Part 1: Random Search

Problem 1.1: Training Neural Network

I decided to use the Isolet (isolated spoken letters) data set from assignment 1. The original data set contained 618 features. I used decision trees to select 113 features with the best information gain. However, 113 attributes is still too many to use optimization algorithms to train the network. With 26 classification categories and even a low number of hidden nodes (say 20) the total number of weights would be intractable (26 * 20 + 20 * 113 = 2780 weights – which are dimensions to train). So I selected the 20 features with best information gain. I choose to use only 10 hidden-nodes so that the total number of weights is 460. This is still a very high number of dimensions to explore, but perhaps we can adjust our algorithms accordingly.

The first decision is how to represent the real values and choose neighbors. I decided to try several different methods: round robin, converge, and random.

Round robin: Change each weight/dimension in round-robin fashion by an increment of +/- 0.5 choosing the direction that improves the evaluation the most.

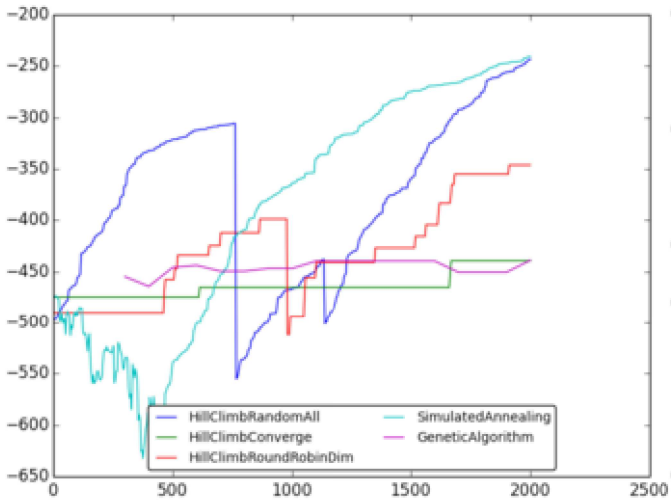
Converge: Focus on each dimension until the evaluation has converged for that dimension. Then proceed in round-robin fashion. Start with an increment of +/- 0.5 but each subsequent time in a dimension multiply increment by 0.5. This means we will converge each dimension with our increment, then more finely tune the parameters on second, third and subsequent passes.

Random: Select random dimension, random increment (between 0 and 0.5). Choose direction that improves the evaluation the most.

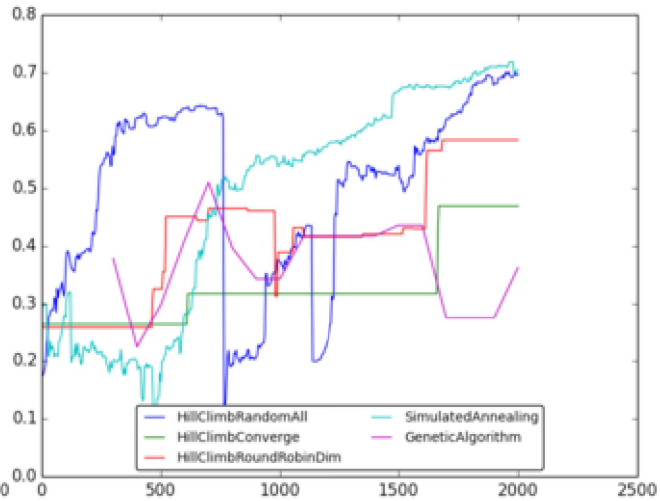
For evaluation I used negative Sum of Squares Error (ABAGAIL was already implemented that way). Misclassification Error would have made an interesting comparison, but I did not have time to implement and re-run all my analysis or do systematic comparison.

Rather than compare neighbor selection for all algorithms (which would make for very cluttered graphs) I decided to use Random neighbor for Simulated Annealing and Genetic algorithms. For Hill Climbing I analyzed each neighbor selection strategy in conjunction with the following restart strategy. Restart at random intervals (p = 0.001) for Round Robin, once converged for Converge (with high threshold for relatively quick convergence), and random intervals (p = 0.001) for Random.

With small evaluation count we can begin to see some patterns.



Fitness

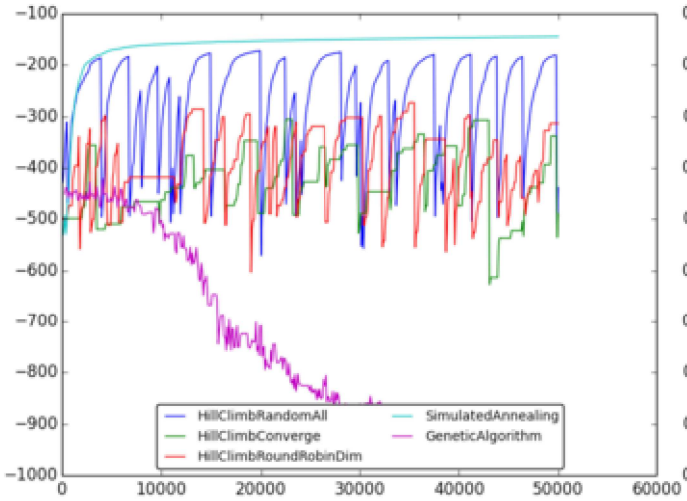


Train Correct Proportion

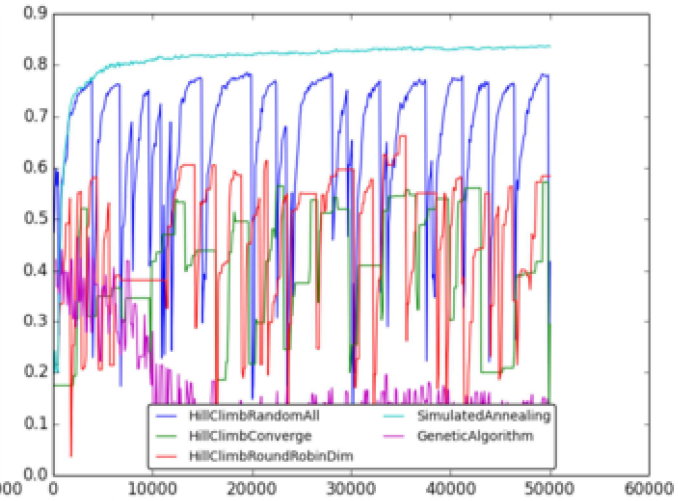
The random restarts can be seen by the sharp down ward cliffs. Random dimension and increment demonstrates a very regular improvement in both fitness and correct proportion of training (essentially the difference between sum-of-squared error and misclassification). Round robin displays improvement in fits and starts. Converge is very slow to improve and has not yet restarted. This seems to imply that most dimensions

have improvement and randomizing that improvement allows us to find a reasonable optimum. The dynamic nature of Neural Networks allows for many reasonable optima from adjusting weights iteratively, but a narrow focus (on one dimension at a time) deters overall system health.

A wider view (out to 60000 evaluations) furthers our understanding.

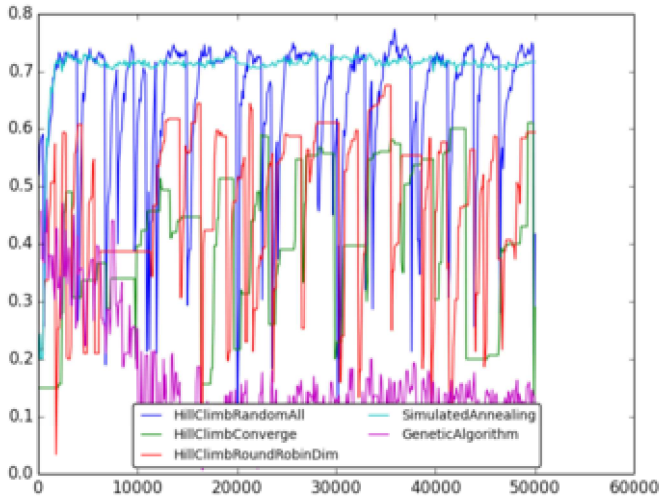


Fitness



Train Correct Proportion

Fitness and Train Correct Proportion are nearly identical if adjusted for scale. As with the previous zoomed in view Random neighbor selection is consistently better than Converge or Round Robin. And it looks like Converge neighbor selection (green) does actually converge enough to restart, but never gets above 60% correct of the training data set. Genetic Algorithms degrade over time! There must be some systematic problem causing convergence to poor weights. If we looked at fitness and train correct proportion alone, Simulated Annealing we be the hands down winner. Not only for finding the best optimum, but converging quickly and consistently improving.



Test Correct Proportion

But when we evaluate the network against the test set, the picture is not so clear. It seems that the random restart produces many optimal values with better test performance. The restart serves as a limit on overtraining while converging to a variety of optima. Of course, we could apply the same restart technique against Simulated Annealing and would expect to get results similar to random hill climb since the dimension selection and increment is random for both algorithms.

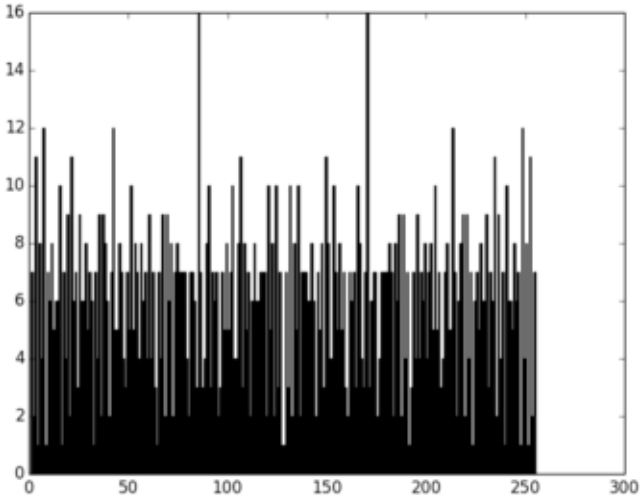
It is hard to compare backpropogation to our optimization algorithms because number of iterations/evaluations don't easily map between algorithms. But if we take our best weights after 5000 iterations, the following table makes a reasonable comparison showing that neural networks can perform at the same level as backpropogation. I'm not certain if these results would hold with higher dimensions.

	iterations	time	train correct	test correct
Backpropagation	1146	12.5 sec	84.0%	72.7%
Simulated Annealing	5000	23.4 sec	79.2%	72.7%
Hill Climbing Random	5000	23.4 sec	76.8%	73.7%
Hill Climbing Converge	5000	23.4 sec	52.1%	49.0%
Hill Climbing Round Robin	5000	23.4 sec	58.2%	60.7%
Genetic Algorithms	5000	23.4 sec	46.6%	47.0%

Problem 1.2: Compliment Substring

I defined the Compliment Substring problem as follows:

given a string, s, of length N
count ← 0
for all substrings ss of s starting at 0^{th} index
    c ← compliment(ss)
    count ← count + times c occurs in s
return count

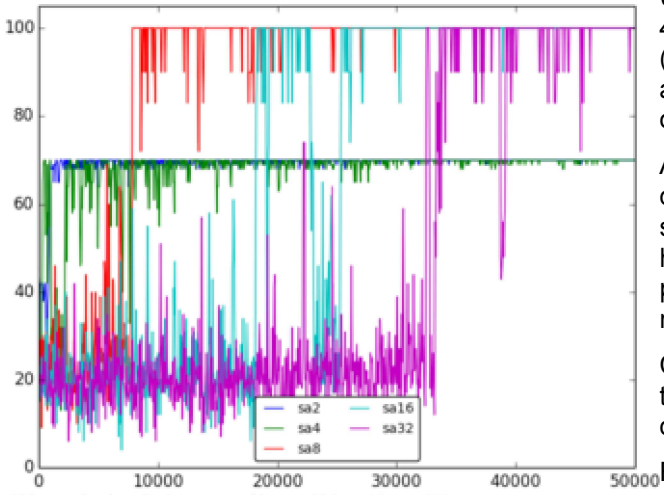


Compliment Substing Distibuton for N = 8

Compliment Substring has two global optima: alternating '01' and alternating '10'. The distribution for N = 8 (to the right) shows this evaluation function is very stochastic with many local optima. And the values most close to the global optima are quite low showing that strict hill climbing would be quickly stuck.

My initial prediction is that MIMIC will consistently raise the bar doing the best job finding one of the higher local optima.

My initial runs with best guesses for parameters of Simulated Annealing, Genetic Algorithms and MIMIC, showed poor performance for all with MIMIC doing the best. But, Simulated Annealing, flat-lined after 10-50 evaluations. So my first goal was to understand how the starting temperature effects Simulated Annealing progression.



Simulated Annealing Starting Temps

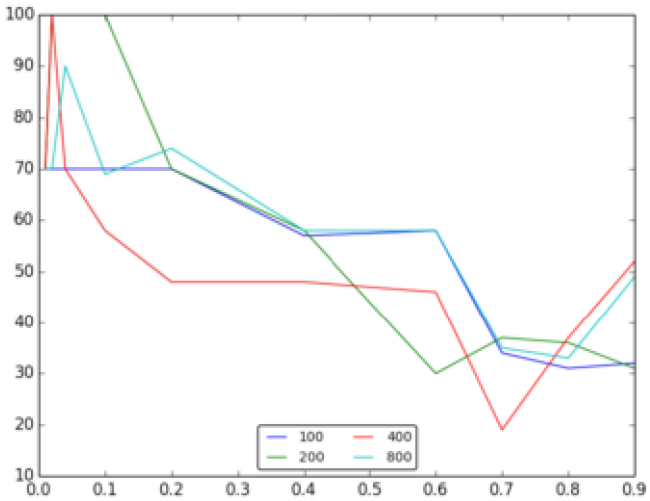
Using N = 20 and examining starting temperatures in 2, 4, 8, … 1024 is most interesting for values 2, 4, 8, 16, 32 (graph on left). I calculated cooling exponent so that after total evaluations temperature would be 1/10th its original. The global optimum for N = 20 is 100.

At low starting temperature, the best value converges quickly, but gets stuck at a local optimum. The higher the starting temperature, the later SA converges, but has higher likelihood of achieving the global optimum. This pattern continues for starting temperatures not represented in the graph.

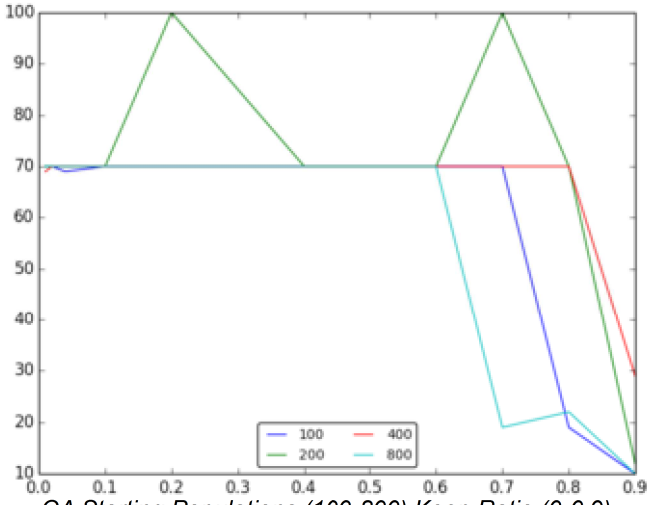
Connecting the low points for each plot demonstrates SA that as evaluation count goes up, the values get constrained closer to the eventual converged optimum.

For this problem, with this number of evaluations, a starting temperature of 10 seems effective.

I discovered that the starting population count and ratio of population kept at each iteration can also change the effectiveness for Genetic Algorithms and MIMIC. Below are the best final values after 50000 evaluations for N=20 and Population ranging from 100-800 and keep ratio 0.0 to 0.9.



MIMC Starting Populations (100-800) Keep Ratio (0-0.9)



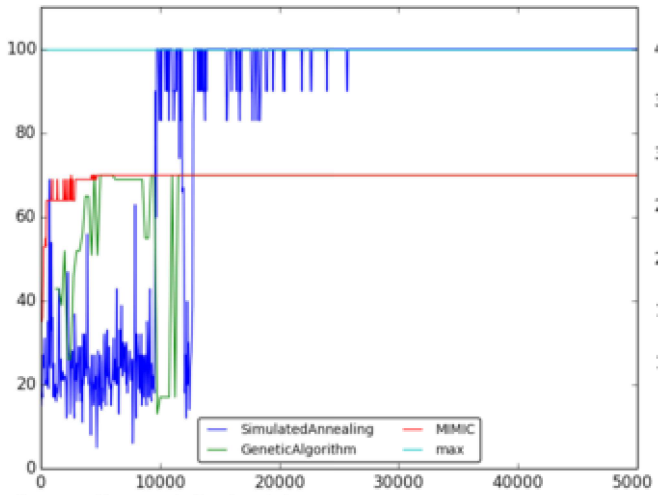
GA Starting Populations (100-800) Keep Ratio (0-0.9)

MIMIC seems consistently better for smaller keep ratio. Population size doesn't seem to have much singificant effect. As population keep ratio goes down, MIMIC will allow for more random new instances. The remaining points will create a distribution centralizing the search around best-performing loci. Because Compliment Substring is highly stochastic, lower keep ratio allows for a wider search.

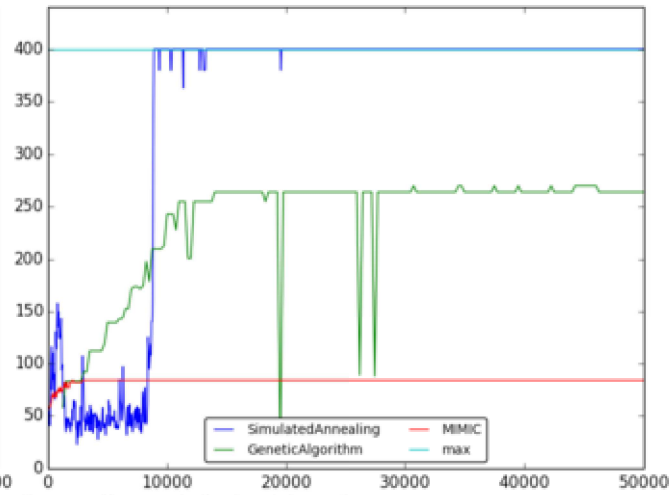
GA seems to be fairly consistent for keep ratio less than 0.6 over all population sizes. This seems to show that if the keep ratio is too high, there is not enough variety to explore the space effectively. Starting populations between 100 and 800 seem to have little effect on the algorithm.

Armed with this analysis, I ran final comparisons for N in (20, 40, 60) with the following settings:

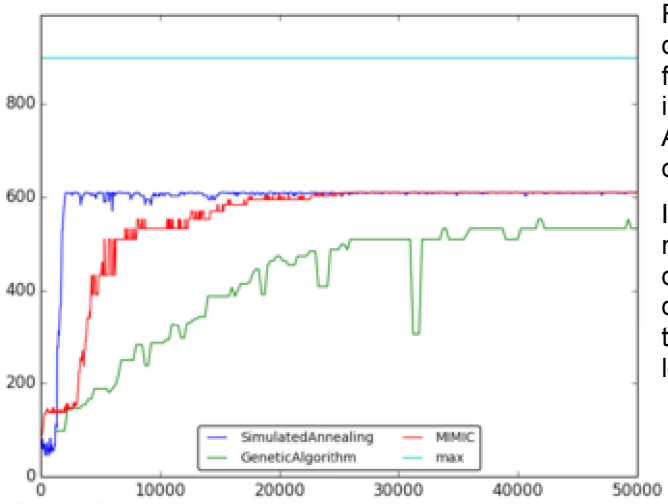
- **Simulated Annealing:** starting temperature of 10
- **Genetic Algorithms:** uniform cross-over (more random), starting population of 1000, keep 750, mate 250, mutate 50
- **MIMIC:** starting population 50, keep 5, using a dependency tree with minimal spanning tree to produce a new distribution



Compliment Substring N = 20



Compliment Substring N = 40



Compliment Substring N = 60

For all values of N, Simulated Annealing performs consistently better not only converging sooner, but also finding a higher optimum. MIMIC seems a little inconsistent, usually performing better than Genetic Algorithms, but sometimes getting stuck at a local optimum below SA and GA.

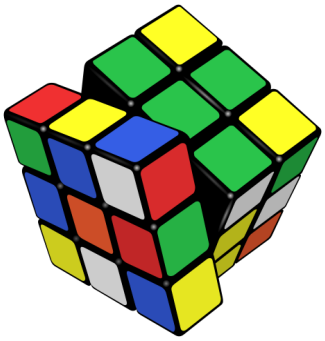
I'm surprised that MIMIC does not perform better. But it makes sense when examining the distribution. MIMIC does well at using information from previous iterations to determine likely possibilities for the next iteration. But the Substring Compliment distribution has very little local consistency making this information less relevant.

Investigations:

While I examined the output of various runs, it would be useful to systematically analyze the differences between runs – perhaps looking at graphs of averages. Random restarts for MIMIC (after convergence) or higher mutation rates for Genetic Algorithms with longer evaluations count might break the barrier that Simulated Annealing hits at higher values of N.

Problem 1.3: Rubik's Cube

The Rubik's cube is a 3x3x3 cube made up of cube pieces that allows two sides of rotation in two directions along three axis. The center pieces are fixed (other than rotating the entire cube) and the goal is the get every piece into the correct position with the correct orientation. Each piece has a position and an orientation and the solution contains a list of moves that returns each piece to the original position with the original orientation.



Standard solutions usually involve subroutines of moves that manipulate one piece at a time only temporarily disturbing other parts of the cube. This is far from optimal – however, some sort of dynamic programming is probably best suited for solving the Rubik's Cube. Yet this is should be a good problem for highlighting differences in the random search algorithms.

To translate into a search space of bits, I defined 6 moves, one for each face (left, right, front, back, top, bottom). We only need to represent rotating the face in one direction because 3 rotations in one direction is the same as 1 rotation in the opposite. Although this encoding likely excludes the optimal number of moves, it simplifies the coding for this exercise. Each move consists of 3 bits (8 values), so to simplify the coding I also included 2 extra moves: the opposite rotation for the left and front faces. We start with each piece in the correct position/orientation and scramble it K times (applying each move 3 times – to reverse moves). This means there will always be a solution that consists of K or less moves. Our space consists of N bits = 3 * K moves.

Hypothesis 1: Because the sequences of moves can be effective subroutines, Genetic Algorithms should out perform Mimic or Simulated Annealing.

My initial results, however, at K = 3 were less than promising. At 1000 iterations (with small beam size) none of the algorithms found a solution. And even more mysterious Genetic Algorithms performed the worst. Analyzing the specific GA implementation and my evaluation function is revealing.

The initial GA uses uniform cross-over, producing a single child. This will randomize moves more than desired – we want to group moves into 3-bits at a time. Uniform cross-over will also frequently break subroutines losing the structure that made a particular instance successful.

My initial evaluation function summed the number of pieces correctly placed with the orientations minus the number of moves (return early if the solution was found before all moves executed). The optimal value of this

evaluation function will be the best solution.

Hypothesis 2: Two point cross-over, grouping 3 bits, and producing two children will be more successful in preserving subroutines and solving our problems.

Hypothesis 3: Measuring value *along the way* (instead of just the final state), will select better algorithms.

Experiments:

Try combinations of mating and evaluation functions. **Mating**: *uniform*, *single point*, *two-point* – try one and two children for each. **Evaluation Functions**: "*CorrectCountEnd*", "*Best*", "*BestIfStopped*", "*SumCorrect*". "*CorrectCountEnd*" simply counts correct pieces minus moves. "*Best*" keeps track of highest correct count and subtracts number of moves at the end. "*BestIfStopped*" keeps track of the highest correct count minus number of moves after each move. "*SumCorrect*" adds all the correct pieces after each move (and if a solution is found adds correct pieces for each remaining move)

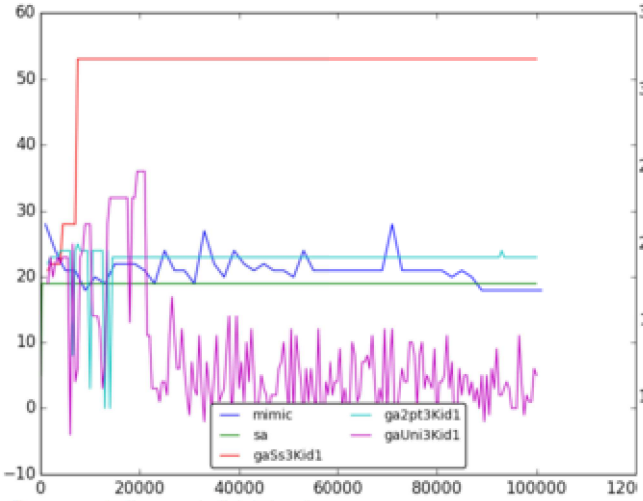
I predict that "*SumCorrect*" will best capture value along the way, producing better results.

Results:

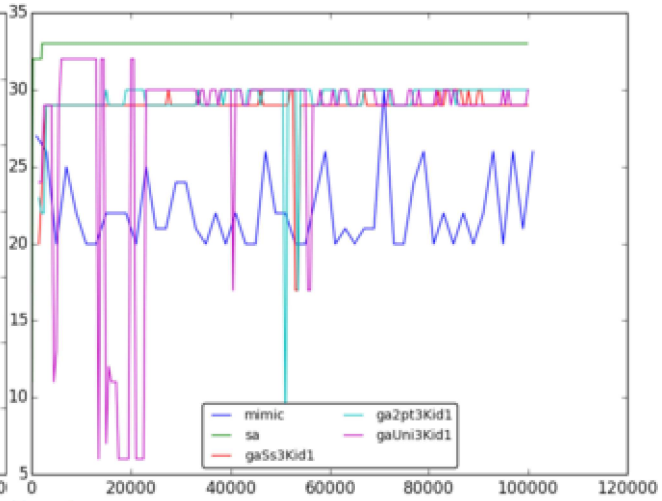
All runs were for 8 moves run to a maximum of 100,000 evaluations trying to solve the same scrambled configuration. After a little bit of tweaking the following experiments were run for the different evaluation functions with the following settings:

- **Simulated Annealing:** starting temperature of 100, cooling exponent of 0.98
- **MIMIC:** population of 1000, keep 500 for each iterations
- **Genetic Algorithms:** population of 1000, mate 500 children, keep top 500, mutate 200

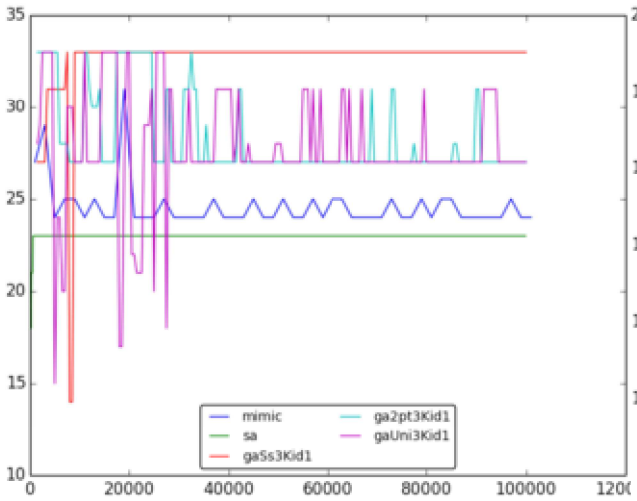
Note: The exact evaluation functions produce different true maximums and are on different scales



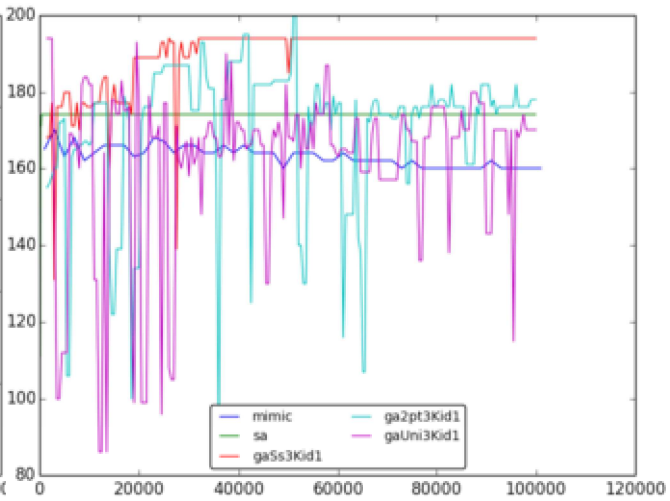
Correct Count At End



Best



Best If Stopped



Sum Correct

Each graph compares MIMIC, Simulated Annealing (sa), Genetic Algorithms at 3-bit boundaries with single child with crossovers: Single (gaSs3Kid1), Two-point (ga2pt3Kid1), and Uniform (gaUni3Kid1). Changing other parameters (3-bit vs 1-bit boundaries, single vs. two children) seemed to have little real effect. I also ran the experiments 10 times and came up with similar graphs for each. The only complete solution was "*Correct Count At End"* for gaSs3Kid1.

Simulated Annealing and MIMIC are nearly flat-lined. SA finds an optimum and sticks to it. MIMIC does have some alternating values but never seems to improve on its initial population. The genetic algorithms are much more stochastic. They seem to be trying more unique combinations. Single crossover with the "*SumCorrect"* evaluation function does seem to improve over time but also seems to get stuck at a non-solution. Overall, it appears that the "*Sum Correct"* evaluation causes the Genetic Algorithms to behave more optimally – and in almost all examples single crossover does better than two point or uniform.

Selecting the correct evaluation function has far reaching consequences (especially for Genetic Algorithms) and can help the algorithm keep instances that will help the population perform better.

Investigations and Improvements:

All algorithms seem to suffer getting stuck in local optima. Systematically analyzing the starting temperature, and cooling exponent might help Simulated Annealing to open its search space more. Systematically analyzing the population size, number of new samples from mating, amount of mutation might also reveal some better values to open Genetic Algorithms search space. (Some tweaking was tried, but not systematically analyzed.)

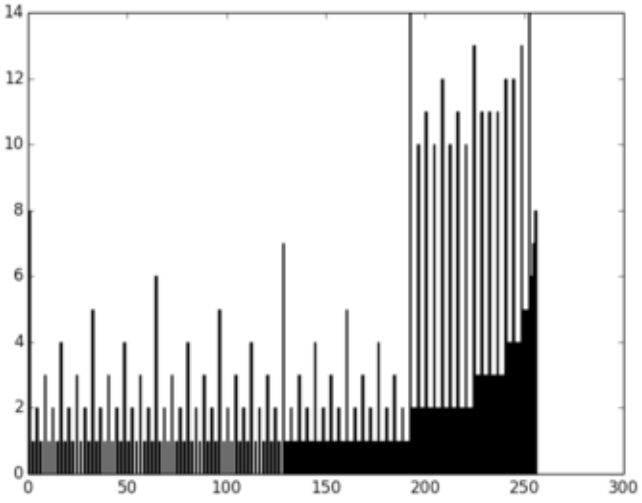
It might be insightful to capture other measures of the population: average/min/max/median/quartiles for all values of a population at a given step. Perhaps a measure of how stochastic the population is actually changing (some mating might be inbreeding – resulting in very similar offspring).

Another improvement my be to adapt the Genetic Algorithm crossover functions splicing at number of moves where best values occur. This would probably do better at capturing useful subroutines. For maximum flexibility this would probably need variable length bit-strings.

Problem 1.4 Four Peaks

The Four Peaks problem is defined as follows:

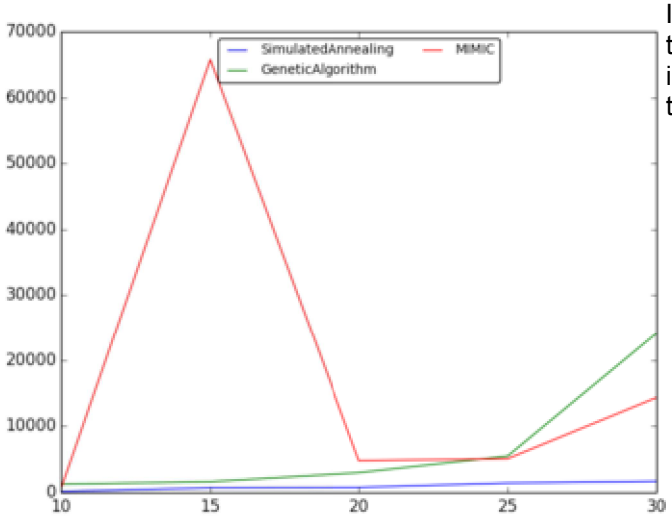
> given a string, s, of length N, some T < N/2
> tail0 ← number of consecutive 0's at end of string
> head1 ← number of consecutive 1's at beginning of string
> R ← N if (tail0 > T and head1 > T)
> R ← 0 otherwise
> return max(tail0, head1) + R



I decided on a different approach for measuring the effectiveness of the optimization algorithms. According to Prof. Isbels paper (see credits below) MIMIC is very effective at deconstructing the Four Peaks problem. I decided to try an experiment similar to the one in the paper of running until the optimum value is reached. Because we can predict the optimum values. There are two of them:

1. T+1 leading 1's followed by all 0's
2. T+1 trailing 0's preceeded by all 1's.

I ran MIMIC, Genetic Algorithms and Simulated Annealing against the Four Peaks problem for N in (10, 15, 20, 25, 30) this time counting how many evaluations until reaching one of the optima. I decided to average the results of 4 runs at each N. After leaving the test run for hours, I returned and found that all three algorithms had gotten stuck in local optima. Some were stuck as early as N=20. Unfortunately I did not have enough time to diagnose the problems and rerun the tests.



Evaluations to find Maximum Value for N in [10, 30]

I was able to glean enough data to make the graph to the left. It looks like there is either a bug in the MIMIC implementation or the parameters need more fine tuning.

Conclusion

Random Hill Climb, Simulated Annealing, Genetic Algorithms and MIMIC can each apply to different problems. Random Hill Climb works well with highly interdependent systems. Simulated Annealing seems to perform the best on highly stochastic models. Genetic Algorithms are able to capture subrountines that improve effectiveness.

In all these problems, however, parameters often require tuning to perform the best. There is no simple answer to all problems – and even the best answers have difficulty.

Part 2: Unsupervised Learning Algorithms

Data Sets

I chose to examine the same data sets from assignment 1. While the NBA draft/success data set did not produce good test results with the supervised classification, I'm hoping the unsupervised algorithms might expose some useful information. It also has a relatively small sample size and dimensionality making it easier to iterate and understand the algorithms. The isolet data set (isolated spoken letters) is interesting because it has so many attributes and classifications. The large number of attributes presents a challenge in trying to understand the shape of the data – but it should highlight the algorithms and demonstrate the usefulness of dimensionality reduction.

Clustering

In order to make simple comparisons to our labels I decided to run both clustering algorithms with *k = number of labels*. I used euclidean distance as this would be also be a useful comparison to kNN from assignment 1.

It is difficult to visualize the clustering in more than 3 dimensions, but we can look at how the labels compare to assigned clusters. Below are tables showing percent for each label and cluster. For the EM table I selected the cluster with the highest probability.

NBA data set

Kmeans Clustering

	Clusters		
Labels	1	2	3
1	4%	2%	6%
2	6%	1%	4%
3	23%	21%	33%

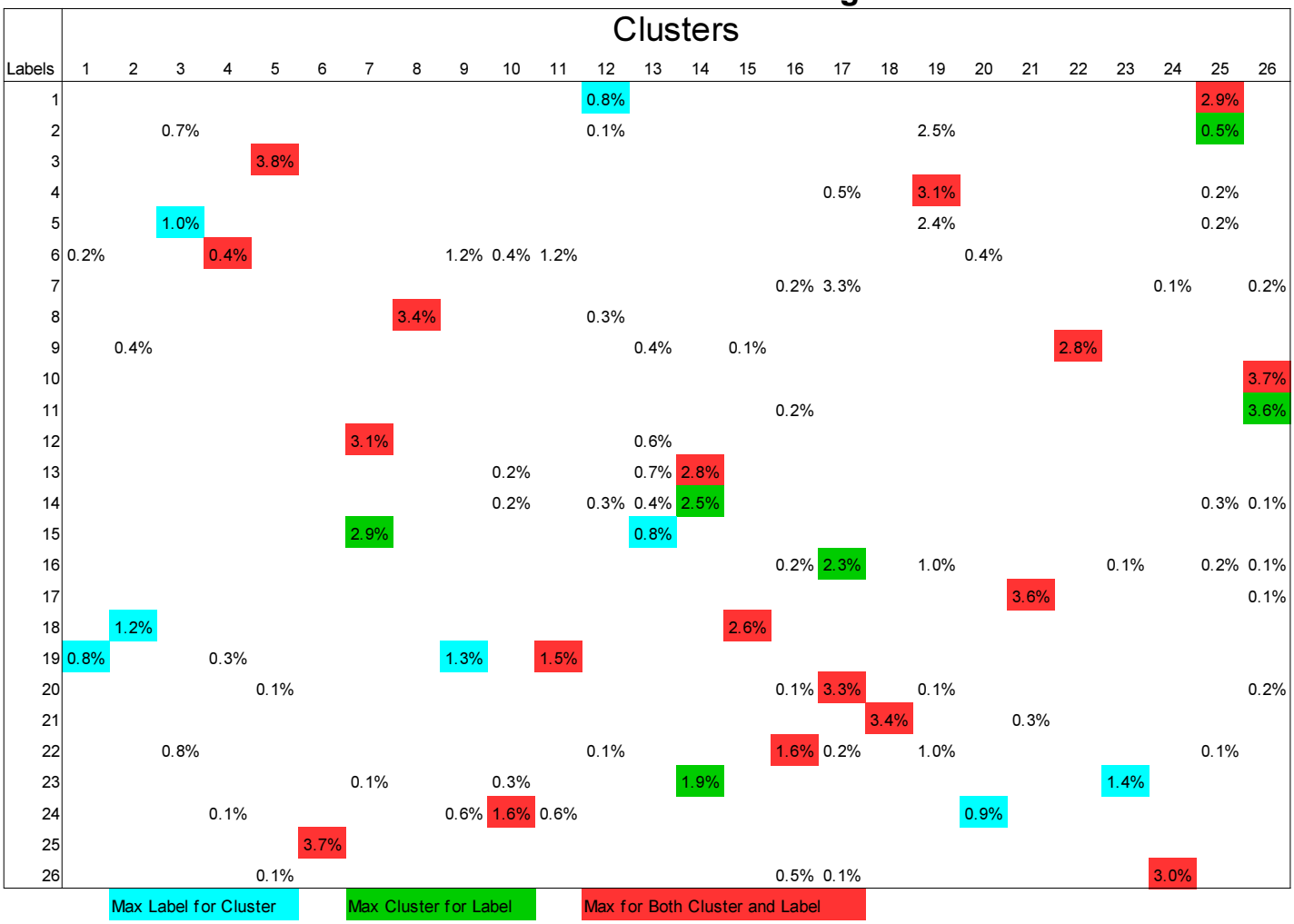
EM Clustering

	Clusters		
Labels	1	2	3
1	2%	3%	7%
2	3%	1%	7%
3	9%	21%	47%

For both clustering algorithms the majority of data points end up in a single cluster (cluster 3 for both). This is consistent with the 3rd label which also has the most instances. However the ratios would not be helpful as a predictive measure because the ratios of labels *within* each cluster are consistent across all clusters. This makes sense when compared to the KNN data from assignment 1: a distance-similar instance does a poor job predicting an instances label.

Isolet

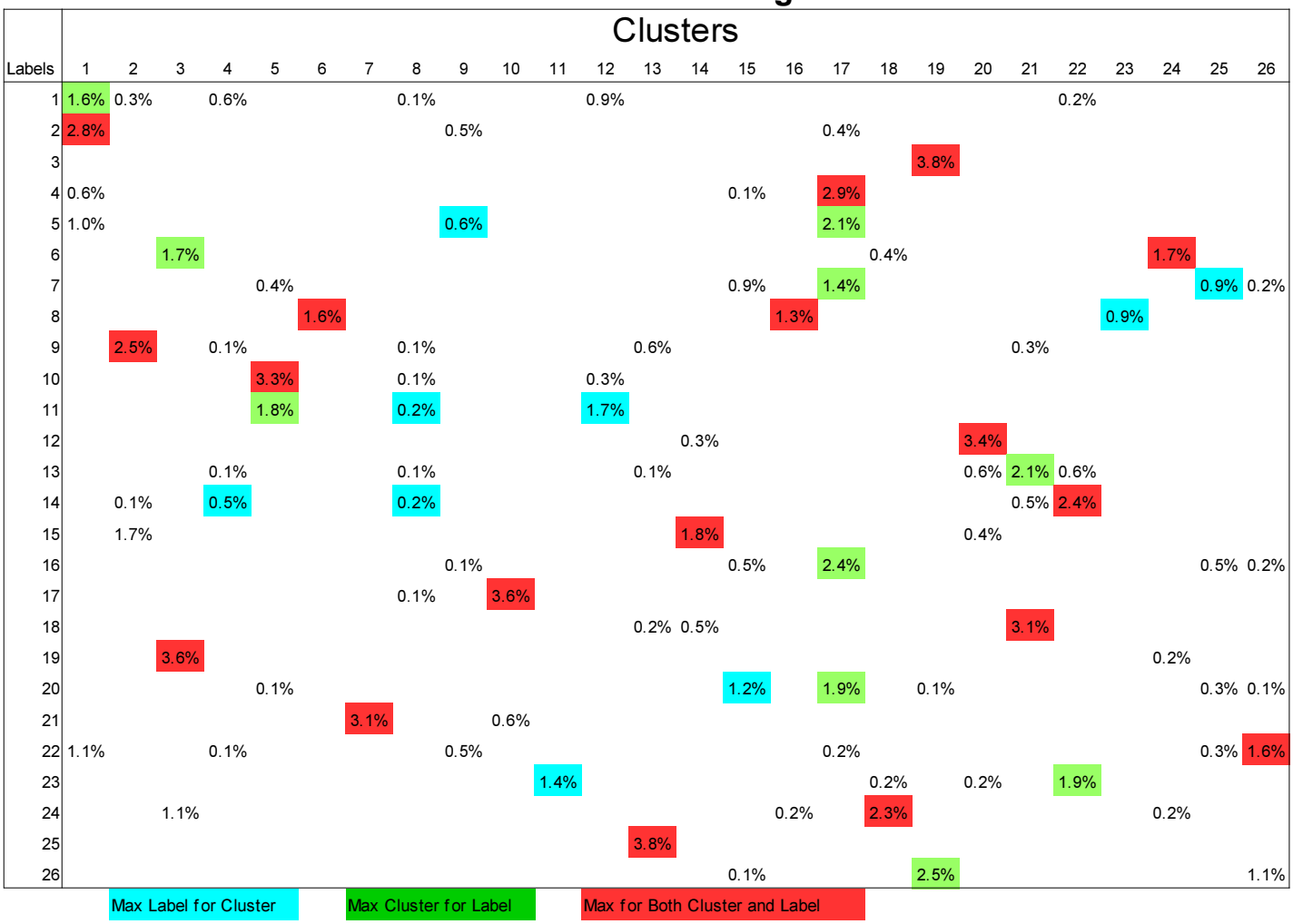
Kmeans Clustering



The KMeans Clustering matrix shows all zeros as blanks (or shadowed in the pdf). And the majority of the entries are zero. Only 82 of the 676 possible combinations are non-zero, and most of those are significantly smaller than the max one or two values for each row or column. This sparse distribution shows that the KMeans algorithm has a high correlation with the label. Of course, there are only a few clusters that could directly predict the label (cluster 6 and 8 are comprised entirely of members with labels 25 and 8 respectively). Note also that label 25 has a one-to-one mapping with cluster 6. KMeans definitely reveals an underlying structure to the data.

Isolet

EM Clustering



Similar to the KMeans, EM has many zero entries and one or two relatively large values for each row and column – again showing high correlation between clusters and labels. EM clustering has four clusters (6, 7, 11, and 23) that have members with only one label (8, 21, 7 and 8 respectively). And if we check on label 25 (the one-to-one mapping from KMeans) – only one cluster (13) maps to label 25.

We can also examine other measures for our KMeans clustering.

NBA Kmeans Stats

cluster	instances	mean to centroid	min to centroid	max to centroid	std dev to centroid	mean to next centroid	normalized volume
1	32.5%	0.84714	0.15617	4.9538	0.87092	1.85408	0.22642
2	23.8%	1.03707	0.14081	7.96943	1.22753	8.1186	1
3	43.6%	0.65986	0.0571	7.99513	0.88027	1.6642	0.67938

The instances are fairly evenly distributed, and the mean distance is fairly similar. Cluster 2 seems to have a larger extent (greater mean distance-to-centroid). Cluster 2's and 3's relative high standard deviation implies that they might have more outliers and Cluster 3's max distance-to-centroid confirms it has an outlier as far away as the extent of Cluster 2. The mean distance-to-next-centroid is significant for Cluster 2 which seems to say its members are more isolated than the other two clusters. Normalized volume was calculated by multiplying the

range of all dimensions and then normalizing so that the largest volume is 1.0. Cluster 2's members occupy a significantly larger space than Cluster 3's and almost 5 times as large a space as Cluster 1.

Isolet KMeans Stats

cluster	instances	mean to centroid	min to centroid	max to centroid	std dev to centroid	mean to next centroid	normalized volume
1	1.0%	11.06508	5.20009	23.81858	4.32418	13.79588	0
2	1.7%	8.62647	3.33965	18.37333	2.98703	15.28284	0
3	2.7%	10.36438	4.89084	30.33576	3.65043	18.13602	0
4	0.9%	8.45509	3.41574	13.59463	2.64982	10.93062	0
5	4.1%	10.38324	4.58543	28.46896	3.67751	17.29956	0
6	3.7%	10.22291	4.23603	26.24672	4.09	24.41555	0
7	6.2%	10.47113	4.31791	30.13583	4.41134	17.66801	0
8	3.4%	10.27002	3.39189	33.81535	4.60117	17.34559	0
9	3.0%	7.77759	3.34851	18.48379	2.93912	10.63318	0
10	2.8%	11.10822	4.40503	23.19949	4.07539	15.05943	0
11	3.3%	7.05401	2.45806	26.15613	2.93775	9.88477	0
12	1.7%	10.98266	4.1892	22.02986	4.3907	15.57983	0
13	2.9%	12.649	5.95026	30.38896	3.7413	17.00675	0
14	7.4%	13.05808	5.40324	30.97809	3.831	17.51078	0
15	2.8%	7.04301	2.56592	18.99429	3.11778	15.94959	0
16	3.0%	13.59238	4.75965	34.47835	4.55421	19.52576	0
17	9.7%	10.03281	3.17733	28.78142	3.80922	14.95609	0
18	3.4%	14.18001	7.07865	24.42271	3.57175	23.08851	1.15E-022
19	10.1%	9.2196	3.36062	23.21575	3.32217	14.68476	0
20	1.3%	9.03401	4.15833	19.32374	3.10537	13.00099	0
21	3.9%	14.83132	5.83315	40.5767	5.03677	23.85978	0
22	3.1%	7.50776	2.64523	24.29675	3.42404	15.17195	0
23	1.8%	26.62444	10.89886	52.42398	10.01463	44.45813	1
24	3.2%	11.62859	4.74675	29.82236	3.77488	17.30872	0
25	4.6%	8.7704	3.11632	23.58904	3.77099	13.60985	0
26	8.1%	8.86459	2.67036	32.7014	3.91362	15.18785	0

While the absolute values are larger than the NBA Data Set (which makes sense given there are 10 times as many attributes), the standard deviation for distance-to-centroid is a much smaller proportion of the mean. The mean distance-to-next-centroid also is closer to the mean distance-to-centroid (the NBA Data Set has greater than 2x ratio – here, some are 20% different). Both of these observations seem to imply more dense clusters with closer potential interactions. This is likely due to the higher number of clusters, the higher number of attributes, and more structure to the data.

Also of note is that cluster 23 has nearly 100% of the volume – so some feature has allowed this cluster to include the widest ranges of various attributes. Despite the high volume of cluster 23, the data is still well distributed among all the clusters (and cluster 23 is on the low side of number of instances).

Another measure of EM is is probability distribution which highlights the soft clusters the algorithm creates:

Isolet EM Distribution

cluster	distribution
1	7.2%
2	4.7%
3	6.4%
4	1.6%
5	5.8%
6	1.6%
7	3.2%
8	1.2%
9	2.0%
10	4.2%
11	1.5%
12	2.9%
13	4.8%
14	2.7%
15	2.9%
16	1.5%
17	11.3%
18	3.0%
19	6.5%
20	4.6%
21	6.1%
22	5.2%
23	0.9%
24	2.2%
25	2.2%
26	3.6%

NBA EM Distribution

cluster	distribution
1	14%
2	24%
3	62%

Unlike the KMeans clustering, EM distribution for NBA data is much more heavily weighted into one cluster.

The Isolet data seems to have an EM distribution very similar to the KMeans distribution. The probability of being in the 4 clusters with the highest p value is 31% (nearly 1/3). The top 4 clusters of KMeans contain 35% of the instances. The smallest and largest of the EM distribution and KMeans instance count are very close (0.9% smallest for both and 10.1%, 11.3% largest).

One more useful measure for EM is how many instances are “soft-clustered”. By looking at the highest probability value for each value and counting them in a histogram, the following table shows how “soft” the clusters are.

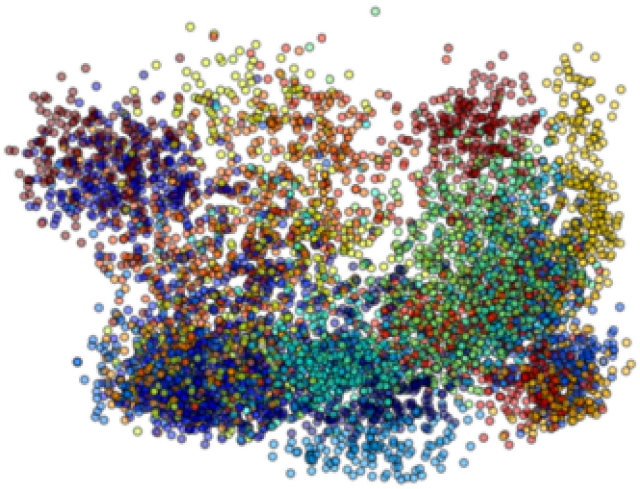
Highest Probability	< 0.1	[0.1, 0.2)	[0.2, 0.3)	[0.3, 0.4)	[0.4, 0.5)	[0.5, 0.6)	[0.6, 0.7)	[0.7, 0.8)	[0.8, 0.9)	[0.9, 1.0]
NBA	0%	0%	0%	0%	0%	1.18%	1.18%	0.71%	3.54%	93.40%
Isolet	0%	0%	0%	0%	0%	0.02%	0%	0%	0.06%	99.92%

It seems that the soft-clustering is allowing some instances of the NBA data to float freely, but the Isolet instances seem more locked down.

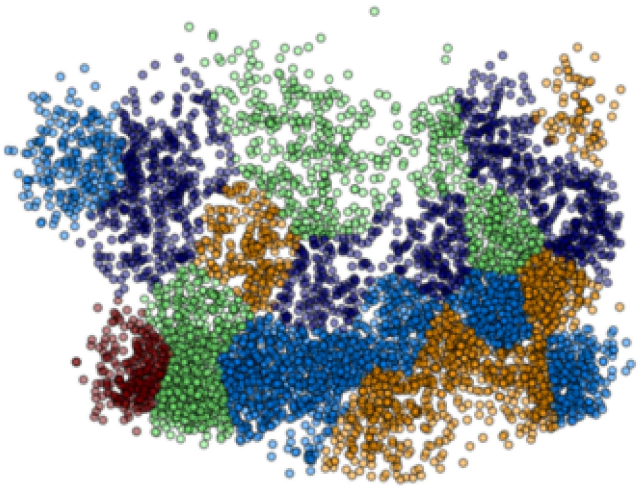
Feature Transformation

I performed feature transformation for ICA, PCA and Random Projections to a range of features for each. For the NBA data I used the range from 2 to 10 features (where the baseline set has 11 features). For the Isolet data I used the range from 2 to 20 features. I also used the Linear Components analysis of ABAGAIL (which automatically determines the number of reduced features). The transformations to 2 features allows us to plot and see how the data interacts. I plotted ICA, PCA, and RP transformations coloring by label, KMeans cluster assignment, and EM Cluster assignment. Unfortunately, I don't have enough space to show all the plots – and they are best seen interactively (flipping back and forth between various plots reveals structure). But I will highlight some interesting plots.

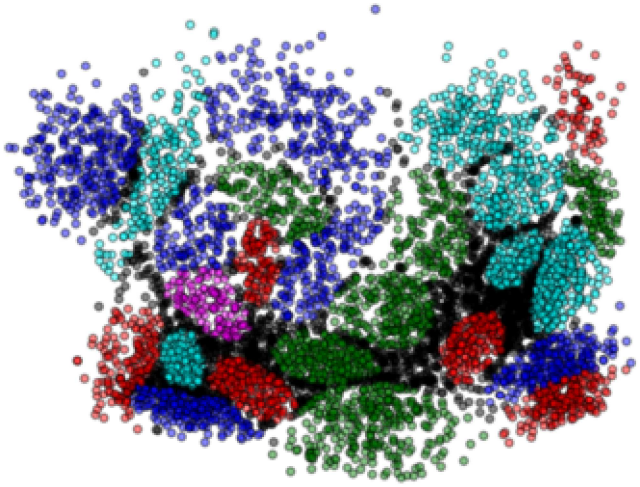
PCA



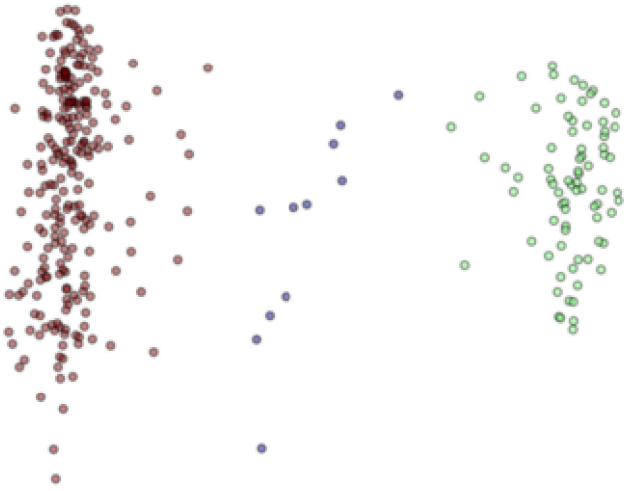
Plot 2.1 Isolet PCA colored by Label



Plot 2.2 Isolet PCA colored by KMeans



Plot 2.3 Isolet PCA colored by EM



Plot 2.4 NBA PCA colored by KMeans

Plot 2.1 shows that the labels can be grouped (although there is a fair amount of overlap). PCA did a good job preserving this structure.

Plot 2.2 is a good looking clustering of the data and matches Plot 2.1 pretty well. KMeans does a good job finding relevant clusters.

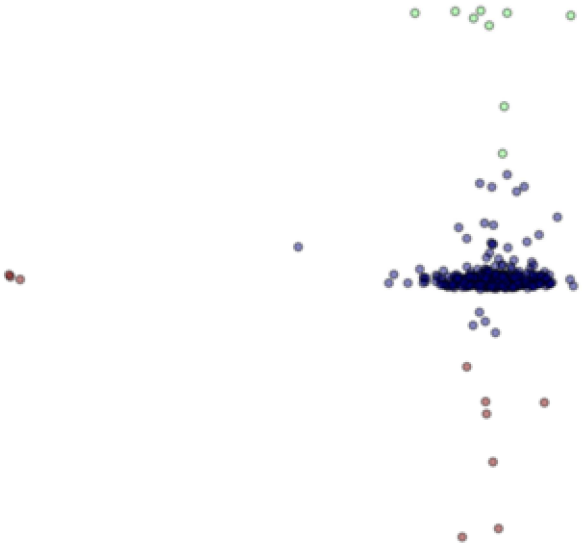
In Plot 2.3 the black instances have a main probability less than 0.5. This shows that there is a fair amount of softness (esp. when compared to the EM clustering using all 113 features, where 99.9% were in the [.9, 1.0] range – see above). This makes sense because we have reduced our dimensions and features that were once distinct are now overlapping.

Plot 2.4 could be very useful. It shows very distinct clusters. Incidentally the labels do not line up with these clusters (not enough space for that plot). This means that we may have some useful grouping that our labels did not capture! The original labels tried to capture how successful a draft prospect was in the NBA after 4 years. Perhaps there is some other distinguishing factor.

ICA

Plot 2.5 Isolet ICA colored by Label

Plot 2.5 makes it obvious that one of our dimesions has completely collapsed. It looks like our features are not very independent – at least when reduced to 2 features.



Plot 2.6 NBA ICA colored by KMeans

Plot 2.6 comparatively has some variation in the different dimensions. But all instances are close to the axes showing not only high correlation, but also that data could be represented as one dimension without hardly any loss of any information.

In order to see if higher dimensions have better results, let's look at the kurtosis:

Isolet After ICA Kurtosis vs Dimension

kurtosis	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	146	3114																		
3	146	3114	111																	
4	329	188	145	3114																
5	3114	145	25	327	188															
6	68	3114	144	333	191	112														
7	69	27	143	3113	331	191	111													
8	191	33	112	3114	106	68	333	143												
9	69	3114	331	27	106	143	191	111	40											
10	69	192	24	32	142	326	3114	111	27	106										
11	51	190	371	32	27	69	140	1386	327	106	111									
12	142	3114	24	106	27	32	69	47	39	330	111	192								
13	47	33	138	324	106	190	69	39	45	27	3113	111	18							
14	652	69	932	18	45	47	27	138	111	92	328	190	33	39						
15	18	27	1532	48	39	104	111	33	305	191	324	139	41	69	27					
16	141	190	103	39	1051	571	50	41	325	13	46	19	69	29	112	28				
17	36	42	191	329	14	18	25	24	102	27	32	39	3113	47	112	141	69			
18	662	39	13	137	29	13	28	19	186	46	112	36	326	69	924	46	105	50		
19	20	860	134	17	50	112	43	46	19	46	33	185	711	13	69	25	36	100	323	
20	39	42	31	13	15	50	187	3113	314	25	68	132	18	45	11	28	24	46	102	111

The high numbers (0 being the normal distribution) shows that ICA is collapsing to extremely small variability (high peakedness) in each feature. Even at 20 features, one of the features (7) has a kurtosis of 3113 – making that feature useless.

Comparing the kurtosis to the NBA data highlights just how peaked ICA algorithm transforms the isolet data:

NBA Kurtosis After ICA

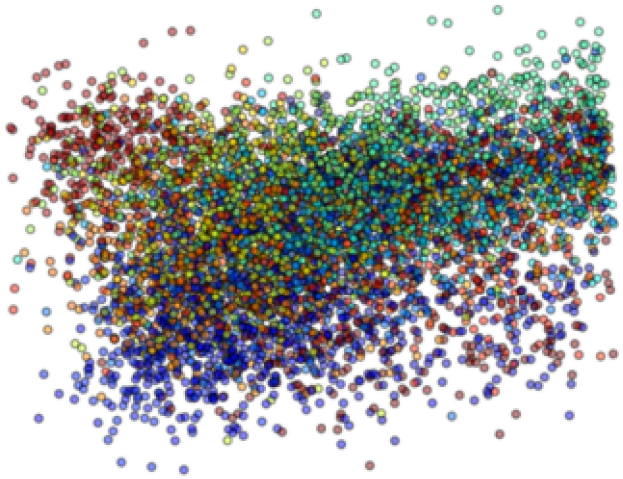
kurtosis	1	2	3	4	5	6	7	8	9	10
2	45.6	17.6								
3	45.6	11.1	9.05							
4	45.7	17.6	10.8	-1.13						
5	17.6	10.8	37.5	-1.12	11					
6	8.8	-1.13	10.8	37.2	10.5	18				
7	9.57	11.2	34.5	8.77	9.01	5.8	8.09			
8	5.54	4.61	11.2	7.45	9.03	6.5	23.9	8.21		
9	8.85	4.7	6.91	-1.12	5.77	11	7.74	7.89	25	
10	17.2	2.58	4.82	5.18	1.84	-1	10.3	8.87	22	7.8

At 4 dimensions there is still a negative kurtosis. The corresponding projection and original kurtosis shows that we are capturing 2 of the 3 lowest (most negative) kurtosis features by weighting those features the most (in absolute terms):

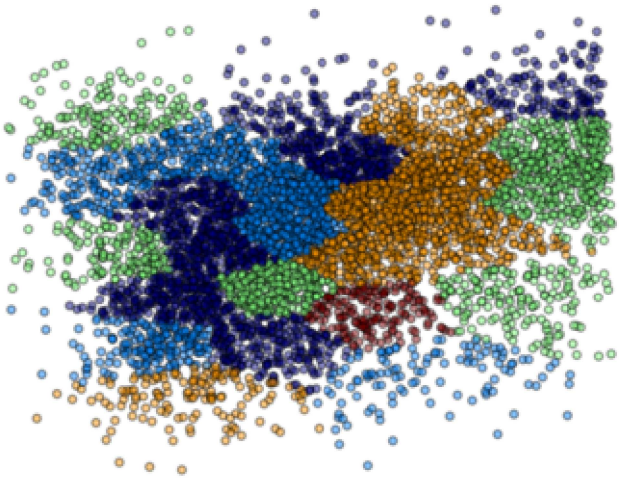
NBA ICA 4 Featues: Kurtosis and Projection

	1	2	3	4	5	6	7	8	9	10	11
kurtosis	0.2	-0.4	2.5	5.7	1.5	9.8	2.3	0.4	7.8	-0.7	-0.8
projection w eights 1	0.2	0.2	0.0	-1.7	-1.2	0.0	-2.8	-0.9	0.5	-1.0	1.5
projection w eights 2	0.0	-0.2	0.1	0.2	-2.1	-2.7	1.1	0.3	0.6	0.0	0.1
projection w eights 3	0.2	0.0	0.0	-1.4	2.0	-1.4	1.2	0.2	0.4	-2.6	-0.3
projection w eights 4	0.5	-1.2	-0.4	0.1	-0.2	0.5	-0.1	0.5	-0.3	0.3	-1.7

Random Projections



Plot 2.7 Isolet RP colored by Label

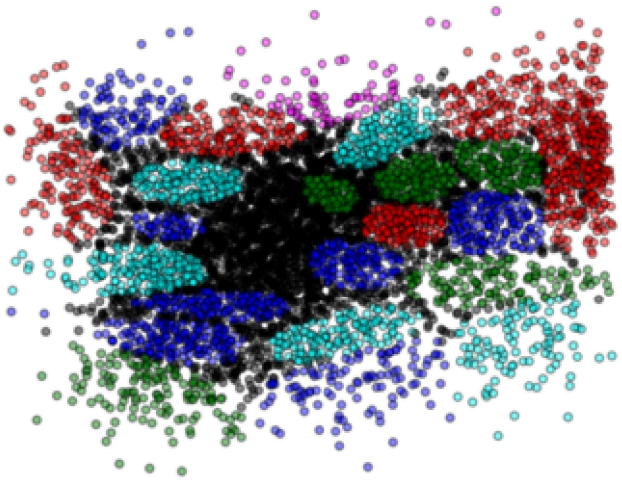


Plot 2.8 Isolet RP colored by KMeans

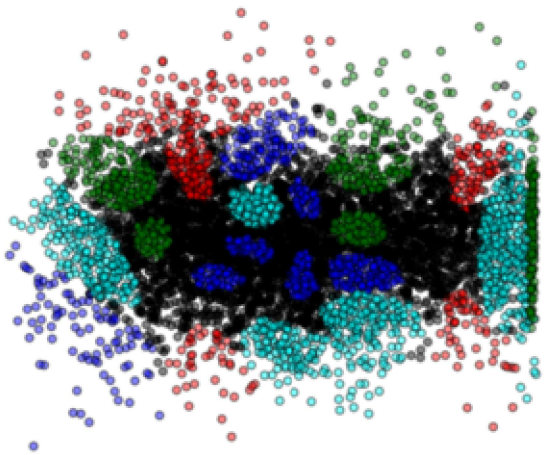
Comparing Plot 2.7 to Plot 2.1 (above) seems to show that labels are preserved slightly better with PCA than Random Projection. Plot 2.8 shows that we get well defined clusters,, but they don't seem to have much correlation with the labels in Plot 2.7.

The randomness becomes even more obvious when we look at Plot 2.9 (below). Comparing to Plot 2.3 (above) this is extremely “soft” with poorly defined boundaries.

Plot 2.10 (below) shows that other Random Projections have similar shape and clustering properties. Other plots of Random Projections had similar properties (but not included to save space).



Plot 2.9 Isolet RP-1 colored by EM



Plot 2.10 Isolet RP-2 colored by EM

Linear Discriminant Analysis

Linear Discriminant Analysis is similar to PCA, in that it attempts to fit linear transformations the data set. However, LDA uses the labels as well as the input data to determine the best transformations. In many ways, it is an attempt to create a dimension for each label that will split the data accordingly.

KMeans clustering after LDA

labels	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1																		3.8%								
2		0.7%												3.0%				0.1%		0.1%						
3	3.8%																									
4		0.2%								3.3%				0.3%						0.1%						
5										0.1%				3.7%												
6																					0.1%		0.2%	3.4%		
7				2.1%			0.3%						1.4%									0.1%				
8																		0.1%			3.8%					
9					0.2%											3.6%										
10								3.8%					0.1%													
11								3.7%					0.1%					0.1%								
12						0.2%																	3.7%			
13			0.3%								3.4%												0.1%			
14			3.2%								0.4%							0.1%								
15						3.6%																	0.3%			
16							0.2%			0.1%			0.1%	0.2%						3.1%						
17													0.1%												3.8%	
18					3.8%																					
19																								3.9%		
20				0.1%			1.1%			0.1%		2.1%	0.2%							0.1%						
21																									0.1%	3.7%
22		2.2%											0.1%	0.3%						0.1%		1.1%				
23			0.1%												3.6%								0.1%			0.1%
24																		0.1%			3.6%			0.1%		
25									3.8%							0.1%										
26	0.1%											0.1%		0.1%			1.7%		1.6%			0.3%				

The above table demonstrates how LDA effects the KMeans clustering of the data. It is surprising that the data is more spread out among clusters than our original KMeans cluster matrix.

Filtering Neural Network

Baseline with all 113 attributes:

- Train Correct: 96.6%
- Test Correct: 89.7%
- Training Time: 101.8 seconds
- Iterations: 104

Running for 2, 10 and 20 dimensions:

Filtered	TrainCorrect	TestCorrect	Time	Iterations
ICA_2	6.7%	6.9%	2.0	11
ICA_10	31.1%	26.2%	40.2	170
ICA_20	36.5%	33.7%	12.2	32
LDA	97.6%	90.7%	32.6	79
PCA_2	37.6%	38.3%	5.9	35
PCA_10	85.4%	83.8%	19.2	94
PCA_20	94.6%	89.1%	49.2	126
RPA_2_0	20.7%	19.3%	4.1	22
RPA_10	64.2%	60.8%	22.1	102
RPA_20_0	84.4%	78.5%	45.8	149

As dimensions increase the train and test % go up as expected. RPA performs poorly, but ICA is even worse – we don't really have good independent non-gaussian features. LDA performs even better than the original on the test set (but that might be simply that it doesn't overfit quite as much – or small random variation). All filtered data sets run significantly faster than the baseline. Although sometimes they run as many iterations but the time saving is that each iteration has fewer weights to train.

Adding Cluster as Feature

W/Cluster	Attribute			
	TrainCorrect	TestCorrect	Time	Iterations
ICA_10	24.6%	21.1%	16.1	66
ICA_2	10.5%	10.2%	9.0	47
ICA_20	41.6%	35.9%	39.4	115
LDA	99.2%	91.5%	54.5	141
PCA_10	85.3%	83.2%	18.2	80
PCA_2	38.5%	38.4%	14.0	83
PCA_20	95.2%	88.6%	52.4	152
RPA_10_0	64.3%	60.2%	16.5	64
RPA_2_0	20.3%	19.1%	7.3	35
RPA_20_0	85.1%	79.6%	57.6	170

Using KMeans cluster, EM cluster with highest probability (mapping to range of -1, 1) and the probability of that EM cluster as three additional attributes there is not much difference in the Test Correct %, iterations or time. It is interesting that both PCA filtered to 20 attributes and LDA have very high Training Correct % without suffering in the Test Correct %. Prehaps this is just luck, but it certainly shows that PCA and LDA capture important aspects of the data set.

Conclusion:

KMeans clustering is effective at finding categories for data. EM Clustering provides shape to our data. PCA gives effective dimension reduction and allows us to view our data in 2d. Combined with KMeans and EM Clustering we can see how even high dimension data gets separated. This can be useful for identifying outliers and similarities and even hidden features. While filtering is useful for reducing time in training a neural network, adding additional features based on clustering seems to be redundant – at least for the isolet data set.

Credits:

I started with code from the ABAGAIL project. I modified some parts heavily to run my experiments, fix bugs and output useful data.

Photo of rubik's cube was taken from http://en.wikipedia.org/wiki/File:Rubik%27s_cube.svg under Creative commons license.

The four peaks problem was taken from mimic-tutorial.pdf, *"Randomized Local Search as Successive Estimation of Probability Densities"*, by Charles L. Isbell, Jr.

Kurtosis code was adapted from a post on Piazza by classmate Herman Sheremetyev, who adapted it from https://github.com/poplav92/OnlineKurtosisSkewnessVariance/blob/master/src/OnlineKurtosisSkewnessVariance.cpp