CS3244 Machine Learning

AY2021/22 Semester 2

1. Concept Learning

1.1. Definitions

- Concept: A boolean-valued function over a set of input instances (each comprising input attributes).
- Hypothesis Space (H): Each hypothesis $h \in H$ ($h: X \to \{0,1\}$) is represented by a conjunction of constraints on input attributes.
    - Each constraint can be a specific value, don't care (?), or no value allowed (∅).
    - An input instance $x \in X$ satisfies a hypothesis $h \in H$ if and only if $h(x) = 1$. In other words, h classifies x as a +ve example.
- Concept Learning: Given an unknown target concept $c: X \to \{0,1\}$ and noise-free training examples D: +ve and -ve examples of the target concept, determine a hypothesis $h \in H$ that is consistent with D.
    - A hypothesis h is consistent with a set of training examples D if and only if $h(x) = c(x)$ for all $\langle x, c(x)\rangle \in D$.
    - h is consistent with D if and only if every +ve training instance satisfies h and every -ve training instance does not satisfy h.
- Inductive Learning Assumption: Any hypothesis found to approximate the targe function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.
- Version Space ($VS_{H,D}$): The version space ($VS_{H,D}$) w.r.t. hypothesis space H and training examples D, is the subset of hypotheses from H consistent with D:
  $VS_{H,D} = \{h \in H \mid h \text{ is consistent with } D\}$
    - If $c \in H$, then a large enough D can reduce $VS_{H,D}$ to $\{c\}$.
    - If D is insufficient, then $VS_{H,D}$ represents the uncertainty of what the target concept is.
    - The general boundary G of $VS_{H,D}$ is the set of maximally general members of H consistent with D.
    - The specific boundary S of $VS_{H,D}$ is the set of maximally specific members of H consistent with D.
    - Version Space Representation Theorem (VSRT):
      $VS_{H,D} = \{h \in H \mid \exists s \in S\ \exists g \in G, g \geq_g h \geq_g s\}$
    - An input instance x satisfies every hypothesis in $VS_{H,D}$ if and only if x satisfies every member of S.
    - An input instance x satisfies none of the hypotheses in $VS_{H,D}$ if and only if x satisfies none of the members of G.

1.2. Properties of Hypotheses

- h_j is more general than or equal to h_k ($h_j \geq_g h_k$) if and only if any input instance x that satisfies h_k also satisfies h_j:
  $$\forall x \in X, (h_k(x) = 1) \to (h_j(x) = 1)$$
    - $\geq_g$ relation defines a partial order (reflexive, antisymmetric and transitive).
    - h_j is more general than h_k if and only if $h_j \geq_g h_k$ and $h_k \ngeq_g h_j$.
    - h_j is more specific than h_k if and only if h_k is more general than h_j.
- Two hypotheses are syntactically distinct if any one of their attributes (specific values, ?, ∅) are different.
- Two hypotheses are semantically distinct if any one of their attributes (specific values, ?) are different. Specially, all hypotheses containing ∅ are semantically identical.

1.3. Find-S Algorithm

```
1. Initialise $h$ to most specific hypothesis in $H$
2. For each positive training instance $x$:
      For each attribute constraint $a_i$ in $h$:
         If $x$ satisfies $a_i$ in $h$:
            Do nothing
         Else:
            Replace $a_i$ by the next more general constraint that
            is satisfied by $x$
3. Output hypothesis $h$
```

- General idea: Start with most specific hypothesis. Whenever it wrongly classifies a +ve training example as -ve, minimally generalize it to satisfy the input instance.
- Correctness: Suppose that $c \in H$. Then, h_n is consistent with $D = \{\langle x_k, c(x_k)\rangle\}_{k=1,2,\dots,n}$.
- Limitations:
    - Cannot tell whether Find-S has learnt target concept.
    - Cannot tell when training examples are inconsistent (contain error or noise).
    - Picks a maximally specific h.
    - Depending on H, there might be many maximally specific h.

1.4. List-Then-Eliminate Algorithm

```
1. $VersionSpace$ ← a list containing every hypothesis in $H$
2. For each training example $\langle x, c(x)\rangle$:
      Remove from $VersionSpace$ any hypothesis $h$ for which
      $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$
```

- General idea: Start with all hypotheses in H. Then eliminate any hypothesis found inconsistent with any training example.
- Limitations:
    - Prohibitively expensive to exhaustively enumerate all hypotheses in finite H.

1.5. Candidate-Elimination Algorithm

```
1. $G$ ← maximally general hypotheses in $H$
2. $S$ ← maximally specific hypotheses in $H$
3. For each training example $d$:
      If $d$ is a +ve example:
         Remove from $G$ any hypothesis inconsistent with $d$
         For each $s \in S$ not consistent with $d$:
            Remove $s$ from $S$
            Add to $S$ all minimal generalisations $h$ of $s$ such that
               $h$ is consistent with $d$ and some member of $G$ is
               more general than or equal to $h$
            Remove from $S$ any hypothesis that is more general
               than another hypothesis in $S$
      If $d$ is a -ve example:
         Remove from $S$ any hypothesis inconsistent with $d$
         For each $g \in G$ not consistent with $d$:
            Remove $g$ from $G$
            Add to $G$ all minimal specialisations $h$ of $g$ such that
               $h$ is consistent with $d$ and some member of $S$ is
               more specific than or equal to $h$
            Remove from $G$ any hypothesis that is more
               specific than another hypothesis in $G$
```

- Limitations:
    - S and G may reduce to ∅ with error/noise in training data.
    - S and G may reduce to ∅ with insufficiently expressive hypothesis representation (biased hypothesis space where $c \notin H$).
- Active learner should query input instance that satisfies exactly half of hypotheses in version space, so that version space reduces by half with each training example and requires at least $\lceil \log_2 |VS_{H,D}| \rceil$ examples to find target concept c.
- Inductive Bias: $B = \{c \in H\}$

2. Decision Tree

2.1. Decision Tree Algorithm

- Number of distinct binary decision trees with m Boolean attributes: 2^{2^m}

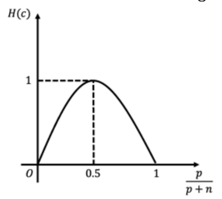
```
function $LEARN(examples, attributes, parent\_examples)$:
  if $examples$ is empty then return $PLURALITY\_VALUE(parent\_examples)$
  else if all $examples$ have the same classification then return the classification
  else if $attributes$ is empty then return $PLURALITY\_VALUE(examples)$
  else:
    $A \leftarrow argmax_{a \in attributes} \text{Imporance}(a, examples)$
    $tree \leftarrow$ a new decision tree with root test $A$
    for each value $v_k$ of $A$ do:
      $exs \leftarrow \{e: e \in examples \text{ and } e.A = v_k\}$
      $subtree \leftarrow \text{Learn}(exs, attributes - A, examples)$
      add a branch to $tree$ with label ($A = v_k$) and subtree $subtree$
    return $tree$
```

<u>2.2. Entropy and Information Gain</u>

- Entropy: Entropy measures uncertainty of $c \in \{c_1, \dots, c_k\}$:

$$H(c) = E(\log_2 P(c_i)) = -\sum_{i=1}^{k} P(c_i) \log_2 P(c_i)$$

  - Entropy of Boolean function that is true with probability q:
  $$B(q) = -q \log_2 q - (1-q) \log_2 (1-q)$$
  - Entropy of target concept c with a training set containing p +ve examples and n -ve examples:
  $$H(c) = B\left(\frac{p}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$



  For example,
  if $p = n \neq 0$, $H(c) = 1$ (maximum uncertainty)
  if $p = 0, n \neq 0$, $H(c) = 0$ (no uncertainty)

- Information Gain: Expected reduction in entropy from the attribute test on A:

$$Gain(c, A) = B\left(\frac{p}{p+n}\right) - H(c|A)$$

  - Expected remaining entropy after testing A:
  $$H(c|A) = E\left(B\left(\frac{p_i}{p_i + n_i}\right)\right) = \sum_{i=1}^{d} \frac{p_i + n_i}{p+n} B\left(\frac{p_i}{p_i + n_i}\right)$$
  - Continuous-valued attributes: Partition into a discrete set of intervals.
  - Attributes with many values: Use $GainRatio$:
  $$GainRatio(C, A) = \frac{Gain(C, A)}{SplitInformation(C, A)}$$
  $$SplitInformation(C, A) = -\sum_{i=1}^{d} \frac{|E_i|}{|E|} \log_2 \frac{|E_i|}{|E|}$$
  where E_i is the subsets of E divided by A.
  - Attributes with differing costs: Use
  $$\frac{Gain^2(C, A)}{Cost(A)}$$
  $$\frac{2^{Gain(C,A)} - 1}{(Cost(A) + 1)^w}$$
  where $w \in [0,1]$ denotes importance of cost.
  - Attributes with missing values:
    - Assign most common value
    - Assign most common value with same value of target concept
    - Assign probability p_i to each possible values
- Inductive Bias of Decision Tree Learning:
  (a) Shorter trees are preferred.
  (b) Trees that place high information gain attributes close to the root are preferred.
  - Occam's Razor: Prefer shortest/simplest hypothesis that fits the data.
    - Short hypothesis unlikely to be coincidence
    - Many ways to define small set of hypotheses
    - Can be obtained using different hypothesis representations

<u>2.4. Overfitting</u>

- Hypothesis $h \in H$ overfits the set D of training examples if and only if
$$\exists h' \in H \backslash \{h\} \text{ s.t.}$$
$$(error_D(h) < error_D(h')) \wedge (error_{D_X}(h) > error_{D_X}(h'))$$
where $error_D(h)$ and $error_{D_X}(h)$ denotes errors of h over D and set D_X of examples corresponding to instance space X.
- How to select best decision tree?
  - Measure performance over training data.
  - Measure performance over a separate validation data set.
  - Minimise $size(tree)$ & $size(misclassifications(tree))$
- Avoiding overfitting:
  - Stop growing decision tree when expanding a node is not statistically significant.
  - Allow decision tree to grow and overfit the data, then post-prune it.
- Pruning:
  - Reduced-Error Pruning

```
Partition data into training and validation sets
Do until further pruning is harmful:
1. Evaluate impact on validation set of pruning each possible node
2. Greedily remove the one that most improves validation set accuracy
```

  - Rule Post-Pruning

```
1. Convert learning decision tree to an equivalent set of rules
2. Prune each rule by removing any precondition that improves its estimated accuracy
3. Sort pruned rules by estimated accuracy into desired sequence for use when classifying unobserved input instances
```

3. Neural Network

<u>3.1. Perceptron Training Rule</u>

- Perceptron Unit:
$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{, if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1, & \text{otherwise} \end{cases}$$
$w_0 + w_1 x_1 + \cdots + w_n x_n$ can also be written as $\boldsymbol{w} \cdot \boldsymbol{x}$.
- Perception Training Rule: Initialise $\boldsymbol{w}$ randomly, apply perception training rule to every training example, and iterate through all training examples till $\boldsymbol{w}$ is consistent:
$$w_i \leftarrow w_i + \Delta w_i, \Delta w_i = \eta(t - o)x_i$$
for $i = 0,1,2, \dots, n$ where
  - $t = c(\boldsymbol{x})$ is the target output;
  - $o = o(\boldsymbol{x})$ is the perceptron output;
  - η is some small +ve constant called learning rate.

It is guaranteed to converge if training examples are linearly separable and η is sufficiently small.

<u>3.2. Linear Unit Training Rule with Gradient Descent</u>

- Loss function: $L_D(\boldsymbol{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$
- Training rule: Initialise $\boldsymbol{w}$ randomly, and then repeatedly updating it in the direction of steepest descent:
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\Delta w}, \ \ \boldsymbol{\Delta w} = -\eta \nabla L_D(\boldsymbol{w})$$
  - Gradient: $\nabla L_D(\boldsymbol{w}) = \left[\frac{\partial L_D}{\partial w_0}, \frac{\partial L_D}{\partial w_1}, \dots, \frac{\partial L_D}{\partial w_n}\right]$

```
1. Initialise each $w_i$ to some small random value
2. Until termination condition is met, do:
  2.1. Initialise each $\Delta w_i$ to zero
  2.2. For each $d \in D$, do:
    2.2.1. Input instance $\boldsymbol{x}_d$ to linear unit and compute $o$
    2.2.2. For each linear unit weight $w_i$, do
      $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_{id}$
  2.3. For each linear unit weight $w_i$, do
    $w_i \leftarrow w_i + \Delta w_i$
```

3.3. Stochastic Gradient Descent

- Batch Gradient Descent:

```
Do until satisfied:
1. Compute gradient $\nabla L_D(\boldsymbol{w})$
2. $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla L_D(\boldsymbol{w})$
   where $L_D(\boldsymbol{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$
```

- Stochastic Gradient Descent:

```
Do until satisfied:
For each training example $d \in D$
   1. Compute gradient $\nabla L_d(\boldsymbol{w})$
   2. $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla L_D(\boldsymbol{w})$
      where $L_D(\boldsymbol{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$
```

3.4. Backpropagation Algorithm

- Sigmoid Unit: $\sigma(net) = \frac{1}{1+e^{-net}}$
  - $\frac{d\sigma(net)}{dnet} = \sigma(net)\big(1 - \sigma(net)\big)$
- Backpropagation: Initialise $\boldsymbol{w}$ randomly, propagate input forward and then errors backward through the network for each training example.

```
Initialise all network weights to small random numbers.
Do until satisfied:
  For each training example $\langle \boldsymbol{x}, (t_k)_{k \in K}^{\mathrm{T}} \rangle$ do
    1. Input instance $\boldsymbol{x}$ to the network and compute output
       of every sigmoid unit in the hidden and output layers.
    2. For each output unit $k$, compute error $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
    3. For each hidden unit $h$, compute error $\delta_h \leftarrow o_h(1 - o_h)\sum_{k \in K} w_{hk}\delta_k$
    4. Update each weight $w_{hk} \leftarrow w_{hk} + \Delta w_{hk}$
       where $\Delta_{W_{hk}} = \eta \delta_k o_k$
    5. Update each weight $w_{ih} \leftarrow w_{ih} + \Delta w_{ih}$
       where $\Delta_{W_{ih}} = \eta \delta_h x_i$
```

3.5. Alternative Loss Functions

- Penalise large weights:
$$L_D(\boldsymbol{w}) = \frac{1}{2}\sum_{d \in D}\sum_{k \in K}(t_{kd} - o_{kd})^2 + \gamma \sum_{j,l} w_{j,l}^2$$
- Train on target values as well as slopes:
$$L_D(\boldsymbol{w}) = \frac{1}{2}\sum_{d \in D}\sum_{k \in K}\left[(t_{kd} - o_{kd})^2 + \mu \sum_{i=1}^{n}\left(\frac{\partial t_{kd}}{\partial x_{id}} - \frac{\partial o_{kd}}{\partial x_{id}}\right)^2\right]$$
- Tie together weights

4. Bayesian Inference

4.1. Bayes Theorem

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h)$: Prior belief of hypothesis h
- $P(D|h)$: Likelihood of data D given h
- $P(D) = \sum_{h \in H} P(D|h)P(h)$: Marginal likelihood of D
- $P(h|D)$: Posterior belief of h given D

4.2. Choosing Hypothesis

- Maximum *a posteriori* hypothesis:
$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D) = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$
- Maximum likelihood hypothesis: Used when $P(h) = P(h')$ for all $h, h' \in H$.
$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} P(D|h)$$
- Minimum description length:
$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h)$$
  - $L_C(x)$ is the description length of x under encoding C.

4.3. Classification

- Bayes-optimal classification:
$$\underset{t \in T}{\operatorname{argmax}} P(t|D) = \underset{t \in T}{\operatorname{argmax}} \sum_{h \in H} P(t|h)P(h|D)$$
  - Computationally costly if H is large
- Gibbs classifier: Sample a hypothesis h from posterior belief $P(h|D)$ and use h to classify new instance $\boldsymbol{x}$.
- Naïve Bayes classifier:
$$t_{NB} = \underset{t \in T}{\operatorname{argmax}} P(t) \prod_{i=1}^{n} P(x_i|t)$$
  - Naïve Bayes assumption:
$$P(x_1, x_2, \dots, x_n|t) = \prod_{i=1}^{n} P(x_i|t)$$

```
NAÏVE_BAYES_LEARN($D$)
  **for** each value of target output $t$ **do**
    $\hat{P}(t) \leftarrow$ estimate $P(t)$ using $D$
    **for** each value of attribute $x_i$ **do**
      $\hat{P}(x_i|t) \leftarrow$ estimate $P(x_i|t)$ using $D$

CLASSIFY_NEW_INSTANCE($\boldsymbol{x}$)

  $t_{NB} = \underset{t \in T}{\operatorname{argmax}} \hat{P}(t) \prod_{i=1}^{n} \hat{P}(x_i|t)$
```

  - Bayesian estimate: Used when none of the training instances with target value t has attribute x_i.

$$\hat{P}(x_i|t) = \frac{|D_{tx_i}| + mp}{|D_t| + m}$$

where $|D_{tx_i}|$ is the number of training examples with target output value t and attribute value x_i; $|D_t|$ is the number of training examples with target output value t; p is the prior estimate for $\hat{P}(x_i|t)$ and m is weight given to prior p.

4.4. Expectation Maximization

- EM Algorithm

Pick random initial $h = \langle \mu_1, \mu_2 \rangle$. Then, iterate

E step: Calculate the expected value $\mathbb{E}[z_{dm}]$ of each hidden/latent variable z_{dm}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

$$\mathbb{E}[z_{dm}] = \frac{p(x_d|\mu_m)}{\sum_{l=1}^{2} p(x_d|\mu_l)} = \frac{e^{-\frac{1}{2\sigma^2}(x_d - \mu_m)^2}}{\sum_{l=1}^{2} e^{-\frac{1}{2\sigma^2}(x_d - \mu_l)^2}}$$

M step: Calculate a new ML hypothesis $h' = \langle \mu_1', \mu_2' \rangle$, assuming the value taken on by each latent variable z_{dm} is its expected value $\mathbb{E}[z_{dm}]$ computed above. Replace h by $h' = \langle \mu_1', \mu_2' \rangle$.

$$\mu_m' = \frac{\sum_{d \in D} \mathbb{E}[z_{dm}] x_d}{\sum_{d \in D} \mathbb{E}[z_{dm}]}$$

- General EM Algorithm: Given
  - Observed data $\{\boldsymbol{x}_d\}_{d \in D}$
  - Unobserved data $\{\boldsymbol{z}_d\}_{d \in D}$ where $z_d = \langle z_{d1}, \dots, z_{dM} \rangle$
  - Parametrised probability distribution $p(D|h)$ where $D = \{d\}$ is the complete data where $d = \langle \boldsymbol{x}_d, \boldsymbol{z}_d \rangle$ and h comprises the parameters

  Determine ML hypothesis h' that locally maximises $\mathbb{E}[\ln p(D|h')]$.

Define function $Q(h'|h) = \mathbb{E}[\ln p(D|h')|h, \{\boldsymbol{x}_d\}_{d \in D'}]$ given current parameters h and observed data $\{\boldsymbol{x}_d\}_{d \in D}$ to estimate the latent variables $\{\boldsymbol{z}_d\}_{d \in D}$.

Pick random initial h. Then, iterate

E Step: Calculate $Q(h|h')$ using current hypothesis h and observed data $\{\boldsymbol{x}_d\}_{d \in D}$ to estimate the latent variables $\{\boldsymbol{z}_d\}_{d \in D}$.

$$Q(h'|h) \leftarrow \mathbb{E}[\ln p(D|h')|h, \{\boldsymbol{x}_d\}_{d \in D}]$$

M Step: Replace hypothesis h by the hypothesis h' that maximizes this Q function:

$$h \leftarrow \underset{h'}{\operatorname{argmax}} Q(h'|h)$$