Determining Probabilities of Handwriting Formations using PGMs

Yashankit Shikhar
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY
yshikhar@buffalo.edu

Abstract

This aim of this project is to develop probabilistic graphical models (PGMs) to determine probabilities of observations which are described by several variables using PGMPY libraries. Handwriting patterns which are described by document examiners have been analyzed. The probabilities obtained using Bayesian Networks were used to determine whether a particular handwriting sample is common or rare which in turn can be useful to determine whether a sample was written by a certain individual or different individuals.

1 Introduction

In this study, Handwriting samples were analyzed using Bayesian networks and the observations were applied to find the most common and rare handwriting sample. Two data-sets have been considered for this study, 'th' conditional probability data-set and 'and' data-set. The 'th' data-set contains six feature vectors and the 'and' data-set contains nine feature vectors. Following approaches were taken to reach the conclusion:

1. Pairwise correlations and dependencies were evaluated that exist in the data between feature variables using the conditional probability provided for 'th' data-set. All dependencies and magnitude of dependence were calculated.
2. Several Bayesian Networks were created using the results obtained from previous steps and conditional probability provided. A data-set is generated using Bayesian sampling and score for the Bayesian network is calculated. Based on the best model, the most common and most rare 'th' sample is calculated.
3. The Bayesian Network is converted to a Markov Network and the factors are observed.
4. The same process is repeated on the 'and' data-set, the only difference being that the whole data-set is available instead of conditional probabilities and hence, the Conditional Probability is generated by the PGMPY library.

2 Task 1: Pairwise correlations and dependencies

In this task, pairwise dependencies have been calculated between two features of the data-set. We know that two variables are independent if $p(xi, xj) = p(xi)p(xj)$, where the joint probability between a pair of variables can be determined from the data-set provided as $p(xi, xj) = p(xi|xj)p(xj)$. To calculate dependencies, the following formula was used

$$\sum_i \sum_j \mid p(xi|xj)p(xj) - p(xi)p(xj) \mid \tag{1}$$

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Table 1: Conditional Dependencies

Features	X1	X2	X3	X4	X5	X6
X1	-	0	0	0.1196	0	0.1606
X2	0.16	-	0.2182	0.1157	0.1293	0.1139
X3	0	0.2182	-	0	0.1156	0.0778
X4	0.1196	0	0	-	0	0.117
X5	0	0.1293	0.1156	0	-	0
X6	0.1606	0	0.0948	0.1431	0	-

The calculations obtained were used to find out the pairwise relations as shown in Table-1. The rows represent dependency of a variable on different other variable given by columns of the table. The entry of 0.16 with the row as X2 and column as X1 represent the magnitude of how strong X2 is dependent on X1. The closer the value is to zero the more independent the two variables are respect to each other. If conditional probabilities are not provided, it has been assumed that the two variables are independent. From Table-1 it can be seen that there are strong dependencies between X2 -> X3, X3 -> X2 and, X5 -> X3. Similarly, there are very weak dependencies between X3 -> X6, X6 -> X2 and, X6 -> X3. For the next step, we set up a threshold and based on it construct Bayesian Networks by varying the threshold.

3 Task 2: Bayesian Network Creation and Inference

Once the dependencies were calculated using the results from the Task 1, the Bayesian network can be created. The Bayesian Networks were created for this project using the following code and PGMPY Library. A threshold is set on the previous calculated result to determine if two variables are independent or not. We assume independence for pairs of variables not appearing in the CPD table.

```
from pgmpy.models import BayesianModel
model1 = BayesianModel([('x1', 'x6'), ('x6', 'x4'), ('x6', 'x2'),
                        ('x2', 'x3'), ('x2', 'x5')])
model2 = BayesianModel([('x3', 'x6'), ('x3', 'x5'), ('x6', 'x2'),
                        ('x6', 'x1'), ('x6', 'x4')])
model3 = BayesianModel([('x5', 'x3'), ('x5', 'x2'), ('x3', 'x6'),
                        ('x6', 'x4'), ('x4', 'x1')])
model4 = BayesianModel([('x4', 'x1'), ('x1', 'x6'), ('x1', 'x2'),
                        ('x2', 'x3'), ('x3', 'x5')])
model5 = BayesianModel([('x5', 'x3'), ('x5', 'x2'), ('x3', 'x2'),
                        ('x3', 'x6'), ('x6','x1'),('x1','x4')])
```

In the above code, ('x1','x6') represents x1 -> x6. The Conditional Probability from the data-set provided were given as a parameter to each model and the network is checked for structure and CPDs. This is done by following sample code.

```
from pgmpy.factors.discrete import TabularCPD
cpd_x1 =TabularCPD(variable='x1', variable_card=4, values=[px1])
...
cpd_x6 =TabularCPD(variable='x6', variable_card=5, values=px61t,
evidence=['x1'], evidence_card=[4])
model1.add_cpds(cpd_x1, ... ,cpd_x6)
model1.check_model()
```

Here, 'variable_card' tells how many parameters does that variable take for example, x1 takes 4 parameters, 'values' have the CPD table for the given dependencies, 'evidence' is the parent of the given node and, 'evidence_card' is how many parameter can the parent variable take.

We use likelihood to compare these different models. Calculating likelihood needs data. Since we don't have the concrete data-set for this project, we generate the data ourselves. As seen from the code above, five Bayesian networks were generated from the results of Task 1 and by the use of Bayesian sampling, data-set were generated for each Bayesian Network. The sampling code is given

Table 2: K2 Scores

Datasets	Model1	Model2	Model3	Model4	Model5
Dataset 1	-6417	-6504	6532	-6497	-6511
Dataset 2	-6526	-6492	-6563	-6549	-6577
Dataset 3	-6528	-6542	-6470	-6523	-6526
Dataset 4	-6625	-6648	-6632	-6511	-6577
Dataset 5	-6545	-6563	-6518	-6502	-6489

below to generate data-set of size 1000 from each of the models. This helps us in calculating the K2 score for each model.

```
from pgmpy.sampling import BayesianModelSampling
inference = BayesianModelSampling(model)
data=inference.forward_sample(size=1000)
```

Once, the data-set was generated, K2 Score for each model was calculated for each data-set to figure out the best model. The K2 score for each model on the five generated data-set is given in Table-2. The code to generate the K2 score is given below.

```
from pgmpy.estimators import K2Score
k1 = K2Score(generated_dataset)
```

When the following order elimination code was executed for our best model, the following order was obtained which was expected.

```
from pgmpy.inference.EliminationOrder import WeightedMinFill
WeightedMinFill(model).get_elimination_order()
```

»['x1', 'x4', 'x3', 'x5', 'x2', 'x6']

From the observations in Table-2, We conclude that the Model 1 generated is the best Bayesian network. Model 1 is shown in Fig-1. Using this Bayesian Network, we calculate it's joint distribution from the formula

$$p(x1, x2, x3, x4, x5, x6) = p(x1)p(x6|x1)p(x4|x6)p(x2|x6)p(x3|x2)p(x5|x2) \tag{2}$$

for each combination of parameters. Using the joint probability distribution, we calculate the highest and lowest joint probability combination and report the most probable and most rare 'th' pattern. According to the results obtained, the most common and most rare 'th' pattern is shown in Table-3 and Table-4 respectively.

Figure 1: Best Model obtained for 'th' data-set

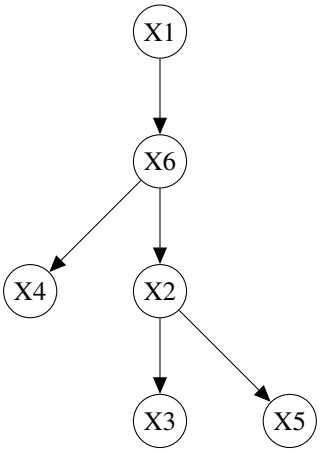


Table 3: Most Common 'th' pattern

Features	Parameters
X1: Height Relationship of t to h	1: t shorter than h
X2: Shape of Loop of h	1: retraced
X3: Shape of Arch of h	2: pointed
X4: Height of Cross on t staff	1: upper half of staff
X5: Baseline of h	1: slanting upward
X6: Shape of t	4: closed

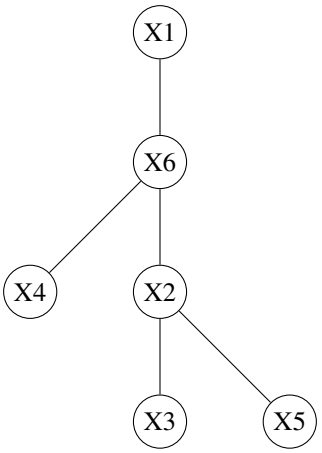
Table 4: Most Rare 'th' pattern

Features	Parameters
X1: Height Relationship of t to h	1: retraced
X2: Shape of Loop of h	1: pointed
X3: Shape of Arch of	1: rounded arch
X4: Height of Cross on t staff	1: upper half of staff
X5: Baseline of h	1: slanting upward
X6: Shape of t	1: tented

4 Task 3: Conversion of Bayesian Network to Markov Network

In Task 3, Bayesian Network has been asked to be converted to Markov Network using Moralization. In Moralization, A Markov Network is created from a Bayesian Network by following two rules. First, Markov Network will contain an un-directed edge between X and Y if there is a directed edge between them in the Bayesian Network or X and Y are both parents of the same node. Model 1, our best model is converted to Markov Network and observed. Since each node in our Bayesian Network has just one parent, our Markov Network will be same as Bayesian Network. The Bayesian

Figure 2: Markov Model obtained for 'th' data-set



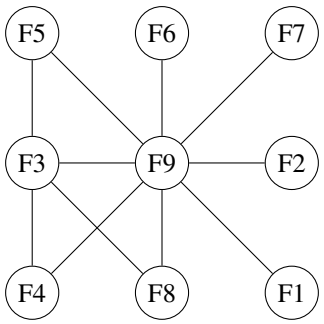
Network constructed during the task 2 is converted to Markov Network using the following code and the factors are obtained.

```
markov_model = model.to_markov_model()
markov_model.get_factors()
```

»[<DiscreteFactor representing phi(x1:4) at 0x7fc37ef78b00>,
<DiscreteFactor representing phi(x6:5, x1:4) at 0x7fc37ef78048>,
<DiscreteFactor representing phi(x4:4, x6:5) at 0x7fc37f315160>,
<DiscreteFactor representing phi(x2:5, x6:5) at 0x7fc37f315e10>,
<DiscreteFactor representing phi(x3:3, x2:5) at 0x7fc37f315470>,

<DiscreteFactor representing phi(x5:4, x2:5) at 0x7fc37f315fd0>]

Figure 3: Markov Model obtained for 'and' data-set



As most of our models have only one parent nodes therefore, the Markov Model is of same structure as the Bayesian Model and thus, it returns similar results from inference when compared to Bayesian Network. However, when a Markov model is created from the Bayesian Network generated from Task 4, we see a difference in structure. There is an extra-edge in Fig-3 compared to Fig-4. This is moralization, as node F8 has both F3 and F9 as parent so there will be an un-directed edge between them.

5 Task 4: Bayesian Network Construction and Inference for 'and' data-set

We use the "and" image data-set to construct a Bayesian network and evaluate the goodness score (likelihood of a data-set) of several Bayesian networks. We repeat the process given in Task 2 on the

Table 5: Most Common 'and' pattern

Features	Parameters
F1: Initial stroke of formation of a	3: Center of Staff
F2: Formation of staff of a	2: Retraced
F3: Number of arches of n	2: Two
F4: Shape of arches of n	1: pointed
F5: Location of mid-point of n	3: At baseline
F6: Formation of staff of d	3: Looped
F7: Formation of initial stroke of d	1: Overhand
F8: Formation of terminal stroke of d	4: No obvious ending stroke
F9: Symbol in place of the word and	2: Symbol

'and' data-set however, we have been provided with the entire data-set this time instead of conditional probabilities. Our first task is to perform a greedy hill climb search to get the best Bayesian Network possible for the data-set. The Hill Climb search is performed using the following code

```
from pgmpy.estimators import HillClimbSearch
from pgmpy.estimators import K2Score
hc = HillClimbSearch(data, scoring_method=K2Score(data))
best_model = hc.estimate()
one_parent=hc.estimate(max_indegree=1)
```

best model: »[('f3', 'f4'), ('f3', 'f9'), ('f3', 'f8'), ('f5', 'f9'), ('f5', 'f3'), ('f9', 'f8'), ('f9', 'f7'), ('f9', 'f1'), ('f9', 'f6'), ('f9', 'f2'), ('f9', 'f4')]
model with at-most one parent: »[('f3', 'f4'), ('f5', 'f9'), ('f5', 'f3'), ('f9', 'f8'), ('f9', 'f7'), ('f9', 'f1'), ('f9', 'f6'), ('f9', 'f2')]

The max_indegree parameter can be tuned which returns the best Bayesian Model with at most one parent to each node. Based on the code above, following Bayesian Network is obtained as shown in

Table 6: Most Rare 'and' pattern

Features	Parameters
F1: Initial stroke of formation of a	4: No fixed pattern
F2: Formation of staff of a	4: No staff
F3: Number of arches of n	1: One
F4: Shape of arches of n	5: No fixed pattern
F5: Location of mid-point of n	2: below baseline
F6: Formation of staff of d	1: tented
F7: Formation of initial stroke of d	4: No fixed pattern
F8: Formation of terminal stroke of d	4: No obvious ending stroke
F9: Symbol in place of the word and	1: Formation

Figure 4: Best Model obtained for 'and' data-set

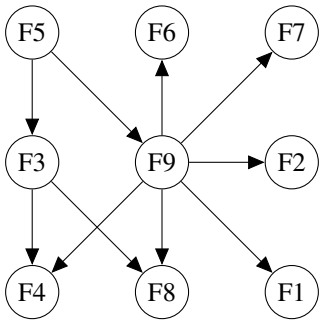


Fig-3. The K2 score obtained for our best model was -9462.704892371388 while K2 score obtained for the model with at-most one parent was -9472.972012154245. Once the best Bayesian Model is created, we compute the conditional probabilities for each node and thus find out the most common and most rare 'and' handwriting pattern. The code to find out conditional probabilities is given below:

```
from pgmpy.estimators import BayesianEstimator
estimator = BayesianEstimator(best_model, data)
for node in best_model.nodes():
cpd = estimator.estimate_cpd(node)
```

Once, we have the conditional probabilities for each node, we find the most common and most rare occurring 'and' pattern. This is shown in Table-5 and Table-6 respectively.

6 Conclusion

In this study, Probabilistic Models were explored for the given 'th' conditional probability data-set and 'and' data-set. We created several probabilistic graphical model (PGM) so that we can evaluate the probability of any given combination of the six feature values of 'th' and nine feature values of 'and'. This process of evaluation is called the process of inference. From the data-sets, we were able to generate conditional probabilities and obtain the best model possible as shown in Fig-1 and Fig-4. We used the model to report the most common and most rare pattern of both 'th' and 'and' in Table-3,4,5 and, 6 respectively. This study can be useful for the next project regarding handwriting comparison.

References

[1] Determining Probabilities of Handwriting Formations using PGMs - Project Description

[2] Advanced Machine Learning Course Slides

[3] PGMPY Documentation: http://pgmpy.org/index.html