Part 1. Randomized Optimization

Neural Network training through Randomized Optimization

The Skin dataset is a set of RGB points taken from portraits of human faces. Each point is labeled skin or non skin. In assignment one, I showed that neural network trained using backpropagation of error can learn a function that labels points very accurately. In part one of this assignment, I train the same neural network using a variety of randomized optimization functions, including randomized hill climbing, simulated annealing, and a genetic algorithm.

Randomized hill climbing begins training by selecting a starting set of weights as a random point near the origin in the space of the neural network weights. It then samples weights in the neighborhood of the current instance up to a limit that I call the neighborhood size. It moves to the first point that improves upon the fitness of the current instance and begins sampling again, or terminates if no point in the neighborhood can improve on the current point as it has reached a peak. Three functions, `random`, `fitness` and `neighbor` must now be defined.

The `random` function returns a set of random weights between -0.1 and 0.1. Small weights are preferred when training neural networks because large weights can overcome features of the data and fail to learn the target function. Tests with random weights scaled by factors of 1 and 10 were found to be less suitable than the 0.1 scale.

The `fitness` function takes as input an instance of network weights. Its output must be real valued and increasing as the suitability of the network weights to the classification task at hand increases. To accomplish this, I initially select a small training set at random from the full skin dataset. Each time `fitness` is called it finds the output of the network given the input weights over the training set and outputs the count of the correctly classified instances, less the margin in the wrong direction of misclassified instances. This definition allows fitness to increase by getting more classifications right, or by getting more classifications less wrong.

The `neighbor` function takes an instance of network weights and returns an instance of network weights that are nearby on some distance measure. Exhaustive search in the neighborhood of all orthogonal unit vector offsets from the current point finds lower peaks than sampling in random directions because peaks between steps are never reachable with fixed units and orthogonal directions. Sampling random neighbors require a limit on the sample size, as there are an infinite number of neighbors that are one random vector away from the current point. This `neighbor` function returns a new set of network weights that is the sum of the given weights and a new set of weights returned by `random`.

The rest of the implementation of randomized hill climbing was a control loop to train the neural network until stopping criteria are met. Here, training stops when no point in the neighborhood can improve on the fitness of the current point more than a minimum improvement parameter set to 0.01.

Figure 1. Fitness of Randomized Hill Climbing on skin classification problem

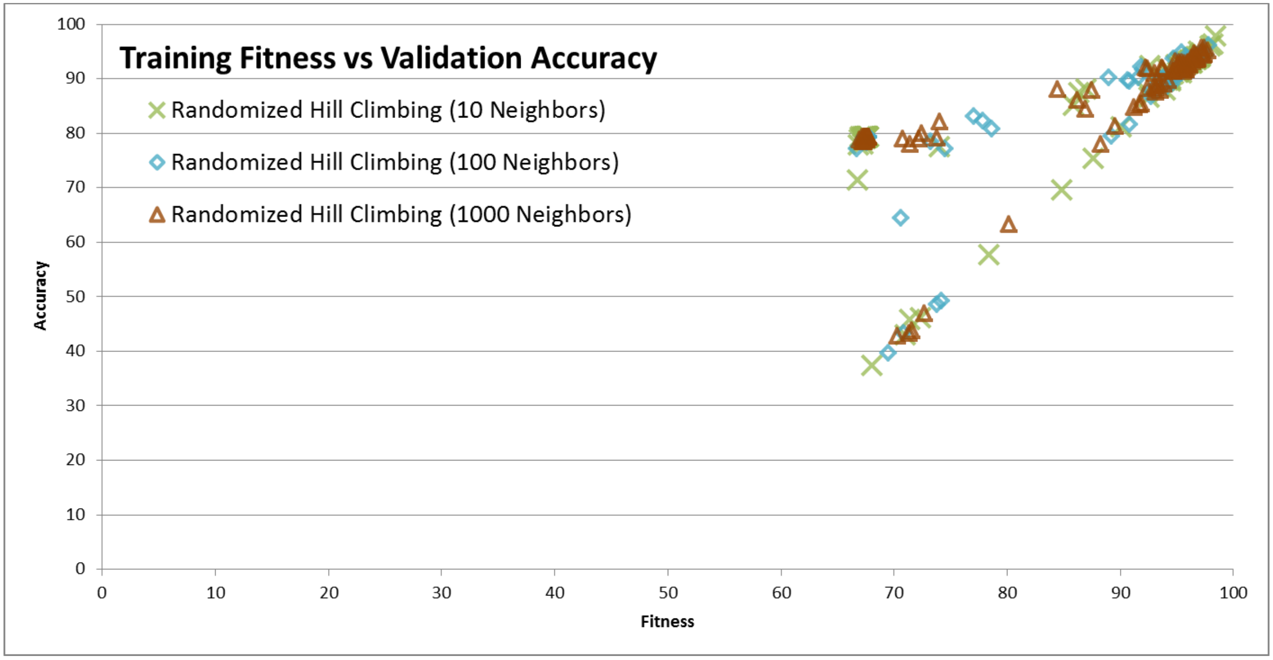


Table 1. Results of 100 Randomized Hill Climbing starts on skin classification problem

Neighborhood Size	Training time for 100 starts (sec)	Optimum fitness	Validation accuracy of optimum (%)	Standard deviation of fitness	Minimum fitness
10	28.518	98.46	97.75	13.31	66.82
100	28.197	97.85	96.07	13.01	66.70
1000	28.065	97.74	95.17	12.74	66.96

The training time did not increase while increasing the number of samples tried before stopping. Increasing the neighborhood size parameter decreases standard deviation of fitness slightly, which decreases slightly the expected number of random starts before a suitably fit optimum is found with a certain probability. The validation accuracy of the most fit result decreased as neighborhood size increased, which implies that larger neighbor sample sizes lead to overfitting the training data.

Other variables that were not varied include the size of the training set, the minimum fitness improvement criteria, and the scale parameter used for the random step. Randomized hill climbing is known to be prone to stopping in local optima that are less fit than the global optima. Simulated annealing overcomes this limitation with a temperature parameter that allows it to randomly step out of local optima while hot, and climb to the nearest peak when as it cools.

Simulated annealing can be built on randomized hill climbing with a few modifications. Instead of stopping criteria based solely on exhaustive neighborhood search, there is a random component that will accept a less fit instance from time to time. The temperature parameter is used to control the selection of the next instance, such that the next step is nearly random when temperature is high, but seeks fitness improvement when the temperature falls. A cooling parameter controls the rate of temperature decay with each step.

The formula used for the probability of accepting a less fit instance is taken from the ABAGAIL implementation `SimulatedAnnealing`. The class itself is not used here; the encapsulation of temperature and exposure of a single train() method made it difficult to take the current temperature into account in the stopping criteria. Instead, the algorithm is implemented with the ABAGAIL `RandomizedHillClimbing`, and the outer control loop takes into account temperature and adds the random step component.

For the starting temperature, I arbitrarily chose to use one half of the maximum double precision floating point value in Java. A cooling factor of 0.1 was chosen after some experimentation. The neighborhood size of 10 was retained from the previous experiment.

The fitness of weights found by simulated annealing improves on optima found by randomized hill climbing, and the validation accuracy did not suffer. The standard deviation of fitness was cut in half, but training time doubled. Most notably, there are fewer misfit optima with high fitness but low validation accuracy found using simulated annealing. Whether this is a feature of the problem space, or a feature of the algorithm remains to be seen. If the increase in time complexity is affordable, or the cost of random starts is sufficiently high, then simulated annealing is preferable to randomized hill climbing.

The third optimization algorithm tested was the ABAGAIL implementation `StandardGeneticAlgorithm`. The algorithm maintains a population of instances, and randomly mutates and mates them each generation, replacing the least fit with offspring of the fittest. The parameters are population size, offspring per generation, and mutations per generation. Two new functions have to be defined for neural network weights, a mating function and a mutation function. The `mate` function takes two instances of weights and returns a new instance that is some combination of the two. The `mutate` function modifies the weights of the given instance. Here the offspring of a pair is created from the average of their weights, and the mutation function adds weights from a random instance to the given instance.

The parameters chosen impact how the population develops. Each mating will result in a new instance between two other successful instances, and each mutation will result in a random step. After some experimentation, I elected to mutate half of the population each generation, and mate five times per generation. More mutation might lead to faster convergence, or possibly even divergence; more mating might lead to slower search and a more homogenous population.

The runtime for the standard genetic algorithm is significantly slower than hill climbing or simulated annealing. This makes sense given that each generation must mutate and mate numerous instances, where hill climbing only had to mutate once per iteration. The advantage to this is the population explores many paths at once, moving by random steps and jumping from less profitable paths to explore the midpoint of two profitable ones. The downside is that there is no pressure for individual members to improve through search, and only the periodic replacement through mating will prune the poor performers that have wandered into low fitness space.

The stopping criterion for the genetic algorithm's control loop is a fixed number of generations. I varied the number of generations from 100 to 300 in increments of 100. Training time scaled linearly with the number of generations trained. The fittest member of each population after stopping was measured on the training and validation sets.

As the generations increase from 100 to 300, the fitness and accuracy approach the peak found by simulated annealing, but does not quite reach the same optimum performance. Additional iterations would likely allow the algorithm to improve further, but the time cost discourages the selection of this genetic algorithm over simulated annealing to find absolute optima. Another possible solution is to initialize a population of instances with this genetic algorithm as exploration, then train each member with a hill climbing algorithm to find the nearest peak

Figure 3. Performance of optimization algorithms on skin classification

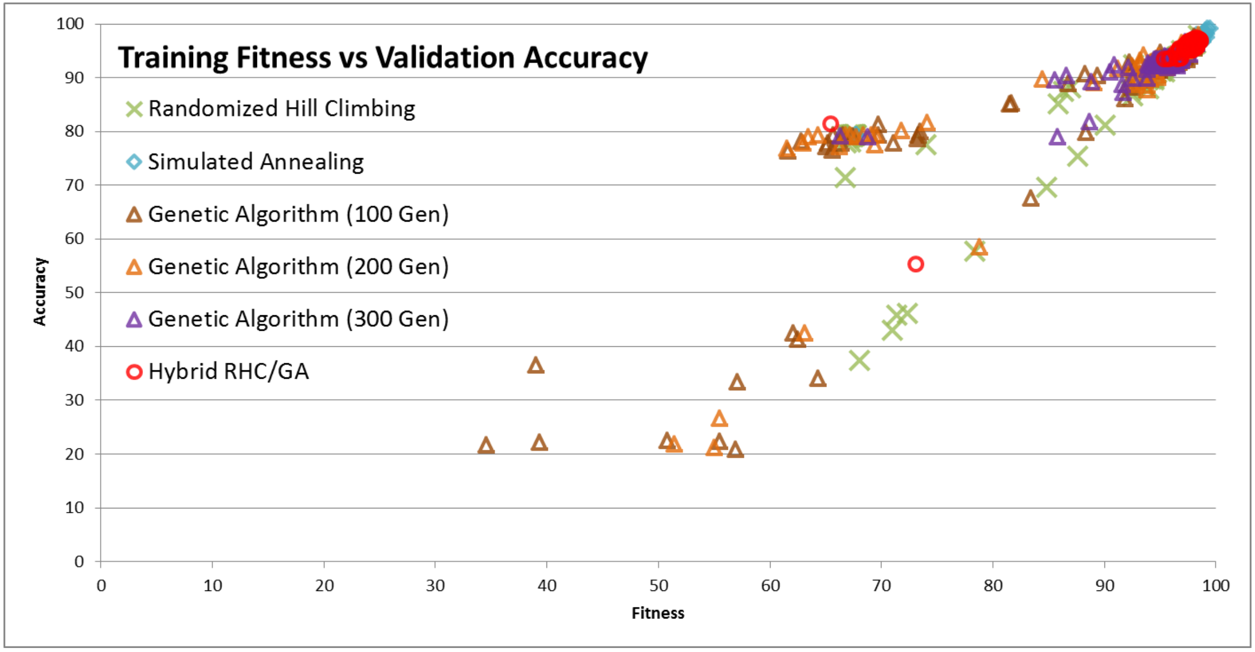


Table 3. Results of optimization algorithms on skin classification

Optimization algorithm	Training time (sec)	Optimum fitness	Accuracy of optimum	StDev fitness	Minimum fitness
Randomized hill climbing	28.518	98.46	97.75	13.31	66.82
Simulated Annealing	57.432	99.53	99.04	6.61	67.75
Genetic Algorithm	782.300	98.27	96.30	4.52	66.32
Hybrid RHC/GA	54.959	98.63	96.97	4.09	65.44

The simulated annealing algorithm found the best weights for our skin classifier in terms of fitness and validation accuracy. The standard genetic algorithm found optima with least variance across starts, although it did not hit the peak of simulated annealing. A hybrid genetic algorithm followed by hill climbing performed in similar time to simulated annealing and similar standard deviation to genetic algorithms, but failed to find a global optima comparable to the best found by simulated annealing. This could be chance, or it could be that hill climbing and genetic algorithms were both attracted to a particular region of optima that simulated annealing was able to escape to find a better solution.

Contrast of optimization algorithms over selected problems

The flip flop evaluation function counts the number of alternating bits pairs from right to left until the first non-alternating consecutive bit pair is encountered, then returns the count. Hill climbing and simulated annealing should have no trouble optimizing this problem, given enough iterations, since for any given instance, there is exactly one neighbor within a single bit flip that improves the function. A genetic algorithm with a single bit mutation function and a single point crossover function should also be able to find the absolute optimum eventually, since each mating has a good chance of preserving an alternating string of bits to the right and potentially extending that string to the left from an otherwise unfit instance. The MIMIC algorithm should be able to learn a dependency tree distribution that captures the alternating structure of bits from right to left. Any flat uniform distribution with no conditional dependence could not learn to optimize the flip flop function, because the alternating bit pattern of the optima averages to the uniform distribution.

The four peaks evaluation function counts the consecutive leading ones in the head, the consecutive trailing zeros in the tail, and then awards a bonus equal to the size of the bit string if both the head and tail are longer than the parameter t. This is tricky for hill climbing algorithms, because for any given instance there are two neighbors that can increase fitness, but if the head and tail do not both grow to t before the algorithm stops, or if one of them grows beyond length minus t, then the largest fitness boost will never be found. Simulated annealing might allow more time to find the balanced head and tail in some iterations, but it will still tend to find lesser optima. Genetic algorithms have a better chance of finding the bonus through a single point crossover. MIMIC should excel at this, once one sample finds the bonus, it will be preserved in the kept pool to train the optimal distribution for all remaining iterations.

The continuous peaks evaluation function is like the four peaks function in that it has a parameter and it counts runs of ones and zeros, but unlike the four peaks function, it counts the longest run of ones and longest run of zeros anywhere in the bit string. This provides more ways to improve fitness in any given iteration by flipping a bit on either side of the longest run of zeros and the longest run of ones, rather than on the inside of either the head or the tail as in four peaks. That difference makes it easier for hill climbing algorithms. It should not change the fitness of genetic algorithms or MIMIC to optimize this problem.

The count ones algorithm does what it says. It should be trivial for the hill climbing based algorithms to find a neighbor that improves any given instance in the early iterations, but once the zero positions become sparse, further improvements become harder to find. Genetic algorithms should perform well, because as they replace the poor performers with crossovers of good performers, zeros should be selected out. The MIMIC algorithm should also quickly approach an optimum distribution of all ones, but the probabilistic nature might still return a suboptimal result from the optimum distribution.

Each test used the same criteria used by the ABAGAIL test classes, and was repeated 10 times to get a distribution of optima found by the implementation. Randomized hill climbing and simulated annealing were each allowed 200,000 iterations to find a peak. The standard genetic algorithm was allowed 1,000 iterations with a population size of 200, a mating rate of 100 per generation and a mutation rate of 10 per generation. MIMIC was allowed 1000 iterations of 200 samples, keeping 20 per iteration, and configured to use a discrete dependency tree for the learned distribution. In each case the experiment is designed to be roughly equivalent to 200,000 iterations of randomized hill climbing.

Table 4 Optimization Problem Results

Problem	**Algorithm**	**Total Time**	**Min**	**Max**	**Median**
Flip Flop	RHC	00:02.6	56	69	65
Flip Flop	SA	00:03.7	78	79	79
Flip Flop	GA	00:02.0	68	73	71
Flip Flop	MIMIC	00:50.5	68	74	73
Four Peaks	RHC	00:01.6	80	80	80
Four Peaks	SA	00:02.7	80	151	80
Four Peaks	GA	00:01.3	24	118	98
Four Peaks	MIMIC	00:49.7	52	146	80
Continuous Peaks	RHC	00:02.7	74	95	86
Continuous Peaks	SA	00:03.9	89	112	105.5
Continuous Peaks	GA	00:02.2	76	91	83
Continuous Peaks	MIMIC	00:33.5	87	106	103
Knapsack	RHC	00:01.5	3367.9	3558.2	3438.0
Knapsack	SA	00:03.2	3433.5	3610.6	3514.7
Knapsack	GA	00:01.4	3628.8	3766.5	3731.9
Knapsack	MIMIC	00:23.6	3725.4	3894.1	3792.5

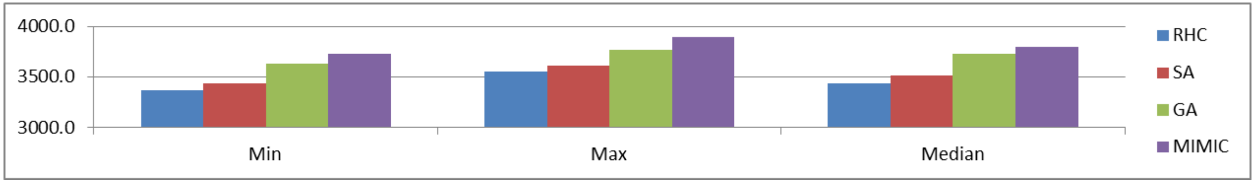
The maximum possible fitness for the flip flop function given a bit string length of 80 is 79. Simulated annealing consistently found the global optima. MIMIC never returned the global optima, and took 33 times as long to arrive at a result. The genetic algorithm performed almost as well as MIMIC, and was faster than simulated annealing. Randomized hill climbing is dominated by all others in this application, failing to find any optima more fit than the least optimum found with other methods.

The maximum possible fitness for the four peaks given a bit string length of 80 and a peak size of 8 is 151. Randomized hill climbing consistently found the local optima that just maximized the head or tail, never meeting the peak criteria. Simulated annealing, MIMIC and the standard genetic algorithm met the peak criteria more often than not, but only simulated annealing and mimic found the global optimum in the iterations allotted. MIMIC's median optima were better than those found by simulated annealing. MIMIC has a better chance of keeping instances that meet the peak criteria to train its distribution once they are discovered, and then reproducing something resembling them in sampling. Simulated annealing merely has a means of escaping the basin of attraction of the local optima that always traps randomized hill climbing, but random chance determines whether the hot random walk will meet the peak criteria and become attracted to the global optimum before the cooler fitness improvement phase takes over.

The maximum possible fitness for the continuous peaks problem given a bit string length of 60 and a peak size of 6 is 113. It is very similar to the four peaks problem but the criteria for fitness are relaxed as the calculations from the lengths of leading 1s and trailing 0s are replaced by the lengths of the longest consecutive run of 1s and longest consecutive runs of 0s. All algorithms were able to find optima that met the peak criteria. Simulated annealing and MIMIC both performed very well, but failed to find the global optima. Simulated annealing found better maximum and median optima than MIMIC. Randomized hill climbing found a better maximum optima than the standard genetic algorithm, but the latter had less variance and a higher central tendency.

The maximum possible fitness for the knapsack problem varies for each iteration. The weights and values of the items to fill the knapsack are randomly generated according to certain parameters, and each algorithm strives to maximize the value of the knapsack with the problem at hand. At this task, MIMIC consistently outperforms all other algorithms tested. This may have to do with the structure of the learned distribution. Knapsack problems are frequently solved with dynamic programming, and the strength of the dynamic programming approach is that, when broken into sequential stages, the optimal solution given the current stage must include the optimal solution of the next stage. Solutions are worked backwards by selecting the optimal decision given all prior decisions. In probabilistic optimization problems, the fitness distribution of the available choices at the current stage can be conditioned on all the prior choices. MIMIC is a similar process, working backwards from solutions, iteratively recalculating a distribution from the most valuable generated solutions, each variable conditioned on the value taken by the parent variable. The learned dependency tree represents a sequence of decisions of how many of each item to include at each stage, and each path taken through the tree resembles stages of a dynamic program.

Figure 4. Knapsack Results



Part 2. Unsupervised Learning

Discussion of Datasets for Clustering

The house-votes-84 dataset contains yea or nay votes on 16 selected issues for each house representative in 1984. The votes are labeled with party affiliation, either republican or democrat. Clustering algorithms should be effective at finding similar distributions of votes that align to party affiliation. Clusters may also align to political dimensions and coalitions not classified in this dataset. The only known ideological information is the party affiliation, so I examine how closely the clusters adhere to party lines, and how much extra information that is significant to the clusterer cannot be explained by party affiliation.

The credit dataset is a sample of loan applications from Japan. There are 15 attributes, anonymized, that characterize the loan application. There is a class attribute indicating whether the application was approved. Clustering algorithms may identify sets of features that correspond to likely acceptance or rejection of a loan. Classifying applications with cluster assignments that correspond to acceptance or rejection could make loan processing quicker and less error prone. Clusters that do not indicate one class or another could be used to flag a loan application for further review.

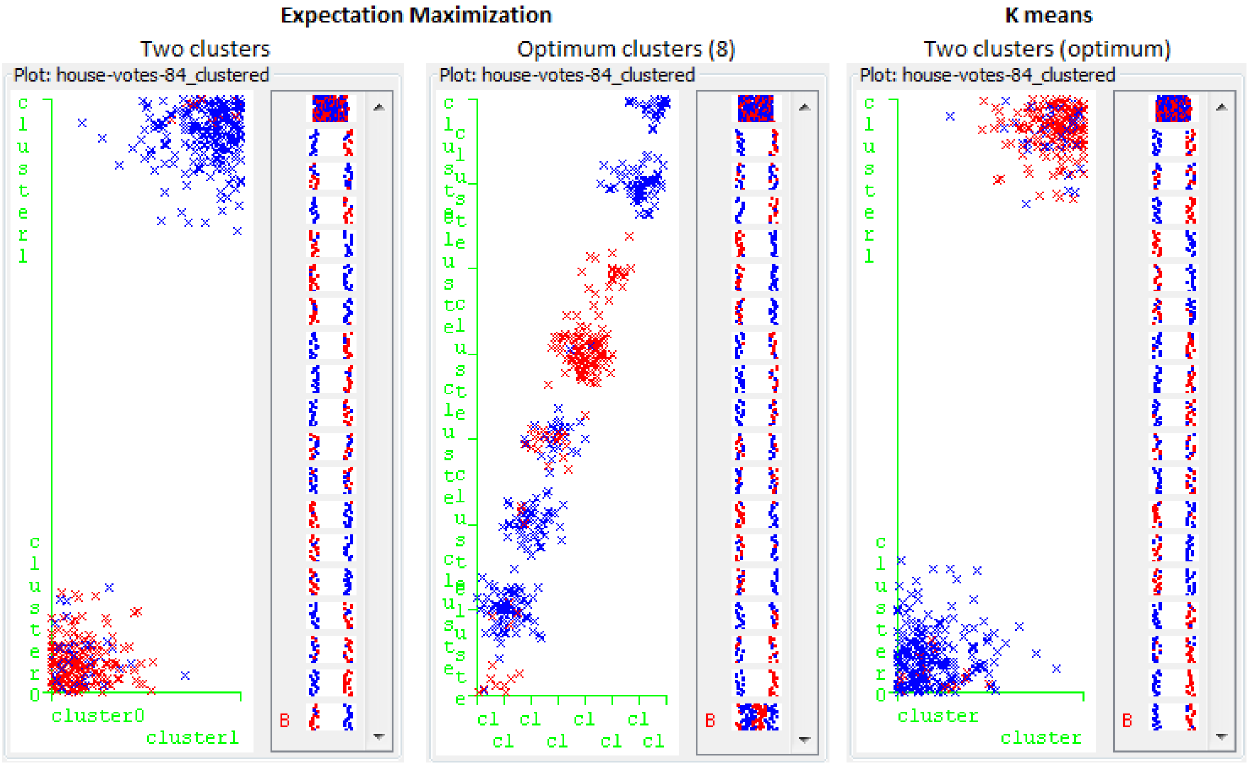
The loan-default-prediction dataset from assignment one is different from the credit dataset in that its loans were originated and either paid in full or recovered in default. Its instances categorize loans in many more dimensions, with a label indicating the percent loss in default. This dataset is a good candidate for feature reduction through projection and cluster labeling. In assignment one, the dataset proved too large to work with the full dataset, so here I use a small sample.

I used K-means and expectation maximization (EM) to estimate the distribution of clusters. I estimated two clusters by votes using each method, and observed how closely the ideological distribution aligns to republican and democrat labels. I also used Calinski-Harabasz (CH) criteria to find the optimum number of clusters with K-means, and cross validation error to find the optimum number of clusters with EM. I project the dataset into different feature spaces with principal components analysis (PCA), independent components analysis (ICA), and randomized projections (RP). I reduced features using PCA, ICA, RP and a random subset. I compare the various methods of feature reduction under projections to the random subset and discuss the benefits and tradeoffs of each

Clustering and Feature Reduction of House Votes

With the unmodified data and two clusters, both EM and K-means found clusters that were mainly composed of one party or the other. The best fitting K was two, and the number of clusters that minimized EM cross validation error was eight. In the eight cluster model, seven of the clusters still appear to be primarily single party, and one is either a bipartisan group that votes alike, or a catch-all cluster for misfits.

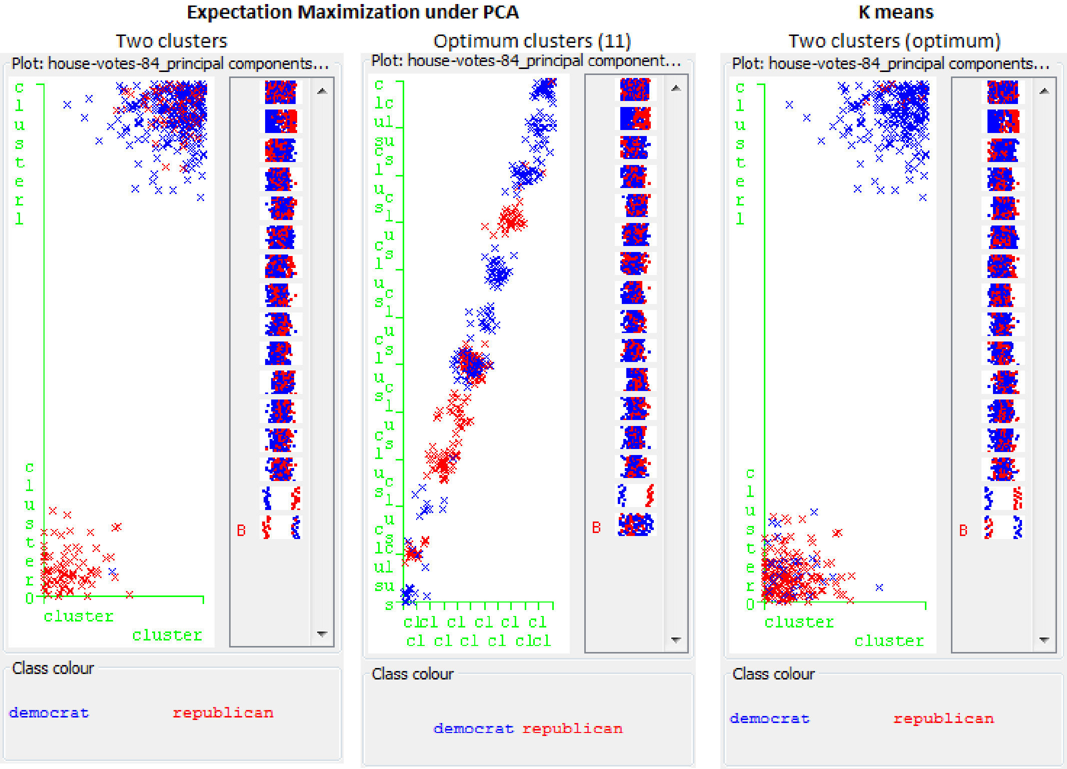
Figure 5. House Representatives clustered by votes, class coloring



The PCA projection yielded different clusters than the original data. The two cluster EM distribution found a smaller, primarily republican cluster with 80 members, and a second cluster with everyone else. Instead of finding two groups that approximate party affiliation, it found one nearly homogenous group, and placed everyone else in the other one. The optimum number of clusters with EM under PCA was 10. Some of the ten clusters were nearly all one party or the other, but one in particular was larger, and a clear mix of parties. I suspect that what EM did with the PCA data could be approximated by taking the most principal component, selecting the point in that line where the class mix begins to change, making a cluster of the homogenous side, and then taking the next most principal component and repeating the process for the unclassified instances, repeating until no components remain or cross validation error is not improved by adding additional clusters.

K-means clusters on the principal components found that the optimum number of clusters is two. Based on its apparent distribution of classes to clusters, it might be well approximated by splitting the principal component down the middle. This dataset's principal eigenvector has an eigenvalue of 7.27 compared to the next largest at 1.35. Most of the differentiating information is contained in the first component, so the best location for cluster centroids is at opposite ends of the principal component axis, centered around the center of mass of the data.

Figure 7. House Representatives clustered by principal components, class coloring



Expectation Maximization under PCA

Two clusters

Optimum clusters (11)

K means

Two clusters (optimum)

Plot: house-votes-84_principal components...
cluster1
cluster0
cluster
cluster
B
Class colour
democrat
republican
Plot: house-votes-84_principal component...
Plot: house-votes-84_principal components...

Figure 6. Class distribution on first and second principal components

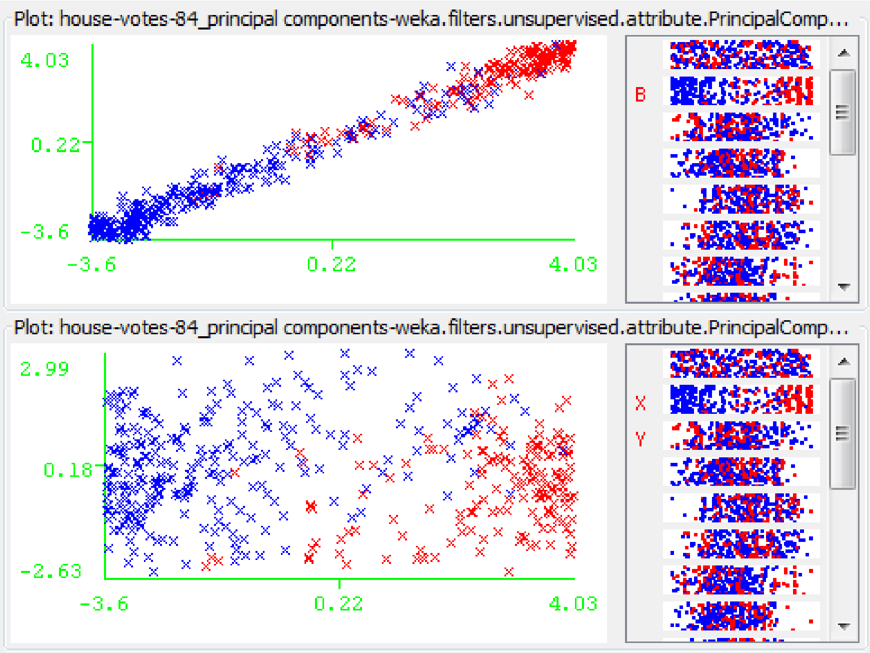
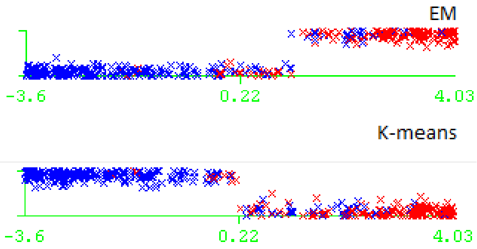


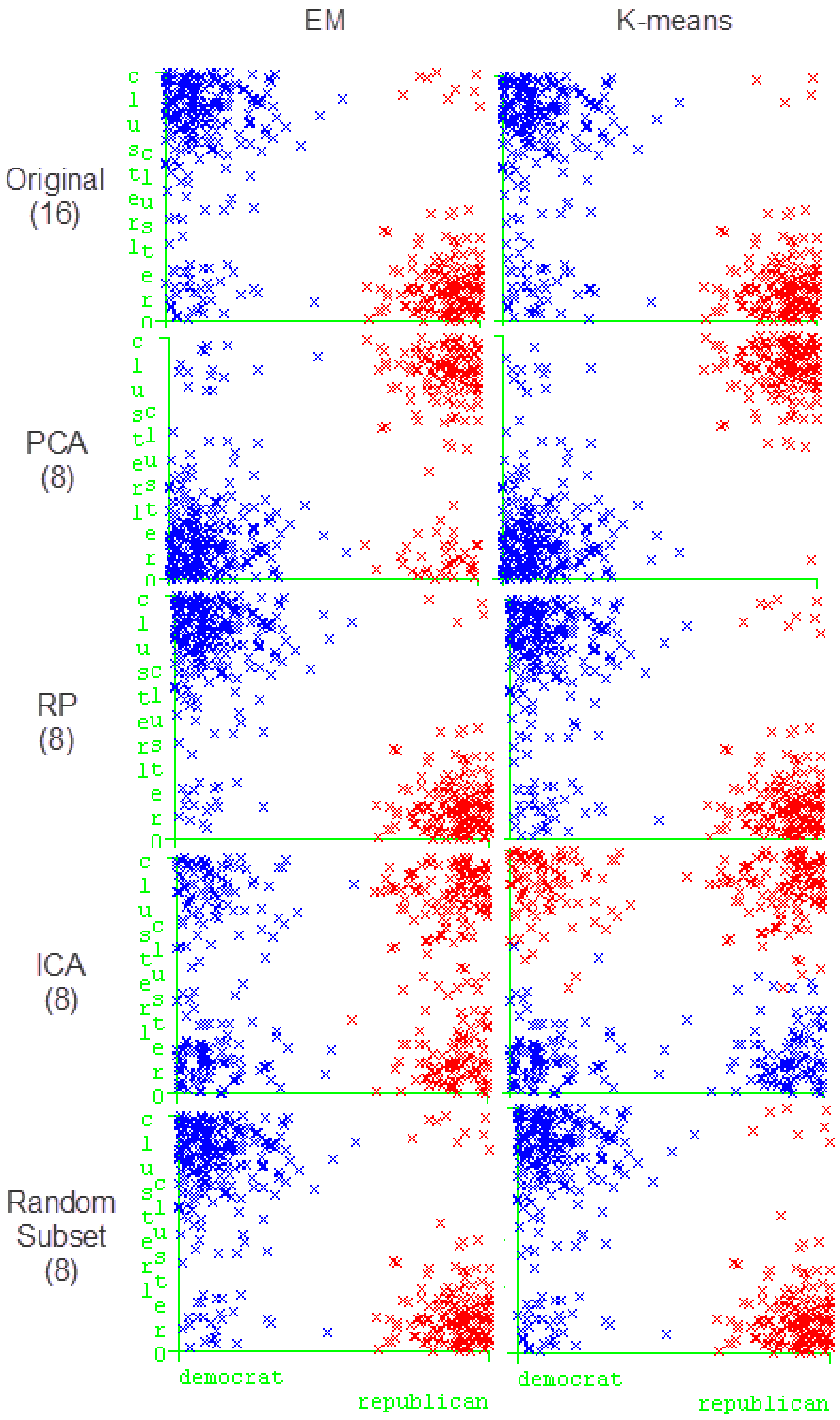
Figure 8. Cluster split on principal component



Comparing K-means to EM on just the principal component feature confirmed that K-means splits exactly on the middle of the principle component, while EM splits a point where the blend of classes is equal.

Principal component analysis packs information about the class into components and orders them by decreasing information. This allows the analyst to select components and control information loss. Randomized projection compresses the dataset into the selected number of components without regard to information loss. In this case, compressing 16 votes into 8 random projections did not reduce the class separation achieved by the EM algorithm, and only slightly reduces that of the K-means over the original dataset. EM finds 6 clusters to be optimal in cross validation, compared to 8 in the untransformed data, suggesting there is now less differentiation among instances possible in the random projection.

Figure 9. Separation of party affiliation by voting record on full dataset and reduced feature projections into two clusters



Separates classes

Separates one class more than the other

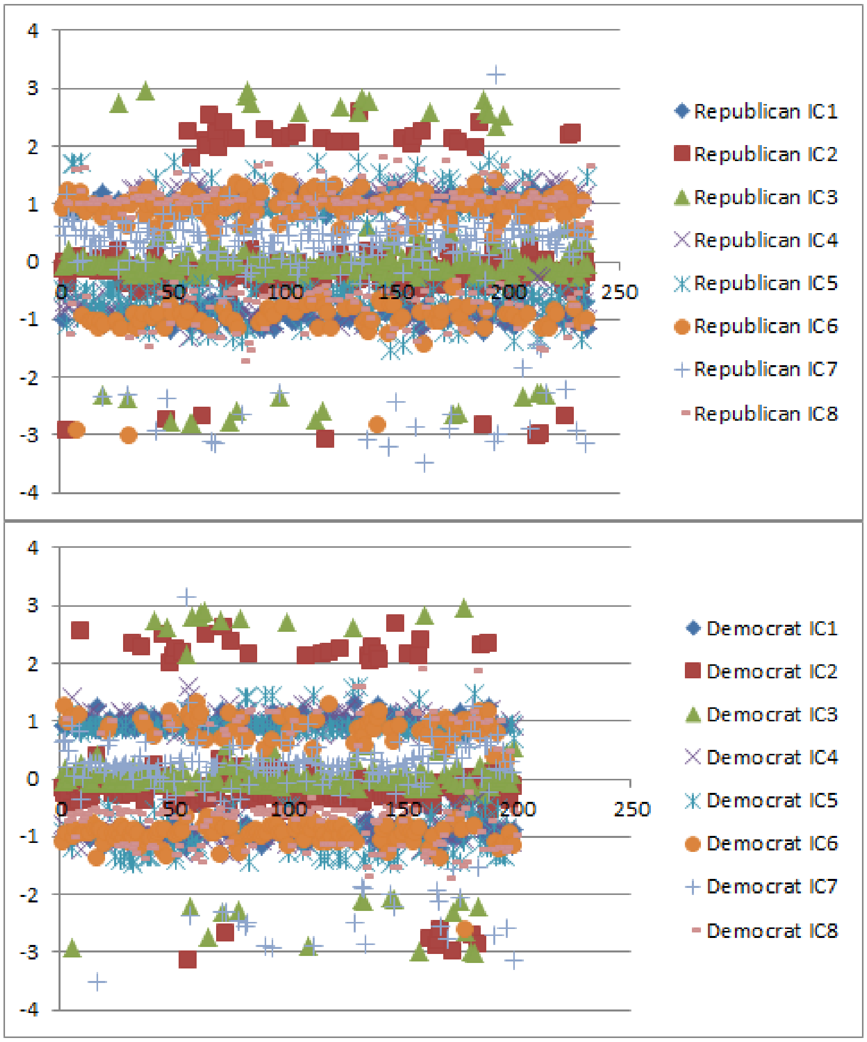
Separates classes

Does not separate classes

Separates classes

In figure 9 there is a clear difference in the clusters found by K-means in the reduced PCA dataset from those found in the original dataset. One cluster is almost entirely democrat, and the other cluster contains the remainder of the representatives. On the eight-dimension PCA political spectrum derived from these votes, most democrats and one republican distanced themselves from the rest of the house. Expectation maximization found largely the same clusters under PCA as in the original dataset, indicating it is not significantly affected by the transformation. It follows that K-means is more sensitive to changes in relative distance between instances during projections, since its membership is based on raw distance. The K-means algorithm did not find significantly different class distributions in the RP and the random subset, suggesting that they did not change the relative distance between instances. K-means did find radically different clusters in the reduced ICA dataset, as did EM. This suggests that the relative distance between instances changed, and the information differentiating democrat from republican in the untransformed data was lost in the ICA projection. It's possible that the independent signals extracted from the data give no information about the class.

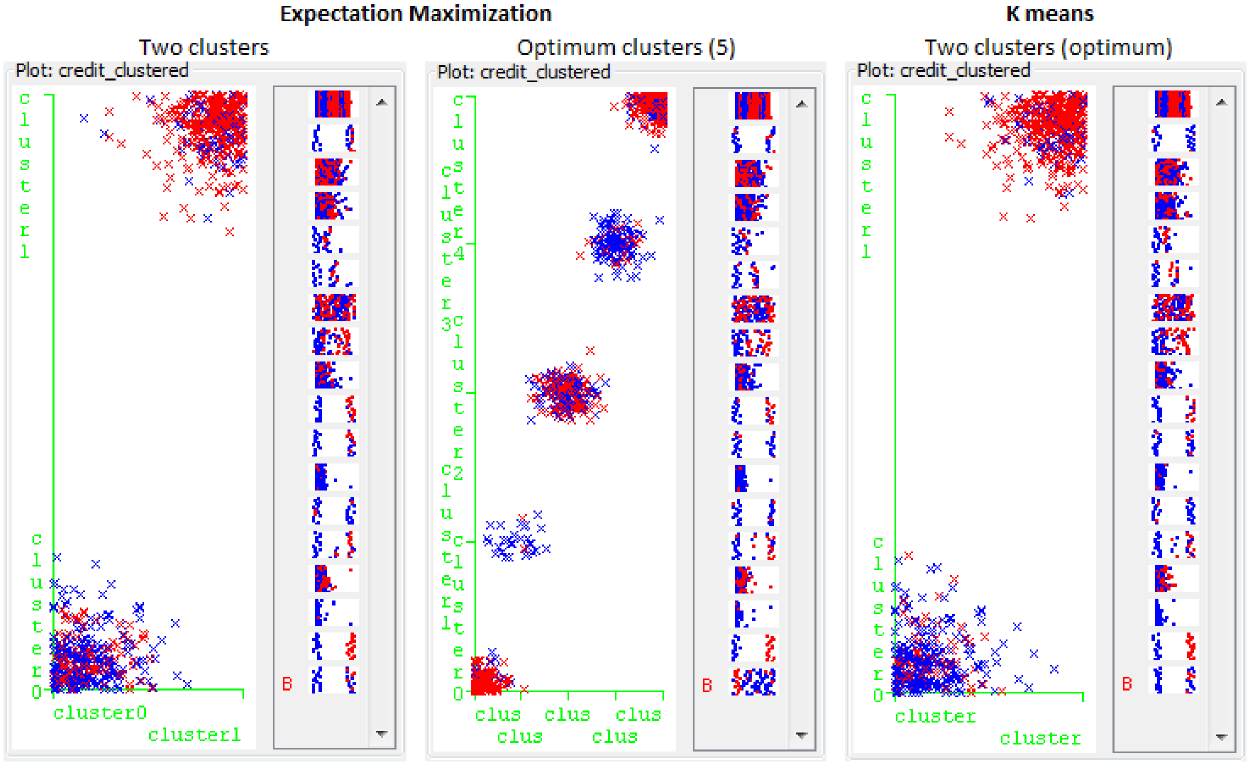
Figure 10. Plots of independent components by class



There is no obvious difference between the components in the ICA projection among republicans and democrats. Whatever signals were extracted, they are not related to the class variable.

Clustering and feature reduction of loan applications

Figure 11. Loan application clusters by class



Expectation Maximization

Two clusters

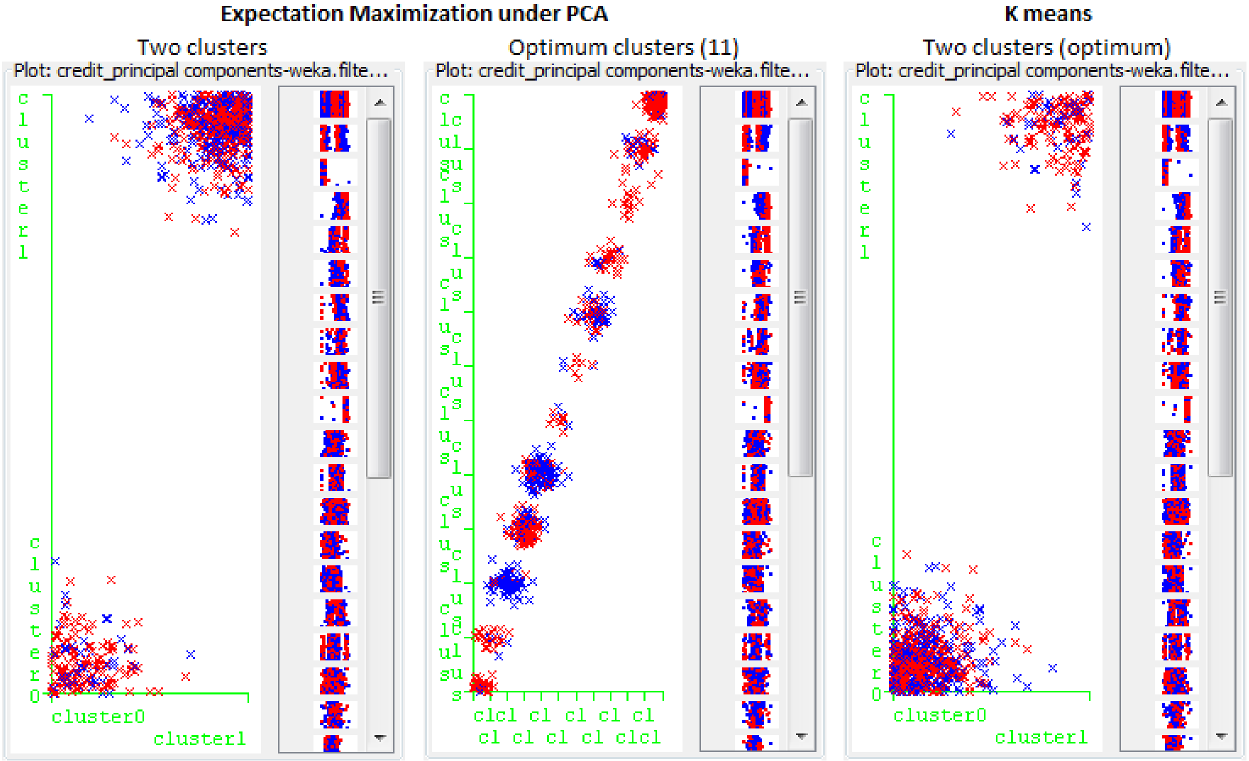
Optimum clusters (5)

K means

Two clusters (optimum)

The loan application dataset did not cluster along class lines as neatly as the house votes when two clusters were estimated. This suggests that there may be criteria in accepting or rejecting a loan application that are not present in the given factors. Four clusters of five in the optimal EM distribution have a majority class. One cluster contains a more even mix, again resembling a catch-all class. This appears to be a feature of EM. The five cluster EM model might be more useful for finding discriminating cluster labels, with two cluster for each class label and the third for instances that aren't part of the cluster. Using the five cluster model to label instances with the majority class correctly labels 72.3% of instances. The two cluster model correctly labels only 65.5% of instances. As comparison, a decision tree model correctly labels 86% on the training set in cross validation. Clustering may be useful to see what approved or denied loan applications have in common in clusters.

Figure 12. Loan application clusters by class, from PCA projection



Expectation Maximization under PCA

K means

Two clusters

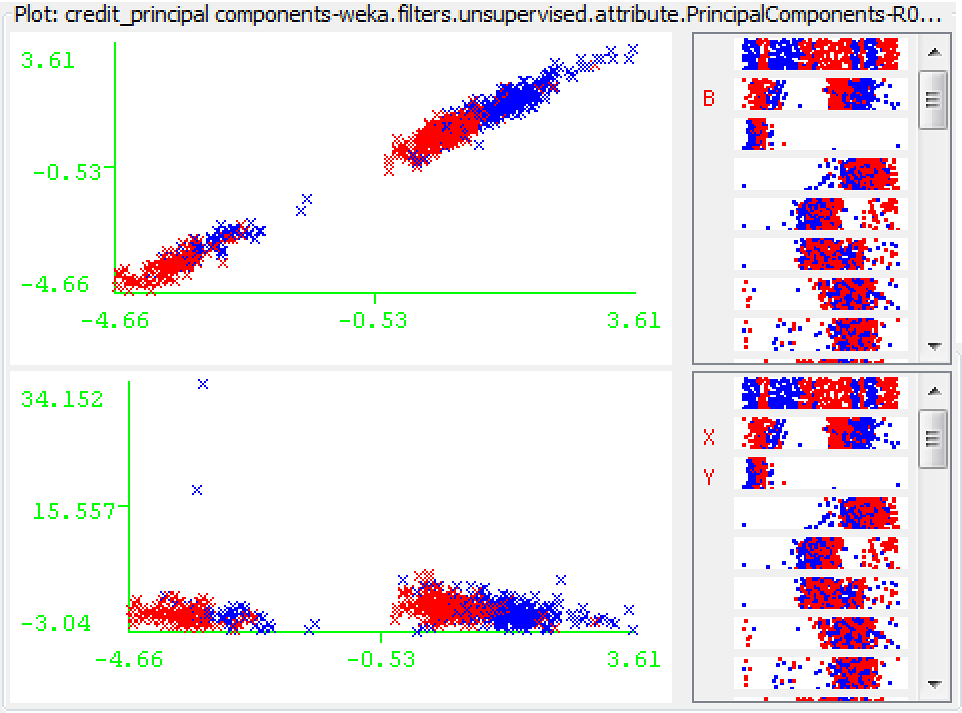
Optimum clusters (11)

Two clusters (optimum)

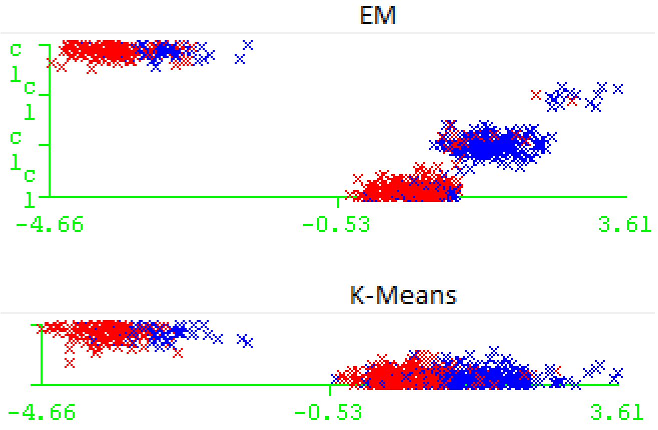
Plot: credit_principal components-weka.filte…
cluster1
cluster0
cluster0
cluster1
Plot: credit_principal components-weka.filte…
Plot: credit_principal components-weka.filte…
cluster1
cluster0
cluster0
cluster1

Under PCA, even using only the first eight principal components, the EM algorithm found eleven clusters in the optimal distribution. There are still mixtures of classes within clusters. Only 63% are correctly classified by the majority class of the most likely cluster in the eleven cluster model. The two cluster model produces exactly the same number of incorrectly classified instances. PCA enables more differentiation between instances, and identifies more clusters, but those clusters are less useful for classification of loan applications.

Class distribution on principal component, two principal components


Plot: credit_principal components-weka.filters.unsupervised.attribute.PrincipalComponents-R0...
3.61
-0.53
-4.66
-4.66
-0.53
3.61
B
34.152
15.557
-3.04
-4.66
-0.53
3.61
X
Y

Cluster and class distribution on principal component


EM
c
l
c
l
c
l
c
l
K-Means