Machine learning general rules and concepts

Justin

# Variance and covariance

One conclusion out of the 2nd equations is when and are unrelated, .

# Why i.i.d

The short answer is in order to learn useful model one needs to assume the training samples to be representative of the entire data. Check this [link](https://stats.stackexchange.com/questions/213464/on-the-importance-of-the-i-i-d-assumption-in-statistical-learning).

# High bias vs. high variance

High bias: the selected model is too simple; high variance: the selected model is too complex

# Bias-variance decomposition

Assume the true relationship between the independent variable and the dependent variable is given by: , where is normally distributed noise with a mean of 0 and a standard deviation of . The fit function is: . The squared error for any data point is the following:

Note the first term describes the variability of the model if running over many data sets, and hence is the variance term.

# Principles of model selection

Based on speed, performance, or interpretability

# How to validate the model? How do we know the problem is solvable by machine learning models?

The rule of thumb is k-folds cross-validation. There are many reasons why selected models may fail:

* The selected model is too simple or too complicated
* The given data is skewed
* Too much noise
* Too many outliers
* Selected features are not informative enough – add new features
* Not enough data
* i.i.d condition is not satisfied

# Learning curve

The learning curve is a convenient tool for diagnose bias and variance in model fitting.

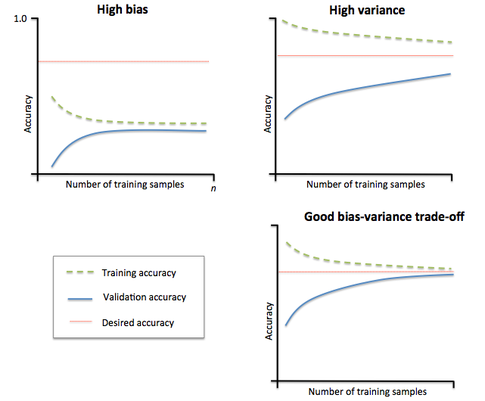


Figure 1. Learning curve. A good model should have both high training and validation scores

# Hyper-parameter tuning

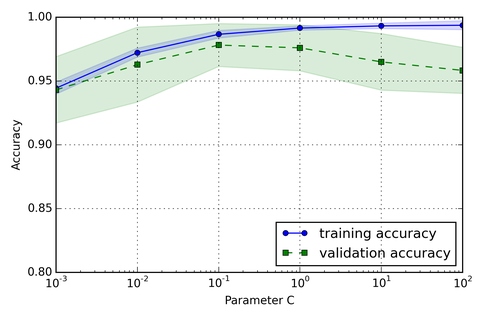


Figure 2. Validation curve -- dependence of training and validation score on a certain hyper-parameter

# Two fundamental milestones of classification algorithms

## Linear discriminant analysis (LDA)

LDA is essentially a linear transformation technique, which is mainly used for dimensionality reduction. The objective is to find the k-dimensional feature subspace that – linearly – separates the samples from different classes the best. LDA has a close-form solution

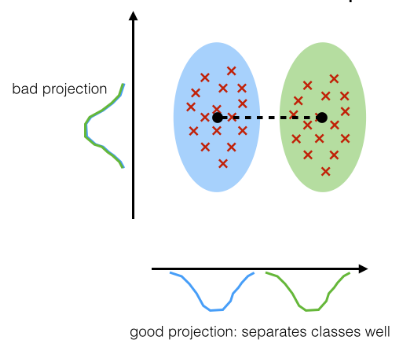


Figure 3. LDA. The horizontal axis is better than the vertical axis in separating the two groups

## Perceptron algorithm

This category essentially includes logistic regression, SVM, neuron networks. Different from LDA, this is an **incremental learner** – for each training sample, it compares the predicted class label to the actual label and modifies the model weights accordingly.

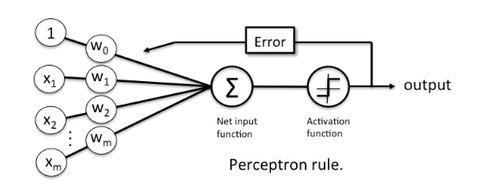


Figure 4. Perceptron algorithm has an error feedback loop

# A few commonly used activation functions

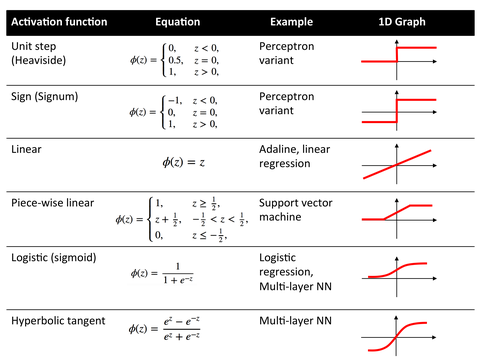


Figure 5. Commonly used activation functions

Resources:  
<https://github.com/rasbt/python-machine-learning-book/blob/master/faq/classifier-history.md>

# Parametric vs. non-parametric algorithms

Non-parametric models (can) become more and more complex with an increasing amount of data. Hence, a parametric model has a finite number of parameters, while a non-parametric model has (potentially) infinite number of parameters.

Parametric models: linear regression, logistic regression, linear SVM, etc.

Non-parametric models: K-nearest neighbor, decision trees, RBF kernel SVMs

# Gradient descent (GD, or batch GD), stochastic gradient descent (SGD), and min-batch gradient descent (MB-GD)

In GD, updates are made after computing the cost gradient based on the complete training set.

In SGD, updates are made after computing each training example. SGD is sometimes also referred to as iterative or on-line GD. Due to its stochastic nature, the path towards the global cost minimum is not "direct" as in GD, but may go "zig-zag" if visualized in a 2D space. SGD almost surely converges to the global cost minimum if the cost function is convex (or pseudo-convex)

MB-GD is a compromise between batch GD and SGD. MB-GD converges in fewer iterations than GD because the weights are updated more frequently; however, MB-GD enables vectorized operation, which typically results in a computational performance gain over SGD.

Some further notes:

* For SGD, and MB-GD, shuffling the data is important to avoid pre-existing order of the examples
* For MB-GD, the number of samples within each mini-batch is usually a power of 2 due to computer hardware requirement

|  |  |  |  |
| --- | --- | --- | --- |
|  | SGD | | MB-GD |
| Pros | 1. Can converges faster due to more frequent updates 2. Can hop out of local minima, very useful for models that have lots of local minima 3. Less stress on RAM 4. Online algorithm | 1. Can take advantage of vectorized computation 2. Somewhere between SGD and MB-GD | |
| Cons | 1. Too much fluctuation | |  |

Table 1. Pros and cons of SGD and MB-GD compared to batch GD



Figure 6. Visualization of batch GD, SGD, and MB-GD

Resources:  
<https://stats.stackexchange.com/questions/49528/batch-gradient-descent-versus-stochastic-gradient-descent>  
<https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>

# Ticks to improve the GD-based learning

There are a few tricks to improve the GD-based learning:

* Use an adaptive learning rate that shrinks over time
* Adding a factor of previous gradient to the weight update for faster updates

# Importance of scaling the features

Scaling the data makes the GD algorithm converge faster.

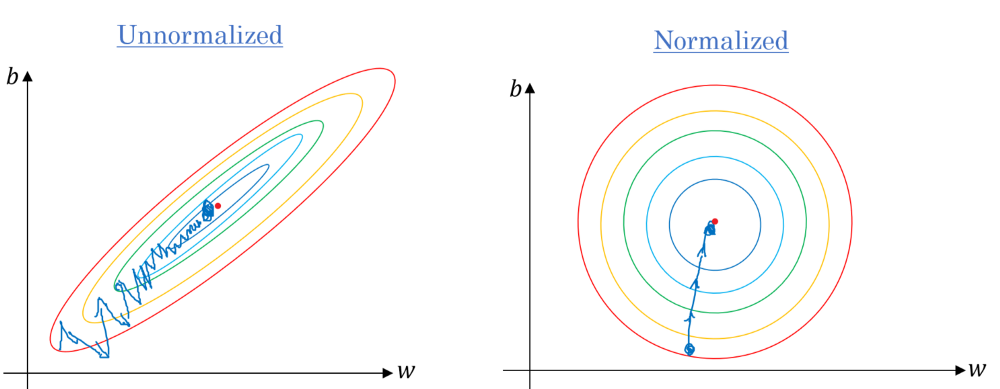


Figure 7. Advantage of feature normalization in GD. Normalizing the data puts the weights in roughly the same scale, and hence makes the GD algorithm converges faster. In the left figure, it’s very easy for the algorithm to overshoot in the squashed direction, and hence the minimization path is zig-zag shaped

# Connection between Pearson R and simple linear regression

Where is the slope of simple linear regression, is the Pearson correlation, and are the standard deviations.

Resources:  
<https://github.com/rasbt/python-machine-learning-book/blob/master/faq/pearson-r-vs-linear-regr.md>

# Bagging vs. boosting

Bagging and random forests are "bagging" algorithms that aim to reduce the complexity of models that over-fit the training data. In contrast, boosting (Ada-boost) is an approach to increase the complexity of models that suffer from high bias.

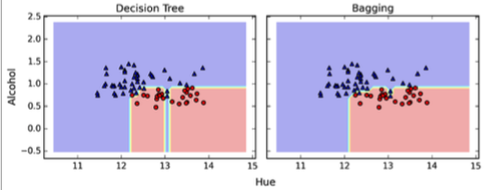


Figure 8. Bagging is capable of correcting the high variance of a single decision tree

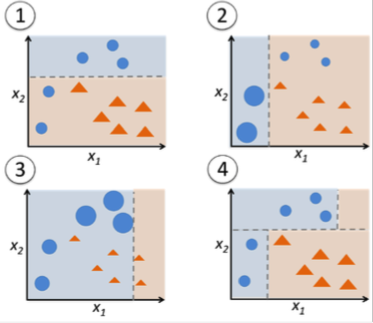


Figure 9. Boosting is able to incrementally learn from the mistakes and reduce the high bias. In other words, boosting gradually increases the complexity of the model

# The curse of dimensionality

## Counter-intuition in high dimensionality

### Most of the mass of a multivariate Gaussian distribution is not near the mean, but in an increasingly distant “shell” around it

### Most of the volume of a high-dimensional orange is in the skin, not the pulp

### If a constant number of examples is distributed uniformly in a high-dimensional hypercube, beyond some dimensionality most examples are closer to a face of the hypercube than to their nearest neighbor

The fact that all points are closer to a face of the hypercube than to each other is very bad news for machine learning algorithms. This means any prediction run in extrapolating mode instead of interpolating mode.

### If we approximate a hypersphere by inscribing it in a hypercube, in high dimensions almost all the volume of the hypercube is outside the hypersphere

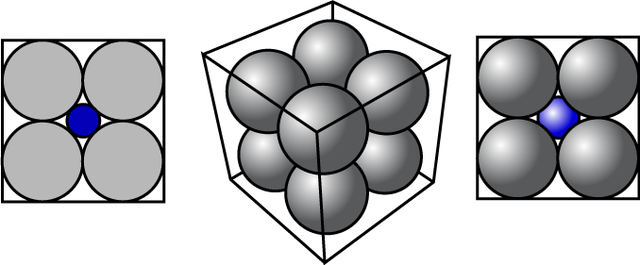


Figure 10. Another example of the curse of the dimensionality – put a blue hypersphere in the center gap of gray hyperspheres, which have a radius of ¼, and are placed tightly inside a hypercube of sides of length 1. Initially, the volume of the blue sphere is small but grows quickly as the number of dimensions increases. With 16 dimensions, the radius of the blue center sphere is exactly ½ and touches the sides of the hypercube. With > 16 dimensions, its radius is even larger and crashes through the sides of the cube, which is very counter-intuitive. See this [link](https://shapeofdata.wordpress.com/2013/04/02/the-curse-of-dimensionality/) for details

## The consequences of the high dimensionality in machine learning

### Over-fit (one extreme case if the number of features >= the number of samples)

For instance, the goal of the linear regression is the find one hyperplane that best fits the data. In 2D, a line can perfectly fit 2 distant points; similarly in 3D, a plane can perfectly fit 3 distant points. Hence, with the number of features equal to the number of samples, one can always find a hyperplane that perfectly fit all data points. With number of features exceeding the number of samples, more than one solution is possible.

### Sparsity (bad of clustering)

### Distance metric starts to fail (related to sparsity)

Both the sparsity and the failure of the distance metric can be understood in the following. Suppose we have features all uniformly distributed in the range of (-1, 1). As increases, more and more samples are found around the corners of the -D hypercube (due to the fact that the volume of the inscribed hypersphere diminishes). In addition, since there are corners, if the number of samples , it’s likely that each sample occupies one corner. Hence, each pair of samples has roughly the same distance, making the distance based algorithms (k-means, KNN) fail to work.

## How to overcome the curse of dimensionality

### Non-uniformity blessing

First of all, the curse of dimensionality occurs due to uniform distribution. The real data is neither uniformly distributed nor random, e.g., the handwriting in a figure is more distributed in the center. This is known as “non-uniformity blessing”.

### Dimensionality reduction, e.g., LDA, PCA

### Change the algorithm

Some algorithms (e.g., neuron networks) are more resistant to the curse of dimensionality than others. The idea behind is the Manifold Hypothesis. At a high level the Manifold Hypothesis suggests that the high dimensional data actually sits on a lower dimensional manifold embedded in higher dimensional space.

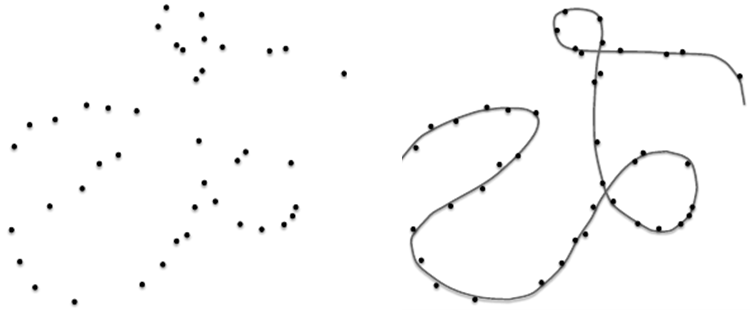


Figure 11. Manifold Hypothesis -- the high dimensional data sits on a lower dimensional manifold

## Relation to other concepts

### Random forest only selects randomly a subset of features to split each node because this reduces the chance of over-fit (or the curse of dimensionality)

### For larger degree of freedoms, the bulk part of the PDF of the chi-squared distribution shifts outward

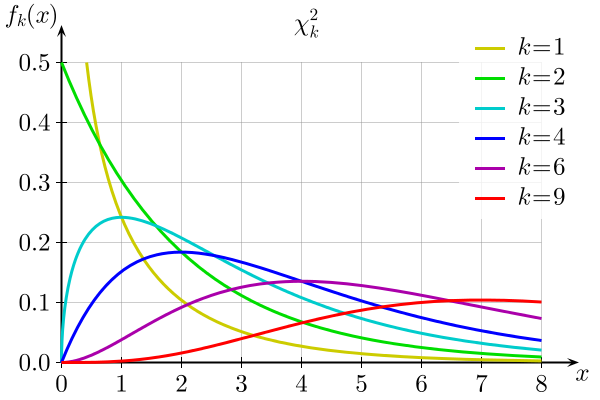


Figure . -distribtion. For larger degree of freedoms, the bulk part of the PDF of the chi-squared distribution shifts outward

# Chi-squared () distribution

If , …, are independent, standard normal random variables, then the sum of their squares,

is distributed according to the chi-squared distribution with k degrees of freedom.

## Derivation of

Let , the PDF of is given by:

Note is standard normally distributed, , hence:

The PDF of is then the derivative of against :

# Over-fit issue

Over-fit is when the model fits well on the training set with good score but fails to generalize to unseen data. In other words, the training score is high but the validation score is low.

## How to identify over-fit

Plotting the learning curve or the validation curve indicates that the training score is higher than the validation score.

## Remedies against over-fit

### Add regularization penalties

### Adopt ensemble techniques that “average” the models, e.g., bagging (bootstrap aggregation) used in the random forest

Ensembling is probably the most convenient way to build robust predictive models on somewhat small-sized datasets. As in real life, consulting a bunch of "experts" is usually not a bad idea before making a decision.

### Reduce the dimensionality of the feature space – related to the curse of the dimensionality

### Add more data

In case of adding more data, it is useful to plot the learning curve.

### Start with simpler model

# Statistical errors

## Type I error (false positive, false alarm)

Type I error occurs when **rejecting the null hypothesis when it’s true**. The type I error rate or significant level is usually set to 5%, implying that it is acceptable to have a 5% probability of incorrectly rejecting the null hypothesis.

## Type II error (miss)

Type II error occurs when **failing to reject the null hypothesis when it’s false**. The type II error rate is denoted by , and is related to the power of a test (which is ).

# Receiver operating characteristic (ROC or TPR vs. FPR)

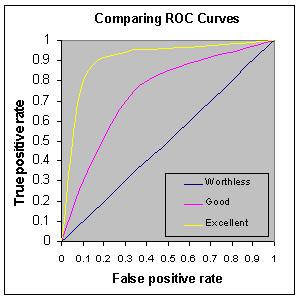


Figure . ROC curve. Note when TPR == FPR, the prediction is worthless random guess!

## ROC vs. precision-recall

ROC is not a good (or more precisely, not a convenient) metric when dealing with the class imbalance issue. Consider the outcome of the following two classification algorithms:

Method 1:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Actuality** | |
|  |  | **P** | **N** |
| **Prediction** | **P** | TP = 90 | FP = 10 |
| **N** | FN = 10 | TN = 1e9 - 110 |

Method 2:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Actuality** | |
|  |  | **P** | **N** |
| **Prediction** | **P** | TP = 90 | FP = 1910 |
| **N** | FN = 10 | TN = 1e9 - 2010 |

Clearly method 1 is superior to method 2 in that it has much lower FP. However, when using ROC metric, both methods have the same TPR. The FPR are both very close to 0 due to the dominant TN, even though method 1 shows 2 order of magnitudes lower FPR – this is where the inconvenience arises.

In such cases, the precision-recall (or F-score) is preferred. This is because in order to have a high F-score, one must have both high precision (few false alarms) and high recall (few misses).

## Multiclass classification

The simplest solution to extend binary ROC to a multiclass setting is to involve one vs all technique.

# Performance metrics and confusion matrix

A high-level note: what F1-score metric is trying to achieve is to find an equal balance between precision and recall, which is extremely useful when dealing with the class imbalance issue. A side note – there are several ways to calculate the F1-score arising from k-fold cross-validation, check this [link](https://github.com/rasbt/python-machine-learning-book/blob/master/faq/computing-the-f1-score.md).

## Binary class

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Actuality** | |
|  |  | **P** | **N** |
| **Prediction** | **P** | TP (hit) | FP (false alarm, type I error) |
| **N** | FN (miss, type II error) | TN (correct rejection) |

Total is the sum of TP, FP, FN and FN

Sensitivity, hit rate, recall, TPR (no miss → high recall):

Precision (no false alarm → high precision):

Accuracy:

FPR:

F1 score:

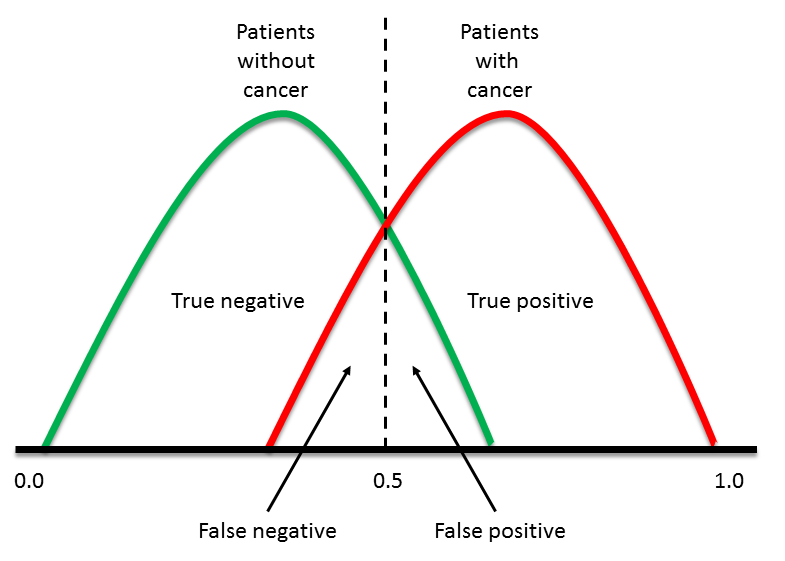


Figure . One way to visualize the confusion matrix

## Multiclass

The accuracy metric will always be:

In the context of one-vs-all classifiers, there are two major approaches to calculate the recall and precision metrics: micro- and macro-average. Check this [link](https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin) for details.

### Macro-average

This approach treats each class equally by calculating the recall or precision score for each class, and averaging them over the number of classes to find the mean recall or precision score.

First calculate the recall and precision for an individual class :

The overall recall or precision is simply the mean of all individual recalls or precisions for each class. Note all classes are assumed to be equal in this method.

### Micro-average

This approach aggregates the contributions of all classes to compute the average metric. In case of imbalanced classes, this approach is preferred.

# Model selection

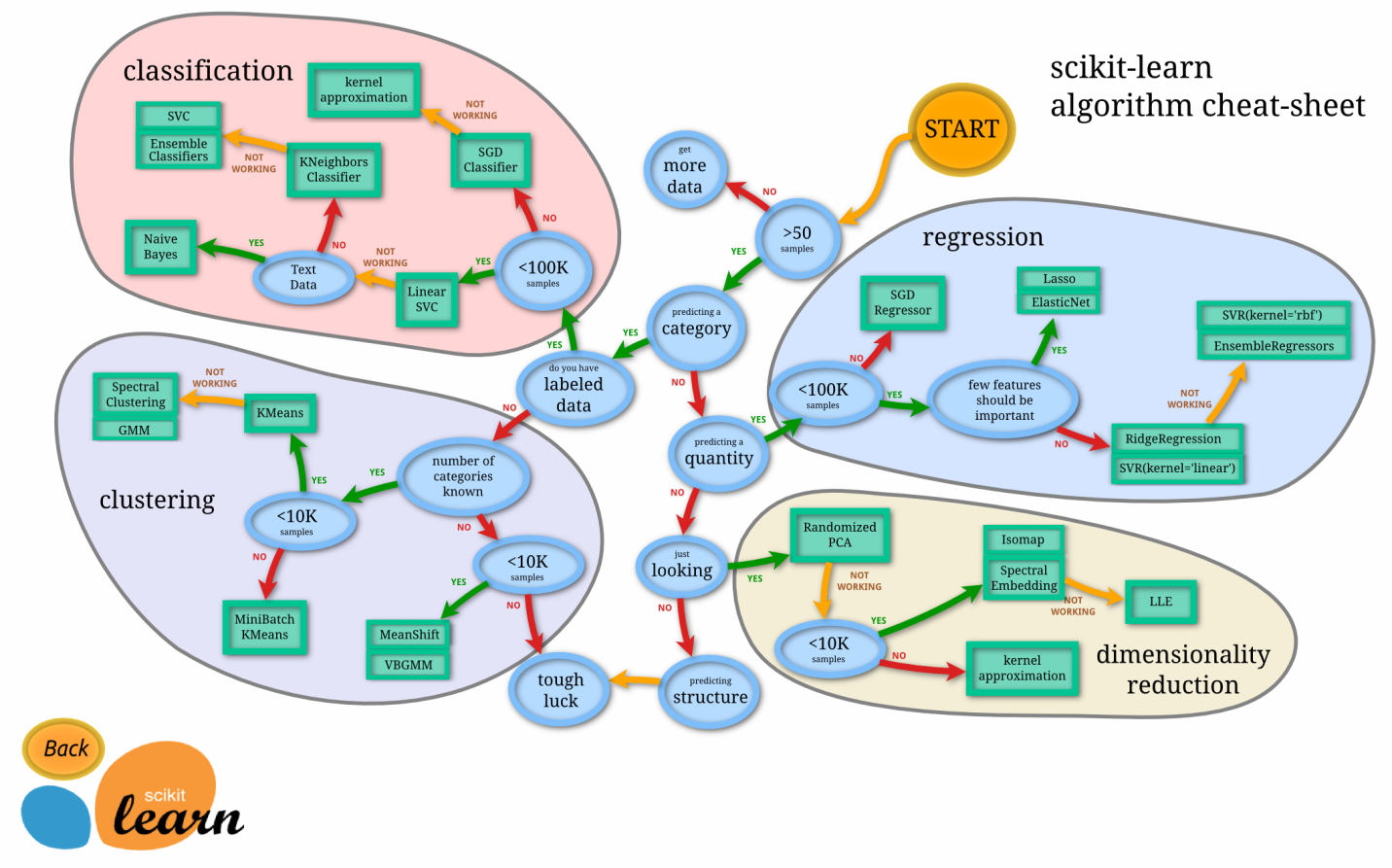
[](http://scikit-learn.org/dev/tutorial/machine_learning_map/index.html)

Figure . Machine learning algorithms cheat sheet

# Model evaluation and cross-validation

One word prior to the discussion – in order to have unbiased evaluation of the model performance, one should use a clean test set that is unseen during the training, hyper-parameter tuning, or model selection steps.

There are three scenarios:

## Split the dataset to training/test sets and train a simple model

Split the dataset to a separate training and test set. Train the model on the training set and evaluate its performance on the test set.

## Split the dataset to training/cross-validation/test sets and tune the hyper-parameters

Spit the dataset to two parts – one reserved for training/cross-validation and one for model evaluation (test set). One commonly used technique for hyper-parameter tuning is to perform k-fold cross-validation. The model with the best hyper-parameter is then evaluated on the test set.

## Nested cross-validation (nested CV)

One key drawback of scenario b) is the need of a separate test set, which is not used in the training procedure at any level at all. To address this issue, the nested cross-validation is proposed – the inner CV is used for hyper-parameter tuning/model selection while the outer CV is for model performance evaluation. For instance, assume the outer CV is a k-fold CV, then at any iteration step of the outer CV, its current training set is further split to multiple folds in the inner CV for hyper-parameter tuning or model selection. Once this is done, we have the best model so far that corresponds to the current iteration step of the outer CV. Since the outer CV will run k times, we essentially have k different best models given by the inner CV. If things are stable, then the k selected hyper-parameters or models should all be the same.

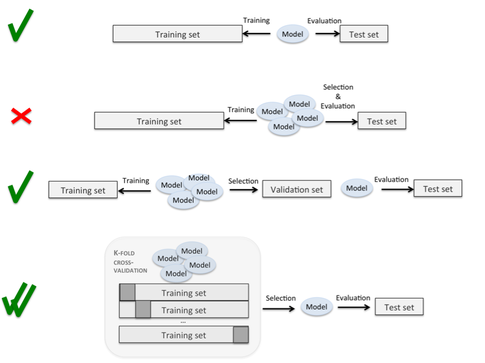


Figure . Scenarios of model evaluation. Row 1: scenario 1. The dataset is simply spit to the training and test sets. The model learns from the training set and is evaluated in the test set. Row 2: a wrong scenario. The test set cannot be used for model selection and performance evaluation simultaneously. Row 3: scenario 2. The dataset is split to training/cross-validation/test sets. The hyper-parameter tuning/model selection is performed via k-fold cross-validation. The final best model is tested on the unseen test set. Row 4: scenario 3. Nested cross-validation. The inner CV is used for hyper-parameter tuning/model selection while the outer CV is for model performance evaluation. The advantage over scenario 2 is all data have been used in the training

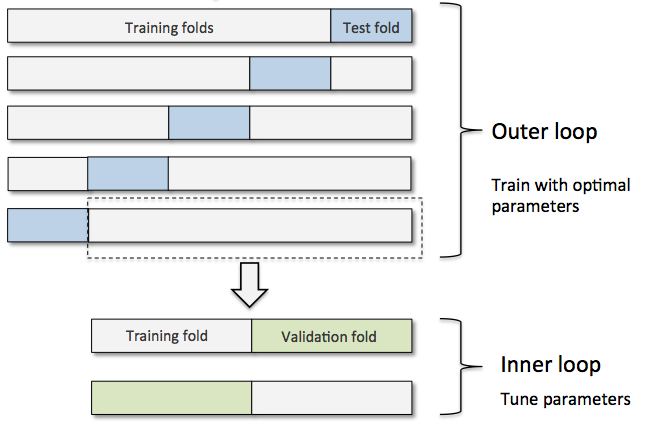


Figure . Visualization of the nested cross-validation. The inner CV loop is used for hyper-parameter tuning/model selection

One note on cross-validation and the optimum number of folds – It’s not always a good idea to use a large number of folds, because of:

* The increase in the run time
* The increase in the variance of the estimate – the overlap between training sets increases with an increasing number of folds – note that the test sets never overlap though

However, for small data sets, it helps with a large number of folds as it uses more training samples.

Resources:  
<https://www.elderresearch.com/blog/nested-cross-validation>   
<https://github.com/rasbt/python-machine-learning-book/blob/master/faq/evaluate-a-model.md>  
<https://chrisalbon.com/machine_learning/model_evaluation/nested_cross_validation/>

# Online machine learning algorithms

## Lazy learners, e.g., K-nearest neighbors

Need to keep training data around. No learning process is necessary but predictions are expensive.

## Stochastic GD

# On imbalanced datasets

## Performance metric selection level: choose F1-score as the performance metric

## Model selection/building level: Increase the weight of the small population class

Machine learning algorithms that support class weight: random forest (via over-sampling of the small population class); SVM; logistic regression

Add penalties

## Model evaluation level: use stratified k-fold instead of the normal k-fold

# On missing data

# Cross product

# Connection between the ordinary least square and the Gaussian maximum likelihood

For linear regression (here is a vector that includes all samples, has a shape of n\_samples × n\_features, is the weight vector including the bias term, is the normally distributed noise with a mean of 0 and a standard deviation of ), the close-form solution is given by:

## Derivation of the closed-form solution via the ordinary least square (OLS)

The idea of OLS is to minimize the sum of square , hence

Both and are scalars. For a scalar, . Hence, , and the previous equation can be reduced to:

The solution for minimizing is:

## Connection to the Gaussian maximum likelihood

For each sample, the error between the true target value and the predicted target value is . Assume is i.i.d, and is normally distributed with a mean of 0 and a standard deviation of . Our objective is to find out a spectral distribution function that maximizes the likelihood, given the samples. The likelihood is given by:

In reality, it’s more convenient to deal with the log-likelihood:

Since the goal is to maximize the log-likelihood, it is equivalent to minimization of the sum of square:

Check this [link](https://zhuanlan.zhihu.com/p/33899560) for details.