Stochastic Feature Selection

Instructor: Professor Felisa Vazquez-Abad
PhD Student: James Diotte, Soumi Maiti, Jiaxing Tan and Viet Anh Trinh

January 2, 2018

1 Introduction

1.1 Overview

Feature selection is a common technique used in the creation of models in machine learning and statistical applications. Feature selection is a type of dimensionality reduction which seeks to find a subset on which to project the data, i.e. it is a function $\Pi : \mathbb{R}^p \to \mathbb{R}^{p'}$ where $p' < p$ and $\Pi(x_1, x_2, \ldots, x_p) = (x_{i_1}, x_{i_2}, \ldots, x_{i_{p'}})$. Feature selection is important, especially in the case where are data has very high dimension, as too many features can cause problems in our models such as:

- Increased training time
- Curse of dimensionality
- Overfitting on noisy features

The goal of this project is cast feature selection as a discrete optimization problem (DOP) and then use methods from stochastic optimization to get a approximate solutions. The general form of a DOP is to find:

$$s_* = \underset{s \in \{s_1, s_2, \ldots, s_N\}}{\text{argex}} C(s)$$

Where argex is a stand in for either argmax or argmin, C we will refer to as a cost function, the set $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ we will refer to as the search space and a element of it as a state. The size N of the search space can be extremely large making the naive approach of evaluating of all possible states intractable, hence this is where stochastic methods become desirable.

1.2 Feature Selection as a DOP

When developing machine learning algorithms for classification applications it is necessary to have a robust technique for benchmarking the performance of a model. One of the most common techniques to do this is to split all the available labeled training data into two disjoint sets known as a training and testing split. The model in question is then fit using the training data and the trained model makes predictions on the test set which are then compared to the known labels. Common measures of performance used when comparing test predictions to labels include overall accuracy, precision,

recall, F1 score, balanced error rate (BER) and area under the ROC curve (AUROC). Now to set up the DOP assume we have disjoint training and testing datasets for a binary classification problem: $\mathcal{D}_{trn} = \{(x_1^i, x_2^i, \ldots, x_p^i, y^i\}_{i=1}^{D_1}$ and $\mathcal{D}_{tst} = \{(x_1^j, x_2^j, \ldots, x_p^j, y^j\}_{j=1}^{D_2}$, a black-box machine learning algorithm f, and a performance metric P. Our search space will consist of all non-empty subsets of features, i.e. non-empty subsets of $\{1, 2, \ldots, p\}$ there are $N = 2^p - 1$ such subsets and we can mathematically represent elements of this set as binary vectors $\vec{b} = (b_1, b_2, \ldots, b_N) \in \{0,1\}^N \setminus \{\vec{0}\} = \mathcal{S}$. Now given $\vec{b} \in \mathcal{S}$ we will define out feature selection map as $\Pi_{\vec{b}}(x_1, x_2, \ldots, x_p, y) = (x_{i_1}, x_{i_2}, \ldots, x_{i_{p'}}, y)$ such that $\{i_1, i_2, \ldots, i_{p'}\} = \{i \mid b_i = 1\}$ and our cost function evaluated at $\vec{b}$, $C(\vec{b})$, will be the performance metric P of f trained on $\Pi_{\vec{b}}(\mathcal{D}_{trn})$ and tested on $\Pi_{\vec{b}}(\mathcal{D}_{tst})$. Thus our feature selection DOP becomes:

$$\vec{b}_* = \underset{\vec{b} \in \mathcal{S}}{\text{argex}} \; C(\vec{b})$$

2 Algorithm Overview

2.1 Discrete Stochastic Approximation

In order to solve the DOP above we employ various methods from discrete stochastic approximation. All of our methods can broadly be considered random search methods as none of them use the gradient, approximate the gradient or even assume differentiability of the cost function. Our approaches will further be split into local search methods (Tabu search, hill climbing and simulated annealing) and population based evolutionary methods (genetic algorithm and particle swarm optimization). Local search methods work with one candidate solution at a time, the candidate is updated by searching for a better candidate in a neighborhood. The neighborhood and selection criteria are algorithm specific. Evolutionary algorithms, unlike local search methods employ a collection of candidate solutions which are updated according to both local (individual candidate) and global (entire collection) attributes.

2.2 Tabu Search and Revised Algorithm

2.2.1 Tabu Search

Tabu search, created by Fred W. Glover in 1986[4] and formalized in 1989[5], is a metaheuristic search method employing local search methods used for mathematical optimization. As shown in 1, it performs a local or neighborhood search procedure to iteratively move from one potential solution s to an improved solution $s^{'}$ in the neighborhood of s, (denoted as $Neib(s)$), until some stopping criterion (denoted as SC) has been satisfied (generally, an attempt limit or a score threshold)[5][6].

Based on the convergence proof in [8], a tabu search method will converge if $getNeighbors()$ method works as follow:

1. If all neighbor solutions of x_k are already visited i.e. $N(x_k) \subseteq S_k$, then Select the earliest solution x_i^* examined. Update the last time the solution x_i^* has been visited. $S_{k+1} = S_k - x_i^* + x_{k+1}$
2. Otherwise, Select an unvisited solution $x_{k+1} \in (N(x_k) - S_k)$ with respect to certain criterion; Update the trajectory $S_{k+1} = S_k + x^{k+1}$

Algorithm 1 Tabu Search

```
sBest ← s_0
bestCandidate ← s0
tabuList ← [ ]
while not stoppingCondition() do
    sNeighborhood ← getNeighbors(bestCandidate)
    bestCandidate ← sNeighborHood.firstElement
    for sCandidate ∈ sNeighborhood do
        if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
then
            bestCandidate ← sCandidate
        if fitness(bestCandidate) > fitness(sBest) then
            sBest ← bestCandidate
        tabuList.push(bestCandidate)
        if (tabuList.size > maxTabuSize) then
            tabuList.removeFirst()
return sBest
```

However, in reality, the searching task is to find needle in haystack, where the optimal solution usually is a subspace contained a few or one point while the whole space is in the level of $O(m^n)$. In such situation, although the algorithm can converge, as has been proved in the related works, it takes quite a long time for the algorithm to converge, which ususally is undurable.

Based on experiment, we give some possible explanations on its bad performance:

- Searching space is too huge while the target is a tiny set lies in it.
- Neighbour selection is very inefficient which is measured by edit-distance and could be highly relied on starting point.
- The starting point is chosen randomly, which, considering the difficulty of the task, is highly unlikely to be a good one.

Considering all these points, we proposed a revised tabu search algorithm specifically for feature selection, which we will discussed in the next section.

2.2.2 Revised Tabu Search

To improve the performance of Tabu Search in the task of feature selection, we made the changes as below:

- We fixed the number of features to be selected as h.
- Our object is to estimate a multinomial distribution $P(f_i|b^*)$, which estimates the probability that feature f_i occurs in the best feature set b^*.
- Instead of randomly choosing a starting point, we choose a relatively optimal one based on a global search algorithm.

- For neighbourhood search, or local search in contrast to the global search, we design a sampling algorithm to estimate the distribution $P(f_i|b^*)$.

The revised algorithm could be seen in 2. The algorithm keeps track of a table $C(\vec{F})$ and update it according to the generated feature set. The stopping criteria is decided by how many features are from feature importance list returned by a feature selection algorithm and epoched running under each feature selection setting. In the rest of this chapter, we will introduce each part accordingly.

Algorithm 2 Revised Tabu Search For Feature Selection

```
Data: Feature Selection Task Dataset D
FeatureImportance ← GlobalFeatureAnalysis(D)
FeatureFromAnalysis ← 100
TotalFeature ← 300
Initialize Table C(F⃗) as Count of each features
while FeatureFromAnalysis > FeatureThreshold do
    s0 ← FeatureGeneration(FeatureImportance, FeatureFromAnalysis)
    bestCandidate ← s_0
    tabuList ← [ ]
    for Epoch from 1 to EpochThreshold do
        s1 ← FeatureGeneration(FeatureImportance, FeatureFromAnalysis)
        Added = AddComp(s_0, s_1)
        Removed = RemoveComp(s_0, s_1)
        UpdateTable(s0, s1)
        s_0 = s_1
    FeatureFromAnalysis ← FeatureFromAnalysis − 10
return bestCandidate, C(F⃗)
```

2.2.3 Global Searching Algorithm

The motivation of using a global searching algorithm is to find a starting point that has a high probability to find the optimal solution. To achieve this, we design to use feature selection algorithm to evaluate all features and return a feature importance list. This list provides a direction where the optimal solution most likely lies in considering the whole space. Then we can perform a local search and it will converge much faster.

There are many feature selection methods. In this paper, we applied Random Forest [1]. A random forest is an ensemble learning method that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The trained random forest model could return a list of feature importance.

2.2.4 Feature Set Generation

The feature set generation algorithm is illustrated in 3. Each time a feature set is generated by combining top k features from the feature ranking list returned by the global searching algorithm mentioned in the previous section while the rest is based on sampling.

And the sampling algorithm contains two part. The first part is to sample based on the count in table $C(\vec{f})$ based on 1 with probability ϵ. Otherwise randomly sampled one based on an uniform

distribution.

$$P(f_i) = \frac{C_i}{\sum_{i=1}^{n} C_i} \tag{1}$$

Algorithm 3 Algorithm for Feature Generation

```
function FeatureGeneration(FeatureImportance,FeatureFromAnalysis)
    NewSet ← [ ]
    k ← FeatureFromAnalysis
    NewSet ← top k features from FeatureImportance
    while NewSet < h do
        Randomly Generate a number q
        if q > threshold then
            Sample one based on count in table C(f⃗)
        else
            randomly pick one
    return NewSet
```

2.2.5 Fitness Score

As a searching algorithm, we have to specify the Fitness Score function which gives a score as the evaluation of current searched point. Specifically, we used the Balanced Error Rate (BER) as our fitness score considering the fact that our dataset is highly imbalanced. The equation for BER is shown in 2, where TP, TN and FN stands for True Positive, True Negative and False Negative accordingly. As our measurement is the error rate, the lower the fitness score, the better the searched point is.

$$1 - \frac{1}{2}(TP/(TP + FN) + TN/(TN + FP)) \tag{2}$$

The fitness score is calculated by the classifier built based on the given feature set. We use logistic regression as our classifier.

2.2.6 Updating Rule

As our revised Tabu Search algorithm is a probabilistic algorithm, we have to specify the updating rule of the sampling procedure. We compare the performance of the current searched point with the previous one by comparing the fitness score. And we keep track of two sets, the first set is new features added comparing to the previous one, the other is features removed.

The updating idea is, if the new searched point s_1 is better than the previous point s_0, then all the added feature might be more likely to be contained in the b^*, while the removed ones are less likely to be contained. If s_1 has worse result, the added ones should have low probability with the removed ones having high probability.

Based on the criteria described above, we update corresponding $C(\vec{F})$ with count of 1 each time according to the comparison result, added feature list and the removed feature list.

Algorithm 4 Algorithm for update table

```
function UpdateTable(s0,s1)
    if (fitness(s1) < fitness(s0) then
        bestCandidate ← s1
        for f_i ∈ Added do
            C(f_i)+ = 1
        for f_i ∈ Removed do
            if (C(f_i) > 0) then
                C(f_i)− = 1
    else
        for f_i ∈ Removed do
            C(f_i)+ = 1
        for f_i ∈ Added do
            if (C(f_i) > 0) then
                C(f_i)− = 1
```

2.3 Stochastic Hill Climbing

Stochastic hill climbing (SHC) is a probabilistic variant of deterministic hill climbing. As the term hill climbing suggests, if we view an optimization problem as a landscape in which each point corresponds to a solution and the height of the point corresponds to the fitness of the solution, F(s), then hill climbing aims to ascend to a peak by repeatedly moving to an adjacent state with higher fitness. It is an optimization technique which belongs to the family of local search. Hill climbing is relatively simple to implement, making it a popular first choice in family of random search algorithms.

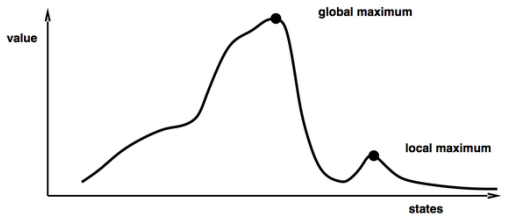


Figure 1: Hill Climbing

It is an iterative procedure where we start at a initial state and at each state locally search the neighborhood of the current state for better fitness. Hence, it is also a greedy search algorithm as at each step it selects the best possible solution at that state. Algorithm repeats iteratively until converges or for a very high number of iterations. Conceptually fitness of the states can be thought of surface of the hill and we climb through the hill towards the peak. Hill climbing is a simple local search algorithm, the algorithm cannot see the landscape all at once. Only the neighborhood is visible at each step, more similar to the idea of climbing in a fog.

Algorithm 5 Stochastic Hill Climbing: Algorithm

```
1: cur = Initial solution
2: while i < max iteration do
3:     nbr = Randomly select a neighbor
4:     if  Fitness(nbr) > Fitness(cur) then
5:         cur = nbr
```

There are mainly two different family of hill climbing algorithms. First is simple hill climbing,

where we choose the first neighbor that gives us a better solution than current. This is simple and much faster as we do not have to look through every possible neighbors. The second is steepest hill climbing, where at each step we look through all the neighbors and select the best solution. This method is also called steepest ascent hill climbing or gradient search. This generally provides better results but takes higher time to converge. It is similar to the idea of steepest gradient descent, where at each step we choose the steepest direction. Instead of calculating gradient we are estimating the steepest direction by looking through the neighbors.

As a discrete optimization problem it can be formulated as finding maximum of an objective function F in a feasible region or state space Θ. Where $F : \Theta \to \mathbb{R}$ is the fitness function or objective function. For each $\theta \in \Theta$, there exists a set $N(\theta) \subset \Theta \setminus \theta$ consisting of the neighbors of the point θ. Also the feasible set Θ is connected, so it is possible to go from any point in Θ to any other point in Θ by moving through neighboring points. So we start with a initial state θ_0 and iteratively for each step $m > 0$ generate candidate solutions.

The structure of neighborhood is a design parameter. Depending on the problem we can decide how the neighborhood of each state looks like. The efficiency of Stochastic Hill climbing is highly influenced by neighborhood function used. The choice of neighborhood enforces a topology on the problem. It has been proven that a neighborhood structure is more preferable if it imposes a 'smooth' topology instead of a 'bumpy' one [9]. This is also true for other generalized class of random search algorithms like simulated annealing[12]. For our particular application to feature selection, it is difficult to measure what kind of neighborhood structure would provide a smoother topology. We can formulate the problem of selecting K features from available D features as a binary string of size D, where only K positions have 1. In the binary string, having 1 at a feature position identifies selecting that feature and having 0 signifies removing it. Size of D for datasets used are of order of several thousands.

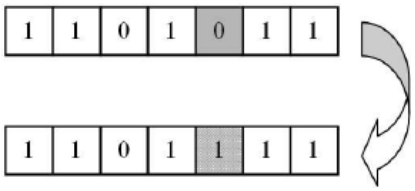


Figure 2: Selecting a neighbor

Hence, each state θ is a binary string of size D having some number of ones(It can't be all zeros as we want to select some features.). Then neighborhood of θ, $N(\theta)$ can be defined as similar strings to θ. We experimented with some different choices for similarity measures for neighborhood structure. From the results, we will see that, the choice of neighborhood structure has significant effect on the performance of the algorithm.

The deterministic version hill climbing has the problem of being stuck at local optima because of its greedy approach. Stochastic version randomly samples some neighbors as candidates, Hence the the evaluation of fitness will always have some noise in them. This amount of random noise prevents the algorithm from falling into traps like saddle point and local optima.

2.4 Simulated Annealing

Simulated Annealing is another local search method capable of escaping local optima. This algorithm is inspired with the idea of annealing metal or glass or crystal. For annealing generally metal or glass are heated with a very high temperature and then the temperature is lowered slowly until metal is in a

freezing state.

Simulated annealing starts with a random solution. At each iteration looks in the neighborhood and evaluate candidate neighbor's fitness. If there is a better solution in any of the neighbors it accepts that solution. But even if there is no better solution algorithm can choose to move with a small probability. This small probability is controlled by temperature T. We start with a high temperature, hence probability of accepting worse solutions are high. As we decrease temperature through a cooling schedule, probability of accepting bad solutions decreases. The algorithm enters a freezing stage and moves to a worse solution with very less probability.

Algorithm 6 Simulated Annealing Algorithm

```
1: cur = Initial solution
2: T = Initial temperature
3: while T > T_min do
4:     nbr = Randomly select a neighbor
5:     if F(nbr) > F(cur) then
6:         cur = nbr
7:     else
8:         cur = Nbr with small probability
9:     reduce T
```

There are three functions that plays fundamental role: accept, generate and update. The accept probability depends on the temperature T and comparative fitness of current and candidate state. The pseudo-code of the accept algorithm is described in Algorithm 7. So if fitness of a neighbor is higher than current, it is accepted. Even if the fitness is not higher, depending on T algorithm can move. This move allows the algorithm to jump out of local optima.

Algorithm 7 Simulated Annealing: acceptance probability

```
1: F(i) = Fitness of current solution
2: F(j) = Fitness of neighbor solution
3: T = current temperature
4: Δ_ij = F(j) − F(i)
5: y = min[1, exp(−Δ_ij / T)]
6: r = random(0, 1)
7: if r ≤ y then
8:     Accept the solution
9: else
10:    Reject the solution
```

The generate function generates new candidate solutions. New solutions are generated by looking through neighbors and their fitness. Update function can also be called as annealing schedule or cooling schedule. At each step update function generates a new value for temperature. For Simulated Annealing we want the update schedule to return monotonic decreasing values of T, i.e. for iteration(m) > 0,

$T_{m+1} < T_m$ and $\lim_{m\to\infty} T_m \to 0$. In practice different cooling schedules are used, linear decrement, logarithmic decrement and decreasing by a fixed value. In theory, not all of the update function converges to global optima.

Simulated annealing can be represented by a Markov chain, the connectivity of the chain is represented by the generate function and transition probabilities are determined by accept and generate functions [14]. Let's assume the underlying graph is G. The graph is connected though neighborhood(N) structure and is a directed graph. We also assume cooling schedule to be monotonically decreasing such that,

$$T_m > T_{m+1}, \forall m \geq 0$$
$$\lim_{m\to\infty} T_m = 0$$

Given two elements i and j, where $j \in N(i)$ the probability of generating j from i is given by $\frac{g(i,j)}{g(i)}$. $g(i,j)$ provides the weights of each neighbor of i and $g(i)$ is a normalizing function to ensure that,

$$\frac{1}{g(i)} \sum_{j\in N(i)} g(i,j) = 1$$

Then one-step transition probabilities of the Markov chain can be calculated as,

$$P_{ij}(T) = \begin{cases} \frac{g(i,j)}{g(i)} \min[1, \exp(\frac{-F(j)-F(i)}{T})], & \text{if } j \in N(i). \\ 0, & \text{if } j \notin N(i). \end{cases}$$
$$P_{ii}(T) = 1 - \sum_{j\in N(i)} P_{ij}(T)$$

The Markov chain is time-inhomogeneous wrt to decreasing temperature (T_m). However if temperature is frozen at a particular value T, then we have time-homogeneous Markov chain. Time inhomogeneous Markov chain also has a quasi-stationary probability distribution.

$$\pi_i(T) = \frac{g(i)\exp(\frac{-c(i)}{T})}{G(T)}$$

With $G(T)$ as a scaling factor such that $\left\|\pi(T)\right\| = 1$. Time inhomogeneous Markov chain can be proven to be strongly ergodic with some assumptions on cooling schedule. Proving strongly ergodicity proves convergence.

With simulated annealing with update function,

$$T_m = \frac{\gamma}{\log(m + m_0 + 1)}, m = 0, 1, 2, ...$$

where m_0 is any parameter satisfying $1 \leq m_0 \leq \infty$. If $\gamma \geq rL$, where r is the radius of the underlying chain and L is the Lipschitz-like constant of the fitness function, Markov chain is strongly ergodic.

So sequence of the variables $\{\theta_m\}$ converges in probability to one of the global optimal solutions of

Θ^* as we run for infinite iterations.

$$\lim m \to \infty P\{\theta_m \in \Theta^*\} = 1$$

With logarithmic cooling schedule convergence can be proven. Moreover in order for the method to converge, it is necessary for the method to be become progressively more and more conservative as the number of iteration grows. This allows for the algorithm not to move away from promising points unless there is a strong evidence that this move will improve. As the number of iteration grows temperature decreases and this helps the algorithm to freeze.

2.5 Genetic Algorithms

One approach to stochastic optimization is Genetic algorithms (GAs), which relate to the problem of finding the best solution among multiple local minima. [3]. Instead of finding vector θ that minimize cost function $J(\theta)$, GAs finds the vector θ that maximize the fitness function $F(\theta)$:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} F(\theta) = \underset{\theta}{\operatorname{argmax}} \frac{1}{J(\theta)} \tag{3}$$

GAs initialize with a set of candidates θ and on iteration, this set is moving toward a global optimum. The specific cost function used for feature selection is discussed in section 3.2 DOP Set Up

2.5.1 Related Works

The pioneer in the theoretical analysis of GAs is schema theory [10] . However, the mathematics of this theory is not fully rigorous and is challenged in [2] . This lead to the second approach which uses the application of stochastic models based on Markov chain theory. [16] show that a canonical GA without elitism might not converge. GAs with elitism means that we keep the best individual to next generation, instead of doing crossover and mutation for all individual because the individual corresponding to θ^* might be lost. However, [17] proved that GAs with elitism converges in probability to set of optima. This is the algorithm we use for the feature selection project.

2.5.2 Genetic Algorithms with elitism

Algorithm 8 GAs with elitism

Input: Size of population N, number of iterations n , probability of cross over, probability of mutation η

Output: Candidates θ^*

function GA

Generate N individual θ randomly in population
count = 0
while $count < numberofiteration$ **do**
Calculate the fitness $F(\theta)$of all individuals θ
Sort individual by descending order of fitness $F(\theta)$.
The 1 best individual θ comes directly to next generation
Select N - 1 individuals by Roulette Wheel Selection on N individuals
Do cross-over on these N - 1 individuals with probability 1.
Do mutation on these N - 1 individuals with probability η
Add these N - 1 individuals to next generation
count := count + 1

Our purpose is to find the individual - vector θ to minimize the cost function $J(\theta)$, or say in another say, maximize the fitness function $F(\theta)$. Every individual θ is a vector, with the same dimension L as the dimension of the dataset. Therefore, every individual has L bits, a bit has value 0 if the feature is not selected and value 1 if this feature is selected.

Initialization:

In the initialization, the GAs algorithm randomly generated N vectors (individual) with length L: $a_{i1}a_{i2}...a_{iL} \in A^L$ where A = $\{0, 1\}$. Each individual can be represented as an integer $\sum_{j=1}^{L} a_{ij}2^{L-j}$. Every individual has L bit, so there are total 2^L different individuals. Each population k, is indicated in the vector $(Z(0,k), Z(1,k), Z(2,k)...., Z(2^L-1,k))$, where Z(i,k) denotes the occurrences of the individual labeled as $i = 0, 1, ..., 2^L - 1$ in the population k. The cardinality of the different populations is derived as [17]:

$$C = \binom{N + 2^L - 1}{N} \tag{4}$$

The new population is generated after an iteration. This is a stochastic process based on the population of each generation. Iterating the generation changes n times results in a Markov chain which generates a population labeled as k = 0,1,2...C after n generation changes

Selection:

We do Roulette Wheel Selection, in which a proportion of the wheel is assigned to each of the possible individual based on their fitness value. Every individual has a probability to be chosen as:

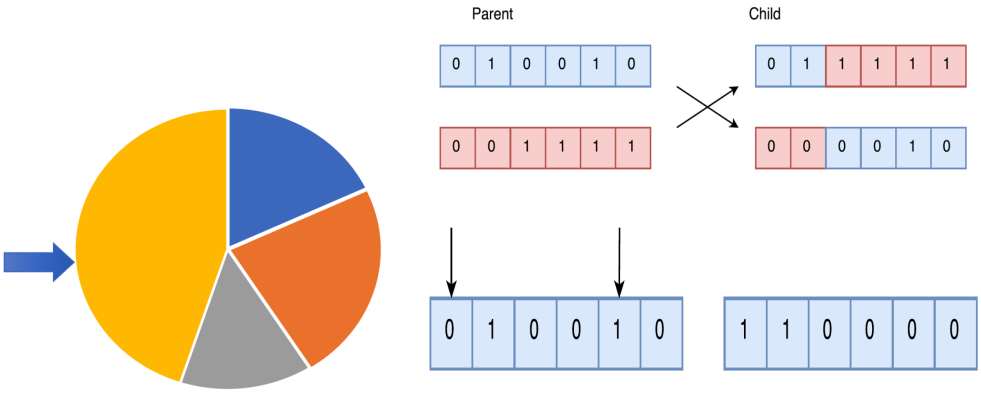


Figure 3: Roulette Wheel Selection, cross-over and mutation

$$p(i,k) = \frac{f(i)Z(i,k)}{\sum_{h=0}^{2^L-1} f(h)Z(h,k)} \tag{5}$$

Cross-over:

In elitism GAs, we reserve the best $N_e$individuals and do cross-over and mutation on the rest $N - N_e$. In this paper, we reserve only 1 best individual ($N_e = 1$)and do cross-over and mutation on the rest N-1 individuals.

Cross-over generates two new children from two individuals ("parents") . First, a random number $r \in (0, 1)$ is generated , if this number is smaller than the cross-over probability $r \leq 1$ then we do cross-over, otherwise, the two children are the same as the parents. In this case, cross-over probability is chosen as one for simplicity. We exchange the $1 \leq l \leq L$ rightmost bits of the two parents individuals, where l is a random integer from [1,L-1] to create the children

Mutation:

Every individual is a vector of L bits. Every bit has a probability to be flip as η. For every bit, we generate a random number r from (0,1), if $r < \eta$ then this bit is flipped $0 \to 1$ or $1 \to 0$

2.5.3 Markov Chain analysis

[18], [15] demonstrated that the stochastic transition through these genetic operations can be described by the transition matrix $P = (P_{k,v})$ of size C x C for one generation, where the conditional probability that population v is generated from population k is $P_{k,v}$

Denote $p_k^{(n)}$ is the probability that the Markov chain is in population k after n iterations

Denote a set of populations, in which every population includes the individual with the highest fitness is $\mathcal{K}$, in order to prove convergence, we investigate the elitism method satisfies :

$$\sum_{k \in \mathcal{K}} p_k^{(\infty)} = 1 \tag{6}$$

and makes the value of $\sum_{k \in \mathcal{K}} p_k^{(n)}$ close to 1 for a finite number n of iterations.

Intuitively, the method satisfies two conditions: first, the probability that the Markov chain is in population k belong to the set $\mathcal{K}$ of highest fitness after infinity iterations is 1; and this probability after a finite number of iterations n is close to 1.
The transition matrix of the Genetic Algorithms with elitism is irreducible [11]

2.5.4 Assurance for asymptotic solution

[17] assume the following conditions:
Every individual θ has a label $i = 0, 1...2^L - 1$ according to the descending order of its fitness function
Every population has a label k = 1,2,...C according to the ascending order of $i^*(k)$, where $i^*(k)$ is the individual such that $i^*(k) < j$ for any individual j in population k
Best individual $i^*(k)$ is reserved to next generation
If there are two individuals have the same fitness, there is a predetermined rule to sort them. The same for two populations with identical $i^*(k)$
Similar to [15], the transition probability $P_{k,v}$ from a population k to a population v is given as:

$$P_{k,v} = \begin{cases} (N-1)! \prod_{j=0}^{2^L-1} \frac{1}{Y(j,v)!} r(j,k)^{Y(j,v)} & for\ i^*(k) \geq i^*(v) \\ 0 & for\ i^*(k) < i^*(v) \end{cases}$$

where:

$$Y(j,k) = \begin{cases} Z(j,k) & [j \neq i^*(k)] \\ Z(j,k) - 1 & [j = i^*(k)] \end{cases}$$

Theorem 1 and lemma 2 below is from [17]:
Lemma 2: In the modified elitiest strategy, the transition matrix $P = (P_{k,v})$ from a population k to a population v has 2^L sub-matrices P(i) of size C(i) x C(i), $i = 0, 1, ...2^L - 1$, as the dialog elements, and all the components to the upper right of the diagonal entries are zeros, where C(i) is the number of populations k in which $i = i^*(k)$
Theorem 1:

$$C(i) = \binom{N - 1 + 2^L - i}{N - 1} \tag{7}$$

The transition matrix $P = (P_{k,v})$ illustrates a Markov chain with absorbing states. Thus, the condition in (6) is satisfied when the population changes follow these steps: first, the state go to the sub-matrix P(i) of size C(i) x C(i),$i = 1, 2, ...2^L - 1$, up and left until it enters P(0) of size C(0) x C(0) which represents the transition in $\mathcal{K}$; and second, it transits among the population in $\mathcal{K}$, according to P(0)

2.5.5 Convergence rate analysis

As Theorem 2 and lemma 3 in [17]:
Lemma 3: The C(i) eigenvalues of each sub-matrix P(i), $i = 1, 2, ...2^L - 1$, are identical to the $C = \sum_{i=0}^{2^L-1} C(i)$ eigenvalues of matrix P
Theorem 2 : There exists a constant B such that

$$\sum_{k \in \mathcal{K}} p_k^n \geq 1 - B|\lambda_*|^n \tag{8}$$

where

$$|\lambda_*| = \max_{1 \leq i \leq 2^L - 1} \max_{1 \leq j \leq C = C(s)} |\lambda_{i,j}| < 1 \tag{9}$$

and $\lambda_{i,j}$, $j = 1, 2....C(i)$, denotes the $C(i)$ eigenvalues of the sub-matrix $P(i), i = 0, 1, 2....2^L - 1$
From theorem 2, $\sum_{k \in \mathcal{K}} q_k^n$ is lower bound by $1 - B|\lambda_*|^n$. Thus, the value of $\sum_{k \in \mathcal{K}} p_k^n$ close to 1 for a finite number n of iterations. [17] also discuss the choice of mutation probability η to minimize $|\lambda_*|^n$ as:
Theorem 3:As for λ_*, there exists a constant, A, which does not depend on the mutation probability η, satisfying:

$$|\lambda_*^n| \leq 1 - A\eta^\delta (1 - \eta)^{L - \delta} \tag{10}$$

where the mutation order δ is the minimax value of the Hamming distance d(i,j) between individuals i and $j (< i)$:

$$\delta = \max_{1 \leq i \leq 2^L - 1} [\max_{1 \leq j \leq 2^L - 1, f(i) < f(j)} d(i, j)] \tag{11}$$

Theorem 3 shows how much the mutation probability quantitatively effects on the eigenvalue λ_*, which determines the worst-value of the convergence rate

2.6 Particle Swarm Optimization

2.6.1 Basic Algorithm:

Particle swarm optimization (PSO) is a type of evolutionary algorithm first devised by Kennedy and Eberhart in 1995 [13]. The PSO algorithm was inspired by birds in a flock searching for food, as one member of the flock finds a promising area all the other members will fly towards that member and possibly find an even more promising feeding area along the way. The implementation of the algorithm relies on a population of objects called particles which each have a position vector in the search space and real valued velocity vector of the same dimension as the search space, in PSO the population is referred to as the *swarm*. During the execution we also maintain a list that tracks the 'best' positions Pr of the individual particles, that is the position visited with the lowest cost thus

seen by that individual particle, we also track the global best position thus visited by the swarm as a whole. The Pr's in some sense represent local information about the search space while Pg imparts global information to the swarm as a whole. Algorithm 6 below details the basic algorithm for PSO with a binary search space. In order to use the velocity to update the binary position vector the logistic sigmoid $\sigma(x) = (1 + \exp(-x))^{-1}$ is used. The constants c_1 and c_2 are known as the acceration constants and are used to control the balance between local and global information.

Algorithm 9 Basic Binary PSO Algorithm

Input: Swarm size N, acceleration constants c_1 and c_2, cost function $C(\vec{b})$ and stopping criteria stoppingCondition().

Initialize swarm $\mathfrak{S}$ with N particles.
for $s \in \mathfrak{S}$ **do**
  $s.p \leftarrow \text{random}(\vec{b}) \in \{0,1\}^{|\mathcal{S}|}$
  $s.v \leftarrow \text{random}(\vec{v}) \in \mathbb{R}^{|\mathcal{S}|}$
  $Pr_s \leftarrow s.p$
$Pg \leftarrow \left(\underset{s\in\mathfrak{S}}{\text{argmin}} C(s.p)\right).p$
while not stoppingCondition() **do**
  for $s \in \mathfrak{S}$ **do**
    $r_1, r_2, r_3 \sim U(0,1)$
    $s.v \leftarrow s.v + r_1c_1(Pr_s - s.p) + r_2c_2(Pg - s.p)$
    $s.p \leftarrow \vec{b} \in \{0,1\}^{|\mathcal{S}|}$ s.t. $b_j = 1$ when $\sigma(s.v_j) > r_3$
    if $C(s.p) < C(Pr_s)$ **then**
      $Pr_s \leftarrow s.p$
  $Pg \leftarrow \left(\underset{s\in\mathfrak{S}}{\text{argmin}} C(s.p)\right).p$
return Pg

2.6.2 Modified PSO

One small novel change was added to the our version of the PSO algorithm. To help deal with non-optimal local minima from non-convex cost functions a technique inspired by dropout regularization in neural networks was employed. After updating all the particle's positions a fraction of the particles are randomly drawn and reinitialized. The swarm interpretation of this is as the swarm begins to converge on a location, which may or may not be a non-optimal local minimum, a fraction dies off and new members off in the distance will begin to flock towards the swarm and may themselves find a better position along the way towards which the swarm will then move. Algorithm 7 details the slight variation which we are naming die-off regularization.

Algorithm 10 Binary PSO with Die-Off Regularization

Input: Swarm size N, acceleration constants c_1 and c_2, cost function $C(\vec{b})$, stopping criteria stoppingCondition(), and die-off percentage d .

Initialize swarm $\mathfrak{S}$ with N particles.
for $s \in \mathfrak{S}$ **do**
  $s.p \leftarrow \text{random}(\vec{b}) \in \{0,1\}^{|\mathcal{S}|}$
  $s.v \leftarrow \text{random}(\vec{v}) \in \mathbb{R}^{|\mathcal{S}|}$
  $Pr_s \leftarrow s.p$
$Pg \leftarrow \left(\underset{s \in \mathfrak{S}}{\text{argmin}} C(s.p) \right).p$
while not stoppingCondition() **do**
  for $s \in \mathfrak{S}$ **do**
    $r_1, r_2, r_3 \sim U(0,1)$
    $s.v \leftarrow s.v + r_1 c_1 (Pr_s - s.p) + r_2 c_2 (Pg - s.p)$
    $s.p \leftarrow \vec{b} \in \{0,1\}^{|\mathcal{S}|}$ s.t. $b_j = 1$ when $\sigma(s.v_j) > r_3$
    if $C(s.p) < C(Pr_s)$ **then**
      $Pr_s \leftarrow s.p$
  $Pg \leftarrow \left(\underset{s \in \mathfrak{S}}{\text{argmin}} C(s.p) \right).p$
  Randomly select $\mathfrak{S}_d \subset \mathfrak{S}$ with $\left[\left[\frac{|\mathfrak{S}_d|}{|\mathfrak{S}|} \right] \right] = d$.
  for $s \in \mathfrak{S}_d$ **do**
    $s.p \leftarrow \text{random}(\vec{b}) \in \{0,1\}^{|\mathcal{S}|}$
    $s.v \leftarrow \text{random}(\vec{v}) \in \mathbb{R}^{|\mathcal{S}|}$
return Pg

2.6.3 Convergence and Complexity Analysis

Convergence of the PSO algorithm can be studied by casting the update process as a stochastic approximation algorithm of the form:

$$\theta_i = \theta_{i-1} + \epsilon Y_n$$

The details of which have been modified for the binary case from [19]. First we think of $\nu_i \in \mathbb{R}^{N|\mathcal{S}|}$ as the concatenated vector of all the individual particle's velocities at iteration i and let $\vec{P}(\nu,\eta), \vec{Pr}(\nu,\eta), \vec{Pg}(\nu,\eta) \in \{0,1\}^{N|\mathcal{S}|}$ be the concatenation of the position, individual best positions and global best position vectors which we view as random variables depending on the velocity and random noise η. Convergence in this set up happens when all the particles in the swarm have concentrated on top of the global best at a minimum, at this point all of the individual bests, Pr_s should become very close to the global best Pg and thus the velocity vector update will become:

$$s.v = s.v + \vec{0} + \vec{0} = s.v$$

Note: If the minimum is not optimal there is still a chance for the particles to escape and find a better location, for as long as $s.v$ is not zero the particle's position can move. Hence convergence properties of binary PSO can be studied by analyzing the velocity vectors. The velocity update scheme can be seen as:

$$\nu_i = \nu_{i-1} + \epsilon \left(\begin{bmatrix} c_1\mathtt{I} & c_2\mathtt{I} \end{bmatrix} X_{i-1} \right)$$

$$X_{i-1} = \begin{bmatrix} \text{diag}(\rho_{1,i}) & \text{diag}(\rho_{2,i}) \end{bmatrix} \begin{bmatrix} (\vec{Pr}(\nu_{i-1}, \eta_{i-1}) - \vec{P}(\nu_{i-1}, \eta_{i-1})) \\ (\vec{Pg}(\nu_{i-1}, \eta_{i-1}) - \vec{P}(\nu_{i-1}, \eta_{i-1})) \end{bmatrix}$$

Here $\rho_{k,i}$ are assumed to be random vectors with components being iid $U(0,1)$ random variables. Now the issue we face here is that the random processes X_i can be quite complicated in general but as seen in [19] under certain assumptions we can guarantee weak convergence of an interpolation scheme (at least to a local minimum) almost surely to the stable points of an ODE. Here we will repeat the assumptions from [19]:

(A1) The random variables $\vec{P}(-,\eta), \vec{Pr}(-,\eta)$ and $\vec{Pg}(-,\eta)$ are weakly continuous for each η. Let $\vec{P}_l(-,\eta) := \text{diag}(\rho_1)(\vec{Pr}(-,\eta) - \vec{P}(-,\eta))^T$ and $\vec{P}_G := \text{diag}(\rho_2)(\vec{Pg}(-,\eta) - \vec{P}(-,\eta))^T$. For each bounded $\nu, \mathbb{E}[X(\nu,\eta_i)]^2 < \infty$. There are continuous functions $\widehat{\vec{P}_l}(\nu)$ and $\widehat{\vec{P_G}}(\nu)$ such that:

$$\frac{1}{n} \sum_{j=m}^{n+m-1} \mathbb{E}_m \vec{P}_l(\nu, \eta_j) \xrightarrow{p} \widehat{\vec{P}}_l(\nu)$$

$$\frac{1}{n} \sum_{j=m}^{n+m-1} \mathbb{E}_m \vec{P_G}(\nu, \eta_j) \xrightarrow{p} \widehat{\vec{P_G}}(\nu)$$

Where $\mathbb{E}_m$ is the conditional expectation on the σ-algebra $\mathcal{F}_m$ generated by $\{\nu_0, \rho_{k,j}, \eta_j : j < m, 1 \leq k \leq 3\}$. Also for each ν in a bounded set:

$$\sum_{j=n}^{\infty} |\mathbb{E}_n \vec{P}_l(\nu, \eta_j) - \widehat{\vec{P}}_l(\nu)| < \infty$$

$$\sum_{j=n}^{\infty} |\mathbb{E}_n \vec{P_G}(\nu, \eta_j) - \widehat{\vec{P_G}}(\nu)| < \infty$$

(A2) Let

$$\hat{X}(\nu) = [c_1\mathtt{I} \quad c_2\mathtt{I}][\widehat{\vec{P}}_l(\nu) \quad \widehat{\vec{P_G}}(\nu)]^T$$

The ODE:

$$\frac{d\nu(t)}{dt} = \hat{X}(\nu(t))$$

has a unique solution for each initial condition $\nu(0)$.

(A3) $\{\rho_{k,n}\}$ and $\{\eta_n\}$ are mutually independent.

Then under these assumptions Yuan and Yin prove in [19] the interpolation process:

$$\nu^{\epsilon}(t) = \nu_i \quad \text{for} \quad t \in [\epsilon i, \epsilon(i+1))$$

Converges weakly as $\epsilon \to 0$ to a solution of the ODE in (A2).

The computational time of the algorithm mostly comes from the evaluation of the cost function on a particle's position which depends on the position,we will call the time to evaluate the cost function at a vector $T(p)$. Assuming our stopping criterion is just a fixed number of iterations M then the cost is a random variable $MN(T(\vec{P}))$.

3 Experimental Design

3.1 Data Sets

Three datasets were used for the experimental portion of the project. The data sets used were originally provided for the NIPS 2003 Feature Selection Challenge [7]. All of the data sets represent a binary classification problem and have had random noise called probes intermixed into their real features. The breakdown of the datasets is here:

Table 1: Data Set Overview

Dataset	TrainSize:(pos/neg)	TestSize:(pos/neg)	Feats:(real/probes)
Arcene	100 : (44 / 6)	100 : (44 / 56)	10,000:(7,000/3,000)
Dexter	300 : (150 / 150)	300 : (150 / 150)	20,000:(9,947/10,053)
Dorothea	800 : (78 / 722)	350 : (34 / 316)	100,000:(50,000/50,000)

Arcene comes from mass spectrometry data and is has dense real feature vectors. Dexter data is extracted from text and has sparse integer valued feature vectors. Dorothea data comes from drug discovery in proteins and is has sparse binary feature vectors.

3.2 DOP Set Up

In the original challenge [7] one of the main performance metrics used was the *balanced error rate* (BER) of a trained classifier's predictions on an unseen test set. The balanced error rate is the average of the errors rates of the positive and negative classes and is more robust than accuracy rate as it is less sensitive to imbalanced classes. For our classifier we fixed L2-regularized logistic regression implemented using Python's Scikit-Learn package with inverse regularization constant $C = 0.0001$ and a class balanced weighting scheme ($class\_weights$ ="balanced" in Scikit-Learn). Once a subset of features was selected this algorithm was trained on the training set (only using selected features) and the predictions on the test set were used to generate the BER score. Each individual algorithm has slightly different ways of using the cost function described below.

A slightly different local and global cost functions were used. The local cost was just the BER score for the selected features. However for the global cost a small regularization term involving fraction of features selected:

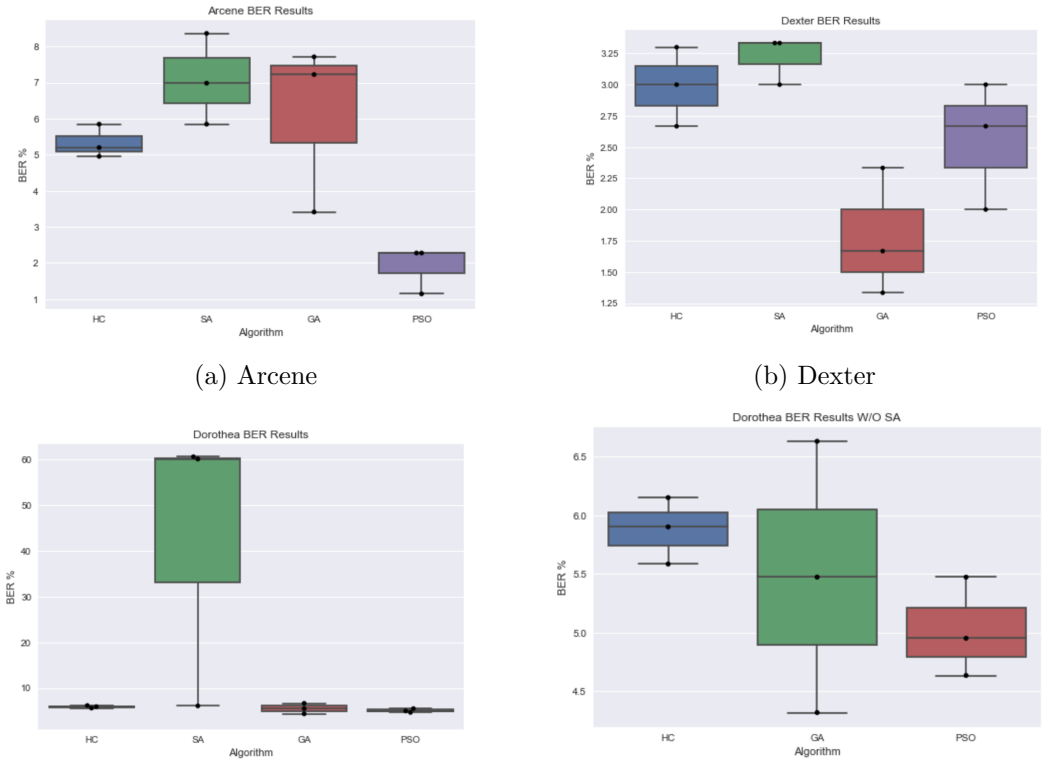
$$C_{global}(\vec{b}) = BER + 0.001(\%\text{features selected})$$

This small change influences the optimization to find the best BER with smallest set of features. It also helps escape non-optimal local minima since even if the global BER doesn't decrease if a smaller set of features has the same BER or even ever so slightly larger the swarm will move in that direction.

4 Results and Discussion

4.1 BER Results

Each algorithm was run on each data set three times. We calculate average BER and 95% confidence interval for each algorithm and report in Table 2. The results are also shown visually in Figure 4. We also measure how many features were selected for each dataset in Table 3.



(a) Arcene

(b) Dexter

(c) Dorothea

(d) Dorothea No SA

Figure 4: BER Results from 3 Experiments

4.2 Running time

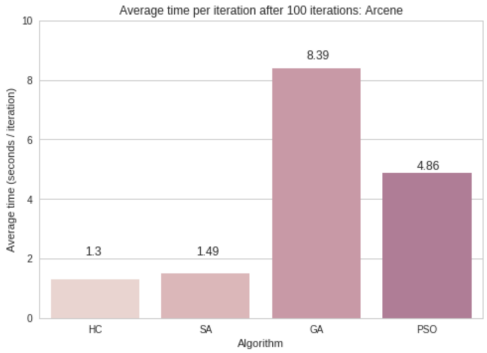
We measure the running time of all algorithms on the same desktop for 100 iterations. With local search algorithms(SA and HC) each iteration is much faster and with evolutionary algorithms(GA and PSO) each iteration is slower. As local searches works on one candidate solution they were faster per iteration.

Dataset	Full Feature	RF	SA	SHC	GA	PSO
Arcene	0.17	0.13 ± 0.28	0.07 ± 0.031	0.05 ± 0.01	0.06 ± 0.06	$\mathbf{0.02 \pm 0.02}$
Dexter	0.078	0.07 ± 0.13	0.03 ± 0.01	0.03 ± 0.01	$\mathbf{0.02 \pm 0.01}$	0.03 ± 0.01
Dorothea	0.14	0.18 ± 0.60	0.06 ± 0.01	0.06 ± 0.01	0.06 ± 0.03	$\mathbf{0.05 \pm 0.01}$

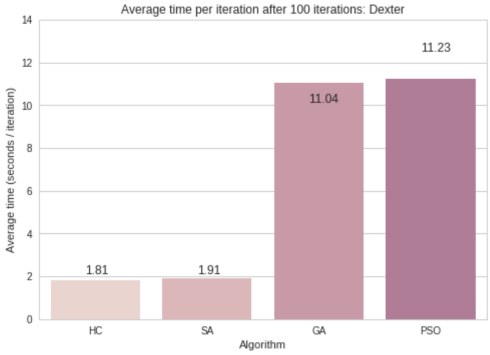
Table 2: Results: Balanced Error Rate

Dataset	Full Feature	RF	SA	SHC	GA	PSO
Arcene	10,000	300	643	580	1901	303
Dexter	20,000	300	13462	14400	12586	9875
Dorothea	100,000	300	32866	37991	43712	31975

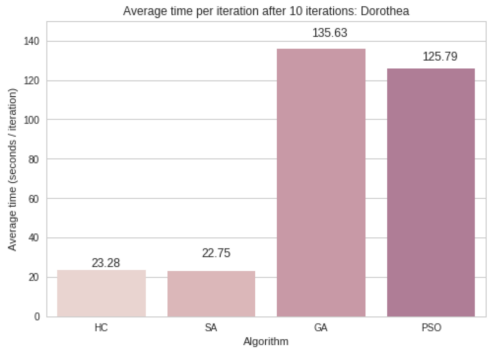
Table 3: Average number of features selected



(a) Arcene



(b) Dexter



(c) Dorothea

Figure 5: Average Time for 100 iterations in 3 datasets

4.3 Discussion

4.3.1 Tabu search

The experiment result could be seen in 2. We compare the performance of purely used top 300 features from feature ranking list with our algorithm which combines 100 from list and 200 by sampling. The results in the table shows generally our algorithm outperforms purely rely on feature ranking list.

In practice, our settings runs 300 iterations for each dataset within a pretty short amount of time comparing to the original tabu search with good performance.

We revised tabu search to fit the task of feature selection. Although the original Tabu Search has been proved to convergent, it requires a huge waiting time to reach optimal solution. Our modified version tabu search combines global optimal search with local optimal search. The global optimal search predicts the location where the optimal solution lies in. Then the local search acts as tuning the searched points to find the optimal solution.

In this task, we only adapt Random Forest as global searching method. Other feature selection methods could also be utilized for global search. As time limited, we only run experiments with limited iterations, which might be too small for a good result. Also, more hypothesis, such as mixture model or mixture component model, could be used for the design of the table.

4.3.2 PSO

From the results, we can see that using a fix number of 150 particle and 100 iterations, the PSO achieves good results on all three datasets. It achieves the higher mean BER result in Arcene and Dorothea with 0.02 and 0.05 respectively. It also uses less features than SA, SHC and GA.

4.3.3 Genetic Algorithms

The results in table 2 demonstrate that Genetic Algorithm has successfully chosen a subset of features that brings lower BER than the full set of features (Lower BER means higher accuracy, so it is better). In the Arcene dataset, the BER decreases from 0.17 (10000 features) to 0.06 (1901 features) . Similarly, in Dexter dataset, the BER reduces from 0.078 (20000 features) to 0.02 (12586 features); in Dorothea dataset, the BER declines from 0.14 (100000 features) to 0.06 (43712 features). Compare to other methods, the GAs has the best BER mean on the Dexter dataset, and the best BER min in Dorothea dataset. However, it running time is higher than HC,SA and PSO.
The result shows that in practice the GAs with elitism converges, which is strengthen by the above theoretical analysis.
In the future, we can extend the method to find the best parameters: size of population, cross over probability and mutation probability in order to increase the convergence rate.

4.3.4 Stochastic Hill Climbing and Simulated Annealing

For generalized random search algorithms tuning the parameter was very much important. For stochastic hill climbing using simple hill climbing the algorithm did not converge within 1500 iterations. With steepest ascent hill climbing the results improved. This comparison is mentioned in Table 4.

HC	# Sample	#Features	Error(%)
Baseline	100	10000	0.17
Simple HC	100	10000	0.16
Steepest HC	100	10000	0.14

Table 4: Simple vs Steepest HC

The second choice was selection of neighborhood for each state. As mentioned in Section 2.3 each state is a binary string, so next neighbor can be achieved by changing(flip/add/remove) some positions of the string. We report results with varying position 2, 6, 100 and randomly choosing how many bits to change. With the random choice, we receive much better results. As we can see in Table 2 SHC was able achieve comparable performance to evolutionary algorithms.

Bits	# Sample	#Features	Error(%)
Baseline	100	10000	0.17
2	100	10000	0.14
6	100	10000	0.16
100	100	10000	0.14
Any	100	10000	0.04

Table 5: Different choice of neighbors

For simulated annealing we choose the best neighborhood procedure from HC. We chose different cooling schedules but only with logarithmic schedule we got better results. This is expected as with logarithmic cooling schedule convergence is achieved. We ran both Hill climbing and simulated annealing for 500 iterations.

5 Conclusion and future works

In order to solve the DOP above we employ various methods from discrete stochastic approximation. Local search methods (Tabu search, hill climbing and simulated annealing) shown different type of results. As they work with one candidate solution, choosing proper neighborhood is very critical to the performance of the algorithms. With proper tuning hill climbing and simulated annealing was able to improve performance significantly. Though compared to evolutionary algorithms they were slower. Population based evolutionary methods (genetic algorithm and particle swarm optimization) employ a collection of candidate solutions and hence they were better in searching feature space. We got best results with PSO and with GA almost comparable results. Though evolutionary search algorithms took more time per iteration, they also were able to achieve convergence much faster. Future work could concentrate on analyzing how many iterations these algorithms take for convergence and how to choose related parameters to make the algorithm converge faster.

References

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] R Reeves Colin and ER Jonathan. Genetic algorithms-principles and perspectives, a guide to ga theory. *USA: Kluwer Academic Publisher*, 19:60, 2002.

[3] James E Gentle, Wolfgang Karl Härdle, and Yuichi Mori. *Handbook of computational statistics: concepts and methods*. Springer Science & Business Media, 2012.

[4] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[5] Fred Glover. Tabu searchpart i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[6] Fred Glover. Tabu searchpart ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[7] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, 2005.

[8] Saïd Hanafi. On the convergence of tabu search. *Journal of Heuristics*, 7(1):47–58, 2001.

[9] Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. The theory and practice of simulated annealing. In *Handbook of metaheuristics*, pages 287–319. Springer, 2003.

[10] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[11] Marius Iosifescu. *Finite Markov processes and their applications*. Courier Corporation, 2014.

[12] Alan W. Johnson and Sheldon H. Jacobson. On the convergence of generalized hill climbing algorithms. *Discrete Applied Mathematics*, 119(1):37–57, 2002.

[13] R. Kennedy and J. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks IV, pages*, volume 1000, 1995.

[14] Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in applied probability*, 18(3):747–771, 1986.

[15] Allen E Nix and Michael D Vose. Modeling genetic algorithms with markov chains. *Annals of mathematics and artificial intelligence*, 5(1):79–88, 1992.

[16] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, 5(1):96–101, 1994.

[17] Joe Suzuki. A markov chain analysis on simple genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(4):655–659, 1995.

[18] Michael D Vose and Gunar E Liepins. Punctuated equilibria in genetic search. *Complex systems*, 5(1):31–44, 1991.

[19] Quan Yuan and George Yin. Analyzing convergence and rates of convergence of particle swarm optimization algorithms using stochastic approximation methods. *IEEE Transactions on Automatic Control*, 60(7):1760–1773, 2015.