Introduction

Definitions of Machine Learning

Definition
The automatic extraction of valid, novel, useful and comprehensible knowledge (rules, regularities, patterns, constraints, models etc) from arbitrary sets of data.

What are we learning?
We are learning how to create a **task** that accomplishes a goal. For example:

- Assigning continuous values to inputs (essay → grade)
- Grouping inputs into known classes (email → spam, no spam)
- Understand regularities in data

What are we learning from?
We use **data** as a basis for machine learning. There are a few considerations that we need to make in regards to the quality of the data, for example is it reliable or representative of a population? If we are taking a survey on the students, then if we only survey domestic students we would have data biased towards domestic students.

How do we learn?
We can define a **model** that explains how to get from input to output.
We can derive a **learning algorithm** to find the best model parameters.

How do we know the quality of the output?
If the algorithm improves at its task with exposure to more data, then we know that some form of learning is happening. We do however need to be able to evaluate its performance objectively (Eg. we don't want a chat bot to learn racist terms).

Ingredients for machine learning

Data

- Discrete vs continuous vs ....
- Big data vs small data
- Labelled data vs unlabelled data
- Public vs sensitive data

Models

- Function mapping from inputs to outputs
- Motivated by data generating a hypothesis
- Probabilistic machine learning models
- Geometric machine learning models
- Parameters of the functions are unknown

Learning

- Improving (on a task) after data is taken into account
- Finding the best model parameters (for a given task)
- Supervised vs unsupervised learning

Examples of Machine Learning

Clustering

A type of unsupervised learning, this is a technique that groups similar entities or data together. It can be used to find similarities or patterns in data (creating its own grouping or classifications) as well as identifying any outliers.

We don't know class labels beforehand, and use this method to separate data into groups.

Example

An archaeologist is in charge of classifying a mountain of bones, and wants to quickly identify any 'finds of the century' before sending the bones off to a museum.

Solution

Identify any bones which are of different size/dimensions/characteristics to others in the sample and/or pre-identified bones.

Classification

An example of supervised learning that provides labelled data to a machine learning algorithm, which then bases its output or classification of future data into groups using that input data. It can be used to process unsorted data into pre-defined groups. The groups are finite/discrete.

We know class labels prior to training, and the goal of this technique is to predict a class label.

Example

An archaeologist is in charge of classifying a mountain of bones, and wants to come up with a consistent way of determining the species and type of each bone which doesn't require specialist skills.

Solution

Identify some easily measurable properties, such as size or shape, and compare them to a pre-classified database of bones. For example, humans have a very specific skull shape, so future bones which look similar could be classified as human.

Regression

A technique that uses mathematical methods to produce a continuous outcome (y) based on the values of the predictor variables (x). An extremely common example is linear regression $y = mx + c$. Used with numeric data and the output is continuous.

Similar to Classification, however done with continuous (and numeric) values. Also an example of supervised learning.

Example

A person in charge of developing the next release of Coca Cola wants to be able to estimate how well a given recipe will be.

Solution

Carry out taste tests over various recipes with varying proportions of sugar, caramel, caffeine, and other variables/ingredients, then estimate the function which predicts customer satisfaction from these numbers. We can then use this equation to determine the optimal proportion of these ingredients that will maximise customer satisfaction.

Basics of Machine Learning

Machine Learning Workflow

1. **Business Understanding**
   What is the purpose and scope of the problem? What is the issue we want to solve, or what are we trying to predict?
2. **Data Understanding**
   What type of data do we have? Is the data clean? How should we be using the data in our machine learning model?
3. **Data Preparation**
   Cleaning the data if it is noisy, or converting the data into the correct format (eg. discrete to continuous)
4. **Modelling**
   Training the model using the training data
5. **Evaluation**
   Testing the model using the test data, and evaluating how well it performs. If it doesn't produce the right outcome, then we might have to go back to step 1, and restart the process with a better understanding of business needs or data training. This cycle can be repeated several times.
6. **Deployment**
   If we are happy with the outcome of the model, and are confident that we can use it in the real world, then we can deploy the model.

Terminology

Instances
The individual, independent examples of a concept, also known as exemplars.

In other words, these are the inputs to the machine learning models that we use the train the model with. For example, to test if an email is spam or not, we can provide an instance of an email, with the message, sender ... etc, as mark if that particular instance is spam or not.

Attributes
Measures aspects of an instance, also known as features or attributes.

Attributes are used to help quantify the value of an instance. For example, if we have an email instance, then the attributes would be the message, grammar and sender. To determine if the message is spam or not, we can use these features to predict an output. An email with a lot of grammar mistakes will be more likely to be spam than one without.

Concepts
The purpose of the machine learning model; or the thing that we're aiming to learn to predict. Generally, the output of a machine learning model comes in the form of a label or classes.

The outcome of the model. Eg. Is this email spam or not spam? Yes would be an outcome. This can also come in the form of a *special attribute* where a feature/label is also the concept we want.

- Predicting a discrete class (**Classification**)
- Grouping similar instances into clusters (**Clustering**)
- Predicting a numeric quantity (**Regression**)
- Detecting associations between attribute values (**Association Learning**)

Supervision

Supervised Learning
Supervised methods have prior knowledge of a closed set of classes, and set out to discover and categorise new instances according to these classes.

Unsupervised Learning
Unsupervised methods do not have access to an inventory of classes, and instead discover groups of *similar* examples in a given dataset.

There are two types of unsupervised learning.

- **Strong**
  Dynamically discovers the 'classes', which are implicitly derived from the grouping of instances, during the process of categorising the instances. No prior knowledge of classes whatsoever.
- **Weak**
  Categorises instances as certain labels without the aid of pre-classified data. The model doesn't what the class labels are exactly, but might know to have 10 groups to classify data into.

Classification
Assigns an instance a discreet class label.

- Classification learning is *supervised*
- Scheme is provided with an actual outcome or *class*
- The learning algorithm is provided with a set of classified *training data*
- Can use *test data* to measure success. Can compare the known class labels with the label from the ML model.

Clustering
Finds groups of items that are similar.

- Clustering is *unsupervised*, the learner operates without a set of labelled training data
- The class of an example is not known, or not given to the learning algorithm
- Success is measured subjectively, and there's no definite method to measure success, which can be problematic.

Regression
Classification learning, however the class is continuous, and regression produces a numeric prediction.

- Learning is *supervised*
- Unlike classification, infinite labels are possible, so a correct prediction is when the numeric value is acceptable close to the true value.

Instances

Instances can be characterised as 'feature vectors', and defined by a predetermined set of attributes.

Input to learning scheme: dataset

- Flat file representation (can be represented as a vector)
- No relationships between objects - this may or may not be true
- No explicit relationship between attributes

Attribute data types:

- Discrete: nominal, categorical or ordinal
- Continuous: numeric

Nominal Quantities

- Values are distinct symbols eg. {`sunny, overcast, rainy`}
- Values should only be used as a label or name
- Also called categorical or discrete
- Special case: dichotomy ('Boolean' attribute {`0,1`})
- No relation is implied among nominal values (no ordering or distance measure), and only equality tests can be performed

Ordinal Quantity

- Numeric quantities are real-valued attributes
- Scalar(a single number) has the attribute `distance`
- Vector-valued (a vector of numbers which each pertains to a feature or feature value) has the attribute `position` which is a set of coordinates (x, y)
- All mathematical operations are allowed (such as addition, subtraction and scalar multiplication)

Conversion of attribute types

Some machine learning models assume a certain type of attribute. For example if we have linear regression, we can only have numeric inputs. To accommodate for these models, we have several methods to ensure that the input is always compatible with the model we want to use.

- Select only attributes with the correct type - discard attributes which do not
- Change the model assumptions to match the data - use a different ML model
- Change the attributes to match the model (most common)

Converting Nominal to Numeric Attributes

Mapping category names to numbers
`{red, blue, green, yellow}` $\rightarrow$ `{0,1,2,3}`

Problems: Creates an artificial ordering, we're saying that red is closer to blue than yellow. This can be problematic with a large number of categories as well.

One-hot Encoding
```
'red' = [1,0,0,0]
'blue' = [0,1,0,0]
'green' = [0,0,1,0]
'yellow' = [0,0,0,1]
```

Overall, better method of encoding categorical attributes in a numeric way

Problems: Increases the dimensionality of the feature space, which can be an issue if we have a large number of dimensions. Distances between each feature will then be too small and become meaningless.

Numeric Feature Normalization

Features of vastly different scales can be problematic. Some machine learning models assume features to follow a normal distribution, and some learning algorithms are overpowered by large feature values and can as a result, ignore smaller values. (eg. size of a house is 500sqm but prices can be 1M - the price of the house could have a bigger weighting due to the scale of the numbers)

Feature **standardization** rescales features to be distributed at around a 0 mean with a unit standard deviation.

$$x' = \frac{x - \mu}{\sigma}$$

Feature **scaling** rescales features to a given range. For example *min-max scaling* rescales values to be between 0 and 1, using the minimum and maximum feature value observed in the data.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Converting Numeric to Nominal Attributes

Distcretization
Grouping numeric values into a pre-define set of distinct categories.
Eg. housing prices $\rightarrow$ `{high, medium, low}`

To do this, we need to decide on the number of categories and the category boundaries. There are a few options for this.

Equal Widths Discretisation
Finds the minimum and maximum of the data, the partitions the values into n bins of width $\frac{\text{max - min}}{n}$ bins.

Problems: Outliers would be grouped together on both sides (have entire bins of outliers). Bins wouldn't have an even distribution of items, and we would have to choose how to select n.

Equal Frequency Discretisation
Sort the values, then partition them into n bins such that each bin has an identical number of items.

Problems: Boundaries could be hard to interpret, and we would also have to choose how to select n.

Clustering
Uses unsupervised machine learning to group the value to into n clusters. Eg. K-means clustering.

Problems: How would we evaluate the result, and how do we select K (number of groups)?

Preparing Input

A few considerations when preparing data for input into a ML model, is missing attributes or inter-dependent attributes. If a piece of data is missing, there could be a variety of reasons from no being able to record the measurement, changes in experimental design etc, which are random. However if data is missing on purpose, then we might also have to find a way to encode that into a data eg. using a `missing` value in our class.

Another consideration we have to make is getting to know the data or visualising it before we decide on the data and ML model to use. For example, we can use histograms to check if the distribution is consistent with our background knowledge (i.e if we're expecting a normal distribution but get a different one), or scatter plots to see if there is any obvious outliers we can remove before putting the data in. We can also check for dependencies between features (such as age and D.O.B) using 2-D/3-D plots. We can also consult domain experts if necessary.

K-Nearest Neighbours

Algorithm

Training

- Store all training examples

Testing

- Compute *distance* of test instance to all training data points
- Find the K closest training data points
- Compute *target concept* of the test instance based on labels of the training instances. By target concept, it could be class label (most common label among neighbours) or a number (average among neighbours)

Feature Vector

For a dataset, each instance can be represented as a feature vector.

$$\texttt{feature vector} = \begin{bmatrix} \texttt{Outlook} \\ \texttt{Temperature} \\ \texttt{Humidity} \\ \texttt{Windy} \end{bmatrix}$$

Comparing Nominal Feature Vectors

Example dataset:

instance	red	yellow	round	sweet	curved	small
apple	1	0	1	1	0	?
banana	0	1	0	1	1	?
cherry	1	0	1	1	0	1

Hamming Distance

The number of differing elements in two 'strings' of equal length.

Eg. For the above dataset: The hamming distance $d(apple, banana) = 4$ as they have 4 different features: red, yellow, round and curved.

Simple Matching Distance

The number of matching features divided by the number of all features in the sample.

$$d = 1 - \frac{k}{m}$$

Where d is distance, k is number of matching features, m is the total number of features.
For the example dataset, $d(apple, banana) = 1 - \frac{2}{6} = \frac{4}{6}$. There are a total of 6 features, 2 of which match.

Jaccard Distance

The Jaccard distances is 1 - J where J is the Jaccard similarity. The Jaccard similarity J is the intersection of two sets divided by their union.

$$d = 1 - J = 1 - \frac{|A \cap B|}{|A \cup B|} = 1 - \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Eg. $d(apple, banana) = 1 - \frac{1}{5} = \frac{4}{5}$. Note that only features with `1` can be included in sets.

Comparing Numerical Feature Vectors

Manhattan Distance

Also known as L1 distance, the distance is the sum of absolute differences of each feature.

$$d(a,b) = \sum_{i=1}^{m} |a_i - b_i|$$

Eg. For two instances $a = [2.0, 1.4, 4.6, 5.5]$, $b = [1.0, 2.4, 6.6, 2.5]$

$$d(a,b) = |2.0 - 1.0| + |1.4 - 2.4| + |4.6 - 6.6| + |5.5 - 2.5| = 1 + 1 + 2 + 3 = 7$$

Euclidean Distance

Also known as L2 distance, the distance is the squared root of the sum of squared differences of each feature.

$$d(a,b) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}$$

$$d(a,b) = \sqrt{(2.0 - 1.0)^2 + (1.4 - 2.4)^2 + (4.6 - 6.6)^2 + (5.5 - 2.5)^2} = \sqrt{15} = 3.87$$

Cosine Distance

One minus cosine similarity, which is the cosine of the angle between two vectors (or inner product of the normalized vectors)

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

$$d(a,b) = 1 - cos(a,b)$$

Cosine distance is normalized by the magnitude of both feature vectors, so we can compare instances of different magnitude.

feature	doc1	doc2	doc3
word1	200	300	50
word2	300	200	40
word3	200	100	25

$cos(doc1, doc2) = \frac{200x300+300x200+200x100}{\sqrt{200^2+300^2+200^2}\sqrt{300^2+200^2+100^2}} = 0.93$
$d(doc1, doc2) = 1 - 0.93 = 0.07$

Comparing Ordinal Feature Vectors

Normalized Ranks

- Sort values, and return a rank $r \in \{0...m\}$
- Maps ranks to evenly spaced values between 0 and 1 using $z = \frac{r}{m}$
- Compute a distance function for numeric features (eg. Euclidean)

Eg. For ratings `-2,-1,0,1,2` we can map this to `0,1,2,3,4` then do our usual distance calculations.

Deciding on neighbours

Majority Voting
Each neighbour within the range is given an equal weight, or vote. $w_1 = ... = w_k = 1$

Inverse Distance
Weight is assigned based on distance

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

Where $\epsilon \approx 0$, and d_j is distance.

Inverse Linear Distance
Weight is assigned based on distance

$$w_j = \frac{d_k - d_j}{d_k - d_1}$$

Where d_1 is the min d among neighbours, d_k is the max d among neighbours and d_j is the distance of the jth neighbour.

Selecting the value of K

Small K

- Jagged decision boundary
- Captures noise
- Lower classifier performance

Large K

- Smooth decision boundary
- Danger of grouping together unrelated classes
- Lower classifier performance

Breaking Ties

- Avoid an even K
- Random tie breaking
- Change distance metric
- Pick class with highest prior probability

Probability

Conditional Probability
The probability of A given that we know B is denoted as $P(A|B) = \frac{P(A \cap B)}{P(B)}$

Independence
A and B are independent iff $P(A \cap B) = P(A)P(B)$

Disjoint Events
The probability of two disjoint events such that $A \cap B = \emptyset$ is $P(A \text{ or } B) = P(A) + P(B)$

Product Rule
$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$

Chain Rule
Note we can also choose the order of factorisation so that it makes sense. Eg. Pr(Rain|July) vs Pr(July|Rain), the first one makes more sense.
$P(A_1 \cap ... \cap A_n) = P(A_1)P(A_2|A_1)P(A_3|A_2 \cap A_1)...P(A_n| \cap_{i=1}^{n-1} A_i)$

Bayes Rule
Allows us to manipulate the order of the conditional.
$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$

Posterior Probability $P(A|B)$
The degree of belief having accounted for B.

Prior Probability $P(A)$
The initial degree of belief in A, or the probability of A occurring given no additional knowledge about A.

Likelihood $P(B|A)$
The support B provides for A.

Normalising constant ('Evidence') $P(B)$
$P(B) = \sum_A P(B|A)P(A)$

Binomial Distribution
A series of independent trials with only two outcomes (Bernoulli Trials) creates a binomial distribution.

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

Where p = probability of success, m = num successes we want, n = total trials

Multinomial Distribution
Models the probability of counts of different events from a series of independent trials with more than two possible outcomes. Eg. Probability of observing 5 threes when rolling a dice 5 times.

$$P(X_1 = x_1 ... X_n = x_n; p) = \frac{(\sum_i x_i)!}{x_1! ... x_n!} \prod_i p_i^{x_i}$$

Where X_1, X_2, X_3 = events, $p = p_1, p_2 ... p_n$ = probabilities, $x_1, x_2 ... x_n$ is the occurrences.

Categorical Distribution

The probability of events resulting from a single trial with more than two possible outcomes. Eg. Rolling a dice once and observing a 5.

$$P(X_1 = x_1 ... X_n = x_n; p) = \prod_i p_i^{x_i}$$

Marginalization

Summing all the possible outcomes of features/events we don't care about to obtain the probability of an event occurring.

$$P(A) = \sum_{b \in \beta} P(A, B = b)$$

$$P(A) = \sum_{b \in \beta} P(A|B = b)P(B = b)$$

$$P(A|C) = \sum_{b \in \beta} P(A|C, B = b)P(B = b|C)$$

Maximum Likelihood Estimate (MLE)

Picking a value of θ that would maximise the probability of the observed data. Eg. If we have 100 emails, 20 of which are spam, the value of $\theta = 0.2$ would maximise the observed data.

$$\hat{\theta} = \text{argmax P}(X; \theta; N)$$

We can then use this value to predict values for unseen data. Eg. Predicting likelihood from a binomial distribution.

$$\mathcal{L}(\theta) = P(X; \theta; N) = \binom{n}{m} \theta^x (1 - \theta)^{N-x}$$

When simplifying the likelihood, we can $\propto$ make it proportional and remove constants that are independent from θ

Prior Belief

An alternative to choosing an estimator, used when we don't have observational data, but we believe that around 80% of emails are not spam. Therefore we can set $\theta = 0.8$ where θ is the probability of an email not being spam.

$$\hat{\theta} = \text{argmax P}(\theta) P(x|\theta)$$

We can use this to get the **posterior probability distribution** of θ

$$P(\theta|x) = \frac{P(\theta)P(x|\theta)}{P(x)} \propto P(\theta)P(x|\theta)$$

Optimization

Machine Learning

Involves building models, and finding model parameters that optimize some measure of performance.

Objective Functions

To find parameter values θ that maximize or minimize the value of a function $f(\theta)$. In other words, we want the extreme points of the objective function.

For a maximum this is:

$$\hat{\theta} = \text{argmax} f(\theta)$$

For a minimum (where f is called a **loss function**)

$$\hat{\theta} = \text{argmin} f(\theta)$$

At its extreme point, $f(\theta)$ is flat, so its slope (derivative) is equal to zero. *This can also be a point on inflection.

$$\frac{\delta f}{\delta \theta} = 0$$

Eg. For a function $f(\theta)$, we can solve for $\text{argmax} f(\theta)$

$$\begin{aligned} f(\theta) &= \theta^2 \\ \frac{\delta f}{\delta \theta} &= 2\theta \\ 2\theta &= 0 \\ \theta &= 0 \end{aligned}$$

Finding Minima/Maxima

1. Define your function of interest $f(\theta)$
2. Compute its first derivative with respect to its input θ
3. Set the derivative equal to zero
4. Solve for θ

Eg. For a data set of emails, where we want to identify each email x as `spam, not spam`. We have N observations, each with two possible outcomes, and data that follows a binomial distribution. The objective function can be modelled as:

$$\mathcal{L}(\theta) = p(X; N, \theta) = \frac{N!}{x!(N-x)!}\theta^x(1-\theta)^{N-x}$$

Where the parameter $\theta = \text{P(spam)}$.
If we have a set of 100 emails, and 20 are `spam`, then intuitively, $P(spam) = \theta = \frac{20}{100} = \frac{x}{N}$. We can also prove this as follows.

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{N!}{x!(N-x)!}\theta^x(1-\theta)^{N-x} \\ &\propto \theta^x(1-\theta)^{N-x} \\ log(\mathcal{L}(\theta)) &= log(\theta^x(1-\theta)^{N-x}) \\ &= log(\theta^x) + log((1-\theta)^{N-x}) \\ &= xlog(\theta) + (N-x)log(1-\theta) \end{aligned}$$

Finding the derivative:

$$\begin{aligned}\frac{\delta log\mathcal{L}(\theta)}{\delta\theta} &= \frac{x}{\theta} + (N-x)\frac{(-1)}{1-\theta} \\ &= \frac{x}{\theta} - \frac{N-x}{1-\theta}\end{aligned}$$

Maximising this equation (set it to zero)

$$\begin{aligned}\frac{x}{\theta} - \frac{N-x}{1-\theta} &= 0 \\ x(1-\theta) - (N-x)(\theta) &= 0 \\ x - x\theta - N\theta + x\theta &= 0 \\ x - N\theta &= 0 \\ N\theta &= x \\ \hat{\theta} &= \frac{x}{N}\end{aligned}$$

Constrained optimization
When the parameters that we want to learn has to obey constraints.

$$\text{argmin} f(\theta) \text{ subject to } \text{g}(\theta) = 0$$

Lagrange multipliers
Allows us to incorporate equality constraints into our optimization, denoted by λ. We essentially introduce another variable multiplied by λ to represent our constraints.

$$\mathcal{L}(\theta, \lambda) = f(\theta) - \lambda g(\theta)$$

Eg. Find an optimal parameter vector θ such that each all θ_i sum up to a certain constant b.

Formalizing the constraint:

$$\sum_i \theta_i = b$$

Set the constraint to zero

$$\sum_i \theta_i - b = 0$$

Set the constraint and write the Lagrangian:

$$\begin{aligned}g_c(\theta) &= -b + \sum_i \theta_i \\ \mathcal{L}(\theta, \lambda) &= f(\theta) - \lambda g_c(\theta) \\ &= f(\theta) - \lambda(-b + \sum_i \theta_i)\end{aligned}$$

Then we can proceed as before, derive, set to zero and solve for θ.

Naive Bayes

Notation

label: y (eg. spam, play, ...)
observation: x (eg. email, day, ...)
features: $x_m, m \in \{1, 2, ..., M\}$ (eg. temperature, words, ...)
observation-label pair: (x^i, y^i)
parameters: $\theta, \phi, \psi, ...$
function: $f(x, y; \theta)$ function of x and y with parameters θ, also denoted $f_\theta(x, y)$

Probabilistic Model:
We can build a probabilistic model that encompasses all of the training data D^{train}. We learn from our model parameters θ such that they maximise the data log likelihood. And use the train model to predict the class labels of the test data.

$$P_\theta(x, y) = \prod_{i \in D^{train}} (x^i, y^i)$$

Based on the θ, we can also find

$$\hat{y} = \text{argmax}_{y \in Y} P(y|x)$$

The obvious method of doing this would require a lot of data, more specifically $O(|Y| \cdot k^m)$ instances, where Y = classes, m = num attributes, and k is the number of values those attributes can take. We would require every combination of features to accurately predict classes.

Instead we can use Bayes' rule for this.

Bayes' Rule:

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

Finding $\hat{y}$

$$\begin{aligned}\hat{y} &= \text{argmax}_{y \in Y} P(y|x) \\ &= \text{argmax}_{y \in Y} \frac{P(x|y)P(y)}{P(x)} \\ &\propto \text{argmax}_{y \in Y} P(x|y)P(y) \\ &= \text{argmax}_{y \in Y} P(x_1, x_2..., x_M|y)P(y)\end{aligned}$$

Eg. The resulting equation is still infeasible, so we can make a naive assumption that all features are conditionally independent. Occasionally, this is nonsense, but the model still works.

Naive Bayes:
Modelling the conditional independence assumption:

$$\begin{aligned}\text{argmax}_{y \in Y} P(x_1, x_2..., x_M|y)P(y) &\approx P(x_1|y)P(x_2|y)...P(x_M|y)P(y) \\ &= P(y) \prod_{m=1}^{M} P(x_m|y)\end{aligned}$$

Naive Bayes Classifier:

$$\hat{y} = argmax_{y \in Y} P(y) \prod_{m=1}^{M} P(x_m|y)$$

Probabilistic Model:

$$P(x, y) = \prod_{i=1}^{N} P(y^i) \prod_{m=1}^{M} P(x_m^i|y^i)$$

Eg. Naive Bayes with a binary class label, and real valued feature vectors of length M. Assuming y is drawn for Bernoulli distribution, and x_m is drawn from a Gaussian distribution.

$$\begin{aligned} p(x, y) = p_{\phi,\psi}(x_1, x_2, ..., x_m, y) = p_\phi(y) \prod_{m}^{M} \\ = BN(y|\phi) \prod_{m}^{M} N(x_k|\psi = \{\mu_{m,y}\sigma_{m,y}\}) \\ = \phi^y (1-\phi)^{(1-y)} \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi\sigma_{m,y}^2}} exp(-\frac{1}{2}\frac{(x_m - \mu_{m,y})^2}{\sigma_{m,y}^2}) \end{aligned}$$

Eg. Naive Bayes with binary classes and binary features. Assuming y is drawn for Bernoulli distribution, and x_m is drawn from a Bernoulli distribution.

$$\begin{aligned} p(x, y) = p_{\phi,\psi}(x_1, x_2, ..., x_m, y) = p_\phi(y) \prod_{m}^{M} \\ = BN(y|\phi) \prod_{m}^{M} BN(x_k|\phi_{m,y}) \\ = \phi^y (1-\phi)^{1-y} \prod_{m=1}^{M} (\phi_{y,m})^{x_m} (1-\phi_{y,m})^{(1-x_m)} \end{aligned}$$

Eg. Naive bays with two categorical distributions. y is drawn from a Categorical distribution with C classes, and x_m is drawn from a Categorical distribution over K classes.

$$\begin{aligned} p(x, y) = p_{\phi,\psi}(x_1, x_2, ..., x_m, y) = p_\phi(y) \prod_{m}^{M} \\ = Cat(y|\phi) \prod_{m}^{M} Cat(x_k|\phi_{m,y}) \\ = \phi_y \prod_{m=1}^{M} \prod_{k=1}^{K} (\phi_{y,m,k}) \end{aligned}$$

Categorical Naive Bayes:

Categorical distribution over class labels:
ϕ is found by observing the relative frequencies of classes in the training data.

$$\phi_y = \frac{\text{count}(y)}{N}$$

Categorical distribution over features given a class label:
ψ is found by observing the relative frequency of each class,label pair among all instances with that class.

$$\psi_{y,m} = \frac{\text{count}(y, m)}{\text{count}(y)}$$

Gaussian Naive Bayes:

Mean:
The average of all observed feature values for x_m under class y.

$$\mu_{y,m} = \frac{1}{\text{count}(y)} \sum_{i:y_i=y} x_m^i$$

Standard deviation:
Sum of squared differences of observed values from the mean. Normalized and square rooted.

$$\sigma_{y,m} = \sqrt{\frac{\sum_{i:y_i=y}(x_m^i - \mu_{y,m})^2}{\text{count}(y)}}$$

Smoothing

If we have any term that is unseen, or has probability = 0, then the product of all the probabilities will also be 0. There are various methods we can use to adjust the values of the model parameters to ensure that every pr is bigger than 0, but the sum of total probabilities is still equal to 1.

Epsilon Smoothing

Replace any 0 with a ϵ, which is a very small constant. The ϵ needs to be much smaller than $\frac{1}{N}$, which is the minimum probability of an instance.

Laplace Smoothing

Adds a pseudocount α to each feature observed during training.

$$P(x_m = j | y = k) = \frac{\alpha + \text{count}(y = k, x_m = j)}{M\alpha + \text{count}(y = k)}$$

Generally the value of α is 1, and all counts are incremented to maintain monotonicity. M is also the number of values x_m can take on.

Generally good for large amounts of data, however overall reduces variance and adds bias to the classifier. As a result our MLE may not be true.

Evaluation

Classification Evaluation

Input: A set of labelled training instances and a set of unlabelled test instances.
Model: An estimate of the underlying target function
Output: Prediction of the classes of the test instances.

Goals in Machine Learning

1. **Model Evaluation**
   Making predictions that are correct, and estimating the true performance of a model based on the errors it makes.
2. **Model Selection**
   Choosing the best model, which can be based on a number of reasons. Eg. Combination of algorithms, parameters etc that give the best error rate, or efficiency, understanding etc.

Evaluation Strategies

Brute Force Training and Testing

Build the model using all of the instances and evaluates the model using all of the instances. Very bad as we tend to over-estimate classifier performance, we essentially give the classifier all the answers, then tests it using the same questions. The classifier just memorises all of the answers, which gives it a high accuracy rate, which may perform poorly on unknown test sets.

Holdout Evaluation Strategy

Each instance is randomly assigned as either a training instance or a testing instance. The data is randomly partitioned so there is no overlapping instances between data set, and the split can be assigned according to probabilities eg. 50-50, 80-20, 90-10.

Advantages

- Simple to work with and implement
- Fairly high reproducability

Disadvantages

- Size of the split affects the estimate of the model's behaviour, while it is randomly assigned, it isn't certain that we will have an exact split for percentages (depending on implementation).
- Have a lot of test instances, and few training instances means the learner won't have enough information to build an accurate model.
- Lots of training instances, few test instances means the learner builds an accurate model but the test data might not be representative. (Estimates could be too high or too low)

Repeated Random Subsampling

Similar to holdout, but the process is iterated multiple times. A new training and test set are chosen each time, a new model is built, and is evaluated by averaging over the iterations.

Advantages

- Averaging holdout method tends to product more reliable results

Disadvantages

- More difficult to reproduce
- Slower than holdout
- Wrong choice of training set/test size can lead to misleading results (difficult to check as averaged)

Cross Validation

Data is progressively split into a number of partitions (m). Iteratively, one partition is used as test data, and all the other $m-1$ partitions are used as training data. This repeats until all the partitions are used as test data. The evaluation metric is then aggregated across all the models.

Advantages

- Every instance is a test instance for some partition. Evaluation metrics would be calculated with respect to a dataset that looks like the entire dataset.
- Takes roughly the same time as Repeated Random Subsampling
- Very reproducible
- Can be shown to minimize bias and variance of our estimates of the classifier's performance.

Deciding m

- Number of folds impacts the runtime and size of datasets// Fewer folds means more instances per partition, more variance in performance
  More folds means fewer instances per partition, less variance but slower to run
- Most common choice of m (how many partitions)
  $m = 10$ mimics a 90-10 holdout, and is more reliable
- Best choice: $m = N$, where N is the number of instances. This is also known as the leave one out cross validation method.
  It maximises training data for the model, and mimics actual testing behaviour. It is however, not practical to use due to the time it takes.

No Free Lunch Theorem

*A learner that makes no **a priori assumptions** regarding the identity of the target concept has no **rational basis** for classifying any unseen instances.*

Essentially, you can't get a good classifier without making assumptions.
Additionally, averaged cross all possible problems, the performance of any two algorithms is identical. If Algorithm A does well on p1, then it will perform worse on some p2.

Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large training data set will also approximate the target function over unseen test examples.
not always applicable eg. data in 1970s might not work in 2021* **Stratification

A typical inductive bias (assumption) in our evaluation framework. This assumes that the class distribution of unseen instances will be the same as the distribution of seen instances. I.e. that the training and test data both have the same class distribution as the dataset.

Evaluation Measures

Error E

$$E = \frac{1}{N}(1 - I(y_i, \hat{y}_i)) \text{ where } I(a,b) \begin{cases} 1, & \text{if } a == b \\ 0, & \text{otherwise} \end{cases}$$

Error Rate Reduction (ERR)

$$ERR = \frac{E_0 - E}{E_0}$$

Types of Errors

Contingency Tables

	1 ($\hat{y} = 1$)	0 $\hat{y} = 0$
1 ($y = 1$)	True Positive (TP)	False Negative(FN)
0 ($y = 0$)	False Positive (FP)	True Negative(TN)

Some errors might have a higher cost depending on the type. For example we might want to penalise false negative errors for cancer detection, as the consequences of that can be quite disastrous.

Accuracy

A basic evaluation metric. Equal to $1 - E$, or number of correctly labelled test instances over total number of test instances. Measures how frequently the classifier is correct.

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

Precision

How often are we correct, when we predict that an instance is interesting (1)? Out of all the 1s predicted, which ones are correct?

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

Recall

What proportion of the truly interesting instances have we correctly identified as interesting? Out of all the true 1s, which ones did we correctly predict?

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

The relationship between Precision and Recall are inverse. We can set up our classifier so that it has high Precision but low Recall, or high Recall but low Precision.

If we want both Precision and Recall to be high, we can evaluate this using the F-score.

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$
$$F_1 = \frac{2PR}{P+R}$$

Where β is the weight we want towards either Recall or Precision. Typically $\beta = 1$.

Receiver Operating Characteristics (ROC)

A curve where we aim to minimize the false alarms, and maximize the true alarms. Ideally, we want to have the maximum area under the curve, though there is a tradeoff. The x-axis is the FP rate, and the y-axis is the TP rate.

Multi Class Evaluation

Confusion Matrix

Typically, all classes are considered 'Interesting' in a multi-class context. We can typically represent this in a matrix which has all the labels, and identifies the type of error it is.

Macro-Averaging

Calculates P,R per class, and then averages over number of classes. Good for small classes, as all classes are treated as equal.

$$\text{Precision}_M = \frac{\sum_{i=1}^{c} \text{Precision}_i}{c}$$
$$\text{Recall}_M = \frac{\sum_{i=1}^{c} \text{Recall}_i}{c}$$

Micro-Averaging

The TP, FP rates are calculated per class, but averaged together over the total pool of test instances. Micro average is dominated by large classes, should use if more important.

$$\text{Precision}_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FP_i}$$
$$\text{Recall}_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FN_i}$$

Weighted Averaging

Calculates P,R per class, and then averages based on the proportion of instances in that class.

$$\text{Precision}_w = \sum_{i=1}^{c} \text{Precision}_i \times (\frac{n_i}{N})$$
$$\text{Recall}_w = \sum_{i=1}^{c} \text{Recall}_i \times (\frac{n_i}{N})$$

Model Comparisons

Baseline: A naive method in which we would expect any reasonably well-developed method to be better. Basically the bare minimum that a model should perform, eg. accuracy should be better than randomly assigning a label.

Benchmark: Established rival technique in which we are pitching our method against. Eg. We want it to be better than the accuracy of Naive Bayes.

Random Assignment: Randomly assign a class to each test instance.

Random Weighted Assignment: Randomly assigns a class to each test instance, weighting the class assignment according the prior belief.

Zero - R The majority class baseline. Classifies all test instances according to the most common class in the training data. Inappropriate if the majority class is `FALSE` and the goal is to identify a `TRUE` value in a majority.

One - R Select one attribute and use it to predict an instance's class. Test each attribute and select the one with the smallest error rate. Eg. If Outlook = Sunny has majority `Yes` class, we assign any test instance with Outlook = Sunny, Class = `Yes`

Advantages: Easy to understand and implement, often presents good results.

Disadvantages: Unable to capture attribute interactions, and has a bias towards attributes with many possible values.

Feature Selection

Goals: Leads to better performance according to some evaluation metric. Seeing important features can also give more information about the problem, and reducing features can result in a faster answer. Generally, we care more about accuracy than speed.

Iterative Feature Selections: Wrappers

Wrapper
Chooses the subset of attributes that give the best performance on the development data. Builds, trains and tests a model on all combinations of features (eg. [f1], [f1,f2], [f2]).

Advantages
Feature set with the optimal performance on development data is chosen.

Disadvantages
Takes a long time (around 2^m for m features) **Greedy Search**
Trains and evaluates model on each single attribute. Based on the evaluation metric, will pick the best attribute. Eg.($\{f_0\} = 0.2$, $\{f_1\} = 0.4$, $\{f_2\}= 0.5$), f_2 will be picked. The process is then repeated for 2 attributes and so on, until the accuracy stops increasing based on some small value epsilon.

Advantages
Is quadratic in speed, takes $\frac{1}{2}m^2$ cycles for m attributes, though in practice it often converges quicker. Faster than exponential time, but still slow.

Disadvantages
Converges to a sub-optimal and even bad solution. Due to the greedy nature of the algorithm, if make the wrong choice at step 1, there could be a better solution out there.

Ablation
Starts with all the attributes, then removes one attribute at a time until it diverges (according to some value epsilon). Eg. ($\{f_0, f_1, f_2\} = 0.5$, $\{f_0, f_1\} = 0.499$, $\{f_0, f_2\} = 0.3$), then will pick $\{f_0, f_1\}$ for the next iteration. It removes the attribute that causes the least performance degradation, it stops when it degrades more than epsilon.

Advantages
Mostly removes irrelevant attributes (at the start)

Disadvantages
Assumes attributes are independent, and not feasible on non-trivial data sets. It also takes quadratic time m^2.

Feature Filtering

Filtering: Evaluates the 'goodness' of each feature, separate from other features.

Pointwise Mutual Information (PMI) Gives a value that describes how related a class (C) and feature (A) are.

$$PMI(A, C) = log_2 \frac{P(A, C)}{P(A)P(C)}$$

If PMI >> 0: Attribute and class occur together much more often than randomly.
If PMI 0: Attribute and class occur as often as we'd expect from random chance.
If PMI << 0: Attribute and class are negatively correlated.

Generally, we want attributes with the greatest PMI, however any non-zero PMI is also considered helpful information.

Contingency Table
Compact representation of frequency counts for each class and attribute.

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

Where $P(a, c) = \frac{\sigma(a,c)}{N}$ etc.

Mutual Information
The expected value of PMI over all possible events.

$$MI(A, C) = \sum_{i \in \{a, \bar{a}\}} \sum_{j \in \{c, \bar{c}\}} P(i, j) log_2 \frac{P(i, j)}{P(i)P(j)}$$

Where 0 log 0 ≡ 0

Chi Square

Similar idea to MI, but different solution.Considers the expected value $E(W)$ if an attribute and class were independent.

Contingency Table(Chi Square)

Compact representation of frequency counts for each class and attribute.

	a	$\bar{a}$	Total
c	W	X	W+X
$\bar{c}$	Y	Z	Y+Z
Total	W+Y	X+Z	N = W+X+Y+Z

Process

We make the assumption that an attribute a, and class c are independent.

$$P(a,c) = P(a)P(c)$$
$$\sigma(a,c) = \frac{\sigma(a)\sigma(c)}{N}$$
$$E(W) = \frac{(W+Y)(W+X)}{W+X+Y+Z}$$

We then calculate the expected value $E(W)$ as above, and compare it to the observed value $O(W)$.
If the observed value is much greater than the expected value, then a occurs more often with c than we would expect at random. The value W can be predictive.
If the observed value is much smaller than the expected value, then a occurs less often with c than we would expect at random. The value W can be predictive.
If the observed value is close to expected value, then a occurs as often with c as we would expect randomly. The value W is not predictive (same as independent).
We can repeat the process with other attributes X, Y, Z etc.

Chi square calculation

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

Where i sums over rows, and j sums over columns.
Because the values are squared, χ^2 becomes much greater when $|O - E|$ is large, even if E is also large.

Non Binary Attributes

Nominal

- Can treat as multiple binary attributes (not great, loses information)
- Can modify contingency tables and formula

Modified MI

$$MI(O,C) = \sum_{i \in s,o,r} \sum_{j \in c,\bar{c}} P(i,j) log_2 \frac{P(i,j)}{P(i)P(j)}$$

Continuous

- Estimate probability based on a Gaussian distribution
- Most random variables are normally distributed due to central limit theorem
- Otherwise, for small data sets might need to use binomial/multinomial distributions

Ordinal

- Treat as binary (Good for frequency counts)
- Treat as continuous
- Treat as nominal (Not great)

Multi-class problems

- PMI, MI, χ^2 are all calculated per-class
- Need to make a point of selected features for each class to give our classifier the best chance of predicting everything correctly
- Features that are seen rarely but always with a given class will be seen as 'good'

Iterative Optimisation

Iterative Optimization

Used when there is no closed form solution to an equation, and we are unable to find the minimum and maximum from deriving and setting to zero. In these cases, we iteratively improve our estimate of $\hat{theta}$ until we arrive at a satisfactory solution. One of these methods is called *Gradient descent*.

Gradient Descent

Descends the function to find the optimum.

- Picks a random point in the function
- 'Descends' the function by moving in the opposite direction of the gradient (towards the minimum)
- Increments the location by a step size
- Stops when it reaches the minimum, or the value stops improving by ϵ

Gradient Descent Equation

For each step, a new θ is calculated

$$\theta \leftarrow \theta + \Delta\theta$$

Where $\Delta\theta$ is the derivative $\frac{\delta f}{\delta \theta}$, and the step size

- If $\frac{\delta f}{\delta \theta} > 0 : f(\theta) : \nearrow$ as $\theta \nearrow$
- If $\frac{\delta f}{\delta \theta} < 0 : f(\theta) : \nearrow$ as $\theta \searrow$
- If $\frac{\delta f}{\delta \theta} = 0$: we are at a minimum

Approaching the minimum

From the prior equation, we know the direction of the gradient is the opposite of the minimum. So to approach the minimum:

$$\theta \leftarrow \theta - \eta \frac{\delta f}{\delta \theta}$$

Gradient Descent with multiple parameters

If we have multiple parameters that need to be optimised,then we compute the partial derivative of $f(\theta)$ wrt θ_n, then we update the parameter individually.

- $\theta_1 \leftarrow \theta_1 - \eta \frac{\delta f}{\delta \theta_1}$
- $\theta_2 \leftarrow \theta_2 - \eta \frac{\delta f}{\delta \theta_2}$
- etc

Local Minima and Maxima

If a function is not convex, then we may reach a local minima or maxima instead of the global one. Gradient guarantees a global extrema for convex functions which are differentiable, and at least a local extrema for non convex functions. While local extrema can still work, we note that it won't necessarily be the optimal solution in non convex functions.

Logistic Regression

Model

Logistic Regression differs from Naive Bayes as it models $P(y|x)$ directly, whereas Naive Bayes *generates* this value from priors and likelihoods. It is also a binary classifier.

Generative Models

Naive Bayes is a generative model, and uses the prior distribution of a dataset to generate a probability for a specific instance. (Uses Bayes rule to convert into the desired probability, and makes the assumption that features are conditionally independent).

Discriminative Models

Logistic Regression is a discriminative model, and uses the observed data to determine the class (y) of an instance based on x. It does this based on the optimized value of $P(y|x)$. It also doesn't need to make any assumptions about the features.

Linear Regression

Used to predict a value y given x.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... = \theta_0 + \sum_i \theta_i x_i$$

Where $\theta_0, \theta_1, ...$ are weights and model parameters, and are what needs to be optimized.
We can measure the error by the sum of squared errors (SSE):

$$L = \sum_{i=1}^{N} (\hat{y}^i - y^i)^2$$

Log Odds

Another name for logistic transformation. The odds are the fraction of successes of the fraction of failures.

$$\text{odds} = \frac{\text{P(success)}}{\text{P(failures)}} = \frac{\text{P(success)}}{1 - \text{P(success)}}$$

Deriving Logistic Regression

We use a logistic transformation for $p(x)$ to obtain the sigmoid function. The logistic transformation is unbounded for x, but binds the probability $p(x)$ to be between 0 and 1.

$$log(\frac{p(x)}{1-p(x)}) = \theta_0 + \theta_1 x_1 + ... + \theta_F x_F$$

$$p(x) = \frac{1}{1 + exp(-(\theta_0 + \sum_{f=1}^{F} \theta_f x_f))}$$

- $(\theta_0 + \sigma_{f=1}^{F} \theta_f x_f) > 0$ means y = 1
- $(\theta_0 + \sigma_{f=1}^{F} \theta_f x_f) \approx 0$ means y has a lot of uncertainty (cannot be classified, or questionable classification)
- $(\theta_0 + \sigma_{f=1}^{F} \theta_f x_f) < 0$ means y = 0

We can also denote the logistic function as a sigmoid function $\sigma(x; \theta)$.
We can also define a **decision boundary**, such as $p(x) = 0.5$, where $p(x) > 0.5$ means y = 1, and $p(x) < 0.5$ means y = 0.

$$P(y = 1|x_1, x_2, ..., x_F; \theta) = \frac{1}{1 + exp(-(\theta^T x))} = \sigma(\theta^T x)$$

Finding Parameters(θ)

Minimize the negative conditional log likelihood.

$$\mathcal{L}(\theta) = -P(Y|X;\theta) = -\prod_{i=1}^{N} P(y^i|x^i;\theta)$$

Where $P(y=1|x;\theta) = \sigma(\theta^T x)$, $P(y=0|x;\theta) = 1-\sigma(\theta^T x)$

Substituting:

$$\mathcal{L}(\theta) = -\prod_{i=1}^{N} (\sigma(\theta^T x^i))^{y^i} \times (1-\sigma(\theta^T x^i))^{1-y^i}$$

Taking the log of this function:

$$log(\mathcal{L}(\theta)) = -\sum_{i=1}^{N} y^i log\sigma(\theta^T x^i) + (1-y^i) log(1-\sigma(\theta^T x^i))$$

Notes:

The derivative of the sigmoid function is $\frac{\delta\sigma(z)}{\delta z} = \sigma(z)[1-\sigma(z)]$

The chain rule: $\frac{\delta A}{\delta D} = \frac{\delta A}{\delta B} \times \frac{\delta B}{\delta C} \times \frac{\delta C}{\delta D}$

We also compute one instance at a time, and one feature at a time.

The derivative of the log likelihood wrt. a single parameter θ_j for all training examples can be given as:

$$\frac{log\mathcal{L}(\theta)}{\delta\theta_j} = \sum_{i=1}^{N} (\sigma(\theta^T x^i) - y^i) x_j^i$$

We would usually solve this by setting the derivative to zero and solving, but since we can't, we can use Gradient Descent.

$$\theta_j \leftarrow \theta_j - \eta \sum_{i=1}^{N} (\sigma(\theta^T x^i) - y^i) x_j^i$$

Multinomial Logistic Regression

Instead of using the sigmoid function, we can use the softmax function (which is a generalization of the sigmoid function).

$$p(y=c|x;\theta) = \frac{exp(\theta_c x)}{\sum_k exp(\theta_k x)}$$

And for each different class c, we learn a parameter vector $\theta_c = [\theta_1, \theta_2 ... \theta_k]$

Pros

- Probabilistic interpretation
- No assumptions on features
- Often outperforms Naive Bayes
- Suited to frequency based features

Cons

- Only works for linear feature-data relationships
- Some feature scaling issues
- Often needs a lot of data to work well
- Regularization is required as overfitting is a big problem