Chapter 3

Classical Optimization and Search Techniques

In this chapter we discuss a few popular optimization techniques in use in current day natural language processing algorithms. First we present the Hidden Markov Model (HMM) used for part-of-speech tagging (POS-tagging) among other tasks. Then we formulate the POS-tagging problem using HMM and present its classical solution which is due to the Viterbi algorithm [13].

Then we present Conditional Random Fields, an alternative to HMMs, which find applications in pattern recognition and machine learning problems such as shallow parsing [3], named entity recognition and gene finding.

Conditional Random Fields are followed by a description of a popular variant of the gradient descent algorithm, namely the iterative scaling algorithm. This finds use in solving for the training objectives of exponential models which use maximum likelihood estimation on the training data to get the parameters.

Then we present Support Vector Machines (SVM) which are supervised learning models used for classification and regression analysis. They have been successfully applied to numerous applications mainly because of their ability to perform non-linear classification via the usage of a technique called the *Kernel* trick.

Finally a couple of probabilistic meta-heuristic optimization techniques are presented. The first is genetic algorithms which are inspired from the principle of natural selection. The second is simulated annealing which efficiently finds a good approximation to the global optimum of a given function in a large search space.

3.1 Hidden Markov Model

The Hidden Markov model is a stochastic model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. A key aspect of the HMM is its Markov property which is described in brief below along with other background definitions required.

3.1.1 Stochastic Process

Definition 1. ***Stochastic Process****: A stochastic process is a collection of random variables often used to represent the evolution of some random value over time.*

There is indeterminacy in a stochastic process. Even if we know the initial conditions, the system can evolve in possibly many different ways.

3.1.2 Markov Property and Markov Modelling

Definition 2. ***Markov Property****: A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state, not on the sequence of events that preceded it. That is, the process is memoryless.*

A Markov model is a stochastic model that follows the Markov property. Next we present the HMM through the urn problem which eases the exposition. In a further sub-section the formal description of the HMM is given.

3.1.3 The urn example

There are N urns, each containing balls of different colours mixed in known proportions. An urn is chosen and a ball is taken out of it. The colour of the ball is noted and the ball is replaced. The choice of the urn from which the n^{th} ball will be picked is determined by a random number and the urn from which the $(n-1)^{th}$ ball was picked. Hence, the process becomes a Markov process.

The problem to be solved is the following: Given the ball colour sequence, find the underlying urn sequence. Here the urn sequence is unknown (hidden) from us and hence the name *Hidden* Markov Model. The diagram[1] below shows the architecture of an example HMM. The quantities marked on the transition arrows represent the transition probabilities.

3.1.4 Formal Description of the Hidden Markov Model

The hidden Markov model can be mathematically described as follows:

N	$=$	number of states
T	$=$	number of observations
$\theta_{i=1\ldots N}$	$=$	emission parameter for an observation associated with state i
$\phi_{i=1\ldots N,j=1\ldots N}$	$=$	probability of transition from state i to state j
$\boldsymbol{\phi}_{i=1\ldots N}$	$=$	N-dimensional vector, composed of $\phi_{i,1\ldots N}$; must sum to 1
$x_{t=1\ldots T}$	$=$	state of observation at time t
$y_{t=1\ldots T}$	$=$	observation at time t
$F(y\|\theta)$	$=$	probability distribution of an observation, parametrized on θ
$x_{t=2\ldots T}$	$\sim$	Categorical($\boldsymbol{\phi}_{x_{t-1}}$)
$y_{t=1\ldots T}$	$\sim$	$F(\theta_{x_t})$

[1]Source:`http://en.wikipedia.org/wiki/Hidden_Markov_model`

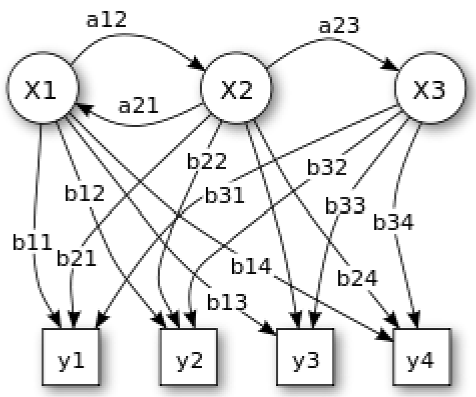

Figure 3.1: An example Hidden Markov Model with three urns

3.1.5 The Trellis Diagram

Given the set of states in the HMM, we can draw a linear representation of the state transitions given an input sequence by repeating the set of states at every stage. This gives us the trellis diagram. A sample trellis is shown in Figure 3.2[2]. Each level of the trellis contains all the possible states and transitions from each state onto the states in the next level. Along with every transition, an observation is emitted simultaneously (in the figure a time unit is crossed and observations vary with time).

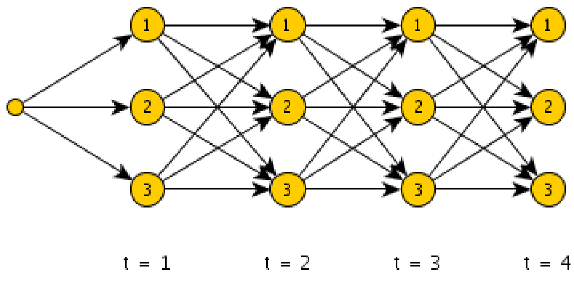

Figure 3.2: An Example Trellis

[2]Source: Prof. Pushpak Bhattacharyya's lecture slides on HMM from the course CS 344 - Artificial Intelligence at IIT Bombay, spring 2013

3.1.6 Formulating the Part-of-Speech tagging problem using HMM

The POS tagging problem can be described as follows. We are given a sentence which is a sequence of words. Each word has a POS tag which is unknown. The task is to find the POS tags of each word and return the POS tag sequence corresponding to the sentence. Here the POS tags constitute the hidden states. As in the urn problem, we again assume that words (balls) are emitted by POS tags (urns), a property called the lexical assumption. That is, the probability of seeing a particular word depends only on the POS tag previously seen. Also, as was the case in the urn problem, the probability of a word having a particular POS tag is dependent only on the POS tag of the previous word (urn to urn probability). Having modelled the problem as given above, we need to explain how the transition tables are constructed. The transition probabilities come from data. This is a data-driven approach to POS tagging, and using data on sentences which are already POS tagged we construct the transition tables. Given this formulation, we next present an algorithm which given an input sentence and the transition tables outputs the most probable POS tag sequence.

3.1.7 The Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states that result in the sequence of observed states. Here the hidden states are the POS tags (or urns in the example) and the observed sequence is the word sequence (ball colours).

The state transition probabilities are known (in practice these are estimated from labeled data) and so are the probabilities of emitting each word in the sentence given the POS tag of the previous word. We start at the start of the input sentence. We define two additional POS tags ˆ and **$** to represent the tag for the start of the sentence and the terminal character at the end of the sentence (full stop, exclamation mark and question mark).

A straight-forward algorithm to find the most probable POS tag sequence (hidden sequence) would be to just try all possibilities starting from the beginning of the sentence. Here, our problem has more structure. We will exploit the Markov assumption we made earlier to get a much more efficient algorithm which is precisely the Viterbi algorithm.

In the trellis for POS tagging problem the following are the major changes to be done.

- The observations (words) do not vary with time. Instead they vary with the position of the pointer in the input sentence.
- The states are the POS tags. The state transition probabilities are pre-computed using a POS-tagged corpus.

Next, we observe that due to the Markov assumption, once we have traversed a part of the sentence, the transition probabilities do not depend on the entire sentence seen so far. They depend only on the previous POS tag. This crucial observation gives rise to the Viterbi algorithm:

Suppose we are given a HMM with S possible POS tags (states), initial probabilities π_i of being in state i, the transition probabilities $P(s_j|s_i)$ of going from state i to j and the emission probabilities $P(x_t|s_i)$ of emitting x_t from the state s_i. If the input sentence is $x_1, x_2, \ldots, x_T$ then the most probable state sequence that produces the sentence $y_1, y_2, \ldots, y_T$is given by the recurrence relations

$$V_{1,k} = P(y_1|s_k)\pi_k \quad (3.1)$$

$$V_{t,k} = P(y_t|s_k)\mathrm{max}_{s_x \in S}(P(s_k|s_x).V_{t-1,x}) \quad (3.2)$$

where $V_{t,k}$ is the probability of the most probable state sequence which emitted the first t words that has k as the final state. The Viterbi path (most likely state sequence) can be remembered by storing back pointers which contain the state s_x which was chosen in the second equation. The complexity of the algorithm is $O(|T||S^2|)$ where T is the set of words, the input sequence and S is the set of POS tags.

3.1.8 Pseudocode

Pseudocode for the Viterbi algorithm is given below:

```
# Given
# Set of states: Array S
# Start state: s0
# End state: se
# Symbol sequence: Array w
# State transition probabilities: Matrix a
# Symbol emission probabilities: Matrix b
# alpha: Matrix alpha

# All indices in arrays start on 1 in this pseudocode

# Returns
# Total probability: p

# Initialisation F1
foreach s in S do
   alpha [1][s] := a[s0][s]*b[s][w[1]]
done

# Induction F2
for i := 1 to length(w)-1 do
  foreach s in S do
    foreach s' in S do
```

```
      alpha[i+1][s] += alpha[i][s′]*a[s′][s]
   done
    alpha[i+1][s] *= b[s][w[i+1]]
  done
done

# Termination F3
foreach s in S do
  p += alpha[length(w)][s]*a[s][se]
done

return p
```

3.1.9 Measuring the quality of output given by the algorithm

Here, we give the mathematical definitions of two terms popularly used in machine learning to evaluate results, namely, precision and recall. These will be used later in analysing the performance of an algorithm in Chatper 6. Since we are interested in POS tagging, we define them for the current setting and skip giving the most general definition possible.

Definition 3. ***Precision:*** *The fraction of words which were correctly tagged is computed as the precision. The precision for any particular tag is defined as the total number of times it was correctly given as the output divided by the total number of tims it was given as output.*

Definition 4. ***Recall:*** *Overall recall is defined as the fraction of correct tags among the total number of words tagged. If every word in the corpus is assigned a tag, then recall equals precision. Recall for a specific tag is defined as the total number of times the tag was correctly given as output divided by the total number of times it should have been given as output.*

In the next section, we present Conditional Random fields simiar to HMMs are used for sequence labeling tasks in NLP. However, they have a different approach to the task.

3.2 Conditional Random Fields

Lafferty, McCallum and Pereira developed conditional random fields in 2001 [1] and proposed them as a probabilistic model for segmenting and labeling sequence data. One of the main advantages of CRFs over HMMs is the ability to relax the strong independence conditions in HMM. In CRFs we can incorporate dependence between words in any part of the sentence.

First we, present what random fields are.

3.2.1 Random Fields

A random field is a generalization of a stochastic process such that the underlying parameter of the process, namely time, need no longer be a simple real value but can take on values that are multi-dimensional vectors.

Definition 5. ***Random Fields***: *Given a probability space, an X-valued random field is a collection of X-valued random variables indexed by elements in a topological space T. That is, a random field F is a collection $\{F_t : t \text{ is in } T\}$ where each F_t is an X-valued random variable.*

A topological space is one that allows for the definition of concepts such as continuity, connectedness, neighbourhood. For our purposes, a rigorous mathematical definition of topological space is not essential and hence will be skipped.

There are many types of random fields. The most common ones are listed here:

1. Markov Random Field
2. Gibbs Random Field
3. Conditional Random Field
4. Gaussian Random Field

A more detailed description of the types of random fields is outside the scope of this thesis. But we will need the description of a Markov random field to understand the definition of a conditional random field and hence it will be discussed in brief here.

A **Markov random field** is derived from the definition of a random field in a way similar to the derivation of a Markov process from a stochastic process. A Markov random field exhibits the Markov property, that is the probability that the random variable assumes a certain value depends on other random variables only through the ones that are its immediate neighbours. Mathematically,

$$P(X_i = x_i | X_j = x_j, j \neq i) = P(X_i = x_i | \partial_i)$$

where ∂_i is the set of neighbours of the random variable X_i in the underlying topological space.

3.2.2 Hammersley Clifford Theorem

Before, we present the formal definition of conditional random fields we will need a fundamental theorem in random fields to proceed. First, we give the definition of the Gibbs distribution (also Gibbs random field).

Definition 6. ***Gibbs distribution***: *A probability distribution $P(X)$ on an undirected graphical model G is called a Gibbs distribution if it can be factorized into* *positive functions defined on cliques that cover all the nodes and edges of G. That is,*

$$P(X) = \frac{1}{Z} \prod_{c \in C_G} \phi_C X_C$$

where C_G is the set of all maximal cliques in G and Z is the normalizing factor $Z = \sum_x \prod_{c \in C_G} \phi_C X_C$.

The statement of the Hammersley-Clifford theorem [8] is given next.

Theorem 1. *The Gibbs random field (Gibbs distribution) is exactly equivalent to the Markov Random field.*

This theorem is used in deriving the functional form of the conditional probability distribution for conditional random fields. In the next section, we formally define conditional random fields and then we go on to study their application to sequence labeling tasks.

3.2.3 The model

Let X be a random variable over data sequences to label, and Y be a random variable over corresponding label sequences. All components Y_i of Y are assumed to range over a finite label set. In the discriminative framework of conditional random fields, the conditional distribution $P(Y|X)$ is constructed from paired observation and label sequences. A formal definition of CRFs is given below.

Definition 7. ***Conditional Random Fields****: Let $G = (V, E)$ be a graph such that $Y = (Y_v)_{v \in V}$, so that Y is indexed by the vertices of G. Then (X, Y) is a conditional random field in case, when conditioned on X, the random variables Y_v obey the Markov property with respect to the graph: $P(Y_v|X, Y_w, w \neq v) = P(Y_v|X, Y_w, w \in N(v))$, where $N(v)$ is the set of neighbours of v.*

CRF is very much similar to a Markov random field. The difference is that the Markov condition should now hold on the conditional probability distribution $P(Y|X)$. Although, G could be any graph, we will be concerned only with sequences. Hence, we restrict G to be a simple chain and X and Y to be sequences. By the Hammersley and Clifford theorem of random fields (Hammersley and Clifford, 1971), the joint distribution of the label sequence Y given X has the form

$$P_\theta(Y|X) = \exp\left(\sum_{e \in E, k} \lambda_k f_k(e, y|_e, x) + \sum_{e \in V, k} \mu_k g_k(v, y|_v, x)\right)$$

where $y|_S$ is the set of components of y associated with the vertices in the subgraph S. The features f and g are known and given. The next step would be to estimate the parameters θ using the training data. This is done using an iterative scaling algorithm similar to the Improved Iterative Scaling algorithm.

It can be seen easily that CRFs encompass the class of HMMs. THe class of CRFs is much more expressive as they allow arbitrary dependencies on the observation sequence. Most sequential classifiers are trained to make the best local decision and hence cannot trade off at different positions against each other. They are myopic about the impact of their current decisions on later decisions.

3.3 Improved Iterative Scaling

Iterative Scaling and its variants are all based on the central idea of the Gradient Descent algorithm for optimizing convex training objectives. It is presented here using a model which occurs at many places in a maximum entropy approach to natural language processing.

3.3.1 The Model in parametric form

The problem we consider is a language modelling problem [12], which is to define the distribution $P(\mathbf{y}|\mathbf{x})$, where **y** and **x** are sequences. For eg, **y** can be the POS tag sequence and **x** the input sequence. Henceforth the boldface indicating that **x** is a sequence will be dropped unless the context demands further elucidation.

Given just the above information, the maximum entropy approach maximises the entropy of the model giving us a model of the following form.

$$P_\Lambda(y|x) = \frac{1}{Z_\Lambda(x)} \text{exp} \left(\sum_{i=1}^{n} \lambda_i f_i(x, y) \right). \tag{3.3}$$

where

- $f_i(x, y)$ is a binary-valued function, called a feature of (x,y), associated with the model. The model given above has n features.
- λ_i is a real-valued weight attached with f_i whose absolute value measures the 'importance' of the feature f_i. Λ is the vector of the weights: $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$.
- $Z_\Lambda(x)$ is the normalizing factor which ensures that P_Λ is a probability distribution.
$$Z_\Lambda(x) = \sum_y \text{exp} \left(\sum_{i=1}^{n} \lambda_i f_i(x, y) \right)$$

3.3.2 Maximum Likelihood

The next thing to do would be to train the model, i.e find the parameters λ_i so as to maximize some objective over the training data. Here, we choose to maximize the likelihood of the training data. The likelihood is computed by assuming that the model is the correct underlying distribution and hence is a function of the parameters of the model. The likelihood of the training data is expressed as follows (N is the number of training instances):

$$\begin{aligned} M(\Lambda) &= \prod_{i=1}^{N} P(x_i, y_i) \\ &= \prod_{i=1}^{N} P_\Lambda(y_i|x_i)P(x_i) \end{aligned}$$

Now, we note that $log(x)$ is a one-to-one map for $x > 0$. Therefore the value of x which maximizes $f(x)$ is the same as that which maximizes $log(f(x))$. Henceforth we work with the logarithm of the likelihood expression as it is mathematically easier to work with. The log-likelihood expression denoted by $L(\Lambda)$ is given below:

$$\begin{aligned} L(\Lambda) &= log(M(\Lambda)) \\ &= \sum_{i=1}^{N} log\left(P_\Lambda(y_i|x_i)\right) + C \end{aligned}$$

where C is independent of Λ and is hence treated as a constant. It is dropped from the expression hereforth as it does not affect the maximization problem.
Now, we express the log-likelihood expression in terms of the empirical probability distribution $\tilde{p}(x, y)$ obtained from the training data as follows:

$$\tilde{p}(x, y) = \frac{c(x, y)}{\sum_{x,y} c(x, y)}$$

where $c(x, y)$ is the number of times the instance (x, y) occurs in the training data. The log-likelihood expression becomes the following:

$$\begin{aligned} L_{\tilde{p}}(\Lambda) &= \sum_{x,y} log\left(P_\Lambda(y|x)^{c(x,y)}\right) \\ &= \sum_{x,y} \tilde{p}(x, y) log\left(P_\Lambda(y|x)\right) \end{aligned}$$

We ignore $\sum_{x,y} c(x, y)$ as it is constant for a given training set ($= N$).

3.3.3 The objective to optimize

Hence we arrive the objective to be maximized. The maximum likelihood problem is to discover $\Lambda^* \equiv argmax_{\Lambda} L_{\tilde{p}}(\Lambda)$ where

$$\begin{aligned} L_{\tilde{p}}(\Lambda) &= \sum_{x,y} \tilde{p}(x,y) log\left(P_{\Lambda}(y|x)\right) \\ &= \sum_{x,y} \tilde{p}(x,y) \sum_i \lambda_i f_i(x,y) - \sum_{x,y} \tilde{p}(x,y) log\left(\sum_y \exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right)\right) \\ &= \sum_{x,y} \tilde{p}(x,y) \sum_i \lambda_i f_i(x,y) - \sum_{x} \tilde{p}(x) log\left(\sum_y \exp\left(\sum_{i=1}^{n} \lambda_i f_i(x,y)\right)\right) \end{aligned}$$

From the expression for log-likelihood we note that $L_{\tilde{p}}(\Lambda) \leq 0$ always. Hence $L_{\tilde{p}}(\Lambda) = 0$ is optimal. TO maximize the log-likelihood we could take the partial derivate of the expression with each parameter λ_i which gives

$$\begin{aligned} \frac{\partial L_{\tilde{p}}(\Lambda)}{\partial \lambda_i} &= \sum_{x,y} \tilde{p}(x,y) f_i(x,y) - \sum_{x,y} \tilde{p}(x) f_i(x,y) \frac{\exp\left(\sum_i \lambda_i f_i(x,y)\right)}{\sum_y \exp\left(\sum_i \lambda_i f_i(x,y)\right)} \\ &= \sum_{x,y} \tilde{p}(x,y) f_i(x,y) - \sum_{x,y} \tilde{p}(x) f_i(x,y) P_{\Lambda}(y|x) \end{aligned}$$

The above expression contains all the parameters λ_i in it and is hence difficult to solve for. Since a direct aproach via differentiation yielded too complex a problem we take an iterative approach similar to the gradient descent algorithm which is described next.

3.3.4 Deriving the iterative step

Suppose we have a model with some arbitrary set of parameters $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. We would like to find a new set of parameters $\Lambda + \Delta = \{\lambda_1 + \delta_1, \lambda_2 + \delta_2, \ldots, \lambda_n + \delta_n\}$ which yield a model of higher log-likelihood. The change in log-likelihood is

$$\begin{aligned} L_{\tilde{p}}(\Lambda + \Delta) - L_{\tilde{p}}(\Lambda) &= \sum_{x,y} \tilde{p}(x,y) log P_{(\Lambda+\Delta)}(y|x) - \sum_{x,y} \tilde{p}(x,y) log P_{\Lambda}(y|x) \\ &= \sum_{x,y} \tilde{p}(x,y) \sum_i \delta_i f_i(x,y) - \sum_{x} \tilde{p}(x) log\left(\frac{Z_{(\Lambda+\Delta)}(x)}{Z_{(\Lambda)}(x)}\right) \end{aligned}$$

Now, we make use of the inequality $-log(\alpha) \geq 1 - \alpha$ to establish a lower bound on the above change in likelihood expression.

$$\begin{aligned}
L_{\tilde{p}}(\Lambda+\Delta) - L_{\tilde{p}}(\Lambda) &\geq \sum_{x,y}\tilde{p}(x,y)\sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x)\frac{Z_{(\Lambda+\Delta)}(x)}{Z_{(\Lambda)}(x)} \\
&= \sum_{x,y}\tilde{p}(x,y)\sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x)\frac{\sum_y \exp\left(\sum_i(\lambda_i+\delta_i)f_i(x,y)\right)}{\sum_y \exp\left(\sum_i \lambda_i f_i(x,y)\right)} \\
&= \sum_{x,y}\tilde{p}(x,y)\sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x)\sum_y\left(\left(\frac{\exp(\sum_i \lambda_i f_i(x,y)}{Z_\Lambda(x)}\right)\exp\left(\sum_i \delta_i f_i(x,y)\right)\right) \\
&= \sum_{x,y}\tilde{p}(x,y)\sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x)\sum_y P_\Lambda(y|x)\exp\left(\sum_i \delta_i f_i(x,y)\right) \\
&= A(\Delta|\Lambda)
\end{aligned}$$

Now we know that is we can find a Δ such that $A(\Delta|\Lambda) > 0$ then we have a improvement in the likelihood. Hence, we try to maximize $A(\Delta|\Lambda)$ with respect to each δ_i. Unfortunately the derivative of $A(\Delta|\Lambda)$ with respect to δ_i yields an equation containing all of $\{\delta_1, \delta_2 \ldots, \delta_n\}$ and hence the constraint equations for δ_i are coupled.

To get around this, we first observe that the coupling is due to the summation of the δ_is present inside the exponentiation function. We consider a counterpart expression with the summation placed outside the exponentiation and compare the two expressions. We find that we can indeed establish an inequality using an important property called the *Jensen's inequality*. First, we define the quantity,

$$f^{\#}(x,y) = \sum_i f_i(x,y)$$

If f_i are binary-valued then $f^{\#}(x,y)$ just gives the total number of features which are non-zero (applicable) at the point (x,y). We rewrite $A(\Delta|\Lambda)$ in terms of $f^{\#}(x,y)$ as follows:

$$A(\Delta|\Lambda) = \sum_{x,y}\tilde{p}(x,y)\sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x)\sum_y P_\Lambda(y|x)\exp\left(f^{\#}(x,y)\sum_i \frac{\delta_i f_i(x,y)}{f^{\#}(x,y)}\right)$$

Now, we note that $\frac{f_i(x,y)}{f^{\#}(x,y)}$ is a p.d.f. Jensen's inequality states that for a p.d.f, $p(x)$,

$$\exp\left(\sum_x p(x)q(x)\right) \leq \sum_x \exp(p(x)q(x))$$

Now, using Jensen's inequality, we get,

$$\begin{aligned}
A(\Delta|\Lambda) &\geq \sum_{x,y}\tilde{p}(x,y)\sum_i \delta_i f_i(x,y) + 1 - \sum_x \tilde{p}(x)\sum_y P_\Lambda(y|x)\sum_i\left(\frac{f_i(x,y)}{f^{\#}(x,y)}\right)\exp(\delta_i f_{\#}(x,y)) \\
&= B(\Delta|\Lambda)
\end{aligned}$$

where $B(\Delta|\Lambda)$ is a new lower-bound on the change in likelihood. $B(\Delta|\Lambda)$ can be maximized easily because there is no coupling of variables in its derivative. The derivative of $B(\Delta|\Lambda)$ with respect to δ_i is,

$$\frac{\partial B(\Delta)}{\partial \delta_i} = \sum_{x,y} \tilde{p}(x,y) f_i(x,y) - \sum_x \tilde{p}(x) \sum_y P_\Lambda(y|x) f_i(x,y) \text{exp}(\delta_i f^{\#}(x,y))$$

Notice that in the expression for $\frac{\partial B(\Delta)}{\partial \delta_i}$ δ_i appears alone without the other parameters. Therefore, we can solve for each δ_i individually. The final IIS algorithm is as follows,

- Start with some arbitrary values for λ_is.
- Repeat until convergence
  - Solve for $\frac{\partial B(\Delta)}{\partial \delta_i} = 0$ for δ_i.
  - Set $\lambda_i = \lambda_i + \delta_i$

  for each i.

This concludes the description of the Improved Iterative Scaling algorithm for optimizing maximum likelihood estimation objectives. Next we look at a powerful classification technique popular in Machine Learning called Support Vector Machines.

3.4 Support Vector Machines

Support vector machines were invented by Vladimir N. Vapnik in the 1963 as a supervised learning model. The basic SVM is a linear binary classifier although both the restrictions can be lifted. First, we look at the motivation for SVMs from a machine learning standpoint. Then the basic linear SVM will be presented from which we can derive non-linear SVMs too.

3.4.1 Motivation

Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a p-dimensional vector, and we want to know whether we can separate such points with a $(p-1)$-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines if known as a maximum margin classifier. The model of support-vector machines achieve precisely the maximum margin classifier.

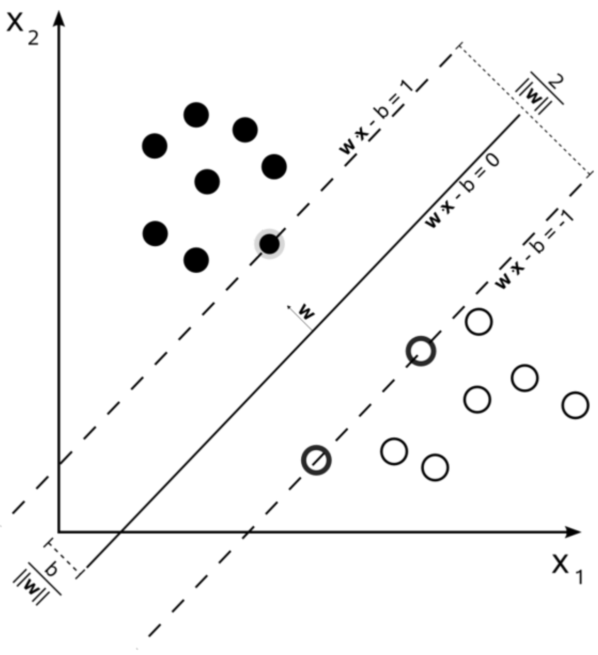

Figure 3.3: An example of maximum margin hyperplane classification over perfectly separable data.

3.4.2 Linear SVM

Given some training data, $D = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}_p, y_i \in \{+1, -1\}\}_{i=1}^n$, we want to find the maximum-margin hyperplane that divides the points having $y_i = 1$ from those having $y_i = -1$. The algebraic expression of a general hyperplane is

$$\mathbf{w}.\mathbf{x} - b = 0$$

where **w** is the normal vector to the hyperplane. If the training data are linearly separable, we can select two hyperplanes in a way that they separate the data and there are no points between them, and then try to maximize their distance. The region bounded by them is called ”the margin”. These hyperplanes can be described by the equations

$$\begin{aligned} \mathbf{w}.\mathbf{x} - b &= 1 \\ \mathbf{w}.\mathbf{x} - b &= -1 \end{aligned}$$

The distance between these two hyperplanes is $\frac{2}{||\mathbf{w}||}$. Hence our aim is to minimize **w**. Also, we want to penalize those data points which fall within the margin. So ideally,

$$\begin{aligned} \mathbf{w}.\mathbf{x} - b &\geq 1 \text{ for data points of the first class} \\ \mathbf{w}.\mathbf{x} - b &\leq -1 \text{ for data points of the second class} \end{aligned}$$

This can be rewritten as

$$y_i(\mathbf{w}.\mathbf{x} - b) \geq 1 \, \forall \, i.$$

If the data is not linearly separable, then finding such a hyperplane is impossible. To still get a 'good' classifier we introduce non-negative slack variables ξ_i, which measure the degree of misclassification of the point $\mathbf{x}_i$.

The final objective function has two terms, one corresponding to the maximization of the margin size, the other corresponding to penalising the misclassified examples (non-zero ξ_i), and the optimization becomes a trade-off between a large margin and a small error penalty.

$$\arg\min_{\mathbf{w},\boldsymbol{\xi},b} \left(\frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi_i \right).$$

subject to

$$\begin{aligned} y_i(\mathbf{w}.\mathbf{x} - b) &\geq 1 - \xi_i \, \forall \, i \\ \xi_i &\geq 0 \, \forall \, i \end{aligned}$$

The constrained optimization problem we have obtained above can be solved by using Lagrange multipliers which convert constrained optimization problems to unconstrained objectives via the use of additional variables. Using Lagrange's method here gives us the objective,

$$\arg\min_{\mathbf{w},\boldsymbol{\xi},b} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \left(\frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}.\mathbf{x} - b) - 1 + \xi_i] - \sum_{i=1}^{N} \beta_i \xi_i \right)$$

where $\alpha_i, \beta_i \geq 0$.

To solve the above problem, we use the technique of Primal-Dual algorithms. That is, we first write its dual. The dual of the above problem is,

$$\arg\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i.\mathbf{x}_j \right)$$

subject to

$$0 \leq \alpha_i \leq C \, \forall \, i \text{ and}$$
$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

The theory of how the dual is generated from the primal is outside the scope of this thesis. The dual problem is convex and hence can be solved by classical convex optimization techniques. From the strong duality theorem we have that the optimal values of the primal and the dual objectives coincide. From a property of duality not presented here, we get that the solution to the primal can be expressed as a linear combination of the training vectors,

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

Since, $\alpha_i \geq 0$ we notice that in the optimization for the primal if $y_i(\mathbf{w}.\mathbf{x}_i - b) > 1$ the corresponding α_is are going to be set to zero. The $\mathbf{x}_i$ whose corresponding α_i are greater than 0 are precisely the support vectors. They lie on the margin and satisfy $y_i(\mathbf{w}.\mathbf{x}_i - b) = 1$.

3.4.3 Non-linear SVM

SVMs as originally proposed by Vapnik are linear classifiers. In 1992, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik [14] suggested a way to create non-linear classifiers by using what is called the 'kernel trick' to maximum-margin hyperplanes. The resulting algorithm is formally similar except that every occurrence of the dot product function is replaced by a general kernel function which can be non-linear but still has to be an inner product. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space which can possibly be high-dimensional. Though the classifier in the transformed space is a hyperplane, in the original input space it maybe non-linear. Using this, non-linear decision boundaries as shown in Figure 3.4 can be generated.

3.4.4 The Kernel function

There can be many kernel functions each leading to different results. For example, the Gaussian radial bias function is used as kernel, then the transformed feature space is of infinite dimensions. Some common kernel functions are listed below.

- Polynomial (homogeneous): $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j)^d$
- Polynomial (inhomogeneous): $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + 1)^d$
- Gaussian radial bias function: $k(\mathbf{x}_i, \mathbf{x}_j) = \text{exp}(-\gamma||x_i - x_j||^2)$ where $\gamma = \frac{1}{2\sigma^2}$.

With that we finish SVMs and we move on to look at a couple of probabilistic frameworks which are based on heuristics. The first is genetic algorithms.

3.5 Genetic Algorithms

Genetic algorithms were inspired by the mechanism of natural selection. They represent a stochastic, discrete-event and nonlinear process. The obtained optimum is an end product containing the best elements of previous generations where the attributes of a stronger individual tend to be carried forward. The rule of the game is survival of the fittest. The basics of the framework are described below. [**?**]

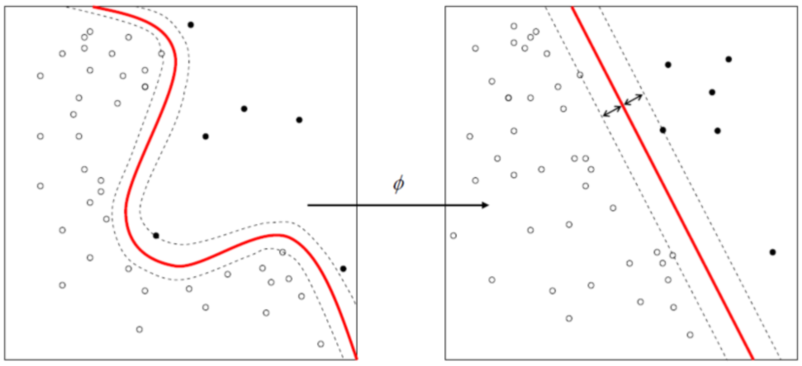

Figure 3.4: An example transformation of input space using kernel functions and the corresponding decision boundaries in the two spaces.

3.5.1 Model

A potential solution for a problem is an individual who can be represented via a set of parameters. These parameters are regarded as the genes of a chromosome; they can be structured by a string of values in binary form. A positive value called the fitness value, is used to reflect the degree of goodness of the chromosome for solving the problem, and this value is closely related to its objective value. A fitter chromosome has the tendency to yield good-quality offspring. A population pool of chromosomes has to be installed and they can be randomly set initially. The size of this population varies from one problem to the other.

In each cycle of the genetic process, the subsequent generation is created from the chromosomes of a set of parents in current generation called the mating pool who are selected via a selection routine. There can be multiple selection routines. A popular one called the *Roulette wheel selection* is presented next.

The cycle of evolution is repeated until a desired termination criterion is reached. This criterion can also be set by the number of evolutional cycles, the amount of variation of individuals between generations, or a predefined value of fitness.

3.5.2 Mating Pool Selection

One of the most popular routines to select the parents for the mating pool is presented below.

Roulette Wheel parent selection:

- Sum the fitness of all the population members and call it total fitness (N).
- Choose a random number n between 0 and total fitness (N).
- Return the first population member whose fitness added to the fitness of the preceding population members is greater than n.

Next we look at genetic operators. These operators act on chromosomes and give rise to the next pool of chromosomes (or offspring). These operators result in a change in the fitness value of the chromosomes in the pool and hence are ultimately responsible for finding the optimal chromosome.

3.5.3 Genetic Operators

The framework consists of two fundamental operators, namely, crossover and mutation. Crossover is responsible for the creation of new offspring whose chromosome is a mix of the chromosomes of its parents. Mutation is a unary operator which when applied on a chromosome probabilistically changes its structure.

The Crossover Operation: Given two parents from the mating pool, the crossover operator partitions the chromosome length into partitions of random sizes and probabilistically exchanges the subchromosomes within a partition between the parents. The new chromosome generated at the end of the crossover operation is called the child chromosome. An example of a single-point crossover (the chromosomes are split into two partitions only) is shown in Figure 3.5.

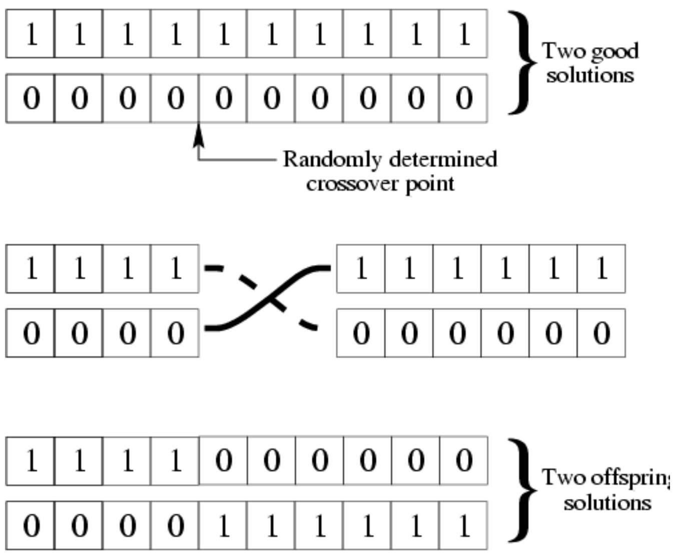

Figure 3.5: A depiction of the crossover operator. Here only a single-point crossover is shown.

The Mutation Operation: The mutation operation changes each bit in the chromosome with some probability. If each bit is allowed to take on values from a set of size more than 2, then the mutation operator might end up exchanging two bits in the chromosome, replacing a bit by another at a different position, creating a new bit with no relation to existing bits et cetera. However, since we are only concerned with chromosomes whose bits take binary values, for our purposes the mutation operator does the following. *It flips each bit of the chromosome with a certain probability.* An example of mutation is shown in Figure 3.6. Although, in the figure only a single bit is shown to be flipped, in practice multiple bits can get flipped via a single mutation operation.

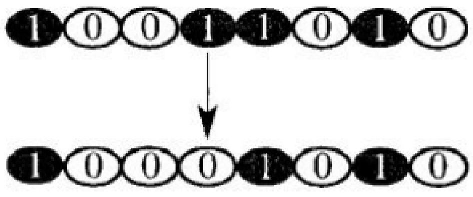

Figure 3.6: A depiction of the mutation operator. Here only a single bit is shown to be mutated.

Choosing the probability parameters for both the operators can be a complex, non-linear optimization problem. This selection issue continues to remain largely open but some guidelines exist as to how to choose the parameters given the population size.

Both the operators do not necessarily always improve the population. The improvement is enforced by the parent selection and the survivor selection routines. However, the apparently random nature of the operations is what offers genetic algorithms to overcome a serious limitation of many other optimization frameworks. They do not get trapped at local optima in contrast to some gradient descent algorithms. Even when a local optima is achieved a genetic algorithm will produce some offspring which explores other possibilities for the optima beyond the current local optima.

3.5.4 The Algorithm

Given the above two operators, and the corresponding parameters this is how the algorithm proceeds. It first generates an initial population of chromosomes of size N via some method. The fitness of this population is evaluated and stored. Now, until the termination criterion is met, the algorithm repeatedly does the following:

- Select the mating pool from the parent population.
- Recombine the genes of the parents in the mating pool using crossover to generate a new child population.
- Perturb the genes of the child population via the mutation operator.
- Evaluate the fitness of the child population.
- Select the chromosomes with the best N fitness values among the parent and child populations as survivors for the next generation.

The pseudocode for the algorithm is given below.

Pseudocode:

```
Standard Genetic Algorithm()
{
//start with an initial value
t = 0;
initpopulation P(t);
evaluate_fitness P(t);
//test for termination criterion (time, fitness etc)
while(not done)
{
t = t+1;
P′ = selectparents P(t);
//recombine the genes of the parents using crossover
recombine P′(t);
//stochastically mutate (perturb) the mated population
mutate P′(t);
evaluate_fitness P′(t);
//select the survivors (best N fitness values)
P = survive P, P′(t);
}
}
```

3.5.5 The building block hypothesis

To understand why genetic algorithms work, the building block hypothesis was proposed.

Definition 8. ***The Building Block Hypothesis****: A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low -order, high performance schemata, called the building block. (short sub-sequences of binary bits)*

The genetic operators, crossover and mutation have the ability to generate, promote, and juxtapose(side by side) building blocks to form the optimal strings. Crossover tends to conserve the genetic information present in the strings. Thus, when the strings for crossover are similar, its capacity to generate new building blocks diminishes. Mutation, however, is not a conservative operator but capable of generating new building blocks radically.

In addition, parent selection is an important procedure to be devised. It tends to bias toward building blocks with higher fitness values, and at the end ensures their representation from generation to generation.

3.5.6 Limitations

Genetic algorithms have their share of limitations. The first among them was already touched upon. The question of what parameters to choose for the operations of crossover and mutation does not have a satisfactory answer and it appears that getting to the answer is complex and might be inefficient too. The second major concern is that of efficiency of the algorithm. Genetic algorithms might perform well on the average but in the worst case they can take large amounts of time to improve the solution.

In the next section, we look at another probabilistic heuristic for optimization which was inspired from a metallurgical process.

3.6 Simulated Annealing

Simulated annealing (SA) [7] is a probabilistic method proposed by Kirkpatrick, Gelatt and Vecchi and Cerny in the 1980s for finding the global minimum of a cost function that may possess several local minima. It works by emulating the physical process whereby a solid is slowly cooled so that eventually when its structure is frozen; this happens at a minimum energy configuration. For our purposes, energy would mean the objective function we are trying to optimize.

3.6.1 The Elements of Annealing

The basic elements of simulated annealing are the following:

1. A finite set S.
2. A real valued cost function J defined on S. Let $S \subset S$ be the set of global minima of J, assumed to be a proper subset of S.
3. For each $i \in S$, a set $S(i) \subset S - \{i\}$ called the set of neighbours of i.
4. For every i, a collection of positive coefficients $q_{ij}, j \in S(i)$, such that $\sum_{j \in S(i)} q_{ij} = 1$. It is assumed that the neighbourhood relation is symmetric. That is, if i is a neighbour of j, then j is a neighbour of i.
5. A non-increasing function $T : \mathbb{N} \to (0, \infty)$ called the cooling schedule. Here $T(t)$ is the temperature at time t.
6. An initial state $x(0) \in S$.

This is a complete and abstract description of the model which can be applied to problems beyond thermodynamics too, in particular to optimization problems in computer science. Next, we look at the heuristic of simulated annealing and how it solves the energy minimization problem.

3.6.2 The Method

Given the above elements, the SA algorithm consists of a discrete-time inhomogeneous Markov chain $x(t)$, which evolves as follows.

1. If the current state $x(t)$ is equal to i, choose a neighbour j of i at random. The probability that a particular j will be chosen is q_{ij}.

2. Once j is chosen, the next state is determined as follows.

   - If $J(j) \leq J(i)$ then $x(t+1) = j$
   - Otherwise, $x(t+1) = j$ with probability $exp(\frac{J(i)-J(j)}{T(t)})$ and $x(t+1) = i$ with the remaining probability.

Hence, the SA algorithm can be viewed as a local search algorithm in which there are occasional 'upward' moves that lead to a cost increase. These moves help the algorithm escape from local minima. This is shown pictorially in Figure 3.7. The function $T(t)$ called the cooling schedule is crucial in deciding the performance of SA for a problem. From the probability expression above, we can see that lower temperatures imply smaller chance of perturbation. Hence as we near our optima, the temperature should fall down so that we converge at that state.

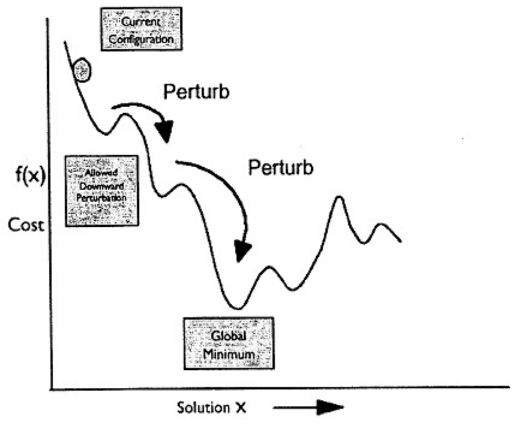

Figure 3.7: A depiction of the process of simulated annealing. Perturbations at local minima help it escape them to reach a global minima.

3.6.3 Performance

SA has gained popularity among researchers mainly due to its speed of convergence. Despite the lack of a rigorous theoretical justification for the convergence speed, SA has grown in popularity and widely used in image processing. In a comprehensive study of SA, Johnson et al. (1990, 1991, 1992) present the performance of SA in combinatorial optimization problems. In general, its performance was mixed. In some problems such as the traveling salesman problem, SA outperformed the best known heuristics. In other cases, such as the graph partitioning problem, specialized heuristics performed better. In the case of the graph coloring problem there was no significant difference between the solutions given by SA and specialized heuristics.

We now conclude this chapter. We have looked at various classical techniques used for search and optimization, all the time having the main motive of developing quantum techniques in the back of the mind. This broad exploration of classical techniques was essential because it is these techniques that we would like to improve using the power of quantum mechanics and quantum computing.

We next go on to look at the development of quantum computing ideas in literature by studying some popular and landmark quantum computing algorithms.