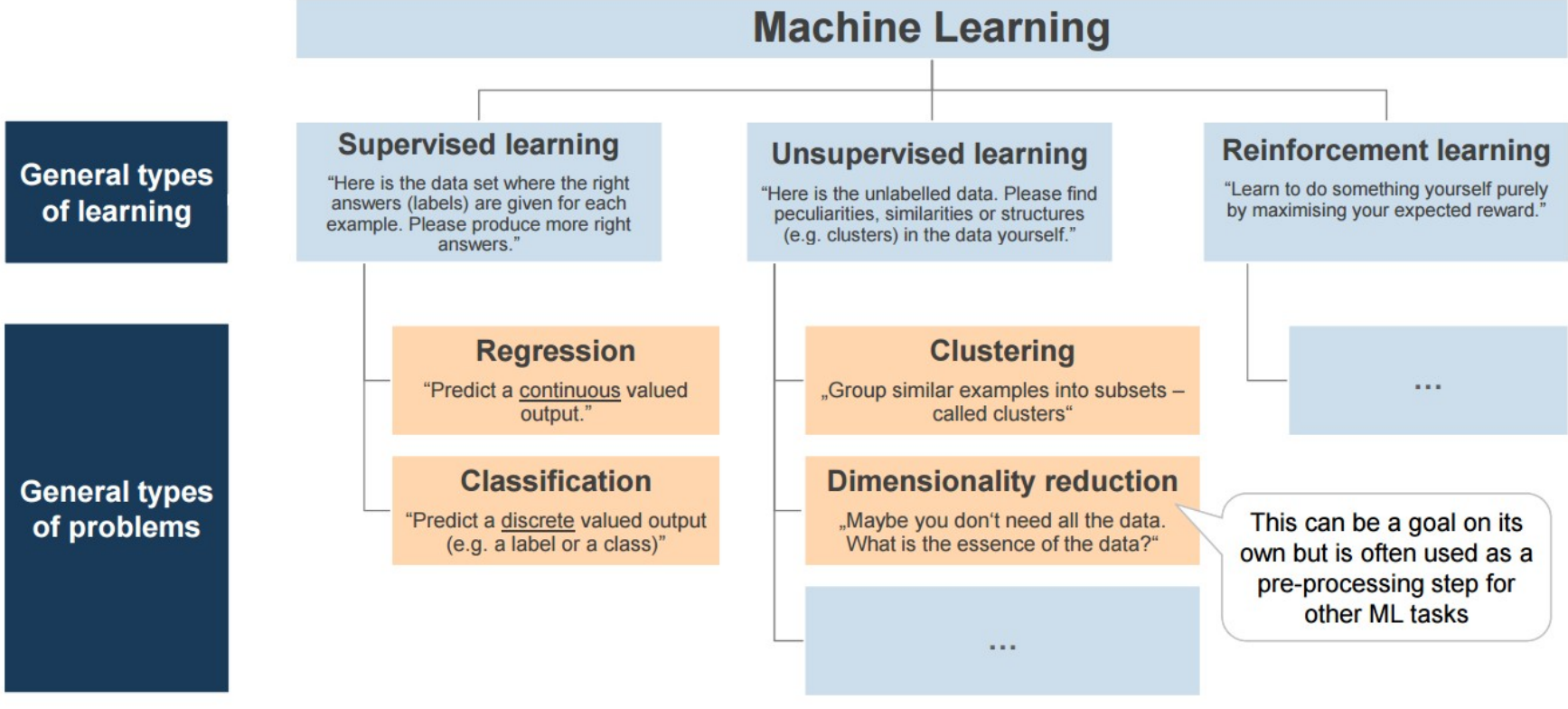
Machine Learning Recap

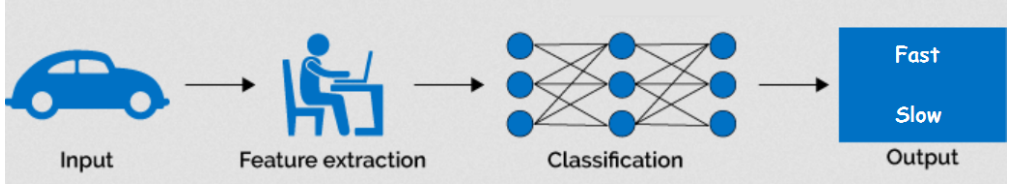
Categories of Machine Learning



Classification

You have a set of observations with features and they have been given a label or class. The question is:

- How do you classify a new observation ?



How to tackle with scikit learn

Load training data sample - Represents the features of the sample. The features must have a NUMERICAL value. If data is categorical, you must create a new feature per category of the categorical feature.

```
X = [ [1,2], [2,3] ]
```

Training Data Labels or targets or class - Each sample in the training data has a label. The first sample in X takes the first label in y and subsequentially:

```
y = [ 1,2 ]
```

Import the model

```
# In this example, we load a decision tree
from sklearn.tree import DecisionTreeClassifier
clf= DecisionTreeClassifier()
```

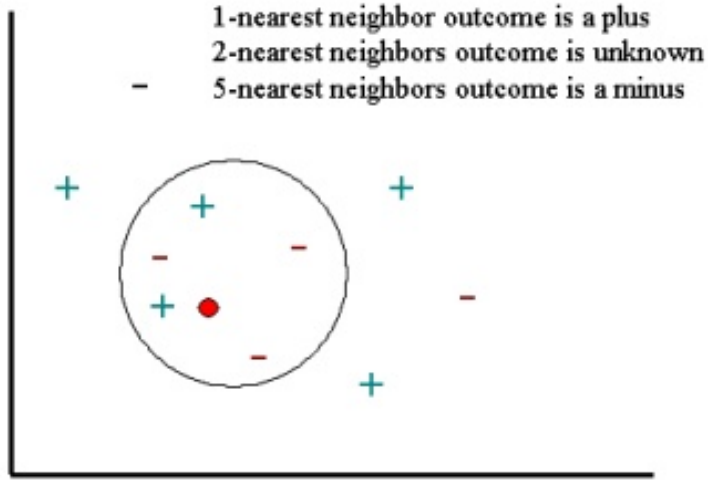
Train the model

```
clf.fit(X,y)
```

Now you can do **predictions**

```
clf.predict([[5,2],[3,5]])
```

k-nearest Neighbors



Sklearn Implementation

Link: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

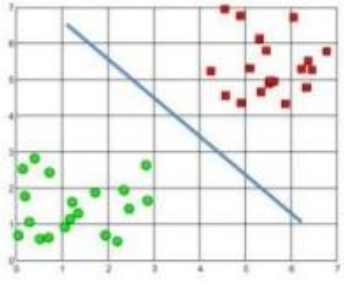
```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
KNeighborsClassifier(...)
print(neigh.predict([[1.1]]))
print(neigh.predict_proba([[0.9]]))
```

Main Parameters:

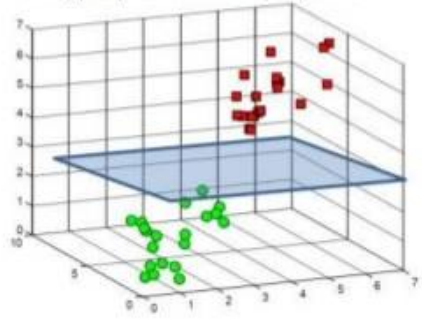
- n_neighbors : Number of neighbors to use by default for k_neighbors queries.
- weights : weight function used in prediction. Possible values: 'uniform', 'distance'
- algorithm : Algorithm used to compute the nearest neighbors. Possible values: 'auto', 'ball_tree', 'kd_tree', 'brute'

Support Vector Machines

A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane



A hyperplane in $\mathbb{R}^n$ is an n-1 dimensional subspace

Sklearn Implementation

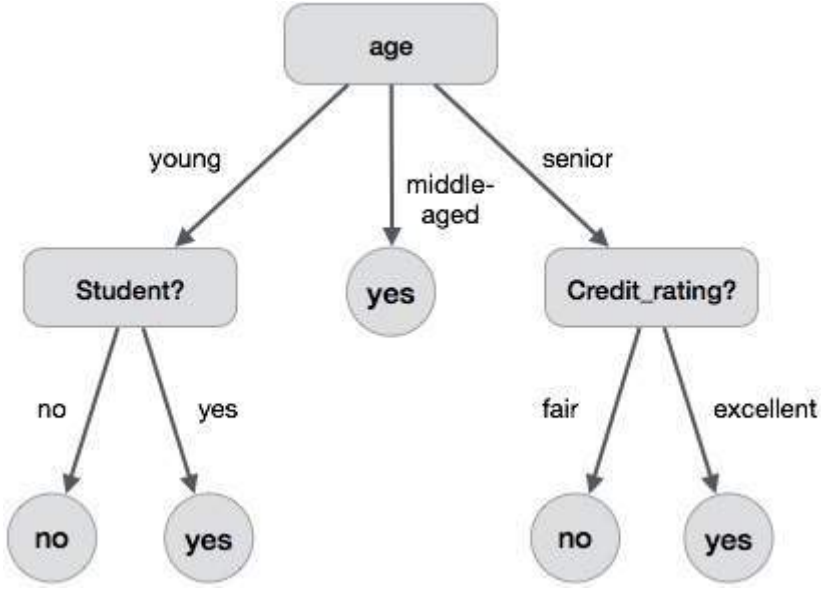
Link: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
y = np.array([1, 1, 2, 2])
from sklearn.svm import SVC
clf = SVC()
clf.fit(X, y)
print(clf.predict([[-0.8, -1]]))
```

Main Parameters:

- C : Penalty parameter C of the error term.
- kernel : Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.
- degree : Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
- gamma : Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_features will be used instead.

Decision Tree



Sklearn Implementation

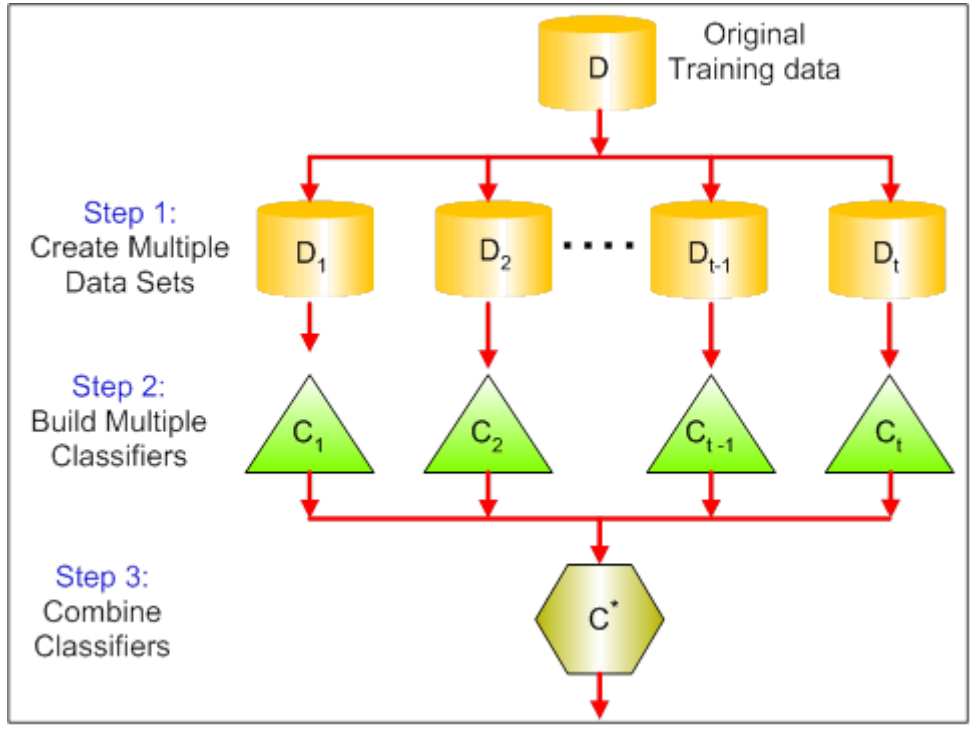
Link: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)
iris = load_iris()
cross_val_score(clf, iris.data, iris.target, cv=10)
```

Parameters:

- criterion : The function to measure the quality of a split. Supported criteria are 'gini' for the Gini impurity and 'entropy' for the information gain.
- max_features : The number of features to consider when looking for the best split:
- max_depth : The maximum depth of the tree.

Ensemble: Bagging



Sklearn Implementation

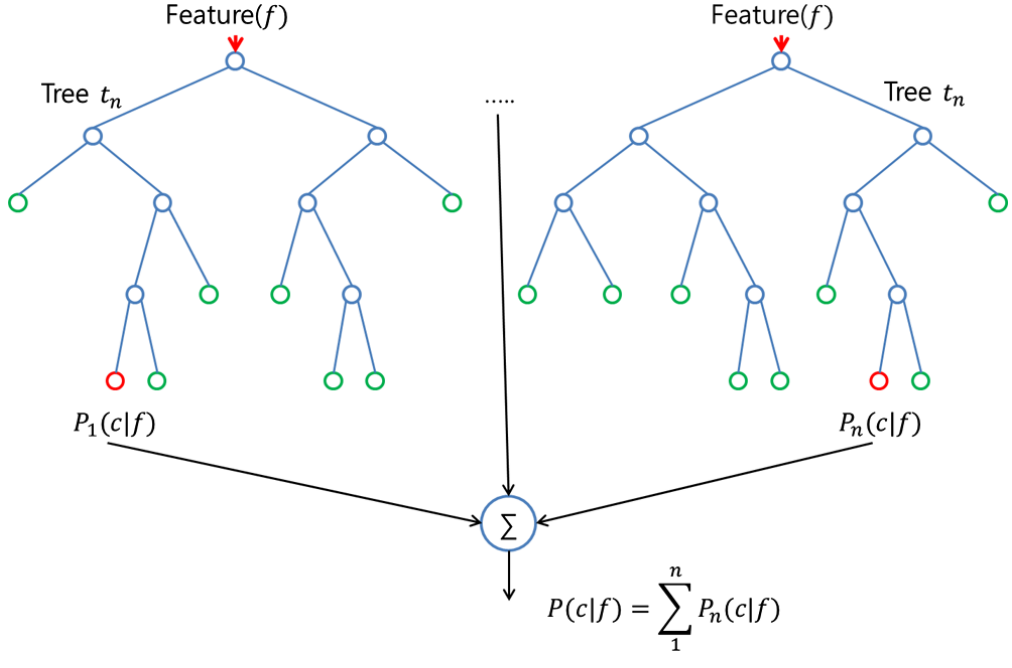
Link: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

```
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
clf = BaggingClassifier(DecisionTreeClassifier(), max_samples=0.5, max_features=0.5)
clf.fit(X,y)
```

Main Parameters:

- base_estimator : The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.
- n_estimators : The number of base estimators in the ensemble.
- max_samples : The number of samples to draw from X to train each base estimator.
- max_features : The number of features to draw from X to train each base estimator.

Random Forest



Sklearn Implementation

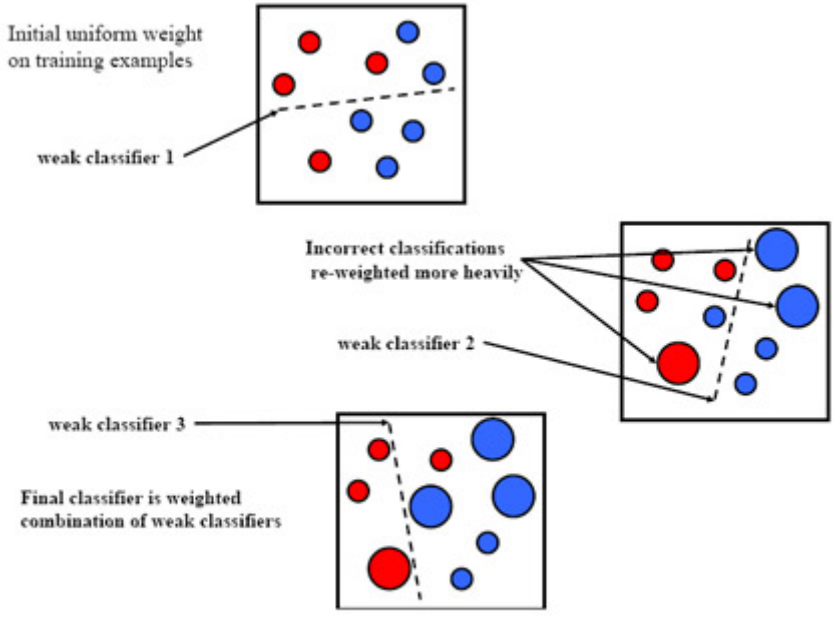
Link: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10)
```

Parameters:

- n_estimators : The number of trees in the forest.
- criterion : The function to measure the quality of a split. Supported criteria are 'gini' for the Gini impurity and 'entropy' for the information gain. Note: this parameter is tree-specific.
- max_features : The number of features to consider when looking for the best split:
- max_depth : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. min_samples_split : The minimum number of samples required to split an internal node:

Adaboost



$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$

Sklearn Implementation

Link: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100)
cross_val_score(clf,X,y,cv=5).mean()
```

Parameters:

- baseestimator : The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper classes and nclasses attributes.
- n_estimators : The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

Voting Classifier

How to combine the classifiers?

- (weighted) Majority voting
  - Class label output
  - Select the class most voted for

$$\sum_{t=1}^{T} d_{t,J} = \max_{j=1}^{C} \sum_{t=1}^{T} d_{t,j}.$$

$$\sum_{t=1}^{T} w_t d_{t,J} = \max_{j=1}^{C} \sum_{t=1}^{T} w_t d_{t,j}$$

- Mean rule
  - Continuous output
  - Support for class w_j is average of classifier output

$$\mu_j(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} d_{t,j}(\mathbf{x})$$

- Product rule
  - Continuous output
  - Product of classifier output

$$\mu_j(\mathbf{x}) = \frac{1}{T} \prod_{t=1}^{T} d_{t,j}(\mathbf{x})$$

www.danielroggen.net droggen@gmail.com

Sklearn Implementation

Link: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

```
from sklearn.ensemble import VotingClassifier
  from sklearn.naive_bayes import GaussianNB
clf1 = RandomForestClassifier(random_state=1)
                clf2 = GaussianNB()
eclf = VotingClassifier(estimators=[('random', clf1), ('gaussian', clf2)], voting='hard')
  cross_val_score(eclf, X, y, cv=5).mean()
```

Main Parameters:

- estimators : Invoking the fit method on the VotingClassifier will fit clones of those original estimators that will be stored in the class attribute self.estimators_.
- voting : If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers.
- weights : Sequence of weights (float or int) to weight the occurrences of predicted class labels (hard voting) or class probabilities before averaging (soft voting). Uses uniform weights if None.

GridSearchCV: Finding out the best parameters

Link: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svr = svm.SVC()
clf = GridSearchCV(svr, parameters)
clf.fit(iris.data, iris.target)
clf.get_params()
```