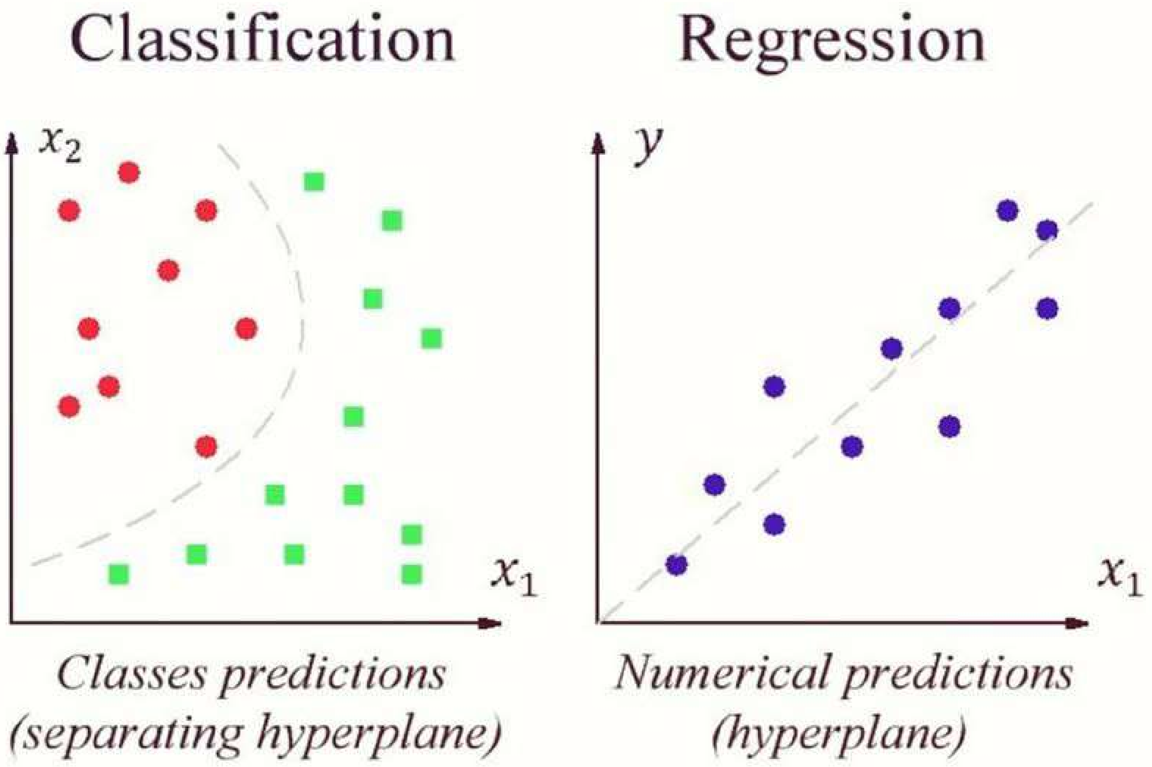
Supervised Learning Algorithms

(ALL-IN-ONE CHEATSHEET)

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used later for mapping new examples.

The most popular supervised learning tasks are: *Regression* and *Classification*.

- The result of solving the *regression* task is a model that can make *numerical predictions*. For example:
  - Real estate value prediction
  - Predicting your company's revenue next year
- The result of solving the *classification* task is a model that can make *classes predictions*. For example:
  - Spam detection
  - Classifying news articles
- The line between these tasks is sometimes fuzzy (predicting the probability of cancer based on blood tests)


Classification
Regression
x_2
x_1
y
x_1
Classes predictions (separating hyperplane)
Numerical predictions (hyperplane)

Like This? Repost to your Network and Follow @data_science_learn

Linear Regression

In the simplest case, the regression task is to draw a line through the data points so that an error between this line (predictions) and real values is minimal. In general, this is the problem of *minimizing the loss function*, so *the optimization problem*. Usually, the loss function is the *MSE - mean square error* (because of *maximum likelihood estimation*), and the optimization algorithm is *gradient descent*. Anyway, any other loss function of optimization algorithm can be used.

One of the important properties of linear regression is that optimal parameters (according to *MSE*, again, because of *maximum likelihood estimation*) can be calculated with simple **Normal Equation**. But this method does not scale well with large number of features, so any other optimization method can be applied instead.

If the data dependences is more complex, than a straight line, we can add powers of each feature as new features (*PolynomialFeatures* class from *sklearn* can be used) and then train a Linear Regression model. This technique is called **Polynomial Regression**. Process of creating new features (e.g. x^n, or log(x), e^x etc.) is called *feature engineering* and can significantly increase linear model performance.

Other popular version of this algorithm is **Bayesian Linear Regression**, that predicts not only values, but also it's probabilities, by building a *confidence interval*. This is possible thanks to *Bayes' theorem*.

One of the most efficient way to avoid overfitting and outliers influence with regression is **regularization**. *Regularization term* is added to loss function so regression coefficients have to be as little as possible.

- **LASSO regression** - implements L1 regularization, + |coeff|.
- **Ridge regression** - implements L2 regularization, + coeff^2. Also known as *Tikhonov regularization*.
- **Elastic Net regression** - implements both L1 and L2 regularization.

Regularized regression can also be used like *feature selection* tool. Thanks to some properties, LASSO regression, for example, can delete insignificant features (set their coefficients to zero).

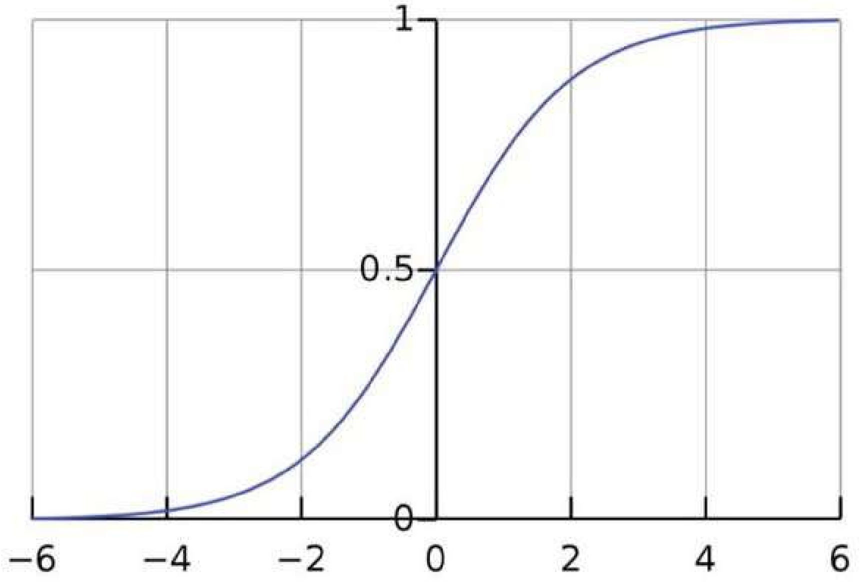
As mentioned earlier, *linear regression solve only regression task*.

Main hyperparameters:

- *feature engineering* and *feature selection*
- regularization type and parameter
- solver - optimization algorithm

Logistic Regression

Like Linear Regression model, Logistic Regression (is also known as **logit regression**) computes a weighted sum of the input features (plus bias) but instead of outputting this result directly, it outputs the *logistic* of the result. The *logistic* is a *sigmoid function*, that outputs a number between 0 and 1, so Logistic Regression is **soft binary classifier** that estimates the probability that instance belongs to the positive class. Depends of some threshold different values of accuracy/recall can be obtained. The same types of regularization as in Linear Regression can be used.



Sigmoid Function. Public Domain

Very similar **probit regression** uses a little different function - probit function instead of sigmoid.

The Logistic Regression model can be generalized to support multiple classes directly, without training multiple classifiers. This is called **Softmax Regression** (or *Multinomial Logistic Regression*). This model computes a score for each class and then estimates the probability of each class by applying *softmax function* (also called *normalized exponential*).

As mentioned earlier, *logistic regression solve only classification task*.

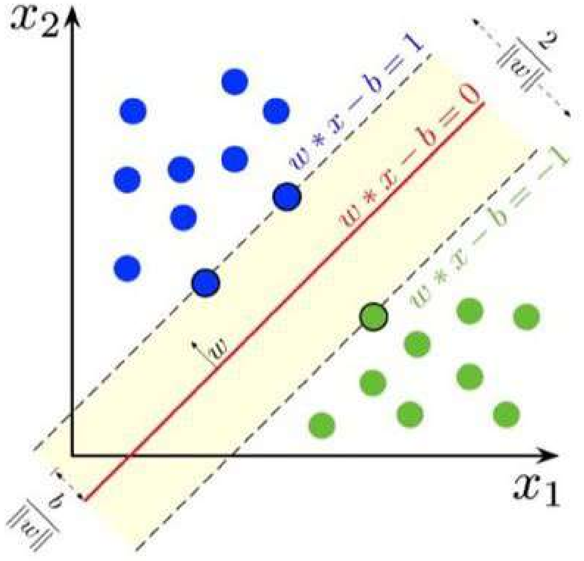
Is based on Linear Regression, so inherits all the hyperparameters, pros and cons of this algorithm. What can be noted separately - *high interpretation* level of this algorithm, so it is usually widely used in *credit scoring* tasks and *medical diagnostics*.

Support Vector Machines

Support Vector Machines algorithm is based on *support vectors* concept - the extreme points (circled in black in the image).
In case of *classification task* it tries to draw a separating line between classes such that *support vectors* are located as far as possible from this line (separating hyperplane in general case):

- *Hard Margin Classification* - it is assumed that instances of the same class are on the same side of the separating hyperplane without exceptions.
- *Soft Margin Classification* - allows violation of the decision boundary, which is regulated by the regularization parameter.

In case of *regression task*, instead, it tries to draw a line to fit as many instances as possible inside the border, "on the street".



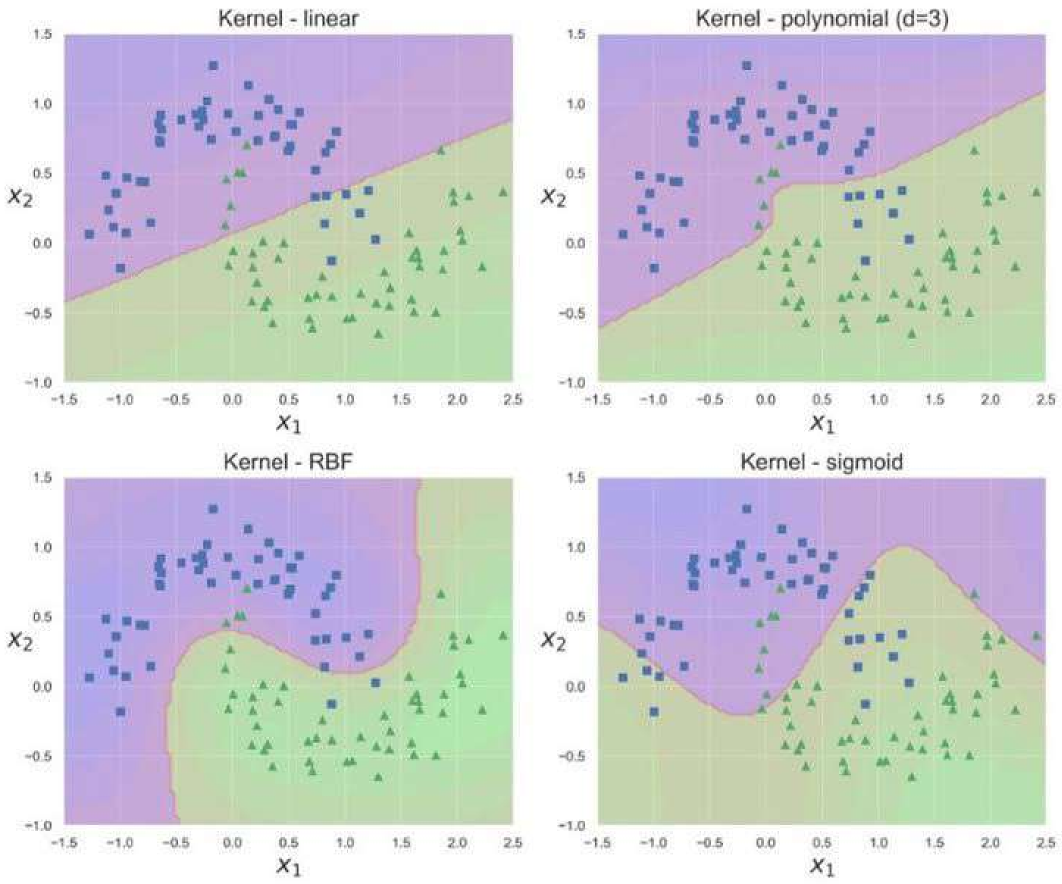
***Support Vectors.* [Public Domain]**

Since SVM requires calculating distances between points it also requires *feature scaling*.

The most important and mathematically elegant feature of SVM is that the solution of the *Dual Problem* (which is the basis of SVM) does not depend on the features directly (as vectors), but *only on their pairwise scalar products*. This allows us to replace the scalar product with a certain function *k(a, b)*, which is called the *kernel*. In fact, the kernel is a *scalar product in some other space*. This procedure allows you to build nonlinear classifiers (which are actually linear in a larger dimension space) without adding new features and is called **kernel trick**.

The use of different kernels allows this algorithm to recover very complex dependencies in both *classification* and *regression* tasks. The most popular kernels are:

- polynomial
- RBF - Gaussian Radial Basis Function
- sigmoid and others



SVM with different kernels and default parameters. Image by Author

One-class SVM also can be used for the *Anomaly Detection* problem.

Main hyperparameters:

- kernel type
- regularization parameter C - a penalty for each misclassified data point (usually 0.1 < C < 100)
- regularization parameter gamma - controls regions separating different classes. Large gamma - too specific class regions (overfitting). (usually 0.0001 < gamma < 10)

Pros:

- One of the most powerful and flexible models
- As linear model inherits the pros of linear regression

Cons:

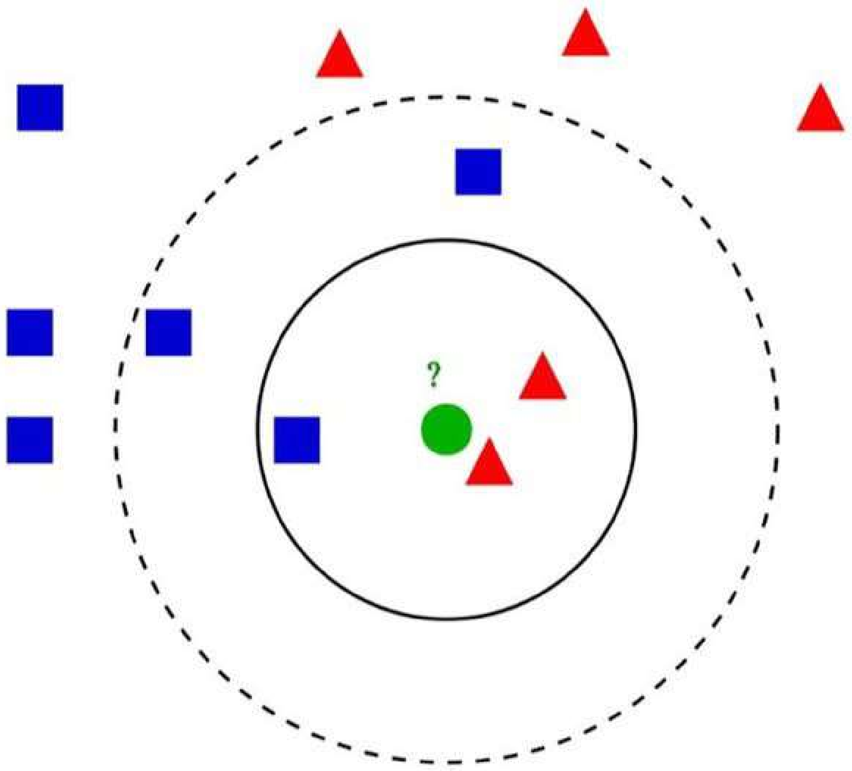
- Requires data preprocessing
- It scales well with number of features, but not samples, so works well only on small and medium-sized datasets

k-Nearest Neighbors

The nearest neighbor algorithm, as a representative of *metric methods*, makes two hypotheses about the data distribution:

- Continuity hypothesis for *regression* - close objects correspond to close answers, and
- Compactness hypothesis for *classification* - close objects correspond to the same class.

For a new object we have to find k nearest neighbors. Definition of *nearest* depends on the distance metric that we want to use (Manhattan, Euclidean etc.).



k-Nearest Neighbors algorithm. The result may differ depending on k.
Public Domain

The most important hyperparameter is number of neighbors - k. A good initial approximation of k is to set *k to square root of data points number*, but, of course, k can be found with *Cross Validation*. *Classification* then is computed from a simple majority vote of the nearest neighbors of each point, *regression* - from a mean value of the nearest neighbors of each point.

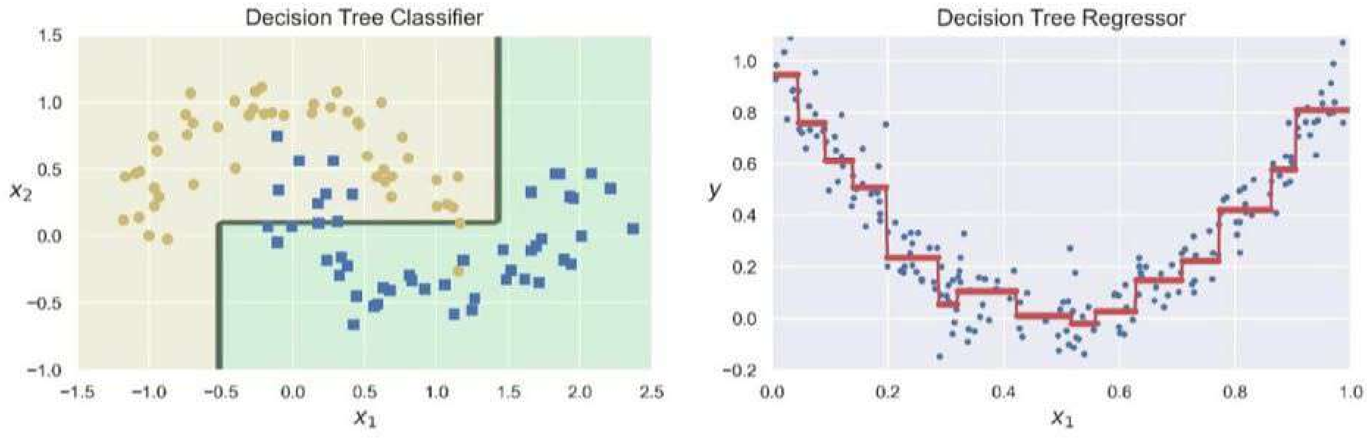
Main hyperparameters:

- k - number of neighbors
- distance metric

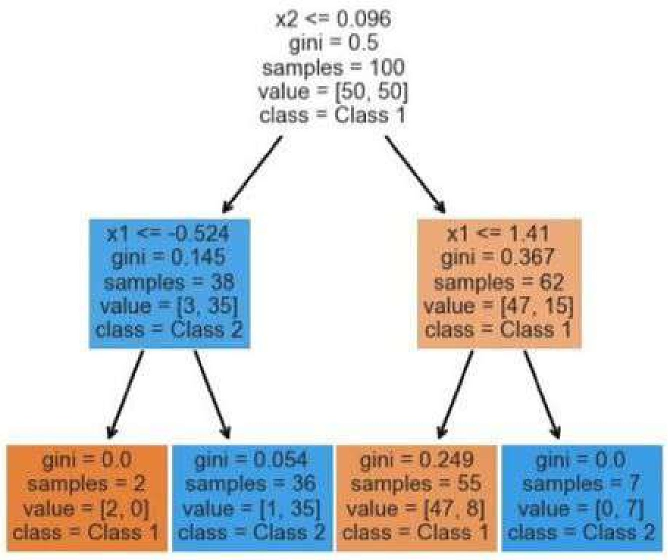
Decision Trees

At each step, the train set is divided into two (or more) parts, depending on a particular choice. Usually these algorithms are *greedy*, that means, that they are looking for a *local* optimal solution at a specific step. The popular algorithms for building trees are:

- **ID3** (one of the oldest algorithm, *Iterative Dichotomiser 3* was invented by *Ross Quinlan*),
- **C4.5, C5.0** (an extensions of ID3 algorithm, they were developed by the same person and consists in *pruning* the tree after using ID3),
- **CART** (*Classification And Regression Tree* is optimized for both classification (*Gini Inpurity* as measure) and regression (*MSE* as measure) trees and is implemented in scikit-learn).



Decision Tree Classifier and Regressor. Image by Author



Different measures for calculating *information gain* can be used. Then decision tree algorithm use information gain to split a particular node:

- *Entropy* - measure of disorder.
- *Gini Impurity*.

The so-called **decision tree pruning** shows itself better than simply limiting the length of the tree. This is the procedure when we build a tree of full depth, after that we remove insignificant nodes of the tree. However, this process is more resource-intensive.

Main hyperparameters:

- maximum depth of the tree - the less the less overfitting, usually 10-20
- minimum number of objects in a leaf - the greater the less overfitting, usually 20+

Pros:

- Simple interpretation
- Simple realization
- Computational simplicity
- Does not require features preprocessing and can handle with missing values
- Feature importance can be calculated using information gain

Cons:

- Unstable and variable (investigation of greedy algorithm) - a small change in the input data can completely change the structure of the tree
- High sensitivity to the content of the training set and noise
- Poorly restores complex (non-linear) dependencies
- The tendency to overfitting at a large depth of the tree
- Unlike linear models, they are not extrapolated (they can only predict the value in the range from the minimum to the maximum value of train set)