Chapter 18 – Learning from Examples

1. Forms of Learning
    - Motivation – machine learning helps solve problems where the designers can't anticipate all situations the agent might experience, can't anticipate changes in situations over time, and have no idea how to program the solution themselves.
    - 4 major factors for learning -
        - which component of the agent is to be improved
        - what prior knowledge the agent has
        - what representation is used for the data and the components
            - factored representation – a vector of attribute values
        - what feedback is available to learn from
            - Unsupervised Learning – an agent learns patterns in the input even though no explicit feedback is supplied.
            - Supervised Learning – an agent observes some input-output pairs and learns a function that maps from input to output.
            - Reinforcement Learning – an agent learns from a series of reinforcements (rewards or punishments)
2. Supervised Learning
    - Given a training set where each output y was generated by an unknown function y = f(x), discover a function h that approximates the true function f.
    - Learning is a search through the hypothesis space for a function h that generalizes well.
        - A consistent hypothesis agrees with all of the data
    - Classification – predicting output y that is one of a finite set of values
    - Regression – predicting output y that is a number
    - Ockham's Razor – prefer the simplest hypothesis consistent with the data
    - Main idea - $best\ hypothesis = \underset{h \in H}{argmax}\, P(h|data)$ == $\underset{h \in H}{argmax}\, P(data|h)P(h)$
    - There is a tradeoff between the expressiveness of a hypothesis space and the complexity of finding a good hypothesis within that space
3. Learning Decision Trees
    - Decision trees represent a function that takes as input a vector of attribute values and returns a single output value.
    - Any function in propositional logic can be represented by a decision tree.
    - We want a decision tree that is consistent with the examples and is as small as possible.
    - Take a greedy divide & conquer approach to learning the trees (try to minimize tree depth)
    - Binary classification DT learning algorithm cases:
        - All remaining examples have the same label: we are done, answer yes or no
        - Some positive and some negative examples: choose best attribute to split them
        - No examples left: no example observed for this combination of attribute values, classify node as most frequent class among remaining examples
        - No attributes left, but positive and negative examples remain: Noise in data, select class
    - Use learning curves to monitor learning progress
    - Entropy – a measure of the uncertainty of a random variable (gaining info reduces entropy)
        - $Entropy = H(V) = -\sum_k P(v_k)\log_2 P(v_k)$ where V is a RV and v_k is its values.
        - $Remainder(A) = \sum_k \frac{p_k + n_k}{p+n} H(V)$
        - $Gain(A) = H(V) - Remainder(A)$

- Overfitting – becomes more likely as the hypothesis space and the number of input attributes increases, and less likely as we increase the number of training examples.
- Pruning – technique to reduce overfitting by removing irrelevant nodes (info gain ≈ 0)
  - X^2 *Pruning* - How large of a gain should we require to split on an attribute?
    - Use statistical significance test to determine how likely it is to observe certain patterns in the samples assuming there are no underlying patterns.
    - We need to calculate the probability that, under the null hypothesis, a sample of size v = n + p would exhibit the observed deviation from the expected distribution of positive and negative examples.
    - $Expected\ Numbers: \hat{p}_k = p\frac{p_k+n_k}{p+n} \qquad \hat{n}_k = n\frac{p_k+n_k}{p+n}$
    - $Total\ Deviation = \sum_{k=1}^{d} \frac{(p_k-\hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k-\hat{n}_k)^2}{\hat{n}_k}$
    - The total deviation should follow the chi-squared distribution with n-1 DOF
- Potential Issues:
  - Missing data – what do we do about missing values?
  - Multivalued attributes – worst case = each example has a different attribute value
  - Continuous & integer-valued input attributes – find a split point that gives the highest information gain (very expensive computation)
  - Continuous-valued output attributes – we'll need a regression tree, where each leaf node contains a linear function using some subset of attributes as input.

4. Evaluating and Choosing the Best Hypothesis
- We want a hypothesis that fits future data the best
  - 'future data' = we make the stationary assumption that the probability distribution of the examples does not change much. (examples are independent and identically distributed.)
  - 'best' = use error rate (proportion of mistakes made) to determine quality of hypothesis
  - holdout cross-validation – train on 90% of data and test on the other 10%
  - k-fold cross-validation – split data into k subsets, perform k rounds of learning where on each round 1/k of data is used for testing. Avg score over k rounds is a better metric.
  - Use a validation set if using data to tune hyperparameters
- Model selection – is the task of defining the hypothesis space, while optimization is the process of finding the best hypothesis within that space.
- Loss function – the amount of utility lost by predicting $h(x)=\hat{y}$ when the correct answer is $f(x)=y$
  - $Loss(x, y, \hat{y}) = Utility(y|x) - Utility(\hat{y}|x)$
  - $L_1(y, \hat{y}) = (|y-\hat{y}|)$
  - $L_2(y, \hat{y}) = (y-\hat{y})^2$
  - $L_{0/1}(y, \hat{y}) = 0\ if\ y=\hat{y}, else\ 1$ okay for discrete-valued inputs
  - Generalization Loss = $GenLoss_L(h) = \sum_{(x,y)\in E} L(y, h(x))P(x, y)$
    - best hypothesis $h^* = \underset{h\in H}{argmin}\, GenLoss_L(h)$
  - P(x,y) is not known so we estimate GenLoss with the empirical loss
    - $EmpLoss_{L,E}(h) = \frac{1}{N}\sum_{(x,y)\in E} L(y, h(x))$
- 4 reasons that our hypothesis may differ from the true function: unrealizability, variance, noise, and computational complexity

- Regularization - Let's search for a hypothesis that directly minimizes the weighted sum of empirical loss and hypothesis complexity: $Cost(h) = EmpLoss(h) + \lambda Complexity(h)$
  - where λ becomes a hyperparameter tuned on the validation set
- Feature selection – discard attributes that appear to be irrelevant (Chi-squared pruning)
  - Minimum description length: ??????

5. The Theory of Learning
   - Probably Approximately Correct(PAC) – any hypothesis that is seriously wrong will almost certainly be found out with high probability after a small number of examples, because it will make an incorrect prediction. Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: that is, it must be PAC.
     - Assumptions: Stationary distribution & h() is deterministic and known to be in H
   - Sample Complexity – If a learning algorithm returns a hypothesis that is consistent with $\frac{1}{\epsilon}(\ln(\frac{1}{\delta}) + \ln|H|) \leq N$ examples, then with probability at least 1 – δ, it has error ≤ ϵ
     - We can restrict the search space by using prior knowledge, ignoring complex hypothesis, and only exploring subsets of the search space.
   - Ex: Learning Decision Lists (see book)
6. Regression and Classification with Linear Models
   - Univariate Linear Regression – (line fitting) simply adjust weights to minimize loss
     - If loss equations have no closed-form solution, use local-search optimization
   - Gradient Descent – continuously adjust function weights so as to minimize loss
     - step size – parameter α (learning rate) that determines how much to adjust weights
     - batch-GD – 1 step along the gradient per N examples. Guaranteed convergence.
     - stochastic-GD – 1 step along the gradient per example. Not guaranteed to converge with a fixed learning rate, but can use simulated annealing on α to fix this.
   - Multivariate Linear Regression – extension of univariate lin. reg. where vectors are input
     - $h_{sw}(x_j) = w \cdot x_j = w^T x_j = \sum_i w_i x_{j,i}$ and $optimal\ weights = \underset{w}{argmin} \sum_j L_2(y_j, h_{sw}(x_j))$
     - Normal Equation Method - $optimal\ weights = (X^T X)^{-1} X^T y$ (set gradient to 0 and solve for weights)
     - For regularization, let $Complexity(h_w) = L_q(w) = \sum_i |w_i|^q$
       - L_1 regularization provides sparse models (discards attributes it deems irrelevant)
       - L_1 regularization requires log(N) examples(L_2 requires N)
       - L_2 regularization is rotationally variant and L_1 is not.
   - We can do classification by turning a fit line into a decision boundary and using a threshold function: $h_w(x) = Threshold(w \cdot x)\ where\ Threshold(z) = 1\ if\ z \geq 0,\ 0\ otherwise$
     - Can use the same gradient descent update rule as linear regression
     - Since this threshold is discontinuous and non-differentiable, we need a softer threshold
       - So we use the logistic function: $h_w(x) = Logistic(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}$
         - now the output is the probability of belonging to a class
   - Logistic Regression - fitting weights to the logistic function. Do so with gradient descent.
     - Gradient descent works the same but substitute in the new hypothesis function
     - $Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$
7. Artificial Neural Networks
   - Artificial neural networks are composed of nodes connected by directed links.
     - A link from node i to node j serves to propagate the activation a_i from i to j.

- Each link also has a weight $w_{i,j}$ associated with it, which determines the strength and sign of the connection.
- Each unit j first computes a weighted sum of its inputs: $in_j = \sum_{i=0}^{n} w_{i,j} a_i$
- Apply an activation function g to the sum to get the output: $a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i)$
  - hard threshold = perceptron and soft threshold = sigmoid perceptron

- Feed-forward network – has connections only in one direction. No state other than weights.
- Recurrent network – feeds outputs back into its own inputs. (Short term memory)
- Multilayer networks -
  - when there are multiple outputs treat them collectively as an output vector **y**
  - the error at the output layer $\boldsymbol{y} - \boldsymbol{h_w}$ needs to be propagated back to the hidden layers
- Backpropagation – emerges directly from the derivation of the overall error gradient
  - Compute the Δ values for the output units, using the observed error
  - Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
    - Propagate the Δ values back to the previous layer
    - Update the weights between the two layers
  - See algorithm and derivation in book (page 734)
- We can choose network architecture based on the cross-validation performance
  - If we consider networks that are not fully connected, then the search space is very large
  - Can use the optimal brain damage algorithm, where connections and units are dropped such that the model's performance is unaffected
  - Can also use a tiling algorithm, where recursively the unit that satisfies the most examples is added to the network

8. Nonparametric Models
   - Parametric Model - a learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples)
   - Nonparametric Model – one that cannot be characterized by a bounded set of parameters
     - Instance/memory-based learning – hypothesis retains all of the training examples and uses them to make predictions. Simplest example: a lookup table
   - K-nearest neighbors – given a table and a query $\boldsymbol{x_q}$ find the k examples nearest to $\boldsymbol{x_q}$
     - to do classification, we find the k-nearest neighbors and choose the most frequent class
       - avoid ties by always choosing an odd k
     - cross-validation can select the best value of k
     - Distances measured with the Minkowski distance: $L^p(\boldsymbol{x_j}, \boldsymbol{x_q}) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{\frac{1}{p}}$
       - where p = 2 = Euclidean distance, p = 1 = Manhattan distance, and for boolean attribute values the # of attributes where two points differ is the Hamming distance
     - We must apply normalization to the dimensions if we want consistent neighbors
       - with mean μ_i and std. dev. σ_i let $x_{j,i}$ become $(x_{i,j} - \mu_i)/\sigma_i$
       - or use the Mahalanobis distance which factors in covariance between dimensions
     - k-d trees - used for O(log N) lookups. See algorithm in book (page 739)
       - only really useful when we have 2^n examples compared to n dimensions
     - locality-sensitive hashing – Idea: try to get neighbors to hash into the same bin
       - the probability of points sharing a hash value needs to be high for near points

        - the trick is to create multiple hash tables using different random projections (random subset of bit-string representations) and combine them using the union op
    - Curse of dimensionality – Data points appear more like outliers in higher dimensions
    - Nonparametric Regression – can simply "connect the dots" when noise is low
        - Locally weighted regression – Idea: at each point x_q the examples close to it are weighted heavily and the ones far from it are weighted lightly or not at all
            - We decide the weight with a kernel(proximity) function $K(Distance(x_j, x_q))$
            - For a query point x_q we solve $\boldsymbol{w}' = \underset{\boldsymbol{w}}{argmin} \sum_j K(Distance(\boldsymbol{x_q}, \boldsymbol{x_j}))(y_j - \boldsymbol{w} \cdot \boldsymbol{x_j})^2$ where the answer is $h(\boldsymbol{x_q}) = \boldsymbol{w}' \cdot \boldsymbol{x_q}$
9. Support Vector Machines
    - SVMs construct a max margin separator(generalizes well), can embed data into higher-dimensional spaces using the kernel-trick(can explore more hypotheses), and are a nonparametric method(retain training examples).
    - Minimize expected generalization loss instead of expected empirical loss
        - the separator is defined as: $\{\boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0\}$ and we can search the space of w and b w/ G.D. to find the params that maximize the margin and correctly classify all examples.
        - Can also solve as a quadratic programming problem using the "dual representation"
    - If data is not linearly-separable try to map it to dimensions that are (HOW?)
        - use a kernel function on the input and substitute into a learning algorithm
        - Mercer's theorem – any reasonable (matrix $\boldsymbol{K_{jk}} = K(\boldsymbol{x_j}, \boldsymbol{x_k})$ is positive definite) kernel function corresponds to *some* feature space(which can be very large).
            - Polynomial kernel $K(\boldsymbol{x_j}, \boldsymbol{x_k}) = (1 + \boldsymbol{x_j} \cdot \boldsymbol{x_k})^d$ has feature space exponential in d
    - Kernel trick – plug kernels(mapping algorithms) into learning algorithm and linear separators can be found efficiently in feature spaces with billions+ dimensions
        - can use a soft margin to allow (but penalize) some incorrectly classified examples
        - works on any algorithm that can be reformulated to work on dot products of ex pairs
10. Ensemble Learning
    - Idea: combine the predictions of a collection/ensemble of hypothesis from the search space
        - works best when hypotheses are different enough to reduce correlation between errors
        - allows us to learn a more expressive class of hypotheses
    - Boosting – start with a weighted training set, then learn a hypothesis, then increase (decrease) the weights of incorrectly (correctly) classified examples, then learn a new hypotheses. Repeat this K times. The final hypothesis is a weighted-majority combination of all K hypothesis, each weighted according to its performance on the training set.
        - AdaBoost – if learning algorithm is a weak learner (better than random), then AdaBoost will reach 100% on the training set for large enough K.
        - Predictions actually improve as the ensemble hypothesis gets more complex!
            - Boosting might approximate Bayesian learning (optimal learning algorithm)
    - If the data are not independent and normally distributed, try an online learning algorithm with a randomized weighted majority algorithm (see page 752) to select a hypothesis
        - technique: no-regret learning
11. Practical Machine Learning
    - Case Study: Handwritten Digit recognition
        - 3-nearest-neighbors: is slow and needs a lot of memory (error=2.4%)
        - single-hidden-layer neural net (300 hidden units) – (error=1.6%)
        - specialized neural net (LeNet) – (error=0.9%)
        - boosted neural net – (error=0.7%)

    - support vector machine – (error=1.1%)
    - virtual(specialized) svm – best algorithm so far(error=0.56%)
    - Shape matching – 3-nearest-neighbors with a smarter distance algorithm (error=0.63%)
- Case Study: Word senses and house prices -
- Decision trees are generally good when there are a lot of discrete features and many of them may be irrelevant
- Nonparametric methods are generally good when we have a lot of data but no prior knowledge, and we don’t want to worry about choosing the correct features.

12. Summary, etc