Machine Learning Notes

Patrick O'Neil

April 24, 2014

Abstract

These notes provide a quick and dirty exploration of several areas of machine learning. The focus of these notes is on implementation as opposed to theory. Very little background material is covered and it is assumed the reader has a decent grasp of probability theory and statistics. Most of these notes are derived from the Pattern Recognition notes available on the TAMU PSI lecture notes page (`http://psi.cse.tamu.edu/teaching/lecture_notes/`). I will be periodically expanding the notes.

Contents

1 Decision Theory

Likelihood Ratio

$$\Lambda(x) = \frac{p(x|\omega_1)}{p(x|\omega_2)}$$

Likelihood Ratio Test To classify x into classes ω_1 or ω_2 using the likelihood ratio,

$$\Lambda(x) > P(\omega_2)/P(\omega_1) \Rightarrow \omega_1$$

$$\Lambda(x) < P(\omega_2)/P(\omega_1) \Rightarrow \omega_2$$

Bayes Error Rate is the probability of error.

$$P(\text{error}) = \int_{-\infty}^{\infty} P(\text{error}|x)p(x)dx$$

Bayes Risk Let C_{ij} represent the cost of choosing class ω_i when ω_j is the true class.

$$\mathscr{R} = E[C] = \sum C_{ij} P(\text{choose } \omega_i \text{ and } x \in \omega_j) = \sum C_{ij} P(x \in R_i|\omega_j)P(\omega_j)$$

2 Density Estimation

General expression for non-parametric density estimation is given by $p(x) \cong \frac{k}{NV}$ where V is the volume surrounding x, k is the number of samples found within V, and N is the total number of samples. Fixing V and approximating k leads to *Kernel Density Estimation*. Fixing k and approximating V leads to *k-Nearest Neighbors Estimation*.

2.1 Kernel Density Estimation

Definition The *Parzen Window* is a kernel function defined on the unit hypercube of dimension d given by $K(u) = 1$ if $|u_j| < \frac{1}{2}\ \forall j$ and $K(u) = 0$ otherwise. Then

$$K\left(\frac{x - x^n}{h}\right) = \begin{cases} 1 & x^n \in H_h(x) \\ 0 & \text{else} \end{cases}$$

where $H_h(x)$ is a hypercube of side length h centered at x. Using this, we can define a kernel-density estimate based on Parzen windows:

$$p_{KDE}(x) = \frac{1}{Nh^D} \sum_{n=1}^{N} K\left(\frac{x - x^n}{h}\right)$$

The Parzen window approach yields discontinuities and does not differentiate between points that are close to the estimation point and points that are far (but still fall within the window). Thus, we often use smooth kernels.

Definition The *Gaussian Kernel* is given by

$$K(x) = \left(\frac{1}{2\pi}\right)^{\frac{D}{2}} e^{-\frac{1}{2}x^T x}$$

This is a smooth and radially symmetric kernel. The associated KDE estimate is given by

$$p_{KDE}(x) = \frac{1}{Nh^D} \sum_{n=1}^{N} K\left(\frac{x - x^n}{h}\right)$$

the smoothing parameter h is referred to as the *bandwidth*.

Note that we are using a single smoothing parameter for all directions. This is not always appropriate (for example, there may be more variance in certain directions). To cope with this, one may,

- Pre-scale: normalize each axis to unit variance.
- Pre-whitening: lineary transform data so that the covariance matrix is the identity matrix. This transform is given by $y = \Lambda^{-\frac{1}{2}} M^T x$ where Λ is the vector of eigenvalues of the covariance matrix and M is the matrix of eigenvectors of the covariance matrix.

Definition A good alternative for multivariate KDE (dimension D) is the *Product Kernel*,

$$p(x) = \frac{1}{N} \sum_{n=1}^{N} K(x, x^{(n)}, h_1, ..., h_D)$$

where

$$K(x, x^{(n)}, h_1, ..., h_D) = \frac{1}{h_1 ... h_D} \prod_{d=1}^{D} K_d \left(\frac{x_d - x_d^{(n)}}{h_d}\right)$$

2.2 k-Nearest Neighbor Estimation

Definition Recall that the general form of the non-parametric density estimation is given by $p(x) \cong k/NV$. The *k-Nearest Neighbor Estimate* is given by

$$p_{kNN}(x) = \frac{k}{Nc_D R_k^D(x)}$$

where $R_k^D(x)$ is the distance between the estimation point x and its k-th nearest neighbor. Also, c_D is the volume of the unit D-dimensional sphere.

Notice that this will produce discontinuous probability estimates.

Definition Assume a dataset with N samples, N_i from class ω_i. The *kNN Bayes Classifier* is given by

$$p(\omega_i|x) = \frac{p(x|\omega_i)p(\omega_i)}{p(x)} = \frac{\frac{k_i}{N_i V} \cdot \frac{N_i}{N}}{\frac{k}{NV}} = \frac{k_i}{k}$$

Thus, our decision rule reduces to choosing the class ω_i such that the largest percentage of the k-nearest neighbors of x are from class ω_i.

Since the kNN approach is based on Euclidean distance, it is highly sensitive to noise. Instead of using plain Euclidean distance, we can use a weighted Euclidean distance

$$d_w(x_u, x) = \sqrt{\sum_{k=1}^{D} [w_k(x_{u,k} - x_k)]^2}$$

where w_k is a weight assigned to the k-th dimension of the data.

The follwing includes two approaches to computing kNN:

- *Bucketing (Elias's Algorithm)*
  1. Divide space into identical cells. For each cell, store data points in a list.
  2. Examine cells in order of increasing distance from query point. For each cell, distance is computed between internal points and query point.
  3. Terminate search when distance from query point to cell exceeds distance to closest point already visited.
- *k-d Trees*: k-dimensional generalization of a binary search. See Figure 1.

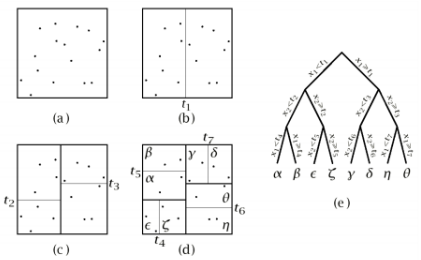


Figure 1: $k-d$ Tree Example

2.3 Mixture Models

A mixture models seeks to model the presence of subpopulations within the overall population. The general set up is as follows:

- N random variables (observations) each assumed to be distrubuted according to a mixture of K components.
- Each of the K components belongs to the same family of distributions (eg all Gaussian) but with different parameters.
- K different parameters (or sets of parameters) for the distributions.
- N latent variables indicating the identity to which component each observation belongs.
- A set of K non-negative mixture weights which sum to 1.

3 Dimensionality Reduction

Two approaches are available for dimensionality reduction:

- *Feature Extraction*: Create set of new features (smaller in dimension) by combinations of existing features.
- *Feature Selection*: Choose a subset of all the features.

For feature selection methods we want to do one of

- *Signal Representation*: Goal is to represent the samples accurately in a lower-dimensional space. (example: PCA)
- *Signal Classification*: Goal is to enhance the class-discriminatory information in the lower-dimensional space. (example: LDA)

3.1 Principle Component Analysis (PCA)

Principle Component Analysis: Linear mapping of data to a lower dimensional space such that the variance of the data in low-dimensional representation is best preserved. In practice, the correlation matrix of the data is constructed and the eigenvectors on this matrix are computed. The eigenvectors that correspond to the largest eigenvalues (the principal components) can now be used to reconstruct a large fraction of the variance of the original data. (See Also: Kernel-PCA for non-linear application)

Kernel Principle Component Analysis: Perform a kernel trick Φ to the data and compute covariance matrix C of $m \times n$ matrix $\Phi(\mathbf{X})$. Compute eigenvectors of C and project data onto these eigenvectors.

Kernel Trick: For data points $\{x_1, x_2, ..., x_n\}$ pick and inner product $k(x, y)$ and compute the inner product space structure for the data. That is compute $(K)_{ij} = k(x_i, x_j)$. Now project data onto the principle components.

3.2 Linear Discriminant Analysis (LDA)

In the case of two-dimensions, we can think of LDA as finding the line where the projected data best separates the classes. See Figure 2 for an example. The left figure is a poor choice while the right figure separates the classes very effectively.

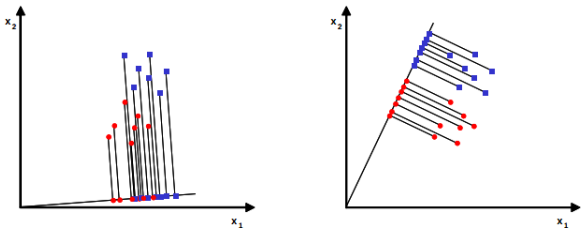


Figure 2: 2-dimensional LDA example

Definition For each class, the *scatter* is given as

$$\tilde{s}_i^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2$$

where $\tilde{\mu}_i = \frac{1}{|\omega_i|} \sum_{y \in \omega_i} y = \frac{1}{|\omega_i|} w^T x = w^T \mu_i$. The *Fisher Linear Discriminant* is the linear function $w^T x$ that maximizes the *criterion function*

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

The above definition implies we are looking for a projection where examples from the same class are projected very close to each other and, at the same time, the projected means are as far apart as possible.

Definition The *Fisher Linear Discriminant* is given by

$$w^* = S_W^{-1}(\mu_1 - \mu_2)$$

where $S_W = S_1 + S_2$ is called the *Within-Class Scatter* and

$$S_i = \sum_{x \in \omega_i} (x - \mu_i) \cdot (x - \mu_i) = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

For the case when the number of classes is C, we seek $C-1$ projections $[y_1, y_2, ..., y_{C-1}]$ by means of $C-1$ projection vectors w_i arranged by columns into a projection matrix $W = [w_1|w_2|...|w_{C-1}]$ and so $y_i = w_i^T x \Rightarrow y = W^T x$. We now have *Within-Class Scatter* given by $S_W \sum_{i=1}^{C} S_i$ and

$$S_i = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

$$S_B = \sum_{i=1}^{C} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

Then $S_T = S_B + S_W$ is called the *Total Scatter*. The projected versions, $\tilde{\mu}_i, \tilde{\mu}, \tilde{S}_W, \tilde{S}_B$ are defined in the obvious way. Then our objective function becomes

$$J(W) = \frac{|\tilde{S}_B|}{|\tilde{S}_W|} = \frac{|W^T S_B W|}{|W^T S_W W|}$$

where $|\cdot|$ denotes the determinant. We seek W^* that maximizes J. The optimal projection W^* is the one whose columns are the eigenvectors corresponding the the largest eigenvalues of the following generalized eigenvalue problem,

$$W^* = [w_1^*|w_2^*|...|w_{C-1}^*] \Rightarrow (S_B - \lambda_i S_W) w_i^* = 0$$

Variants:

- Non-Parametric LDA (Fukunaga): Computes S_B using local information and kNN. Able to preserve structure of the date more closely. Can find more than $C-1$ features.
- Orthonormal LDA (Okada and Tomita): Computes projections that maximize Fischer criterion and are pair-wise orthonormal. Combines eigenvalue solution of $S_W^{-1} S_B$ and Gram-Schmidt orthonormalization procedure. Can find more than $C-1$ features.
- Generalized LDA (Lowe): Incorporates a cost function C_{ij} to weigh importance of separating certain classes.
- Multilayer Perceptrons (Webb and Lowe)

3.3 Feature Subset Selection

The following are some search algorithms for determining a subset of features to use.

Definition The *Plus-L Minus-R Selection* begins with either the full set of features ($L < R$) or the empty set ($L > R$). Then we iteratively add L features and remove R features. If $L < R$, we begin by removing features and if $L > R$, we begin by adding features. To determine which feature to add or remove, we test the resulting subset with an objective function (filter or wrapper). Floating selection methods generalize this and allow L and R to vary during execution.

Filters: Distance between classes (Euclidean, Mahalanobis, etc), Determinant of $S_W^{-1} S_B$ matrix[1], correlation coefficient ($\sum_{i=1}^{M} \rho_{iC} / \sum_{i=1}^{M} \sum_{j=i+1}^{M} \rho_{ij}$)[2], information theoretic measures (mutual information is usually slow and replaced by a Heuristic).
Wrappers: Some sort of classifier. Train a classifier on the subset of features and check performance.

[1] see LDA section for definition
[2] ρ_{iC} is the correlation coefficient between the i-th feature and the class labels

3.4 Mainfold Learning: ISOMAP

1. Build sparse graph G of the data using only the k-nearest neighbors.
2. Build distance matrix by finding shortest paths along G (Dijkstra's algorithm).
3. Build low-D embedded space to best preserve the complete distance matrix. That is, minimize

$$E = \|\tau(D_G) - \tau(D_Y)\|_{L^2}$$

The solution is to project points to the top n eigenvectors of D_G. (The smaller the eigenvalue, the more clumped the data is along the projection of that eigenvector)

4 Classification

4.1 Naïive Bayes Classifier

The naïve Bayes classifier assumes that the features are class-conditionally independent,

$$p(x|\omega_i) = \prod_{d=1}^{D} p(x_d|\omega_i)$$

5 Cross-Validation

In this section, we present some cross-validation techniques.

5.1 Holdout Method

Definition The *Random Subsampling* method performs K data splits of the entire dataset. Each split selects a fixed number of samples randomly without replacement for use in testing. See Figure 3.

Definition The *K-fold Cross Validation* method creates a K-fold partition of the data. For each of K experiements, use the $K-1$ folds for training and a different fold for testing. Advantage of this approach over Random Subsampling is that all samples are guaranteed to be used.[3] See Figure 4.

Definition The *Bootstrap* method is a resampling technique with replacement. Given N samples, randomly select N samples (with replacement) for training. Use the left out samples for testing. Repeat this process K times. For each bootstrapped sample, compute the desired statistic. Then combine the results for each bootstrap iteration (via averaging, etc) and use this result to analyze the bias or variance in the original statistic estimate.

Bootstrap Example: Given $X = \{3, 5, 2, 1, 7\}$, We seek to compute the bias of the sample mean $\mu' = 3.6$. We generate the following bootstrap samples

$$X_1 = \{7, 3, 2, 3, 1\} \Rightarrow \mu_1 = 3.2$$
$$X_2 = \{5, 1, 1, 3, 7\} \Rightarrow \mu_2 = 3.4$$
$$X_3 = \{2, 2, 7, 1, 3\} \Rightarrow \mu_3 = 3.0$$

This yields a bootstrap average value of $\mu_B = 3.2$. Thus, $\text{Bias}(\mu') = 3.2 - 3.6 = -0.4$ and so there is a -0.4 bias in our sample mean. Thus, an unbiased estimate would be $\mu_U = 3.6 + 0.4 = 4.0$.

[3] If $K = 1$, this is referred to as leave-one-out cross validation

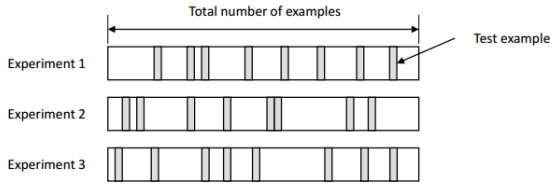


Figure 3: Random Subsampling Method

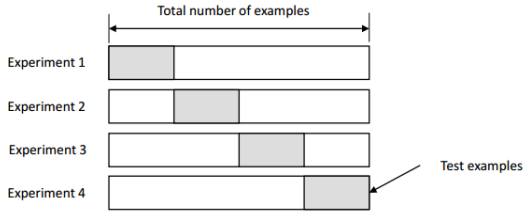


Figure 4: K-Fold Cross Validation

6 Artifical Neural Networks

6.1 Multilayer Perceptron

Definition A *Multilayer Perceptron* consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. All nodes except input nodes are hidden nodes with nonlinear activation functions. The Multilayer perceptron uses backpropagation for learning. Since it contains at least three layers (input, output, hidden), it is considered a deep neural network. See the notation in Table 1.

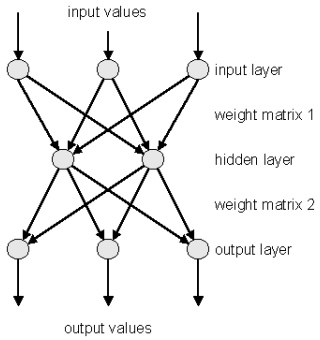


Figure 5: Simple Multilayer Perceptron

Activation Functions and Weights: Commonly used activation functions include $\tanh(v_i)$ and $(1 + +e^{-v_i})^{-1}$ where v_i is the weighted sum of the input synapses. Other activation functions include, the rectifier function, $\max(0, x)$, and the softplus function, $\log(1 + e^x)$. The softplus function is a smooth approximation of the rectifier function. The weights are given by a matrix at each layer $W = (w_{ij})$ where w_{ij} is the weight of the directed edge from input node i to processing node j.

Learning: The learning stage involves finding appropriate weights for the edges. Suppose the error in output node j is given by $e_j(n) = t_j(n) - y_j(n)$. Then the total error is given by

$$J(W) = \frac{1}{2}\sum_{j}^{N}(t_k - y_k)^2$$

n, m, N	Number of input, hidden, and output nodes respectively
x_i	ith input
w_{ij}^1	weight of edge connecting x_i to h_j
$\hat{h}_j^1$	$(x_1, x_2, ..., x_n) \cdot (w_{1j}, w_{2j}, ..., w_{nj})$
h_j	output of the jth hidden node
w_{jk}^2	weight of edge connecting h_j to y_k
$\hat{y}_k$	$(h_1, ..., h_m) \cdot (w_{1k}, w_{2k}, ..., w_{mk})$
y_k	output of the kth output node
t_k	target value of the kth output node

Table 1: Multi-Layer Perceptron Notation

We proceed using gradient decent. Thus, we have

$$w_{n+1} = w_n + \Delta w_n = w_n - \eta \frac{\partial J(W)}{\partial w_n}$$

where η is the learning rate, or step-size. In the case of the *output weights*, we have,

$$\frac{\partial J(W)}{\partial w_{jk}^2} = \frac{\partial J(W)}{\partial y_k} \frac{\partial y_k}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial w_{jk}^2}$$

where

$$\frac{\partial J(W)}{\partial y_k} = \frac{\partial}{\partial y_k} \left(\sum_{i=1}^{N} (y_i - t_i)^2 \right) = (y_k - t_k)$$

$$\frac{\hat{y}_k}{\partial w_{jk}^2} = \frac{\partial}{\partial w_{jk}^2} \left(\sum_{i=1}^{m} w_{ik}^2 h_i \right) = h_j$$

and the middle partial derivative is the derivative of the nonlinear activation function.

Example 1. For the case the activation function being

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

we have $y_k = \phi(\hat{y}_k)$ and so

$$\frac{\partial y_k}{\partial \hat{y}_k} = \frac{\partial}{\partial \hat{y}_k} \left(\frac{1}{1 + e^{-\hat{y}_k}} \right) = \frac{e^{-\hat{y}_k}}{(1 + e^{-\hat{y}_k})^2} = \left(\frac{e^{-\hat{y}_k}}{1 + e^{-\hat{y}_k}} \right) \left(\frac{1}{1 + e^{-\hat{y}_k}} \right) = (1 - y_k) y_k$$

So for this activation function, we have

$$\frac{\partial J(W)}{\partial w_{jk}^2} = (y_k - t_k)(1 - y_k) y_k h_j$$

Now we turn to the case of the *hidden weights*. We have

$$\frac{\partial J(W)}{\partial w_{ij}^1} = \frac{\partial J(W)}{\partial h_j} \frac{\partial h_j}{\partial \hat{h}_j} \frac{\partial \hat{h}_j}{\partial w_{ij}^1}$$

where

$$\frac{\partial \hat{h}_j}{\partial w_{ij}^1} = x_i$$

and the middle derivative is the derivative of the activation function as before. However, the first derivative is not so straight forward since we don't have target variables for the hidden units. This is referred to as the *credit assignment problem*. We must propagate the error and compare to the target values at the output stage. That is,

$$\frac{\partial J(W)}{\partial h_j} = \sum_{i=1}^{N} \frac{\partial J(W)}{\partial y_i} \frac{\partial y_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_j}$$

From earlier, we have

$$\frac{\partial J(W)}{\partial y_i} \frac{\partial y_i}{\partial \hat{y}_i} = (y_i - t_i) \frac{\partial y_i}{\partial \hat{y}_i}$$

where the derivative on the RHS is the derivative of the activation function. The last term is

$$\frac{\partial \hat{y}_i}{\partial y_j} = w_{ji}^2$$

Therefore,

$$\frac{\partial J(W)}{\partial h_j} = \sum_{i=1}^{N} (y_i - t_i) w_{ji}^2 \frac{\partial y_i}{\partial \hat{y}_i}$$

Note 1. (*Momentum Gradient Descent*) Encountering local minima while performing gradient descent is not uncommon. A popular method to avoid local minimia is to compute a temporal average direction. An easy implementation is to use an exponential average,

$$\Delta w(n) = \mu[\Delta w(n-1)] + (1-\mu) \left[\eta \frac{\partial J(w)}{\partial w} \right]$$

the term $\mu \in (0, 1)$ is called the *momentum*.

There are also methods which use adaptive learning rates. These work by increasing the learning rate if the gradient direction has remained unchanged and decreasing the learning rate if the direction is changing.

Note 2. Tricks of the trade:

(a) MLPs train faster with anti-symmetric activation functions ($f(-x) = -f(x)$).

(b) Target values must be within range of activation functions. Recommended target values are not the asymptotic values of the activation function.

(c) Input variables should have mean 0, same variance (Fukunaga's whitening transform), and uncorrelated (PCA).

(d) Initial weights shold be small. H-O weights whould be larget than I-H weights since they carry the back-propagated error.

(e) Weight Updates: The weights may be updated online (update after running each example through) or in batch (update after running all examples, calculate Δw after each example and sum at the end). Batch training is recommended. Online training is sensitive to the ordering of the examples.

6.2 Other Versions

- *Restrictive Boltzman Machine*: To be added.
- *Convolution Neural Network*: Consists of multiple layers of small neuron collections which look at small portions of the input image. The results of these collections are then tiled so that they overlap to obtain a better representation of the original image (repeated for every layer).
- *Radial Basis Function Network*: To be added.

7 Genetic Algorithms

An example genetic algorithm is available ("GeneticAlgorithm.R"). It uses one point crossover and no mutation.

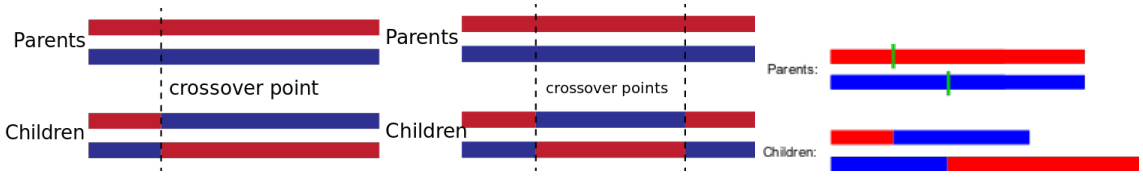
The solutions need to be encoded as genetic informations (bits commonly). Randomly generate a large first generation of solutions. Then perform the following process iteratively

1. Evaluate the population (or a sample) using a fitness function.
2. Choose pairs of the most fit.
3. Create offspring for new generation. (Mutation and Crossover)

until a stopping condition is reached.

7.1 Crossover

Here are some images of crossover techniques. In the *uniform crossover scheme* (UX) individual bits in the



(a) One Point (b) Two Point (c) Cut & Splice

Figure 6: Crossover Methods

string are compared between two parents. The bits are swapped with a fixed probability, typically 0.5. In the *half uniform crossover scheme* (HUX), swaps half the hamming distance number of chromosomes.

Three Parent Crossover: Choose 3 parents. Compare triples of chromosomes. Offspring gets most common bit value in the triple.

7.2 Mutation

The classic example of a *mutation operator* involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state.

Mutation Methods

- *Non-Uniform*: The probability that amount of mutation will go to 0 with the next generation is increased by using non-uniform mutation operator.It keeps the population from stagnating in the early stages of the evolution.It tunes solution in later stages of evolution.This mutation operator can only be used for integer and float genes.
- *Uniform*: This operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.
- *Gaussian*: This operator adds a unit Gaussian distributed random value to the chosen gene. If it falls outside of the user-specified lower or upper bounds for that gene,the new gene value is clipped. This mutation operator can only be used for integer and float genes.

7.3 Fitness Function

Human created function to rate the 'fitness' of solutions. Computational complexity is an issue to bear in mind when creating fitness functions. Approximating the fitness may be necessary. The graph of a fitness function is the *Fitness Landscape*.

8 Time-Series Analysis

Autocorrelation For a repeatable process X, the autocorrelation between times r as s is given by

$$R(s,t) = \frac{E[(X_t - \mu_t)(X_s - \mu_s)]}{\sigma_t \sigma_s}$$

If X_t is a second-order stationary process, then μ and σ^2 are time independent and we get

$$R(\tau) = \frac{E[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2}$$

8.1 Fourier Analysis

Cross-correlation: Given two time series $x(t), y(t)$, we define the cross-correlation as

$$\langle x(t), y(t+\tau) \rangle = \sum_{k=1}^{N} x(t_k) y(t_k + \tau)$$

where τ is a time shift. The *autocorrelation* of $x(t)$ is then $\langle x(t), x(t+\tau) \rangle$, i.e. the cross-correlation of x with itself.

Definition The *Fourier Transform* of a time-series $x(t)$ is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} dt$$

where f is referred to as the frequency. The *Reverse Fourier Transform* is given by

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi i f t} df$$

The *Discrete Fourier Transform* (DFT) is given by

$$X(n) = \sum_{k=0}^{N-1} x(k) e^{-\frac{2\pi i}{N} nk}$$

In the example in Figure 7, the signal is given by $x(t) = 10\sin(2\pi 10t) + 3\sin(2\pi 100t)$. Notice the peaks occuring at the frequencies of 10 and 100.

Applying the Fourier transform is only meaningful for stationary processes (mean and variance are static). For more dynamic time-series, one uses the DFT on windows of the data. This is known as *Short-Time Fourier Transform*. This process is given by

- Define analysis window size and overlap.
- Define a windowing function (eg Hann, Gaussian, etc). These functions must be chosen so that when overlapped, their sum is unity. Notice the windowing function sums to 1 across all time interval in Figure 8 (except for the ends).

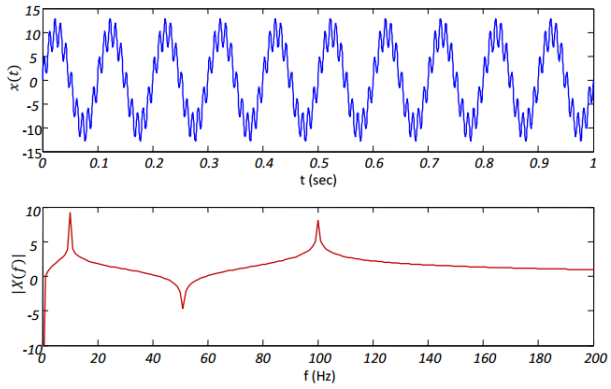


Figure 7: Discrete Fourier Transform Example

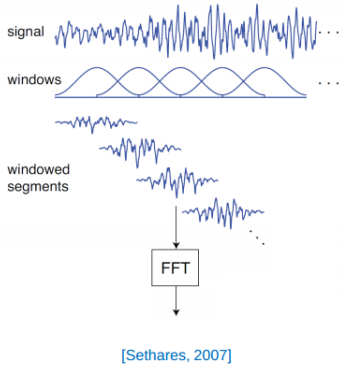


Figure 8: Short-Time Fourier Transform

- Multiply the signal with the windowing function.
- Apply Fast Fourier Transform to each segment.

The short-time fourier transform is then given as

$$X(f_n, t_i) = \sum_{k=0}^{N-1} x[k]w[k-i]e^{-\frac{2\pi i}{N}nk}$$

where f_n is the $n-$th discrete frequency and t_i is the starting time of the i-th analysis window. (w is the windowing function).

9 Additional Notes

Maximum Likelihood Estimator (MLE):

$$Y^{\text{predict}} = \arg\max_v P(X_1 = u_2, ..., X_m = u_m | Y = v)$$

Maximum A-Posteriori Estimator (MAP):

$$Y^{\text{predict}} = \arg\max_v P(Y = v | X_1 = u_2, ..., X_m = u_m)$$

For computing high dimensional joint probablities, use logs to prevent system underflow, i.e.

$$\prod_{j=1}^{n} P(X_j = u_j|Y = v) \quad \text{vs} \quad \sum_{j=1}^{n} \log P(X_j = u_j|Y = v)$$

Mahalanobis Distance Let $x = (x_1, x_2, ..., x_N)^T$ be a multivariate random variable from a group of values with mean $\mu = (\mu_1, \mu_2, ..., \mu_N)^T$ and covariance matrix S. The Mahalanobis distance is defined as:

$$D_M(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}$$

9.1 Expectation Minimization Algorithm

Assume a dataset contains two types of features: (1) a set of features X whose values are known, and (2), a set of features Z whose values are unknown. Now define the *complete-data likelihood* to be the joint pdf $p(X, Z|\theta)$. The following two steps are then repeated iteratively on the data.

- *Expectation*: Find the expected value of $\log p(X, Z|\theta)$ with respect to the unknown data Z, given the data X and the current parameter estimate θ^{i-1}:

  $$Q(\theta|\theta^{i-1}) = E_Z[\log p(X, Z|\theta)|X, \theta^{i-1}]$$

  where θ are the new parameters that we seek to optimize to increase Q. Note here that Z is a random variable defined by $p(Z|X, \theta^{i-1})$.

- *Maximization*: Find the argument that maximizes the expected value $Q(\theta|\theta^{i-1})$, i.e.

  $$\theta^i = \text{argmax} Q(\theta|\theta^{i-1})$$

In a nutshell, since Z is unknown, the best we can do is maximize the average log-likelihood across all possible values of Z. See Figure 9.

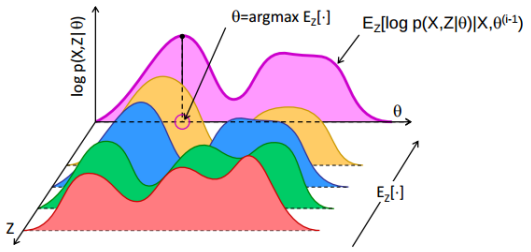


Figure 9: Expectation-Maximization Visualization

9.2 Similarity Measures

Cosine Similarity: The cosine similarity of $u, v \in \mathbb{R}^n$ is given by

$$\frac{u \cdot v}{\|u\|\|v\|}$$