CSE 674: Advance Machine Learning (Spring 2019) Project 2- Explainable AI

Arooshi Avasthy
Department of Computer Science
University at Buffalo
arooshia@buffalo.edu

Abstract

This project is to develop a machine learning system that learns explainable features for a task domain while it is learning to answer a variety of queries in that domain. It involves combining deep learning and probabilistic graphical models. We are given M samples of an input, e.g., a photograph of a person, scanned images of a handwritten word written by a known writer. Our goal is to work with three different sets of features: human determined features, deep learning features, explainable deep learning features. We use a discrimination task to compare the performance of each method.

Introduction

Probabilistic Graphical Model (PGM) is a technique of representing Joint Distributions over random variables in a compact way by exploiting the dependencies between them. PGMs use a network structure to encode the relationships between the random variables and some parameters to represent the joint distribution.

There are two major types of Graphical Models: Bayesian Networks and Markov Networks

Bayesian Network: A Bayesian Network consists of a directed graph and a conditional probability distribution associated with each of the random variables. A Bayesian network is used mostly when there is a causal relationship between the random variables. An example of a Bayesian Network representing a student [student] taking some course is shown in Fig 1.

A Bayesian Network consists of a directed graph where nodes represents random variables and edges represent the relation between them. It is parameterized using Conditional Probability Distributions(CPD). Each random variable in a Bayesian Network has a CPD associated with it. If a random varible has parents in the network then the CPD represents P(var|Parvar) i.e. the probability of that variable given its parents.

A BN model B for a set of n variables X = {X1,X2, ... ,Xn} each having a finite set of mutually exclusive states consists of two main components, B ={G; €}. The first component G is a structure that is a directed acyclic graph (DAG) because it contains no directed cycles. The nodes of G correspond to the variables of X, and thus a variable and its corresponding node are usually referred interchangeably. An edge connecting two nodes in G manifests the existence of direct causal influence between the corresponding variables, and the lack of a possible edge in G represents conditional independence (d-separation) between the corresponding variables.
The second component of BN is a set of parameters €, h, that specify all the conditional probability distributions
(or densities) that quantify graph edges.
The joint probability distribution for X given a structure G that is assumed to encode this distribution is given using the set of parameters by

$$P(\boldsymbol{X}|\mathcal{G}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}_i, \mathcal{G})$$

1. Task 1: Annotation

1.1. *Annotate the "AND" images to generate handcrafted features on CEDAR- Handwriting Truthing Tool.*

Categorize each "And" image with respect to given 15 features and form the dataset:
pen_pressure, letter_spacing, size, dimension, is_lowercase, is_continuous, slantness, tilt, entry_stroke_a, staff_of_a, formation_n, staff_of_d, exit_stroke_d, word_formation, constancy

2. Task 2: Bayesian Inference

2.1. *Model Creation: Steps performed*

- Reading 15Features.csv file and extracting the 15 features for the "AND" image and feeding those features to HillClimbSearch Algorithm for finding the best model based on their K2 scores.
- Created different models by changing the indegree parameter in HillCLimbSearch due to which it changes the model structure based on the indegree set i.e. it includes those nodes that satisfy the minimum indegree values set.

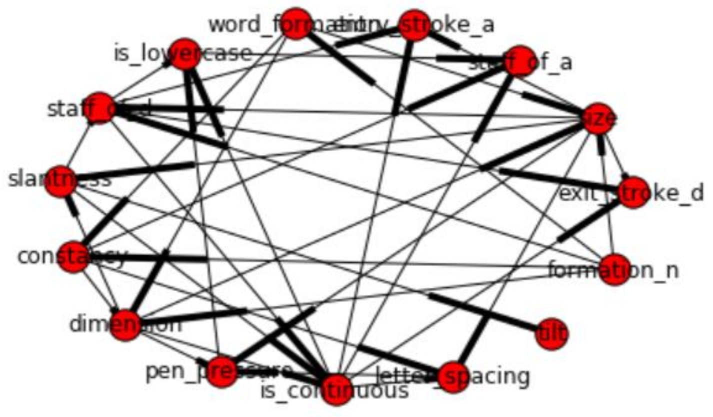
Implementing Hill Climb Search and using K2 score over the dataset to find the best model

We experimentally study the K2 algorithm in learning a Bayesian network (BN) classifier. Starting from an initial BN structure, the K2 algorithm searches the BN structure space and selects the structure maximizing the K2 metric. The K2 uses a greedy search and may impose no restriction on the number of parents a node has. The K2 search begins by assuming that a node has no parents and then adds incrementally that parent from a given ordering whose addition increases the score of the resulting structure the most. We stop adding parents to the node when the score stops to increase. A common scoring metric is the Bayesian score that is, in principle, the posterior probability of a structure G given a random sample D

$$P(\mathcal{G}|D) = \frac{P(D|\mathcal{G})P(\mathcal{G})}{P(D)} = \frac{P(\mathcal{G}, D)}{P(D)}$$

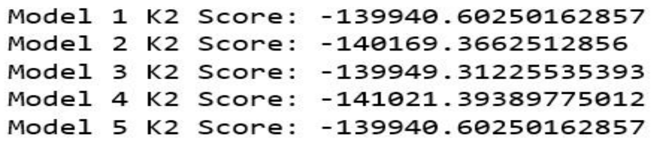
Hill Climb Search is for BayesianModels, is used to learn network structure from data and attempts to find a model with optimal score.

```
hc = HillClimbSearch(Nfeatures, scoring_method=K2Score(Nfeatures))
model2 = hc.estimate(max_indegree=3)
model2 = BayesianModel(model2.edges())
print("model2")
nx.draw(model2, with_labels=True)
plt.show()
```

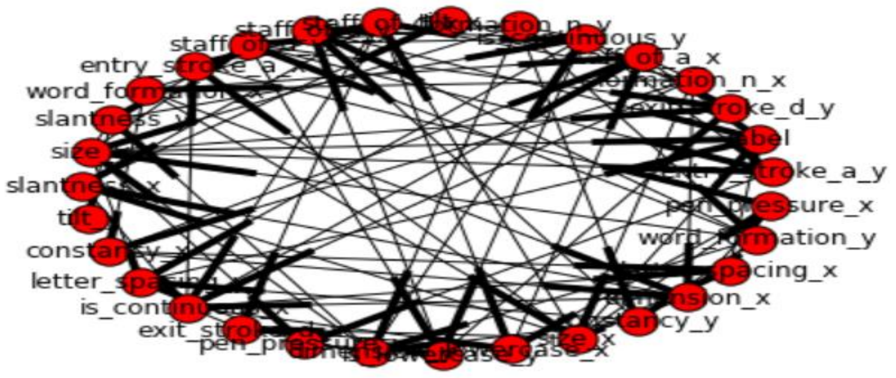


Model formed using 15 features

- Select the model with best K2 score. Make a replica of this model and create a new Bayesian Model that has structural combination of both models and introduce a new node in the model.
- The new node is connected to other nodes in the existing structure based on the maximum indegree values of the existing nodes. The new structure has 31 nodes.

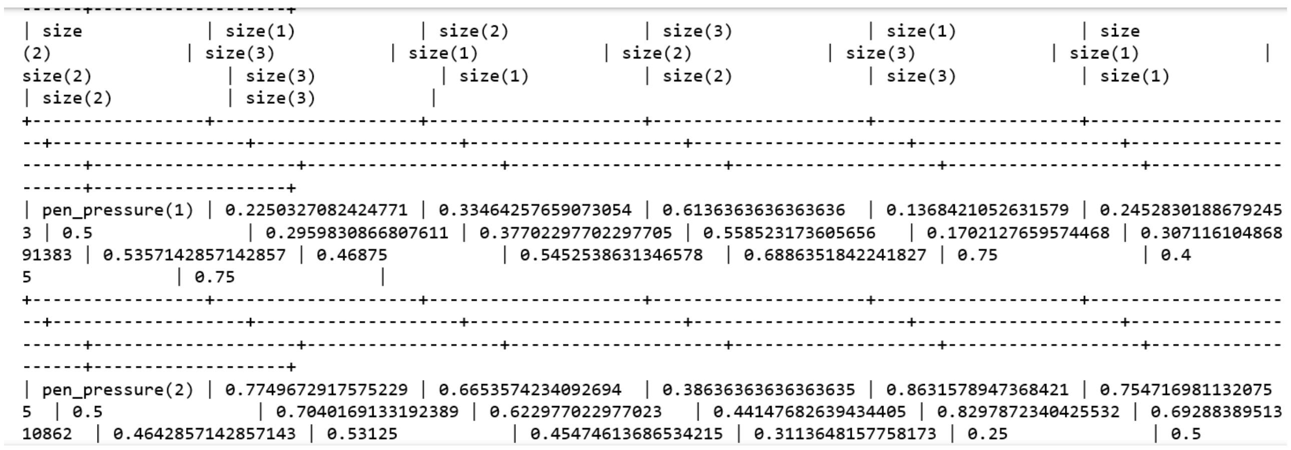


```
Model 1 K2 Score: -139940.60250162857
Model 2 K2 Score: -140169.3662512856
Model 3 K2 Score: -139949.31225535393
Model 4 K2 Score: -141021.39389775012
Model 5 K2 Score: -139940.60250162857
```



Model with 31 nodes

- Generate CPDs for the new model.



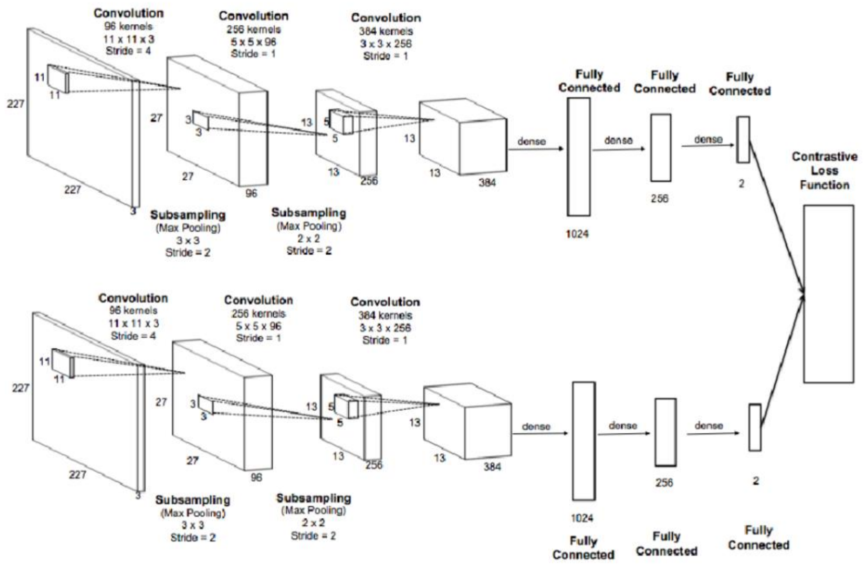
```
------+-------------------+
| size            | size(1)           | size(2)            | size(3)            | size(1)           | size
(2)             | size(3)           | size(1)            | size(2)            | size(3)            | size(1)            |
size(2)            | size(3)            | size(1)            | size(2)            | size(3)            | size(1)
| size(2)            | size(3)            |
+-----------------+-------------------+--------------------+--------------------+-------------------+-------------------
--+-------------------+-------------------+--------------------+--------------------+-------------------+--------------
------+-------------------+-------------------+--------------------+-------------------+------------------+-------------
------+-------------------+
| pen_pressure(1) | 0.2250327082424771 | 0.33464257659073054 | 0.6136363636363636  | 0.1368421052631579 | 0.2452830188679245
3 | 0.5               | 0.2959830866807611 | 0.37702297702297705 | 0.558523173605656   | 0.1702127659574468 | 0.307116104868
91383 | 0.5357142857142857 | 0.46875            | 0.5452538631346578  | 0.6886351842241827 | 0.75               | 0.4
5               | 0.75               |
+-----------------+-------------------+--------------------+--------------------+-------------------+-------------------
--+-------------------+-------------------+--------------------+--------------------+-------------------+--------------
------+-------------------+-------------------+--------------------+-------------------+------------------+-------------
------+-------------------+
| pen_pressure(2) | 0.7749672917575229 | 0.6653574234092694  | 0.38636363636363635 | 0.8631578947368421 | 0.754716981132075
5  | 0.5               | 0.7040169133192389 | 0.622977022977023   | 0.44147682639434405 | 0.8297872340425532 | 0.69288389513
10862  | 0.4642857142857143 | 0.53125            | 0.45474613686534215 | 0.3113648157758173 | 0.25               | 0.5
```

- Fit the new model with data extracted from Seen, Unseen and Shuffled Datasets and calculate accuracy.

Accuracy on seen Dataset is: 46.909492273730685
Accuracy on Unseen Dataset is: 45.844327176781
Accuracy on Shuffled Dataset is: 47.842686219917

3. Task 3: Siamese Network

Siamese networks are neural networks containing two or more identical subnetwork components. Siamese Network is a semi-supervised learning network which produce the embedding feature representation for the input. By introducing multiple input channels in the network and appropriate loss functions, the Siamese Network is able to learn to represent similar inputs with similar embedding features and represent different inputs with different embedding features.


Convolution 96 kernels 11 x 11 x 3 Stride = 4
Convolution 256 kernels 5 x 5 x 96 Stride = 1
Convolution 384 kernels 3 x 3 x 256 Stride = 1
Subsampling (Max Pooling) 3 x 3 Stride = 2
Subsampling (Max Pooling) 2 x 2 Stride = 2
Fully Connected
dense
1024
256
2
Contrastive Loss Function

3.1. *Flow explained*

- Here we have implemented Siamese Network with two input channels. The two identical networks, which are Convolutional Neural Networks (CNN) in this case. Even if the two sister networks are of the same architecture, they do not have to share weights but use distinct weights. Usually, if the inputs are of different "type", the sister networks usually use different architectures, or use distinct weights for the same architecture.

- The two statue images were input into the two channels of the Siamese Network. The L2 distance (Euclidean distance) of the outputs of the two channels were calculated and subjected to the loss function l(x1,x2,δ) minimization. Here, the loss function is a function called contrastive loss.

- **Contrasive Loss function implementation:**

  $$L(I_1, I_2, l) = l d(I_1, I_2)^2 + (1 - l)\max(m - d(I_1, I_2), 0)^2$$

  I1 is the high-dimensional feature vector for input 1, and I2 is the high-dimensional feature vector for input 2. "l" is a binary-valued correspondence variable that indicates whether the two feature vector pair match (l=1) or not (l=0).

  d(I1,I2) is the Euclidean distance of I1 and I2.
  m (m>0m>0) is the margin for non-matched feature vector pair.
  To minimize the loss, d(I1,I2) could neither be too large nor too small, but close to the margin m. If the dimension of feature vector is fixed, increasing the value of margin m may allow better separation of data clusters, but the training time may also increase given other parameters are fixed.

  However, in the implementation, using this exact Contrasive Loss function will cause issues with accuracy because the gradient property for this Contrasive Loss function is not very good.

- **The RMSprop optimizer:**
  The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated.

- **Model Creation:**

```
        input = Input(shape=input_shape)
         x = Flatten()(input)
         x = Dense(128, activation='relu')(x)
         x = Dropout(0.1)(x)
         x = Dense(128, activation='relu')(x)
         x = Dropout(0.1)(x)
         x = Dense(128, activation='relu')(x)
```

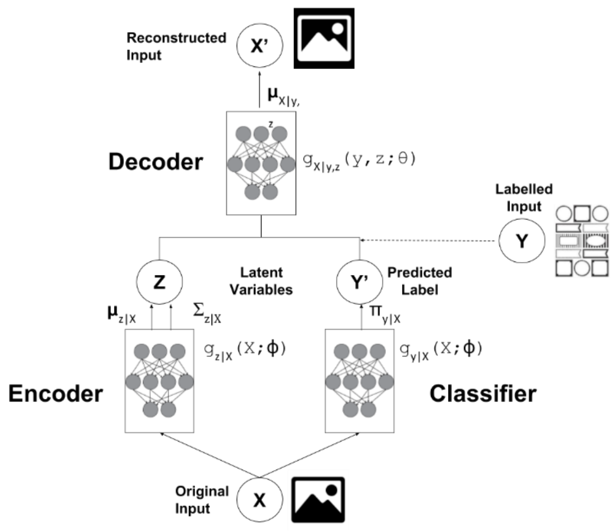
```
Accuracy calculation using Siamese Network:

      Accuracy of shuffled data is 50.0
      Accuracy of Seen data is 50.0
      Accuracy of Unseen data is 50.0
```

4. Task 4: Explainable AI

4.1. Auto Encoder

- The basic framework of autoencoder is a neural network, which comprises an input layer, an output layer and at least one hidden layer. The aim of an autoencoder is to transform inputs into outputs with the least possible amount of deviation. So it is usually used as an information compressor and it contains the encoding and decoding processes. In an autoencoder, the input is encoded into a lower-dimension representation, then decoded into an output with the same size of input.



- This process can be summarized as follows:
  Given input $x_i \in \mathbb{R}^{m\times 1}$, weight matrix $W_1 \in \mathbb{R}^{k\times m}$, $W_2 \in; \mathbb{R}^{m\times k}$, and bias vector $b_1 \in \mathbb{R}^{k\times 1}, b_2 \in \mathbb{R}^{m\times 1}$,

  $$\xi_i = f(W_1 x_i + b_1), \hat{x}_i = f(W_2 \xi_i + b_2).$$

  As the objective of autoencoder is to enforce the output x^i as close as possible to the input xi, we try to minimize the distance of the input and output. The goal of autoencoder is to minimize the reconstruction error using Euclidean distance,

  $$\min_{W_1,b_1,W_2,b_2} \mathcal{J}_r = \sum_{i=1}^{n} \|\hat{x}_i - x_i\|^2.$$

- **Softmax Regression**

  Softmax regression [9] is a generalization of logistic regression, which can handle multi-class classification problems, and the class label y∈{1,2, …, c}, where c≥2 is the number of classes. Given a test input xi, softmax regression can estimate the probability that p(yi=j|xi) for each value of j=1,⋯,c,

  $$p(y_i = j|x_i;\theta) = \frac{e^{\theta_j^{\mathrm{T}} x_\lambda}}{\sum_{l=1}^{c} e^{\theta_l^{\mathrm{T}} x_z}}$$

  After training the model, the probability of a new instance xi belonging to label j can be computed, then xi is labeled to the class which has the biggest conditional probability,

  $$y_i = \arg\max_j \frac{e^{\theta_j^{\top} x_i}}{\sum_{l=1}^{c} e^{\theta_l^{\top} x_i}}.$$

4.2. *Multi Task Learning*

- Regularized multi-task learning is a framework based on the minimization of regularization functions. It learns all tasks simultaneously, and gets the common sub-model shared by all tasks and specific sub-models owned by each task privately.
- Multi-Task learning is a subfield of machine learning where your goal is to perform ***multiple related*** tasks at the same time. The system learns to perform the two tasks simultaneously such that both the tasks help in learning the other task. This is a way to mimic human intelligence i.e. how humans performs multiple tasks at same time.

4.3. *Code Implementation*

We implement autoencoder with Convolutional Network layers in the following given way:

[32 filters - 32x32 kernel - relu activation]
[64 filters - 32x32 kernel - relu activation]
[128 filters - 16x16 kernel- relu activation]
[256 filters - 32x32 kernel- relu activation]
[128 filters - 8x8 kernel- relu activation]
[256 filters - 8x8 kernel- relu activation]

The output of this layer gives 15 latent features where each output corresponds one of the human observed features. This is then fed into a neural network which then predicts the output of one of the 15 observed human features.

Seen, Unseen and Shuffled dataset accuracy is predicted as followed.

```
Accuracy calculation using Siamese Network:

    Accuracy of shuffled data is 62.83
    Accuracy of Seen data is 61.33
    Accuracy of Unseen data is 62.87
```

References

[1] http://pgmpy.org/models.html
[2] https://web.stanford.edu/~jurafsky/slp3/A.pdf
[3] Probabilistic Graphical Models: Principles and Techniques Book by Daphne Koller and Nir Friedman
[4] https://www.stat.auckland.ac.nz/~brewer/stats331.pdf
[5] https://cedar.buffalo.edu/~srihari/CSE574/Chap3/3.5-BayesianModelComparison.pdf
[6] https://ieeexplore.ieee.org/abstract/document/7373449
[7] https://medium.com/@kajalgupta/multi-task-learning-with-deep-neural-networks-7544f8b7b4e3