UFJF - Machine Learning Toolkit

Release 0.51.1-beta.8

Jul 05, 2022

Table of contents

UFJF-MLTK is a cross-platform framework written in the C++ language for the development and usage of machine learning algorithms, addresses several types of learning problems such as classification, regression, clustering, feature selection, and ensemble learning. It aims to provide an always growing set of algorithms and tools for machine learning researchers and enthusiasts in its projects.

API Reference

You can find the API Reference at our repository Gihub Pages.

Cite us

If you use our project in your research, you can cite us by adding the bibtex from the project paper:

```
@inproceedings{10.1145/3330204.3330273,
   author = {Marim, Mateus Coutinho and de Oliveira, Alessandreia Marta and Villela,␣
↪Saulo Moraes},
   title = {UFJF-MLTK: A Framework for Machine Learning Algorithms},
   year = {2019},
   isbn = {9781450372374},
   publisher = {Association for Computing Machinery},
   address = {New York, NY, USA},
   url = {https://doi.org/10.1145/3330204.3330273},
   doi = {10.1145/3330204.3330273},
   booktitle = {Proceedings of the XV Brazilian Symposium on Information Systems},
   articleno = {63},
   numpages = {8},
   keywords = {object-oriented programming, machine learning, Framework},
   location = {Aracaju, Brazil},
   series = {SBSI'19}
}
```

Authors

- Mateus Coutinho Marim
- Saulo Moraes Villela
- Alessandreia Oliveira

Contact

Feel free to contact me at any time to clear doubts that you could have and if you want to contribute to the development of the framework. You can contact me at my e-mail address mateus.marim@ice.ufjf.br.

CHAPTER 1

Overview

1.1 Modules

1.2 Architecture

CHAPTER 2

Installation

As one of the main pourposes of UFJF-MLTK is the easy of use, the compilation and installation can't be different, for that the project was made using the cross-platform build management tool `cmake` as most of the known C++ open source projects.

2.1 Requirements

- CMake
- C++ compiler with support to C++17
- Gnuplot >= 5 (Optional, but needed for the Visualization module)

2.2 Build on any system

The project can be compiled using the same commands in any system, the only difference is that on `Windows` you'll need to make sure that the folder containing UFJF-MLTK is in your include path, so you can use include statements as `#include <ufjfmltk/Core.hpp>`. For the standart instalation you only need to execute the following commands on the project folder:

```
cmake -B build
cmake --build build
```

`CMake` can generate projects for several IDEs, if you have more than one C++ IDE in your operational system you can especify which one you want to use by adding the flag -G to CMake, for example, if you want to configure the project for Visual Studio, you could execute the command as `cmake -B build -G "Visual Studio 16 2019"` and then open the generated project on it.

UFJF-MLTK was projected to be as modular as possible, so if you don't want to compile some module, you could just turn off it's configuration on cmake, keeping in mind that it would be compiled in the same way if it's a dependency for another module to be compiled. The available options to be set on cmake are listed below:

CMake option	Default value	Description
-DBUILD_LIBVISUALIZE	ON	Tells if the visualization module must be built
-DBUILD_LIBCLASSIFIER	ON	Tells if the classifier module must be built
-DBUILD_LIBREGRESSOR	ON	Tells if the regressor module must be built
-DBUILD_LIBCLUSTERER	ON	Tells if the clusterer module must be built
-DBUILD_LIBFEATSELECT	ON	Tells if the feature selection module must be built
-DBUILD_LIBVALIDATION	ON	Tells if the validation module must be built

2.3 Including to your CMake project

Following are minimal scripts to include ufjfmltk to your CMake project. The first method is by simply cloning ufjfmltk repository into the main project folder and include it with `add_subdirectory`, it's a good method if you wish to use the latest updates on the framework, but it may break your application in future updates.

```
cmake_minimum_required(VERSION 3.15)
project(project_name)
set(BUILD_SHARED_LIBS OFF)

set(CMAKE_CXX_STANDARD 17)

# mltk is a folder with ufjfmltk project
add_subdirectory(mltk)
add_executable(project_name main.cpp)
target_link_libraries(project_name ufjfmltk)
```

The second and most recommended method is by using `FetchContent`, with this approach you need to select one of the releases on the repository and copy the link to its code `tar.gz` file, this way you garantee that your project will work even when the framework receive major updates.

```
cmake_minimum_required(VERSION 3.15)
project(project_name)
set(CMAKE_CXX_STANDARD 17)

include(FetchContent)
FetchContent_Declare(
        ufjfmltk
        # Specify the commit you depend on and update it regularly.
        URL https://github.com/mateus558/UFJF-Machine-Learning-Toolkit/archive/refs/
↪tags/v0.52.7-beta.tar.gz
)
FetchContent_MakeAvailable(ufjfmltk)

add_executable(project_name main.cpp)
target_link_libraries(project_name ufjfmltk)
```

2.4 Adding UFJF-MLTK libraries to Windows environment

You need to enter into "System properties" and the environment variable CPATH with the value pointing to the folder containing the binaries, the default folder is "C:/UFJF-MLTK/bin".

2.5 Compiling your project including UFJF-MLTK

With the libraries compiled and installed on the system you only need to add the UFJF-MLTK flag to the compiler to link the libraries to your program. Supose that we want to compile a source called `foo.cpp` containing a main function, to compile it on the command line, you just need to add the flag `-lufjfmltk`, for example, `g++ foo.cpp -o foo -lufjfmltk` and on Windows `g++ foo.cpp -o foo -L<install_folder> -lufjfmltk`.

Unix systems: `g++ foo.cpp -o foo -lufjfmltk`

Windows: `g++ foo.cpp -o foo -L<install_folder> -lufjfmltk`

With these steps complete, everything is set up and ready to use!

2.6 Going through installers

To make the framework installation easier for whom only whants to use the framework API, at each release are generated installers that installs the framework and make it available to all system. You can find all releases here.

2.6.1 Ubuntu and Debian based OS

Download the `.deb` file corresponding to the desired framework release and execute the following command.

```
sudo dpkg -i ufjfmltk-<version>-Linux-<cpu_architecture>.deb
```

2.6.2 Windows

2.6.3 Other linux based OS

Download the `.run` file corresponding to the desired framework release and follow these steps.

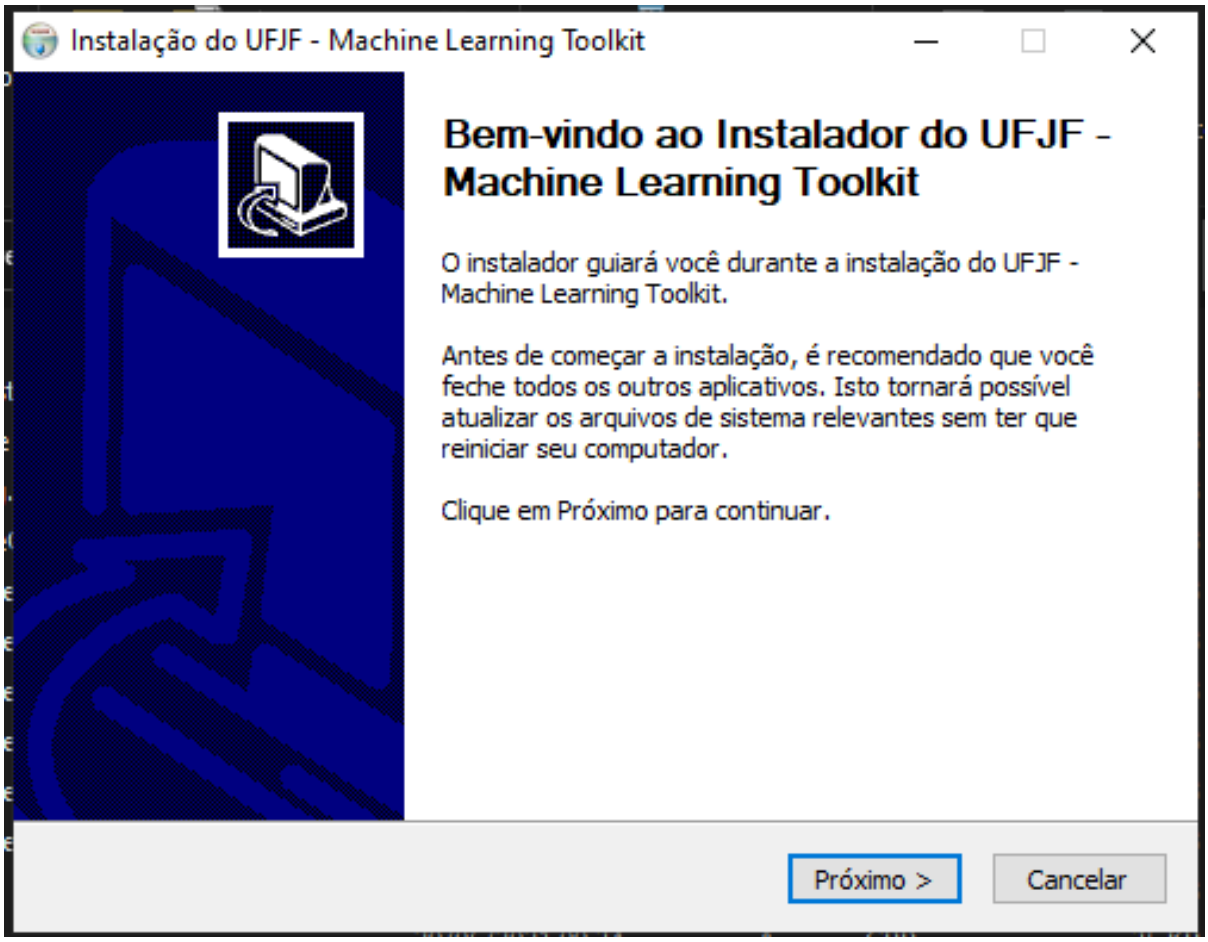

Fig. 2.1: 1 - Click **Next** button.

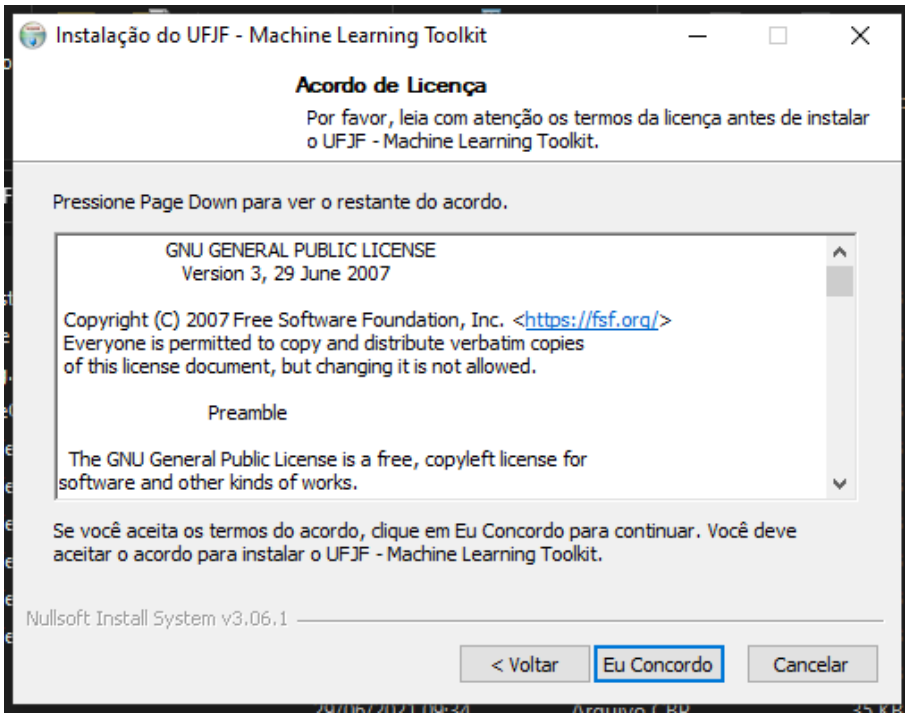

Fig. 2.2: 2 - Click **I agree** button.

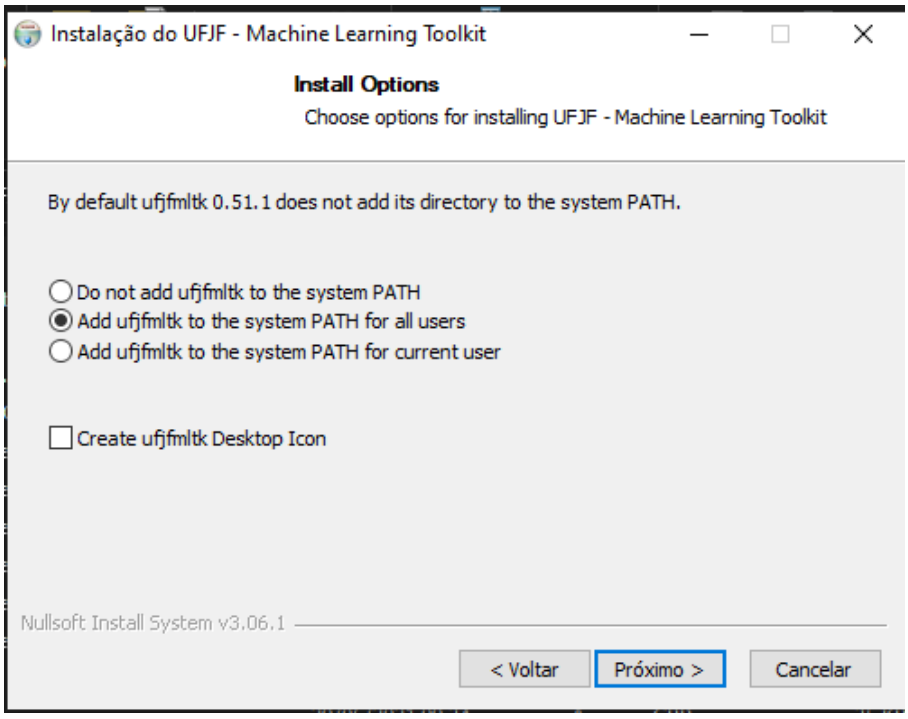

Fig. 2.3: 3 - Add ufjfmltk to system PATH so it'll be available to all system and click **Next**.

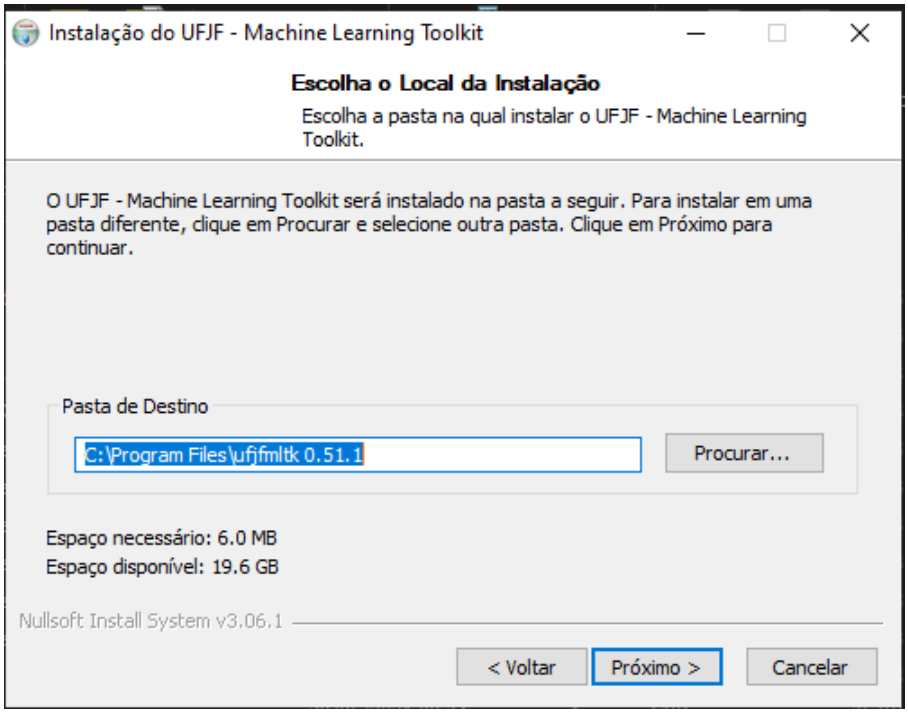

Fig. 2.4: 4 - Click **Next** button.

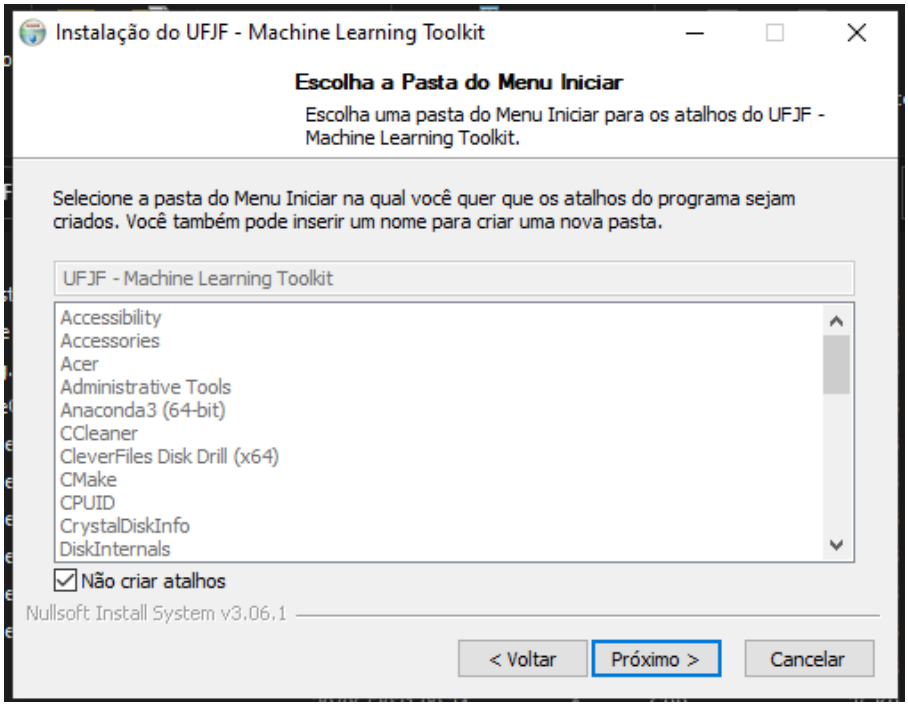

Fig. 2.5: 5 - Check **Don't create shortcuts** and click **Next** button.

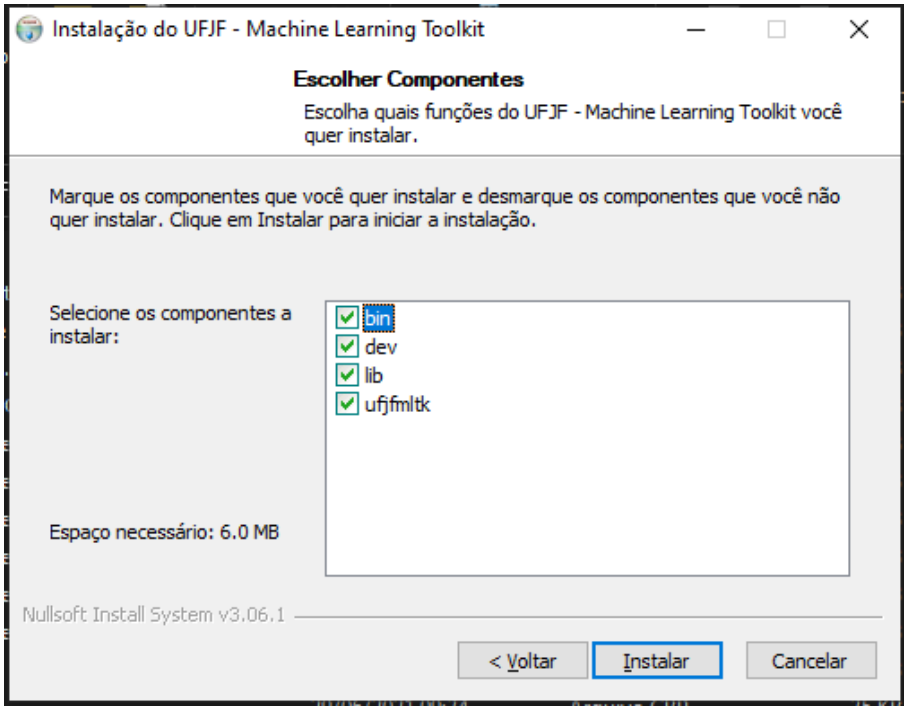

Fig. 2.6: 6 - Click **Install** button.

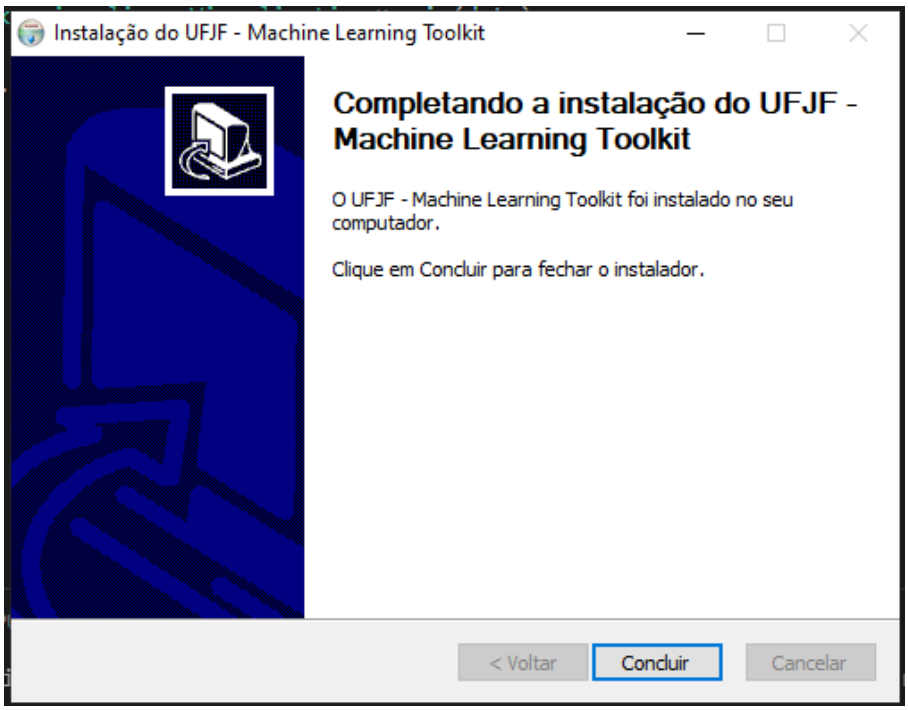

Fig. 2.7: 7 - Click **Finish** button.

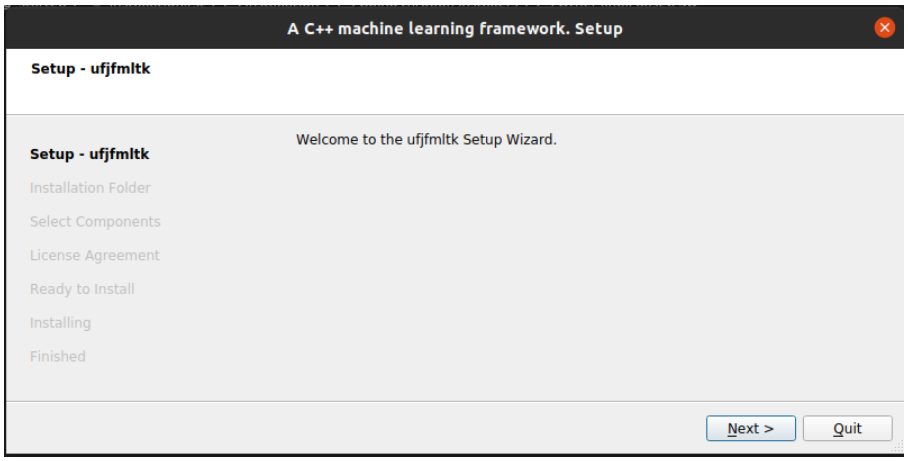

Fig. 2.8: 1 - Click **Next** button.

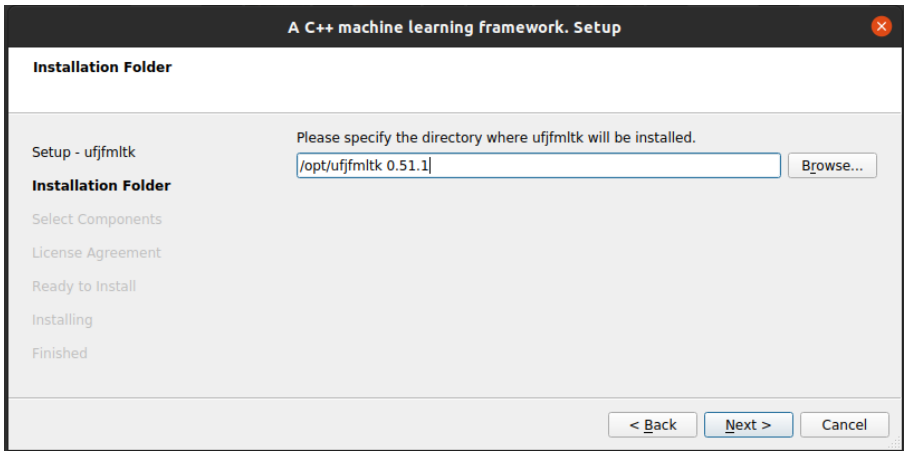

Fig. 2.9: 2- Choose where do you want to install ufjfmltk.

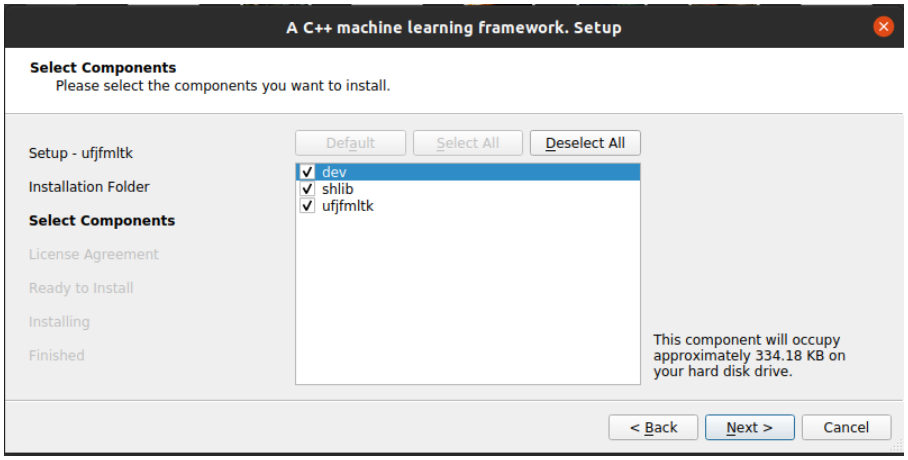

Fig. 2.10: 3 - Click **Next** button.

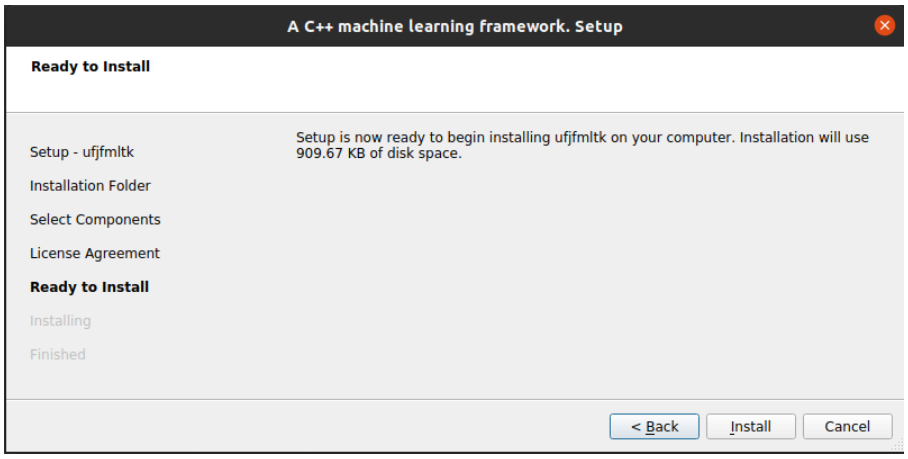

Fig. 2.11: 4 - Click **Install** button.

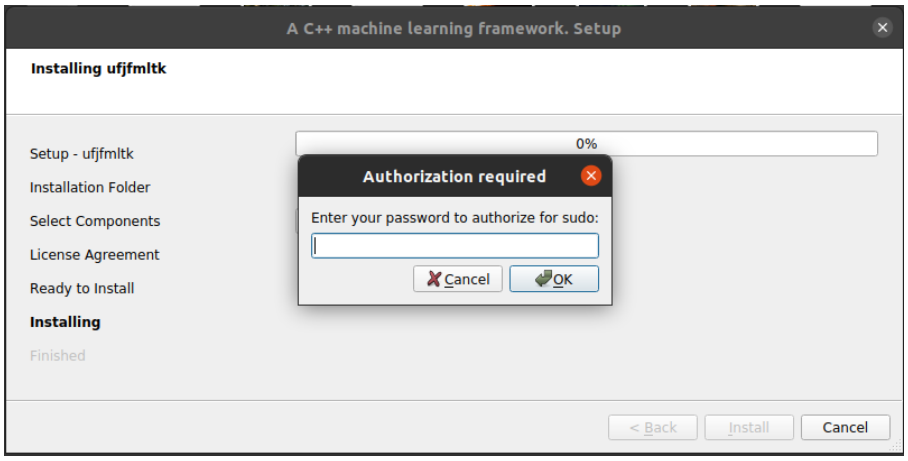

Fig. 2.12: 5 - If you had choosen to install the framework in a system folder, you need to provide your `sudo` password.

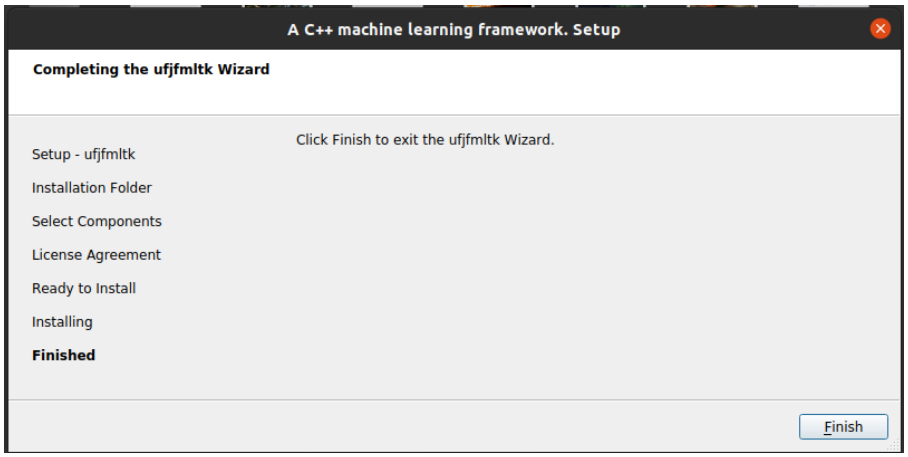

Fig. 2.13: 6 - Click **Finish** button.

CHAPTER 3

Data management

All the framework is composed of templates that make the development and use of ML (Machine Learning) algorithms easy and allow to the user to select the best available data type for the points of the data features.

One of the main concerns on ufjfmltk is to make ML algorithms development and usage simple, to accomplish that its necessary for the framework to have algorithms that can be applied on the underlying data structures, one problem with that is that the user needs to get familiar with the available methods which can increase the framework learning curve. To solve that problem, data structures like the Data and Point templates are implemented trying to keep compatibility with C++ STL library whethever is possible. With that, there's no need to implement a, for example, sorting algorithm when there's already the STL implementation for sorting operations.

In all the examples we'll be using the double data type as default, but you can substitute by any type that makes sense for your problem.

3.1 The Point template

This class is a wrapper for the n-dimension variables of the dataset in the feature space, it also includes along with the features values **X**, the target function **Y** and the _alpha_ weight associated to each point used in dual versions of ML algorithms.

In this section we'll be learning the basics of a point manipulation and the operations that can be aplied to it, at the end I'll also show point expressions, a feature that can make the code easier to read and less error prone than the raw usage of loops.

3.1.1 Initialization

There are several ways to instantiate the Point template, the most simple instatiation is to just initialize the point with *n* positions filled with zeros. The other properties can be initialized with its own methods.

```
int n = 3,
mltk::Point<double> p(n);
```

(continues on next page)

(continued from previous page)

```
p.Y() = 1;
p.Alpha() = 0;
p.Id() = 5;
```

A more advanced initialization would be, for example, filling it with random values following a uniform distribution, something that can be accomplished calling the `random_init` method.

```
mltk::Point<> p1;

mltk::random_init(p1, 3, 42);
```

In the code snipet above we initialized *p1* with 3 dimensions and values following a uniform random distribution with seed 42. Bellow we can see another example using a different distribution.

```
mltk::Point<double> p1;

p1 = mltk::random_init<double, std::fisher_f_distribution<double>>(2.0, 2.0, 3, 42);
```

3.1.2 Point expressions and operations

The main advantages of the Point template is the possibility to write code similar to math equations, taking away the burden of writting complex loops to implement functions. Point expressions are made possible through the implementation of a expression template, it makes possible to overload the math equations with similar sintax as the paper written version without losing performance when compiler optimizations are turned on (`-O3`). Even on debug mode, the performance loss is acceptable given that the code is less error prone and easier to read. One example of application is on the implementation of distance metrics like the euclidean distance.

Euclidean distance:

$$d = \sqrt{\sum_{i=0}^{n-1}(p1_i - p2_i)^2}$$

```
mltk::Point<> p1(3, 2), p2(3, 1.5);

double d = std::sqrt(mltk::pow(p1 - p2, 2).sum()); // Euclidean distance between p1␣
↪and p2
```

Another advantage is that the equation is lazy evaluated, i.e only evaluated when it's result is required, like when doing a point assignment. This becomes an advantage on multithreaded environments, when we want to evaluate an equation when a thread requires it.

Alongside with the Point template there are some operations that can be applied on the points, they are the basic arithmetic operations and some common math functions.

Examples of arithmetic operations:

```
mltk::Point<> a(3, 1.0), b(3, 2.0);
mltk::Point<> c = a + b;

b = a - c;
a = c / 2;
c *= 3;
```

Also were implemented on the framework common math functions that can be aplied directly to the point objects.

```
mltk::Point<> a(3, 1.0), b(3, 2.0);
mltk::Point<> c = mltk::sin(a) + 2 * mltk::cos(b);

auto d = mltk::pow(mltk::exp(c), 3);
double sum = d.sum();
```

This example above only shows a subset of operations that can be applied, for more you can see the list below.

- abs `mltk::abs(p)` - absolute values;
- max `mltk::max(p)` - maximum value;
- min `mltk::min(p)` - minimum value;
- norm `mltk::norm(p, norm_type)` - computes the norm of a point, by default `norm_value = 2`;
- dot `mltk::dot(p, q)` - computes the dot product between *p* and *q*;
- log `mltk::log(p)` - natural log values;
- normalize `mltk::normalize(p, norm_type)` - normalize a point, by default `norm_value = 2`;
- linspace `mltk::linspace(lower, upper, N)` - returns a point with *N* linear values from *lower* to *upper*.

We can also print the point content using the stream overload operator.

```
std::cout << p << std::endl;
```

You can access the features values of a point accessing the elemens of the **x** vector member or by treating the point as a container:

```
int i, dim = p.x.size();

for(i = 0; i < dim; i++){
    std::cout << p[i] << std::endl;
}

// using iterators
for(auto it = p.begin(); it != p.end(); it++){
    std::cout << (*it) << std::endl;
}
```

3.1.3 Algorithms

As we are keeping the compatibility with STL, there are several algorithms that are supported by the framework, for example if we want to fill a Point with integers we can use the `std::iota` algorithm for that, like standart C++ containers.

```
mltk::Point<> p(5);

std::iota(p.begin(), p.end(), 1);
```

This code will fill *p* with values ranging from 1 to 5.

We also could initialize a point with random values ranging from 1 to 10 and sort it after.

```
mltk::Point<int> p(5);

p = mltk::random_init<int, std::uniform_int_distribution<int>>(1, 10, 5, 42);
std::sort(p.begin(), p.end());
```

3.2 The Data template

As we're normally dealing with datasets we have multiple points to work, so there's the necessity to have a class to wrap all the information about this dataset and the operations that we can apply to these data. As the Point the Data template is also compatible with STL algorithms.

These are the supported formats to load datasets:

- arff
- csv
- data
- txt (Embrapa datasets format)

3.2.1 Memory sharing between Data objects

Sometimes we need to run several algorithms in the same dataset and, if we'll not transform the feature space of the variables, copying all the data to each algorithm that we'll run can be a waste of memory and at sometimes a simple computer can't handle the memory consumption. Thinking in that the Data class was developed using smart pointers, a tool introduced at C++11 that handles the sharing of memory between objects with almost the same speed of raw pointers, but memory safe.

Because of that an array of points in the data class is defined with T as a generic data type as:

```
std::vector<std::shared_ptr<Point< T > > > points;
```

So if you use the = operator with other data object, they will be point to the same memory space of the original object, to make a deep copy the content of an object to another you'll have to use the `copy()` method.

```
Data<double> other;

other = data.copy()
```

3.2.2 Loading a dataset to a Data object

This can be easily done with the Data class initialization, accomplished with only one line of code.

```
Data<double> data("wine.csv");
```

Or if you want the data object initially empty.

```
Data<double> data;

data.load("wine.csv");
```

If the target function value or expected value is at the end of the dataset, it must be informed to the constructor.

```
Data<double> data("wine.arff", true);
```

Note that in all formats the target function must be at the beginning or at the end of each line of the file. You can print all the dataset with the C++ standard output stream operator.

```
Data<double> data("wine.csv");

std::cout << data << std::endl;
```

3.2.3 Getting information about the dataset

After the data is loaded into the memory, we can get some useful information about the data.

```
std::cout << "Dataset information: " << std::endl;
std::cout << "Number of points: " << data.size() << std::endl;
std::cout << "Dimension: " << data.dim() << std::endl;
std::cout << "Classes: " << data.classes() << std::endl;
std::cout << "Classes distribution: " << data.classesDistribution() << std::endl;
```

3.2.4 Scanning through the data points

There are two ways to access the points contained on a Data object, the first is the operator `[]` that returns a smart pointer to a point contained in the Data object, the other way is through the function call operator `()` that returns a reference to the Point object. Almost all the times we would want to use the second option to avoid the pointer sintax.

Though the smart pointers are intended to be preferred in the place of the raw pointers, they work almost the same way as we are used with the classic pointers, so there's no much difference in this.

In this example we'll see how we can print each point of the dataset:

```
int i, j, size = data.size(), dim = data.dim();

for(i = 0; i < size; i++){
    std::cout << data(i) << std::endl;
}
```

Treating the Data object as a container:

```
for(i = 0; i < size; i++){
    for(j = 0; j < dim; j++)
        std::cout << data(i)[j] << std::endl;
}
```

3.2.5 Applying transformations to data

Often we dont want only to load the data but also want to apply transformations to it, be it a point/feature removal or a preprocessing step. For it the Data template provide methods for point and features removal/insertion and the method `apply` that allows to apply a function to the points contained in the object.

We could normalize the dataset points like this, instead of looping through the points:

```
mltk::Data<> data("iris.csv");

auto normalization = [](mltk::PointPointer<double> point){
    *point = mltk::normalize(*point, 2);
};

data.apply(normalization);
```

Initializing a data object with 10 uniform distributed random points:

```
mltk::Data<> data;
for(int i = 0; i < 10; i++){
    auto p  = mltk::random_init(3, 42);

    data.insertPoint(p);
}
```

Below are the methods of insertion/removal:

- insertPoint - insert a point to the dataset;
- removePoint - remove a point with the given unique id;
- removeFeature - remove a dimension with the given id (1..dim) from the dataset.

You can see the concepts presented here in practice on the implementation of algorithms for artificial datasets generation.

CHAPTER 4

Classification

Often we are given the task, from ourselves or from others, to label things according to a set of already existing classes:

- Is the object in the image a vehicle or a cat?
- Is this animal a dog or a cat?

Classification is the problem of giving the right label to a record given as input. The task is different from regression because here we have discrete labels instead of continuous values [SKIENA2017]. In this chapter we'll give a brief introduction on binary and multi-class classification tasks and show how to tackle these problems using **UFJF-MLTK**.

Add `#include <ufjfmltk/Classification.hpp>` to include the classification module.

4.1 Binary classification

Let $Z = (x_i, y_i)$ be a set of samples of size m, where $x_i \in R^d$, called input space of the problem, y_i is a scalar representing the class of each vector x_i and for binary classification $y_i \in \{+1, -1\}$, for $i = \{1, \ldots, m\}$. A linear classifier, in a linearly separable input space, is represented by a hyperplane with the following equation [VILLELA2011]:

$$h(x) = \langle w, x \rangle + b$$

The classification result can be obtained through a signal function φ applied to the discriminant value associated to the hyperplane equation, i.e:

$$\varphi(h(x)) = \begin{cases} +1, & \text{if } h(x) \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

4.1.1 The Perceptron algorithm

Considered the first learning algorithm, the Perceptron model is a pattern recognition model proposed by [ROSENBLATT1958]. It's structured by a input layer connecting each input unit to a component from a d-dimension

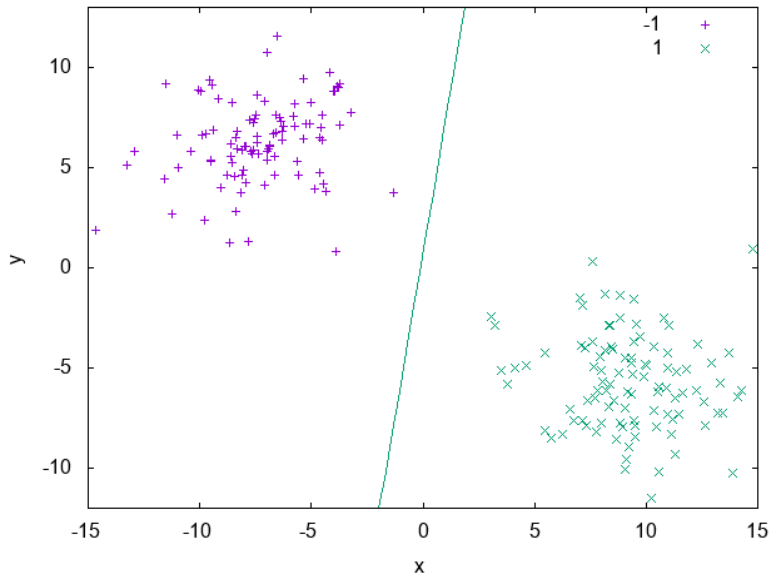

Fig. 4.1: Example of a binary classification problem with a linear discriminant.

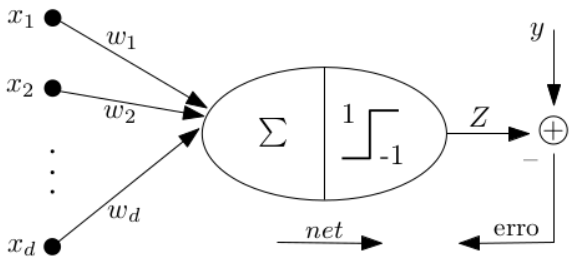

Fig. 4.2: Perceptron model topology.

vector, and a output layer composed of m units. Therefore, it's an artificial neural network model with only one processing layer. In its simplest form, the Perceptron algorithm is a classification algorithm involving only two classes [VILLELA2011].

The algorithm developed by Rosenblatt can be utilized to determine the w vector in a limited number of iterations, where the number of iterations is related to the number of updates of the weights vector. As the weights vector w is determined by successive corrections in order to minimize a loss function, we can say that the separating hyperplane is constructed in a iterative way characterizing an *online* learning process [VILLELA2011].

Listing 4.1: Primal Perceptron example

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace vis = mltk::visualize;
namespace classifier = mltk::classifier;

int main() {
    mltk::Data<double> data("iris.data");
    vis::Visualization<> vis(data);
    classifier::PerceptronPrimal<double> perceptron(data);

    perceptron.train();

    vis.plot2DwithHyperplane(1, 2, perceptron.getSolution(), true);
}
```

On Listing 4.1 we can see a simple usage of the **UFJF-MLTK** perceptron implementation in it's primal form. In this example we first load the binary `iris.data` dataset where two of the three original classes were merged into one in order to generate a binary problem, after that we instantiate the `PerceptronPrimal` wrapper with the same data type as the dataset and the default parameters. With the object from the algorithm wrapper we call the method `train` to learn a model from the data and, finally, the decision boundary is ploted with features 1 and 2 from the dataset and passing the perceptron solution. Fig. 4.3 shows the hyperplane generated by the model.

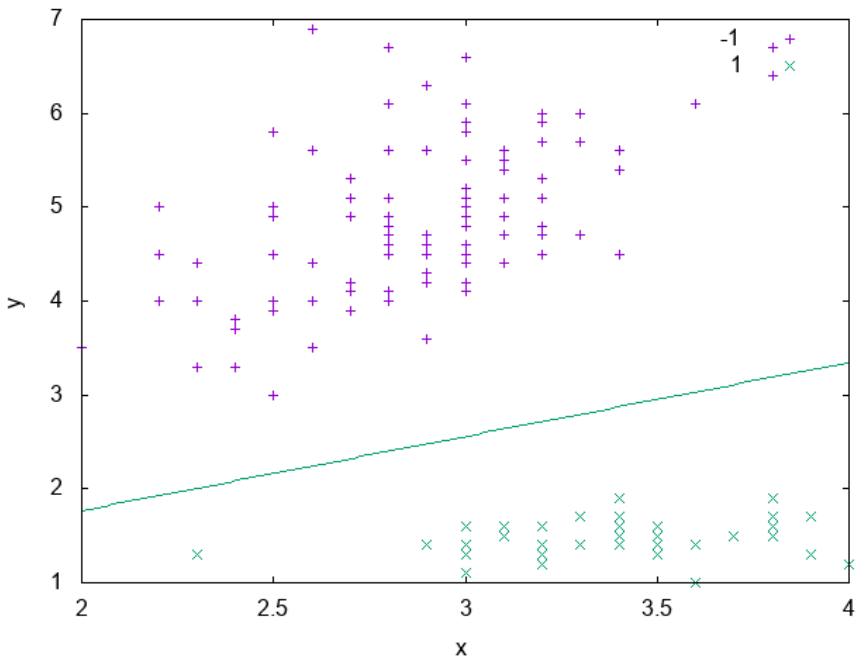

Fig. 4.3: Solution generated from the model trained by the Perceptron classifier.

4.2 Kernel methods

Often in real datasets is not possible to do a linear separation of the data. In these cases is necessary to utilize more complex functions for labels separation. One way to define a non-linear separator is through a mapping function from input space X to a higher dimensional space where the separation is possible [MEHRYAR2018].

In models based on a mapping from the fixed non-linear features space $\Phi(x)$, the kernel function is defined as following [BISHOP2007]:

$$k(x, x^{'}) = \Phi(x)^{T}\Phi(x^{'}) \tag{4.1}$$

Fig. 4.4 shows an example of a dataset that isn't linearly separable. It's composed of two spirals and as we can see, there isn't a way to draw a line that separates the samples belonging to each spiral. In the *Dual Perceptron* section we'll see how to solve this problem.

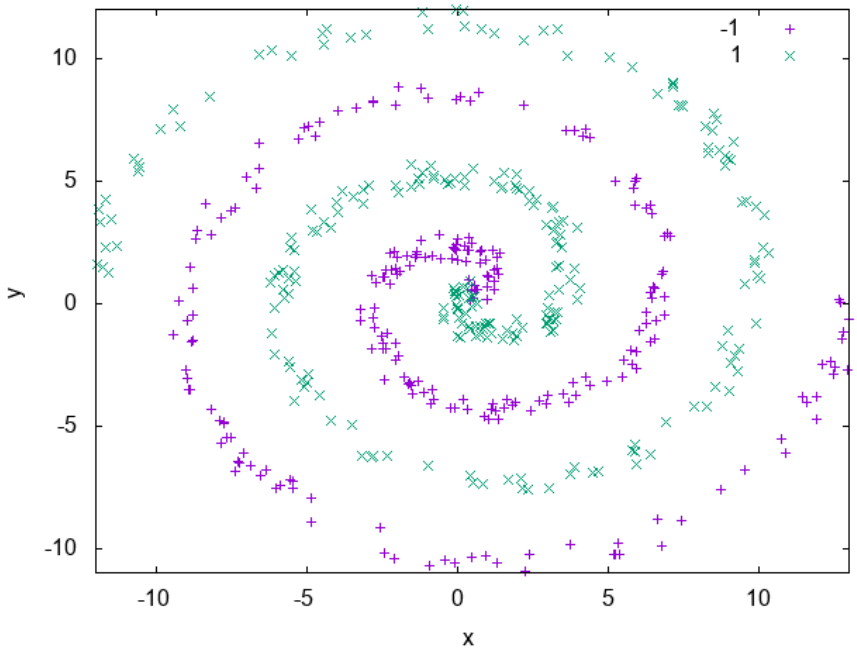

Fig. 4.4: Spirals artificial dataset.

The simplest kernel considering the mapping on Eq. (4.1) is the linear kernel where $\Phi(x) = x$ and $k(x, x^{'}) = x^{T}x$. The kernel concept formulated as a inner product in the input space allows the generalization of many known algorithms. The main idea is that if an algorithm is formulated in such a way that the input vector x is presented in a scalar product form, the inner product can be replaced by another kernel product. This kind of extension is known as **kernel trick** or kernel substitution [BISHOP2007].

4.2.1 The Perceptron dual algorithm

The derivation and implementation of the dual form of the Perceptron algorithm will be shown in Section ??, since it's a more complex topic. For now, we'll use **UFJF-MLTK** implementation to solve the spirals dataset problem presented earlier.

Listing 4.2: Dual perceptron training on spirals artificial dataset.

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace vis = mltk::visualize;
namespace classifier = mltk::classifier;

int main() {
    auto data = mltk::datasets::make_spirals(500);
    vis::Visualization<> vis(data);
    classifier::PerceptronDual<double> perceptron(data, mltk::KernelType::GAUSSIAN,␣
↪1);

    perceptron.setMaxTime(500);
    perceptron.train();

    vis.plotDecisionSurface2D(perceptron, 0, 1, true, 100);
}
```

Listing 4.2 example generates a spirals dataset with 500 samples using the `make_spirals` function from `mltk::datasets::` namespace, initialize the visualization object and instantiate the `PerceptronDual` wrapper with a gaussian kernel with standard deviation of 1.0 as a kernel parameter. To guarantee the algorithm convergence, the maximum training time of the algorithm is set as 500ms, after that, the model is trained and its decision boundary is ploted as in Fig. 4.5.

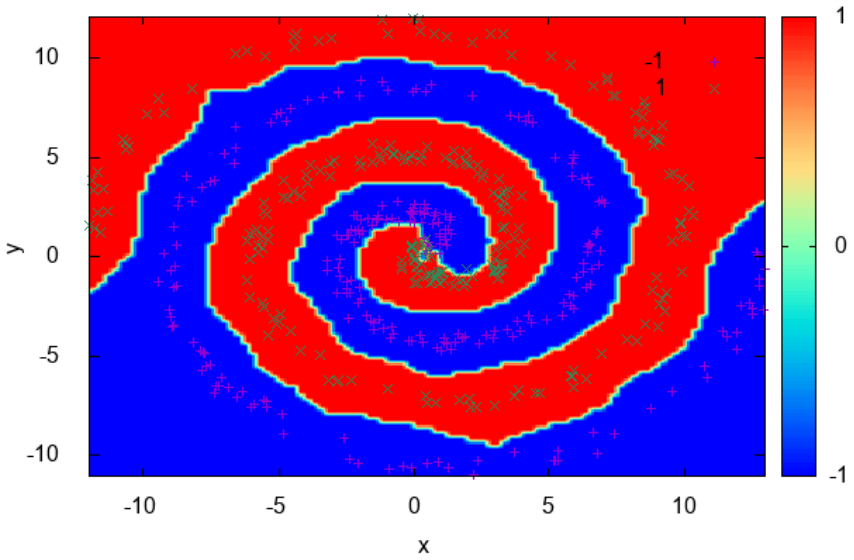

Fig. 4.5: Decision contour surface from Perceptron dual for spirals dataset.

4.3 Multi-class classification

Until now we've been discussing algorithms for classification problems were we have only two labels, but often we face problems where we need to choose a class between tens, hundreds or even thousands of labels, like when we need to assign a label to an object in an image. In this chapter, we'll be analysing the problem of multi-class classification learning.

Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ the output space, and let $\mathcal{D}$ be an unknown distribution over $\mathcal{X}$ according to which input points are drawn. We'll be distinguishing between the *mono-label* (binary classification) and *multi-label* cases, where we define $\mathcal{Y}$ as a set of discrete values as $\mathcal{Y} = \{1, \ldots, k\}$ and $\mathcal{Y} = \{+1, -1\}^k$ for the *mono-label* and *multi-label* cases, respectively. In the *mono-label* case, each sample will be assigned to only one class, while in the *multi-label* there can be several. The latter can be illustrated as the positive value being the component of a vector representing the classes where the example is associated [MEHRYAR2018].

On both cases, the learner receives labeled samples $\mathcal{S} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X}, \mathcal{Y})^m$ with $x_1, \ldots, x_m$ drawn according to $\mathcal{D}$, and $y_i = f(x_i)$ for all $i \in [1, \ldots, m]$, where $f : \mathcal{X} \to \mathcal{Y}$ is the target labeling function. The multi-class classification problem consists of using labeled data $\mathcal{S}$ to find a hypothesis $h \in H$, where H is a hypothesis set containing functions mapping $\mathcal{X}$ to $\mathcal{Y}$. The multi-class classification problem consists on finding the hypothesis $h \in H$ using the labeled data $\mathcal{S}$, such that it has smallest generalization error $R(h)$ with respect to the target f, where Eq. (4.2) refers to the *mono-label* case and Eq. (4.3) to the *multi-label* case [MEHRYAR2018].

$$R(h) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}[1_{h(x) \neq f(x)}] \tag{4.2}$$

$$R(h) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}[\sum_{l=1}^{k} 1_{[h(x)]_l \neq [f(x)]_l}] \tag{4.3}$$

In the following sections we'll be discussing two algorithms for adapting models for binary classification to the multi-class case, namely One-vs-All and One-vs-One. For that, the blobs artificial dataset generated with 50 examples for each of 3 labels. The plot for the dataset data can be seen on Fig. 4.6.

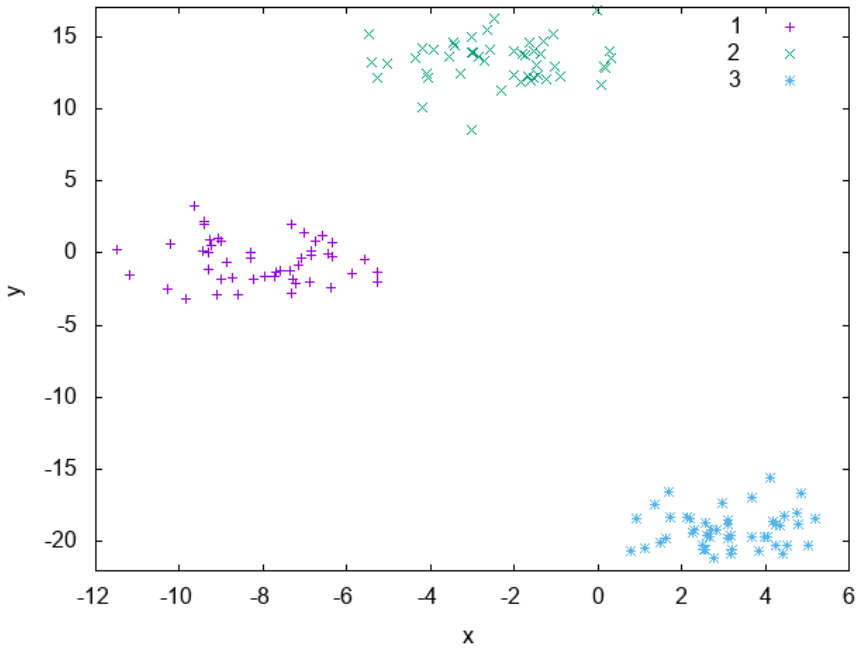

Fig. 4.6: Blobs artificial dataset.

4.3.1 The One-vs-All algorithm

This method consists in learning k binary classifiers $h_l : \mathcal{X} \rightarrow -1, +1,\ l \in \mathcal{Y}$, each one of them designed to discriminate one class from all the others. Each h_l, for any $l \in \mathcal{Y}$, is constructed by training a binary classifier after relabeling points in class l with 1 and all the others as -1 on the full sample $\mathcal{S}$. The multi-class hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$ defined by the One-vs-All (OVA) technique is given by [MEHRYAR2018]:

$$\forall x \in \mathcal{X},\ h(x) = \underset{l \in \mathcal{Y}}{arg\,max}\ f_l(x)$$

Listing 4.3 shows how to use the **UFJF-MLTK** primal perceptron implementation with the OVA technique to tackle the blobs dataset classification problem. As can be seen, the only thing needed to do is to instantiate the `OneVsAll` wrapper and pass the training data and the algorithm wrapper to be used. Something to be noted, is that the base algorithm parameters must be passed on its initialization or before calling the OVA `train` method.

Listing 4.3: OVA example with the primal perceptron model.

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace vis = mltk::visualize;
namespace classifier = mltk::classifier;

int main() {
    auto data = mltk::datasets::make_blobs(50, 3, 2, 1.5, -20, 20, true, true, 10).
↪dataset;
    vis::Visualization<> vis(data);
    classifier::PerceptronPrimal<double> perceptron;
    classifier::OneVsAll<double> ova(data, perceptron);

    ova.train();

    vis.plotDecisionSurface2D(ova, 0, 1, true, 100, true);
}
```

Fig. 4.7 shows the decision boundary generated after training, it's possible to note that each region drawn accomodates points with the same class, indicating that the technique was effective on learning a aproximation of the data distribution. For non linearly separated data, the only changes is that we need to use an algorithm capable of learning a non-linear function like the dual perceptron from `PerceptronDual` wrapper.

4.3.2 The One-vs-One algorithm

The One-vs-One (OVO) technique consists in learning a binary classifier $h_{ll'} : \mathcal{X} \rightarrow -1, +1$ for each pair of distinct classes $(l, l^{'}) \in \mathcal{Y}, l \neq l^{'}$, discriminating l and $l^{'}$. $h_{ll'}$ is obtained by training a binary classifier on the sub-sample containing exactly the points labeled as l and $l^{'}$, with the value +1 returned for $l^{'}$ and -1 for l. For that, it's needed to train $\binom{k}{2} = \frac{k(k-1)}{2}$ classifiers, which are combined to define a multi-class classification hypothesis h via majority vote [MEHRYAR2018]:

$$\forall x \in \mathcal{X},\ h(x) = \underset{l' \in \mathcal{Y}}{arg\,max}\ |\{l : h_{ll'}(x) = 1\}|$$

Listing 4.4: OVO example with the primal perceptron model.

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace vis = mltk::visualize;
namespace classifier = mltk::classifier;
```

(continues on next page)

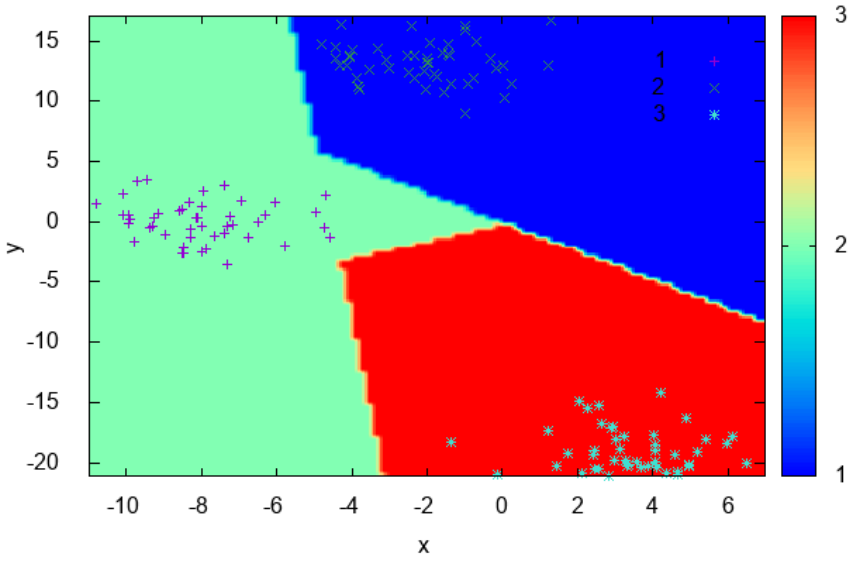

Fig. 4.7: Decision contour surface from OVA with perceptron for blobs dataset.

(continued from previous page)

```
  int main() {
      auto data = mltk::datasets::make_blobs(50, 3, 2, 1.5, -20, 20, true, true, 10).
↪dataset;
      vis::Visualization<> vis(data);
      classifier::PerceptronPrimal<double> perceptron;
      classifier::OneVsOne<double> ovo(data, perceptron);

      ovo.train();

      vis.plotDecisionSurface2D(ovo, 0, 1, true, 100, true);
  }
```

Listing 4.4 is analogous to Listing 4.3 except that it's using the `OneVsOne` wrapper instead of the OVA one. As expected, it could also learn the data distribution, this can be seen by the decision boundary shown at Fig. 4.8.

4.4 Model evaluation and selection

So far, you may have been able to build a classifier, but only that is not enough. Supose you've trained a model to predict the purchasing behavior of future clients using data from previous sales. For that, you need to estimate how accurately your model can be on unseen data, i.e, how accurately your model can predict the behavior of future customers. You may have built several classifiers and need to compare how well they can be between each other [HAN2011]. This section address metrics that can be used to compare those methods and how reliable this comparison can be.

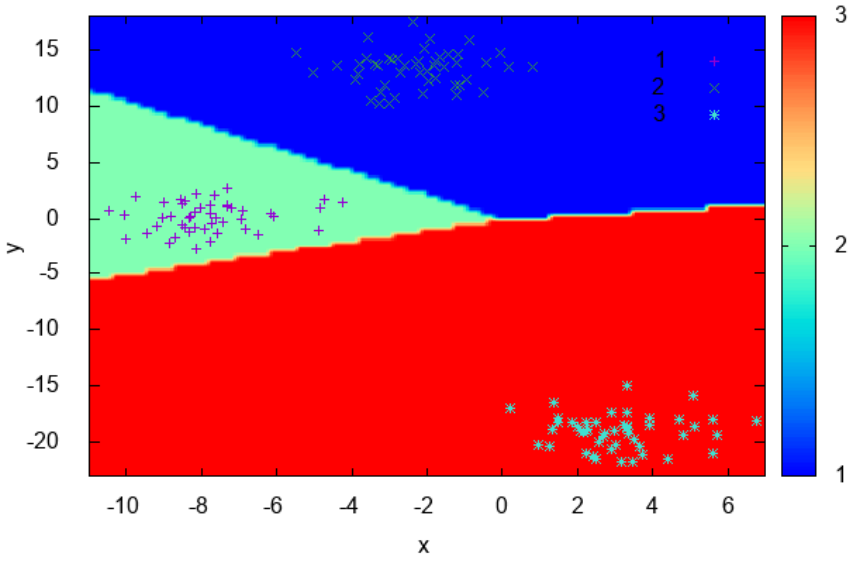

Fig. 4.8: Decision contour surface from OVO with perceptron for blobs dataset.

4.4.1 Metrics for classifiers evaluation

The usage of training data for accuracy estimation of a classifier, can lead to overoptimistic estimates due to overspecialization of the model to the data. A better option to avoid this issue is to measure the classifier accuracy using a *test set*, that is are examples from the entire dataset that weren't used during model training [HAN2011].

In this section we'll be discussing several metrics to measure a classifier performance, but before we need to be confortable with some terminologies that'll be used throghout metrics definitions. Two important terms are **positive samples**, points labeled with the class of main interest, and **negative samples**, that are the rest of the samples. Given two classes, for example, the positive samples may be *buy_computer = yes* and the negative samples *buy_computer = no*. Supose a classifier is used on a *test set* of labeled data. P is the number of positive samples and N is the number of negative samples. for each sample we compare For each sample, we compare the predictions made by the classifier with the sample known class. There are four other terms that must be understood since they are the building blocks of many evaluation measures computations [HAN2011].

- **True positives** (TP): positive samples that were correctly labeled by the classifier;
- **True negatives** (TN): negative samples that were correctly labeled by the classifier;
- **False positives** (FP): positive samples that were incorrectly labeled as negative;
- **False negatives** (FN): negative samples that were incorrectly labeled as positive.

A **confusion matrix** is a tool used to analyse if the classifier is doing well on prediction of examples of different classes. TP and TN indicates if the classifier is labeling right. FN and FP tells when the classifier is doing wrong predictions. These terms are sumarized on the confusion matrix from Table 4.1. It's a matrix at least of size *m* x *m* where *m* is the number of classes, an entry CM_{ij} represents the number of examples from class i that were labeled as j [HAN2011].

Table 4.1: Confusion matrix where the lines represent the **actual class** and the columns the **predicted class**.

	yes	*no*
yes	TP	FN
no	FP	TN
Total	P'	N'

Below is a list of important metrics for classifiers evaluation and selection:

- **Accuracy**: percentage of examples on the test set that were correctly classified.

$$accuracy = \frac{TP + TN}{P + N}$$

- **Error rate**: is $1 - accuracy(M)$ where $accuracy(M)$ is the accuracy of the classifier M. It can also be computed as follows:

$$error\ rate = \frac{FP + FN}{P + N}$$

- **Sensitivity** and **Specificity**: are the proportion of the positive samples that were correctly classified and the true negative proportion, respectively.

$$sensitivity = \frac{TP}{P}$$
$$specificity = \frac{TN}{N}$$

- **Precision**: can be thought as a measure of exactness, i.e, the percentage of examples labeled as positive are actually such.

$$precision = \frac{TP}{TP + FP}$$

- **Recall**: is a measure of completeness, i.e, the percentage of positive samples labeled as such, the same as *sensitivity*.

$$recall = \frac{TP}{TP + FN}$$

- **F-score**: can be viewed as a weighted average of the *precision* and *recall*. It reaches its best value at 1 and the worst at zero. Both precision and recall have the same contribution to the F-score.

$$F - score = \frac{2(precision * recall)}{precision + recall}$$

Listing 4.5: metrics computation for KNN classifier on spirals dataset.

```
#include <iostream>
#include <ufjfmltk/ufjfmltk.hpp>

namespace classifier = mltk::classifier;
namespace valid = mltk::validation;

int main() {
    auto data = mltk::datasets::make_spirals(500, 3, true, 2);
    valid::TrainTestPair traintest = valid::partTrainTest(data, 3);
    classifier::KNNClassifier<double> knn(traintest.train, 3);

    auto cfm = valid::generateConfusionMatrix(traintest.test, knn);
```

(continues on next page)

(continued from previous page)

```
13      valid::ValidationReport report = valid::metricsReport(traintest.test, cfm);
14
15      std::cout << "True positive = " << report.tp << std::endl;
16      std::cout << "True negative = " << report.tn << std::endl;
17      std::cout << "False positive = " << report.fp << std::endl;
18      std::cout << "False negative = " << report.fn << std::endl;
19      std::cout << "Errors = " << report.errors << std::endl;
20      std::cout << "Accuracy = " << report.accuracy*100.0 << std::endl;
21      std::cout << "Error = " << report.error*100.0 << std::endl;
22
23      mltk::utils::printConfusionMatrix(data.classes(), data.classesNames(), cfm);
24  }
```

Listing 4.5 shows how to compute some of the earlier mentionated metrics. On Lines 8-10 the program generates an instance of the spirals artificial dataset, split the data in training and test sets (see the next section) and create an object of the KNN classifier algorithm wrapper. After that, on Lines 12-13 the confusion matrix `cfm` is generated to be used by the `metricsReport` method, that evaluate the metrics of the model on the passed data and returns it as a `ValidationReport` object. From Line 15 to 21 some metrics are printed and, finally, the program prints the confusion matrix on the screen and exits.

4.4.2 Holdout method and random subsampling

With the **holdout** method the data is randomly partitioned in two independent sets, the *training set* and the *test set*. Usually, two thirds of the data is reserved for training and one third for testing. The training set is used to train the model and the test set for estimating the accuracy. The problem of this method is that it usually pessimistic because only a portion of the data is used to derive the model. The holdout accuracy estimation process is illustrated in Fig. 4.9 [HAN2011].

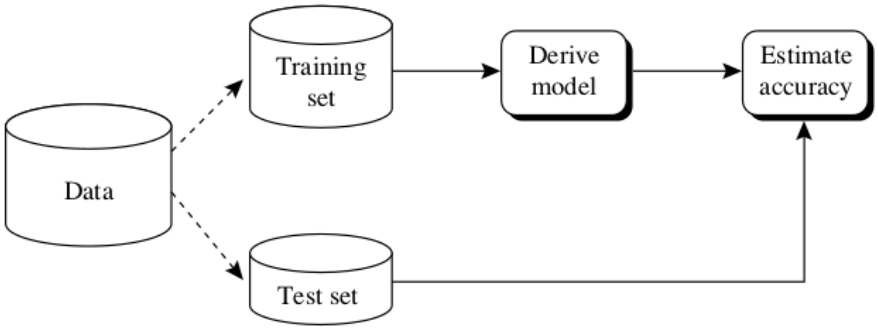

Fig. 4.9: Accuracy estimation with the holdout method.

Random subsampling is a variation of the holdout method where the holdout method is repeated k times. The accuracy estimate is given as the average of the accuracies obtained from each iteration [HAN2011].

Listing 4.6 loads the iris dataset for a multi-class classification task and splits the dataset in a training and test set pair. The data split is done dividing the data, in a stratified manner by default, in 3 folds and selects one of them for the test set and merge the other ones into a training set. After the data split, the OVA wrapper is instantiated with an object from the perceptron wrapper and the training set. Finally, the model is trained and the accuracy is estimated on the test set.

Listing 4.6: holdout accuracy estimation on iris dataset.

```
#include <iostream>
#include <ufjfmltk/ufjfmltk.hpp>

namespace classifier = mltk::classifier;
namespace valid = mltk::validation;

int main() {
    mltk::Data<> data("iris_mult.csv");
    valid::TrainTestPair traintest = valid::partTrainTest(data, 3);
    classifier::PerceptronPrimal<double> perc;
    classifier::OneVsAll<double> ova(traintest.train, perc);

    ova.train();

    std::cout << "Accuracy = " << valid::accuracy(traintest.test, ova) * 100.0 <<␣
↪std::endl;
}
```

4.4.3 Cross-validation

In **k-fold cross-validation** the data is partitioned in k mutually independent sets or *folds* $D_1, D_2, ..., D_k$ of approximately equal size, performing training and testing k times. On each iteration i the i-th partition is used as test set and the $k-1$ remaining partitions as training data. Unlike the holdout and random subsampling methods, here the folds are used for training k times and once for testing. For classification, the accuracy is given by the number of correct classifications divided by the total number of samples on the initial data [HAN2011].

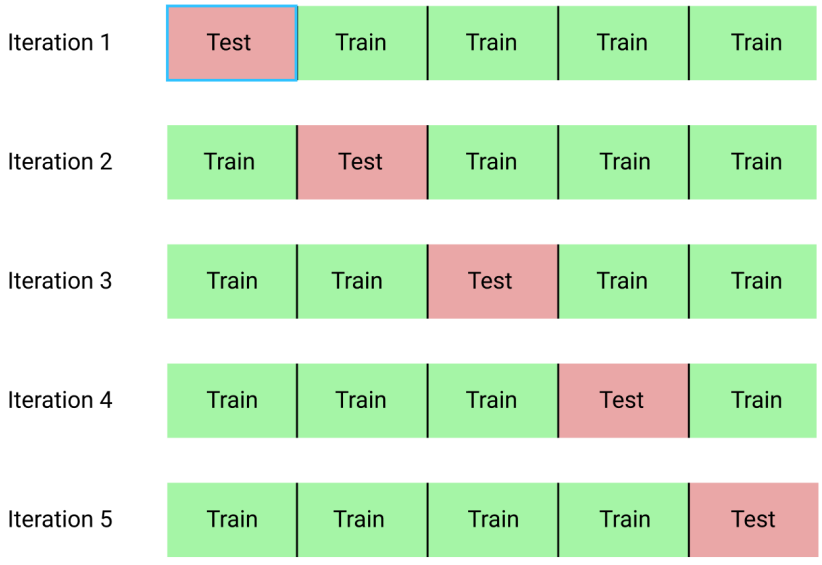

Fig. 4.10: k-fold cross validation process. Source: Towards data science

Leave-one-out is a special case from k-fold cross-validation where k is set as the total number of samples, i.e, only one sample is used as test set. In **stratified cross-validation** the folds are stratified so the distribution of the data is approximately equal to the initial data distribution. In general, 10-fold cross validation is recommended for accuracy

estimation due its low bias and variance [HAN2011].

Listing 4.7: k-fold cross-validation accuracy estimation on iris dataset.

```
#include <iostream>
#include <ufjfmltk/ufjfmltk.hpp>

namespace classifier = mltk::classifier;
namespace valid = mltk::validation;

int main() {
    mltk::Data<> data("iris_mult.csv");
    classifier::PerceptronPrimal<double> perc;
    classifier::OneVsAll<double> ova(data, perc);

    valid::ValidationReport report = valid::kfold(data, ova, 10);

    std::cout << "Accuracy = " << report.accuracy << std::endl;
}
```

Listing 4.7 loads the iris dataset for a multi-class classification task and instantiate the OVA wrapper with a perceptron wrapper object. After that, it calls the k-fold cross-validation, stratified by default, with $k = 10$ on OVA with all the dataset data generating a report with metrics from k-fold. Finally, it prints the k-fold accuracy on the screen.

CHAPTER 5

Feature Selection

Feature selection is the process of selecting a subset from the original feature set, following some criterion, without changing the results [VILLELA2011].

5.1 Filter methods

Filter selection methods introduces a serate step before the learning algorithm application. This preprocessing step consider some general features from samples to select some attributes and remove others. It acts by utilizing some criterion to "filter" the irrelevant features [VILLELA2011].

5.1.1 Golub and Fisher

These methods are based on the different of the magnitude of the expression levels of each feature. A feature can be said differentially expressed in two distinct classes if the difference between the weighted average of these classes divided by the sum of the standard deviation is high. Defining μ_1 and μ_2 as both classes average, and σ_1 and σ_2 as its standard deviations, respectively. The scores Golub G and Fisher F can be defined as following:

$$G = \frac{|\mu_1 - \mu_2|}{(\sigma_1 + \sigma_2)}$$

$$F = \frac{(\mu_1 - \mu_2)^2}{(\sigma_1 + \sigma_2)}$$

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace classifier = mltk::classifier;
namespace feat = mltk::featselect;

int main(){
    auto data = mltk::datasets::make_blobs(100, 2, 30).dataset;
    classifier::IMAp<double> ima;
```

(continues on next page)

(continued from previous page)

```
    feat::Golub<double> g(data, &ima, 3);
    feat::Fisher<double> f(data, &ima, 3);

    std::cout << "Initial dims: " << data.dim() << std::endl;
    mltk::Data gresult = g.selectFeatures();
    std::cout << "Final dims: " << gresult.dim() << std::endl;

    std::cout << "\nInitial dims: " << data.dim() << std::endl;
    mltk::Data fresult = f.selectFeatures();
    std::cout << "Final dims: " << fresult.dim() << std::endl;
}
```

5.2 Embedded methods

This strategy is based on the fact that some induction algorithms can perform their own feature selection. These methods use the induction algorithm to estimate the value of the selected feature subset during the training phase and they generaly are specific for a given classification algorithm [VILLELA2011].

5.3 Wrapper methods

The wrapper approach generates several features subsets as candidates, executes the induction algorithm individually in each subset and uses the classifier precision to evaluate the subset in question. The process is repeated until it reaches a stop criterion. The general idea is that the feature selection algorithm exists as a wrapper around the inductor that is responsible to conduct the search by a good feature subset [VILLELA2011].

5.3.1 Recursive Feature Elimination (RFE)

This method works by removing a fixed number of features at a time. The features selected for removal at each step are those that have the smallest rankings computed using a classifier trained after each removal. The elimination of only one feature at a time generates a classifier with less errors. For computational reasons, its prefered to remove several features at a time, at the expense of the classifier degradation [GUYON2002].

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace classifier = mltk::classifier;
namespace feat = mltk::featselect;

int main(){
    auto data = mltk::datasets::make_blobs(100, 2, 30).dataset;
    classifier::IMAp<double> ima;
    feat::RFE<double> rfe(data, &ima, 3);

    std::cout << "Initial dims: " << data.dim() << std::endl;
    mltk::Data rfe_result = rfe.selectFeatures();
    std::cout << "Final dims: " << rfe_result.dim() << std::endl;

}
```

5.3.2 Admissible Ordered Search (AOS)

This wrapper method uses as predictor a large margin classifier, such as IMA [VILLELA2016]. For each dimension, it finds the classifier with the largest margin [VILLELA2015]. *TODO*

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace classifier = mltk::classifier;
namespace feat = mltk::featselect;

int main(){
    auto data = mltk::datasets::make_blobs(100, 2, 30).dataset;
    classifier::IMAp<double> ima;
    feat::AOS<double> aos(data, &ima, 3);

    std::cout << "Initial dims: " << data.dim() << std::endl;
    mltk::Data aos_result = aos.selectFeatures();
    std::cout << "Final dims: " << aos_result.dim() << std::endl;

}
```

CHAPTER 6

Regression

The **Regression** problem consists in predicting the real valued labels of the points being considered [MEHRYAR2018]. Here we're interested in regression as a tool for value forecasting. Each observed point $p = (x, y)$ is the result of the functon $y = \{(x)$, where x are the feature variables and y is the target variable. Given $\mathcal{S} = p_1, \ldots, p_n$, the objective of regression is to find $\{(x)$ that best explains the points on $\mathcal{S}$ [SKIENA2017].

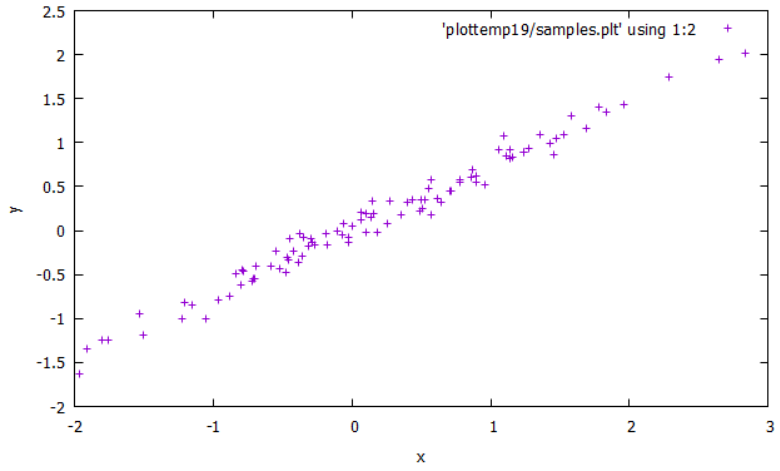

Fig. 6.1: Example of a regression dataset, where x is the regression feature and y is the desired value.

6.1 Linear regression

Given a set of n points, **linear regression** seeks to fit a line which best describes the points, as shown in Fig. 6.2. There are several reasons to use linear regression models, some of them are simplification and comprehension. As the regression shows the underlying trend in the data and highlights the location and magnitude of outliers, it's a useful tool for visualization [SKIENA2017].

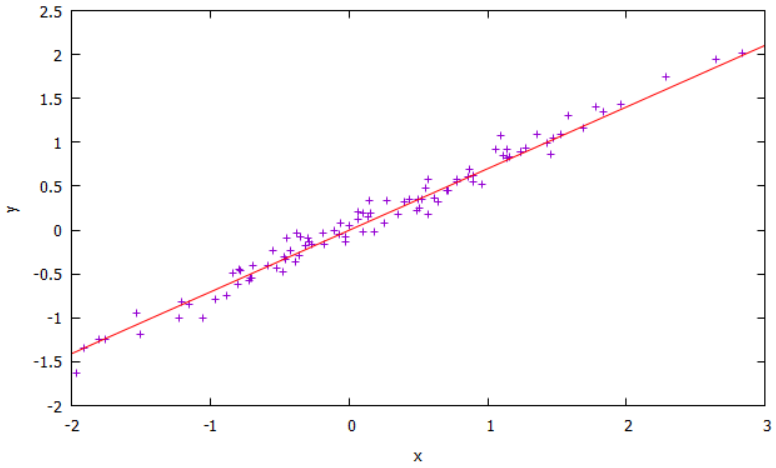

Fig. 6.2: Example of a linear regression model fit on artificial data.

Listing 6.1 generates an artificial regression problem with one dimension on the feature set and one target variable. After the data generation, the data is plotted as in Fig. 6.1 and a wrapper for the **Least Mean Squares (LMS)** algorithm is instantiated with a learning rate of 0.05. The wrapper object is configured with a max training time of 200ms and is trained. Finally, the final hyperplane is plotted as Fig. 6.2.

Listing 6.1: Regression example using the LMS algorithm.

```
#include <ufjfmltk/ufjfmltk.hpp>

int main(){
    auto data = mltk::datasets::make_regression(100, 1, 0, 0.1, 0.01, 10, true, 2).
↪dataset;
    mltk::visualize::Visualization<double> vis(data);
    mltk::regressor::LMSPrimal<> lms(data, 0.05, 1);

    lms.setMaxTime(200);
    lms.train();

    vis.plot1DRegresion();
    vis.plot1DRegresionHyperplane(0, lms.getSolution());
}
```

CHAPTER 7

Clustering

Clustering is the process of grouping a set o data objects into several groups. Each group has points that are very similar to others in the same group, while at the same time they are very dissimilar to points in other groups. Similarities and dissimilarities are usually measured with a distance metric in clustering algorithms [HAN2011].

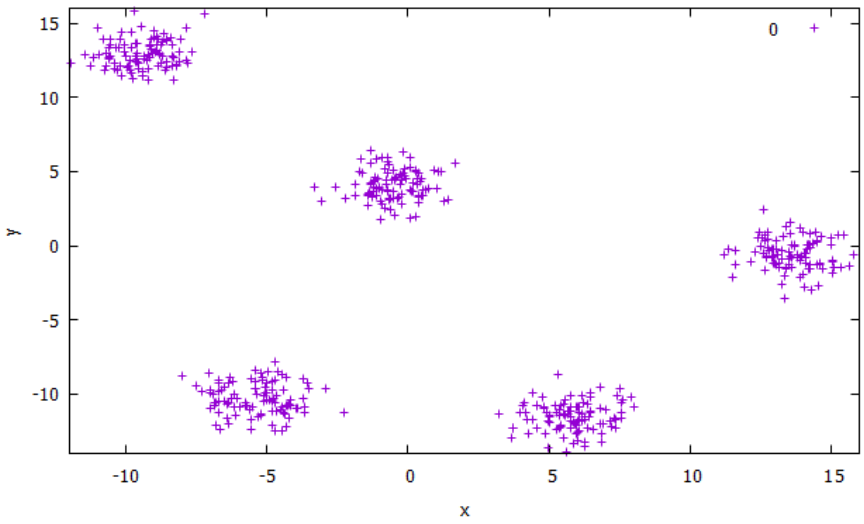

Fig. 7.1: Example of blobs dataset with 5 point clouds.

7.1 Partitioning methods

Partitioning methods are based on the construction of k partitions of the data from a set of n objects, where each partition represents a cluster and $k \leq n$. Usually, partitioning methods adopt *exclusive cluster separation*, i.e., each object must belong to exactly one group. Most partitioning methods are distance-based [HAN2011].

7.1.1 The k-Means algorithm

The k-means algorithm works by first assigning as centroids of the clusters k random points from the dataset, representing the mean of the clusters. For each remaining objects, each of them is assigned to the cluster were it is most similar based on the euclidean distance and the cluster mean. Then, it iteratively improves the within-cluster variation. At each iteration it computes the new clusters means using the objects assigned in the previous iteration and only stops when the assignment is stable, that is, the clusters formed in the current round are the same as those formed in the previous round [HAN2011]. Listing 7.1 shows how to generate blobs artificial data for clustering algorithms, plots the dataset, instantiates the k-means clustering algorithm wrapper and train it.

Listing 7.1: k-means training on blobs dataset with $k = 5$

```
#include <ufjfmltk/ufjfmltk.hpp>

namespace visual = mltk::visualize;
namespace cluster = mltk::clusterer;

int main(){
    auto data = mltk::datasets::make_blobs(100, 5, 2, 1, -15, 15,
                                           true, false, 2).dataset;
    visual::Visualization<double> vis(data);

    vis.plot2D();

    cluster::KMeans<double> kmeans(data, 5, "kmeanspp", 2, 1);

    kmeans.train();
    mltk::Data result = kmeans.batchEvaluate(data);
    vis.setSample(result);
    vis.plot2D();
}
```

After training the model, the data can be passed to the method `batchEvaluate` that returns a new dataset object with the points associated to the clusters. As we can see at Fig. 7.2, the k-means algorithm was able to correctly separate similar data in well defined clusters.

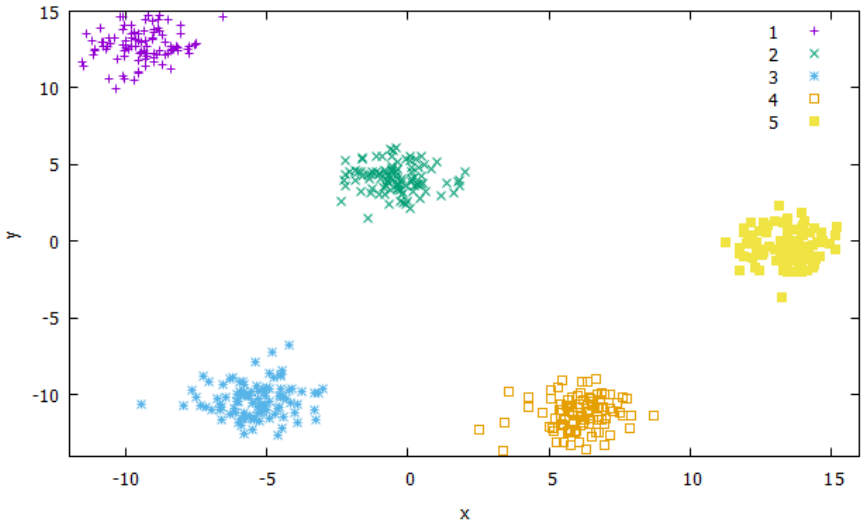

Fig. 7.2: Clustering result using k-means with $k = 5$.

CHAPTER 8

Extending the framework

8.1 Implementing classifiers

TODO

8.1.1 Perceptron primal algorithm

TODO

8.1.2 Perceptron dual algorithm

CHAPTER 9

Contributor Covenant Code of Conduct

9.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

9.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

9.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

9.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

9.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at mateus.marim@ice.ufjf.br. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

9.6 Attribution

This Code of Conduct is adapted from the **Contributor Covenant** homepage, version 1.4, available at version

CHAPTER 10

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

10.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

10.2 TERMS AND CONDITIONS

10.2.1 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

10.2.2 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only

to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

10.2.3 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

10.2.4 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

10.2.5 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

10.2.6 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

10.2.7 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge

under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

10.2.8 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

10.2.9 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

10.2.10 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10.2.11 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

10.2.12 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

10.2.13 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

10.2.14 13. Use with the GNU Affero General Public License.

Not with standing any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

10.2.15 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

10.2.16 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

10.2.17 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRO-

GRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

10.2.18 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

10.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.

Bibliography

[SKIENA2017] Skiena, Steven S. The data science design manual. Springer, 2017.

[VILLELA2011] Villela, Saulo Moraes, et al. “Seleção de Características utilizando Busca Ordenada e um Classificador de Larga Margem.” (2011).

[ROSENBLATT1958] Rosenblatt, Frank. “The perceptron: a probabilistic model for information storage and organization in the brain.” Psychological review 65.6 (1958): 386.

[MEHRYAR2018] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning. MIT press, 2018.

[BISHOP2007] Bishop, Christopher M. “Pattern recognition and machine learning (information science and statistics).” (2007).

[HAN2011] Han, Jiawei, Jian Pei, and Micheline Kamber. Data mining: concepts and techniques. Elsevier, 2011.

[VILLELA2011] Villela, Saulo Moraes, et al. “Seleção de Características utilizando Busca Ordenada e um Classificador de Larga Margem.” (2011).

[GUYON2002] Guyon, Isabelle, et al. “Gene selection for cancer classification using support vector machines.” Machine learning 46.1 (2002): 389-422.

[VILLELA2015] Villela, Saulo Moraes, Saul de Castro Leite, and Raul Fonseca Neto. “Feature selection from microarray data via an ordered search with projected margin.” Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.

[VILLELA2016] Villela, Saulo Moraes, Saul de Castro Leite, and Raul Fonseca Neto. “Incremental p-margin algorithm for classification with arbitrary norm.” Pattern Recognition 55 (2016): 261-272.

[MEHRYAR2018] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning. MIT press, 2018.

[SKIENA2017] Skiena, Steven S. The data science design manual. Springer, 2017.

[HAN2011] Han, Jiawei, Jian Pei, and Micheline Kamber. Data mining: concepts and techniques. Elsevier, 2011.