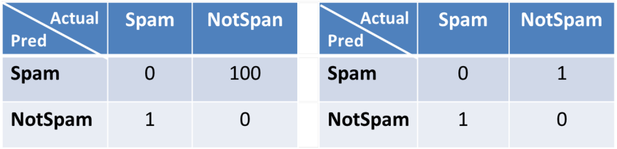
Evaluation

Machine Learning

Loss Functions I



Actual Pred	Spam	NotSpan
Spam	0	100
NotSpam	1	0

Actual Pred	Spam	NotSpam
Spam	0	1
NotSpam	1	0

- A *loss function* is a function $L(y, \hat{y})$ that defines the entries of the loss matrix
- The general framework for decision theory is this: (a) there is a true state s; (b) there are data D; (c) there is a decision rule δ that results in some action $a = \delta(D)$. Then, the loss function is $L(s, a)$. Sometimes, this is also called a *cost function* and written as $C(a|s)$ or $\lambda(a|s)$
  - For classification problems, for example, s is the true class label of some observation x and a the class label we decide for x

Loss Functions II

  - For problems where the values of D and s are only known probabilistically (that is, they are values of random variables), then we can only deal with the *expected loss* $E[L(S, A)]$, obtained by averaging over all possible values of the random variables S and A (for states and actions)
- The loss function for the loss matrix on the right is called a $0 - 1$ loss function
  - This is sometimes written as $L(y, \hat{y}) = I(y \neq \hat{y})$
- Another commonly used loss function in machine learning is the *squared loss*
  - This is $L(y, \hat{y}) = (y - \hat{y})^2$
- Two kinds of problems:
  1. Given $(x_1, y_1), \dots, (x_n, y_n)$ and an observation (x, y) predict $\hat{y}$ to minimise $L(y, \hat{y})$
     - BUT: how can we do this, since we do not know y

Loss Functions III

2. Given $(x_1, y_1), \ldots, (x_n, y_n)$, chose $f(x)$ to minimise $L(y, f(x))$ when (x, y) are drawn from some probability distribution (this is the *generalisation error*)
    - BUT: how can we do this, since we do not know x or y

- We will have to turn to probability theory to deal with these problems:
    - Treat x's and y's as instances of random variables X and Y
    - Minimise expected loss
- Let us look at the first kind of problem

Minimising Conditional Expected Loss I

- What is the expected loss when we are given an x. That is, $X = x$? For discrete values:

$$E[L(Y, \hat{y})|X = x] \; = \; \sum_{y} L(y, \hat{y}) P(y|x)$$

  This is called minimising conditional expected loss, or *conditional risk minimisation*. If we know, or can estimate $P(Y|X)$, and are given a loss function, then the conditional expected loss can be calculated

- For example, using a $0-1$ loss function:

$$E[L(Y, \hat{y})|X = x] \; = \; \sum_{y} I(y \neq \hat{y}) P(y|x) \; = \; \sum_{y \neq \hat{y}} P(y|x)$$

  which is:

$$E[L(Y, \hat{y}|x] \; = \; 1 - P(\hat{y}|x)$$

Minimising Conditional Expected Loss II

- So, to find $\hat{y}$ to minimise $E(.)$, maximise $P(\hat{y}|x)$
- IMPORTANT: with a 0-1 loss function, to minimise expected loss for a given x, we need $P(Y|X)$
- What about other loss functions? For a square-loss function:

$$E[L(Y, \hat{y}|x] \;=\; \int_{-\infty}^{\infty} L(y, \hat{y}) p(y|x) dy$$

Or:

$$E[L(Y, \hat{y}|x] \;=\; \int_{-\infty}^{\infty} (y - \hat{y})^2 p(y|x) dy$$

We will have to assume that $p(y|x)$ is smooth, we want to find the value of $\hat{y}$ s.t.

$$\frac{\partial E[L(Y, \hat{y}|x]}{\partial \hat{y}} \;=\; 0$$

Minimising Conditional Expected Loss III

Some straightforward manipulation leads to the result that the value of $\hat{y}$ that minimises $E[L(Y, \hat{y}|x)]$ is:

$$\hat{y} \; = \; E(Y|X = x)$$

In simpler terms: *just find the average of all the Y values that result from a particular value of x*

Generalisation Error I

- In general, we want to select a model m to minimise:

  $$e_m = E[L(Y, \hat{Y})] = E[L(Y, f_m(X))] = \sum_{x,y} L(y, f_m(x))P(x, y)$$

  This is called the *generalisation error* of f_m
- ACTUALLY: this is just the error we get given a specific training set $d = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. So, $e_m = E[L(Y, f_m(X))]$ above is actually:

  $$e_{m,d} = E[L(Y, f_{m,d}(X))|d] = \sum_{x,y} L(y, f_{m,d}(x))P(x, y|d)$$

Generalisation Error II

- There is also randomness arising from the choice of the training set. To account for that, take d as some value of some random variable D, and $e_{m,d} = E[L(Y, f_{m,d}(X))|D = d]$. Then we have to consider the the *expected generalisation error* over all possible values of D

$$\epsilon_m = E[L(Y, \hat{Y})] = E[e_{m,d}]$$

- Calculating $e_{m,d}$ (the generalisation error of a model constructed given data d) is not straightforward, but there are ways to estimate is expected value ϵ_m

Training Error I

- We want to minimise the generalisation error $e_{m,d} = E[L(Y, f_{m,d}(X))]$ over the true distribution (X, Y) This is not straightforward since: (a) the probability distribution (X, Y) is not known; (b) There will usually be parameters θ in the model: we will either have to average over all values of θ (hard), using the distribution of θ (hard); or choose a specific point estimate (approximate)
- SO: it is somewhat easier to estimate the *expected* generalisation $\epsilon_m = E[e_{m,d}]$,
- We can attempt to use a proxy for this: the *training error*. Given training data $d = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and a loss function L, the training error for a model m is:

$$e'_{m,d} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_{m,d}(x_i))$$

Training Error II

- The difficulty is that this estimate ends being an *optimistic* estimate
- For both 0-1 loss and squared-loss it can be shown that the expected value of optimism (for the simpler case where the expectation is over data sets with the same x values as d) is

$$\omega_m = \frac{2}{N} \sum_{i=1}^{N} cov(\hat{y}_i, y_i)$$

  where $\hat{y}_i$ is the value predicted by the model m for $X = x_i$
- Since the purpose of the model m is to approximate the y_i, we would expect $cov(\hat{y}_i, y_i)$ to be *positive*
- That is, ω_m will be positive, and so the training error will be an optimistic estimate of the generalisation error. In fact, as we start to fit the y_i values closer, ω_m increases

Estimating Generalisation Error I

- Given a training set d and a model m, we ideally want to be able to calculate $e_{m,d} = E[L(Y, f_{m,d}(X))|d]$,
- BUT: this is hard, often not possible, so we will have to settle for an estimate
  - The easiest is to get a point estimate, by drawing an independent sample of (x, y) pairs and estimate $e_{m,d}$
  - BUT: it may not be feasible to get new data
  - WHY NOT: use some of the training data d as a "test set"?
- If it is not possible to get new data, then the best we can do is to use some part of d to estimate generalisation error
  - BUT: this is not the same as estimating $e_{m,d}$ since models are no longer selected using the same d
  - However, this will allow us to get an estimate of the expected generalisation error ϵ_m, since that is averaged over datasets

Estimating Generalisation Error II

- Recall: $\epsilon_m = E[L(Y, f_m(X))]$, the expected generalisation error or the error of $f_m(X)$ is used to predict values on a samples drawn from $P(X, Y)$
  - The only data we have are d. But d is also drawn from $P(X, Y)$, so: randomly split d into a smaller part (v) called the *validation set*, and use the rest (d') as the actual training set
  - Use d' to obtain $f_{m,d'}$, and use the error of m on v to select the best model m^*
  - Construct $f_{m^*,d}$
- *Selection by validation*: the error of m on v is an estimate of ϵ_m
- BUT: there was a random choice involved in selecting v. What if the selected model m^* just happened to get lucky?
  - Randomly repeat K times: Randomly partition data into training d' and validation v; Construct a model on d' calculate the the loss on v. Return average loss.

Estimating Generalisation Error III

  - K-Partitions: Shuffle the d; partition into K subsets; use each subset as a validation set and the rest as training set d'
- *Selection by cross-validation*: K-fold cross-validation
  - If $K = N$, then this is known as *leave-one-out* cross-validation

Reporting performance (Classification) I

- Confusion matrix (on “test” data)

		Actual		
		Pos	Neg	
Predicted	Pos	n_1 (e_1)	n_2 (e_2)	n_a
	Neg	n_3 (e_3)	n_4 (e_4)	n_b
		n_c	n_d	N

n_1 are called the *true positives*; n_2 are called the *false positives*; n_3 are called the *false negatives*; and n_4 are called the true negatives.
A number of other performance measure can be computed from this table (*precision*, *recall*, *wt. relative accuracy*, etc.)

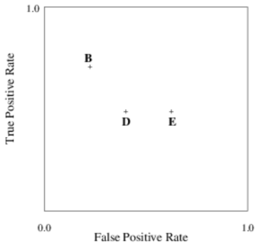
Reporting performance (Classification) II

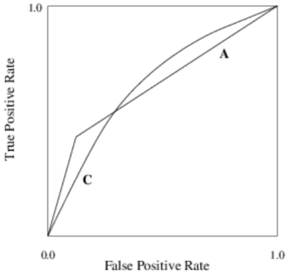
The e_i are the expected values of for the corresponding cell, under the hypothesis that the actual class is independent of the predicted one.

$$e_1 = \frac{n_a \times n_c}{N}$$

(and so on).

- ROC curves





Reporting performance (Classification) III

The axes are unbiased estimates of the $P(+|-)$ and $P(+|+) = 1 - P(-|+)$. These can be obtained from the contingency table, if the numbers are from a new data drawn from the population.

Comparing Performance (Classification) I

- Is there any association between predicted and actual performance? The probability that there is (isn't) an association can be calculated, using:

$$x^2 = \sum_{i=1}^{4} \frac{(n_i - e_i)^2}{e_i}$$

If there is no association between the Actual and Predicted values, then x^2 will be distributed according to the χ^2 distribution with some parameter called *degrees of freedom* (for contingency tables, the d.f is $(Nrows - 1) \times (Ncols - 1)$. So, we can ask, what is the probability of the calculated x^2 value coming from such a distribution. If the probability is high, then there is probably no association. The meanings of "high" and "low" depend on the *level of significance*

Comparing Performance (Classification) II

A high x^2 value indicates a low probability of a chance association between predicted and actual values. In this study, we have stipulated that this probability is no more than 0.05. That is, χ^2 values must be at least 3.84.
If there is an association between Predicted and Actual values, then The predictive accuracy for the model is estimated as: as $p = (n_1 + n_4)/N$. For 2-class problems, this is just the estimation of proportion of success in a binomial population. The error in this estimate is $\pm\sqrt{pq/N}$ where $q = 1 - p$.

- The performance of a pair of algorithms on a problem can be done by a cross-comparison of the test instances correctly and incorrectly classified:

Comparing Performance (Classification) III

$$\begin{array}{cc|c|c|c}
 & & \multicolumn{2}{c}{\text{Predicted } (A_1)} & \\
 & & \text{Correct} & \text{Incorrect} & \\
\hline
\text{Predicted } (A_2) & \text{Correct} & n_1 & n_2 & n_a \\
\hline
 & \text{Incorrect} & n_3 & n_4 & n_b \\
\hline
 & & n_c & n_d & \text{N}
\end{array}$$

The null hypothesis is that the proportions of examples correctly classified by both algorithms is the same. If there is no significant difference in the performance of the two algorithms, half of the $n_2 + n_3$ cases whose classifications disagree should be classified correctly by A_1 and A_2.

We can again use the χ^2 test, this time with observed values n_2, n_3, and corresponding expected values of $\frac{n_2+n_3}{2}$

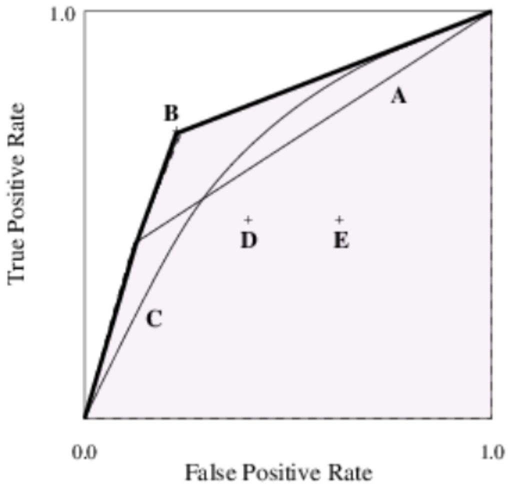
Comparing Performance (Classification) IV

- A pair of algorithms can be compared across a range of problems by a simple *sign* test: if the algorithms perform similarly then in about 50% of the problems, A_1 will perform better ($=$ *eads*) than A_2. We can either calculate the binomial probability of observing the number of *heads* under the assumption that the expected probability is 0.5
  - A variation of the sign-test, is the *Wilcoxon* sign-rank test, which takes into account not just how often A_1 is better (or worse), but by how much. Differences between the performance of A_1 and A_2 are computed and ranked, with a sign ($+$ or $-$) to indicate better or worse.
  - If there is no difference between A_1 and A_2, then the sum of signed ranks should be close to 0
  - This test is related to the Area Under The ROC curve (AUROC)

Comparing Performance (Classification) V

- When comparing multiple algorithms on multiple problems, the probability of A_i looking better than A_j just by chance increases. There is a way to correct for this (called the *Bonferroni* adjustment)

ROC Curves and Optimal Classifiers I



ROC Curves and Optimal Classifiers II

- Optimal classifiers lie along the edge of the convex-hull – whatever the current $\pi(\cdot)$ or $C(\cdot|\cdot)$ may be
- Classifers **D** and **E** can never be optimal
- Classifiers **A**, **B**, **C** are best for different contexts ($\pi(\cdot)$ and $C(\cdot|\cdot)$)
- Once context is known, a choice can be made amongst them

ROC Curves and Optimal Classifiers III

- Can a single classifier be optimal for all misclassification costs?
  - YES: for any given problem, it is possible that whatever the context, the ROC curve of a classifier is on the convex hull
- Can a single classifier be optimal for all classification problems
  - NO: it is not possible for a classifier to be on the convex hull of ROC curves of all possible problems for all possible contexts

The No-Free-Lunch Theorem

Let A be any learner that for the binary classification problem of identifying a function $\mathcal{X} \to \{0, 1\}$, with 0-1 loss. Let S be a training set of size $m < |\mathcal{X}|/2$. Then, there exists a distribution $\mathcal{D}$ over $\mathcal{X} \times \{0, 1\}$ s.t:

1. There is a function $f : \mathcal{X} \to \{0, 1\}$ s.t. $L(y, f(x)) = 0$; and
2. With probability $\geq 1/7$ the $L(y, A(S))] \geq 1/8$

That is, for every learner, there is a task for which it id very likely to fail, even though there exists another learner which succeeds

BUT: To know more, you will need to take a course on Statistical Learning Theory

Course Take-Aways

1. What is the problem to be solved?
2. Representation matters
3. ML is not the first thing to do
4. Data are samples
5. Correlation isn't causation
6. Bayes $=$ Likelihood $\oplus$ Priors
7. Models $=$ Structure $\oplus$ Parameters
8. Models aren't perfect
9. Empirical relationships aint laws
10. ML $=$ Probability $\oplus$ Statistics $\oplus$ Decision Theory $\oplus$ Computation