Predictive model using Bayesian networks in R

Madhav Thakker — 2016159

1 Libraries used

```
library(bnlearn)
library(gplots)
library(GGally)
library(polycor)
```

2 Data overview

2.1 Random Attributes

The columns that I selected randomly are -

```
my_col_names = "age,cp,trestbps,chol,thalach,exang,oldpeak,ca,thal,num"
continous_col_names = "age,trestbps,chol,thalach,oldpeak"
discrete_col_names = "cp,exang,ca,thal,num"

data[,"num"][data[,"num"] > 0]  = 1 # binarize heart disease (presence or absence)
```

2.2 Missing Values

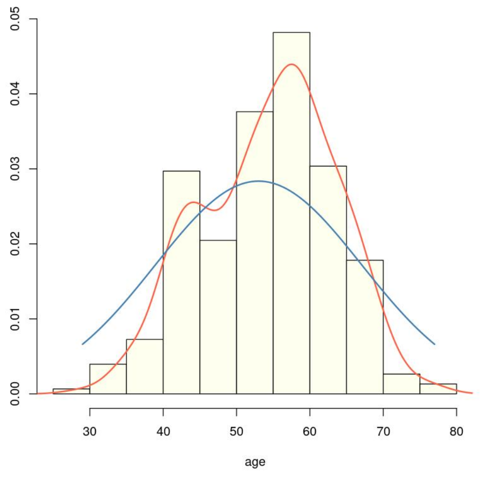
There are some feature values which are missing. Those missing values are represented by '?' in the data. To handle those values, I fit a Bayesian network by removing the rows with at-least 1 missing value, and then impute the missing values using *impute* function of bnlearn.

2.3 Continous Values

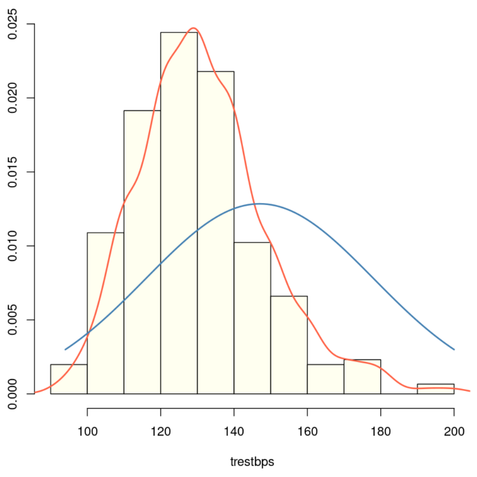
```
data[,discrete_col_names] <- lapply(data[,discrete_col_names], as.factor)
data[,continous_col_names] <- lapply(data[,continous_col_names], as.numeric)
```

I converted all the discrete columns to *factor* type and all continous columns to *numeric* type.

I try to visualize the continous features for checking if they can be approximated as Gaussian, and as visible most of them follow the Gaussian distributions. So, for handling the continuous features, I approximate them as Gaussian and discrete features as Multinomial distribution.



(a) Age



(b) Trestbps

Figure 1: Continuous attributes

3 Structure Learning

Using some information from dataset, I could tell that these features should not change when one is changed, so I added a black-list in the network corresponding to these attributes.

```
bl = tiers2blacklist(list("exang",c("chol", "trestbps")))
bl = rbind(bl, c("exang", "chol"), c("chol", "exang"))
```

Similarly, I also added white-list connections in the network.

```
wl = matrix(c("thalach", "trestbps"),ncol = 2, byrow = TRUE, dimnames = list(NULL, c("fr
```

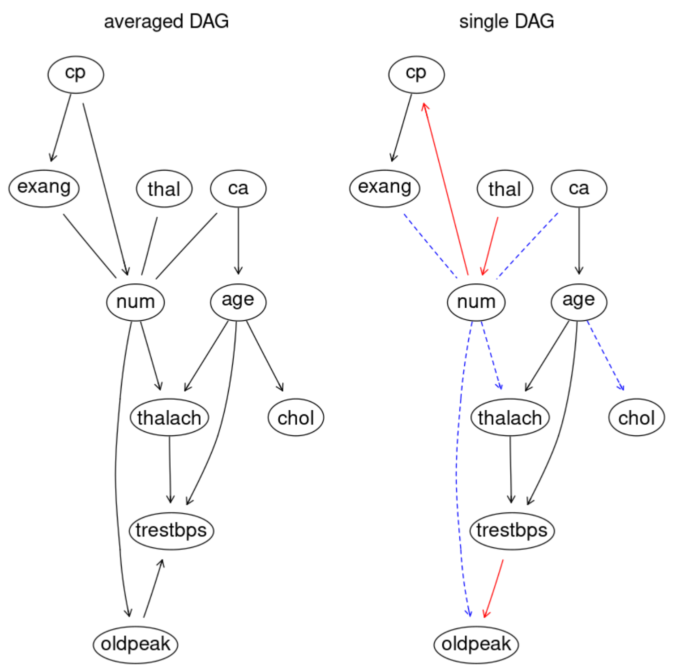
Learning structure using data; finding the network structure with the best goodness-of-fit on the whole data.

```
dag = mmhc(data_no_na, whitelist = wl, blacklist = bl)
```

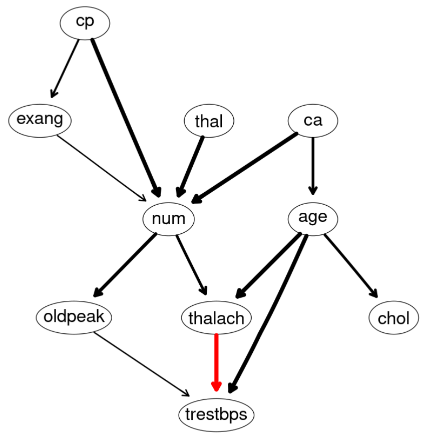
The quality of the above learn structure depends on whether features are normally distributed, since some of the continuous features do not strictly follow normal-distribution, I used *boot.strength()* to learn a separate network from each bootstrap sample.

```
strength = boot.strength(data_no_na, R = 200, algorithm = "hc",
                                algorithm.args = list(whitelist = wl, blacklist = bl))
```

I use *averaged.network()* to take arcs from bootstrapped network with strength of atleast 0.6 and take the averaged consensus network which is the best network structure for our data.



(a) Comparing



(b) Final Structure

Figure 2: Strucure Learning

4 Parameter Learning

Now, I try to learn the parameters of the network.

```
fitted = bn.fit(simpler, data_no_na) #data_no_na has NA rows removed
imputed_data = impute(fitted, data) # getting values for '?', using impute function
fitted_all = bn.fit(simpler, imputed_data)
```

Predicting the values for node "num" and checking accuracy.

```
predicted_test = predict(fitted_all, node = "num", data = test_data)
accuracy_test = sum(test_data[,"num"] == predicted_test) / length(predicted_test)

predicted_train = predict(fitted_all, node = "num", data = train_data)
accuracy_train = sum(train_data[,"num"] == predicted_train) / length(predicted_train)
```

I got train-accuracy of **87%** and test-accuracy of **84%**.

5 Bnlearn conditional probability

Bayesian networks in Bnlearn simplify the learning process by local independence. It means that any variable in the network is independent of its non-descendents given its parents.

From chain rule - $P(A, B) = P(A|B) * P(B)$

Joint Distribution over all the variables can be written as the product of the probabilities of variable given the parents. Hence encoding the independencies in the Joint Distribution in a graph structure helped us in reducing the number of parameters that we need to store This is how bnlearn does parameter learning.

During inference, we try to answer probability queries over the network given some other variables. There can a lot of variables which can take a lot of computation time. But, in graphical models we exploit the independencies to break these operations in smaller parts making it much faster by using variable elimination.

5.1 Loop-holes

The algorithm bnlearn uses for paramter learning is Maximum Likelihood Estimation. Some of the loopholes of this methods are -

- Overfitting - MLEs are known to overfit when sample size is small. This is also true for bnlearn. It can be biased when there are not enough samples in training set.
- For continous variable we assume a Gaussian distribution which may not be true in all cases. If a continous variable does not follow Gaussian distribution, the structure of the network may not be correct.
- If our features are such that all the features have the same parent, the memory consumption can be overwhelming even for latest computers. The reason is because we can no longer break these operations by variable elimination.

6 References

- https://www.bnlearn.com/
- https://www.bnlearn.com/examples/useR19-tutorial/