Machine Learning Algorithms Pros and Cons

Updated: 2020-06-29

TL;DR

Start with Logistic Regression, then try Tree Ensembles, and/or Neural Networks.

Occam's Razor principle: use the least complicated algorithm that can address your needs and only go for something more complicated if strictly necessary.

Neural Networks(both traditional and deep neural nets) and **Gradient Boosted Decision Trees(GBDT)** are being widely used in industry.

Pros and Cons

Here discusses the most popular algorithms. For a full list of machine learning algorithms, check out the cheatsheet.

Naive Bayes

- super simple(just doing some counts) yet performing well in practice.
- compute the multiplication of independent distributions
- require less training data
- no distribution requirements
- converge quicker than discriminative models(e.g. logistic regression) under conditional independence assumption
- good for few categories variables
- suffer multicollinearity

Logistic Regression

Logistic regression is probably the most widely used

Learn More

- easy to interpret. **the output can be interpreted as a probability: you can use it for ranking instead of classification.**
- good for cases where features are expected to be roughly linear, and the problem to be linearly separable.
- can easily "feature engineering" most non-linear features into linear ones.
- robust to noise
- can use l2 or l1 regularization to avoid overfitting(and for feature selection)
- efficient, and can be distributed(ADMM)
- no distribution requirement
- compute the logistic distribution
- cannot handle categorical(binary) variables well

- compute Confidence Interval
- suffer multicollinearity
- no need to worry about features being correlated, like in Naive Bayes.
- easily update the model to take in new data (using an online gradient descent method)
- use it if you want a probabilistic framework (e.g., to easily adjust classification thresholds, to say when you're unsure, or to get confidence intervals)
- use it if you expect to receive more training data in the future and want to quickly be incorporate into the model.

Lasso(L1)

- no distribution requirement
- compute L1 loss
- variable selection
- suffer multicollinearity

Ridge(L2)

- no distribution requirement
- compute L2 loss
- no variable selection
- not suffer multicollinearity

When NOT to use

- if the variables are normally distributed and the categorical variables all have 5+ categories: use Linear discriminant analysis
- if the correlations are mostly nonlinear: use SVM
- if sparsity and multicollinearity are a concern: Adaptive Lasso with Ridge(weights) + Lasso

Linear discriminant analysis

LDA: Linear discriminant analysis, not latent Dirichlet allocation

- require normal distribution
- not good for few categories variables
- compute the addition of Multivariate distribution
- compute Confidence Interval
- suffer multicollinearity

Support Vector Machines

- Support Vector Machines (SVMs) use a different loss function (Hinge) from LR.
- they are also interpreted differently (maximum-margin).
- SVM with a linear kernel is similar to a Logistic Regression in practice
- if the problem is not linearly separable, use an SVM with a non linear kernel (e.g. RBF). (Logistic Regression can also be used with a different kernel)
- good in a high-dimensional space (e.g. text classification).

- high accuracy
- good theoretical guarantees regarding overfitting
- no distribution requirement
- compute hinge loss
- flexible selection of kernels for nonlinear correlation
- not suffer multicollinearity
- hard to interpret

Cons:

- can be inefficient to train, memory-intensive and annoying to run and tune
- not for problems with many training examples.
- not for most "industry scale" applications (anything beyond a toy or lab problem)

Decision Tree

- Easy to interpret and explain
- Non-parametric, no need to worry about outliers or whether the data is linearly separable.
- no distribution requirement
- heuristic
- good for few categories variables
- not suffer multicollinearity (by choosing one of them)
- can easily overfit,
- tree ensembles

    - e.g. Random Forests and Gradient Boosted Trees, using bagging or boosting
    - generally outperform single decision tree.
    - handle very well high dimensional spaces as well as large number of training examples.

Random Forests

- train each tree independently, using a random sample of the data, so the trained model is more robust than a single decision tree, and less likely to overfit
- 2 parameters: number of trees and number of features to be selected at each node.
- good for parallel or distributed computing.
- lower classification error and better f-scores than decision trees.
- perform as well as or better than SVMs, but far easier for humans to understand.
- good with uneven data sets with missing variables.
- calculates feature importance
- train faster than SVMs

Gradient Boosted Decision Trees

- build trees one at a time, each new tree corrects some errors made by the previous trees, the model becomes even more expressive.

- 3 parameters - number of trees, depth of trees, and learning rate; trees are generally shallow.
- usually perform better than Random Forests, but harder to get right. The hyper-parameters are harder to tune and more prone to overfitting. RFs can almost work "out of the box".
- training takes longer since trees are built sequentially

Neural Network

- good to model the non-linear data with large number of input features
- widely used in industry
- many open source implementations
- only for numerical inputs, vectors with constant number of values, and datasets with non-missing data.
- "black box-y", the classification boundaries are hard to understand intuitively("like trying interrogate the human unconscious for the reasons behind our conscious actions.")
- computationally expensive.
- the trained model depends crucially on initial parameters
- difficult to troubleshoot when they don't work as expect
- not sure if they will generalize well to data not in training set
- multi-layer neural networks are usually hard to train, and require tuning lots of parameters
- not probabilistic, unlike their more statistical or Bayesian counterparts. The continuous number output (e.g. a score) can be difficult to translate that into a

continuous number output (e.g. a score) can be difficult to translate that into a probability.

Deep Learning

- not a general-purpose technique for classification.
- good in image classification, video, audio, text.

Summary

Factors to consider

- number of training examples, (how large is your training set?)
  - if small: high bias/low variance classifiers (e.g., Naive Bayes), less likely to overfit
  - if large: low bias/high variance classifiers (e.g., kNN or logistic regression)
- dimensionality of the feature space
- is the problem linearly separable?
- are features independent?
- are features expected to linearly dependent with the target variable?
- is overfitting expected to be a problem?
- system requirement: speed, performance, memory usage
- Does it require variables to be normally distributed?
- Does it suffer multicollinearity issue?
- Dose it do as well with categorical variables as continuous variables?

- Does it calculate CI without CV?
- Does it conduct variables selection without stepwise?
- Does it apply to sparse data?

Start with something simple like Logistic Regression to set a baseline and only make it more complicated if you need to. At that point, tree ensembles, and in particular Random Forests since they are easy to tune, might be the right way to go. If you feel there is still room for improvement, try GBDT or get even fancier and go for Deep Learning.