1.Representation: A classifier must be represented in some formal language that the computer can handle. Conversely, choosing a representation for a learner is tantamount to choosing the set of classifiers that it can possibly learn. This set is called the hypothesis space of the learner. It is basically the space of allowed models (the hypothesis space), but also considers the fact that we are expressing models in some formal language that may encode some models more easily than others (even within that possible set). This is akin to the landscape of possible models, the playing field allowed by a given representation. For example, 3-layer feedforward neural networks (or computational graphs) form one type of representation, while support vector machines with RBF kernels form another.

Example: Instances

K-nearest neighbor

Support vector machines

Hyperplanes

naive bayes

logistic regression

Decision trees

Sets of rules

Propositional rules

logic programs

neural networks

Graphical models

bayesian networks

conditional random field

2.Evaluation: An evaluation function (also called objective function or scoring function) is needed to distinguish good classifiers from bad ones. The evaluation function used internally by the algorithm may differ from the external one that we want the classifier to optimize, for ease of optimization and due to the issues, I will discuss. It's essentially how you judge or prefer one model vs. another; it's what we might have seen as a utility function, loss function, scoring function, or fitness function in other contexts. Think of this as the height of the landscape for each given model, with lower areas being preferable/desirable than higher areas (without loss of generality). Mean squared error (of a model's output vs. the data output) or likelihood (the estimated probability of a model given the observed data) are examples of different evaluation functions that will imply somewhat different heights at each point on a single landscape.

Example: Accuracy/error rate

Precision and recall

Squared error

likelihood

Posterior probability

Information gain

K-l divergence

cost/Utility

Margin

3.Optimization: Finally, we need a method to search among the classifiers in the language for the highest scoring one. The choice of optimization technique is key to the efficiency of the learner and helps determine the classifier produced if the evaluation function has more than one optimum. It is common for new learners to start out using off-the-shelf optimizers, which are later replaced by custom-designed ones. finally, is how you search the space of represented models to obtain better evaluations. This is the way we expect to traverse the landscape to find the promised land of ideal models; the strategy of getting to where you want to go. Stochastic gradient descent and genetic algorithms are two (very) different ways of optimizing a model class. Note that once we provide a trained model, it's quite possible that you may no longer be able to recover exactly how it was optimized. Just because we can see someone standing in the pit doesn't mean we know how they got there.
Example: combinatorial optimization
Greedy search
beam search
branch-and-bound
continuous optimization
Unconstrained
Gradient descent
conjugate gradient
Quasi-newton methods
constrained
linear programming
Quadratic programming
----
4. There is different combination of features of the three components and the component of representation is most popular to solve different problems of machine learning. But I believe that combining other components as well to find the best result should be the approach.
There is a brief discussion regarding the importance of the three components:
A learning algorithm is an algorithm that learns the unknown model parameters based on data patterns. Data are linked to models Generally, there are three different angles components.
Representation: Find the mapping (i.e., function) that relates the input to the output f(x) = y. The function f maps data from the input space x to some output values y. f isn't necessary assumed to have a specific functional form (i.e., non-parametric).
Decide the model that will structure our problem:

- decision theory in RL algorithms,
- proximity in KNN,
- rules/split points in decision trees etc.

Representations include models of states (e.g., dynamic Bayesian networks, Markov decision process), models of instances (e.g., KNN, SVM), rules (e.g., decision trees), hyperplanes (e.g., Naive Bayes, logistic regression, linear regression), or artificial neural networks (a combination the previous). That distinction is not strict, and representations can be combined.
Feature Engineering: It is a representation problem that turns the inputs x into 'things' the algorithm can understand. How to represent the x in a feature or in a state space? A good set of features brings you closer to the true underlying structure of the problem that we are trying to

learn. If we follow the thought that we'll include many features in the model to provide it with all the important information of the problem, we will probably end up with an overfitted model.
Bias vs Variance Trade off: There are modeling steps that helps you find out whether you have an incorrect description of the problem that you are solving with the algorithm. These are repeatedly intervening the learning process. They include:
Compare bias (under-fitting) and variance (overfitting) of the model
Balance bias and variance of the model.
Analyze the performance of the model.
Overfitting: The model is being sticked to random patterns instead of the true evidence that generated the dataset. Although an overfitted model will predict well in the current data set, it won't generalize to new data sets.

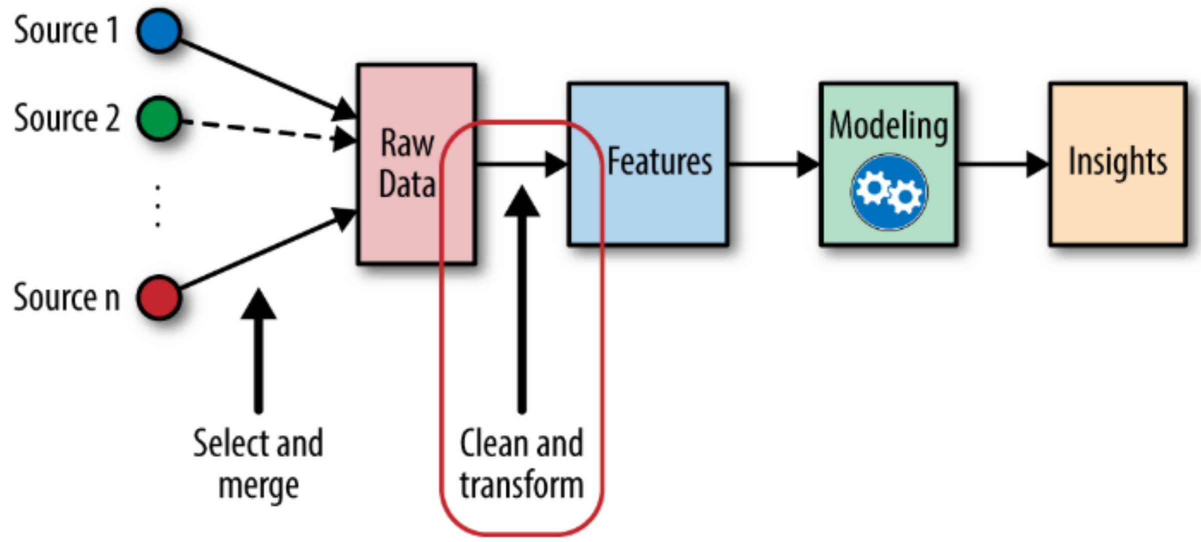
Evaluation: Evaluate the loss function given different values for the parameters. The loss function is also called a utility, scoring, or a reward function. An objective function is a loss function that desired to be maximized. It may be different for the internal algorithm (e.g., log loss) and the external model (e.g., AUC for the ROC). The evaluation function of the external model is typically called a performance metric. The function being learned depends on the representation you choose including the output type of the function:
Classification: outputs discrete values, usually the log (likelihood) loss objective. Discrete values may be probabilities, in which case a cross-entropy loss is used.
Regression: continuous output, usually the mean squared error (or least squares).
Density Estimation: outputs a probability function of the data
Reinforcement: in the simplest form the output is a scalar value, and the objective function is the value function.

Optimization: a (solver) function that seeks the optimal values of the loss function. It estimates the parameters and select those values that result in the lowest error (minimum) or highest reward value (maximum).
Combinatorial (e.g., local search, greedy search), unconstrained continuous (e.g., gradient descent), and constrained continuous functions (e.g., linear programming) are types of optimization functions.
Analytic vs Numerical methods: For some objectives, the optimal parameters can be found exactly, which is known as the analytic solution. For other objectives, the parameters need to be approximated with a computational algorithm. To illustrate that with an example, an analytic method in a linear regression model is to solve the equations of least squares; whilst a numerical method is to compute the optimal values with maximum likelihood estimates.
Bayesian networks: Naive Bayes, as the second assumes independence among the features.
Probabilistic graphical models: artificial neural nets, as the latter assumes some nodes to be non-linear computational units. In the hidden layer, each node performs a sigmoid function or rectifier linear unit given its inputs from the previous layer. The nodes are usually stacked in 3 layers, the input, output, and the hidden layer.

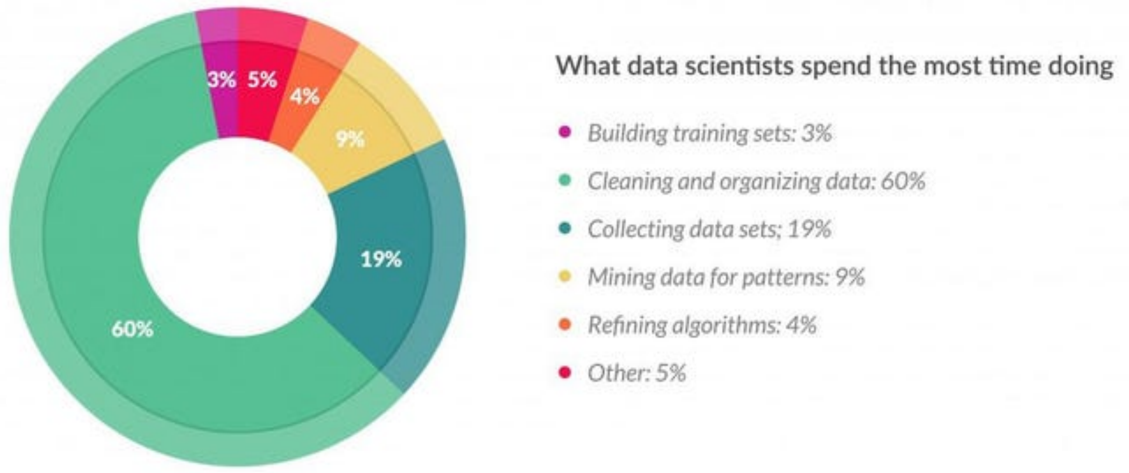
Therefore, considering all the different aspects of the components and their uses I would like to suggest that we should consider all the components to cover various problems.

----

5. For me feature engineering is the most interesting part. Because:
Feature Engineering: Feature Engineering is the process of extracting and organizing the important features from raw data in such a way that it fits the purpose of the machine learning model. It can be thought of as the art of selecting the important features and transforming them into refined and meaningful features that suit the needs of the model.



Why is Feature Engineering so important?
What it takes the maximum amount of time and effort in a Machine Learning workflow?
Well to analyze that, let us have a look at this diagram.



This pie-chart shows the results of a survey conducted by Forbes. It is abundantly clear from the numbers that one of the main jobs of a Data Scientist is to clean and process the raw data. This can take up to 80% of the time of a data scientist. This is where Feature Engineering comes into play. After the data is cleaned and processed it is then ready to be fed into the machine learning models to train and generate outputs.

Benefits of Feature Engineering

An effective Feature Engineering implies:

Higher efficiency of the model

Easier Algorithms that fit the data

Easier for Algorithms to detect patterns in the data

Greater Flexibility of the features

Dimensionality Reduction

Reduces computation complexity

Well, cleaning up bulks of raw, unstructured, and dirty data may seem like a daunting task, but that is exactly what this guide is all about.