Feature selection is a very important technique in machine learning. We need to be able to solve it to produce models. It's not an easy technique though. Feature Selection requires heuristic processes to find an optimal machine learning subset.

In the previous post I discussed the brute force algorithm as well as forward selection and backward elimination which were both not a great fit. What other options are there? We can use one of the most common optimization algorithms for multi-modal fitness landscapes; evolutionary algorithms.

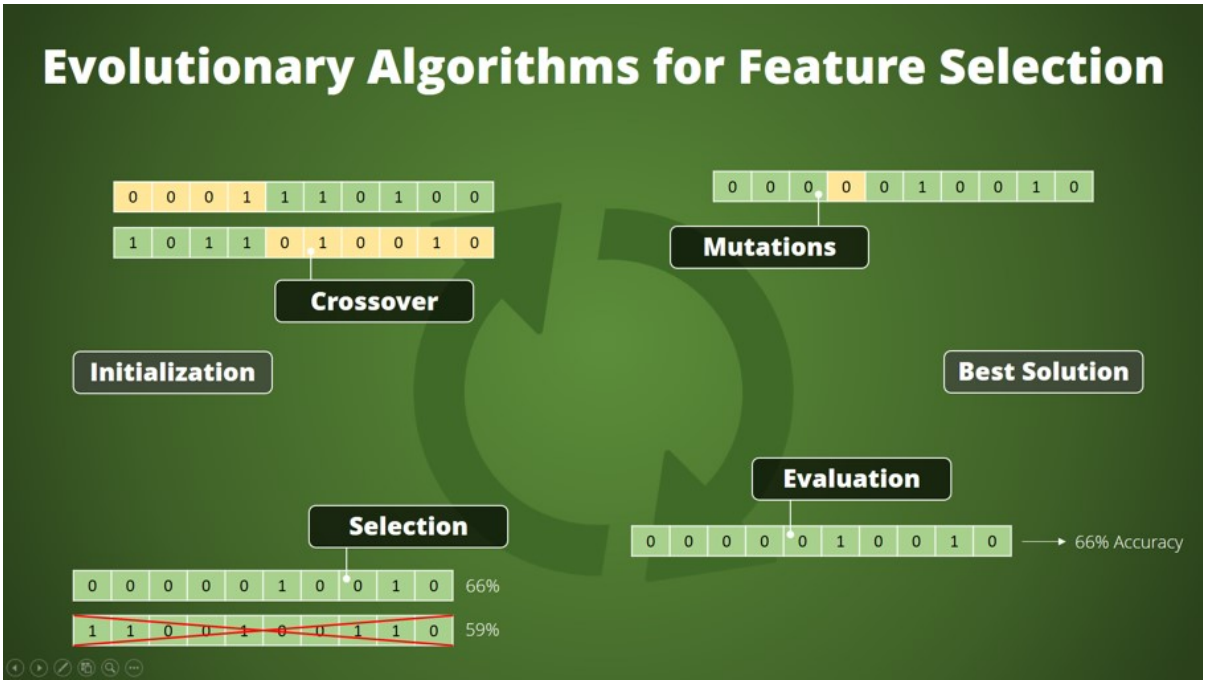
How do Evolutionary Algorithms work?

Evolutionary algorithm is a generic optimization technique mimicking the ideas of natural evolution. There are three basic concepts in play. First, parents create offspring (*crossover*). Second, there is a chance that individuals undergo small changes (*mutation*). And third, the likelihood for survival is higher for fitter individuals (*selection*).

We now know that we can represent possible attribute subsets with bit vectors. We can represent the selection of all attributes of a data set which consists of 10 attributes in total such as (1 1 1 1 1 1 1 1 1 1). A bit vector for the subset which only uses the third attribute would be (0 0 1 0 0 0 0 0 0 0) instead.

The first step of an evolutionary algorithm is to create a population of individuals. This population will evolve over time. We call this the *initialization* phase of the evolutionary algorithm. The individuals in our starting population are randomly generated. And each individual is a bit vector like those described above. We create individuals by tossing a coin for each available attribute. Depending on the result we either include the attribute in this individual or not. There are no fixed rules for the size of a starting population. However, we should have at least 2 individuals in the population if we want to do crossover. A good rule of thumb is to use between 5% and 30% of the number of attributes as population size.

After we create the initial population, there is a series of steps we need to keep performing. That is until we reach a stopping criterion. The image below depicts one possible way to implement such a loop. Please note that there are different approaches for the implementation of evolutionary algorithms. Some people prefer to perform the selection only for crossover. Others first select a new generation and perform crossover afterwards. Some people mutate only the offspring, but others also mutate parents. In my experience, the exact details of the algorithm have only little effect on how well they perform so I will explain only one variant below.



Step one

The first step is to pick parents from our population for *crossover*. I prefer to randomly pick parents from my population until I use all parents in exactly one crossover. Since the selection of survivors happens directly before (see below), we do not need to use the fitness for the parent selection. The selection step will already make sure that we mate only fitter parents in the next generation. There are many variants to implement this.

There are also many variants for the actual crossover. The variant in the picture above uses a single split point at the same position for both parent bit vectors. We create the offspring by using the front part from the first parent combined with the back part of the second parent and vice versa. Crossover can lead to large jumps in the fitness landscape. These jumps allow evolutionary algorithms to cope much better with multi-modal fitness landscapes. They simply have less likelihood to get stuck in a local extremum.

Step two

The next step is to perform *mutation*. Mutation means that we flip a single bit from 0 to 1 or the other way around. Many people only mutate the offspring of crossover, but I prefer to also mutate the parents. Please note that sometimes there is no mutation at all anyway. The likelihood for flipping the selection of a single attribute is $1 / m$ if m is the number of attributes. That means that we flip on average one attribute for each individual in the mutation. Mutation leads to a small movement in the fitness landscape. It can be useful to climb up towards a close-by extremum in the landscape. Finally, we can decide if we want to keep the original individuals as well or only keep the mutated individuals. I often keep both.

If we do decide to keep the original and the mutated individuals, then our population size is now 4 times bigger. We first created two new offspring from each pair of parents which doubled the population size. And then we created one mutation for each individual which doubles that population size again. In the next step, we should bring the population size down again to a smaller number. And we should prefer fitter individuals for survival while we are doing this.

Step three

But first we will *evaluate* all individuals in our current population. We call this the evaluation phase of the evolutionary algorithm. We can, for example, use the accuracy of a cross-validated model trained on this feature subset. We store those accuracies together with the individuals, so we can perform a fitness-driven selection in the next step.

Step four

The last step in our iterative process is *selection*. This step implements the concept of "survival of the fittest". That means that individuals, i.e. attribute subsets, which lead to models with a higher accuracy should have a higher likelihood to survive. And survivors build the basis for new attribute sets in the next generation. There are many ways of performing a selection. We will in fact see in my next blog post that we have several interesting options for solving the feature selection problem in even better ways. But for now, we go with one of the most widely used selection schemes, namely *tournament selection*.

For tournament selection we pit a small group of individuals against each other. The fittest one of the group survives and makes it into the next generation. We use sampling with replacement for creating our tournament groups. This means that individuals can be also selected multiple times. We keep pitting individuals until the population of the next generation has reached the desired size again.

The group size has some impact on how likely it is that weaker individuals have a chance to survive. If the size of the tournament is equal to the population size, only the fittest individual will survive. But with a tournament size of only 2, even weaker individuals have a chance if they are up against another weak candidate. We can adapt the so-called *selection pressure* by picking different values for the tournament size. But be careful: too large and the algorithm might become too greedy. This increases the likelihood for sub-optimal results. But if the tournament size is too low, the optimization runs longer.

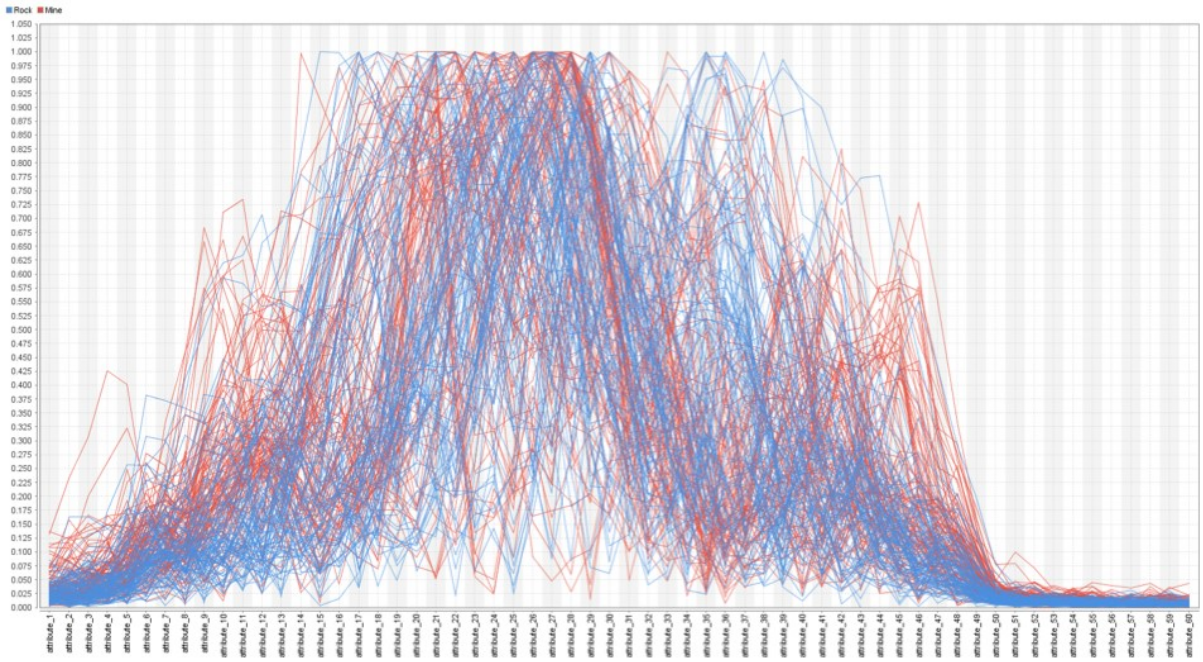
The loop ends when we meet a *stopping criterion*. This can be a maximum number of generations or the fact that there was no improvement any longer for some time. Or the optimization reached a time limit. But whatever the reason was, we deliver the best individual so far as result.

How can we easily use Evolutionary Feature Selection?

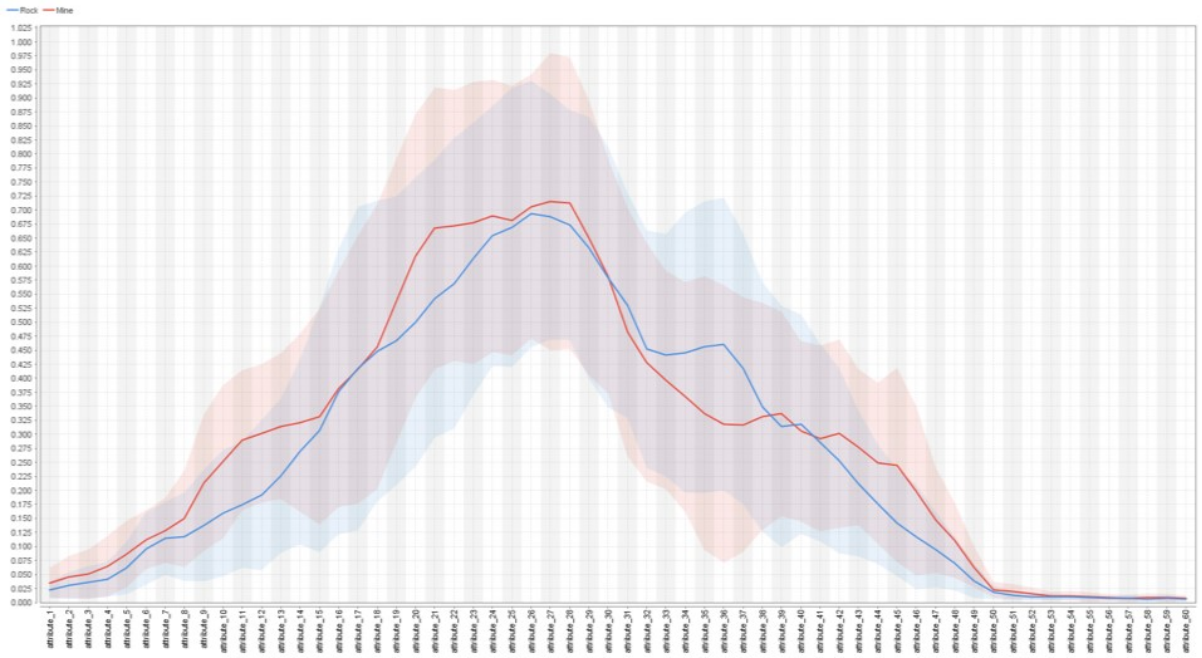
Let's discuss how we can setup those different heuristics for feature selection in RapidMiner. The RapidMiner processes can be downloaded here.

There have been numerous papers showing that evolutionary feature selection is effective. In this blog post, we want to show that the results of each of the different approaches can differ. We will use a single data set. Keep in mind that the approaches are heuristics and results can be completely different for other data sets, but in practice we will often see patterns like those we will discuss below.

The data set we will analyze is the Sonar data. It has 208 examples (rows, observations, cases...) and each one is a frequency spectrum of objects in the water. In total, we have 60 different frequency bands which form the attributes of our data set. Our goal is to detect from the frequency spectrum of an object if this object is a rock or a naval mine. The image below shows a visualization of the data in a parallel plot. Each line in this chart is representing one example in the data set. The color specifies if the corresponding object is a rock (blue) or a mine (red).



As we can see, this is a pretty hard task, but there are certain areas in the frequency spectrum where the two classes differ a bit more. These areas are even easier to see in the deviation chart below. This chart only shows the mean values for each attribute of both classes. The transparent regions represent the standard deviations of the attributes for each class.



There are 4 different areas of the frequency spectrum where the two classes differ most. Those are the areas around attribute 11, attribute 20, attribute 35, and attribute 45. We saw in the first image that a single area will be sufficient to find a good model. But some combination of attributes in those areas should help a lot to get over the noise.

Process 1: Cross-validation of Naïve Bayes on Complete Data (Benchmark)

Before we do any feature selection, let's first establish a benchmark by evaluating how well our model works on the complete data set. We will use Naïve Bayes as model type for the experiments here. And we will evaluate all models with the same 5-fold cross-validation. This means that there is no impact from different data splits during our experiments. We can do this in RapidMiner by defining a local random seed for the cross-validation operator.

As mentioned before, the process can be found at the end of this blog post. We achieve **67.83%** accuracy on the Sonar data set with Naïve Bayes. And of course, we expect that feature selection will improve the accuracy even further. We will not try the brute force approach here though. For the 60 attributes (frequency bands) of our data set, we would end up with $2^{60} - 1 =$ 1,152,921,504,606,849,999 combinations.

Process 2: Forward Selection

The next experiment we perform is a forward selection approach. We use the same 5-fold cross-validation as used above to evaluate the performance of each attribute subset. The optimization stopped already after 4 rounds and delivered attributes 12, 15, 17, and 18. The accuracy of this subset was **77.43%**. This is much higher than our benchmark without any feature selection. But another thing is worth noticing. The feature selection has only selected attributes from the left two areas out of the 4 relevant areas of the frequency spectrum. The right two areas have not made it into the result. The forward selection simply stopped before after running into a local extremum.

Process 3: Backward Elimination

We will now try backward elimination as an alternative as it might do a better job in finding a good model considering all four important areas of the frequency spectrum. Unfortunately, this isn't the case. This algorithm only removed 8 out of the 60 attributes from the data set. All other noise attributes stayed in the data set before the backward elimination was running into the first local extremum. The lower accuracy of only **73.12%** should not come as a result then, although it is still better than using the full data. At least we can see that getting rid of those 8 attributes paid off already. But there is still too much noise in the data which covers the actual signals.

Process 4: Evolutionary Feature Selection

Lastly, let's look at the evolutionary approach for feature selection. We use a population size of 20 and stop the optimization after a maximum of 30 generations. The optimization runs slightly longer than forward selection or backward elimination. But the accuracy is clearly higher: **82.72%** and with that the best result we have achieved. What is equally interesting is the subset of 11 attributes selected by the algorithm:

- attribute_6
- attribute_11
- attribute_12
- attribute_16
- attribute_20
- attribute_21
- attribute_28
- attribute_36
- attribute_39
- attribute_47
- attribute_49

This subset covers all four important areas of the frequency spectrum. We have attributes around attribute 11, attribute 20, attribute 35, and attribute 45. Which is exactly the reason why this feature selection has outperformed the other approaches. The evolutionary algorithm was simply not running into local extrema and stopped too early. It kept going until it found a much better result.

Key Takeaway

Forget about forward selection and backward elimination. Evolutionary feature selection should be the technique used to solve this problem. In my next blog post I will discuss how we can get better results and additional insights by not only optimizing for the most accurate model alone. Instead, we will also optimize for the simplest model at the same time. This will be the topic for the next blog post though.

RapidMiner Processes

Make sure to have RapidMiner downloaded. Then download the processes here to build this machine learning model yourself in RapidMiner.

- Process 1: Cross-validation of Naïve Bayes on Complete Data
- Process 2: Forward Selection
- Process 3: Backward Elimination
- Process 4: Evolutionary Feature Selection

Please download the zip-file and extract its content. The result will be an .rmp file which can be loaded into RapidMiner via “File” -> “Import Process”.