CZ4041 - Machine Learning

1 Bayesian Classifiers

Probabilities

Sum Rule

$P(A) = \sum_B P(A,B)$
$P(A) = \sum_B \sum_C P(A,B,C)$

Product Rule

$P(A,B) = P(B|A) \times P(A) = P(A|B) \times P(B)$

Bayes Theorem

$$P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A))}{P(B)}$$

(Generalised case)

$$P(A_1...A_k|B_1...B_p) = \frac{P(B_1...B_p, A_1...A_k)}{P(B_1...B_p)}$$

Bayesian Classifiers

Bayesian classifiers aim to find the mapping $f: \mathbf{x} \Rightarrow y$ for supervised learning in the form of conditional probability $P(y|\mathbf{X})$ via Bayes rule.

$$P(y|\mathbf{X}) = \frac{P(y,\mathbf{X})}{P(\mathbf{X})} = \frac{P(\mathbf{X}|y)P(y)}{P(\mathbf{X})}$$

For a classification with C classes, given a data instance $\mathbf{x}^*$:

$$y^* = c^* \, if \, c^* = \operatorname*{argmax}_c P(y=c|\mathbf{x}^*)$$

Applying Bayes rule,

$$P(y=c|\mathbf{x}^*) = \frac{P(\mathbf{x}^*|y=c)P(y=c)}{P(\mathbf{x}^*)}$$

Therefore,

$$y^* = \operatorname*{argmax}_c \frac{P(\mathbf{x}^*|y=c)P(y=c)}{P(\mathbf{x}^*)}$$

$$= \operatorname*{argmax}_c P(\mathbf{x}^*|y=c)P(y=c)$$

2 Bayesian Decision Theory

Incorporating cost of misclassification on top of simple Bayesian Classifiers.

Loss/Cost

Actions: a_c, i.e., predict $y = c$
Define λ_{ij} as the cost of a_i when optimal action is a_j. E.g.:
$\lambda_{00} = 0$ (predict correctly)
$\lambda_{11} = 0$ (predict correctly)
$\lambda_{01} = 10$ misclassify 1 as 0
$\lambda_{00} = 1$ misclassify 0 as 1

Expected Risk

Expected risk for taking action a_i:

$$R(a_i|\mathbf{x}) = \sum_{c=0}^{C-1} \lambda_{ic} P(y=c|\mathbf{x})$$

To classify, for all actions, calculate expected risk, then choose the action with the minimum risk.

Special Case: 0/1 loss

$$\lambda_{ij} = \begin{cases} 0 \text{ if } i = j \\ 1 \text{ if } i \neq j \end{cases}$$

$\therefore R(a_i|\mathbf{x}) = 1 - P(y=i|\mathbf{x})$
In this case,

$$\text{Choose } a_i \text{ if } R(a_i|\mathbf{x}) = \min_{a_c} R(a_c|\mathbf{x})$$

Is equivalent to:

$$\text{Predict } y = c^*$$

$$\text{if } P(y=c^*|\mathbf{x}) = \max_c P(y=c|\mathbf{x})$$

3 Naïve Bayes Classifiers

Independence

A is **independent** of B, if:
$P(A,B) = P(A|B) \times P(B) = P(A) \times P(B)$
$P(A,B) = P(B|A) \times P(A) = P(A) \times P(B)$
Or,
$P(A|B) = P(A)$
$P(B|A) = P(B)$

Conditional Independence

A is **conditionally independent** of B, given C if:
$P(A|B,C) = P(A|C)$

Naïve Bayes Classifier

1. Assumption: conditional independence of features given label

$$p(\mathbf{x}|y=c) = P(x_1,...,x_d|y=c)$$

$$= P(x_1|y=c)P(x_2|y=c)...P(x_d|y=c)$$

$$= \prod_{i=1}^{d} P(x_i|y=c)$$

To classify a test record $\mathbf{x}^*$, compute the posteriors for each class:

$$p(y=c|\mathbf{x}^*) = \frac{(\prod_{i=1}^{d} P(x_i^*|y=c))P(y=c)}{P(\mathbf{x}^*)}$$

Since $P(\mathbf{x}^*)$ is constant for each class c, it is sufficient to choose the class that maximises the numerator term.

$$y^* = \operatorname*{argmax}_c (\prod_{i=1}^{d} P(x_i^*|y=c))P(y=c)$$

Estimating Cond Prob (Discrete)

$$P(x_i = k|y=c) = \frac{|(x_i - k) \wedge (y=c)|}{|y=c|}$$

Estimating Cond Prob (Continuous)

$$P(x_i|y=c) = \frac{1}{\sqrt{2\pi\sigma_{ic}^2}} e^{-\frac{(x_i - \mu_{ic})^2}{2\sigma_{ic}^2}}$$

Supposing there are N_c instances in class c,
Sample mean:

$$\mu_{ic} = \frac{1}{N_c} \sum_{j=1}^{N_c} x_{ij}$$

Sample variance:

$$\sigma_{ic}^2 = \frac{1}{N_c - 1} \sum_{j=1}^{N_c} (x_{ij} - \mu_{ic})^2$$

Laplace Estimate

Alternative prob estimation for discrete features.

$$P(x_i = k|y=c) = \frac{|(x_i - k) \wedge (y=c)| + 1}{|y=c| + n_i}$$

where n_i is #distinct values of x_i. In extreme cases with no training data, $P(x_i = k|y=c) = \frac{1}{n_i}$

M-estimate

A more general estimation:

$$P(x_i = k|y=c) = \frac{|(x_i - k) \wedge (y=c)| + m\tilde{P}}{|y=c| + m}$$

Where m is a hyperparameter and $\tilde{P}$ is prior information of $P(x_i = k|y=c)$. (e.g., domain knowledge)
Extreme case with no training data: $P(x_i = k|y=c) = \tilde{P}(x_i = k|y=c)$

4 Bayesian Belief Networks

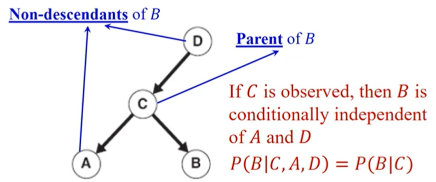
Suppose all features are **discrete** (if there are continuous and discrete, estimation is much more difficult)

Two key elements:
1. A directed acyclic graph (DAG) encoding dependence relationships between a set of variables

2. A probability table associating each node to immediate parent nodes

DAG: Conditional Independence

A node in a Bayesian network is conditionally independent of its non-descendants, **if its parents are known.**



IMPORTANT! If A and B are conditionally independent given C, we have:
1. $P(A|B,C) = P(A|C)$
2. $P(A,B|C) = P(A|C)P(B|C)$

Important! Using BBN for Inference

Given a BBN, and an inference(prediction) task:
1. Translate problem into probabilisitc language
2. If the probabilities to be estimated cannot be obtained from the probability tables of the BBN, then
A. Identify a subgraph which captures the dependence between input variables (features) and output variable (class)

B. Based on the network topology, apply product rule, sum rule and the properties of conditional independence and independence to induce equivalent forms of the probabilities until all probabilities can be found from the probability tables.

5 Decision Trees

- Greedy strategy, split records based on feature test that optimises certain criterion
Key issues:
1) How to split the records?
- Specifying feature test condition
- Determining best split
2) When to stop splitting?

Determining Test Conditions

Splitting based on binary features

2 Possible outcomes (e.g. Yes/No)

Splitting based on discrete features

- Multi-way split: Use as many partitions as distinct values
e.g.: Marital Status $\Rightarrow$ [Single], [Divorced], [Married]

- Binary split: Divides possible values as 2 subsets, need to find optimal partitioning
e.g.: Marital Status $\Rightarrow$ [Single, Divorced], [Married]

Splitting based on continuous features

- Binary split: $(x_j < v)$ or $(x_j \geq v)$
- Multi-way split (Discretization)

Consider all possible splits and find the best cut
Can be very computationally intensive

Determining Best Split

Using measure of node impurity – favour split with low degree of impurity

Measure of Impurity: Entropy

Entropy at a given node t:

$$E(t) = -\sum_c P(y=c;t) log_2 P(y=c;t)$$

Information Gain

$\Delta_{info} = E(\text{parent}) - E(\text{children})$
To get entropy for children, get entropy of all children nodes and normalize by # of training examples in each child node.
Suppose a parent node t is split into P partitions (children),
$\Delta_{info} = E(t) - \sum_{j=1}^{P} \frac{n_j}{n} E(j)$

Disadvantage: Tends to prefer splits that result in large number of partitions.

Penalizing large number of partitions (Gain Ratio)

$$\Delta_{\text{InfoR}} = \frac{\Delta_{\text{info}}}{\text{SplitINFO}}$$

$$\text{SplitINFO} = -\sum_{i=1}^{P} \frac{n_i}{n} log_2(\frac{n_i}{n})$$

Stopping Criterias

1. All data belong to same class
2. Stop expanding when all data have similar feature vals
3. Early termination (avoid overfitting)

CZ4041 - Machine Learning

6 Generalisation

Overfitting: Test error rate increase when training error decrease
Underfitting: Model too simple, both training and test error large

Training errors: error on training set, $e(T)$
Generalisation errors: error on previously unseen testing set, $e'(T)$

Estimating Generalisation Errors

Optimistic Estimate
Assume training set is good representation of overall data
$e'(T) = e(T)$
Decision tree induction algo select model with lowest training error rate.

Occam's Razor
Include information of model complexity when evaluating a model.
$e'(T) = e(T) + N \times k$
where N is the number of leaf nodes and k is a hyperparameter $k > 0$

Using Validation Set
Divide training data to 2 subsets, 1 for training and 1 for estimating generalisation error.

Addressing overfitting

Pre-Pruning
- Stop if number of instances is less than user-specified threshold
- Stop if expanding current node does not improve generalisation errrors

Post-Pruning
- Grow tree to its entirety
- Trim nodes in bottom-up fashion
- If generalisation error improves after trimming, replace sub-tree by new leaf

7 KNN Classifiers

- Instance based, lazy learner - no model built
- - "Training" is very efficient
- - Classifying unknown test instances are relatively expensive
- Requires training data to be stored in memory

Classification steps:
1. Compute distance to other training instances
2. Identify K nearest neighbors
3. Use class labels of neighbors to determine class of instance

Choosing K
- K too small, sensitive to noise
- K too large, neighborhood may include points from other classes

Distance Metric
Euclidean distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{d}(x_{ik} - x_{jk})^2}$$

Voting Schemes
- Majority voting (sensitive to choice of k)
- Distance-weight voting (weight the influence of neighbor x_i according to distance to test data)

$$w_i = \frac{1}{d(x^*, x_i)^2}$$

$$y^* = \operatorname*{argmax}_c \sum_{(x_i, y_i) \in \mathcal{N}_{x^*}} w_i \times I(c = y_i)$$

Other issues with KNN
Scaling issues - features may need to be scaled
Solution: Normalisation on features of different scales.

Normalisation
- Min-max normalisation

$$v_{new} = \frac{v_{old} - \min_{old}}{\max_{old} - \min_{old}}(\max_{new} - \min_{new})$$

- Standardisation (z-score normalisation) (μ: mean, σ:standard deviation)

$$v_{new} = \frac{v_{old} - \mu_{old}}{\sigma_{old}}$$

8 ANN

$y = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
$w_0 = -\theta,\ X_0 = 1$
Where θ is the threshold term, $\mathbf{w}$ is the weights vector and $\mathbf{x}$ is the input vector. An additional dimension is added to both vectors such that the sum of products would minus the threshold term, θ.

Activation functions

Sign Activation function

$$\text{sign}(z) = \begin{cases} 1, & z \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Since function is not differentiable, when finding derivative of the activation function, we set y = z, and the derivative of y with respect to z would be = 1

Sigmoid Activation function

$$a(z) = \frac{1}{1 + e^{-\lambda z}}$$

When $\lambda = 1$, it's called the sigmoid function.

Derivative of sigmoid:

$$\frac{\partial \hat{y}(z)}{\partial z} = y(z) \cdot (1 - y(z))$$

Error/Loss

$$E = \frac{1}{2}(y_i - \hat{y}_i)^2$$

Updating Weights

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

Applying chain rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \frac{\partial E(\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}(z)}{\partial z} \frac{\partial z(\mathbf{w})}{\partial \mathbf{w}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda(-(y_i - \hat{y}_i))(1)(\mathbf{x}_i)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda(-(y_i - \hat{y}_i))(1)(\mathbf{x}_i)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda(y_i - \hat{y}_i)\mathbf{x}_i$$

9 Support Vector Machines

Decision Boundary
The decision boundary of a SVM can be defined as:
$w_1 x_1 + w_2 x_w + b = 0$
General form:
$\mathbf{w} \cdot \mathbf{x} + b = 0$

Making predictions
During training, the values of $\mathbf{w}$ and b is learned.

For any test example $\mathbf{x}^*$

$$\begin{cases} f(\mathbf{x}^*) = +1, \text{if} \mathbf{w} \cdot \mathbf{x}^* + b \geq 0 \\ f(\mathbf{x}^*) = -1, \text{if} \mathbf{w} \cdot \mathbf{x}^* + b < 0 \end{cases}$$

Other notes (Linear Algebra):

Inner Product

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{d}(u_i \times v_i)$$

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|_2 \times \|\mathbf{v}\|_2 \times cos(\theta)$$

L2 Norm (Length of vector)

$$\|\mathbf{u}\|_2 = \sqrt{\mathbf{u} \cdot \mathbf{u}} = \sqrt{\sum_{i=1}^{d}(u_i \times u_i)}$$

Induction
- Direction of $\mathbf{w}$ is orthogonal (perpendicular) to the decision boundary.

Parallel hyperplanes:
$\mathbf{w} \cdot \mathbf{x} + b = k$
$\mathbf{w} \cdot \mathbf{x} + b = -k$
(After rescaling $\mathbf{w} = \mathbf{w}/k$, $b = b/k$)
$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = -1$

$$\|\mathbf{w}\|_2 \times d = 2$$

$$d = \frac{2}{\|\mathbf{w}\|_2}$$

Margin Maximisation
Therefore, decision boundary can be learnt by maximising the margin, $d = \frac{2}{\|\mathbf{w}\|_2}$. However, this is not easy. Change this into a minimisation problem.

Minimise: $\frac{\|\mathbf{w}\|_2^2}{2}$
Constraints:
$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1, \text{if} y_i = 1$
$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1, \text{if} y_i = -1$
OR, $y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$

Optimisation Problem for SVM

$$min_{w,b} \frac{\|\mathbf{w}\|_2^2}{2}$$

$$\text{s.t.} y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

Multi-Class SVM
Given 3-class problem C_1, C_2 and C_3
Create 3 SVM binary classifiers: 1. Positive C_1, Negative C_2 & C_3
2. Positive C_2, Negative C_1 & C_3
3. Positive C_3, Negative C_1 & C_2

Use majority voting to determine class for test example.

	C_1	C_2	C_3
$f_1(x^*) = -1$	0	1	1
$f_2(x^*) = 1$	0	1	0
$f_3(x^*) = -1$	1	1	0
Total Votes:	1	3	1

10 Linear Regression

Error for 1-D Linear Regression Model
Sum-of-squares (SSE) error:

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(w \times x_i - y_i)^2$$

Learn linear model in terms of w by minimising the error

$$w^* = \operatorname*{argmin}_w E(w)$$

To solve the unconstrained minimisation problem, set derivative of $E(w)$ w.r.t w to zero

$$\frac{\partial E(w)}{\partial w} = \frac{\partial(\frac{1}{2}\sum_{i=1}^{N}(w \times x_i - y_i)^2)}{\partial w} = 0$$

Closed form solution:

$$w = \frac{\sum_{i=1}^{N} y_i \times x_i}{\sum_{i=1}^{N} x_i^2}$$

More general case (multi-dimension)
$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$
- By defining $w_0 = b$ and $X_0 = 1$, w and x are of d+1 dimensions
$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$

Error for Linear Regression Model

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w} \cdot \mathbf{x}_i - y_i)^2$$

Learn linear model in terms of $\mathbf{w}$ by minimizing the error (with regularisation term)

$$\mathbf{w}^* = \operatorname*{argmin}_w E(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

Closed-Form Solution

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T y$$

11 Ensemble Learning

Necessary Conditions
1) Base classifiers are independent of each other
2) Base classifiers should do better than classifier that performs random guessing (i.e., acc > 0.5)

Error rate of ensemble
Supposing N independent base classifiers with error ϵ:

$$P = \sum_{i=(N//2)+1}^{N} \epsilon^i(1-\epsilon)^{N-i}$$

Ensemble Methods

Bagging

- Sample examples **with replacement** and build model on each bootstrap sample.
- Use majority voting to determine class label of ensemble classifier
- A bootstrap sample contains approximately 63.2% of original training data

Boosting

1) Initially, all examples are assigned equal weights
2) Bootstrap sample is drawn and a model is trained from sample
3) Model is then used to classify examples from training set
4) Update weights of examples after the end of boosting round

- Wrongly classified - increase
- Correctly classified - decrease
- Examples not drawn - unchanged

5) Use weighted voting, each classifier would have different weights

Random Forests

- Specifically designed for decision tree classifiers 1) Choose T, number of trees to grow
2) Choose m', number of features used to calculate best split (Typically 20%)
3) For each tree
- Choose training set via bootstrapping
- For each node, randomly choose m' features and calculate best split
- Trees fully grown and not pruned
4) Use majority vote among all trees

Combination Methods

- Majority voting
- Weighted voting
- Simple average:
$f_M(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} f_i(\mathbf{X})$
- Weighted average:
$f_M(\mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} w_i f_i(\mathbf{X})$
Where $w_i \geq 0$, and $\sum_{i=1}^{T} w_i = 1$

Combining by Learning

Combiner: second-level learner, or meta-learner
Combiner takes output of base classifiers as features, and learn to classify based on output label.