Chapter 20 – Learning Probabilistic Models

1. Bayesian Learning – calculates the probability of each hypothesis, given the data, and makes predictions on that basis
   - "predictions are made by using all the hypotheses, weighted by their probabilites, rather than by using the single best hypothesis."
     - Probability of each hypothesis $P(h_i|\boldsymbol{data}) = \alpha P(\boldsymbol{d}|h_i) P(h_i)$
       - where $P(\boldsymbol{data}|h_i)$ is the likelihood of the data under each hypothesis
         - assuming i.i.d. $P(\boldsymbol{data}|h_i) = \prod_j P(d_j|h_i)$
       - where $P(h_i)$ is the prior for each hypothesis
     - Probability of quantity X: $P(X|\boldsymbol{data}) = \sum_i P(X|h_i) P(h_i|\boldsymbol{data})$
       - this Bayesian prediction eventually agrees with the true hypothesis
   - Bayesian learning is optimal, but can have huge hypothesis spaces for real problems
   - One solution is to approximate by making predictions using only the most probable hypothesis $P(h_i|\boldsymbol{data})$ (this is MAP hypothesis)
     - This works since $P(X|data) \approx P(X|h_{MAP})$ as we obtain more and more data
     - Finding h_{MAP} is much faster than finding $P(X|\boldsymbol{data})$
   - Bayesian and MAP learning use the prior to penalize hypothesis complexity
     - there are many more complex hypothesis than simpler ones, so they are less common
   - Fun fact: Choosing h_{MAP} to minimize $P(h_i|\boldsymbol{data})$ is equivalent to minimizing $-\log_2 P(\boldsymbol{data}|h_i) - \log_2 P(h_i)$
     - where $-\log_2 P(h_i)$ equals the # of bits needed to describe the hypothesis
     - where $-\log_2 P(\boldsymbol{data}|h_i)$ equals the # of bits needed to describe the data
     - basically, MAP learning gives us a hypothesis that maximally compresses the data
       - does about the same thing as the MDL learning method
   - We can further simplify by assuming a uniform distribution of priors
     - In this case we want the hypothesis that maximizes $P(\boldsymbol{data}|h_i)$
       - this is called the maximum-likelihood hypothesis and is reasonable to use when there is no reason to prefer particular hypothesis and they are equally complex
2. Learning with complete data – we want to learn parameters for fixed-structure probability models
   - Maximum-likelihood parameter learning: Discrete models
     - Standard steps in the algorithm:
       1. Write an expression for the likelihood of the data as a function of the parameters
       2. Write down the derivative of the log likelihood w.r.t. each parameter
       3. Find the parameter values such that the derivatives are zero
     - step 3 can be hard so we often need to resort to more advanced algorithms
     - Caution: initialize counts to 1 to avoid 0 probabilities for unseen events
     - IMPORTANT: With complete data, the maximum-likelihood parameter learning problem for Bayesian networks decomposes the problem into separate learning problems for each parameter.
     - Also: the parameter values for a variable, given its parents, are just the observed frequencies of the variable values for each combination of parent values.
   - Naive Bayes Model – it is naive because it assumes that the attributes are conditionally independent of each other given the class

- For a class C and variable X with observed values $x_1, \ldots, x_n$ the probability of each class is given by: $P(C|x_1, ..., x_n) = \alpha P(C) \prod_i P(x_i|C)$
- It scales well, needs no search to find h_{ML} , and handles noisy/missing data

- Maximum-likelihood parameter learning: Continuous Models – the principles for learning ML learning parameters of continuous models are the same as in the discrete case
  - The parameters for the linear Gaussian model are the mean and standard deviation
    - The ML value for each is the sample average and the square root of the variance
  - For a linear Gaussian model with Y dependent on X, maximizing the likelihood is the same as minimizing the L_2 loss (linear regression)
- Bayesian parameter learning – this approach starts by defining a prior probability distribution over the possible hypotheses.
  - The hypothesis prior is the distribution $P(\boldsymbol{data}|h_i)$ of the random variable $\boldsymbol{\theta}$
    - $P(\boldsymbol{\theta})$ must be continuous, nonzero from 0 to 1, and integrate to 1
    - The Uniform distribution can be a good choice (it is in the beta distribution family)
  - $beta[a,b](\theta) = a\,\theta^{a-1}(1-\theta)^{b-1}\ for\ \theta\ \ between[0,1]\ with mean = \frac{a}{(a+b)}$
    - large values of a suggest theta is closer to 1 than 0
    - large values of a+b suggest we are more certain about the value of theta
    - after a beta prior is updated, the posterior is also a beta distribution
  - When there are more than one parameter in the hypothesis prior, we usually assume parameter independence: $P(\theta, \theta_1, \theta_2) = P(\theta)P(\theta_1)P(\theta_2)$
    - Now, each parameter has its own beta distribution that is updated as data arrives
    - Basically, we add new evidence nodes to the network and query the parameter nodes
- Learning Bayes net structures – the structure can often be inferred from causal assumptions
  - To learn a structure we must search by making modifications and comparing accuracy
  - For a structure to be good, we must use a statistical significance test to see if the the conditional independence assertions in the structure are supported by the data
    - the lower the threshold of this test, the higher the risk of overfitting
    - Markov Chain Monte Carlo can be used to sample structures from the search space
- Density estimation with nonparametric models – Given data points, can we recover the distribution from whence it came?
  - We can apply k-nearest-neighbors to learn the density of data around query points
    - choose k with cross-validation
  - We can use kernel functions (Gaussian) too. Assuming that each data point generates its own density function, the estimated density at a query point x is then the average density as given by each kernel function:

    $P(\boldsymbol{x}) = \frac{1}{N}\sum_{j=1}^{N} K(\boldsymbol{x}, \boldsymbol{x_j})\ where\ K(\boldsymbol{x}, \boldsymbol{x_j}) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{-D\frac{(\boldsymbol{x}, \boldsymbol{x_j})^2}{2w^2}}$
    - where d is the number of dimensions in x and D is the Euclidean distance function
    - choose w with cross-validation

3. Learning with Hidden Variables: The EM Algorithm – Learning Bayesian networks with hidden variables lets us learn fewer parameters, but it's not obvious how to learn where in the network hidden variables go
   - Unsupervised clustering: Learning mixtures of Gaussians – we assume that some data is generated from a mixture distribution with k components, but we know neither the assignments of the data or the parameters of the distributions.

- Let C be a component with values 1 to k, then $P(\boldsymbol{x}) = \sum_{i=1}^{k} P(C=i) P(\boldsymbol{x}|C=i)$
- The parameters to the Gaussian mixture are the weights, means, and covariances of C
- The EM approach is to pretend we know the model parameters and infer the probability that each data point belongs to each component.
  1. Compute the probabilities $p_{ij} = P(C=i|\boldsymbol{x_j})$, the probability that datum x_j was generated by component i. Basically, compute the expected values p_{ij} of the hidden indicator variables Z_{ij} where Z_{ij} is 1 if datum $\boldsymbol{x_{ij}}$ was generated by the ith component and 0 otherwise.
  2. Compute new mean, covariance, and component weights. Basically, find the new parameters that maximize the log likelihood of the data given the expected values of the hidden indicator variables.
- EM increases the log likelihood of the data at every itereation. It is similar to gradient descent hill climbing, but has no step size parameter.

- Learning Bayesian networks with hidden variables ?????????
  - ….
  - "the parameter updates for bayesian network learning with hidden variables are directly available from the results of inference on each example"
- Learning hidden Markov models - ????????????
  - ….
- The general form of the EM algorithm - "compute the expected values of hidden variables for each example and then recompute the parameters, using the expected values as if they were observed values"
  - Let x be all the observed values in all the examples, let Z be all the hidden variables for all the examples, and let theta be all the parameters for the probability model, EM = $\theta^{i+1} = \underset{\theta}{argmax}\, P(Z=z|x,\theta^{i})\, L(x, Z=z|\theta)$
    - where $P(Z=z|x,\theta^{i})$ is the posterior over the hidden variables, given the data
- Learning Bayes net structures with hidden variables – we can either pretend the data is complete and possibly learn a parameter-intensive model or add new hidden variables to it to simplify the model
  - Unfortunately, calculating posteriors in a Bayes net is an NP hard problem