Unsupervised Machine Learning

DESKTOP-H800RKQ

November 20, 2024

Contents

1 Motivation: Missing/Incomplete Data

Many real-world problems have **hidden (latent) variables** which have incomplete data or values of some attributes missing. This requires **unsupervised learning**.

To deal with missing data:

1. ignore hidden variables, instead drawing connections between other variables directly
2. ignore records with missing values (does not work with variables always missing)

   (a) cannot ignore missing data unless it is missing at random (but it is often correlated with a variable of interest) $\rightarrow$ must model why data is missing

3. maximize likelihood directly: suppose Z is hidden and E is observable with values e

$$h_{ML} = \arg\max_h P(e \mid h) = \arg\max_h \left[\log \sum_Z \prod_{i=1}^{n} P(X_i \mid \text{parents}(X_i), h)_{E=e} \right]$$

   (a) the issue here is that the logarithm cannot be pushed into the sum to linearize

4. if the missing values are known, h_{ML} would be easy

   (a) this can be done with **Expectation Maximization**: guess h_{ML} and iterate

   (b) **expectation**: based on h_{ML}, compute expectation of missing values $P(Z \mid h_{ML}, e)$

   (c) **maximization**: based on expected missing values, compute new estimate of h_{ML}

5. simple version (K-means algorithm)

   (a) **expectation**: based on h_{ML}, compute most likely missing values $\arg\max_Z P(Z \mid h_{ML}, e)$

   (b) **maximization**: based on expected missing values, using complete data, compute new estimate of h_{ML} using ML learning as before

2 Expectation Maximization

k-means algorithm can be used for clustering: dataset of observables with input features X generated by one of a set of classes C

Inputs:

- training examples
- number of classes k

Outputs:

- a representative value for each input feature for each class
- an assignment of examples to classes

Algorithm:

1. pick k means in X, one per class C
2. do the following until means stop changing
   (a) assign examples to k classes (closest to current means)
   (b) re-estimate k means based on assignment

2.1 E Step

Take the missing data, like the class, and fill in the data with probabilities from maximum likelihood. This should have data like the class and the probablity for that class, and let this be $A[X_1, \ldots, X_n, C]$.

2.2 M Step

Compute the statistics for each feature and class:

$$M_i[X_i, C] = \sum_{X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_n} A[X_1, \ldots, X_n, C]$$

$$M[C] = \sum_{X_i} M_i[X_i, C]$$

Here, $M[C]$ is an unnormalized marginal, so this should be normalized by:

$$P(X_i \mid C) = M_i[X_i, C]/M[C]$$

$$P(C) = M[C]/s$$

where s is the corresponding sum. Pseudo-counts can also be added here when normalizing.

2.3 EM Formal

Approximate the maximum likelihood, by starting with a guess h_0. Then iteratively compute

$$h_{i+1} = \arg\max_h \sum_Z P(Z \mid h_i, e) \log P(e, Z \mid h)$$

Here, the **expectation** step is to compute $P(Z \mid h_i, e)$ (fill in missing data). Then, the **maximization** step is to find a new h that maximizes the above sum.

Note that

$$\log P(e \mid h) \geq \sum_Z P(Z \mid e, h) \log P(e, Z \mid h)$$

Since EM finds a local maximum of the right side, this is a lower bound of the left side. Further, the log inside the sum can linearize the product:

$$h_{i+1} = \arg\max_h \sum_Z P(Z \mid h_i, e) \sum_{j=1}^{n} \log P(X_i \mid \text{parents}(X_i), h)_{E=e}$$

EM monotonically improves the likelihood, so $P(e \mid h_{i+1}) \geq P(e \mid h_i)$.

2.4 General Bayes Network EM

With complete data, use Bayes Net Maximum Likelihood:

With incomplete data, use Bayes Net Expectation Maximization. With observed variables X and missing variables Z, start with some guess for θ.

In the E step: compute weights for each data x_i and latent variables z_j (using variable elimination)

$$w_{ij} = P(z_j \mid \theta, x_i)$$

In the M step: update parameters

$$\theta_{V=j,\text{parents}(V)=v} = \frac{\sum_i w_{ij} \mid V = j \wedge \text{parents}(V) = v \text{ in } \{x_i, z_j\}}{\sum_{ij} w_{ij} \mid \text{parents}(V) = v \text{ in } \{x_i, z_j\}}$$

3 Belief Network Structure Learning

$$P(\text{model} \mid \text{data}) = \frac{P(\text{data} \mid \text{model}) \times P(\text{model})}{P(\text{data})}$$

A model here is a belief network. A bigger network can always fit data better. $P(\text{model})$ allows encoding of a preference for smaller networks. Can search over network structure, looking for the most likely model.

Can do **independence tests** to determine which features should be the parents. Just because features do not give information individually does not mean they will not give info in combination.

The ideal method is to search over total orderings of variables.

4 Autoencoders

A representation learning algorithm that learns to map examples to low-dimensional representation.

An autoencoder has two main components:

1. **encoder** $e(x)$: maps x to low-dimensional representation $\hat{z}$
2. **decoder** $d(\hat{z})$: maps $\hat{z}$ to its original representation x

The autoencoder implements $\hat{x} = d(e(x))$, so $\hat{x}$ is the reconstruction of the original input x. The encode and decoder learn such that $\hat{z}$ contains as much info about x as is needed to reconstruct it.

The goal is the minimize the sum of squares of differences between the input and prediction:

$$E = \sum_i (x_i - d(e(x_i)))^2$$

Deep neural network autoencoders are good for complex inputs, where e and d are feedforward neural networks, joined in series. The network is then trained with backpropagation.

5 Generative Adversarial Networks

A generative unsupervised learning algorithm that learns to generate unseen examples that look like training examples.

GANs are a pair of neural networks:

1. **generator** $g(z)$: given vector z in latent space, produce an example x drawn from a distribution that approximates the true distribution of training examples, where z is usually sampled from a Gaussian distribution
2. **discriminator** $d(x)$: a classifier that predicts whether x is real (from training set) or fake (made by g)

GANs are trained with a minimax error:

$$E = \mathbb{E}_x[\log(d(x))] + \mathbb{E}_z[\log(1 - d(g(z)))]$$

where the discriminator tries to maximize E and the generator tries to minimize E. Once they converge, g should be producing realistic examples and d should output $\frac{1}{2}$ to indicate maximal uncertainty.