Rajat Harlalka Follow
Entrepreneur | Operating Partner at GSF
Jun 18 · 10 min read

Choosing the Right Machine Learning Algorithm

Machine learning is part art and part science. When you look at machine learning algorithms, there is no one solution or one approach that fits all. There are several factors that can affect your decision to choose a machine learning algorithm.

Some problems are very specific and require a unique approach. E.g. if you look at a recommender system, it's a very common type of machine learning algorithm and it solves a very specific kind of problem. While some other problems are very open and need a trial & error approach. Supervised learning, classification and regression etc. are very open. They could be used in anomaly detection, or they could be used to build more general sorts of predictive models.

Besides some of the decisions that we make when choosing a machine learning algorithm have less to do with the optimization or the technical aspects of the algorithm but more to do with business decisions. Below we look at some of the factors that can help you narrow down the search for your machine learning algorithm.

Data Science Process

Before you start looking at different ML algorithms, you need to have a clear picture of your data, your problem and your constraints.

Understand Your Data

The type and kind of data we have plays a key role in deciding which algorithm to use. Some algorithms can work with smaller sample sets while others require tons and tons of samples. Certain algorithms work with certain types of data. E.g. Naïve Bayes works well with categorical input but is not at all sensitive to missing data.

Hence it is important that you:

Know your data

1. Look at Summary statistics and visualizations

- Percentiles can help identify the range for most of the data
- Averages and medians can describe central tendency
- Correlations can indicate strong relationships

2. Visualize the data

- Box plots can identify outliers
- Density plots and histograms show the spread of data
- Scatter plots can describe bivariate relationships

Clean your data

1. Deal with missing value. Missing data affects some models more than others. Even for models that handle missing data, they can be sensitive to it (missing data for certain variables can result in poor predictions)
2. Choose what to do with outliers

- Outliers can be very common in multidimensional data.
- Some models are less sensitive to outliers than others. Usually tree models are less sensitive to the presence of outliers. However regression models, or any model that tries to use equations, could definitely be effected by outliers.
- Outliers can be the result of bad data collection, or they can be legitimate extreme values.

3. Does the data needs to be aggregated

Augment your data

1. Feature engineering is the process of going from raw data to data that is ready for modeling. It can serve multiple purposes:

- Make the models easier to interpret (e.g. binning)
- Capture more complex relationships (e.g. NNs)
- Reduce data redundancy and dimensionality (e.g. PCA)
- Rescale variables (e.g. standardizing or normalizing)

2. Different models may have different feature engineering requirements. Some have built in feature engineering.

Categorize the problem

The next step is to categorize the problem. This is a two-step process.

1. Categorize by input:

- If you have labelled data, it's a supervised learning problem.
- If you have unlabelled data and want to find structure, it's an unsupervised learning problem.
- If you want to optimize an objective function by interacting with an environment, it's a reinforcement learning problem.

2. Categorize by output.

- If the output of your model is a number, it's a regression problem.
- If the output of your model is a class, it's a classification problem.
- If the output of your model is a set of input groups, it's a clustering problem.
- Do you want to detect an anomaly ? That's anomaly detection

Understand your constraints

- What is your data storage capacity? Depending on the storage capacity of your system, you might not be able to store gigabytes of classification/regression models or gigabytes of data to clusterize. This is the case, for instance, for embedded systems.
- Does the prediction have to be fast? In real time applications, it is obviously very important to have a prediction as fast as possible.

For instance, in autonomous driving, it's important that the classification of road signs be as fast as possible to avoid accidents.

- Does the learning have to be fast? In some circumstances, training models quickly is necessary: sometimes, you need to rapidly update, on the fly, your model with a different dataset.

Find the available algorithms

Now that you a clear understanding of where you stand, you can identify the algorithms that are applicable and practical to implement using the tools at your disposal. Some of the factors affecting the choice of a model are:

- Whether the model meets the business goals
- How much pre processing the model needs
- How accurate the model is
- How explainable the model is
- How fast the model is: How long does it take to build a model, and how long does the model take to make predictions.
- How scalable the model is

An important criteria affecting choice of algorithm is model complexity. Generally speaking, a model is more complex is:

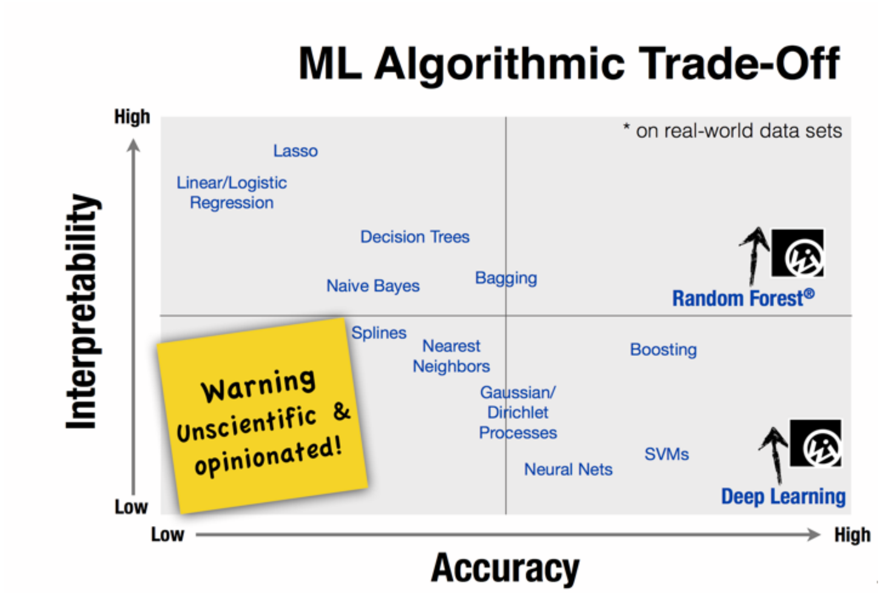
- It relies on more features to learn and predict (e.g. using two features vs ten features to predict a target)
- It relies on more complex feature engineering (e.g. using polynomial terms, interactions, or principal components)
- It has more computational overhead (e.g. a single decision tree vs. a random forest of 100 trees).

Besides this, the same machine learning algorithm can be made more complex based on the number of parameters or the choice of some hyperparameters. For example,

- A regression model can have more features, or polynomial terms and interaction terms.

- A decision tree can have more or less depth.

Making the same algorithm more complex increases the chance of overfitting.



(Overcoming the Barriers to Production-Ready Machine Learning Workflows)

Commonly used Machine Learning algorithms

Linear Regression

These are probably the simplest algorithms in machine learning. Regression algorithms can be used for example, when you want to compute some continuous value as compared to Classification where the output is categoric. So whenever you are told to predict some future value of a process which is currently running, you can go with regression algorithm. Linear Regressions are however unstable in case features are redundant, i.e. if there is multicollinearity

Some examples where linear regression can used are:

- Time to go one location to another

- Predicting sales of particular product next month
- Impact of blood alcohol content on coordination
- Predict monthly gift card sales and improve yearly revenue projections

Logistic Regression

Logistic regression performs binary classification, so the label outputs are binary. It takes linear combination of features and applies non-linear function (sigmoid) to it, so it's a very small instance of neural network.

Logistic regression provides lots of ways to regularize your model, and you don't have to worry as much about your features being correlated, like you do in Naive Bayes. You also have a nice probabilistic interpretation, and you can easily update your model to take in new data, unlike decision trees or SVMs. Use it if you want a probabilistic framework or if you expect to receive more training data in the future that you want to be able to quickly incorporate into your model. Logistic regression can also help you understand the contributing factors behind the prediction, and is not just a black box method.

Logistic regression can be used in cases such as:

- Predicting the Customer Churn
- Credit Scoring & Fraud Detection
- Measuring the effectiveness of marketing campaigns

Decision trees

Single trees are used very rarely, but in composition with many others they build very efficient algorithms such as Random Forest or Gradient Tree Boosting.

Decision trees easily handle feature interactions and they're non-parametric, so you don't have to worry about outliers or whether the data is linearly separable. One disadvantage is that they don't support online learning, so you have to rebuild your tree when new examples come on. Another disadvantage is that they easily overfit, but that's where ensemble methods like random forests (or boosted trees) come in. Decision Trees can also take a lot of memory (the more features you have, the deeper and larger your decision tree is likely to be)

Trees are excellent tools for helping you to choose between several courses of action.

- Investment decisions
- Customer churn
- Banks loan defaulters
- Build vs Buy decisions
- Sales lead qualifications

K-means

Sometimes you don't know any labels and your goal is to assign labels according to the features of objects. This is called clusterization task. Clustering algorithms can be used for example, when there is a large group of users and you want to divide them into particular groups based on some common attributes.

If there are questions like how is this organized or grouping something or concentrating on particular groups etc. in your problem statement then you should go with Clustering.

The biggest disadvantage is that K-Means needs to know in advance how many clusters there will be in your data, so this may require a lot of trials to "guess" the best K number of clusters to define.

Principal component analysis (PCA)

Principal component analysis provides dimensionality reduction. Sometimes you have a wide range of features, probably highly correlated between each other, and models can easily overfit on a huge amount of data. Then, you can apply PCA.

One of the keys behind the success of PCA is that in addition to the low-dimensional sample representation, it provides a synchronized low-dimensional representation of the variables. The synchronized sample and variable representations provide a way to visually find variables that are characteristic of a group of samples.

Support Vector Machines

Support Vector Machine (SVM) is a supervised machine learning technique that is widely used in pattern recognition and classification problems—when your data has exactly two classes.

High accuracy, nice theoretical guarantees regarding overfitting, and with an appropriate kernel they can work well even if you're data isn't linearly separable in the base feature space. Especially popular in text classification problems where very high-dimensional spaces are the norm. SVMs are however memory-intensive, hard to interpret, and difficult to tune.

SVM can be used in real-world applications such as:

- detecting persons with common diseases such as diabetes
- hand-written character recognition
- text categorization—news articles by topics
- stock market price prediction

Naive Bayes

It is a classification technique based on Bayes' theorem and very easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Naive Bayes is also a good choice when CPU and memory resources are a limiting factor.

Naive Bayes is super simple, you're just doing a bunch of counts. If the NB conditional independence assumption actually holds, a Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so you need less training data. And even if the NB assumption doesn't hold, a NB classifier still often does a great job in practice. A good bet if want something fast and easy that performs pretty well. Its main disadvantage is that it can't learn interactions between features.

Naive Bayes can be used in real-world applications such as:

- Sentiment analysis and text classification

- Recommendation systems like Netflix, Amazon
- To mark an email as spam or not spam
- Face recognition

Random Forest

Random Forest is an ensemble of decision trees. It can solve both regression and classification problems with large data sets. It also helps identify most significant variables from thousands of input variables. Random Forest is highly scalable to any number of dimensions and has generally quite acceptable performances. Then finally, there are genetic algorithms, which scale admirably well to any dimension and any data with minimal knowledge of the data itself, with the most minimal and simplest implementation being the microbial genetic algorithm. With Random Forest however, learning may be slow (depending on the parameterization) and it is not possible to iteratively improve the generated models

Random Forest can be used in real-world applications such as:

- Predict patients for high risks
- Predict parts failures in manufacturing
- Predict loan defaulters

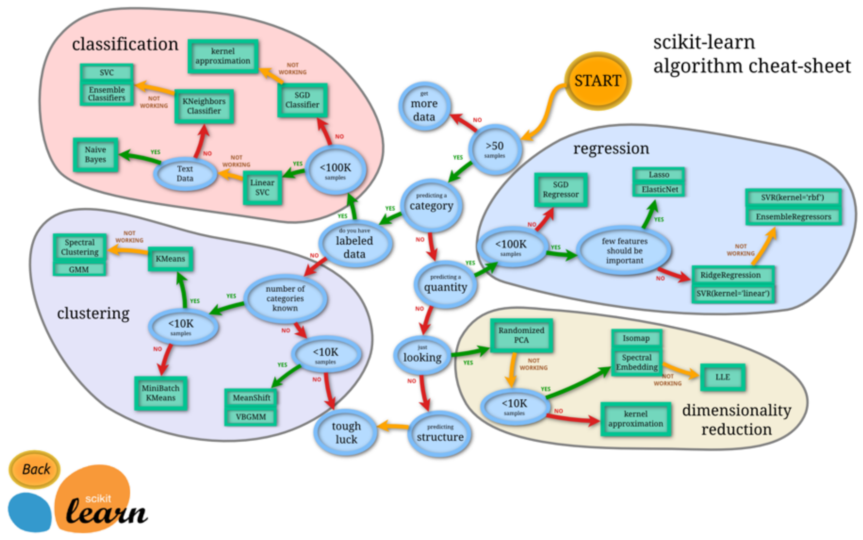
Neural networks

Neural Networks take in the weights of connections between neurons . The weights are balanced, learning data point in the wake of learning data point . When all weights are trained, the neural network can be utilized to predict the class or a quantity, if there should arise an occurrence of regression of a new input data point. With Neural networks, extremely complex models can be trained and they can be utilized as a kind of black box, without playing out an unpredictable complex feature engineering before training the model. Joined with the “deep approach” even more unpredictable models can be picked up to realize new possibilities. E.g. object recognition has been as of late enormously enhanced utilizing Deep Neural Networks. Applied to unsupervised learning tasks, such as feature extraction, deep learning also extracts features from raw images or speech with much less human intervention.

On the other hand, neural networks are very hard to just clarify and parameterization is extremely mind boggling. They are also very resource and memory intensive.

Scikit Cheat Sheet

Scikit learning has put a very indepth and well explained flow chart to help you choose the right algorithm that I find very handy.



(http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

Conclusion

Generally speaking you can use the points above to shortlist a few algorithms but it is hard to know right at the start which algorithm will work best. It is usually best to work iteratively. Amongst the ML algorithms you identified as potential good approaches, throw your data into them, run them all in either parallel or serial, and at the end evaluate the performance of the algorithms to select the best one(s).

And finally, developing the right solution to a real life problem is rarely just an applied mathematics problem. It requires awareness of business demands, rules and regulations, and stakeholders' concerns as well as considerable expertise. In solving a machine problem, being able to combine and balance these is crucial; those who can do this can create the most value.