10 Supervised techniques

Date	@November 19, 2024
Topic	Theory

Supervised learning techniques

LASSO regressor

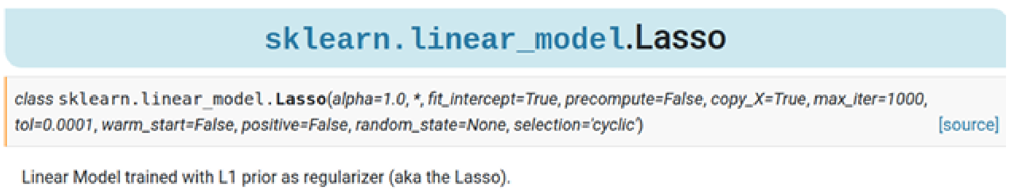
LASSO regressor (Least absolute shrinkage and selection operator) is a regressor for which we want data with numeric features and label, a model as a space of linear maps and regularized squared loss.

Linear regression requires a training set larger than the number of features (m > n) to not overfit. But sometimes m < n, so we need a regularization technique that uses a penalty term in the loss function to discourage using too many features. Thus, the regularized squared loss will be

$$L\left((\mathbf{x}, y), h^{(\mathbf{w})}\right) = \left(y - \mathbf{w}^T\mathbf{x}\right)^2 + \lambda\|\mathbf{w}\|_1.$$

Increasing the regularization parameter λ results in a weight vector **w** with increasing number of zero coefficients. λ works as a feature selection.

In python:



sklearn.linear_model.Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')
```

[source]

Linear Model trained with L1 prior as regularizer (aka the Lasso).

Support vector machines (SVM)

This another widely used techniques, and it is both a binary classifier and regressor. Here we consider data with numeric features, binary label values, a model as the space of linear maps and the regularized hinge loss.

The idea is to find a linear hyperplane (decision boundary) that will separate the data.

The hinge loss is defined as

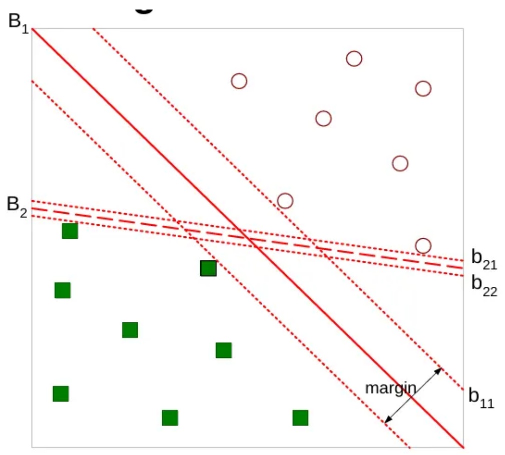
$$L\left((\mathbf{x}, y), h\right) := \max\{0, 1 - yh(\mathbf{x})\}.$$

For a linearly separable dataset, there might be infinite maps with 0 average hinge loss.

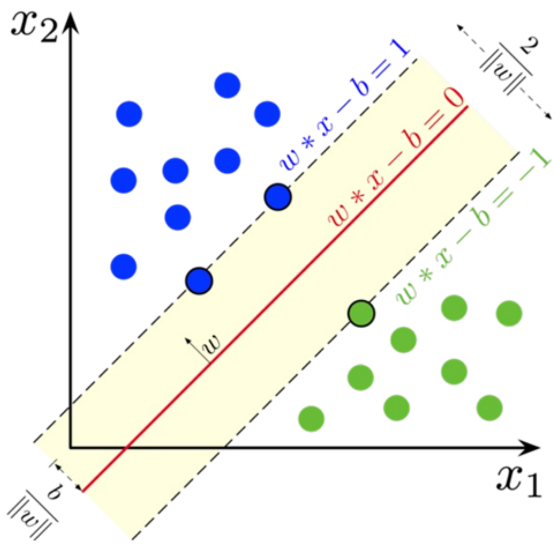
The regularized hinge loss is

$$L\left((\mathbf{x},y),h^{(\mathbf{w})}\right) := \max\{0, 1 - y \cdot h^{(\mathbf{w})}(\mathbf{x})\} + \lambda \|\mathbf{w}\|_2^2$$
$$\overset{h^{(\mathbf{w})}(\mathbf{x})=\mathbf{w}^T\mathbf{x}}{=} \max\{0, 1 - y \cdot \mathbf{w}^T\mathbf{x}\} + \lambda \|\mathbf{w}\|_2^2 .$$

Minimize the average regularized hinge loss means to maximize the margin. The aim is to find a linear hyperplane (decision boundary) that will separate the data and maximizes the margin.

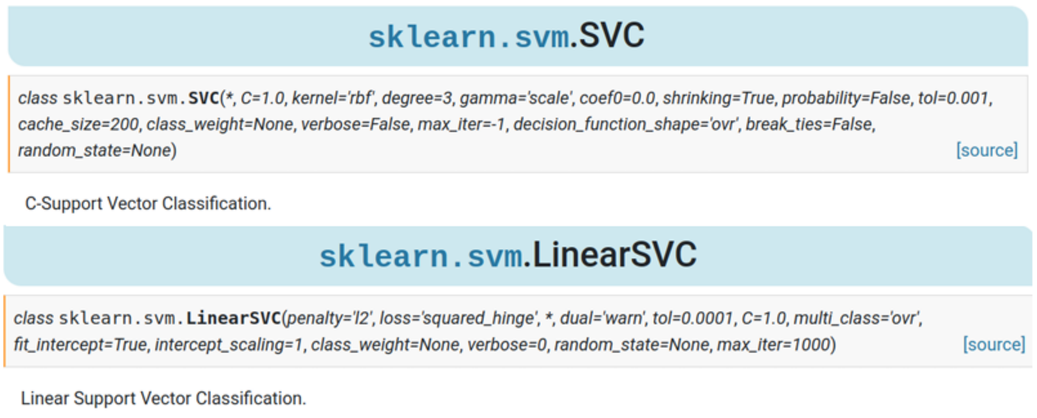


This technique is more robust than logistic regression since the loss function favors linear maps that are resilient to small perturbations in the data points. The margin represents the minimum distance of all closest points to the decision boundary. Points that are misclassified have a negative distance from this boundary.



If the decision boundary is not linear we can transform data into higher dimensional space, for which a linear boundary does exist.

In python:



sklearn.svm.SVC

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001,
cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False,
random_state=None)
```

[source]

C-Support Vector Classification.

sklearn.svm.LinearSVC

```
class sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *, dual='warn', tol=0.0001, C=1.0, multi_class='ovr',
fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)
```

[source]

Linear Support Vector Classification.

Naive Bayes Classifier

Naive Bayes classifier uses data with numeric features, binary label values (for example y=-1 vs. y=1) and a space of linear maps as a model. It also uses the 0/1 loss (which is the loss for which if y=+1, the loss is +1 until 0 and then it becomes 0).

If you're writing the ERM with 1/0 what we are counting for is te accuracy, which can be also seen as an approximation of the expected 0/1 loss, which can be seen as the probability that our output is different from 1 : $p(h(x) \neq y$. In the binary case

$$\widehat{h}(\mathbf{x}) = \begin{cases} 1 & \text{if } p(y=1|\mathbf{x}) > p(y=-1|\mathbf{x}) \\ -1 & \text{otherwise.} \end{cases}$$

We don't have this probability distribution so we an estimate it from training points.

The Bayes' rule is the following:

$$p(y|\mathbf{x}) = p(\mathbf{x}|y) \cdot p(y) \,/\, p(\mathbf{x})$$

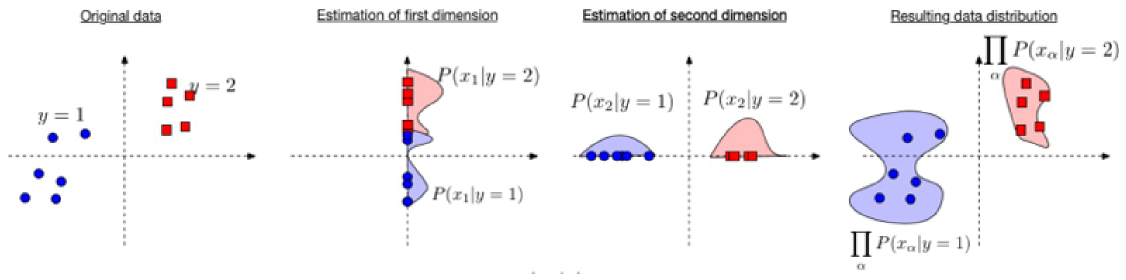
where:

- $p(x)$ is a constant for all y, disregarded for maximum computation
- $p(y)$ is the probability of class y, which is estimated by the relative frequency of class y in the training set

To estimate $p(x|y)$ we can use different ways, but one can be the naive Bayes method:

- using Bayes estimators for different probabilistic models of features $p(x|y)$
- using the naive hypothesis: $p(x_1, ..., x_n|y) = P(x_1|y)p(x_2|y)...p(x_n|y)$
- using statistical independence of attributes

At first we model the distribution of first dimension (x_1) diregarding x_2, and then we do the same with the second dimension (x_2). At the end we multiply the distributions so that we go back tu multidimensional space.

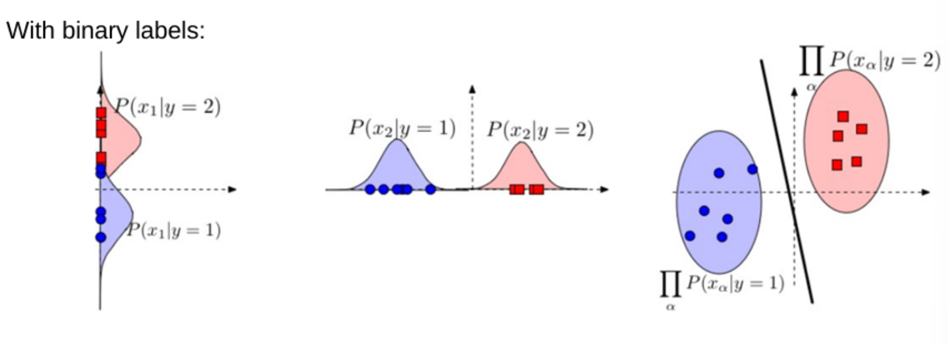

Original data
$y = 1$
$y = 2$
Estimation of first dimension
$P(x_1|y=2)$
$P(x_1|y=1)$
Estimation of second dimension
$P(x_2|y=1)$
$P(x_2|y=2)$
Resulting data distribution
$\prod_\alpha P(x_\alpha|y=2)$
$\prod_\alpha P(x_\alpha|y=1)$

$$\begin{aligned}
h(\mathbf{x}) &= \operatorname*{argmax}_y P(y|\mathbf{x}) \\
&= \operatorname*{argmax}_y \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \\
&= \operatorname*{argmax}_y P(\mathbf{x}|y)P(y) && (P(\mathbf{x}) \text{ does not depend on } y) \\
&= \operatorname*{argmax}_y \prod_{\alpha=1}^{d} P(x_\alpha|y)P(y) && (\text{by the naive Bayes assumption}) \\
&= \operatorname*{argmax}_y \sum_{\alpha=1}^{d} \log(P(x_\alpha|y)) + \log(P(y)) && (\text{as log is a monotonic function})
\end{aligned}$$

To estimate the probability we have again different possibility, among whom we find the gaussian naive bayes. In this case we assume x is a Gaussian with mean and variance depending on y

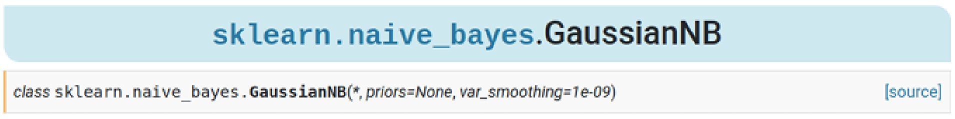
$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

and the mean and variance are estimated by maximum likelihood.



The class is obtained by thresholding a linear map $h(x) = w^t x$.

In python :



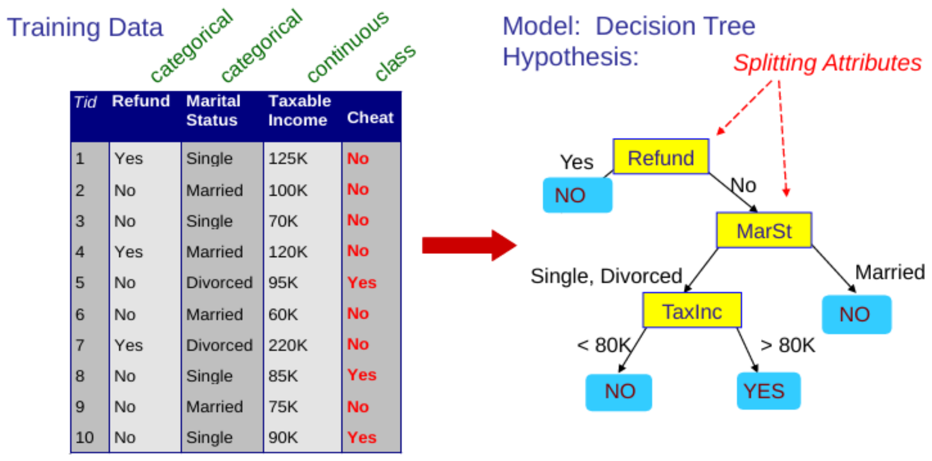
sklearn.naive_bayes.GaussianNB

```
class sklearn.naive_bayes.GaussianNB(*, priors=None, var_smoothing=1e-09)
```

Decision tree and random forest

Decision trees can be used as both regressors and classifiers. They work with datapoints that have numerical and categorical features, as well as arbitrary label values. The model is piece-wise constant, with maps represented by flowcharts. There are different options for the loss function, but losses are optimized locally and greedily, not globally.

Decision trees are models that use a tree-like structure to represent decisions and their possible outcomes or classes. They consist solely of conditional control statements and can be visualized as a flowchart. In this structure, each internal node represents a test on an attribute, each branch represents the outcome of that test, and each leaf node represents a class label.

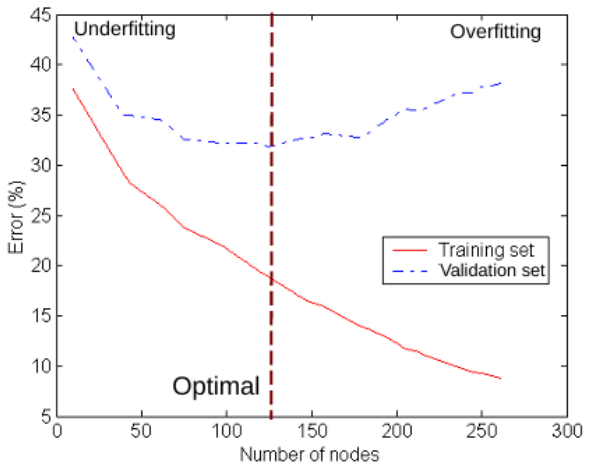


Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

There are many algorithms, but all of them use a greedy strategy. At each step, the best attribute for the split is selected locally, even if it's not the global optimum.

Decision tree validation curve

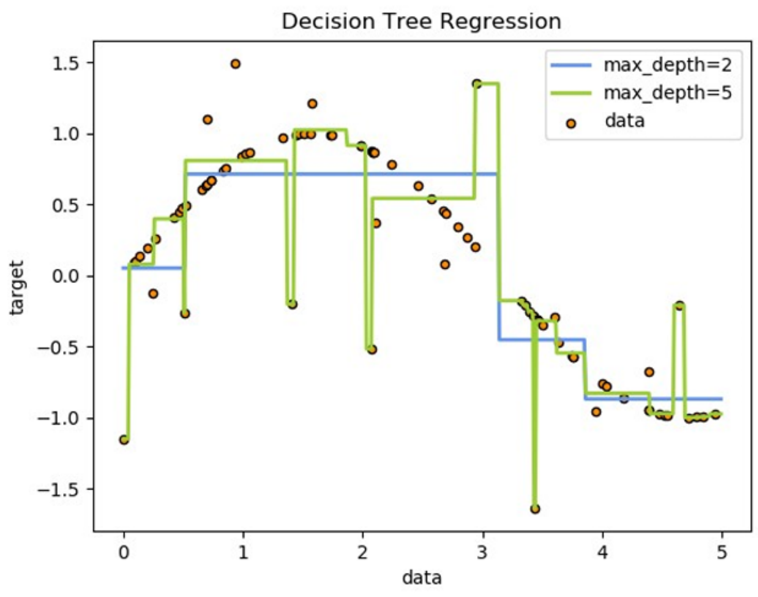
The depth of the tree, which corresponds to the number of nodes, is an important hyperparameter. Increasing the number of nodes always benefits the training set but not necessarily the validation set, even if there's some improvement.



The decision tree allows for non-linear decision boundary, it is extremely fast at classifying unknown record and is easy to interpret for small-sized trees.

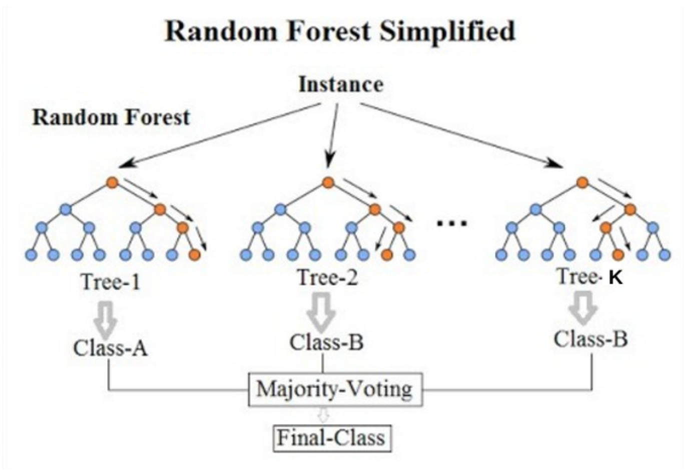
Decision tree regression

Decision trees are applied to regression work similarly to the ones for classification. In regression, the trees output a single number per leaf node instead of a label. A tree can predict a non linear function.



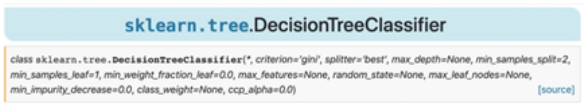
Random forest classifier

Random forest is an ensemble classifier that consists of many decision trees. It outputs the class according to the results of the trees. For each tree of the K trees of the forest, it chooses a subset of n′ features (n is the total number of features) and a subset of m′ training samples (m is the total number of samples in the training data).



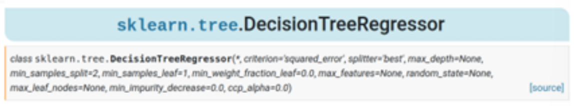
The number of trees could be a default number or based on the validation curve, meaning building trees until the validation error no longer decreases. Keep in mind that increasing the number of trees leads to longer computational times.

As well, the number of feature n′ and samples m′ can be a default number, such as half of them or proportional to data, or you can follow the validation curve.



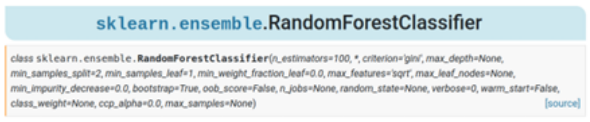
sklearn.tree.DecisionTreeClassifier

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)                                              [source]
```



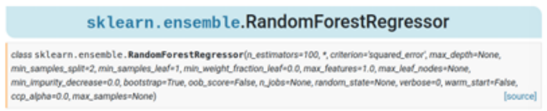
sklearn.tree.DecisionTreeRegressor

```
class sklearn.tree.DecisionTreeRegressor(*, criterion='squared_error', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0)                                            [source]
```



sklearn.ensemble.RandomForestClassifier

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)                                                        [source]
```

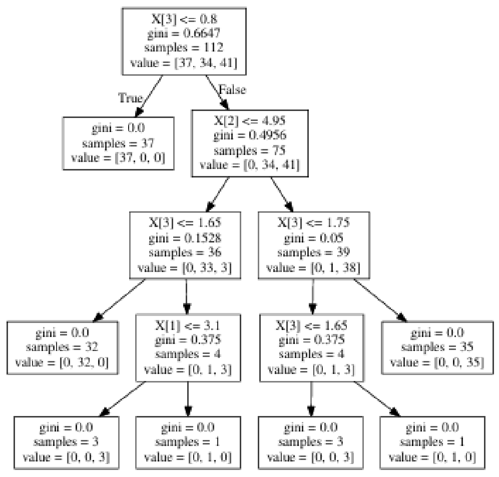


sklearn.ensemble.RandomForestRegressor

```
class sklearn.ensemble.RandomForestRegressor(n_estimators=100, *, criterion='squared_error', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
ccp_alpha=0.0, max_samples=None)                                                                           [source]
```

Once trained, we can export the tree in Graphviz format:

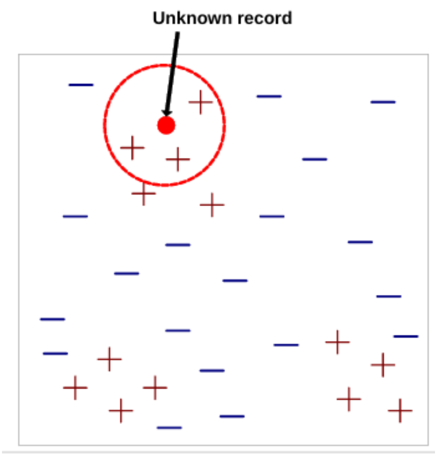
```
import graphviz
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
dot_data = tree.export_graphviz(clf,out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")
```



k-nearest neighbors (K-NN)

K-NN is both a regressor and a classifier where data are numeric features and label are numeric or categoric values. The model is a piecewise constant map and there is no loss function since nothing is optimized nor trained.

This is an instance-based learning as it does not construct a general internal model. It simply stores instances of the training data and uses k closest points for performing classification/regression.

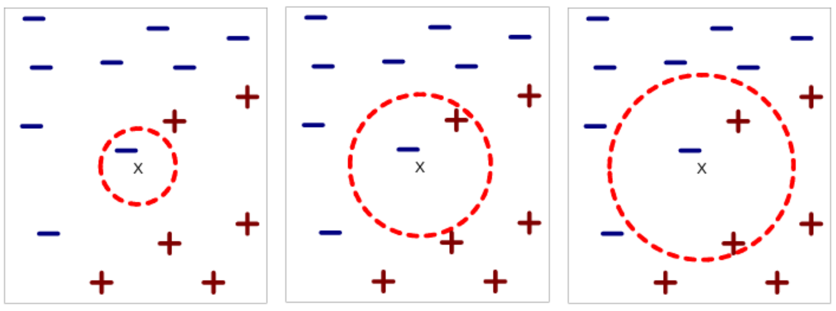


This method requires:

- the set of stored records
- the distance metric to compute distance between records
- the value of k which is the number of nearest neighbors to retrieve

Then, to classify an unknown record, we have to:

- compute the distance to other training records
- identify the k nearest neighbors
- use class labels of nearest neighbors to determine the class label of unknown record



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

When we choose the value of k:

- if k is too small, it can be sensitive to noise points

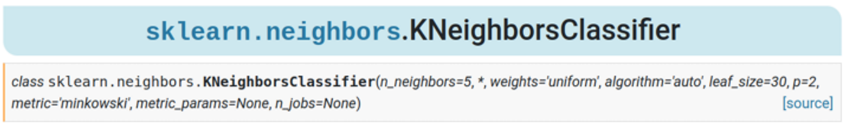
- if k is too large, neighborhood may include points from other classes

If you're not normalizing or standardizing the values, when you compute the distance there would be a single feature which dominates the others.

If you have a lot of features would be hard to find close points.

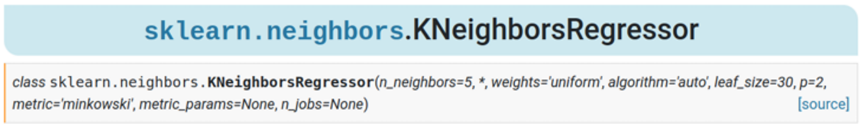
Training time here is 0 cause we are not training anything, but the problem here is the inference time cause computing the distances is not so fast. Moreover, you're wasting a lot of memory.

In python:



sklearn.neighbors.KNeighborsClassifier

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
```
[source]



sklearn.neighbors.KNeighborsRegressor

```
class sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
```
[source]

Multi-class and multi-label classification

Multi-class classification

One vs One

If the label values are 1, 2 and 3, we can break the data points into 3 binary classification problems, one for each pair of classes:

- label values 1 vs 2
- label values 2 vs 3
- label values 3 vs 1

The class which received the most votes is selected.

One vs rest

If the label are 1, 2 and 3, we can break the data points into 3 binary classification problem:

- label values 1 vs either 2 or 3
- label values 2 vs either 1 or 3
- label values 3 vs either 2 or 1

For one vs rest, we can also choose one sub-problem for each label value k, where each sub-problem is like "label = c or not". In this case we learn hypothesis h_k for each sub-problem and then predict c with highest confidence/probability $|h_k|$.

$$\hat{y} = \underset{k \in \{1...K\}}{\operatorname{argmax}} \ f_k(x)$$

Multi-class logistic regression

There are specific loss functions for multi-class data. The 0/1 loss also work for > 2 label values (classes).

Multi-label classification

In multi-label classification each sample can have more than one output.

The possible approaches are:

- naive approach: you can consider each label separately, solve binary/multi-class problem for each label and ignores correlations among different labels
- multi-class approach: each combination of label values defines a category, you can obtain a multi-class problem with many classes but there is the possibility of huge number of resulting categories
- multi-task learning approach: each individual label results in separate learning task, it uses similarities between learning tasks which inform regularization techniques and at the end you can combine the loss to learn together the two hypothesis.