
#sumi(probagivenbi*priorbi)=priora
#probagivenbi + probnotagivenbi = 1.0
#probagivennotbi + probnotagivennotbi = 1.0

#rr_dict: {v:rr} (if no rr it is good) prior_b_dict: {v:prior}
#returns {b:{prob_a_given_b, prob_a_given_not_b, prob_not_a_given_b, prob_not_a_given_not_b}}

Convert Relative Risk to individual conditional probabilities

```
b = {}
sum_prior_b_rr = 0
for v,prior in prior_b_dict.items():
        rr = rr_dict[v] if v in rr_dict else 1.
        sum_prior_b_rr += prior * rr
prob_a_given_good = prior_a/sum_prior_b_rr

for v,prior in prior_b_dict.items():
        b[v] = {}
        rr = rr_dict[v] if v in rr_dict else 1.
        b[v]["prob_a_given_b"] = prob_a_given_good * rr
        b[v]["prob_not_a_given_b"] = 1. - b[v]["prob_a_given_b"]
        b[v]["prob_a_given_not_b"] = -1*prior_a * (sum_prior_b_rr-prior*rr)/((prior-1.)*sum_prior_b_rr)
        b[v]["prob_not_a_given_not_b"]= 1. - b[v]["prob_a_given_not_b"]

return b
```

Convert Sensitivity/Specificity to individual conditional probabilities

```
def ss_prob_a_and_not_a_given_b_and_not_b (sensitivity,specificity,prior_a,prior_b):
    #Strategy:  break down into TP, TN, FP, FN
    # 4 equations with 4 unknowns:
    #1.  sensitivity = TP/(TP+FN)
    #2.  specificity = TN/(TN+FP)
    #3.  prior_b = TP+FP
    #4.  prior_a = FN+TP
    #
    #Therefore:
    TP = sensitivity * prior_a
    TN = specificity * (1.-prior_a)
    FN = prior_a - TP
    FP = (1.-prior_a) - TN

      #Solution:
    prob_a_given_b = TP/(TP+FP)
    prob_a_given_not_b = FN/(TN+FN)
    prob_not_a_given_b = FP/(TP+FP)
    prob_not_a_given_not_b = TN/(TN+FN)


    return prob_a_given_b, prob_a_given_not_b, prob_not_a_given_b, prob_not_a_given_not_b
```

Quadratic Programming inequalities: positive cases

```
#equation1  (doesnt have every one in it )
#non elderly hbp prevalence  =
#  (prevalence of hbp among adult obese psych) * prevalence of adult obese psych
#+ (prevalence of hbp among youngadult obese psych) * prevalence of youngadult obese psych
#+ (prevalence of hbp among teen obese psych) * prevalence of teen obese psych
#+ (prevalence of hbp among child obese psych) * prevalence of child obese psych
#+ (prevalence of hbp among adult overweight psych) * prevalence of adult overweight psych
#+ (prevalence of hbp among youngadult overweight psych) * prevalence of youngadult overweight psych
#+ (prevalence of hbp among teen overweight psych) * prevalence of teen overweight psych
#+ (prevalence of hbp among child overweight psych) * prevalence of child overweight psych
#+ (prevalence of hbp among adult healthy psych) * prevalence of adult healthy psych
#+ (prevalence of hbp among youngadult healthy psych) * prevalence of youngadult healthy psych
#+ (prevalence of hbp among teen healthy psych) * prevalence of teen healthy psych
#+ (prevalence of hbp among child healthy psych) * prevalence of child healthy psych
#+ (prevalence of hbp among adult obese nonpsych) * prevalence of adult obese nonpsych
#+ (prevalence of hbp among youngadult obese nonpsych) * prevalence of youngadult obese nonpsych
#+ (prevalence of hbp among teen obese nonpsych) * prevalence of teen obese nonpsych
#+ (prevalence of hbp among child obese nonpsych) * prevalence of child obese nonpsych
#+ (prevalence of hbp among adult overweight nonpsych) * prevalence of adult overweight nonpsych
#+ (prevalence of hbp among youngadult overweight nonpsych) * prevalence of youngadult overweight nonpsych
#+ (prevalence of hbp among teen overweight nonpsych) * prevalence of teen overweight nonpsych
#+ (prevalence of hbp among child overweight nonpsych) * prevalence of child overweight nonpsych
#+ (prevalence of hbp among adult healthy nonpsych) * prevalence of adult healthy nonpsych
#+ (prevalence of hbp among youngadult healthy nonpsych) * prevalence of youngadult healthy nonpsych
#+ (prevalence of hbp among teen healthy nonpsych) * prevalence of teen healthy nonpsych
#+ (prevalence of hbp among child healthy nonpsych) * prevalence of child healthy nonpsych
```

```
#equation2 (has the balance)
#elderly hbp prevalence  =
#  (prevalence of hbp among elderly obese psych) * prevalence of elderly obese psych
#+ (prevalence of hbp among elderly overweight psych) * prevalence of elderly overweight psych
#+ (prevalence of hbp among elderly healthy psych) * prevalence of elderly healthy psych
#+ (prevalence of hbp among elderly obese nonpsych) * prevalence of elderly obese nonpsych
#+ (prevalence of hbp among elderly overweight nonpsych) * prevalence of elderly overweight nonpsych
#+ (prevalence of hbp among elderly healthy nonpsych) * prevalence of elderly healthy nonpsych
```

Quadratic Programming inequalities: negative cases

```
#equation3  (doesnt have every one in it )
#non elderly non hbp prevalence  =
#  (prevalence of non hbp among adult obese psych) * prevalence of adult obese psych
#+ (prevalence of non hhbp among youngadult obese psych) * prevalence of youngadult obese psych
#+ (prevalence of non hhbp among teen obese psych) * prevalence of teen obese psych
#+ (prevalence of non hhbp among child obese psych) * prevalence of child obese psych
#+ (prevalence of non hbp among adult overweight psych) * prevalence of adult overweight psych
#+ (prevalence of non hbp among youngadult overweight psych) * prevalence of youngadult overweight psych
#+ (prevalence of non hbp among teen overweight psych) * prevalence of teen overweight psych
#+ (prevalence of non hbp among child overweight psych) * prevalence of child overweight psych
#+ (prevalence of non hbp among adult healthy psych) * prevalence of adult healthy psych
#+ (prevalence of non hbp among youngadult healthy psych) * prevalence of youngadult healthy psych
#+ (prevalence of non hbp among teen healthy psych) * prevalence of teen healthy psych
#+ (prevalence of non hbp among child healthy psych) * prevalence of child healthy psych
#+ (prevalence of non hbp among adult obese nonpsych) * prevalence of adult obese nonpsych
#+ (prevalence of non hbp among youngadult obese nonpsych) * prevalence of youngadult obese nonpsych
#+ (prevalence of non hbp among teen obese nonpsych) * prevalence of teen obese nonpsych
#+ (prevalence of non hbp among child obese nonpsych) * prevalence of child obese nonpsych
#+ (prevalence of non hbp among adult overweight nonpsych) * prevalence of adult overweight nonpsych
#+ (prevalence of non hbp among youngadult overweight nonpsych) * prevalence of youngadult overweight nonpsych
#+ (prevalence of non hbp among teen overweight nonpsych) * prevalence of teen overweight nonpsych
#+ (prevalence of non hbp among child overweight nonpsych) * prevalence of child overweight nonpsych
#+ (prevalence of non hbp among adult healthy nonpsych) * prevalence of adult healthy nonpsych
#+ (prevalence of non hbp among youngadult healthy nonpsych) * prevalence of youngadult healthy nonpsych
#+ (prevalence of non hbp among teen healthy nonpsych) * prevalence of teen healthy nonpsych
#+ (prevalence of non hbp among child healthy nonpsych) * prevalence of child healthy nonpsych
```

```
#equation4 (has the balance)
#elderly with non hbp prevalence =
#   (prevalence of non hbp among elderly obese psych) * prevalence of elderly obese psych
#+ (prevalence of non hbp among elderly overweight psych) * prevalence of elderly overweight psych
#+ (prevalence of non hbp among elderly healthy psych) * prevalence of elderly healthy psych
#+ (prevalence of non hbp among elderly obese nonpsych) * prevalence of elderly obese nonpsych
#+ (prevalence of non hbp among elderly overweight nonpsych) * prevalence of elderly overweight nonpsych
#+ (prevalence of non hbp among elderly healthy nonpsych) * prevalence of elderly healthy nonpsych
```

Objective Function encodes Soft Preferences

- The Objective Function is designed to prefer, but not require, function properties in the absence of other information. These properties are smoothness and linearity.
- In Quadratic Programming, the objective function is defined by P,q,lower bound and upperbound.
  - P tells what variables are multiplied by what, q tells the constants to add, lower bound tells the lowest value of a variable, and upper bound tells the highest.
- Because the variables are probabilities we set the lower bound to zero and the upper bound to one for each variable
- We encoded a preference for smoothness and linearity by using an identity matrix multiplied by the variables position for P, with no constants for q
  - Squaring a variable makes it smooth across equations, however if the hard constraints of the rules of probability require it to not be smooth it will not.
  - Increasing the size according to position uses the knowledge that variables to the left, the negative variables, should have a negative effect on the condition in combination and those on the right should have a positive effect. However, if nonlinearities are encoded in the rules then this preference is overridden

Quadratic Programming Objective Function

```
P = np.identity(size,np.double)
 for i in range(len(cartesian)):
                P[i][i] = np.double(i+1)
 for i in range(len(cartesian),size):
                P[i][i] = np.double(size-i)


 q = np.zeros (size,np.double)
 lb = np.zeros(size,np.double)
 ub = np.ones(size,np.double)
```

Validation Score

- We define consonance as how close the Relative Risks (RRs) of our model are to those of the studies, relative to the Confidence Intervals (CIs) of those studies, in order to satisfy the hard constraints of Bayes Rule in the Quadratic Programming equations.
- We derive a Validation score that tells how consonant information sets are.
- The validation score is defined as how far RR must be adjusted in order for the set to be feasible in the Quadratic Programming, and can be used to determine what CIs RR is still within after adjustment
- Usually a CI is a measure if you believe someone else, and if you should support it. Instead of summing up your belief in other perceivers we use the CI to determine the extent that one study supports the other, and which therefore make the most sense together.

Validation Score Algorithm

- Confidence Intervals are on either Relative Risks or Sensitivity/Specificity stats, but we use RR for explanatory purposes
- First normalize the confidence intervals of the RRs of variables that affect a condition (within one Conditional Probability Table (CPT)) , by converting them all to a 99.9% confidence interval, and then finding the relative sizes with the largest 99.9% confidence interval being the largest .
- Multiply the window by these relative sizes to make sure each variable is adjusted by the same proportion to its CI bounds. (The window is the lower and upper bound of the conditional probabilities in the quadratic programming inequalities).
- Use a binary search to make the window as small as possible (making it as close to the RR of the studies as possible) and still feasible according to the quadratic programming algorithm
- Find consonance by converting smallest feasible conditional probability window back to RR confidence intervals , to find what confidence interval the conditional probabilities fall within. This is your validation score.
    - (RR + CI)/RR = (prob_a_given_b + smallest_feasible_window)/prob_a_given_b

Confidence Interval Based Lower and Upper Bounds

```
def align_ci(ci,amt):
    ci_z= {80:1.282,85:1.440,90:1.645,95:1.960,99:2.576,99.5:2.807,99.9:3.291}
    new_amt = amt if ci == 99.9 else ci_z[99.9]*amt/ci_z[ci]
    return new_amt

def normalize_ci(maxi,window_dict):
    factor=1.0/maxi
    new_dict = {k: v*factor for k, v in window_dict.items() }
    return new_dict

def get_window(bayesianNetwork,invars):
        var_val_positions = get_var_val_positions(bayesianNetwork)
        window = {}
        for vardict,numval_dict in invars:
            for var, vallist in vardict.items():
                if "ci" in numval_dict:
                    if var not in window:
                        window[var] = {}
                    for val in vallist:
                        window[var][val]= align_ci(numval_dict["ci"],numval_dict["plus_minus"])
                    #print("window")
                    #print(window)
                    maxi= max(window[var].values())
                    #print("maxi")
                    #print(maxi)
                    for val,dummy in var_val_positions[var].items():
                        if val not in window[var]:
                            window[var][val] = maxi
                    window[var] = normalize_ci(maxi,window[var])
        return(window)
```

Validation score is converted back from a window for the conditional probabilities to the Relative Risk or Sensitivity/Specificity confidence interval:

row["score"] = ((probagivenb + (final_window*window_factor[k][v])) * stat/probagivenb)-stat