Machine Learning course
Practical session 2
Scikit-learn and classification
KNN, Boosting and SVM

This work will be done during at least 2 sessions.
This document is available on
http://perso.univ-st-etienne.fr/habrarda/ENSML/

Once finished, send you work to amaury.habrard@univ-st-etienne.fr

1 Generating datasets

We will need to use some datasets and fortunately, `scikit-learn` offers many ways to use some datasets. You can find some informations about these available datasets here:
http://scikit-learn.org/stable/datasets/index.html

1.1 Random datasets

You can generate automatically some datasets thanks to the function `make_classification`. Here is an example for generating a datasets with 2 features:

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=50,n_features=2, n_redundant=0, n_informative=2,
                           random_state=2, n_clusters_per_class=1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
plt.show()
```

Arguments are relatively explicit, you can anyway find more information here
http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

1.2 Existing datasets

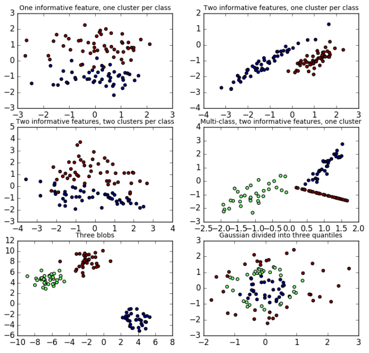
Apply the same procedure as in the previous section to 2 existing datasets.

- The 2-moons dataset provided in the scikit-learn distribution

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
X, y = make_moons(noise = 0.1, random_state=1, n_samples=40)
plt.scatter(X[:,0],X[:,1], c = y, s = 100)
plt.show()
```

-
```
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=400, factor=.3, noise=.05)
plt.scatter(X[:,0],X[:,1], c = y, s = 100)
plt.show()
```

- Randomly generated w.r.t. to Gaussian distributions, blobs.
  More information on
  http://scikit-learn.org/stable/auto_examples/datasets/plot_random_dataset.html



- The toy datasets http://scikit-learn.org/stable/datasets/index.html#toy-datasets
  For this session, you can use the `iris` and `digits` datasets. For 2D visualization, you can use only 2 attributes out of the 4 original ones for the `iris` dataset. The `digits` dataset is constituted of images that you can visualize thanks to the `images` parameter
  http://scikit-learn.org/stable/auto_examples/cluster/plot_digits_agglomeration.html#sphx-glr-auto-examples-cluster-plot-d

1.3 Evaluation of the models

For evaluating a model, you must test the model on data different from the ones used to learn the model. `scikit-learn` offers many solutions:

- Separating data into a training and a testing dataset: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- Using cross-validation, check the previous practical session.
- The previous methods will be sufficient for this session, but note that more methods exist: http://scikit-learn.org/stable/modules/cross_validation.html

2 K-Nearest-Neighbors

Try to implement your own version of KNN, ideally in python.

Note that there exists an implementation of KNN in `scikit-learn` documented here: http://scikit-learn.org/stable/modules/neighbors.html The KNN algorithm is implemented in the neighbors package. The command `KNeighborsClassifier(nb_neighbors)` creates an object "KNN classifier", `fit(X, Y)` indicates that `X` and `Y` define the classifier, the command `predict` can be used to classify new examples whereas `predict_proba` estimates the probability of the given classification. `score` computes the global score of the classifier on a given dataset.

Work to do:

1. Generate different datasets in 2D with the different functions proposed above for binary classification. You should use `moons, circles`, 1 or 2 examples coming from `make_classification`. (If you have time, you can later use real benchmarks or Gaussian generated data)
2. Evaluate the performance obtained with different values for k (the number of nearest neighbors),
3. and try to visualize in 2D the evolution of this frontier. Do not forget to use a test dataset different from the learning sample. For this part, you can try to generate a grid in the following way:

```
import numpy as np
#we create a mesh to plot in
h = .02  # grid step
x_min= X[:, 0].min() - 1
x_max= X[:, 0].max() + 1
y_min = X[:, 1].min() - 1
y_max = X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
#the grid is created, the intersections are in xx and yy
```

Now assuming that you have learned a classifier referenced by `clf`, you can try to classify all the points from the grid and display the result:

```
Z2d = clf.predict(np.c_[xx.ravel(),yy.ravel()]) # we predict all the grid - you use your own algo
Z2d=Z2d.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx,yy,Z2d, cmap=plt.cm.Paired)
plt.show()
```

You can also plot the learning points by adding the following commande just before `plt.show()`:

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

Note some informations about the colors can be found here
http://matplotlib.org/examples/color/colormaps_reference.html.

3 Boosting

Try to implement you own `Adaboost` algorithm. As a weak classifier, you can use `stumps` that correspond to decision trees of depth 1 that are implemented on `scikit-learn` and available here (you may also propose your own implementation but this requires more work):
http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

Note that in `scikit-learn`, the classifier `AdaBoost` is implemented and documented here:
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html (note that `AdaBoost` is part of a more general family of ensemble classifier). You have two main parameters

- `base_estimator`: the type of weak classifier used, by default this is a decision tree classifier, but you can fix the weak classifiers as `stumps`

  ```
  ada = AdaBoostClassifier(base_estimator=dt_stump, n_estimators=100)
  ```

- `n_estimators`: the number of iterations

You still have the `fit` and `predict` methods. Note that you can have access to the list of weak classifiers and their weights in the AdaBoost object attribute.

Work to do: Repeat the same exercise as for the KNN classifiers in particular you can see the evolution of the error with respect to the number of iterations. If possible you can also try to change the weak classifiers used.

Some complementary examples:
http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_hastie_10_2.html
http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_twoclass.html

4 SVM

4.1 A little Warm-up

To familiarize yourself with SVM, you can have a look to the following demos on the Web:

- `http://cs.stanford.edu/people/karpathy/svmjs/demo/`
- `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`
  go in the *Graphic Interface* section (there is a Java and a javascript version).

The first demo allows you to compare the effect of RBF kernel and linear kernel (with margin band highlighted). The second to compare different kernels, you can type the following options in the option bar:

- `-t <value>`: specify the kernel function (0 for the linear kernel, 1 for the polynomial one, 2 for the RBF one, 3 for the sigmoid one),
- `-c <value>`: value of the C parameter,
- `-g <value>`: value of $1/\sigma^2$ in the Gaussian/RBF kernel,
- `-d <value>`: degree of the polynomial kernel,
- `-r <value>`: value of additional parameter in the kernel function (check the document just below the graphic).

4.2 Scikit-Learn

You can try to develop your own implementation of SVM, but this requires to implement an optimization algorithm which is a bit out of the scope of this practical session (anyway you are free to try ;-).

The documentation for SVM in the `scikit-learn` library is available at this url: `http://scikit-learn.org/stable/modules/svm.html`

We will use mainly the SVC class, the documentation can be found here, the implementation is mainly based on the libSVM library.:
`http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`
You will need mainly the following options:

- **C**: penalty parameter of the error term, default 1.0
- **kernel**: default 'rbf', options: linear', 'poly', 'rbf', 'sigmoid', 'precomputed'
- **gamma**: option for rbf kernel, default $1/n\_features$
- **degree**: for polynomial kernel, default 3.
- **coef0**: bias term for polynomial kernel (and sigmoid)
- **decision_function_shape**: for multiclass classification, it uses 'ovr': one versus rest, and 'ovo' one versus one methods.
- field **support_vectors_**: the list of support vectors

Other options/fields exist, like **class_weight** that allows to weight the different classes in unbalanced settings, check the documentation.

A first example:

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]]) #we create 4 examples
y = np.array([-1, -1, 1, 1])
from sklearn.svm import SVC #import de la classe SVC pour SVM
classif=SVC() #we create a SVM with default parameters
classif.fit(X,y) #we learn the model according to given data
res=classif.predict([[-0.8, -1]]) #prediction on a new sample
print(res);
```

Repeat the same exercise as for KNN and Boosting. Here, you must compare the influence of the different kernels and associated parameters.

5 Compare all the models

5.1 On the benchmarks

Try to compare the 3 methods on a synthetic dataset you have generated, on the `iris` dataset and on the digits `dataset`.
As an illustration, you can have look to the following example:
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Note that the best parameters for each model vary with the task, you should do parameter selection for each model.

5.2 Another real dataset - ozon peaks prediction

Download the ozon dataset (`ozone.dat`). This dataset comes from MétéoFrance records and is constituted of the following attributes:

- `JOUR`: type of day, holiday (1) or not (0).
- `O3obs`: ozone concentration observed the next day at 17h (local time), often at the maximum pollution rate.
- `MOCAGE`: prediction of the pollution made by a deterministic model/
- `TEMPE`: temperature for the next day at 17h.
- `RMH2O`: humidity rate
- `NO2`: nitrogen dioxide concentration
- `NO`: nitric oxide concentration
- `STATION`: place from where the observations are taken (`Aix-en-Provence, Rambouillet, Munchhausen, Cadarache` or `Plan de Cuques`)
- `VentMOD`: wind strength
- `VentANG`: wind orientation

A first remark is that the STATION attribute contains categorical values that have to be converted. A first solution would consist in associating an integer to each station, however this is not very relevant: it biases the dataset because some stations will get more "importance" than others just because of the integer choice. A better solution is to replace each value by a binary vector containing only one 1. Example `Aix-en-Provence` could be encoded as $1, 0, 0, 0, 0$, `Rambouillet` as $0, 1, 0, 0, 0$, `Munchhausen` as $0, 0, 1, 0, 0$ and so on.

The dataset is in CSV like format (with space used as a separator). To load the dataset you can use the `read_csv` function from `panda` library.

```
import pandas as pd
input_file = "mydata.csv"
df = pd.read_csv(input_file)
```

The delimiter can be chosen for example, more information here:
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

Another point is data scaling: if you do not scale data, attributes defined over large intervals will take more importance than those defined on a significantly smaller interval. A way to tackle this problem is to use a normalizing function

```
from sklearn.preprocessing import StandardScaler
...
# normalize dataset for easier parameter selection
X = StandardScaler().fit_transform(X) #where X is your data matrix
```

Work to do: classification problem by predicting ozone peaks

- We define an ozone peak as an `O3obs` value greater than 150. Assuming that the `O3obs` value (non normalized) is located in the first column of a matrix Z, then you can use the following commande to create your classes:
  `y = Z[:,0] > 150`
- You must then select the correct attributes from the CSV file to define the data matrix and the label set.
- To evaluate correctly the models, you must use a cross-validation procedure. However, to tune the hyperparameters you must apply another cross-validation on the training dataset and then learn the model on the full training data one the parameters have been chosen.
- You must compare different svm classifiers. What is the best performance with cross validation that you can achieve on this dataset, with or without data normalization.
- If you have time, you can try to predict the ozon peak using regression.