CS3244 Machine Learning AY2021/2022 Sem 2 Summary Notes (Ng Xuan Jun)

Topic 1: Concept Learning

Trade-Off for Hypothesis Representation

Expressive Power
- **Benefit**: More hypothesis that can be expressed leading to a larger hypothesis space and it could give us a more accurate depiction of the target concept
- **Con**: More data require to find the correct function and more time to compute due to the larger hypothesis space

Smaller Hypothesis Space
- **Benefit**: Lesser data required to find the correct function and less time to compute due to the smaller hypothesis space
- **Con**: Lesser hypothesis that can be expressed. It will only be able to express a smaller hypothesis space and the target concept may not be contained inside

Number of Hypothesis

Note that x_i – Number of options for that input attribute i

Total Number of Input Instances:

$$|X| = x_1 \times x_2 \times \cdots \times x_n$$

Total Number of Syntactically Distinct Hypothesis:
- All possible number of combinations that we can have for the hypothesis inclusive of other values such as $(?, \emptyset)$. Note that for this case we have 2 more symbols so we add 2, it could be different

$$|Syn| = (x_1 + 2) \times (x_2 + 2) \times \cdots \times (x_n + 2)$$

Total Number of Semantically Distinct Hypothesis:
- All the possible hypothesis where they are all uniquely different and do not mean the same thing. We categorise those with at least 1 $\emptyset$ to be 1 case. Note that there could be cases whereby those without $\emptyset$ are semantically similar if they represent the same set of input instances

$$|Sem| = \underbrace{1}_{\text{Semantically Similar Cases (With at least 1 } \emptyset)} + \underbrace{[(x_1 + 1) \times (x_2 + 1) \times \cdots \times (x_n + 1)]}_{\text{Remaining Semantically distinct options (Add ? but cannot have } \emptyset)}$$

Difference between Satisfies and Consistent

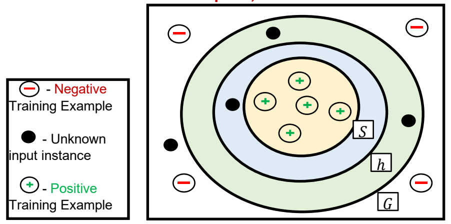
- **Satisfies** only cares about whether the hypothesis labels the training example as positive (**Does not tell us the correctness**)
- **Consistent** cares about whether the hypothesis correctly labels the training example as positive or negative

Version Space

General Boundary:
- There does not exist another consistent hypothesis that is more general than g. There isn't one that is a superset of g that is consistent.
- Good summary of **negative** examples

Specific Boundary:
- There does not exist another consistent hypothesis that is more specific than s. There isn't one that is a subset of s that is consistent.
- Good summary of **positive** examples

Instance Space, X

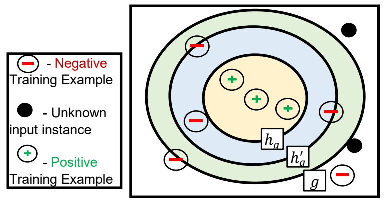


Maximally General
- There will not be another consistent hypothesis that is **more general** than the maximally general hypotheses
- This is the **largest set** before negative training examples satisfy the hypothesis which will make the hypothesis not consistent

Maximally Specific
- There will not be another consistent hypothesis that is **more specific** than the maximally specific hypotheses
- This is the **smallest set** before positive training examples does not satisfy the hypothesis which will make the hypothesis not consistent

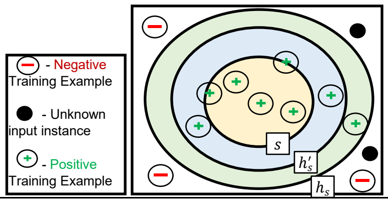
Minimally General
- The minimal generalisation, h_g is
  - Consistent hypothesis & $h_g >_g s$
  - There does not exist another consistent hypothesis (h_g') that will be **more specific** than the minimally general hypothesis

Instance Space X



Minimally Specific
- The minimal specialization, h_s is
  - Consistent hypothesis & $g >_g h_s$
  - There does not exist another consistent hypothesis (h_s') that will be **more general** than the minimally specific hypothesis

Instance Space X



Active Learner

- **For a given version space**: Query an optimal input instance that satisfies exactly half of the hypotheses in the version space
- Version space reduces by half with each training example no matter whether the training example turns out to be positive or negative
- **Optimally**: $\log_2(VS_{H,D})$ examples will be required to find target concept c by using the Active Learner

Unbiased Learner:

Suppose $H = \langle x_1, 1\rangle, \langle x_2, 1\rangle, \cdots, \langle x_n, 0\rangle$
- Each attribute value has 2 choices to be inside or not inside

$$|H| = 2^{|X|}$$

$|X|$ – Size of the input instance space (Total number of attribute values)

Very expressive but cannot classify new unobserved input instances

Comparing Learners with Different Inductive Biases

Stronger Inductive Bias → Increased Generalization Power → Can better use the knowledge from observed data to predict the unobserved data

Rote-Learner: NO Inductive Bias
Candidate-Elimination: Hard Inductive Bias
Find-S: Additional Requirement on top of Candidate Elimination

$\downarrow$ Stronger Inductive Bias

Topic 2: Decision Tree

Expressive Power

- **Complete** and **Expressive** (Can express any function of the input attribute)
- Target concept of a Boolean decision tree can be expressed in **disjunctive normal form** with each of the path being a conjunction attribute-value leading to a leaf. $(Path_1 \vee \cdots)$

Hypothesis Space:

Number of Distinct Binary Decision Trees with m Boolean Attributes
- Each of the attributes-values can have a value of True/False → 2^m **choices**
- For each of the combination of attribute-values, the target concept can either be True/False

$$Number\ of\ Distinct\ Trees: 2^{2^m}$$

DECISION-TREE-LEARNING

PLUARITY-VALUE (Take note of the edge cases)
- This is basically taking the **majority vote** of the positive and negative examples and if it is a tie then just random
- Reach a node where there are **no more examples**. This means that we still have attribute-values to choose but all the examples are for other values of the attribute. Return the **PLUARITY-VALUE(*parent_examples*)** where we just use the value from the parent node
- Reach we reach a node where we still have not concluded the classification but we **no longer have any more attributes left**, we just return the **PLUARITY-VALUE(*examples*).** Where this is just the examples of the current node

Information Theory

Entropy formula of r.v. C:

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

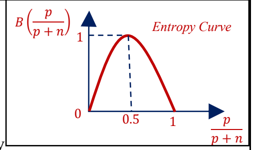
Entropy of Boolean r.v. that is true with probability q:

$$B(q) = -(q \log_2 q + (1-q)\log_2(1-q))$$

Entropy of target concept C with training set containing p +ve examples and n negative examples:

$$H(C) = B\left(\frac{p}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Entropy Curve:
Maximum Entropy, $H(C) = 1$ (**Maximum Uncertainty**)
Minimum Entropy, $H(C) = 0$ (**Minimum Uncertainty**)
- 0 when $p/(p+n)$ is 0 or 1
- 1 when $p/(p+n)$ is 0.5



- Anything in between we will have some uncertainty
- Entropy tells us **how equally the values are spread**, the closer to $1/2$, the more uncertain we are since it means that the values are spread equally and we won't know which is the majority and is the correct label

Information Gain
Suppose we divide the training set E into different subsets which corresponds to each of the attribute values of A. Each subset E_i has $\boldsymbol{p_i}$ **+ve and $\boldsymbol{n_i}$ -ve** examples

$$H(C|A) = \sum_{i=1}^{d} \underbrace{\frac{p_i + n_i}{p+n}}_{\text{Proportion that } E_i \text{ takes out of all examples}} \underbrace{B\left(\frac{p_i}{p_i + n_i}\right)}_{\text{Entropy of the subset } E_i}$$

Information Gain of target concept C from the attribute test on A

$$Gain(C, A) = \underbrace{B\left(\frac{p}{p+n}\right)}_{\text{Entropy } H(C) \text{ of this node before any split (Prior Entropy)}} - \underbrace{H(C|A)}_{\text{Expected remaining entropy after testing } A \text{ (Posterior Entropy)}}$$

Choose the attribute A with the **largest gain** (Largest Value)

Hypothesis Search Space

Number of distinct DTs with m Boolean attributes $\gg 2^{2^m}$ because
1) Order of the Boolean attributes is important
2) Choice of the attribute values will lead to another set of Boolean attributes that can be chosen
3) Smaller trees allowed

Use heuristics to make the search faster even though search space is larger

Hard vs Soft Bias

Bias: Any basis for choosing one generalization over another, other than strict consistency with the instances. Minimum set of assumptions to make the hypothesis provably correct

Hard Bias (Representational Bias): Specifies the representation of the hypothesis (Restriction to Disjunctive Normal Form (DNF) etc.), implement for the hypothesis and the set of primitive terms (allowable features, types, range of values), this defines the set of states.

- Considered correct if it defines a hypothesis space that contains the target concept
- **Strong Hard Bias** implies small hypothesis space (**More restricted**)
- **Weak Hard Bias** implies large hypothesis space (**Less restricted**)

Soft Bias (Procedural/Algorithmic Bias): Determines the order of traversal of the states in the space defined by a representational bias

- Preference for **simple hypotheses** (Occam's Razor)

Overfitting

Definition (In words): A hypothesis $h \in H$ overfits the set D of training examples iff there is another hypothesis h' that is not h where h has a lower training error but a higher test error as compared to h'

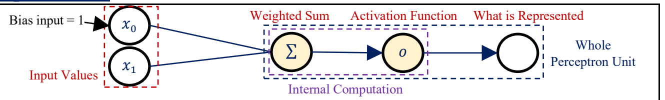
- **New erroneous/ noisy training example** may cause the values to be distorted and we may branch out more causing the results to be distorted
- **Limited data** may cause us to stop at a node without branching out more

Management of Issues

Issue: Overfitting	**Solution**: Reduced-Error Pruning • Partition into **training** and **validation** sets • Try to prune at each of the nodes (Making them a **leaf node** and using **PLUARITY-VALUE**) and check the validation set accuracy • Remove node that **increases the validation set accuracy** **Solution**: k-fold Validation (If we have **limited data**)
Issue: Continuous Valued Attributes	**Solution**: Partition the values into a **discrete set of intervals** for testing
Issue: Attributes with Many Values (Gain (E.g. Date) will select attribute with many values since there are many singletons and expected remaining entropy will be 0 → High information Gain)	**Solution**: Use GainRatio Instead $GainRatio(C,A) = \frac{Gain(C,A)}{SplitInformation(C,A)}$ $SplitInformation(C,A) = -\sum_{i=1}^{d} \frac{\|E_i\|}{\|E\|} \log_2 \frac{\|E_i\|}{\|E\|}$ • ***SplitInformation***: Measures how uniformly A splits training examples • $\|E_i\| \approx \|E\|$ (Not much split) → ***SplitInformation*** small because $\|E_i\|/\|E\|$ large → ***GainRatio*** large (But examples trickling down to one value is also bad, not much discerning) Use *GainRatio* only when ***Gain* > Pre-defined threshold**
Issue: Attributes with Differing Costs (We want to learn consistent DT with low expected cost)	**Solution**: Replace Gain with other functions $\frac{Gain^2(C,A)}{Cost(A)}$ $2^{Gain(C,A)} - \frac{1}{(Cost(A)+1)^{\omega}}$ Where $\omega \in [0,1]$ determined importance of cost • Note that it cannot be $Gain(C,A) - Cost(A)$ because they may not have the same magnitude. Can use $Gain(C,A) - \gamma Cost(A)$ instead to increase the weight of it
Issue: Missing Attribute Values	**Solution**: Use training examples anyway and sort through DT Choose 1 of the 3 methods 1) If node n tests the attribute A, assign the **most common value of A** at that node among other examples at node n 2) Assign **most common value** of A at that node among other examples that have the **same value of output** 3) Assign probability (**Observed frequency of training examples with each value**) p_i to each possible value of A (The number of examples can exist in fractions). Assign fraction p_i of example to those examples without values Classify new unobserved input instances with missing attributes in same manner

Topic 3: Neural Networks

Perceptron Unit



Weighted Sum:

$$\boxed{\underline{w} \cdot \underline{x} = \sum_{i=0}^{n} w_i x_i}$$

Activation Function:

$$\boxed{o(\underline{x}) = \begin{cases} 1, & \underline{w} \cdot \underline{x} > 0 \\ -1, & \underline{w} \cdot \underline{x} \le 0 \end{cases}}$$

Decision Surface of Perceptron

Type of Decision Surfaces:

- $n = 2$: Line, $n = 3$: Plane, $n = 4$: Hyperplane

Decision Surface Equation:

$$\boxed{\underline{w} \cdot \underline{x} = 0}$$

Suppose $n = 2$:

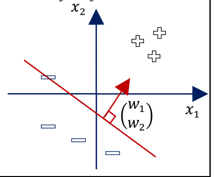
$$x_2 = \underbrace{-\left(\frac{w_1}{w_2}\right)}_{\text{Slope}} x_1 \underbrace{- \left(\frac{w_0}{w_2}\right)}_{\text{y-intercept}}$$

Note if the examples lie on the line, they will be classified as **negative** since $\underline{w} \cdot \underline{x} \le 0$

No bias weight ($x_0 = 1 \ or \ w_o = 0$):

- **Restricted** Hypothesis Space → Line needs to pass the origin; we **cannot translate** along x_2

Example of Decision Surface for $n = 2$



Orthogonal Vector:

- Orthogonal vector points to the **positive examples**

Signs of weights:

For w_1 & w_2:

- Look at the **direction of the orthogonal vector** and in which direction of **x_1 and x_2 it points to**
- For the example on the right, it points in the positive direction of both x_1 and x_2 so **w_1 & $w_2 > 0$** if it is the opposite **w_1 & $w_2 < 0$**. Both can also have different signs depending on the direction of orthogonal vector

For w_0:

- Look at the sign of w_2 and the y-intercept $(-w_0/w_2)$

Orthogonal Vector pointing **upwards & y-intercept +ve**	$w_0 < 0$
Orthogonal Vector pointing **upwards & y-intercept -ve**	$w_0 > 0$
Orthogonal Vector pointing **downwards & y-intercept +ve**	$w_0 > 0$
Orthogonal Vector pointing **downwards & y-intercept -ve**	$w_0 < 0$

Linearly Separable: If we can draw a decision surface and **separate all the positive and negative examples** using the decision surface
Linearly non-separable: If we cannot draw a decision surface and **cannot separate all the positive and negative examples** using the decision surface

Uses: Has **expressive power**. Can be used to represent $AND(x_1, x_2)$ etc
Cons: If the function is **linearly non-separable** then it **cannot be represented** (Deisgn Neural Nets to represent them)

Perceptron Training Rule

$$\text{Vector Form} \quad \underbrace{\boxed{w_i \leftarrow w_i + \Delta w_i,}}_{\text{Weight Update}} \ \underbrace{\boxed{\Delta w_i = -\eta(t - o)x_i}}_{\text{Weight Change}}$$

- $t = c(x)$ – **Target Concept**. $o = o(x)$ – **Perceptron output**
- η – small +ve constant, **Learning Rate**

It will work iteratively; the weight will only update if the perceptron misclassifies the training example, and it will either increase or decrease the current weight

Guaranteed to converge in a finite no. of iterations if:

- Training Examples are **linearly separable**
- Learning rate η is **sufficiently small** (Since we are not talking about convergence rate so if it is small, it will take a long time but will converge)

Possibilities for weight update:

Suppose $x_i > 0$

(a) **Positive but predicted negative**: $t > 0, o < 0$ → Misclassification occurs → Need to increase the $\underline{w} \cdot \underline{x}$ such that $\underline{w} \cdot \underline{x} > 0 \rightarrow o > 0$

$$\boxed{t - o = 2 \rightarrow \Delta w_i > 0 \rightarrow w_i \uparrow \rightarrow w_i x_i \uparrow \rightarrow \underline{w} \cdot \underline{x} \uparrow}$$

(b) **Negative but predicted positive**: $t < 0, o > 0$ → Misclassification occurs → Need to decrease the $\underline{w} \cdot \underline{x}$ such that $\underline{w} \cdot \underline{x} < 0 \rightarrow o < 0$

$$\boxed{t - o = -2 \rightarrow \Delta w_i < 0 \rightarrow w_i \downarrow \rightarrow w_i x_i \downarrow \rightarrow \underline{w} \cdot \underline{x} \downarrow}$$

Note that $o(\underline{x}) = \begin{cases} 1, & \underline{w} \cdot \underline{x} > 0 \\ -1, & \underline{w} \cdot \underline{x} \le 0 \end{cases}$ therefore when we misclassifes as positive, if we put $-o$, it will be the **same sign as *t*** and it will cause w_i to go in the direction of t

Gradient Descent

Simple Linear Unit:

$$\boxed{o = \underline{w} \cdot \underline{x}}$$

- Note that there is no activation function for this as compared to the perceptron unit

Loss Function (Sum of Squared Errors):

$$\boxed{L_D(\underline{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2}$$

D – Set of **training examples**, t_d – **Target Output**, o_d – Output of **linear unit**
d – Training example, $\underline{w}$ – **Hypothesis of the weights**

- Note that if we want to minimize the loss function (minimise the number of misclassifications), the constants does not matter
- $L_D = 0$ (**Fully agree**, linearly separable)
- $L_D > 0$ (**Disagrees to some extent**)

Gradient

$$\boxed{\nabla L_D(\underline{w}) = \left[\frac{\partial L_D}{\partial w_0}, \frac{\partial L_D}{\partial w_1}, \cdots, \frac{\partial L_D}{\partial w_n}\right]}$$

- Take the derivative of the loss function w.r.t each of the weights at each point
- Tells the **direction of the steepest increase**

Training Rule

$$\begin{array}{rll} & \text{Weight Update} & \text{Weight Change} \\ \text{Vector Form} & \underline{w} \leftarrow \underline{w} + \Delta \underline{w}, & \Delta \underline{w} = -\eta \nabla L_D(\underline{w}) \\ \text{Component Form} & w_i \leftarrow w_i + \Delta w_i, & \Delta w_i = -\eta \nabla L_D \dfrac{\partial L_D}{\partial w_0} \end{array}$$

- Note that there is a **-ve** sign for the weight change because we want to find the direction for the **steepest descent**
- η – Learning Rate cannot be **too small** if not the **convergence rate will be too slow** but if it is **too large**, it **may not converge**

Guaranteed to converge to hypothesis with min. squared error/loss:

- If learning rate η is **sufficiently small** (Since we are not talking about convergence rate so if it is small, it will take a long time but will converge)
- Even when training examples are **noisy** and/or **linearly-non separable** by H

Theorem 1.1 For any statement variables p, q and r, a tautology t and a contradiction c,

(a)	Commutativity	$p \wedge q \equiv q \wedge p$	$p \vee q \equiv q \vee p$
(b)	Associativity	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
(c)	Distributivity	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
(d)	Idempotence	$p \wedge p \equiv p$	$p \vee p \equiv p$
(e)	Absorption	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
(f)	De Morgan's Law	$\sim(p \wedge q) \equiv (\sim p) \vee (\sim q)$	$\sim(p \vee q) \equiv (\sim p) \wedge (\sim q)$
(g)	Identities	$p \wedge t \equiv p, \quad p \vee t \equiv t$	$p \vee c \equiv p, \quad p \wedge c \equiv c$
(h)	Negation	$p \vee \sim p \equiv t, \quad \sim(\sim p) \equiv p$	$p \wedge \sim p \equiv c, \quad \sim t \equiv c$

Contrapositive: $p \rightarrow q$ becomes $\sim q \rightarrow \sim p$
Implication Law: $p \rightarrow q \equiv \sim p \rightarrow q$