SEMESTER 1, 2021

MACHINE LEARNING

COMP30027

KEVIN Y. YANG
DOCUMENT MAINTAINER

1 – ML Basics

1.1 Data Terminology

1. Instances (exemplars) – ***rows*** of a data set
2. Attributes (features) – ***columns*** of a data set
3. Concepts (Class labels) – things we are aimed to learn/predict from a dataset
4. Common Variables: ***N*** training instance, ***C*** classes, ***X*** attributes

1.2 Attribute Type

1. ***Nominal quantities (categorical/discrete)*** have no relation and natural order between examples. E.g., sunny, hot, rainy, Boolean values
    a. Algorithms assume Nominal data type: Naïve Bayes, Decision Tree
2. ***Ordinal quantities (categorical/discrete***) have an implied ordering between labels. E.g., (cold, mild, and hot) have an ordering (cold < mild < hot)
3. ***Continuous quantities (numerical)*** are real-valued attributes with a define zero point and have no explicit upper bound.
    a. Algorithm that assumes numeric attributes: SVM, Perceptron, NeuralNet

1.3 Methods

1. ***Supervised methods*** have prior knowledge to a closed set of classes (or having presence of labelled data), and used to predict new value
2. ***Unsupervised methods*** will dynamically discover classes without any knowledge of class labels and will train itself to categorize instances mathematically.

2 – Probability, Statistics & Naïve Bayes

2.1.1 Probability Basics

1. Joint probability: $P(x|y) = \frac{P(x \cap y)}{P(y)}$ = Probability of x occurring, given y

2.1.2 Probability Rule

1. Sum rule: $P(x) = \sum_y P(x \cap y)$
2. Product rule: $P(x, y) = P(x \cap y) = P(x|y)P(y)$
3. Bayes' rule: $P(y|x) = \frac{P(x|y)P(y)}{P(x)}$
4. Chain rule: $P(x_1 \cap \ldots \cap x_n) = P(x_1)P(x_2|x_1)P(x_3|x_2 \cap x_1)P(x_n|\bigcap_{i=1}^{n-1} x_i)$

2.1.3 Prior & Posterior Terminology

1. Independence: $P(x|y) = P(x),\ P(x, y) = P(x)P(y)$
2. Conditional Independence: x and y are independent conditioned on z
    a. $P(x, y|z) = P(x|z)P(y|z)$

2.2 Binomial Distribution

1. Bernoulli trial – events that produce binary output – where binomial distribution is a series of Bernoulli trial
2. Probability of an event with prob. b occurring m out of n times:

$$B(m; n, p) = \binom{m}{n} p^m (1-p)^{n-m} = \frac{n!}{m!\,(m-n)!} p^m (1-p)^{n-m}$$

2.3 Multinomial Distribution

A multinomial distribution results from a series of independent trial with binary outcomes e.g. win, lose, or draw. Probability of events $X_1, \dots, X_n$ with probability of $P_1, \dots P_n$ occurring exactly $x_1, \dots, x_n$ times, respectively:

$$P(X_1 = x_1, \dots, X_n = x_n) = \left(\sum_{i=1}^{n} x_i\right)! \prod_{i=1}^{n} \frac{p_i^{x_i}}{x_i!}$$

2.4 Normal Distribution

A normal distribution (or Gaussian distribution) is often used to represent a noisy continuous variable. The probability from a variable with mean (expected value) μ and the standard deviation σ.

$$N(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\right]$$

2.5 Entropy (Information Theory)

Entropy is a ***measure of unpredictability*** on the information required to predict an event. The entropy – measured in ***bits*** (Binary bits) – of a discrete random variable X is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2[P(x_i)]$$

where $H(X) \in [0, \log_2(n)]$

1. A **high** entropy value means X is ***unpredictable***. Each outcome gives one bit of information.
2. A **low** entropy value means X is more ***predictable***. We don't need to learn anything once we see the outcome.

2.6 Naïve Bayes Implementation

Hyperparameters:

1. ***Smoothing Method***
2. (Optionally) ***The choice of distribution*** used to model the features

Parameters:

1. **The prior and posterior probabilities**
2. **Size = $O(|C| + |C||FV|)$,**
    a. C=set of classes, F=set of features, assuming one-vs-all SVM

Interpretation:

Based on the most *positively weighted* features associated with a given instance.

The naïve Bayes assumes that ***all probabilities are independent***. This mean that for every class j:

$$P(x_1, x_2, \dots, x_n) \approx \prod_{i=1} P(x_i|c_j)$$

This is called the ***conditional independence assumption*** and makes Naïve Bayes a tractable method.

Naïve Bayes works by calculating a set of prior probabilities of classes $P(Class = c)$, and posterior probabilities of attribute given a class j $P(attribute = x_i | class = c_j)$ Then for every test instance, it will calculate the posterior and assign it the largest corresponding prior probability.

$$Prediction\ \hat{c} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \cdot \prod_{i=1} P(x_i|c_j)$$

Strength	***Weakness***
Simple to build, fast	*Inaccurate when there are many missing $P(x_i\|c_j)$ values*
Computations scale well to high-dimensional datasets	*Conditional independence assumption becomes problematic for complex systems*
Explainable – generally easy to understand why the model makes certain decisions	

2.7 Probabilistic Smoothing

As Naïve Bayes predicts using a product of probabilities, $P(x_i|c_j) = 0$ means that final value will be zero. Therefore, to avoid unseen event become "impossible", Smoothing is applied to treat unobserved events but "unlikely".

2.7.1 Epsilon

Assume no event is impossible and having probability > 0, assign some value ϵ to any $P(x_i|c_j) = 0$. Since $\epsilon \to 0$, $\epsilon \ll \frac{1}{n}$, and with a condition $1 + \epsilon \approx 1$.

2.7.2 Laplace Smoothing

The most used alternative is add-k, Laplace smoothing essentially gives any **unseen events a count of $\boldsymbol{\alpha = 1}$** (or $\alpha \in (0, 1]$ unless specified). Then all counts are increased to ensure monotonicity.

Let $N = No.of\ attributes$, then for every class j, and attribute value x_i:

$$P(x_i|c_j) = \frac{(\alpha = 1) + freq(x_i, c_j)}{N + \alpha \times set(x_i).length}$$

2.8 Missing Value

1. If a value is missing in a ***training instance***, then it's possible to simply have it <u>not contribute to the attribute-value count</u> estimates for that attribute
2. If a value is missing in a **test instance**, then you may just **ignore the attribute** for the purposes of classification

3 – Discrete & Continuous Data, Model Evaluation

3.1 Common Datatype Conversions

1. ***Nominal*** Data → ***Numerical*** Data
   a. Converting Category names to numbers, e.g.

Nominal Data	`["bus", "tram", "car", "bike", "walk"]`
Numerical	`[0, 1, 2, 3]`

   b. One-hot Encoding – assign Boolean attribute to vectors (array) of zeros. *Refer to 3.2 for nominal distance metrics* e.g.,

$$\text{"bus"} = \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}, \text{tram"} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}, \text{"car"} = \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}, \text{bike"} = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}, \text{"walk"} = \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}$$

2. Numeric Data → Nominal Data
   a. **Discretization** – translating continuous numeric to discrete nominal attribute. *Refer to 3.3 for Discretization Methods*

3.2 Distance Metrics

3.2.1 Euclidean Distance

If A and B differ on N attributes, $dist_E(A, B) = \sqrt{N}$

$$dist_E(A, B) = \sqrt{\sum_i (a_i - b_i)^2}$$

3.2.2 Hamming Distance

If A and B differ on N attributes, $dist_H(A, B) = N$

$$dist_H(A, B) = \sum_i \begin{cases} 0, & a_i == b_i \\ 1, & else \end{cases}$$

For *Manhattan Distance* and *Cosine Similarity*, visit *section 5.4.3 – 5.4.4*

3.3 Discretization Methods

Discretization is the translation of continuous attributes onto nominal attributes (also known as *binning*). This process is generally performed in two steps:

1. Decide how many (nominal) values/intervals to map the feature onto (*equal-width, equal-frequency, k-Means*)
2. Map the features such that they are in their respective ranges (Also called ***binning***)

$$\{(x_0, x_1), (x_1, x_2], \dots, (x_{n-1}, x_n)\}$$

3.3.1 Unsupervised Discretization

There are 3 main methods:

1. ***Equal-width:*** find the min & max of the dataset, partition in n (bin) of width → $(\frac{max-min}{n})$ intervals. Allocated each value into its respective interval. ***Disadvantage:*** suffer from datasets with a tight group of values with outlier that overfit the model. Must choose *n*.

2. ***Equal-frequency:*** sort values and find the breakpoints that produces n bins with (approx.) equal number of items. For new instances added, use the median as a dividing point. ***Disadvantages:*** sensitive to outlier & sample bias. Arbitrary group boundaries & must choose *n*.
3. **K-Means:** apply an unsupervised clustering. *Disadvantages:* Complicated, equal-width discretization if data doesn't manifest natural clusters. Sensitive to outlier & must choose *n*
    a. Select *k* points (no. of bins) at random to act as seed clusters.
    b. Assign each instance to the cluster with the nearest centroid.
    c. Compute the centroids of each cluster by taking the mean.
    d. Repeat until assignment of instances of the clusters converge to a stable state.

3.3.1 Supervised Discretization (Naïve)

The idea is to group values into class-contiguous interval.

1. Sort values & identify breakpoints in class membership.

64	65	68	70	71	72	72	75	…
Yes	No	Yes	Yes	Yes	No	Yes	Yes	…

   The group then become:

$$(64), (65), (68,69,70), (71,72), (72, 75)$$

2. If there is a tie, re-position any breakpoints where there is *no change* in numerical value. It should be set between the neighbouring values

$$(64), (65), (68,69,70), (71,72,72), (75)$$

Although it is very simple to implement, it usually ***creates too many categories which leads to overfitting***. Solutions to this is **group-value approach**, where each category must have at least n instances of a unique class.

3.4 Naïve Bayes with Continuous Data

The Naïve Bayes assumes that **all probabilities are independent**, this means that for every class j:

$$\Pr(x_1, x_2, \dots, x_n | c_j) \approx \prod_{i=1} \Pr(x_i | c_j)$$

The likelihood of **feature x_i** in **class c_j** can be computed differently for different data types:

1. If x_i is a ***nominal attribute*** with multiple levels, count no. of times each level occurs in class c_j
2. If x_i is a ***numeric attribute***, compute its probability distribution.

For predicting labels, refer to ***section 2.6***

3.4.1 Gaussian Naïve Bayes

Recall that a dataset exhibiting normal distribution has a ***symmetrical graphical representation with mean μ as the pivot***, and ***which the area under the curve (the integral) of the distribution sums to 1***. Then, the probability density function $\sim N(\mu, \sigma^2)$ can be displayed as follows:

$$f_X(x) = \phi_\sigma(x - \mu) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Where:

Mean of N samples of an attribute X: $\mu_X = \frac{1}{N}\sum_{i=1}^{N} \frac{x_i}{N}$

Standard deviation of N samples of an attribute X: $\sigma_X = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N-1}}$

The Gaussian is **optimal** when approximating events that showcased ***binomial, Poisson, chi-square, and Student's t-distribution***. Dataset can be e.g., heights, weight, and reaction time…

Examples, given $\mu_{no} = 25, \sigma_{no} = 6.4, \mu_{yes} = 19.5, \sigma_{yes} = 1.0$:

$$\Pr(temp = 22.8 \mid rain = no) = P(rain = no) \times \phi_{\sigma_{\text{no}}}(22.8 - \mu_{\text{no}})$$
$$= (0.5) \times \frac{1}{6.4\sqrt{2\pi}} \exp -\frac{1}{2}\left(\frac{22.8 - 25.0}{6.4}\right)^2 = 0.092$$
$$Pr(temp = 22.8 | rain = yes) = Pr(rain = yes)\phi_{\sigma_{yes}}(22.8 - \mu_{yes})$$
$$= (0.5) \times \frac{1}{1.0\sqrt{2\pi}} \exp -\frac{1}{2}\left(\frac{22.8 - 19.5}{1.0}\right)^2 = 0.0006$$

3.4.2 Kernel Density Estimation (KDE)

As for data are not assumed with specific probability distribution pattern (e.g., Gaussian). KDE can implement along with naïve bayes using the equations as follows:

$$f_X(x) = \frac{1}{n}\sum_{i=1}^{N} \phi_\sigma(x - x_i)$$

where $\phi \sim N(\mu = 0, \sigma^2)$, $x_i = elevations\ of\ N\ i's\ in\ the\ training\ set.$

KDE has one parameter $\sigma =$ *"Kernel Bandwidth"*

e.g., Computing the probability of the test instances, given $\sigma = 5$:

$$P(x = 15|yes) = P(yes) \times \frac{1}{N}\sum_{n=1}^{N} \phi_\sigma(x_{test} - x_n)$$
$$= (0.266)\left(\frac{1}{266}\right)\sum_{n=1}^{266} \phi_5(15 - x_n) = \mathbf{0.00029}$$
$$P(x = 15|no) = P(no)\frac{1}{M}\sum_{m=1}^{M} \phi_\sigma(x_{test} - x_m)$$
$$= (0.734)\left(\frac{1}{734}\right)\sum_{n=1}^{734} \phi_5(15 - x_m) = \mathbf{0.00497}$$

KDE has **advantages** in **modelling arbitrary probability distributions**, and **no assumptions required about the shape** of the distribution (e.g., Gaussian).

However, **KDE** has **disadvantages in choosing a kernel bandwidth**, and require many parameters to represent the PDF, where its **probability computation at new points is exhaustive.**

3.4.3 Other Types of Naïve Bayes

1. ***Multivariate*** – Attributes are nominal and can take any of a fixed no. of values.
2. ***Binomial (Bernoulli)*** – Attributes are binary (special case of multivariate)
3. ***Multinomial*** – attributes are natural no. corresponding to frequency $P(a_k = m|c_j) \approx \frac{P(a_k = 1|c_j)^m}{m!}$

3.5 Model Strategies

3.6.1 Accuracy

While the goal of a good classifier is to assign the correct class labels to instances never seen before, its identification of a good classifier should

1. ***Train*** on a subset of data,
2. ***Test*** on the rest subset which has not processed during training and
3. ***Evaluate*** performance on the test data.

The basic evaluation metric ***Accuracy***, given as:

$$Accuracy = \frac{No.\,of\ correctly\ predicted\ labels}{Total\ number\ of\ test\ instances}$$

3.6.2 Holdout

1. Each instance is **randomly assigned** as either t*raining instance* or *test instance*.
2. The dataset is **partitioned** with no overlap between data sets.
3. ML Model is ***built based on the training instances*** and evaluate the trained model with test instances.
4. Common train-test splits are: 50-50, 80-20, 90-10 (Leave-one out: $(N-1)-1$)

Advantages	Disadvantages
1. *Simple to work with and to implement* 2. *High reproducibility*	1. *The split-ratio affects the estimate of classifiers behaviours* 2. *Lots of training instances with few test instances leaves the model to be accurate, but the test instances may not be representative of future test instances.*

3.6.3 Repeated Random Sub-sampling

Works like hold out, but over several iterations.

1. New train-test sets are randomly chosen each iteration
2. The size of train-test split is fixed across all iteration
3. A new model and evaluation are done every iteration

Advantages	Disadvantages
1. *Average holdout method tends produce more reliable results.*	1. *Slower than the holdout method (by a constant factor of no. of iterations)* 2. *A wrong choice of the train-test split ratio can lead to highly misleading results.*

3.6.4 (*k*-fold) Cross-Validation

The usual ***preferred method of evaluation.*** The data set is progressively split into a no. of m partition, where:

1. One partition is used as test instances.
2. The other $m - 1$ partitions are used as training instances
3. The *evaluation metrics* is aggregated across m partitions by taking the average.

This is much **better than holdout/ repeated random sub-sampling** since:

1. Every instance is a test instance (for some partition).
2. Take roughly the **same time** as repeated random sub-sampling.
3. Can be shown to minimise the bias and variance of our estimates of the classifier's performance.

Choosing the values of m for Cross-validation

1. $\downarrow m$: More instances per partition, but ***more variance*** in performance estimates
2. $\uparrow m$: Fewer instances per partition, but ***less variance*** at the sacrifice of time complexity
3. A usual default value is to use $m = 10$ or $m = 5$, *which mimics 90-10 or 80-20 holdout strategy.*

3.6.5 Inductive Learning Hypothesis & Stratification

Random sampling may produce different proportion. To prevent this scenario, **vertical sampling/stratification** can be introduced to produce training & testing data that both have ***the same class distribution*** as the dataset.

Stratification is a type of **inductive learning hypothesis**. Any hypothesis found to approximate the target function over a large training data set will also approximate the target function well over any unseen test examples.

However, machine learning suffers from **inductive bias** – ***meaning assumptions must be made about the data to build a model and make predictions***.

Stratification assumes class distribution of unseen instances will **share the same distribution** of seen instances. **This is NOT true for any Machine Learning problems.**

3.6 Model Evaluation Methods.

3.6.1 Classification Evaluations – Terminology

1. ***True Positive (TP)*** are instances where we predicted a label to be +, and the actual label is +.
2. **False Positive (FP)** are instances where we predicted a label to be +, but the actual label was −.
3. **False Negative (FN)** are instances where we predicted a label to be −, but the actual label was +.
4. **True Negative (TN)** are instances where we predicted a label to be −, but the actual label was −.

In an ideal world, we only want **True Negative** and **True Positives.**

3.6.2 Classification Evaluations – Accuracy Methods

NOT to be mixed up with Accuracy, Classification Accuracy is the proportion of instances for which we have correctly predicted the label, given as

$$Classification\ Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

3.6.3 Classification Evaluations – Error Rate

The Error Rate & Its reduction of a classifier is given as:

$$Error\ Rate = 1 - Classification\ Accuracy = \frac{FP + FN}{TP + FP + FN + TN}$$

$$Error\ rate\ reduction = \Delta Error\ Rate = \frac{ER_0 - ER}{ER_0}$$

3.6.4 F-Scores: Precision & Recall

1. **Precision –** How often are we correct when we predict a label. This is given as:

$$Precision = \frac{TP}{TP + FP}$$

2. **Recall/Sensitivity –** What proportion of the predictions have we correctly predicted. This is given as:

$$Recall = Sensitivity = \frac{TP}{TP + FN}$$

3. **Specificity –** What proportions of true negatives cases that the model was able to detect.

$$Specificity = \frac{TN}{TN + FP}$$

High precision gives low recall, and high recall gives low precision. Since we want both precision and recall being high, we can evaluate this using an **F-score.**

$$F_\beta = \frac{(1 + \beta^2)2 \times Precision \times Recall}{\beta^2 \times Precision + Recall}$$

where β is chosen ∃ *recall* is considered β times more important than precision. OR

$$F_1 = \frac{Precision \times Recall}{Precision + Recall}$$

3.6.5 Multiclass Evaluations

Over Multi-class datasets, you must calculate the Precision/Recall/F-Score per class and must be average across c classes. (A **Confusion Matrix** is often sketched to showcase patterns of errors in a multiclass classification task.)

z

1. **Macro-Averaging** takes the mean of all Precision/Recall:

$$Precision_M = \frac{\sum_{i=1}^{c} Precision(i)}{c}$$

$$Recall_M = \frac{\sum_{i=1}^{c} Recall(i)}{c}$$

2. **Micro-Averaging** combines all instances into a single pooled Precision/Recall:

$$Precision_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FP_i}$$

$$Recall_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FN_i}$$

3. **Weighted Averaging** compute P, R per class and take mean, weighted by proportion of instance in that class.

$$Precision_W = \sum_{i=1}^{c} \left(\frac{n_i}{N}\right) Precision(i)$$

$$Recall_W = \sum_{i=1}^{c} \left(\frac{n_i}{N}\right) Recall(i)$$

3.7 Baseline and Benchmarks

Baseline	Benchmark
Simple Naïve method that we would expect any ML methods to be better off. *e.g., Random Guessing, human (amateur performance)*	*Establish rival technique to which we are comparing against our methods.* *e.g., current best-performing algorithm on a leader board*

3.7.1 Random Baseline

1. Random assign a class to each test instances.
2. Randomly assign a class c_k to each test instance and weight the class assignment according to Pr (c_k). (This assumes we know class prior probabilities).

3.7.2 Zero-R/Majority Class

Essentially classifies all test instances as the most common class in the training set. Complexity is $O(1)$

3.7.3 One-R/Decision Stump

Creates a single rule (One-rule) for each attribute in the training data, then selects the rule with the **smallest error rate** as its "one rule". For decision tree specific One-R, refer to *section 4.1*

3.7.4 Other Baselines

1. Regression – always guess the mean value.
2. Object detection always guess the middle of the image.

4 – Decision Trees & K-Nearest Neighbour (KNN)

5.1. Information Gain

Information Gain is the expected reduction in entropy caused by knowing the value of an attribute. Comparing:

1. The entropy before splitting the tree using the attribute's value (*refer to* ***section 2.5 for Entropy*** $H(x)$)
2. The weighted average of the entropy over the children after the split *(refer to* ***section 4.1.1 for Mean Information)***

For a better tree that is **predictable**, we want the **entropy as small as possible.** *(Occam's Razer, generalizability)*

For example**,** selecting attribute R_A(with values $x_1, \dots, x_m$) best splits the instances at a given root node R according to **information gain:**

$$\underset{a \in A}{\operatorname{argmin}}\, IG(R_a|R) = \underset{a \in A}{\operatorname{argmin}} \left[H(R) - \sum_{j=1}^{m} P(x_j) H(x_j) \right]$$

Information Gain also tend to prefer ***highly branching attributes***:

1. A subset of instances is more likely to be homogeneous (all of a single class) if there are only a few instances.
2. Attributes with many values will have fewer instances at each child node.

These factors may result in **over-fitting** or **fragmentation**.

4.1.1. Mean Information

Mean information is a ***weighted average of the entropy*** over the children after the split of a decision tree stump with *m* attribute can be calculated as:

$$Mean\ Info(x_1, x_2, \dots, x_m) = \sum_{i=1}^{m} \Pr(x_i)\, H(x_i)$$

4.1.2. Split Info

The **Split Info (SI)** or **Intrinsic value** is the *entropy* of a given split (evenness of the distribution of instances to attribute values)

4.1.3. Gain Ratio

The Gain Ratio reduces the bias for Information Gain toward highly branching attributes by normalizing relative to the Split Information. The Gain Ratio can be calculated as:

$$\begin{aligned} GR(R_A|R) &= \frac{IG(R_A|R)}{SI(R_A|R)} \\ &= \frac{IG(R_A|R)}{H(R_A)} \\ &= \frac{H(R) - \sum_{j=1}^{m} P(x_j) H(x_j)}{-\sum_{j=1}^{m} P(x_j) \log_2 P(x_j)} \end{aligned}$$

It is discouraged to select attributes with many uniformly distributed values.

5.2. Decision Tree

Hyperparameters:

1. Function used for ***Attribute Selection***
2. The convergence ***(Stopping) Criterion***

Parameters:

1. The decision tree itself
2. **Size =** $\boldsymbol{O(|FV|)}$**,** where FV is the set of feature-value pairs

Interpretation:

Based directly through the decision tree.

Decision Tress are created in a recursive divide-and-conquer fashion. We want the **smallest tree** which minimizes the errors. In decision tree, a choice of function will be used for ***attribute selection*** and ***convergence criterion***

4.3.1. One-R for Decision Tree

(Refer to Section 3.7.3 for One-R definitions). The Pseudo-code for One-R DT is as follow

Function *oneR:*

for each *attribute* $x_i \in \{x_1, x_2, \dots, x_n\}$*:*

 # assume unique label class $y_1, y_2, \dots, y_n \in \boldsymbol{y}$

 # assume unique attributes $a_1, a_2, \dots, a_n \in x_i$

$y_i = y_j \ni \max[\Pr(a_k)] \ \forall j, k = 1, \dots, n$

 # calculate error rate of each attribute iteration

$e_i = error\ rate \ni \sum_i \frac{\{y_i \neq y_j\}}{y}$

predict *label* ***y*** *based on* ***attribute*** $x = x_i \ni \min(e_i)$

Complexity $O(1)$

4.3.2. ID3 Decision Tree

ID3-DT is a highly regarded basic supervised learning algorithms that construct decision tree of arbitrary depth. The algorithm can capture complex feature interact in a ***recursive divide-and-conquered*** fashion.

ID3 is an inductive learning algorithm that performs a simple-to-complex, hill-climbing search through hypothesis space (all possible decision) that fits the training examples. There is no backtracking in ID3, and it optimized for the shortest hypothesis that fits the data.

Function **ID3***(root):*

if all instances have the same class label

then stop

else

Select an attribute to use in partition *Root* node instances

Create a branch for each attribute value and partition the *Root* node according to each value.

for each $LEAF_i$:

ID3$(LEAF_i)$

ID3 is ***advantageous*** to its fast training and test time ***complexities*** $\boldsymbol{O(D)}$***. However***, it is also susceptible to the effects of irrelevant attributes.

4.3.3. ID3 DT – Stopping Criteria

ID3 DT Algorithm is define in a way such that:

1. The **Info Gain/ Gain ratio** allows us to select the seemingly better attribute at a given node.
    a. An Info Gain ≅ 0 means that there is no improvement and can often be pruned.
2. It's an approx. indication of how much absolute improvement we expect from partitioning the data according to the values of a given attribute.

4.3.4. Alternative DTs

1. ***Oblivious Decision Tree:*** Requires the same attribute at every node in a layer.
2. ***Random Tree:*** Uses samples of the possible attributes at any given node. This helps to reduce the effects of irrelevant attributes a basis of a DT variant *(Refer to section Random Forest Tree)*
3. ***C4.5:*** Extension of ID3 that handles both continuous and discrete values and prune branches to avoid overfitting.

5.3. Instance-based Learning

4.3.1. Instance-based Learning: Glossary

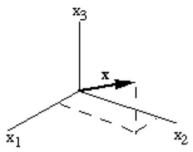
Instances all consists of a class labels and are described by n attribute-value pairs. As supposed to supervised learning algorithms (acting as a functional mapper) learning from label examples and using instances as input, ***instance-based learning required labelled examples to undergo memorization (stored in memory) and learn directly by examples.*** Recall ML systems consists of:

1. **Instances** – the individual, independent example of a concept
2. **Attributes** – Measurement of an instance aspects
3. **Labels/Class** – Targets which ML algorithms aim to learn.
4. **Feature** – distinct vector features that represent that instances

4.3.2. Feature Vectors

Feature vector is an n-dimensional vector of features (Nominal, ordinal, or numeric) that represents an instance. A vector locates an instance as a point in an n-dimensional space and the angle of the vector in that space is determined by relative weight of each term.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$



5.4. Instance Comparison

5.4.1. Similarity & Dissimilarity

Attribute Type	**Similarity** $\in [0,1]$ 1. Quantity measure of how similar two vectors are. 2. **Higher** measure for **closer** similarity	**Dissimilarity** 1. Quantity measure of how different two vectors are. 2. **Lower** measure for closer similarity 3. **Min** dissimilarity is often 0, with a varying upper limit
Nominal	$f(x) = \begin{cases} 0, & if\ p = q \\ 1, & if\ p \neq q \end{cases}$	$f(x) = \begin{cases} 1, & if\ p = q \\ 0, & if\ p \neq q \end{cases}$
Ordinal	$d = \frac{\lvert p - q \rvert}{n - 1}$ (values mapped to integers 0 to $n - 1$ is the no. of values)	$s = 1 - \frac{\lvert p - q \rvert}{n - 1}$
Interval/Ratio	$d = \lvert p - q \rvert$	$s = -d, s = \frac{1}{1 + d}$ Or $s = 1 - \frac{d - \min(d)}{\max(d) - \min(d)}$

5.4.2. Distance Measures (continued)

(For Euclidean & Hamming distance, visit *section 3.2*) A distance measure is a function that takes two parameters (points in a space). It exhibits the following properties

1. No negative distance $d(x, y) \geq 0$
2. Distance is zero from a point to itself $d(x, y) = 0\ iff\ x = y$
3. Distance is symmetric $d(x, y) = d(y, x)$
4. Triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$

5.4.3. Distance Measures – Manhattan Distance

Given two items A and B, and their feature vectors ***a*** and ***b***, we can calculate their similarity via their distance d based on the absolute differences of their Cartesian coordinates

$$d_M(A, B) = \sum_{i=1}^{n} |a_i - b_i|$$

5.4.4. Distance Measures – Cosine Distance

Given two items A and B, and their feature vectors ***a*** and ***b,*** we can calculate their similarity via the cosine of the angle θ between vector ***a*** and ***b***.

$$\cos(A, B) = \frac{\vec{\boldsymbol{a}} \cdot \vec{\boldsymbol{b}}}{|\vec{\boldsymbol{a}}||\vec{\boldsymbol{b}}|} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$$

5.5. k-nearest neighbour (*k*NN)

Hyperparameter:

1. Neighbourhood size *k*
2. Distance/Similarity metrics
3. Feature Weighting/Selection

Parameters:

None, computation occurs during runtime (and doesn't abstract away from training instances)

Interpretations:

Relative to the training instances that give rise to a given classification, and their distribution in the feature space.

The nearest neighbour is defined as the closest object from object of interest $d(x, y) = \min[d(x, z)|z \in Y]$, using a specified distancing metrics.

1. **Given class assignment** of existing data point, classify a new point and consider **class membership of the *closest data point***.
2. Out of the **closest *k* points**, and then categorize the instances based on the **majority of classes** which the k-nearest neighbours are possessing.
3. **the induced decision boundary** will be formulated using nearest neighbor

A ***weighted variant*** *(see Section 4.6)* of k-NN will classify the test instances according to the weighted accumulative class of the *k* nearest training instances, where the weights are based on similarity of the input to each *k* neighbour.

4.5.1. Choosing k (the neighbouring size)

When choosing k, its importance to decide base on the density of data points and perform trial & error over the training data. Generally:

1. **Smaller values of *k*** tend to lead to lower performance due to overfitting (noise)
2. **Larger values of *k*** tend to drive the classifier towards to Zero-R performance

K is generally set to an **odd value** to avoid breaking ties, which requires random tie breaking, taking classes with highest prior probability, or see if $(k + 1)$-th instances breaks the tie.

4.5.2. *k*-NN Analysis

A typical implementation involves a *brute-force* computation of distances between a test instance, and to every other training instances.

For *N* training instance and *D* dimensions, we end up with $O(DN)$ performance. This means for **smaller dataset; the performance is fast** but comes **infeasible as $N \uparrow$.**

This is because

1. The model built by NB or a DT is generally much smaller than the number of training instances in the dataset.
2. The model built by *k*-NN is also the dataset itself, all calculation is done during runtime.

It also plausible to try odd values *k* since the distribution could be split equality, $O(DN + k)$

Strength	Weakness
• Simple, instance-based, model free • Can produce flexible decision boundaries • Incremental (can add extra data during runtime)	• Requires a useful distance function • Arbitrary *k*-value • Calculation is done during runtime (lecturer likes to call it lazy??) • Prone to noise & high dimensionality.

5.6. Weight Strategies

4.6.1. Majority Classes

Given each neighbour weighting and classifies according to the majority class of set of neighbours.

4.6.2. Inverse Distance

Weight the vote of each instance by taking the inverse and adding some ϵ

$$w_j = \frac{1}{d_j + \epsilon}$$

4.6.3. Inverse Linear Distance

Weight the vote of each instances by taking the inverse and its "ranking" into account.

$$w_j = \frac{d_f - d_j}{d_f - d_n}$$

where,

$$f = furthest\ neighbour$$
$$n = nearest\ neighbour$$

5 – Support Vector Machine (SVM) & Model Interpretability

5.1. Nearest Prototype Classification

Hyperparameters:

1. ***Distance/Similarity Metric*** – used to calculate the Prototype and distance to each prototype in classification
2. ***Feature Weighting/Selection.***

Parameters:

1. Prototype for each class
2. Size $= O(|C||F|)$, where C is the *set of classes*; F is *set of features*

Interpretation:

Relative to the geometric distribution of the prototypes and the distance to each prototype for a given test instance.

To find the Nearest Prototype, calculate the centroid (mean of all numeric training instances) of each class: $P_j = [a_1^*, a_2^*, \dots, a_n^*]$; $a_j^* = \frac{1}{C_j}\sum_{i \in C_j} \boldsymbol{x}_i \ , \forall j$

To classify, calculate a distance using a distance metric between the test instance and prototype and assign the label of the closest prototype as prediction $\hat{c} = \underset{j \in P}{\operatorname{argmin}} \, || \, a_j - \boldsymbol{x}||$

Complexity: $O(CD)$

5.2. SVM (Support Vector Machine)

Hyperparameters:

1. ***Penalty term*** $\frac{C}{\epsilon}$ ***for soft-margin SVMs***
2. ***Feature-value scaling***
3. ***The kernel function used***

Parameters:

1. Vector of feature weight + bias term
2. Size $= O(|C||F|)$, assuming a one-vs-rest SVM, where *C* is the *set of classes*; *F* is *set of features*

Interpretation:

The absolute value of the weight associated with each non-zero feature in each instance provides an indication its relative importance in classification.
(Optimization algorithm is outside of scope lmao)

for (x_i, y_i) in $\{(x_1, y_1), (x_2, y_2) \dots, (x_n, y_n)\}$ **do**
optimize $\boldsymbol{w}$ and b s.t. $y_i(w^T x_i + b) - 1 \geq 0$
done

5.2.1. Linear Classifier

SVM consists of a Hyperplane which results in a binary classification through a linear separation in the forms of

$$f(x) = \boldsymbol{w^T} \cdot \boldsymbol{x} + b$$

where *points* $\boldsymbol{x} = [x_1, x_2, \dots, x_D]$, *weight (normal)* $\boldsymbol{w} = [w_1, w_2, \dots, w_D]$, with intercept *b*

5.2.2. SVM Intuitions & Optimization

Assume linearly separable cases, given training set $\{(x_1, y_1), (x_2, y_2) \dots, (x_n, y_n)\}$, where $x_i \in R^D$ and $y_i \in \{-1, 1\}$, the binary regions can be represented as

$$\max \frac{2}{||w||} \exists \begin{cases} \boldsymbol{w}^T x_i + b \geq 1, & y_i = 1 \\ \boldsymbol{w}^T x_i + b \geq -1, & y_i = -1 \end{cases} = [y_i(w^T x_i + b) - 1 \geq 0]$$

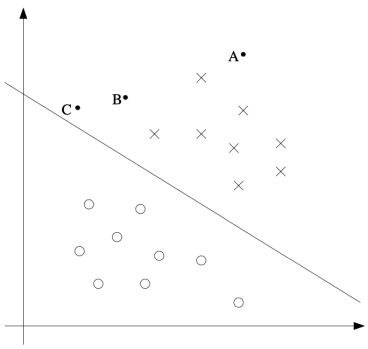
Which is equivalent to,

$$\min_w \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_{i=1}^{N} \xi_i$$

w.r.t. $y_i(w^T x_i + b) - 1 \geq 0, \forall i \in \{1,\ 2, \dots, N\}$

Where hyperplane that act as decision boundary is: $\boldsymbol{w}^T \boldsymbol{x} + b = 0$, and using ***Lagrange Multiplier***

$$w_d = \sum_i^N \alpha_i y_i x_{id}\,, \qquad b = \frac{1}{N_{sv}} \sum_{j \in N_{sv}} \frac{1 - y_j w^T x_j}{y_j}$$



Notice point A is far from decision boundary, where it is quite confident in defining class. Conversely, point C is very close to decision boundary, where a small change to the decision boundary would alter the class prediction. Therefore, geometric margins are needed.

5.2.3. Maximum Margins

How are we rate the different decision boundaries to work out which is the most stable under perturbation of inputs?

1. for a given training set, ***find the decision boundary that allows us to make all correct and confident predictions*** on the training examples
2. ***Some methods find a separating hyperplane***, but not an optimal one. SVM do converge to an optimal solution.
3. ***Maximize the distance*** between *hyperplane & difficult fringe points* which are close to the decision boundary.

5.2.4. Soft Margins

A possible large margin solution is usually better even though the constraints are being violated. This permits some point to be on the *wrong* side of the hyperplane. Using a soft margin is optimal, when the data is hypothesized to be linearly separable.

Slack variable $\boldsymbol{\xi} = [\xi_1, \xi_2, \dots, \xi_n]$, is used to allow some variables to be on the "wrong" side of the hyperplane at some cost. Ultimately, resulting objective function is:

$$\min_{w} \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C\sum_{i=1}^{N}\xi_i$$

w.r.t. $y_i(w^T x_i + b) + \xi_i - 1 \geq 0,\ \xi \geq 0, \forall i \in \{1,\ 2, \dots, N\}$

where C is a hyperparameter representing trade-off between maximizing the margin & minimizing the error.

5.2.5. Kernel Function for Non-linear SVM

To obtain a non-linear classifier, we can transform our data by applying a feature mapping function Φ such that a linear classifier can be used on the transformed feature vectors. The mapper function is called a **Kernel Functions.**

If the dataset is deemed to be non-linearly separable, a kernel function is better than a soft margin since they will end up producing a spurious (non-related) margin.

For example, using a polynomial Kernel with degrees of 2, $K_{P2}(\boldsymbol{x_i}, \boldsymbol{x_j})$, (where both $\boldsymbol{x_i}, \boldsymbol{x_j}$ are vectors), computations are as follow:

$$K_{P2}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \left(\theta + \boldsymbol{x_i^T x_j}\right)^2 = \phi(\boldsymbol{x_i})^T \phi(\boldsymbol{x_j})$$

1. dot products between the vectors
2. additional of θ
3. exponentiation with polynomial degree

in which feature mapping are cheaper to compute. Kernel functions can also be:

- **Linear:** $K_L(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i^T x_j}$
- **Polynomial:** $K_{Pd}(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\theta + \boldsymbol{x_i^T x_j})^d$
- **Radial Basis (Function) Model:** $K_{RBF}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp\left[-\frac{\left||x_i - x_j|\right|^2}{2\sigma^2}\right]$

resulting,

$$f(t) = \begin{cases} \sum_i^N a_i y_i K(\boldsymbol{x_i}, \boldsymbol{t}) + b > 0, & class = 1 \\ else\,, & class = 0 \end{cases}$$

5.2.6. Multi-class SVM

Since SVM's are inherently binary classifiers, most common approaches to extending to multiple classes include:

1. ***One-vs-all:*** One classifier separates one class from the rest of the classes; chooses one class which classifies test data points with the greatest margin.
2. ***One-vs-one:*** One classifier per pair of classes, choose the classes, choose the class selected by most classifier.

5.2.7. SVM Analysis

Geometric classifiers assume all attributes are equally important and calculate some similarity/ distance metric. E.g., if the distance metric between useful attributes is small but the distance metric between non-similar attributes is large, they'll outweigh the classification and deem them *non-similar*

SVMs solve this issue by finding the optimal parameters (***weights***, ***w***) for each attribute so that the basis of predicting is more meaningful. Overall,

1. SVM are a high-accuracy margin classifier.
2. Learning a model means finding the best separating hyperplane
3. Classification is built on projection of a point onto hyperplane normal.
4. If the dataset is non-linear, then a kernel function maybe used to transform it into a linear dataset.
5. SVM have several parameters that need to be optimized and may end up being slow

Complexity: $O(C^2D + C^2)$

5.3. Interpreting Model

Model Interpretability is the mean by which we can interpret the basis of a given model classifying an instance the way it does. Such analysis consists of understanding the error and presence of parameter & hyperparameter

5.3.1. Error Analysis

Error analysis is the manual analysis which studies the misclassification of instances for a given model

1. **Identifying** different *classes* of error that the system makes (in relation to predicted vs. actual labels)
2. **Hypothesizing** as to what has caused the different errors and testing those hypotheses against the actual data.
3. **Quantifying** whether (for different classes) it's a question of data quantity/ sparsity, or something more fundamental than that.
4. **Feeding** those hypotheses back into feature/model engineering to see if the model can be improved.

5.3.2. Parameters & Hyperparameters

Hyperparameter of a model is a set of parameters which can be adjusted to define the bias / constraints in the learning process.

Parameter of a classifier is the result of a given set of hyperparameters applied onto particular training dataset, and then is used to classify the test instances.

(The Hyperparameters & Parameters of respective model can be referenced from their corresponding model sections)

5.4. Data Visualization

5.3.3. Data Visualization

Visually detect any anomalies, check the distribution, or check the decision boundary of the data.

5.3.4. Dimensionality Reduction

Any **dimension reduction method** will NOT be able to faithfully reproduce the original data into two- or three-dimensional reduced version.

Feature selection is one form of dimension reduction, but ideally, we want to be able to reduce the dimensions in a way that captures the fullest feature set.

5.3.5. Principal Component Analysis (PCA)

PCA will reduce the dimensions of a dataset consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the dataset. This is achieved by transforming to a new set of variables ***"The principal Components (PCs)"*** which are uncorrelated & ordered so that the first few retain most the variation present in all of the original variables. PCA is generally performed using an eigenvalue solver.

6 – Linear Regression & Logistic Regression

6.1. Regressions

Regression is often used as model if the ***data are continuous attributes,*** and we want ***continuous class/label as an output***. An example utilizing such model would be *to predict house prices using sizes, distance proximity to public transport, etc.*

6.1.1. Linear Regression -- Intuitions

In the case of linear regression, we want to establish the model where it captures the relationships between.

- An **outcome/response/dependent variable or label $\boldsymbol{y}$**
- **Predictors** $X = [x_1, x_2, \dots, x_D]^T$ – where it's also called independent variable, explanatory variable, or feature

Thereby postulating the linear relationship as follow:

$$\begin{aligned}\boldsymbol{y} = f_X(x) &= \beta_0 + \beta_1 x_1 + \cdots + \beta_D x_D \\ &= \beta_0 + \sum_{k=1}^{D} \beta_k x_k \\ &= \vec{\boldsymbol{\beta}} \cdot \vec{\boldsymbol{x}}\end{aligned}$$

Where $\vec{\boldsymbol{\beta}}$ is the weight/gradient matrix such at each element is corresponded with identical position in $\vec{\boldsymbol{x}}$ matrix

6.1.2. Linear regression – Learning Algorithms

Given $\boldsymbol{x}, \boldsymbol{y} = \{(x_i, y_i): i \in [1, N]\}$, the objective is to formulate learning algorithms that *minimizes the sum of square* of predicted $\hat{y}$ and actual y using ***least squares method***, such that:

$$\begin{aligned}\widehat{\boldsymbol{\beta}} &= arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} MSE = arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 \\ &= arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \beta \cdot x_i)^2\end{aligned}$$

To find the fitting value w.r.t $\widehat{\boldsymbol{\beta}}$, compute its partial derivatives for N instances and D attributes. such that,

$$\frac{\partial}{\partial \beta_k} = -\frac{2}{N} \sum_{i=1}^{N} x_{ik}(y_i - \hat{y})$$

note that $\frac{\partial}{\partial \beta_0} = -\frac{2}{N} \sum_{i=1}^{N} (y_i - \hat{y})$

6.2. Parameter Estimation

6.1.1. Optimization

Learning algorithms can be also identified as an optimization problem, where:

$$\widehat{\boldsymbol{\theta}} = arg\min_{\theta \in \Theta} [Error(\theta; L, F(T)]$$

Where L is the learning algorithms and $F(T)$ is the feature representations.
There are multiple ways to compute optimal solution, summarized as:

1. **Analytical Solutions** – *exact computation* in closed forms, for example
    a. Solving a system of equations $\frac{\partial}{\partial \vec{\boldsymbol{\theta}}} [Error] = 0$
    b. **[!!IMPORANT FOR EXAM]** For OLS (Ordinary Least Squares): $\widehat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}$
    c. **CONS:** Derivatives can be undefined/non-calculable.
2. **Exhaustive Solutions** – *error-centric* iterative computations
    a. $For\ each\ \theta \in k_i, i \in \{1,2, \dots, N\}$ calculate $Error(\theta; L, F(T))$
    b. For M parameters, each taking N unique values, it requires N^M train-evaluate cycles.
    c. PROS: works for methods with small No. of hyperparameters e.g., KNN
        i. Similarity Measures: Euclidean, Manhattan, Cosine.

ii. Voting Strategy: majority, ID, ILD, …
iii. No. of neighbours: 1, 2, 3, …

3. **Grid Search** – *Batch-processing* iterative computations.
   for each θ ∈ R:
   $k := no.of\ sub - samples$
   $boundary[] :=$ [minRange, maxRange]
   $subsamples := sample_i, i \in each\ \{\frac{boundary[]}{k}\}$
   for each $subsamples$:
   Compute $Error(\theta; L, F(T))$ for each sample
   a. **PROS:** closer to optimal estimates **CONS:** run-time complexity.
4. **Iterative Approximation** – *discrete iterations* for optimal Error approximations
   $\theta^0 := random(Z), minErr := \infty, \theta_{optimal} = rand(Z)$
   for $i \in \{0, 1, 2, \dots, n\}$:
   $err := Error\left(\theta^i \middle| L, F(T)\right.$
   if $err < optErr$:
   $\theta_{optimal} = \theta^i$
   $minErr = err$
   return $\theta_{optimal}$

6.1.2. Gradient-Descent

Utilizing the techniques in the previous sections, the **iterative approach for optimal gradient** reduce error in stepwise manner, and is demonstrated as follow for $i \in \{1,2,..,n\}$

$$\theta^{i+1} := \theta^i - \alpha \nabla \text{Error}\left(\theta^i \middle| L, F(t)\right)$$

alternatively,

$$\theta_k^{i+1} := \theta_k^i - \alpha \frac{\partial}{\partial \theta_k^i} \text{Error}\left(\theta^i \middle| L, F(t)\right)$$

Where $\boldsymbol{\alpha}$ **is the learning rate** that if $\boldsymbol{\alpha} \uparrow$ ***the algorithm will get slow***, and if $\boldsymbol{\alpha} \downarrow$***, you might miss the global minimum.***

In the context of ***linear regression***, the GD is:

$$\beta_k^{i+1} := \beta_k^i - \alpha \frac{\partial}{\partial \beta_k^i} \text{Error}(\beta^i)$$
$$= \beta_k^i - \frac{2\alpha}{N} \sum_{j=1}^{N} x_{jk}(y_j - \widehat{y_j^i})$$

6.1.3. Application & Evaluations

While GD computes the $\boldsymbol{\beta}$ that minimize the variances between realized label y and prediction $\hat{y}$ (through its dot products with x), it's also a viable method for category although some data transformation maybe required. e.g., Mapper function that translate nominal attributes through binarization/one-hot-encodings, thereby making use of linear regressions.

Moreover, transformation can be done (elementwise) to matrix $\boldsymbol{x}$**, where:**

$$x \to x^i, for\ i \in \{1, 2, \dots, n\}$$

or,

$$x \to \log(x)$$

6.3. Evaluation Metrics – Continuous Data

There are also variations to compute the absolute/relative difference between prediction and actual values, thereby substituting MSE in the computation of $\widehat{\boldsymbol{\beta}}$:

1. **Mean Squared Error (MSE):** $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
2. **Root Mean Squared Error (RMSE):** $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$
3. **Root Relative Square Error (RRSE):** relative to baseline $\bar{y} = \frac{1}{N}\sum y_i$

$$\sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

4. **Pearson Correlation:**

$$r = \frac{S_{\hat{Y}Y}}{\sqrt{S_{\hat{Y}Y}}}$$

where,

$$S_{\hat{Y}Y} = \frac{\sum_i(\hat{y} - \bar{\hat{y}})(y_i - \bar{y})}{N-1}$$

$$S_{\hat{Y}} = \frac{\sum_i(\hat{y} - \bar{\hat{y}})^2}{N-1},\ S_y = \frac{\sum_i(\hat{y} - \bar{y})^2}{N-1}$$

6.4. Logistic Regression

Logistic regression consists of mapping continuous values onto discrete labels.

6.4.1. LR Intuitions

The model $P(c|\boldsymbol{x})$ is a type of probabilistic classifications for binary datasets, by approximating a numeric membership of a class with a logistic function defined as:

$$\begin{aligned} h_\beta(c|\boldsymbol{x}) &= \Pr\left(c_j \middle| x_1, x_2, \dots x_D; \beta\right) \\ &= logistic(X\beta) \\ &= \frac{1}{1 + e^{-X\beta}} \end{aligned}$$

where

- β = are the likelihood parameters that we want to maximize via *GD*
- X = design matrix of the model

6.4.2. Parameter Estimations

We choose a β such that,

1. Positive $X\beta$ means the class is Y
2. Negative $X\beta$ means the class is N
3. Having $X\beta \approx 0$ means that it is uncertain

Thereby, estimating β in the forms of objective function:

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \prod_{i=1}^{N} \left[h_\beta(x_i)\right]^{y_i} \left[1 - h_\beta(x_i)\right]^{1-y_i}$$

and in terms of ***maximum log-likelihood***, which is a ***concave function***:

$$\log P(Y|\boldsymbol{X},\boldsymbol{\beta}) = \sum_{i=1}^{N} y_i \log h_\beta(\boldsymbol{x_i}) + (1-y_i)\log\left[1-h_\beta(\boldsymbol{x_i})\right]$$

The Pseudocode is as follows

For i in $\{x_1, x_2, \dots, x_N\}\{$

update $\beta_k^{iter+1} = \beta_i^{iter} - \alpha \frac{\partial}{\partial \beta_k^{iter}} \left[\log P\left(Y\middle|X, \beta^{iter}\right)\right]$

$\}$

6.4.3. Multinomial Logistic Regression

With the probability distribution for each instance sums to 1, the pseudo-code is below:

Pseudo-codes:

Pivot class $\boldsymbol{P}(y = pivot|\boldsymbol{x},\boldsymbol{\beta}) = \frac{1}{1+\sum_{c=1}^{|C|-1}\exp(\beta^c \cdot x)}$

for c_j **in** $|C|-1$

$$P\left(y = c_j\middle|\boldsymbol{x},\boldsymbol{\beta}\right) = \frac{\exp(\beta^{c_j}\cdot x)}{\mathbf{1}+\sum_{c=1}^{|C|-1}\boldsymbol{exp}(\boldsymbol{\beta^c}\cdot\boldsymbol{x})}$$

$$\widehat{pred} = \underset{c_j}{\mathbf{argmax}}\, P\left(y = c_j\middle|\boldsymbol{x},\boldsymbol{\beta}\right)$$

6.4.4. Analysis of Logistic Regression

1. Vast improvements on its counterpart Naïve Bayes
2. Suited to frequency-based feature (problems such as Neuro-linguistic Problems)
3. Slow to train & required feature scaling
4. Required large N to be effective

7 – Classifier Combination & Feature Selection

7.1. Classifier Combination / Ensemble Learning

Classifier combination constructs a set of ***base classifiers*** from a given set of training data and aggregates the outputs into a single *meta-classifier*.

1. The combination of several weak classifier can be at least as good as one strong classifier.
2. The combination of selecting strong classifier is usually at least as good as the best of base classifier.

7.2.1. (Majority) Voting

The simplest means of classification over multiple base classifiers is *voting*.

Essentially run the multiple base classifiers over the test data and select the classes predicted by the most no. of base classifier. For example, for three base classifiers C_1, C_2, C_3 and their combination C^* using majority voting on three instances t_1, t_2, t_3.

Classes\Instances	t_1	t_2	t_3	C^*
C_1	√	×	√	√
C_2	√	×	×	×

For continuous class set, take the average over the numeric prediction of base classifiers

7.2.2. Construct Base Classifiers

Instance Manipulation: Generate multiple training datasets sets through different feature subsets and train the base classifier over each trainset.

Feature Manipulation: Generate multiple training datasets through different feature subsets and train the base classifier over each trainset.

Algorithm Manipulation: Semi-randomly adjust internal parameters within a given algorithms within a given algorithms to generate multiple base classifiers.

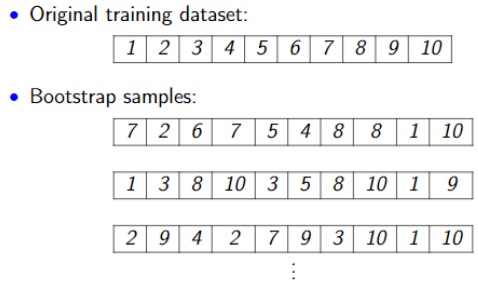
7.2. Ensemble Meta-Algorithms – Bagging

7.2.3. Bagging Intuition

The more data we have, the lower the variance (by the CLT) and therefore better the performance.

By constructing novel datasets through a combination of random sampling & replacement, we can create more training dataset for the combined classifier to use. Where we:

1. Randomly sample the original dataset ***N times with replacement.***
2. The new dataset has the same size, where any instance is absent with probability of $\left(1-\frac{1}{N}\right)^N$
3. Construct k random datasets for k base classifiers & evaluate performance.



7.2.4. Analysis of Bagging

1. Simple method based on sampling and voting
2. Possible to use ***parallel computing*** for each individual base classifier
3. Highly effective over noisy datasets (outlier **may** vanish)
4. Performance is generally ***significantly*** better than the base classifier but maybe substantially worse.

7.3. Random Forests

7.3.1. RF Intuition

Hyperparameters

1. **Number of tree B –** tuned based on *"Out-of-bag"* error
2. **Feature sub-sample size –** as sub-sample size ↑, both the strength & the correlation ↑ $=> log_2|F|+1$

Interpretation

Logic behind predictions on individual instances can be tediously followed through various trees.

A ***Random Tree*** is a variant of a Decision Tree, but

1. At each node, only *some* of the possible attributes are considered (i.e., a fixed proportion τ of all attributes are used)
2. Attempts to control the number of unhelpful attributes in the feature set.

3. Much faster to build a *deterministic* DT but increases model variance.

The structure of a Random Forest is an ensemble of Random Tree which is built using a different bagged training data set and classified via voting.

The idea behind random Forests is to minimize the overall model variance without introducing combined model bias.

7.3.2. RF Analysis

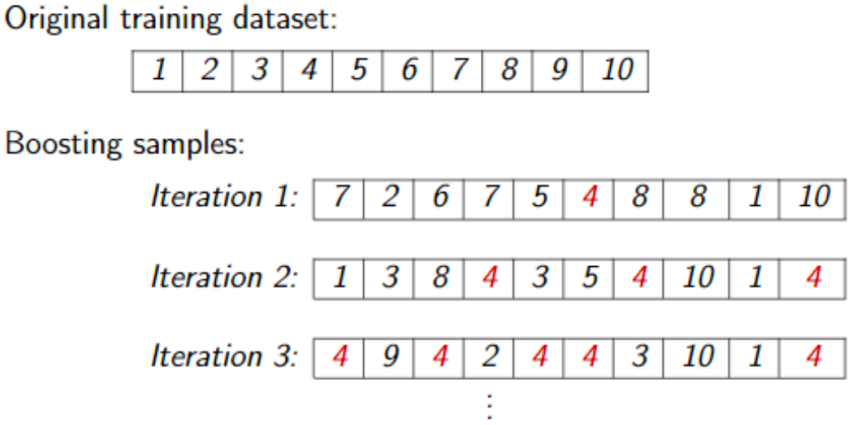
1. Robust to over-fitting at sacrifice of interpretability
2. Generally, performs well & super-efficient

7.4. Boosting

7.4.1. Boosting Intuition

Tune base classifier to focus on the instances which are *hard to classify.*

- Iteratively change the distribution and weights of training instances to reflect the performance of the classifier on the previous iteration.
- Start with each training instance having a probability of $\frac{1}{N}$ to be included in the sample.
- Over *T* iterations, train the classifier & *update the weight* of each instance according to whether it's correctly classified.
- Combine the base classifiers via *weighted voting.*



7.4.2. Example – *AdaBoost*

With base classifiers: $C_1, C_2, \dots, C_i, \dots, C_T$ and training instances $\{(x_j, y_j) | \ j = 1,2, \dots, N\}$, set initial instance weights $\left\{w_j^{(1)} = \frac{1}{N} \middle| \ j = 1,2, \dots N\right\}$

An improvement to the default Boosting. Essentially **re-initialize the instance weights if the classifier error rate ϵ is over 0.5.**

For

1. Construct the classifier C_i in iteration i – and compute the error rate C_i

$$\epsilon_i = \sum_{j=1}^{N} w_j^{(i)} \, \delta\big(C_i(x_j) \neq y_j\big)$$

where δ is an indicator function, which outputs 1 when condition is true?

2. Importance of C_i: α_i the weight associated with the classifiers' votes

$$\alpha_i = \frac{1}{2} \log\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$$

3. Update the instance weight (preparation for iteration $i + 1$:

$$w_j^{i+1} = \frac{w_j^{(i)}}{Z^{(i)}} \begin{cases} e^{-\alpha_i}, & C_j(x_j) = y_j \\ e^{\alpha_i}, & C_j(x_j) \neq y_j \end{cases}$$

4. Continue iterating for $i = 2, \ldots, T$. But reinitialize the instance weights, whenever $\epsilon_i > 0.5$
5. Classification: combine base classifiers

$$C^*(x) = \underset{y}{\text{argmax}} \sum_{i=1}^{T} \alpha_i \delta(C_i(x) = y)$$

7.4.3. Analysis of Boosting

1. Base Classifiers: Decision Stumps (OneR) or *Decision Trees*
2. Mathematically complicated but computationally cheap
3. Based on iterative sampling and weighted voting, but still computationally expensive than bagging.
4. Guaranteed performance in the form of error bond over the training data.
5. Has the tendency to overfit (saturating instances)

7.5. Comparison between Bagging/RF vs. Boosting

Bagging/ Random Forest	Boosting
• Parallel Sampling	• Iterative Sampling
• Simple Voting	• Weighted voting
• Homogeneous Classifier	• Homogeneous classifiers
• Minimize Variance	• Minimize instance bias
• Not prone to overfitting	• Prone to overfitting

7.6. Stacking

7.6.1. Stacking intuitions

Smooth the errors over a range of algorithms with different biases.

1. Can use simple voting but assumes the classifiers have equal performances.
2. Train a classifier over the outputs of the base classifiers by using nested cross-validation to reduce bias.

Example – Given a training dataset (X, y):

1. Train the combination of base classifiers (even in parallel).
2. Predict the probability / label of training instance.
3. Train the meta-classifier (Logistic Regression) over the predicted probability / label.

7.6.2. Analysis of Stacking

1. Mathematically simple but computationally expensive.
2. Able to combine classifiers with varying performance.
3. Generally, yield in better results than the best of base classifiers

7.7. Feature Selection Methods

7.7.1. Full Wrapper Method

Full wrapper method compares all combinations of features, the exhaustive comparative mechanism prompts other feature selections to be performed like below.

7.7.2. Wrapper Method – Native Approach

Choose subset of attributes that give best performance on the validation data. (w.r.t respect to a single learner).

1. Feature sets will have optimal performance on development data.
2. Take a very long time and therefore only practical for very small datasets.
3. Complexity: $O\left(\frac{2^m}{6}\right)$

7.7.3. Wrapper Method – Greedy Approach (Sequential Forward Selection)

A greedy approach would be train & evaluate the model on each single attribute. Then, we choose the best attribute for each train-test until it converges to a model

1. Converges much quicker compared to Naïve Approach
2. Usually converges to a sub-optimal model
3. Complexity: $O(\frac{m^2}{2})$, but converges more quickly in practice
4. Assumes independence of attributes

7.7.4. Wrapper Method – Ablation Approach (Sequential Backward Selection)

The ablation approach starts with all attributes, then removes an attribute each iteration until it diverges to a model.

1. Assumes all attributes are independent
2. Complexity: $O(m^2)$

7.7.5. Embedded Methods

Intuition – evaluate "goodness" of each attribute. To some degree, learners such as SVM, Logistic Regression and DTs perform some form of feature selection within the training process.

However, it should be noted that these learners will still benefit with feature selection prior to building the model.

7.8. Filtering Methods

In the exam, create a **Contingency table** for the attributes to answer problems for full marks.

	$a = Y$	$a = N, (\bar{a})$	$Total$
$c = Y$	$\sigma(a, c)$	$\sigma(a, \bar{c})$	$\sigma(c)$
$c = N, (\bar{c})$	$\sigma(a, \bar{c})$	$\sigma(\bar{a}, \bar{c})$	$\sigma(\bar{c})$
$Total$	$\sigma(a)$	$\sigma(\bar{a})$	M

$$P(a, c) = \frac{\sigma(a, c)}{M}$$

7.8.1. Pointwise Mutual Information (PMI)

Assume attribute A is independent from class C. (i.e., $P(A, C) = P(A)P(C)$). Attributes with the greatest PMI are most correlated with a class, and attributes with low PMI are more uncorrelated with a class. Where,

$$PMI(Attribute = a, class = c) = \log_2 \frac{P(A, C)}{P(A)P(C)}$$

7.8.2. Mutual Information

Mutual Information is the sum of all PMI for every attribute value to every class label.

$$MI(Attribute, Class) = \sum_{i \in D} \sum_{j \in C} \Pr(i, j) \log_2 \left(\frac{\Pr(i, j)}{P(i)P(j)} \right)$$

noted that $0 \log_2 0$ is defined as 0.

7.8.3. $\chi 2$ -Selection

Conduct statistical test to check the independence of a feature and the class. Contingency table.

	$a = Y$	$a = N, (\bar{a})$	$Total$
$c = Y$	W	X	$W + X$
$c = N, (\bar{c})$	Y	Z	$Y + Z$
$Total$	$W + Y$	$X + Z$	M

$$E(W) = \frac{(W + Y)(W + X)}{W + X + Y + Z}$$

- If $O(W) > E(W)$: a occurs more often with c then we would expect at random – Predictive
- If $O(W) < E(W)$: a occurs less often with c then we would expect at random – Predictive
- If $O(W) \approx E(W)$: a occurs as often with c then we would expect at random – NOT Predictive

$$\chi^2 = \sum_{i \in \{a, \bar{a}\}} \sum_{j \in \{c, \bar{c}\}} \frac{\left(O_{i,j} - E_{i,j}\right)^2}{E_{i,j}}$$

High χ^2 indicates the dependency between a feature and the class.

7.9. Common issues in feature selection

1. **Nominal attributes of multivalue** → treat as multiple binary variables
    a. Problem: results can be difficult to implement, e.g. Sunny: 1, {Overcast, Rainy}: 0, is Outlook = rainy?
2. **Fit Continuous Variable in Gaussian** → one solution is to discretize values.
3. **Multiclass Prediction difficulties** → e.g., Geotag for Melbourne Sydney, what about Swanston, Dockland etc.

8 – Evaluation

8.1. Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large training dataset will also approximate the target function well over held-out test examples.

1. **Overfitting** – Has the classifier tune itself to the idiosyncrasies of the training data, rather than learning its generalizable properties?
2. **Consistency** – Is the classifier able to flawlessly predict the class of all training instances?
3. **Generalization** – How well does the classifier generalize from the specifics of the training examples to predict the target function.

8.2. Overfitting

More ***training instance usually results in a better model***, likewise more ***evaluation instances result in a more reliable estimate of effectiveness.***

A good model should fit the training data well and generalize well to unseen data.

The expectation is that training and test data are randomly selected from the same population, but neither are the entire population.

Possible **evidence of overfitting**:

1. **Lack of coverage of population** sample could lead to a poor model
2. Decision boundary distorted by noise (and ***tightly fit the model***/ complex divider)
3. Maybe due to small number of samples, or due to non-randomness or scarcity in training sample (known as ***Sampling Bias***)

8.3. Regularization

Solutions to overfitting. Mathematically, regularisers are the norm of parameters – a total length of a vector: The L_p norm of the $\vec{\beta}$ is defined as:

$$||\beta||_p = \sqrt[P]{\sum_k |\beta_k|^p}$$

and the optimization objective function is

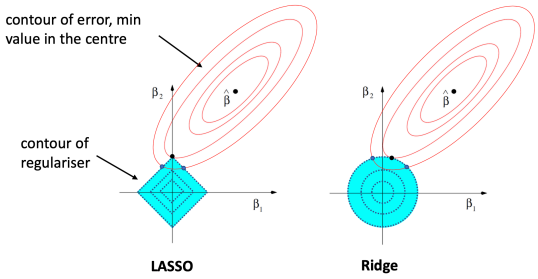
$$\hat{\beta} = \underset{\beta}{\text{argmax}}[Error(\beta; \{\boldsymbol{X}, y\}) + \lambda\psi(\boldsymbol{\beta})]$$

where

$$\lambda > 0 \; is \; hyperparameter$$
$$\psi(\boldsymbol{\beta}) \; is \; regularizer$$

1. **L2- norm Regularization** (Regression) is $\psi(\boldsymbol{\beta}) = ||\beta||_2^2 = \sum_k |\beta_k|^2$, where the penalty term encourages solutions where most parameter values are ***small***

2. **L1- norm Regularization** (LASSO) is $\psi(\boldsymbol{\beta}) = ||\beta||_1 = \sum_k |\beta_k|$, where the penalty term encourages solutions where few parameters are ***non-zero***



8.4. Model Bias & Variance

Model Bias in statistical definition w.r.t estimation $\hat{\theta}$ for true θ are

$$Bias(\hat{\theta}, \theta) = E[\hat{\theta}(x) - \theta(x)]$$
$$Var(\hat{\theta}, \theta) = E[(\hat{\theta}(x) - E[\hat{\theta}(x)])^2]$$

8.4.1. Terminology for Bias

1. **Model Bias:** The tendency of our classifier to make systematically wrong prediction.
2. **Evaluation Bias:** The tendency of our evaluation strategy to over/underestimate the effectiveness of our classifier.
3. **Sampling Bias:** If our training/evaluation data set isn't representative of the population (breaking the Inductive Learning Hypothesis)

We **DON'T** want ***high bias*** and ***low variance*** (always makes the same prediction mistake regardless of the dataset)

Having a **low bias** but **high variance** is acceptable in SOME circumstances.

Having a **low bias** but **low variance** is BETTER for generalization.

8.4.2. Bias in Regression

In Model Bias in regression context, the signed error can be calculated for every evaluation instances. Assuming every instance is independent, bias is the average of these signed error below:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)$$

1. The model is **biased** if the predictions are systematically **higher/lower** than the true value
2. The model is **unbiased** if the predictions are systematically **correct** or some of the predictions are too high/low.

8.4.3. Bias in Classification

Label predictions can't be too high/low – "biased towards the majority class" means our model predicts too many instances as the majority class. Comparing class distribution:

1. An unbiased classifier produces labels with the same distribution as the actual distribution.

2. A biased classifier produces labels with a different distribution from the actual distribution.

8.4.4. Model Variance

Relating to the tendency of different training sets to produce different models or predictions with the same type of learner.

1. A model has high variance if a different randomly sampled training set leads to very different predictions on the evaluation set.
2. A model has low variance if a different randomly sampled training set leads to similar predictions, independent of whether the predictions are correct

8.5 Bias & Variance Evaluation

Controlling bias and variance in evaluation involves:

1. **Holding partition size**
    a. More training Data – less model variance, more evaluation variance
    b. Less training but more test data – more model variance, less evaluation variance
2. **Repeated random subsampling and K-fold Cross-validation**
    a. Less variance than hold out
3. **Stratification** – less model and evaluation bias
4. **Leave-one-out Cross Validation**
    a. No sampling bias, lowest bias/variance in general.

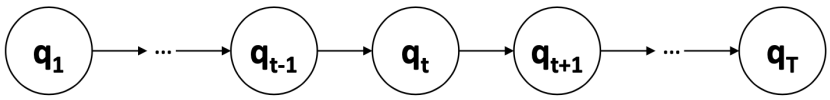
9 – Sequential Model & Hidden Markov Chains

Sequential structure consists of the likes of *time series analysis, speech recognition, and genomic data*, where it needs a structure classification (like Hierarchical structure for web; or graph structure for influence matrix for a social network) to capture the interaction between instances.

9.1. Markov Chains

Markov chain describes the system that transits from one state to another according to certain probabilistic rules

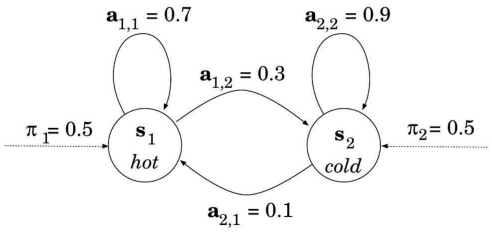
- **States:** a set $S = \{s_i\}$
- **Initial State Distribution:** $\Pi = \{\pi_i\}, \sum_i \pi_i = 1$
- **Transition Probability Matrix:** $A = \{a_{ij}\}, \sum_j a_{ij} = 1\,, \forall i$



Where there is an underlying assumption for a sequence of state $q_1, q_2, \dots, q_{T-1}, q_T$ state t that q_t only depends on the immediately preceding state q_{t-1}

$$P(q_t|q_1, \dots, q_{t-1}) = P(q_t|q_{t-1})$$

For example,



$$S = \{\text{hot}, \text{cold}\}$$

$$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{bmatrix} \quad \pi = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\begin{aligned} P(q_1 = hot, q_2 = hot, q_3 = cold) &= P(q_3 = cold | q_2 = hot, q_1 = hot) P(q_2 = hot) \\ &= P(q_3 = cold | q_2 = hot) P(q_2 = hot | q_1 = hot) P(q_1 = hot) \\ &= 0.3 \times 0.7 \times 0.5 = 0.105 \end{aligned}$$

Running time are $O(T)$

9.2. Hidden Markov Chains

Markov Chain assumes the likelihood of transitioning into a given state depends only on the current state, and not the previous state(s)

HMM take the form $\mu(A, B, \Pi)$:

- **States:** a set $S = \{s_i\}$
- **Observations:** a set $O = \{o_k\}$
- **Initial State Distribution:** $\Pi = \{\pi_i\}, \sum_i \pi_i = 1$
- **Transition Probability Matrix:** $A = \{a_{ij}\}, \sum_j a_{ij} = 1\,, \forall i$
- **Output probability matrix:** $B = \{b_i(o_k)\}, \sum_k b_i(o_k) = 1, \forall i$

and there are two independence assumptions:

$$P(q_t | q_1, \dots, q_{t-1}, o_1, \dots, o_{t-1}) = P(q_t | q_{t-1})$$
$$P(o_t | q_1, \dots, q_{t-1}, o_1, \dots, o_{t-1}) = P(o_t | q_{t-1})$$

There are **three tasks** for HMM:

- **Evaluation:** estimate the likelihood of an observation sequence – *Given an HMM μ and observation sequence Ω, determine the likelihood $P(\Omega|\mu)$*
- **Decoding:** find the most probable state sequence – *Given an HMM μ and observation sequence Ω, determine the most probable hidden state sequence Q*
- **Learning:** estimate parameters of HMM – *Given observation sequence Ω, the set of possible state S and observations O in an HMM, learning parameters $\mu = (A, B, \Pi)$*

Hidden state sequence is hard to calculate $O(TN^T)$, as $P = \sum_Q P(\Omega|Q, \mu) P(Q|\mu)$ where $T =$ length of sequence, and $N =$ the no. of state. There is an requirement to use alternative algorithms

Pros of HMM: efficient approach to structure classification; **Cons:** limited representation of context (only observation and state sequences). Tend to suffer from floating point underflow.

9.2.1. Forward Algorithms (Scaling coefficients)

Starting from the initial state Π, and calculates the next state

Initialization: $t = 1$, state $i = 1, \dots, N$

$$a_1(i) = \pi_i b_i(o_1)$$

Induction: $t = 1, \dots, T-1$ state $i = 1, \dots, N$

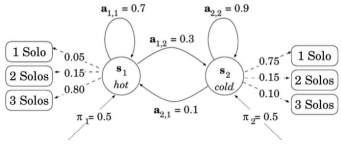
$$a_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) a_{ji} \right) b_i(o_{t+1})$$

Termination:

$$P(\Omega|\mu) = \sum_{i=1}^{N} \alpha_T(i)$$

Decoding: $O(compute\ probability\ +\ sort) = O(TN^T + N^T \log N^T)$

- P(3−Solos, 3−Solos, 1−Solo|μ)?



- Initialisation/induction: $\alpha_t(i) = P(o_1, o_2, \dots, o_t, q_t = s_i|\mu)$

	$t=1$	$t=2$	$t=3$
$\alpha_t(hot)$	0.5 × 0.8 = 0.4	[0.4 × 0.7 + 0.05 × 0.1] × 0.8 = 0.228	[0.228 × 0.7 + 0.0165 × 0.1] × 0.05 = 0.0080625
$\alpha_t(cold)$	0.5 × 0.1 = 0.05	[0.4 × 0.3 + 0.05 × 0.9] × 0.1 = 0.0165	[0.228 × 0.3 + 0.0165 × 0.9] × 0.75 = 0.0624375

- Termination:

P(3−Solos, 3−Solos, 1−Solo|μ) = 0.0080625 + 0.0624375 = 0.0705

9.2.2. Viterbi Algorithm

Use dynamic programming to find the most likely sequence of states. Runtime $O(TN^2)$

Initialization: $t = 1$, state $i = 1, \dots, N$

$$\delta_i = \pi_i b_i(o_{t+1})$$
$$\psi_1(i) = 0$$

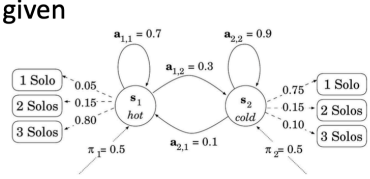
Induction: $t = 1, .., T-1$, state $i = 1, \dots, N$

$$P_{best} = \max_{0 \le i \le N} \delta_T(i)$$
$$q_T^* = \operatorname*{argmax}_{1 \le i \le N} \delta_T(i)$$

Backtrack: to establish the best path

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$
$$for\ t = T-1, T-2, \dots, 1$$

- Most probable state sequence, given the observation sequence: 3−Solos, 3−Solos, 1−Solo



- Initialisation/induction:

	$t=1$	$t=2$	$t=3$
$\delta_t(hot)$	0.5 × 0.8 = 0.4	max(0.4 × 0.7, 0.05 × 0.1) × 0.8 = 0.224	max(0.224 × 0.7, 0.012 × 0.1) × 0.05 = 0.00784
$\psi_t(hot)$	0	← *hot*	← *hot*
$\delta_t(cold)$	0.5 × 0.1 = 0.05	max(0.4 × 0.3, 0.05 × 0.9) × 0.1 = 0.012	max(0.224 × 0.3, 0.012 × 0.9) × 0.75 = 0.0504
$\psi_t(cold)$	0	↖ *hot*	↖ *hot* ★

9.2.3. Learning HMM

Supervised case: assume we have labelled data, it's possible to use simple MLE to learn the parameters of our model.

$$a_{ij} = P(s_j|s_i) = \frac{freq(s_i, s_j)}{freq(s_i)}$$
$$b_i(o_k) = P(o_k|s_i) = \frac{freq(o_k, s_i)}{freq(s_i)}$$
$$\pi_i = P(q_1 = s_i) = \frac{freq(q_1 = s_i)}{\sum_j freq(q_1 = s_j)}$$

Unsupervised case (No state label): use forward-backward algorithm (*Baum-Welch algorithms*, and out-of-scope...lmao)

9.2.4. HMM Application

Test classification, Automation Speech Recognition or Optical Character Recognition.

10 – Deep Learning

Deep learning is the combination of multiple hidden layers with sufficient data to train the models (a very large dataset – on average millions of instances)

10.1. Deep Learning: Representation examples

10.1.1. Text/Document Representation

- A "bag of words" representation of a document is a sparse vector represent word count.
- "Token *n*-grams" maintain more information from the document, but are much more sparse
- A token/n-gram can be suggestive of class, but no more than suggestive.

10.1.2. Image Representation

- An individual RGB pixel can only be suggestive of some image property.
- Only reasonable in combination (sequences of pixels can define a shape/feature)
- Feature engineering for image representation is difficult. (Curse of Dimensionality)

10.2. Representation Learning

In deep learning, each layer of model extracts out a dense real-valued vector representation of test instances (term: ***embedding***). Representation learning is the transformation of raw inputs into a laten intermediate representation that is more amenable to learning.

10.2.1. Embedding Advantage

1. We have fewer features which mean faster train/test – low dimensional representations of the inputs
2. Feature engineering is done on the fly
3. With careful choice of representation, we can represent an instance by combinations of embedding(s), allowing us to use the trained network for problems it was never designed for.

10.3. Neural Networks

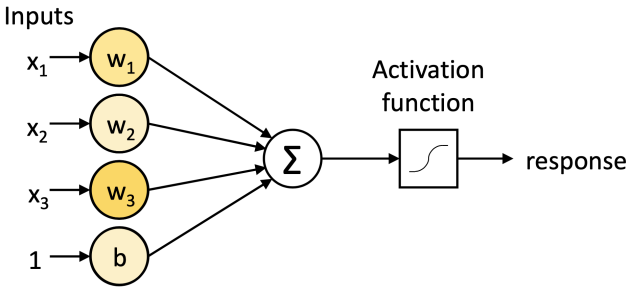
NN (Neural Network) are composed of a network of single neuron. Since NNs ae universal approximators, they are guaranteed to generalize better than any other ML methods: NNs own their own hidden layers, they don't require any feature engineering.

Advantages	Disadvantages
• State-of the-art performance • Embedding may be useful for a range of tasks	• Difficult to interpret • Requires large training dataset • Sensitive to noise, dataset bias

10.3.1. Artificial Neuron

Neurons are defined as

- A vector of numeric inputs $x_i =< x_{i1}, x_{i2}, x_{in} > \in R^n$
- A scalar output $y_i \in R^n$
- An activation function f (which is a hyperparameters)



$$y_i = f\left(\left[\sum_j w_j x_{ij}\right] + b\right) = f(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)$$

Training a NN entails identifying the weight $\boldsymbol{w}$ which try & minimize the errors on the training data.

A classic method of training for neural network is using perceptron, where each iteration over examples in a training dataset is termed an ***epoch***

10.3.2. Perceptron

Perceptron can be written as:

$$f(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) = \begin{cases} 1, & if\ \boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq 0 \\ 0, & e/w \end{cases}$$

Perceptron Algorithms

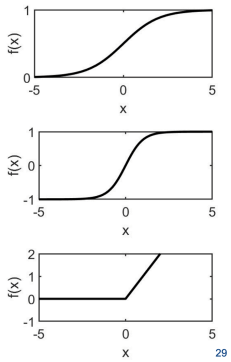
$\boldsymbol{w} \leftarrow rand()$
for $each\ \ (x_i, y_i)$ ***do***:
    $\boldsymbol{y_i} = f(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)$
    for $each\ \ \boldsymbol{w_j}$ ***do***:
        $\boldsymbol{w_j} \leftarrow \boldsymbol{w_j} + \boldsymbol{\lambda}(\boldsymbol{y_i} - \widehat{\boldsymbol{y_i}}) \cdot \boldsymbol{x_{ij}}$
    end
end

EXAM TIPS: you must remember how to setup convergence summary! (λ is η in the exam)

10.3.3. Activation Functions

In a perceptron, activation functions map linear responses to the range we want (often [0,1]). There are few non-linear activation functions are used commonly

1. Logistic sigmoid function, equivalent to LR (σ):
$$f(x) = \frac{1}{1 + e^{-x}}$$
2. Hyperbolic tan ($tanh$):
$$f(x) = \frac{e^{2x} - 1}{e^{2x} - 1}$$
3. Rectified Linear Unit (ReLU):
$$f(x) = \max(0, x)$$



The Perceptron Algorithms guarantees convergence for linearly separable data, but

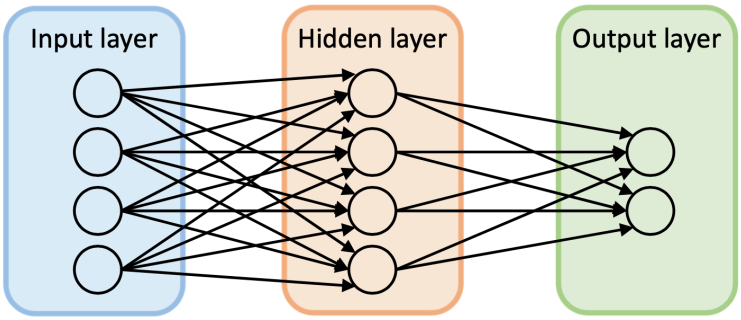
1. The convergence point depends on the initialization
2. The convergence point will also depend on the learning rate.

10.4. Multilayer Perceptron (MLP)

The advantage of NN comes from "stacking" several neurons in different ways such as layering (parallel neurons of varying size) and then creating extra layers of varying size.

Consider a ***fully connected feed-forward neural network*** (also known as *MLP*) meaning the network can learn any linear/non-linear dynamically. it consists of the following:

1. **Input layer** is made up of individual features
2. **Hidden layer** is made up of a set number of neurons, each of which is connected to all neurons in the preceding layers, and all neurons in the following layer.
    a. With **hidden layer**, the learning done via ***back-propagation*** (Stochastic gradient descent for weight + biases)
3. **Output layer** which combines the inputs from the preceding layer into the input. Common options are:
    a. Binary Classification – 1 output neuron with step activation function
    b. N-way Classification – N output neurons and SoftMax activation function.
    c. Regression – 1 output neuron with identity activation function



MLP has **AT LEAST** one hidden layer.

MLP Pro & Cons:

Pro	Cons
• Can be adapted to many types of problems (classification, regression) • Universal approximator – can model arbitrary basis functions • Representation learning in hidden layers	• Very high number of parameters – slow to train, prone to overfitting, high memory requirements • Stochastic GD – not guaranteed to converge to the same solution every time.

10.5. Convolutional Neural Networks (CNN)

CovNET are crucial to the computer vision applications. They are made of the following:

- Convolutional layers
- Max-pooling layers
- Fully connected layers

and two components:

- a **Kernel** in the form of a matrix, which is overlaid on different sub-regions of the image and combine through an ***element-wise product***
- A **Stride** which defines how many positions in the image to advance the kernel on each iteration.

For example, assuming the following 3×4 image and $s = 1$:

1	10	1	1
1	2	10	2
0	0	1	10

image

1	0
0	1

2x2 kernel

the first convolution would be:

x1	x0	1	1
x0	x1	10	2
0	0	1	10

convolution

$1 \times 10 + 10 \times 0 + 1 \times 0 + 2 \times 1 = 3$

The full convolutional output would be:

$$\begin{bmatrix} 3 & 20 & 3 \\ 1 & 3 & 20 \end{bmatrix}$$

10.5.1. CNN: Pooling

Pooling is comprised of 3 components:

- a **Kernel** in the form of a matrix, which is overlaid on different sub-regions of the image to determine the extent of "pool".
- a **Stride** which defines how many positions in the image to advance the kernel on each iteration.
- the pooling basis using either **max** (1-max or *k*-max) or average.

Max-pooling adds *translation invariance* – network response is the same even if features are in a slightly different position in the image. This reduces the amount of data going to the next layer, which reduces computation time. However, it also discards important feature information.

10.5.2. Advantages vs. Disadvantages of Convolutional layers

Pros	Cons

• **Efficient** – just one neuron learns to recognize a feature like "vertical edge" everywhere in the input. • **Preserves spatial relations** (output represents "where" features are present, not just "what" features are present)	• Limited kernel size means model is limited to learning local features

y connected layer vs. Convolutional layers

Fully connected layer	Convolutional Layer
• Each neuron is connected to every neuron in the last layer. • The neuron learns some combination of the last layer's responses. • The output to the next layer is the neuron's response.	• Each Neuron is connected to a small patch of all the last layer's output • The neuron learns a convolutional kernel • The output to the next layer is the input convoluted with the neuron's kernel.

10.5.3. Deep Learning – Advantages

1. Massive improvement in empirical accuracy over standard datasets for vision & speech recognition tasks
2. Possible to model very large contexts/neighbourhoods compared to conventional models.
3. Easy to combine different input modalities
4. Easy to use using Tensorflow, PyTorch, Keras

10.5.4. Deep Learning – Disadvantages

1. Any saving relating to feature engineering are outweighed by costs in terms of building NN architecture
2. Very expensive to train over large datasets (think about the implications this has on hyperparameter tuning)
3. Probably full overblown claims about the capabilities of DL.

10.6. Dropout

Dropout is a form of regularization that forces neuron to find useful feature independently. This is done by randomly discard some neuron (set output = 0). Effectively this trains multiple architectures in parallel.

11 – Unsupervised Learning & Mixture Model

11.1. Clustering

Clustering is ***unsupervised learning*** where there is no implicit definition of class. This means it learn structure from data alone. However, there should be assumption make about the structure of the data such as exclusive/overlapping clusters, hierarchical clusters, and what defines a "good" group.

11.1.1. Deterministic vs. Probabilistic

Deterministic clustering implies that each instance is a member of one cluster and one cluster ONLY. This also means clusters cannot overlap.

Probabilistic cluster implies that each instance has a weight in each class and that the clusters overlap.

11.1.2. "Soft" *k*-means Clustering

It's possible to obtain a **probabilistic version** of *k*-mean, where each instance is probabilistically assigned to each of *k* clusters.

1. Set $t = 0$ and randomly initialize the centroids $\mu_1^0, \mu_2^0, \dots, \mu_k^0$
2. Soft assigned each instance x_j to a cluster based on:

$$z_{ij} = \frac{\exp\left(-\beta \left\| x_j - \mu_i^t \right\|\right)}{\sum_l \exp\left(-\beta \left\| x_j - \mu_j^t \right\|\right)}$$

   where $\beta > 0$ and is called the **stiffness parameter.**
3. Update each of the centroids via ***weighted avg. of all instances***:

$$\mu_i^{t+1} = \frac{\sum_j z_{ij} x_j}{\sum_j z_{ij}}$$

   It's the probability of this centroid occurring, given that the instances belong this centroid.
4. Set $t \leftarrow t + 1$ and repeat until centroids stabilize.

This is a combination of *Overlapping, Probabilistic, Partitionin*g and *Batch Clustering* methods.

11.2. Mixture Model

11.2.1. Finite Mixture

a *finite mixture* is a mixed distribution with *k* component distribution.

11.2.2. The Expectation Maximization (EM) Algorithm

An algorithm with guaranteed *positive* hill-climbing characteristic (greedy approach) and primarily use to estimate hidden parameter values or to cluster memberships.

Essentially a generalized "soft" *k*-means:

1. **Expectation Step**: Based on the current estimate of the parameters, calculate the log-likelihood $L = \sum_i log \sum_j P(c_j) P(x_i | c_j)$
2. **Maximization Step**: Compute the new parameter distribution that maximizes the log-likelihood.

Pros	Cons
• Guaranteed to have a positive hill climbing behaviour. • Converges fast • Results in a probabilistic cluster assignment	• Possibility of getting stuck in a local maximum (if it's not convex) • Relies on an arbitrary k which needs to be estimated • Tends to overfit the data.

11.2.3. Convergence

The log-likelihood gives us an estimate of how "good" the cluster model is.

Convergence can be found by fining the difference between log-likelihoods between iterations. Once it falls below a predefined level ϵ, we can say that the cluster model has converged.

11.2.4. **Gaussian Mixture Model (GMM) w/ EM algorithm**

GMM represents a distribution as composed of *k* Gaussian distribution

Using **GMM with EM algorithms** (Assume data is a mixture of 2 normal distribution):

1. Each instance is drawn from one of the two distribution: probability drawing from distribution 1 is γ and from 2 is $(1-\gamma)$
2. The probability of density can be written as $g(x) = \gamma\phi_{\mu_1,\sigma_1}(x) + (1-\gamma)\phi_{\mu_2,\sigma_2}(x)$, where $\phi_{\mu,\sigma} \sim N(\mu,\sigma)$
3. Estimate the parameter $\gamma, \mu_1, \sigma_1, \mu_2, \sigma_2$ we can postulate how likely "best-fit" distribution for each instance. To do so, we find parameters that ***maximize the log-likelihood*** of the instance x_i:

$$\sum_{i=1}^{N} \log\left[\gamma\phi_{\mu_1,\sigma_2}(x_i) + (1-\gamma)\phi_{\mu_1,\sigma_2}(x_i)\right]$$

   a. This is difficult to solve numerically, but if we know which instances come from which generating distribution, the computation is simpler

$$Distribution\ 1\text{:}\ \sum_{j\in D_1} \log\left[\phi_{\mu_1,\sigma_1}(x_i) + \log(\gamma)\right]$$

$$Distribution\ 2\text{:}\ \sum_{j\in D_1} \log\left[\phi_{\mu_2,\sigma_2}(x_i) + \log(1-\gamma)\right]$$

4. Embed the points above into Expectation step and maximization step.

11.3. Unsupervised Cluster Evaluation

A good cluster "analysis" will have one or both of:

- High cluster cohesion (similar instances all grouped closely)

$$cohesion = \frac{1}{\sum_{x,y\in C_i} proximity(x,y)}$$

- High cluster separation (instances from different clusters are distinctly separated)

$$separtion(C_i, C_j) = \sum_{x\in C_i, y\in C_{j\neq i}} proximity(x,y)$$

An alternative is to use $SSE = \sum_{i=1}^{k}\sum_{x\in C_j} proximity(x, c_j)^2$ using *Euclidean* for numeric data proximity or *Hamming* for nominal.

11.4. Supervised Cluster Evaluation

Supervised evaluation pertains that cluster "validity" measures the degree of predicted labels to the true labels. Evaluation is done within a cluster and across clusters, where we want **LOW entropy and HIGH purity**

$$Purity = \sum_{i=1}^{k} \frac{|C_i|}{N} \max_j P_i(j)$$

$$Entropy = \sum_{i=1}^{k} \frac{|C_i|}{N} H(x_i)$$

where x_i = the distribution of class labels in cluster i.

For example,

Cluster	Play=Yes	Play=No	Entropy	Purity
1	2	0	0	1
2	6	4	0.97	0.6

$$Entropy_{C1} = -1 \log\left(\frac{2}{2}\right) - 0 \log\left(\frac{2}{2}\right) = 0$$

$$Entropy_{C2} = -\frac{6}{6+4}\log\left(\frac{6}{6+4}\right) - \frac{4}{6+4}\log\left(\frac{4}{6+4}\right) = 0.97$$

$$Purity_{C1} = \max\left(\frac{2}{2}, \frac{0}{2}\right) = 0$$

$$Purity_{C2} = \max\left(\frac{6}{6+4}, \frac{4}{6+4}\right) = 0.6$$

12 – Semi-supervised Learning

Semi-supervised algorithms learn from both unlabelled and labelled data, where it's useful for clustering if we have some domain knowledge indicating inter-cluster compatibility. This solves expensive data labelling data via unsupervised components.

- Training data consists of L labelled instance $\langle x_i, y_i \rangle$ and U unlabelled instance $\langle x_j \rangle$.
- Often $U > L$, therefore the goal is to learn a better classifier from $L \cup U$ than is possible L alone.

12.1. Self-training Algorithms

Bootstrapping (Self-training) Algorithms

Repeat ***do***:
    for each f_i *in supervised* ***model F***:
        Train model $\boldsymbol{f_i(L)}$ *using supervised learning method*
        for each u_j *in* U:
            U'← $HighConfidence(U', f_i(u_j))$
        $U \leftarrow U \backslash U'$ *# Remove* U' *unlabelled set*
        $L \leftarrow L \cap U'$ *# Add* U' *to labelled set with classifier prediction as "true" label*
Stop *when L does not change*

Self-training examples include using *1-NN, Naïve Bayes*

12.2. Active learning

Active learning builds off hypothesis that a classifier can achieve higher accuracy with fewer training instances if it's allowed to have some say in the ***selection of the training instances***.

The ***underlying assumption is that labelling is a finite resource,*** which should expend in a way which optimizes ML effectiveness.

Active learners pose ***queries*** (unlabelled instances) for labelling by an ***oracle*** (a human annotator)

12.2.1. Query Strategies

One **Simple strategy** is to query instances where the classifier is ***least confident of the classification:***

$$x = \underset{x}{\operatorname{argmax}}(1 - P_\theta(\hat{y}|x))$$

where

$$y = \underset{x}{\operatorname{argmax}}\, P_\theta(\hat{y}|x)$$

Alternatively, **margin sampling** selects the queries where the classifier is least able to distinguish between two categories e.g.:

$$x = \underset{x}{\operatorname{argmax}}(P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x))$$

where $\hat{y}_1, \hat{y}_2$ are the 1st and 2nd most probable labels for x

$$y = \underset{x}{\operatorname{argmax}}\, P_\theta(\hat{y}|x)$$

Or better, use entropy as an uncertainty measure

$$x = \underset{x}{\operatorname{argmax}} \left[-\sum_i P_\theta(\hat{y}|x) \log_2 P_\theta(\hat{y}|x) \right]$$

12.2.2. Active Learning – Multiple Classifiers

Use a **query-by-committee (QBC),** where a suite of classifiers is trained. Then, the instance with the highest disagreement (entropy) is selected for querying.

12.2.3. Active Learning – Pro & Cons

Advantages	Disadvantages
• Empirically robust strategy to increase accuracy	• Often difficult to justify these strategies theoretically. • Introduce bias, resulting in a dataset that might not be useful for ML tasks. • Maybe sensitive to label noises.