# 01_Incremental_Algs

December 11, 2018

Convex and Distributed Optimization
Franck Iutzeler2018/2019
Lab. 1 - Incremental algorithms

---

```
In [2]: import lib.notebook_setting as nbs
        import warnings

        warnings.filterwarnings('ignore')

        packageList = ['IPython', 'numpy', 'matplotlib', 'pandas' , 'sklearn']
        nbs.packageCheck(packageList)

        nbs.cssStyling()
```

```
[Python version]         3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64

[Packages versions]

IPython              :        6.4.0
numpy                :        1.14.3
matplotlib           :        2.2.2
pandas               :        0.23.0
sklearn              :        0.19.1
```

```
Out[2]: <IPython.core.display.HTML object>
```

Outline 1) Classification 2) Basic Manipulations on Datasets 3) Logisitic loss optimization 4) Incremental algorithms 5) Larger-scale experiments

Warning: This lab assumes basic knowledge about Python and basic machine learning libraries (numpy, scikit-learn, pandas). If you are not familiar with those, check out this introduction.

## 0.1 1) Classification

Go to top

The problem of classfication if the one of finding rules for assigning a class to a given vector from already classified data, for instance, the 2D points below:

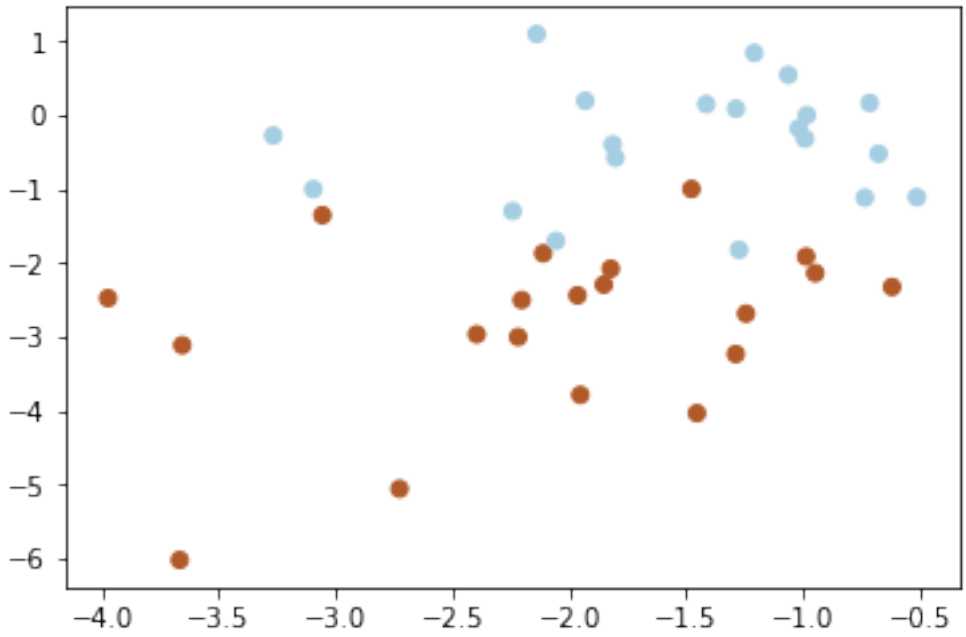
```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.datasets import make_blobs
        %matplotlib inline

        # we create 40 separable points in R^2 around 2 centers
        X, y = make_blobs(n_samples=40, n_features=2, centers=2 , random_state=48443)

        print(X[:5,:],y[:5]) # print the first 5 points and labels

        plt.scatter(X[:, 0], X[:, 1], c=y,  cmap=plt.cm.Paired);
```

```
[[-9.46919425e-01 -2.13903597e+00]
 [-2.72610707e+00 -5.06186514e+00]
 [-2.24209576e+00 -1.29901759e+00]
 [-1.93127085e+00  1.95574242e-01]
 [-9.83070494e-01 -2.42695118e-04]] [1 1 0 0 0]
```




Support Vector Machines (SVM) are based on learning a vector $w$ and an intercept $b$ such that the hyperplane $w^T x - b = 0$ separates the data i.e. $a$ belongs to one class if $w^T a - b > 0$ and the other elsewhere.

The `scikit-learn` library provides a classification module:

```
In [6]: from sklearn.svm import SVC # Support vector classifier i.e. Classifier by SVM

        modelSVMLinear = SVC(kernel="linear")
        modelSVMLinear.fit(X,y)
```

```
Out[6]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
```

The following illustration can be found in the Python Data Science Handbook by Jake VanderPlas.

```
In [7]: def plot_svc_decision_function(model, ax=None, plot_support=True):
            """Plot the decision function for a 2D SVC"""
            if ax is None:
                ax = plt.gca()
            xlim = ax.get_xlim()
            ylim = ax.get_ylim()

            # create grid to evaluate model
            x = np.linspace(xlim[0], xlim[1], 30)
            y = np.linspace(ylim[0], ylim[1], 30)
            Y, X = np.meshgrid(y, x)
            xy = np.vstack([X.ravel(), Y.ravel()]).T
            P = model.decision_function(xy).reshape(X.shape)

            # plot decision boundary and margins
            ax.contour(X, Y, P, colors='k',levels=[ 0], alpha=0.5,    linestyles=[ '--'])

            ax.set_xlim(xlim)
            ax.set_ylim(ylim)
```

```
In [8]: plt.scatter(X[:, 0], X[:, 1], c=y ,  cmap=plt.cm.Paired)
        plot_svc_decision_function(modelSVMLinear)
```