Sequence 1 exercices:

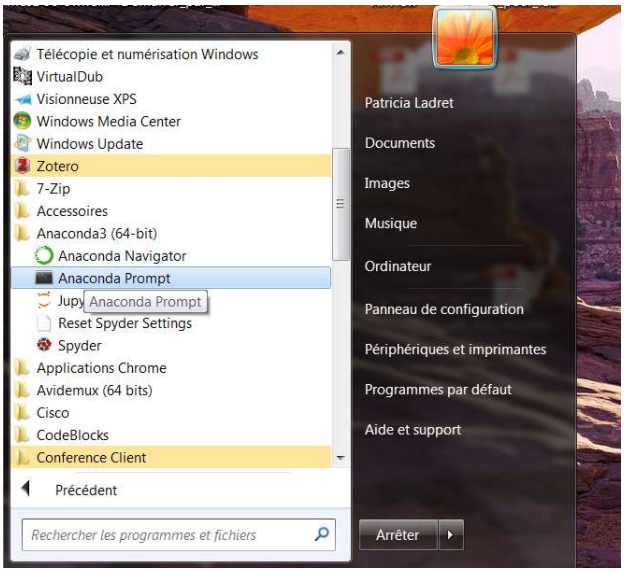
Naive Bayes classification and k-nn classification

0. Python package and Spyder editor installation

All the required softwares are available on any Phelma computer. If you want, you can install the required softwares on your own computer. All softwares are free.

0.1 Python install with Anaconda

- Go to the following site https://www.anaconda.com/distribution/ and choose the version compliant with your system.
- Anaconda for Python is installed with 150 packages. The main ones are used for Image and Signal processing: numpy, scipy, matplotlib, scikit-image, scikit-learn, etc...
- If you need to install other packages, launch an Anaconda command window,



Run the install by typing one of the following commands at the Ananconda prompt:

- conda install [package_name]
- pip install [package_name] (or pip3 according to your system)

0.2 Mises à jours Anaconda et packages

For package updating:

- Global Anaconda package update
  - conda update anaconda (and before: conda update conda)
- Specific package update:
  - pip upgrade [package_name]
  - conda update [package_name]

0.3 Development Interface, IDE : Spyder

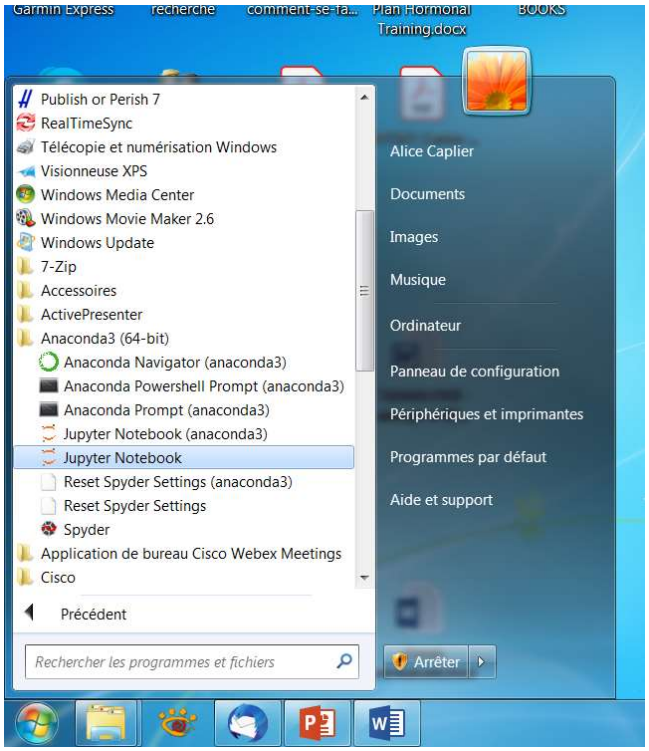
Once Anaconda is installed, access to development interface (IDE) Spyder is available.

Run the program by clicking on the icon: 

Before starting, look at the different available spyder windows

0.4 Jupiter Notebook

With anaconda, you can access Jupiter Notebook which is a convenient tool for quick development. It will be used for some exercises. Files related to Jupiter Notebook have the .ipynb extension.
To access the Jupiter Notebook, go to the Anaconda directory contain and launch Jupiter Notebook as indicated below:



And then launch any file with the extension .ipynb

0. About the scikitlearn library

scikitlearn (https://scikit-learn.org/) is a Python library dedicated to machine learning. Whatever the type of supervised classifier you have chosen, the main steps of a supervised classification algorithm are the following:

- ✓ Download the data, X for the features and Y for the labels
- ✓ Divide the data between train set and test set
- ✓ Create the class instance of the classifier you are going to use. It allows the specification of the classifier parameters if necessary
- ✓ Train the classifier on the train dataset by using the *fit()* function

- ✓ Predict the answers of the classifier on the test dataset by using the *predict()* function
- ✓ Evaluate the performances of the classification by computing the accuracy score, the confusion matrix and other performances criterions (Precision, Recall, AUC…)

1. Diabete prediction (Homework)

Main purpose of the exercise: implement a Naïve Bayes and a K-NN classifier; to be familiar with accuracy score and confusion matrix. To be aware of data normalization requirement.

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabete, based on certain diagnostic measurements included in the dataset.

768 subjects, 8 attributes, 1 decision

8 different data features in the dataset

- ✓ Pregnancies : Number of times pregnant
- ✓ Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- ✓ BloodPressure: Diastolic blood pressure (mm Hg)
- ✓ SkinThickness: Triceps skin fold thickness (mm)
- ✓ Insulin: 2-Hour serum insulin (mu U/ml)
- ✓ BMI: Body mass index (weight in kg/(height in m)^2)
- ✓ DiabetesPedigreeFunction: Diabetes pedigree function
- ✓ Age: Age (years)

A single outcome: diabete or not diabete (one class classification problem)

- ✓ Outcome: Class variable (0 or 1) (in the dataset, 268 of 768 are 1, the others are 0)

1.1 Naïve Bayesian classification

NB: when computing the feature xi probability given a Y class (that is $Pr(X=x_i / Y=y_k)$), the related model assumption is different depending on the feature data nature. As a consequence, when implementing a Naïve Bayes Classifier, it is necessary to choose between three different models of the sklearn.naive_bayes module that are:

- GaussianNB() when the features are continuous values
- BernouilliNB() when the features are binary values
- CategoricalNB() when the features come from categorical values

1.1.1 Open the file *diabete_student_code.py* and split the whole *diabete.csv* dataset in two parts in order to build the training set containing 70% of the data (take the 70% first samples and create X_{train} and Y_{train} for the feature matrix and the label matrix respectively) and the testing set containing 30% of the data (take the last 30% samples and create X_{test} and Y_{test} for the feature matrix and the label matrix respectively).

1.1.2 Considering the GaussianNB() function of the scikitlearn library (`from sklearn.naive_bayes import GaussianNB`), train a Gaussian naïve Bayes classifier from the training set (cf. *fit()*) and predict the labels of the data of the testing set (cf. *predict()*). Look at the scikitlearn documentation (https://scikit-

learn.org/stable/index.html) in order to see how to use the corresponding functions. Compute the classification accuracy on the training set and on the testing set. The accuracy is defined as the proportion of good answers. Comment the results.
Rmk : GaussianNB model is used when the features are non categorical values.

1.1.3 Compute the confusion matrix (cf. *confusion_matrix()* function from scikitlearn.metrics) in order to see more precisely the classifier performances for each class. Report and interpret the TP, TN, FP, FN values.

1.2 K-NN classification

1.2.1 Implement a K-NN classifier (cf *neighbors.KNeighborsClassifier() and fit())* and predict the label on the test dataset (cf. *predict*()). Compute the accuracy and the confusion matrix of the classifier on the test dataset for K= 3.

1.2.2 Data analysis: the graph below gives a sub sample of the provided dataset.

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0

It can be noticed that from one feature to the other, the range might be different. And in case of a feature has an order of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. As a consequence, learning algorithms generally benefit from standardization of the data set. In practice the two main current ways for data normalization are the standard normalization which transforms the data in centering it by removing the mean value of each feature, then scaling it by dividing non-constant features by their standard deviation (cf. *Standard_Scaler()* python function). The second current way for data normalization is the min-max scaler which transforms the data in the following way (cf. *MinMaxScaler()*function): $X-X_{min} / X_{max}-X_{min}$
Implement a MinMax data normalization process before 3-NN classification and compare the results with those obtained without data normalization.

Do you think that data normalization would have been necessary with the Naïve Bayes classifier?

2. Credit card fraud prediction (class work)

Main purpose of the exercise: be careful about the dataset repartition among classes and dealing with unbalanced classes.

It is important for credit card companies to be able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days with 284,807 transactions.

The dataset contains only numerical input variables which are mostly the result of a Principal Component Analysis transformation (PCA). Unfortunately, due to confidentiality issues, neither the original features and nor more background information about the data are available. So in the dataset, there are these 28 input continuous features (V1 – V28) coming from PCA analysis but it is not possible to have the physical meaning of those features. Anyway, this will not hurt the classification process.

Dataset organization

284807 samples

30 input features

- ✓ Time: Number of seconds elapsed between this transaction and the first transaction in the dataset
- ✓ V1-V28: result of a PCA Dimensionality reduction to protect user identities and sensitive features
- ✓ Amount: Transaction amount

1 Output:

- ✓ Class: 1 for fraudulent transaction, 0 otherwise

3.1 Open the file *creditcard_prediction_student_code.py*. Data set analysis: report the total number of samples and the proportion of samples in each class (fraud or not fraud) for the whole dataset.

3.2 By using the *train_test_split()* function, split the whole dataset in two parts in order to build the training set containing 70% of the data and the testing set containing 30% of the data. As a matter of fact, the data splitting has to be random especially when the data are sorted in the dataset (all first class samples then second calls samples and so on).
Rmk: if you want to have reproducible results between different runs, put the random_state variable to 0 in the *train_test_split()* function.

3.3 Train a naïve Bayes classifier on the training set and predict the label of the data in the testing set. Justify the model to be used in order to model $Pr(X=x_i / Y=y_k)$. Report the training and the testing accuracies. What do you think about the global accuracy?

3.4 Compute the confusion matrix on the test set (cf. *confusionmat()*) and report the per class accuracy. Compute the precision and the recall values. Conclusion?

3.5 Proceed to a 3-NN classification with prior date feature normalization. What is the problem?

3.6 When applying a 3-NN classifier, the global accuracy on the test set is 99.95% with the following confusion matrix:

85288	8
37	110

Compute the Precision and Recall criteria in that case (use the *classification_report()* function form sklearn.metrics) and compare the results with those obtained with the naïve Bayesian classifier.

3. Caesarian prediction (optional class work)

The aim of this exercise is to predict if a woman will need a caesarian or not. The prediction is built from the Caesarian.txt dataset. This dataset contains information about caesarian results of 80 pregnant women with the most important characteristics of delivery problems in the medical field.
The chosen features are age, delivery number, delivery time, blood pressure and heart status (see below for detail).

80 subjects, 5 attributes, 1 decision

5 different data features

- ✓ Feature 'Age' { 22,26,28,27,32,36,33,23,20,29,25,37,24,18,30,40,31,19,21,35,17,38 }
- ✓ Feature 'Delivery number' { 1,2,3,4 }
- ✓ Feature 'Delivery time' { 0,1,2 } -> {0 = timely , 1 = premature , 2 = latecomer}
- ✓ Feature 'Blood of Pressure' { 2,1,0 } -> {0 = low , 1 = normal , 2 = high }
- ✓ Feature 'Heart Problem' { 1,0 } -> {0 = apt, 1 = inept }

By looking at the dataset, it appears there are two types of features : continuous values for Age and Feature Delivery Number and categorical values for the other features.

One label as output (one class only)

- ✓ Label Caesarian {0,1} -> {0 = No, 1 = Yes}

3.1 Open the file *caesarian_student_code.py*. Download the caesarian.csv dataset and report the total number of samples and the proportion of samples in each class (caesarian or not caesarian) for the whole dataset.

3.2 Bayesian classification on the continuous features
In this part, we are going to consider Age and Delivery number features only. Train a Gaussian naïve Bayes classifier from the training set (cf. *fit()*) and predict the labels of the data of the testing set (cf. *predict()*). Use now the *CategoricalNB()* model which is convenient for categorical features. Compute the classification accuracy on the training set

and on the testing set. The accuracy is defined as the proportion of good answers. Comment the results.

3.3 Bayesian classification on the categorical features

In this part, we are going to consider the other features only. Train a Gaussian naïve Bayes classifier from the training set (cf. *fit()*) and predict the labels of the data of the testing set (cf. *predict()*). Use now the *CategoricalNB()* model which is convenient for categorical features. Compute the classification accuracy on the training set and on the testing set. The accuracy is defined as the proportion of good answers. Comment the results.

3.4 Propose a way to define a model taking all the features into account. Or define a way to combine both previous results