Term Deposit Subscription Prediction

Anmisha Reddy
Department of Computer Science
University at Buffalo
anmishar@buffalo.edu

Sindhura Uppu
Department of Computer Science
University at Buffalo
suppu@buffalo.edu

Abstract

This project aims to predict if a client will subscribe to a term deposit in a bank. The data is related with direct marketing campaigns, phone calls, of a Portugese banking institution. The goal is to achieve this prediction by generating a probabilistic graphical model or Markov Network to model the data and answer queries. The dataset is obtained from here. Exact and Approximate inference algorithms have been applied on this network using which various queries have been answered that result in classifying which set of variables give accurate prediction.

1 Problem Domain

The dataset that is being modeled in this project is obtained from UCI Machine Learning Repository. The data is related with direct marketing campaigns of a Portugese banking institution. A variety of variables are taken into consideration and their values play a role in the client's decision to either subscribe for a term deposit or not. This marketing campaign is carried out through phone calls. The major goal of this project is to see which set of variables give accurate prediction by observing the inference answers of the training data.

1.1 Data Set

The data set is the collection of the details of the clients of a Portugese banking institution. The dataset contains around 4522 instances and 21 attributes. These attributes help in analyzing the state of a client by identifying whether he/she has taken a personal or home loan, their educational status and whether anyone from the bakn has previously called them for asking about subscription and their decision whether to subscribe or not. The dataset contained many 'string' type variables, which are converted to categories, if found to be discrete. The following are the descriptions of the variables and what they each represent

Age: age of the client.

Job: type of job – admin, blue-collar, entrepreneur, housemaid, management, retired, self-employed, services, student, technician, unemployed, unknown.

Marital: marital status of the client – divorced, married, single, unknown.

Education: how much education does the client have – basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university.degree, unknown.

Default: Does the client have credit in default? – yes or no.

Housing: Did the client take any housing loan in the past? – yes or no.

Loan: Did the client take any personal loan in the past? – yes or no.

Contact: Contact communication type – cellular or telephone.

Month: last contact month of the year – jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec.

Day_of_week: last contact day of the week – mon, tue, wed, thu, fri, sat.

Duration: last contact duration, in seconds.

Campaign: number of contacts performed during this campaign and for this client.

Pdays: number of days that passed by after the client was last contacted from a previous campaign.

Previous: number of contacts performed before this campaign and for this client.

Poutcome: outcome of the previous marketing campaign – failure, nonexistent, success.

Emp.var.rate: employment variation rate.

Cons.price.idx: consumer price index.

Cons.conf.idx: consumer confidence index.

Euribor3m: euribor 3 month rate.

Nr.employed: number of employees.

Y: Has the client subscribed a term deposit? (Yes or No).

Y is the output variable in this dataset that classifies whether the client has subscribed for a term deposit or not.

1.2 Markov Network Model

Markov models are probabilistic graphical models that are undirected. They represent the joint probability distribution over events which are represented by variables. The nodes in the graph represent variables and the edges correspond to a notion of direct probabilistic interaction between the neighboring variables – an interaction that is not mediated by any other variable in the network[1]. Factors are associated with each complete subgraph in the network. They are usually non negative and do not necessarily represent probabilities. The joint distribution is the product of distribution of factors together. The factors can be normalized by taking the entire sum and dividing each factor with this total.

The variables of the data set were assessed and relations were drawn. The Markov Network consists of 9 variables with 9 edges between them. All these variables are considered as categories and the values are changed to numeric codes that are given to categories. These values are given in ascending order to the distinct values in each variable, based on their first letter. After performing this conversion, some of the values of the variables are as follows:

Default: 0 – no, 1 – yes.

Marital: 0 – divorced, 1 – married, 2 – single.

Contact: 0 – cellular, 1 – telephone, 3 – unknown.

Housing: 0 – no, 1 – yes.

Education: 0 – primary, 1 – secondary, 2 – tertiary, 3 – unknown.

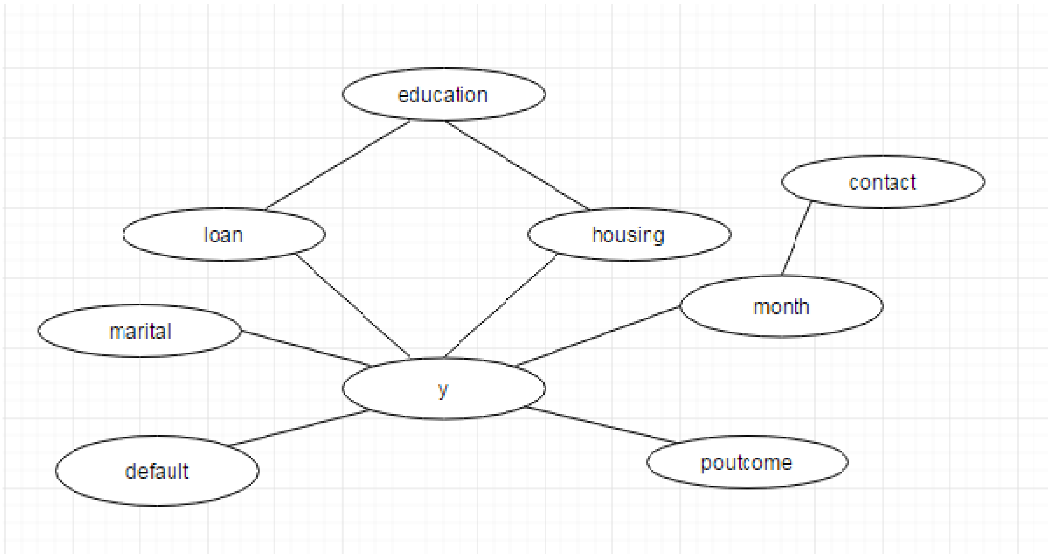
Loan: 0 – no, 1 – yes.

Poutcome: 0 – failure, 1 – other, 2 – success, 3 – unknown.

Month: 0 – apr, 1 – aug, 2 – dec, 3 – feb, 4 – jan, 5 - jul, 6 – jun, 7 – mar, 8 – may, 9 – nov, 10 – oct, 11 – sep.

Y: 0 – no, 1 – yes.

The Markov network that we represented below was constructed manually based on our intuition.



The Markov network can be generated in Python as follows[2]:

```
mark=
MarkovModel([“education”,”loan”),(‘education’,’housing’),(‘loan’,’y’),(‘housing’,’y’),(
‘marital’,’y’),(‘default’,’y’),(‘contact’,’month’),(‘month’,’y’),(‘poutcome’,’y’)])
```

Pgmpy library has inbuilt models including Bayesian Model and Markov Model. The nodes that have edges are sent as parameters using the node names as attributes in the model.

1.3 Factors

It is not possible to fit a Markov model with our existing data set. Hence, we can use factors to represent the joint probability distribution of the Markov model. Factors help parameterize the model. The factors are not probabilities or conditional probabilities between two variables, rather they seem to have values given by a user's intuition[1]. This means that they are significantly harder to estimate from the data.

We aim to initialize the factors in such a way that the accuracy of inference algorithms is high. To this extent, we took the co-occurrence of each subset of variables in the Markov network and gave these values as discrete factors to the model. Thus, by doing this we obtained around 9 factors that are added to the Markov model declared as above. Our representation is complete as the graph structure is associated with a set of parameters.

Code snippet that shows how we obtained the factor values and declared them using DiscreteFactors in pgmpy library[2].

```
data.groupby([“education”,”loan”]).size()
```

Results:

```
education  loan
0          0        584
           1         94
```

```
1          0       1890
           1        416
2          0       1176
           1        174
3          0        180
           1          7
dtype: int64
```

f1= DiscreteFactor([“education”,”loan”],cardinality=[4,2],values=(584,94,1898,416,1176,174,180,7))

Groupby() function gives the count of co-occurrences of each subset of variables given, which are education and loan. DiscreteFactor is a factor type present in pgmpy.factors.discrete and it is used to declare discrete factors for each set of variables. Cardinality is the size of unique elements in each variable and the values must be a list of each co-occurrence in sorted order. In a similar fashion, the 9 discrete factors are declared and these factors are added to the Markov network as shown below:

mark.add_factors(f1)

Local independencies are obtained from the model using get_local_independencies() function.

mark.get_local_independencies()

Results:

```
(poutcome _|_ default, loan, month, contact, education, mar
ital, housing | y)
(default _|_ poutcome, month, loan, marital, contact, educa
tion, housing | y)
(loan _|_ poutcome, default, housing, month, contact, marit
al | y, education)
(marital _|_ poutcome, default, loan, month, contact, educa
tion, housing | y)
(contact _|_ poutcome, default, loan, marital, y, education
, housing | month)
(month _|_ poutcome, default, loan, marital, education, hou
sing | y, contact)
(y _|_ contact, education | poutcome, default, loan, month,
marital, housing)
(education _|_ poutcome, default, month, contact, y, marita
l | loan, housing)
(housing _|_ poutcome, default, loan, month, contact, marit
al | y, education)
```

The factors in a Markov model can be retrieved using the function get_factors() as below:

```
for fact in mark.get_factors():
        print(fact)
```

Results:

```
+-------------+-----------+--------------------------+
| education   | housing   |   phi(education,housing) |
|-------------+-----------+--------------------------|
| education_0 | housing_0 |                 295.0000 |
| education_0 | housing_1 |                 383.0000 |
| education_1 | housing_0 |                 876.0000 |
| education_1 | housing_1 |                1430.0000 |
| education_2 | housing_0 |                 687.0000 |
| education_2 | housing_1 |                 663.0000 |
| education_3 | housing_0 |                 104.0000 |
| education_3 | housing_1 |                  83.0000 |
+-------------+-----------+--------------------------+
+--------+-----+---------------+
| loan   | y   |   phi(loan,y) |
|--------+-----+---------------|
| loan_0 | y_0 |     3352.0000 |
| loan_0 | y_1 |      478.0000 |
| loan_1 | y_0 |      648.0000 |
| loan_1 | y_1 |       43.0000 |
+--------+-----+---------------+
```

2 Inference Models

2.1 Belief Propagation

Computing the a posteriori belief of a variable in a general Markov Network is NP-hard. Belief Propagation is an Approximate Inference algorithm. It is an efficient way to solve inference problems by passing local messages. It is available in the pgmpy.inference library as a class for performing inference using BeliefPropagation model. It creates a junction tree or Clique tree for the input probabilistic graphical model and performs calibration of the junction tree so formed using belief propagation. Thus, we trained our model using belief propagation as follows[3]:

```
belief_prop = BeliefPropagation(mark)
```

Now that the inference is drawn from this model, queries can be run on the model to analyze the variation and the independence of one or more variables over other such variables in the model. A few of the queries implemented are as follows:

Query1:

```
bp1 = belief_prop.query(variables=['y'],evidence={'marital' : 0,'default' : 0})
print(bp1['y'])
```

We aim to find the variation of y, whether the client opts to subsribe or not, based on his/her marital status (divorced) and default value (no).

Results:

```
+-----+----------+
```

```
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |
+-----+----------+
```

Query2:

```
bp2 = belief_prop.query(variables=['y'],evidence={'education' : 2,'housing' : 1})
print(bp2['y'])
```

We aim to find the variation of y, whether the client opts to subscribe or not, based on his/her education (tertiary) and when the client took housing loan.

Results:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |
+-----+----------+
```

Query3:

```
bp3 = belief_prop.query(variables=['y'],evidence={'month' : 11,'poutcome' : 2})
print(bp3['y'])
```

We aim to find the variation of y, whether the client opts to subscribe or not, based on the month they were last contacted (oct) and when the previous outcome is success.

Results:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   0.9997 |
| y_1 |   0.0003 |
+-----+----------+
```

Query4:

```
bp4 = belief_prop.query(variables=['y'],evidence={'poutcome' : 2, 'loan' : 1})
print(bp4['y'])
```

We aim to find the variation of y, whether the client opts to subscribe or not, based on the case where the client has taken a personal loan and previous outcome for term deposit subscription is a success.

Results:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |
+-----+----------+
```

Query5:

```
Bp5 = belief_prop.query(variables=['y'],evidence={'marital' : 1, 'loan' : 1, 'contact' : 1, 'month' : 5})

print(bp5['y'])
```

We aim to find the variation of y, whether the client opts to subscribe or not, based on several factors such as the client is married, has taken loan, contact is through telephone and last contacted month is July.

Results:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |
+-----+----------+
```

2.2 Variable Elimination

Variable Elimination is an exact inference algorithm. It is available in the pgmpy.inference library as a class for performing inference[3].

```
from pgmpy.inference import VariableElimination
infer = VariableElimination(mark)
```

Now that the inference is drawn from this model, queries can be run on the model to analyze the variation and the independence of one or more variables over other such variables in the model. A few of the queries implemented are as follows:

Query1:

```
infer = VariableElimination(mark)

phi_query = infer.query(variables=['y'],evidence={'marital':0,'default':0})

print(bp1['y'])
```

Result:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |

+-----+----------+
```

Query2:

```
infer = VariableElimination(mark)

phi_query = infer.query(variables=['y'],evidence={'education':2,'housing':1})

print(bp1['y'])
```

Result:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |

+-----+----------+
```

Query3:

```
infer = VariableElimination(mark)
phi_query = infer.query(variables=['y'],evidence={'month':10,'poutcome':2})
print(bp1['y'])
```

Result:

```
+-----+----------+
| y   |   phi(y) |
|-----+----------|
| y_0 |   1.0000 |
| y_1 |   0.0000 |
+-----+----------+
```

2.3 MPLP

MPLP is a class for performing approximate inference using Max-Product Linear Programming method[3]. Markov model can be trained using this inference algorithm. The following code snippet shows how we have trained our model using MPLP inference algorithm.

```
from pgmpy.inference import Mplp
mplp = Mplp(mark)
```

3 Sampling

We draw samples from the Markov network so that we can better understand the data and make statistical inferences on them.

3.1 Gibbs Sampling

A distribution $P_{ǿ}$ is a Gibbs distribution parameterized by a set of factors Φ = {Ǿ1(D1),…Φk(Dk)} if it is defined as follows:

$$P_{\Phi}(X_1,\ldots X_n) = 1 / Z\ (\acute{P}_{\Phi}(X_1,\ldots,X_n)$$

GibbsSampling is a class present in pgmpy.sampling. It generates samples from models from which variables are inherited and transition probabilities computed[4]. The following code snippet shows how we obtained samples from Markov network using GibbsSampling.

```
from pgmpy.sampling import GibbsSampling
gibbs_chain = GibbsSampling(mark)
gibbs_chain.sample(size=5)
```

This code generates samples of size 5 from the model.

4 Evaluation Metrics

So far, we developed inference algorithms for determining the mean and entropy of each distribution.

4.1 Mean of a distribution

The mean for a distribution P(x) is:

$$E[p(\mathbf{x})] = \sum_{\mathbf{x}} \mathbf{x} p(\mathbf{x})$$

Using N samples, the mean can be computed as

$$\hat{E}[p(\mathbf{x})] = \frac{1}{N} \sum_{k=1}^{N} \mathbf{x}_k$$

Numpy package in python has predefined functions for computing mean.

Calculation of mean from the sample obtained:

np.mean(df)

Result:

```
marital      1.2
education    1.2
default      0.0
housing      0.8
loan         0.4
contact      0.8
month        6.4
poutcome     1.8
y            0.0
dtype: float64
```

4.2 Entropy

The entropy of a distribution p(x) is

$$H[p(\mathbf{x})] = -\sum_{\mathbf{x}} p(\mathbf{x}) \ln p(\mathbf{x})$$

When using N samples, the entropy can be calculated as:

$$\hat{H}[p(\mathbf{x})] = -\frac{1}{N} \sum_{k=1}^{N} \ln p(\mathbf{x}_k)$$

In order to calculate the entropy, the data frame containing the sample is converted into an array in Python and the probabilities are computed for each cell column-wise. The entropy is calculated using the entropy function in 'Scipy' package as follows:

scipy.stats.entropy(s1)

Results:

```
array([ 1.56071041,  1.32966135,        -inf,  1.38629436,
0.69314718,
        0.69314718,  1.37050239,  1.09861229,        -inf])
```

References

[1] Probabilistic Graphical Models Principles and Techniques, Daphne Koller and Nir Friedman. The MIT Press, Cambridge, Massachusetts.

[2] http://pgmpy.org/models.html

[3] http://pgmpy.org/inference.html

[4] http://pgmpy.org/sampling.html