IAML Notes

Lecture 1

- Supervised Learning
  - Predict output y when given input x
  - Categorical y: classification
  - Real-valued y: regression
- Unsupervised Learning
  - Internal representation of the input e.g. clustering, dimensionality
  - Getting labels is difficult and expensive
- Other areas of ML
  - Learning to predict structured objects e.g. trees, graphs
  - Reinforcement learning - learning from "rewards"
  - Semi-supervised learning - combination

Classification

No text !!!

Regression

$$f(x) = w_0 + w_1 x_1 + w_D x_D$$

$$x = (x_1, \dots, x_D)^T$$

- $f(x)$ is a linear function
- Setting of $w_1, \dots, w_D$ is done by minimising the cost function
- Usual score function is $\sum_{i=1}^{n} = (y^i - f(x^i))^2$.

Clustering

No text !!!

Structure of Supervised algorithms

- Define the task
- Decide on the model structure (choice of inductive bias)
- Decide on the score function (judge quality of fitted model)
- Decide on optimization/search method to optimize the score function

Supervised learning is inductive, i.e. we make generalizations about the form of $f(x)$ based on instances D.

Learning is impossible without making assumptions about f!!

Conditional Probability

$$p(X = x|Y = y) = \frac{p(x, y)}{p(y)}$$
$$p(X, Y) = p(Y)p(X|Y) = p(X)p(Y|X)$$

From the product rule,

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

From the sum rule the denominator is

$$p(X) = \sum_y p(X|Y)p(Y)$$

Say that Y denotes a class label, and X an observation. Then $p(Y)$ is the prior distribution for a label, and $p(Y|X)$ is the posterior distribution for Y given a data point x.

Conditional independence

$$p(X1|X2, Y) = p(X1|Y)$$

Marginal independence

$$p(X|Y) = P(X)$$

Continuous Random Variables

$$\mu = \int xp(x)dx$$
$$\sigma^2 = \int (x - \mu)^2 p(x)dx$$

For numerical discrete variables, convert integrals to sums.

Gaussian distribution

1D

$$p(x|\mu, \sigma) = N(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{-\frac{(x - \mu)^2}{2\sigma^2}\right\}$$

- $\sqrt{2\pi\sigma^2}$ is the normalisation constant

More D

$$p(x) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu)^T \mathbf{\Sigma}^{-1}(x - \mu)\right\}$$

- Σ is the covariance matrix

$$\Sigma = E[(x - \mu)(x - \mu)^T]$$
$$\Sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)]$$

Likelihood

The probability of data D given model M is $p(D|M)$. This is called the **likelihood**.

$$p(D|M) = \Pi_{i=1}^{N} p(x_i|M)$$

i.e. the product of generating each data point individually.

This is a result of the independence assumption. Try different models, pick the one with highest likelihood -> **Maximum likelihood approach**.

Bernoulli distribution

To find most probably number of ones in n tosses, differentiate with respect to p, to find when it is maximised. Useful to take a log - it is monotonic.

$$\prod_i p(x_i|\mu, \sigma^2) = -\frac{1}{2}\sum_i \frac{(x_i - \mu)^2}{\sigma^2} - \frac{n}{2} log(2\pi\sigma^2)$$

$$\mu = \frac{\sum_i x_i}{n}$$

$$\mu = \frac{\sum_i (x_i - \mu)^2}{n}$$

Lecture 2

Attribute-value pairs

- unordered bag of features, if structure essential, embed it
- categorical - red, yellow, blue
  - mutually exclusive
  - only equality meaningful
- ordinal - bad, good, better, best
  - there is natural ordering
  - comparison is meaningful, maths isn't
- numerical - 1, 3, 8, 4
  - usually need to normalise, remove outliers
  - maths is meaningful

Skewed distributions

- Affects regression, kNN, NB; but not DTs
- fix: take a log or atan, then normalise
- cumulative distribution function $x' = F(x) = P(X < x)$

Non-monotonic effect of attributes

- age -> probability of boxing win (not too young, not too old)
- affects regression NB, DTs (gain); less important for kNN
- if there's an obvious way to make monotonic: use it

Picking attributes - instances in the same class y should have representations x that are somehow "similar".

Object recognition - segment image into regions, compute features describing the region.

Dealing with structure - Represent path from root to leaf as a feature.

Outliers - Isolated instances of a class that are unlike any other instances observed. Remove them or threshold.

Missing values - Try to understand why, if categorical have special category, if numerical then fill in mean or remove them. Some learners handle it.

Lecture 3

Bayesian classification

$$P(y|x) = \frac{P(x|y)P(y)}{\sum_{y'} P(x|y')P(y')}$$

- P(y): prior probability of each class
  - encodes how which classes are common, which are rare
  - apriori much more likely to have common cold than Avian flu
- P(x|y): class-conditional model
  - describes how likely to see observation x for class y
  - assuming it's Avian flu, do the symptoms look plausible?
- P(x): normalize probabilities across observations
  - does not affect which class is most likely (arg max)

An outlier has a low probability under every class.

Independence assumption - assume $x_1, x_2 \ldots$ conditionally independent given y.

$$P(x_1 \ldots x_d|y) = \prod_{i=1}^{d} P(x_i|x_1 \ldots x_{i-1}, y) = \prod_{i=1}^{d} P(x_i|y)$$

Probabilities of going to the beach and getting a heat stroke are not independent. May be independent if we know the weather is hot. Hot weather *explains* the dependence between beach and heatstroke. Thus, **class value explains all the dependence between attributes**.

Decision boundary

- Different means, same variance: straight line / plane
- Same mean, different variance: circle / ellipse
- General case: parabolic curve

Problems with Naive Bayes

- Zero frequency problem: add-one (Laplace) smoothing

- Word independence: add lots of hammy words to spam email
- Unable to handle correlated data - double weight to "same" features

Good stuff about Naive Bayes

- Missing data: leave out attribute that's missing (independence assumption).
- Good complexity
  - $O(nd + cd)$ training time complexity
  - $O(dc)$ space complexity
  - $O(d)$ insertion / deletion - store partial sums instead of mean/variance
  - c - classes, n - instances, d - attributes

Naive Bayes structure

- Task: c-class classification
- Model: $c \times d$ independent assumptions
  - Continuous: Gaussian, Discrete: Bernoulli
- Score function: class-conditional likelihood
- Optimisation: analytic solution

Lecture 4

Decision trees - try to *understand* when John plays tennis. Split into sets, are they pure? If not repeat, if yes then done and see which subset new data falls into.

ID3 Algorithm

1. A <- the best attribute for splitting the examples
2. Decision attribute for this node <- A
3. For each value of A, create new child node
4. Split training examples to child nodes
5. If examples perfectly classified then stop else iterate over new child nodes.

Which attribute to split on?

Want to measure purity of the split so we are more certain after the split. We use entropy:

$$H(S) = -p_{(+)} log_2 p_{(+)} - p_{(-)} log_2 p_{(-)}$$

- S: subset of training examples
- $p_{(+)}/p_{(-)}$: % of positive/negative examples in S

Information Gain

We want many items in pure sets and expect drop in entropy after split:

$$Gain(S, A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V)$$

Overfitting

We can always classify training examples perfectly - keep splitting until each node has 1 example. Doesn't work on new data.

To avoid this, we stop splitting when not statistically significant. Grow the tree, then post-prune based on a validation set.

Sub-tree replacement pruning (WF 6.1)

- for each node
  - pretend to remove node + all children
  - measure performance on validation set
- remove node that results in greatest improvement
- repeat until harmful

Structure of DT

- Task: classification, discriminative
- Model: decision tree
- Score function: information gain at each node, prefer short trees and high-gain near root
- Optimisation: greedy search from simple to complex guided by information gain

Problems of DT

- Biased towards attributes with many values
- Doesn't work on new (un-observed) data
- Only axis aligned splits of data

GainRatio

Idea is to penalise attributes with many values.

$$SplitEntropy(S, A) = -\sum_{V \in Values(A)} \frac{|S_V|}{|S|} log \frac{|S_V|}{|S|}$$

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitEntropy(S, A)}$$

Continuous DTs - Create ranges to split on which can be optimised (WF 6.1).

Multi-class classification

- predict most frequent class
- Entropy: $H(S) = -\sum_c p_{(c)} log_2 p_{(c)}$
- $p_{(c)}$ % of examples of class c in S

Regression

- predicted output: mean of the training examples in subset
- requires a different definition of entropy (dunno)
- can use linear regression at the leaves (WF 6.5)

Good stuff about DTs

- interpretable
- easily handles irrelevant attributes (Gain = 0)
- can handle missing data - (WF 6.1)
- very compact # of nodes << d after pruning
- very fast at testing time $O(depth)$

Random decision forest

- Grow K different decision trees
  - Pick a random subset S_r of training examples
  - grow a full ID3 tree T_r (no pruning)
    - when splitting pick from d << D random attributes
    - compute gain based on S_r instead of full set
  - repeat for r = 1 ... K
- Given new data point X
  - classify X using each tree and then use majority vote (state-of-the-art performance)

Lecture 5

Over-fitting

- predictor too complex
  - fits noise in the training data
  - patterns will not reappear
- predictor F over-fits if:
  - we can find another predictor F'
  - which makes more mistakes on the training set
  - makes less mistakes on the future data

Under-fitting

- predictor too simplistic (rigid)
- not powerful enough to capture salient patterns in data
- we can find another predictor with smaller training and future data errors

Under/Over-fitting knobs

- regression: order of polynomial
- NB: number of attributes
- DT: number of nodes in the tree / pruning confidence
- kNN: number of NN
- SVM: kernel type, cost parameter

Training Error - average error over all training data.

$$E_train = \frac{1}{n}\sum_{i=1}^{n} error(f_D(x_i), y_i)$$

Generalisation Error - how well we'll do on future data (can never compute).

$$E_{gen} = \int error(f_D(x), y)p(y,x)dx$$

Estimating Generalisation Error

- set aside part of training set as testing set (unbiased)
- predict on testing set - estimate of generalisation error

Confidence Interval for Future Error

What range of error can we expect for future test sets?

$$CI = E + -\sqrt{E(1-E)/n} \times \phi^{-1} \times \frac{1-p}{2}$$

Training, validation, testing sets

- Training set: construct classifier
- Validation set: pick algorithm + knob settings
- Testing set: estimate future error rate

Cross-validation

Split dataset into n sets test on each, train on the others. Average the results at the end.

- best possible classifier learned
- high computational cost
- classes not balanced in training/testing sets

Stratification - keep classes balanced across training/testing sets. Split each class into K sets and choose one for each validation.

Classification Error

$$\frac{FP + FN}{TP + TN + FP + FN}$$

- true negatives and true positives are good
- false negatives and false positives are bad
- False Alarm Rate = FP / (FP + TN)
- Miss Rate = FN / (TP + FN)
- Recall = TP / (TP + FN)
- Precision = TP / (TP + FP)

Always report at least two, it is trivial to get one to 100% or 0%.

Utility and Cost

- $Detectioncost = C_{FP} \times FP + C_{FN} \times FN$
- $F - measure = \frac{2}{1/Recall + 1/Precision}$

Receiver Operating Characteristic

- $Falsepositiverate = P(f(x) > t|ham)$
- $Truepositiverate = P(f(x) > t|spam)$

ROC - Plot TPR vs. FPR from -infinity to infinity. Alternative to classification error.

Regression error functions

- (Root) mean squared error
  - $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2}$
  - popular, differentiable but sensitive to single large outliers, mean and scale
- Mean absolute error
  - $\frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i|$
  - less sensitive to outliers, sensitive to mean and scale
- Correlation coefficient
  - $n\sum_{i=1}^{n}\frac{f(x_i)-\mu_f}{\sigma_f} \times \frac{y_i-\mu_y}{\sigma_y}$
  - insensitive to mean and scale
  - useful for ranking tasks e.g. recommend a film

Lecture 6

kNN

- k = 1
  - Insensitive to class prior
  - Very sensitive to outliers
- k > 1
  - k affects smoothness of the boundary
  - large value: everything classified as most probable class (priors)
  - small value: highly variable, unstable decision boundaries
  - set aside portion of data (validation set) and vary k to find ideal

Distance measures

- **Euclidian**: symmetric, spherical, all dimensions equal, sensitive to changes in one attribute
- **Hamming**: counts number of attributes that differ (logical AND)
- **Minkovski**: $D(x, x') = \sqrt[p]{\sum_d |x_d - x'_d|^p}$
  - p = 2: Euclidean
  - p = 1: Manhattan
  - p = 0: Hamming (logical AND)
  - p = ∞: logical OR
- **Kullback-Leibler divergence -** measure of information lost (excess bits to encode x with x')
  - $D(x, x') = -\sum_d x_d \log \frac{x_d}{x'_d}$

Resolving ties

- use odd k (when only 2 classes)
- random flip of coin
- pick class with higher prior
- 1NN: use 1st nearest neighbour to decide

Missing values - have to "fill in", otherwise can't compute distance (average)

Parzen Windows Classifier - use constant distance around a point and find dominant class in variable number of examples.

Probabilistically, we could express kNN and Parzen Windows as follows:

$$P(o|x) = \frac{\sum_i 1_{y_i=o} \times 1_{x_i \in R(x)}}{\sum_i 1_{x_i \in R(x)}}$$

that is, we a point has a vote if it falls into a certain radius R(x).

However, using kernels we can give points weight depending on how far they are:

$$P(o|x) = \frac{\sum_i 1_{y_i=o} \times K(x_i, x)}{\sum_i K(x_i, x)}$$

kNN Problems and good stuff

- almost no assumptions about data
  - smoothness: nearby regions of space -> same class
  - assumptions implied by distance function
  - non-parametric approach (let the data speak for itself)
    - only parameters to infer are k and distance function
    - easy to update in online setting
- need to handle missing data
- sensitive to class outliers (mislabelled instances)
- sensitive to irrelevant attributes (distance)
- computationally expensive
  - space: store all training data
  - time: compute distance to every point $O(nd)$
  - expensive at testing, not training (bad)

Making kNN fast

- reduce dimensionality: feature selection
- reduce number of examples to compare to
  - K-D trees: low-dimensional real-valued data, may miss neighbours
    - $O(dlogn)$
  - Inverted lists: high-dimensional, discrete data, sparse data
    - $O(d'n')$ where $d' << d, n' << n$,
  - Fingerprinting: high-dimensional, sparse or dense, may miss neighbours
    - $O(dn')$ where $n' << n$

Using K-D

- pick random dimension, find median, split data, repeat for all 'children'
- find NNs for a point
  - find region of point
  - compare point to all points in region

Using inverted list

- list all training examples containing a particular attribute
- assumption: most attribute values are zero (sparse)
- find NNs for a point
  - merge inverted lists for attributes present in the point
  - compare point to all points in the merged list

Lecture 7

Linear Regression

The relationship between input and output is linear.

$$f(\mathbf{x};\mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D = \mathbf{w}^T \phi(\mathbf{x})$$

where $\phi(\mathbf{x}) = (1, x_1, \ldots, x_D)^T$.

If ϕ is the design matrix (contains all training vectors as rows, prepended by 1), then $\phi \times \mathbf{w}$ is a vector of predicted values on the inputs.

Model parameters

$$\mathbf{y} = \phi \times \mathbf{w}$$

We know $\mathbf{y}, \phi$ but not $\mathbf{w}$. Can we do $\mathbf{w} = \phi^{-1} \times \mathbf{y}$?

- ϕ is not a square matrix (could be but not good)
- system is over-constrained (n equations for D+1 parameters)
- data has noise

We want a loss function $O(\mathbf{w})$ that:

- we minimise w.r.t. $\mathbf{w}$
- at minimum looks like $\mathbf{y}$

Loss function

- Squared error: $O(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x_i})^2$
  - Residual error: $y_i - \mathbf{w}^T \mathbf{x_i}$

Fitting a linear model to data

$$O(\mathbf{w}) = (\mathbf{y} - \phi\mathbf{w})^T (\mathbf{y} - \phi\mathbf{w})$$

- To minimise $O(\mathbf{w})$ set partial derivatives to zero.
- Analytical solution is $\mathbf{w} = (\phi^T \phi)^{-1} \phi^T \mathbf{y}$
  - the phi stuff is a pseudo inverse
  - analytical solution is exact solution (same thing)

Probabilistic interpretation of O(w)

If we assume that $y = \mathbf{w}^T \mathbf{x} + \epsilon$, where $\epsilon = N(0, \sigma^2)$ then this implies that $y|x_i = N(w^T x_i, \sigma^2)$. After we take a log of this, we get:

$$-\log p(y_i|x_i) = \log \sqrt{2\pi} + \log \sigma + \frac{(y_i - \mathbf{w}^T \mathbf{x_i})^2}{2\sigma^2}$$

The last term is squared error so minimising $O(\mathbf{w})$ is equivalent to maximising likelihood! Squared residual errors allow estimation of:

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x_i})^2$$

Linear regression is sensitive to outliers!

Multiple regression - if there are q different targets (e.g. temperature, humidity based on conditions) then we introduce a different weight for each target dimension and do regression separately.

Basis expansion

We can use a function $\phi(\mathbf{x})$ to transform our design matrix. This function could be a polynomial, gaussian or sigmoid etc.

Transforming features

- 1 if Intel
- 2 if AMD
- 3 if Apple

could be transformed into:

- x_1 = 1 if Intel, 0 otherwise
- x_2 = 1 if AMD, 0 otherwise
- x_3 = 1 if Apple, 0 otherwise

Radial basis function (RBF) models

Set $\phi_i(x) = exp(-\frac{1}{2}|\mathbf{x} - \mathbf{c_i}|^2/\alpha^2)$. c_i tells you where it is centred and α tells you the width. Choosing subset of training set as centres is quite effective.

Why not use RBF?

- you might need too many RBFs especially in high dimensions
- Too many RBFs means we probably over-fit

Lecture 8

Decision boundary

In a two class linear classifier we learn a function

$$F(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0$$

that represents how aligned the instance is with y = 1.

the instance belongs to class y = 1 if $F(\mathbf{x}, \mathbf{w}) > 0$.

Two class discrimination

- We want a linear probabilistic model
- $P(y = 1|\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ is not between 0 and 1
- Instead we do $P(y = 1|\mathbf{x}) = f(\mathbf{w}^T \mathbf{x})$
- f must be between 0 and 1
- $P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x})$

Logistic function

We use the sigmoid function:

$$f(z) = \sigma(z) = \frac{1}{1 + exp(-z)}$$

Linear weights + logistic squashing function == **logistic regression**

We model the class probabilities as:

$$P(y = 1|\mathbf{x}) = \sigma(\sum_{j=0}^{D} w_j x_j) = \sigma(\mathbf{w}^T \mathbf{x})$$

$\sigma(z) = 0.5$ when $z = 0$, thus we use $\mathbf{w}^T \mathbf{x} = 0$ as decision boundary (probability is half).

Logistic regression

When we exclude w_0 from $\mathbf{w}$, the magnitude of that vector represents the certainty of the classifications. For small magnitude, the probabilities within the region of the decision boundary will be near 0.5. For large magnitude it will be nearer 0 or 1.

w_0 just shifts the hyper-plane, only the other terms affect the angle.

Learning logistic regression

- Write out the model and the likelihood
- Find derivatives of the log likelihood w.r.t. the parameters
- Adjust the parameters to maximise the log likelihood
- Assume data is independent and identically distributed
- Call the data set $D = (x_1, y_1), \ldots, (x_n, y_n)$

The likelihood is:

$$p(D|\mathbf{w}) = \prod_{i=1}^{n} p(y = y_i|\mathbf{x_i}, \mathbf{w})$$

we split this into our two classes

$$p(D|\mathbf{w}) = \prod_{i=1}^{n} p(y = 1|\mathbf{x_i}, \mathbf{w})^{y_i} (1 - p(y = 1|\mathbf{x_i}, \mathbf{w}))^{1-y_i}$$

Hence the log likelihood is:

$$L(\mathbf{w}) = \sum_{i=1}^{n} y_i \log \sigma(\mathbf{w}^T \mathbf{x_i}) + (1 - y_i) \log (1 - \sigma(\mathbf{w}^T \mathbf{x_i}))$$

To maximise the likelihood

We take a gradient, differentiate w.r.t. components of w (need to do this numerically, no analytical solution):

$$\frac{\delta L}{\delta w_j} = \sum_{i=1}^{n} (y_i - \sigma(\mathbf{w}^T \mathbf{x_i}))x_{ij}$$

Perceptron

Function, simpler than logistic regression:

$$f(z) = sign(z) = 1 if z > 0, -1 \text{ otherwise}$$

- repeat for all i in 1, 2, ... n
- $y' \leftarrow sign(\mathbf{w}^T x_i)$
- if $y'! = y_i$ then $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x_i}$
- until all training examples are correctly classified
- if data is linearly separable, the above algorithm always converges to a weight vector that separates the data
- if the data is not separable, algorithm does not converge, need to choose manually

Generative and Discriminative Models

- We did something different than with Naive Bayes
- Naive Bayes modelled how a class generated the feature vector p(x|y) - **generative approach**
  - Good with missing data
  - Good with detecting outliers
- Logistic regression models p(x|y) directly - **discriminative approach**
  - No need to waste time modelling

When generative classifiers are linear

1. Gaussian data with equal covariance
2. Binary data where each component is a Bernoulli variable

Multi-class classification

- Create a different weight vector $\mathbf{w_k}$ for each class
- Then use softmax function

$$p(y=k|\mathbf{x}) = \frac{exp(\mathbf{w_k}^T\mathbf{x})}{\sum\limits_{j=1}^{C} exp(\mathbf{w_j}^T\mathbf{x})}$$

Note that $0 \leq p(y=k|\mathbf{x}) \leq 1$ and sum is equal to 1.