Topic 1: Concept Learning

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 (?, ∅). 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 ∅ to be 1 case. Note that there could be cases whereby those without ∅ are semantically similar if they represent the same set of input instances

$$|Sem| = 1 + [(x_1 + 1) \times (x_2 + 1) \times \cdots \times (x_n + 1)]$$

Semantically Similar Cases (With at least 1 ∅) | Remaining Semantically distinct options (Add ? but cannot have ∅)

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

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

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

PLURALITY-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 **PLURALITY-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 **PLURALITY-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:

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

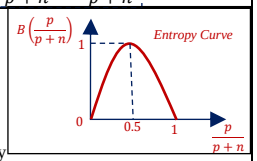
Entropy of target concept C with training set containing p +ve examples and n negative examples:

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

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 p_i **+ve and** n_i **-ve** examples

$$H(C|A) = \sum_{i=1}^{d}\frac{p_i+n_i}{p+n}B\left(\frac{p_i}{p_i+n_i}\right)$$

Proportion that E_i takes out of all examples | Entropy of the subset E_i

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

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

Entropy $H(C)$ of this node before any split (Prior Entropy) | Expected remaining entropy after testing A (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

Management of Issues

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 $GainRatio(C,A) = \frac{Gain(C,A)}{SplitInformation(C,A)}$ $SplitInformation(C,A) = -\sum_{i=1}^{d}\frac{\lvert E_i\rvert}{\lvert E\rvert}\log_2\frac{\lvert E_i\rvert}{\lvert E\rvert}$ • ***SplitInformation***: Measures how uniformly A splits training examples • $\lvert E_i\rvert \approx \lvert E\rvert$ (Not much split) → ***SplitInformation*** small because $\lvert E_i\rvert/\lvert E\rvert$ 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 $\frac{Gain^2(C,A)}{Cost(A)}$ $\frac{2^{Gain(C,A)}-1}{(Cost(A)+1)^{\omega}}$ 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

Topic 3: Neural Networks

Decision Surface of Perceptron

Type of Decision Surfaces:

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

Decision Surface Equation:

Suppose $n = 2$:

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

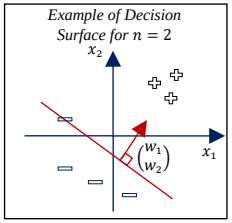
Slope: $-\left(\frac{w_1}{w_2}\right)$ y-intercept: $-\left(\frac{w_0}{w_2}\right)$

$$x_2 = -\left(\frac{w_1}{w_2}\right)x_1 - \left(\frac{w_0}{w_2}\right)$$

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



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

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 $\underline{w}\cdot\underline{x} > 0 \rightarrow o > 0$

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

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

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

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:

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

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

Loss Function (Sum of Squared Errors):

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

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

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

- 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

	Weight Update	Weight Change
Vector Form	$\underline{w} \leftarrow \underline{w} + \Delta \underline{w},$	$\Delta \underline{w} = -\eta \nabla L_D(\underline{w})$
Component Form	$w_i \leftarrow w_i + \Delta w_i,$	$\Delta w_i = -\eta \nabla L_D \frac{\partial L_D}{\partial w_0}$

- 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**

Stochastic (Incremental) Gradient Descent

Batch Gradient Descent (GD)	Stochastic Gradient Descent (SGD)
1. Compute true gradient $\nabla L_D(\underline{w})$ 2. $\underline{w} \leftarrow \underline{w} - \eta \nabla L_D(\underline{w})$ where $L_D(\underline{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$	1. Compute stochastic gradient $L_d(\underline{w})$ 2. $\underline{w} \leftarrow \underline{w} - \eta \nabla L_d(\underline{w})$ where $L_d(\underline{w}) = \frac{1}{2}(t_d - o_d)^2$
Computes over the **entire training examples**	**Sample** from the set of training examples with replication. (There is some level of randomness here), each example has a probability of $\frac{1}{\lvert D\rvert}$

SGD approximation of GD:

$$E[\nabla L_d(\underline{w})] = \nabla L_D(\underline{w})$$

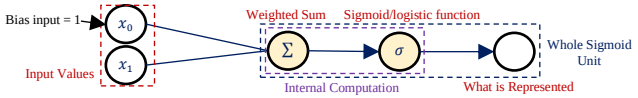
- It is an **unbiased estimator** for the true gradient, but we need **to remove the ½**
- SGD can approximate batch GD arbitrarily closely if learning rate η is sufficiently small (Sufficient condition for convergence)

General Case (Data mini-batch):

$$\min_{\underline{w}} E_{\underline{x}}[L(\underline{x}, \underline{w})]$$
$$\underline{w} \leftarrow \underline{w} - \eta \nabla L(\underline{x}, \underline{w})$$

Objective function (differentiable wrt model parameters $\underline{w}$) can be decomposed into a sum of terms, each depending on a subset of training examples

Sigmoid Units

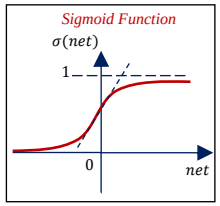


Weighted Sum:

$$net = \sum_{i=0}^{n} w_i x_i$$

Sigmoid/Logistic Function (Output):

$$o = \sigma(net) = \frac{1}{1 + e^{-net}}$$



- Monotonically increasing
- $\sigma(net)$ value is between **0 and 1**
- $net \geq 0 \rightarrow \sigma(net) > \frac{1}{2}$ & $net < 0 \rightarrow \sigma(net) < \frac{1}{2}$
- When net is near 0, it will be a **linear behaviour**

Useful Property:

$$\frac{d\sigma(net)}{dnet} = \sigma(net)(1 - \sigma(net))$$

Issue with linear units:

- Only able to represent a linear function rather than a non-linear function
- The step function is not differentiable

Pros of using sigmoid units:

- Output is non-linear and is a differentiable function so it can be used to represent non-liner functions and we can do gradient descent on it

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: $p \rightarrow q \equiv \sim p \rightarrow q$

Topic 4: Bayesian Inference

Bayes Theorem:

$$\text{Posterior Belief } P(h|D) = \frac{P(D|h)P(h)}{P(D)} = \frac{P(D|h)P(h)}{\sum_{h\in H} P(D|h)P(h)}$$

(Unnormalized Posterior Belief / Marginalised Model Evidence)

$P(h)$ - Probability of h being the right hypothesis, this is more of a preference

$P(D|h)$ - Likelihood of data D given h. This tells us how likely we see such a data given that we have a hypothesis h and h is correct

$P(D) = \sum_{h\in H} P(D|h)P(h)$ - Marginal Likelihood/ Evidence of D. This is also the model evidence. This is how likely that D will appear

$P(h|D)$ – Posterior belief of h given D. This is how much confidence that we have in h after seeing the data D

MAP and ML Hypothesis

MAP Hypothesis: Hypothesis with the highest posterior belief

$$h_{MAP} = \underset{h\in H}{\operatorname{argmax}} P(h|D) = \underset{h\in H}{\operatorname{argmax}} P(D|h)P(h)$$

Note that we can just remove the marginalising term since it is independent of h

ML Hypothesis: Hypothesis with the highest likelihood

$$h_{ML} = \underset{h\in H}{\operatorname{argmax}} P(D|h)$$

Probability Formulas

Product Rule for Probabilities:

$$P(A_1, A_2) = P(A_1|A_2)P(A_2)$$

Product Rule for Conditional Probabilities:

$$P(A_1, A_2|A_3) = P(A_1|A_2, A_3)P(A_2|A_3)$$

Chain Rule for Probabilities:

$$P(A_1, \cdots, A_n) = \prod_{i=1}^{n} P(A_i|A_1, \cdots, A_{i-1})$$

If the events are **conditionally independent**:

$$P(A_1 \cdots, A_n) = \prod_{i=1}^{n} P(A_i)$$

Inclusion and Exclusion Principle:

$$P(\cup_{i=1}^{n} A_i) = \sum_{1\leq i\leq n} P(A_i) - \sum_{1\leq i<j\leq n} P(A_i, A_j) + \cdots + (-1)^{n-1} P(A_1, \cdots, A_n)$$

If the events are **conditionally independent**:

$$P(\cup_{i=1}^{n} A_i) = \sum_{1\leq i\leq n} P(A_i)$$

Marginalization: If $A_1, \cdots, A_n$ are mutually exclusive, where $\sum_{i=1}^{n} P(A_i) = 1$

$$P(B) = \sum_{i=1}^{n} P(B, A_i) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

Tricks that are used:

1. Changing a maximization problem to a **log-maximisation problem**. Note that if we have an **increasing function**, (i.e. log in this case), we can convert it over to a log function instead and try to maximise it and still get the same result
2. **Remove constants** that are independent of the variable that we are optimising
3. $\underset{h\in H}{\operatorname{argmax}} F(h) = \underset{h\in H}{\operatorname{argmin}} -F(h)$. We can change a maximisation problem to a minimisation problem by taking the negative of it
4. Assuming training examples t_d for $d \in D$ are conditionally independent given h, and input instances x_d for $d \in D$ are fixed. Therefore, given h and the input instances, the output should be independent of each other

$$P(D|h) = \prod_{d\in D} P(t_d|h, x_d)$$

5. Assuming training examples $\langle x_d, t_d\rangle$ for $d \in D$ are conditionally independent given h since h is going to express f (target concept) well

$$P(D|h) = \prod_{d\in D} P(x_d|t_d, h)$$

Bayes-Optimal Classifier

Bayes-optimal classification: Note that we are only concerned with what is the classification and note that it could be different from the MAP. After looking at the posterior distributions, we find what is the most probable classification

$$\underset{t\in T}{\operatorname{argmax}} P(t|D) = \underset{t\in T}{\operatorname{argmax}} P(t|h)P(h|D)$$

For this, just need to find all the $P(t|h)P(h|D)$ and find the classification that has the highest value, and it will be the bayes-optimal classification. Like **majority rule**

Naïve Bayes Classifier

Difference from Bayes-Optimal Classifier: Looks at class labels and input attributes instead of data and hypothesis

Data Analysis: Looking at how much data we will need. Need to look at the frequency of each of the combinations

Conditional Independence Case:

$$(\#Att\ Values\ per\ Att)^{(\#Att)} \times (\#\ Values\ of\ Labels)$$

Non-conditional Independence Case:

$$(\#Att\ Values\ per\ Att)^{\#Att} \times (\#\ Values\ of\ Labels)$$

Space Analysis: Looking at how many of different probabilities that we need to keep track of

Note that for space analysis, since we are keeping track of probabilities, we can take note of using complements

Conditional Independence Case:

$$(\#Att\ Values\ per\ Att - 1)^{(\#Att)} \times (\#\ Values\ of\ Labels)$$

Once we have found the first few attribute values for a given label, the remaining value for the attribute can be found by taking the complement

Non-conditional Independence Case:

$$((\#Att\ Values\ per\ Att)^{\#Att} - 1) \times (\#\ Values\ of\ Labels)$$

Once we have found the first few combinations, the last combination can be taken as the complement of the first few combinations

Expectation Maximization (EM)

Estimating *M* Means:

Used when we want to estimate latent features given some input attributes

2 Parts:

1. Randomly initialise the means, $h = \langle \mu_1, \mu_2 \rangle$
2. Calculate the expected value of each hidden latent feature/ variable, z_{dm} assuming that the current hypothesis is true. $E[z_{dm}]$
3. Calculate a new ML hypothesis $h' = \langle \mu_1', \mu_2' \rangle$, we kind of a weighted sample mean where each of expectations are the weights, and we can find the means from there

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

Tells us how much x_d contributes to the m^{th} Gaussian mean using the weights of the expectation that the m^{th} Gaussian generated x_d

General EM Algo:

Determine ML hypothesis, h' that locally maximises $E[\ln P(D|h')]$ Given current parameters h and observed data $\{x_d\}_{d\in D}$

1. Randomly initialise the means, $h = \langle \mu_1, \mu_2, \cdots, \mu_M \rangle$
2. Calculate $Q(h|h')$ using the current hypothesis h and observed data $\{x_d\}_{d\in D}$ to estimate the latent variables $\{z_d\}_{d\in D}$. Technically we are trying to maximise the log likelihood function

$$Q(h'|h) \leftarrow E[\ln(P(D|h')|h, \{x_d\}_{d\in D}]$$

3. Replace the current hypothesis h with a new hypothesis h' such that h' maximises Q function $h \leftarrow \underset{h'}{\operatorname{argmax}}\, Q(h'|h)$

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

Tells us how much x_d contributes to the m^{th} Gaussian mean using the weights of the expectation that the m^{th} Gaussian generated x_d