First Homework: Semisupervised Learning

University of Padova: Optimization For Data Science
Group members:
Massimiliano Conte, Pierpoalo D'Odorico, Eddie Rossi, Luca Solbiati

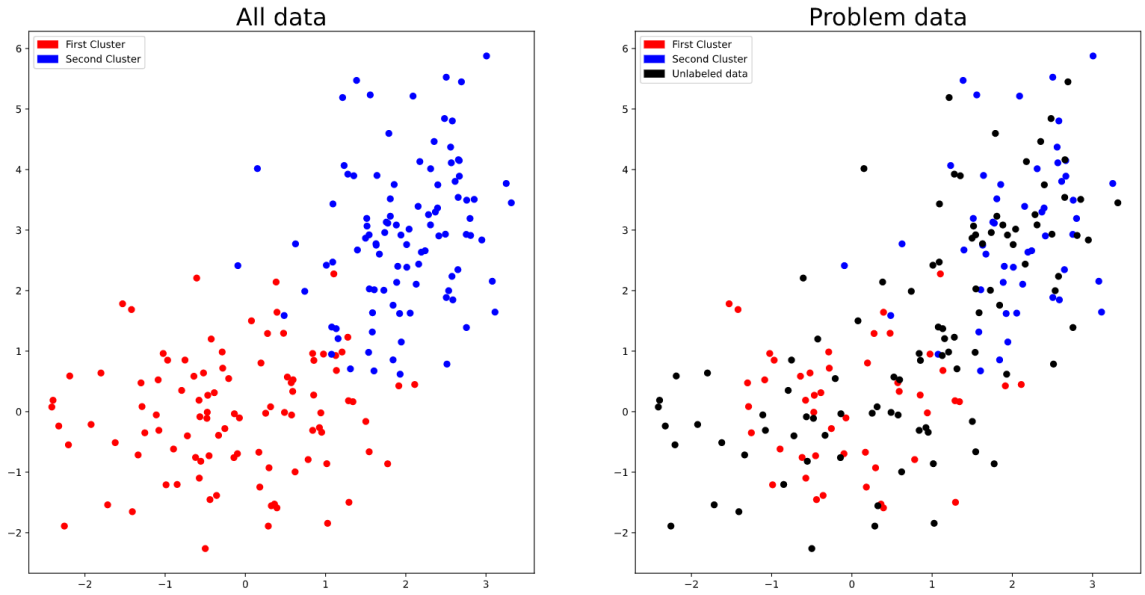
May 2021

Contents

1 Semisupervised Learning

1.1 The Semisupervised Learning Problem

In a semisupervised learning problem we're given l labeled data points $(\overline{x}_i, \overline{y}_i)_{i=1,...,l}$ and u unlabeled points $(x_j)_{j=1,...,u}$. Our goal is to label each unlabeled point x_j with a proper y_j.



Consider the above image as an example: we're given in $\mathbb{R}^2$ two classes of points (red and blue) and a list of unclassified points (black). We want to determine wether a black point is red or blue.

We'll only consider problems with 2 classes, so $y_j \in \{-1, 1\}$. We assume that similar features between x_i and x_j imply similar labels y_i and y_j. We shall also assume that given two features $x_j, x_i \in \Omega$ we're able to measure how much they are similar to one another through a weight function:

$$W : \Omega \times \Omega \to [0, +\infty)$$

The higher $W(x_i, x_j)$ is, the more similar x_i and x_j are. Observe that W is a symmetric function. Let us define:

$$\begin{cases} \overline{w}_{i,j} := W(x_i, \overline{x}_j) & \text{as the similarity between an unlabeled and labeled example} \\ w_{i,j} := W(x_i, x_j) & \text{as the similarity between two unlabeled examples} \end{cases}$$

In order to find the labels $y := (y_1, ..., y_u)$ we want to solve the minimization problem:

$$\min_{y \in \mathbb{R}^u} f(y) := \min_{y \in \mathbb{R}^u} \sum_{i=1}^{u} \sum_{j=1}^{l} \overline{w}_{i,j} (y_i - \overline{y}_j)^2 + \frac{1}{2} \sum_{i=1}^{u} \sum_{j=1}^{u} w_{i,j} (y_i - y_j)^2$$

1.2 Partial Derivatives Of $f(y)$ For GD Schemes

To minimize $f(y)$ we'll use GD like schemes so we need to compute the partial derivatives of $f(y)$:

$$\begin{aligned}
\frac{\partial}{\partial y_k} f(y) &= \frac{\partial}{\partial y_k}\left[\sum_{i=1}^{u}\sum_{j=1}^{l}\overline{w}_{i,j}(y_i-\overline{y}_j)^2+\frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u}w_{i,j}(y_i-y_j)^2\right] \overset{\text{(linearity of derivatives)}}{=} \\
&= \sum_{i=1}^{u}\sum_{j=1}^{l}\frac{\partial}{\partial y_k}\overline{w}_{i,j}(y_i-\overline{y}_j)^2+\frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u}\frac{\partial}{\partial y_k}w_{i,j}(y_i-y_j)^2 = \\
&= \sum_{i=1}^{u}\sum_{j=1}^{l}\overline{w}_{i,j}\,2\,\delta_{k,i}(y_i-\overline{y}_j)+\frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u}w_{i,j}2\,(\delta_{k,i}-\delta_{k,j})(y_i-y_j) = \\
&= \sum_{j=1}^{l}\overline{w}_{k,j}\,2(y_i-\overline{y}_j)+\sum_{j=1}^{u}w_{k,j}(y_k-y_j)-\sum_{i=1}^{u}w_{i,k}(y_i-y_k) \overset{(w_{i,k}=w_{k,i})}{=} \\
&= \sum_{j=1}^{l}\overline{w}_{k,j}\,2(y_i-\overline{y}_j)+\sum_{j=1}^{u}w_{k,j}(y_k-y_j)+\sum_{i=1}^{u}w_{k,i}(y_k-y_i)
\end{aligned}$$

Where $\delta_{k,i}$ is Kronecker's delta. Hence we can conclude that:

$$\frac{\partial}{\partial y_k} f(y) = 2\left[\sum_{j=1}^{l}\overline{w}_{k,j}\,(y_i-\overline{y}_j)+\sum_{j=1}^{u}w_{k,j}(y_k-y_j)\right]$$

1.3 Matrix Notation And Computation of $f(y), \nabla f(y)$

Since u is usually a big number we have to be careful of how we compute the cost function $f(y)$ and its partial derivatives. Indeed most of the computation time will be dedicated to this task. A way to significantly speed it up is by using a vectorized notation and exploiting the power of parallel computing. So let us define the matrix:

$$W := \begin{bmatrix}
\overline{w}_{1,1} & \overline{w}_{1,2} & \dots & \overline{w}_{1,l} & w_{1,1} & w_{1,2} & \dots & w_{1,u} \\
\overline{w}_{2,1} & \overline{w}_{2,2} & \dots & \overline{w}_{2,l} & w_{2,1} & w_{2,2} & \dots & w_{2,u} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\overline{w}_{u,1} & \overline{w}_{u,2} & \dots & \overline{w}_{u,l} & w_{u,1} & w_{u,2} & \dots & w_{u,u}
\end{bmatrix} \in M_{u,u+l}(\mathbb{R})$$

Then if we denote the k-th row of W as W_k we have:

$$\frac{\partial}{\partial y_k} f(y) = 2\left[\sum_{j=1}^{l}\overline{w}_{k,j}\,(y_i-\overline{y}_j)+\sum_{j=1}^{u}w_{k,j}(y_k-y_j)\right] = 2W_k\cdot\begin{bmatrix}
y_k-\overline{y_1} \\ \vdots \\ y_k-\overline{y_l} \\ y_k-y_1 \\ \vdots \\ y_k-y_u
\end{bmatrix}$$

To compute the cost function $f(y)$ we can define the matrix:

$$Y := \begin{bmatrix}
(y_1-\overline{y}_1) & \dots & (y_1-\overline{y}_l) & \frac{1}{\sqrt{2}}(y_1-y_1) & \frac{1}{\sqrt{2}}(y_1-y_2) & \dots & \frac{1}{\sqrt{2}}(y_1-y_u) \\
(y_2-\overline{y}_1) & \dots & (y_2-\overline{y}_l) & \frac{1}{\sqrt{2}}(y_2-y_1) & \frac{1}{\sqrt{2}}(y_2-y_2) & \dots & \frac{1}{\sqrt{2}}(y_2-y_u) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
(y_u-\overline{y}_1) & \dots & (y_u-\overline{y}_l) & \frac{1}{\sqrt{2}}(y_u-y_1) & \frac{1}{\sqrt{2}}(y_u-y_2) & \dots & \frac{1}{\sqrt{2}}(y_u-y_u)
\end{bmatrix} \in M_{u,u+l}(\mathbb{R})$$

Then we can set $C := W * Y * Y$ (where $*$ is the element wise multiplication) and easily verify that:

$$f(y) = \sum_{i=1}^{u}\sum_{j=1}^{l} \overline{w}_{i,j}(y_i - \overline{y}_j)^2 + \frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u} w_{i,j}(y_i - y_j)^2 = \sum_{i=1}^{u}\sum_{j=1}^{u+l} C_{i,j}$$

Moreover to compute the matrix Y we can also use a vectorized notation:

$$Y_1 := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_u \end{bmatrix} \cdot \overbrace{\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}}^{l \text{ components}} - \overbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}^{u \text{ components}} \cdot \begin{bmatrix} \overline{y_1} & \overline{y_2} & \dots & \overline{y_l} \end{bmatrix}$$

$$Y_2 := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_u \end{bmatrix} \cdot \overbrace{\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}}^{u \text{ components}} - \overbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}^{u \text{ components}} \cdot \begin{bmatrix} y_1 & y_2 & \dots & y_u \end{bmatrix}$$

So that $Y = \begin{bmatrix} Y_1 & \frac{1}{\sqrt{2}} Y_2 \end{bmatrix}$

1.4 Weight Matrix Manipulation

Sometimes the data we're given is very skewed. Using a normal (or unbiased) weight function in such cases produces a model with poor generalization. To avoid this we can do some direct manipulation on the weight matrix W. As long as the weights $w_{i,j}$ for the unlabeled examples maintain symmetry the previous gradient calculations remain valid.

The data may have two problems:

1) Unbalanced data set: When one class (let's say the class +1) is over represented. In such cases the model would produce a dummy classifier constant to +1. One basic idea to fix this is to artificially add the same points of the class -1 until a 50/50 representation is reached.

The probem with this solution is that it adds a significant computational burden, as it makes the weight matrix bigger.
A way to obtain the same effect is by changing the weights $\overline{w_{i,j}}$ as follows. Let:

$$b_1 := \frac{|\{\overline{y_j} \mid \overline{y_j} = +1\}|}{l} \quad b_2 := \frac{|\{\overline{y_j} \mid \overline{y_j} = -1\}|}{l}$$

Then we can redefine:

$$\overline{w_{i,j}}^{(new)} = \begin{cases} \frac{0.5}{b1}\ \overline{w_{i,j}} & \text{if} \quad \overline{y_j} = +1 \\ \\ \frac{0.5}{b2}\ \overline{w_{i,j}} & \text{if} \quad \overline{y_j} = -1 \end{cases}$$

This has the effect of rebalancing the weight influence in such a way that the two classes representation is 50/50

2) Too many unlabeled data points: If $u >> l$ then the model mostly ingores the labeled data and performs a normal clustering. In the worst cases it may still produce a dummy classifier. To avoid that we may proceed in a similar way as before and change the weights $\overline{w_{i,j}}$ so that the two classes of labeled and unlabeled data points have a 50/50 representation. We redifine:

$$\overline{w_{i,j}}^{(new)} := \frac{u}{l} \cdot \overline{w_{i,j}}$$

The logic behind the formula is that the "new" number of labeled data points will be $l^{(new)} = l \cdot \frac{u}{l} = u$ hence giving a 50/50 representation.

2 Development on fake dataset

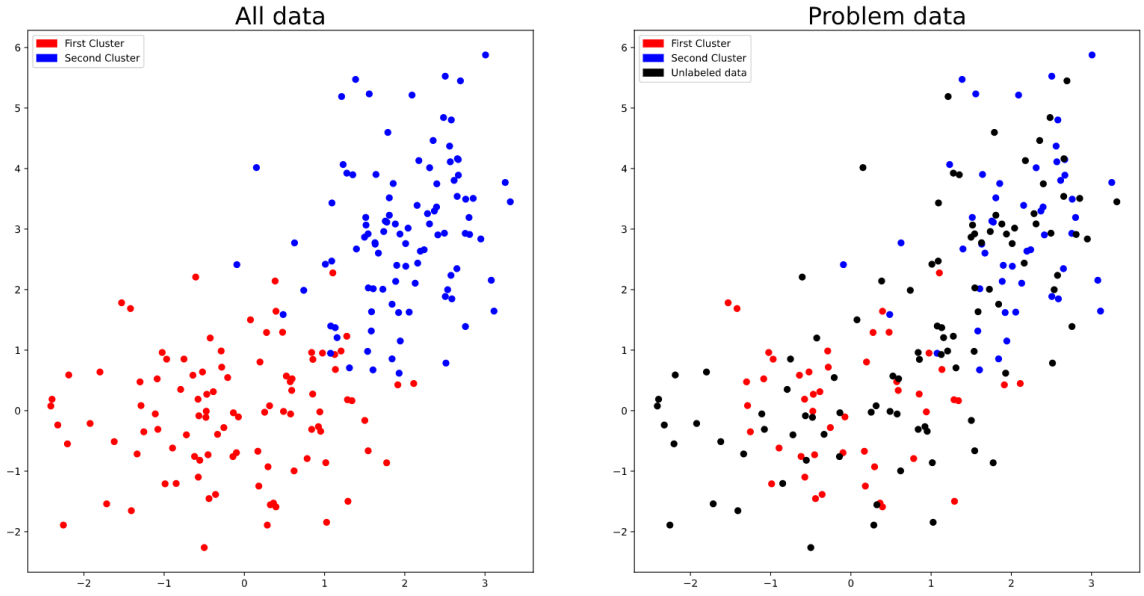
2.1 Instance generation

Our goal is to develop a set of algorithms capable of solving the semisupervised optimization problem. In order to do that, we start by generating a fake dataset. The main steps of the process are:

- **Generate cluster one points**, which are IID observations from a bidimensional normal random variable $N(\mu_1, \Sigma_1)$, where $\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
- **Generate cluster two points**, which are IID observations from a bidimensional normal random variable $N(\mu_2, \Sigma_2)$, where $\mu_2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & \frac{3}{2} \end{bmatrix}$.
- **Forget a subset of labels:** Each label is the cluster in which the point belongs to, coded in:
  - 1 for the first cluster;
  - -1 for the second cluster.

  We give to the model only a randomly selected subset of labels, and because we know the true labels, at the end we can see if the classifications are correct.

For this dataset we generated 100 observations per cluster, and forgot half of the labels.
We decided the actual values for the mean and the variance in such a way that the classifications are easy enough, but non trivial. At the end of the process this is how the dataset looks like:



2.2 Similarity matrix

The objective function and its gradient depend on a measure of similarity between points. The definition of what makes an observation x_i similar to another one x_j is strongly related to the domain of the problem. In this fake data setting, we can't rely on such information, so we define the similarity matrix W based on the euclidean distance, that is the standard distance measure.
We want each $w_{i,j} \in [0, 1]$, where $w_{i,j} = 0$ means that the two observations are infinitely far from

each other, while $w_{i,j} = 1$ means that the two points have the same identical entries. In order to do that we define:

$$w_{i,j} := \frac{1}{1 + ||x_i - x_j||_2^2}$$

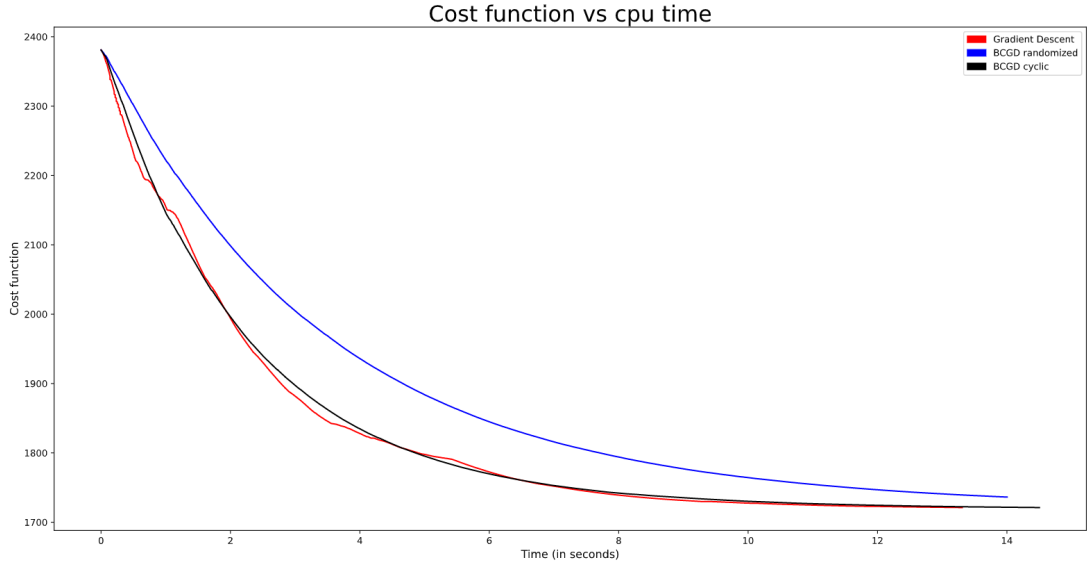
Several modifications of this weight function can be made depending on the problem, for example we can use a different distance measure like the *L1-norm*.

2.3 Algorithms implementation and testing

We implemented, using python, the optimization algorithms:

- *Gradient descent*
- *Block coordinate gradient descent cyclic*
- *Block coordinate gradient descent randomized*

using a fixed stepsize for all of the 3 methods. As a starting point we chose y equal to zero, since it is the mean value inbetween -1 and 1.
Then we ran each algorithm in order to solve the minimization task, measuring their performance. We report the plot of the error function against CPU time, not against iteration, since the *BCGD randomized* does one single block update per iteration, while the others compute the whole gradient per iteration, so it doesn't make sense to compare them that way.



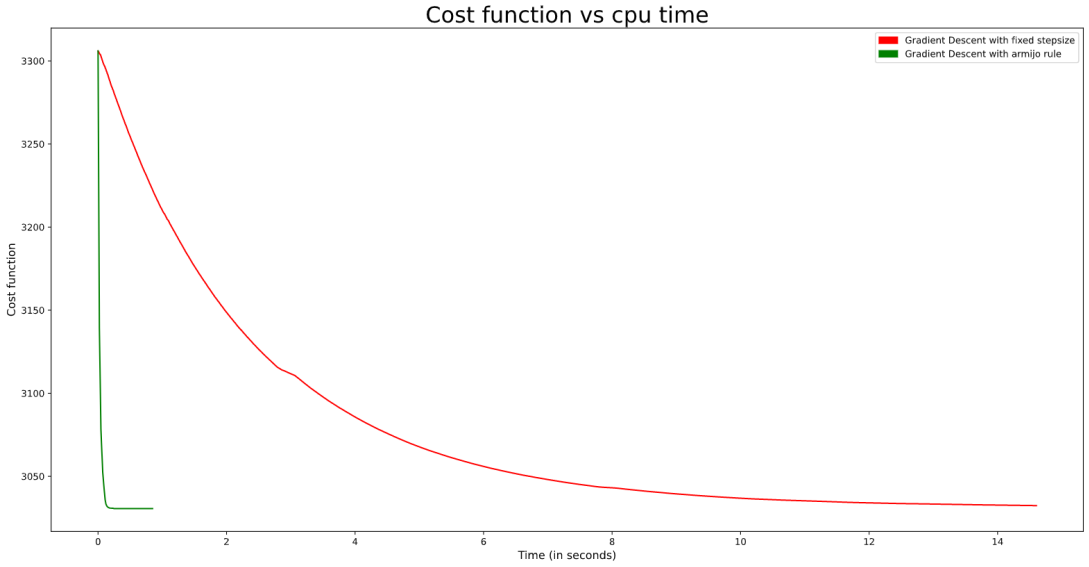
From the plot we can see how *Gradient descent* and *Block coordinate gradient descent cyclic* seem equivalent, while *Block coordinate gradient descent randomized* is slower. This can be due to the fact that *BCGD* methods aren't exploiting some structure, and updating one block at the time doesn't speed up the optimization. Moreover, the random number generation needed in the *BCGD randomized*, for the selection of the block to update, is slowing down the process.

2.4 Line search

We chose the value for the fixed stepsize by making several experiments and analysing the results. Now we want to know if a more sofisticated line search can speed up the optimization. We implemented the *Armijo rule* in the *Gradient descent*, setting $\delta = 0.1$ and $\gamma = 0.4$ [1]. Then we introduced a stopping criteria: if the function evaluation doesn't have a sufficient improvement by trying several

[1] We're using the same notation we've seen in the course lectures

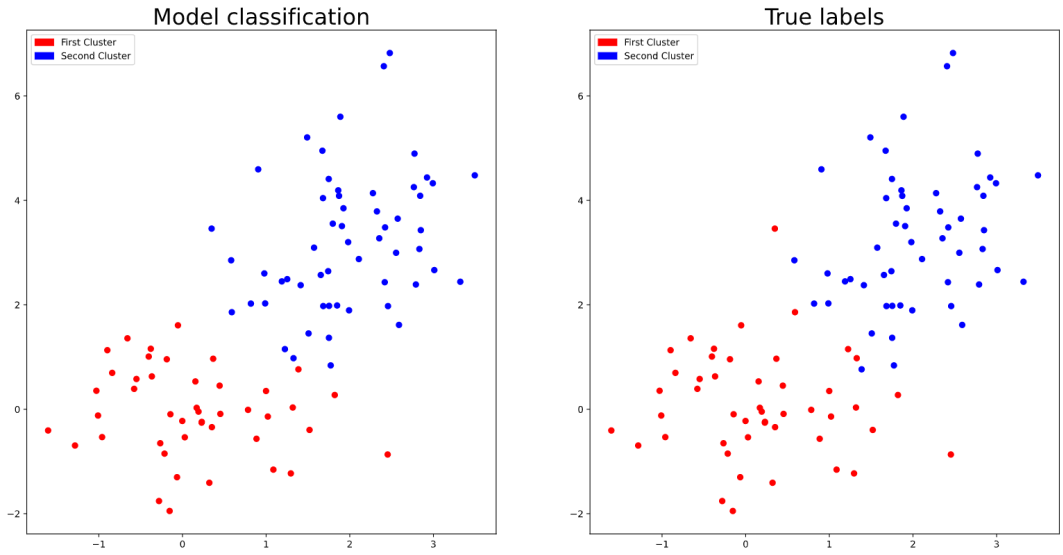
updates (according to δ and γ), then we stop the algorithm. In particular, we tried values of the stepsize down to 10^{-8} and stopped the algorithm in the case of non sufficient descrease of the objective function. The evaluation of the objective function is costly, but it let us use larger stepsizes that can lead to faster optimization. Here there is the comparison between *Gradient descent* with and without *Armijo rule*:



As we can see from the plot, in this settings the line search led to the same cost value, but with a significant reduction to the CPU time. The stopping condition was satisfied and the algorithm stopped the execution.

2.5 Model evaluation

Does the model correctly classify the unlabeled observations? Since we know the true labels, we can plot the differences between the predicted labels and the true labels. The prediction is made by looking at each y_i : we classify x_i as 1 if $y_i > 0$, as -1 otherwise (basically we're picking the nearest value to y_i in $\{-1, 1\}$).



The classifications are almost all correct (93% accuracy), and as we can see from the plot, the wrong ones are mostly on outliers, so the model generalizes well.

3 Development on real dataset

3.1 Pokemon data

We want to test the semisupervised learning model. The algorithms are the same used on the fake data, our goal is to determine how those procedures deal with a real dataset. The dataset contains data about 800 pokemons, divided in two clusters: *Non Legendary* and *Legendary*. The features we will use are the following numerical variables:

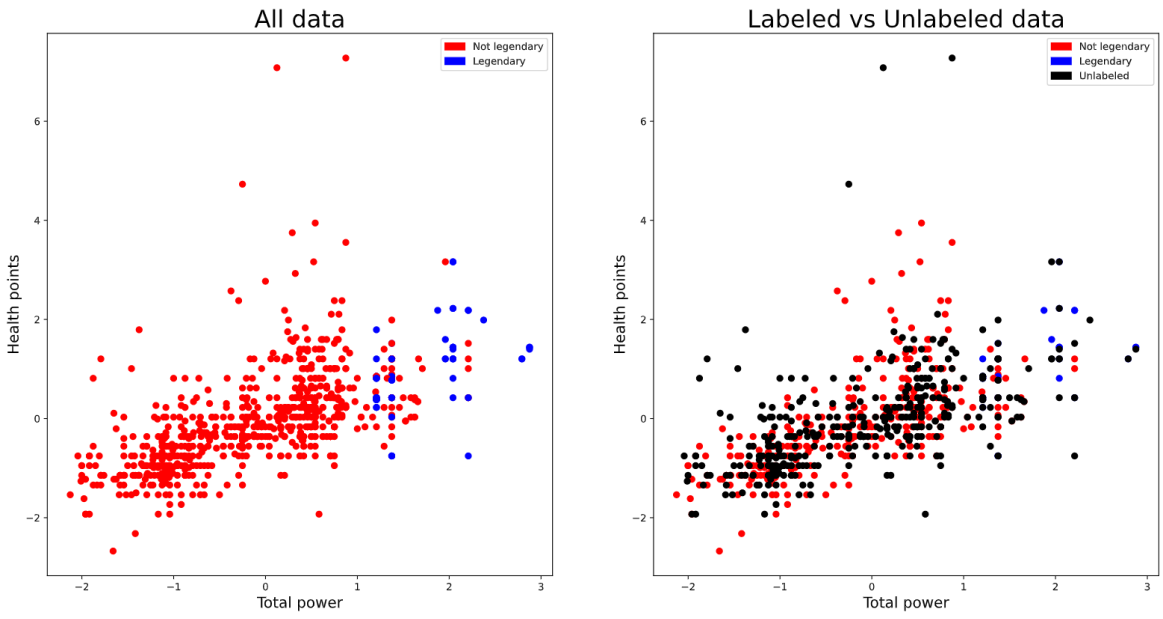
- *Total Power*
- *Health Points*
- *Attack*
- *Defense*
- *Sp. Atk*
- *Sp. Def*
- *Speed*

To standardize the data we computed: $z_{i,j} = \frac{x_{ij}-\mu_j}{\sigma_j}$ where x_{ij} is the observation $i = 0, ..., N-1$ in column $j = 0, .., 6$. With this procedure we find a feature matrix $X = (z_{i,j})_{i,j}$ similar to the bidimensional one created as fake dataset. Then we create a response vector y composed by:

· 1 for Legendary pokemon;

· -1 for Non Legendary pokemon.

The weight matrix W is computed as before using the eulcidean distance.
We give to the model only a randomly selected subset of labels, and because we know the true labels, at the end we can see if the classification of legendary pokemon is correct. Looking at the data we can see that the two output classes are highly unbalanced. We specifically chose the dataset for this reason, as unbalanced data is really common in real world applications.



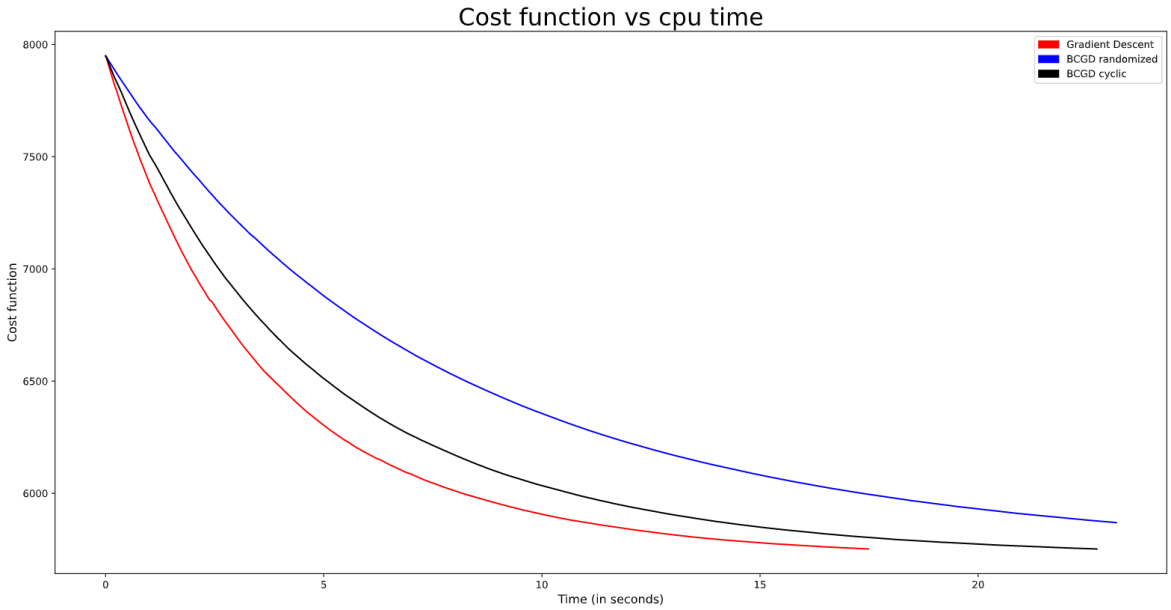
At the end of the process we plotted the Legendary and Non Legendary pokemon in a plot using the first and the second features in X matrix, called "Total Power" and "Health Points". We can notice two clusters, in fact the legendary pokemon have a huge total power and a high number of health points.

3.2 Algorithms testing on pokemon data

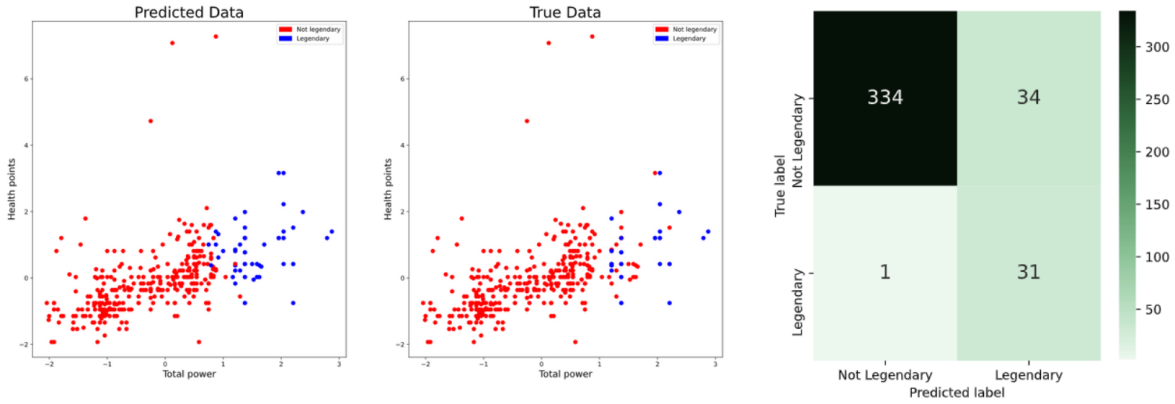
We tested the following optimization algorithms, as we did for the fake dataset:

- *Gradient descent*
- *Block coordinate gradient descent cyclic*
- *Block coordinate gradient descent randomized*

To compare their behaviour we plotted a graph with the CPU time in the x axes and the Cost Function decrease on the y axes:



From the plot we can see how *Gradient descent* performs better in terms of time, while *Block coordinate gradient descent randomized* seems to be the slower one.
We looked at the classification matrix because the common problem with this type of unbalanced data is that we can have a super high accuracy using a dummy classifier. In fact the semisupervised model classifies all the pokemons as Not Legendary. To fix this problem we can use the manipulation of the weight matrix described before. With this trick we give more importance to the pokemons labeled as Legendary. Using this normalization we obtained the following predictions with lower accuracy but better recall metric:



As we said this approach makes possible to classify some pokemon as legendary and it does not produce a dummy classifier. We noticed that the wrong classified legendary pokemon are very close to the little cluster containing the most powerful ones, so the model generalizes well.

3.3 Heart disease data

We selected also a balanced dataset for having a better view on the algorithms performing on balanced data. This dataset consists of 270 patients between 29 and 77 years old with or without heart disease. The two classes are quite balanced: the presence of heart disease was observed in 120 samples, while the remaining 150 were healthy. For each patient a series of medical information and health indicators are reported, among which we have selected the following 9 features:

- *Age*
- *Chest pain type*
- *Resting blood pressure*
- *Cholesterol level*
- *Electrocardiogram at rest*
- *Maximum heart rate achieved*
- *ST depression induced by exercise relative to rest*
- *Slope of the peak exercise ST segment*
- *Number of major vessels colored by fluoroscopy*

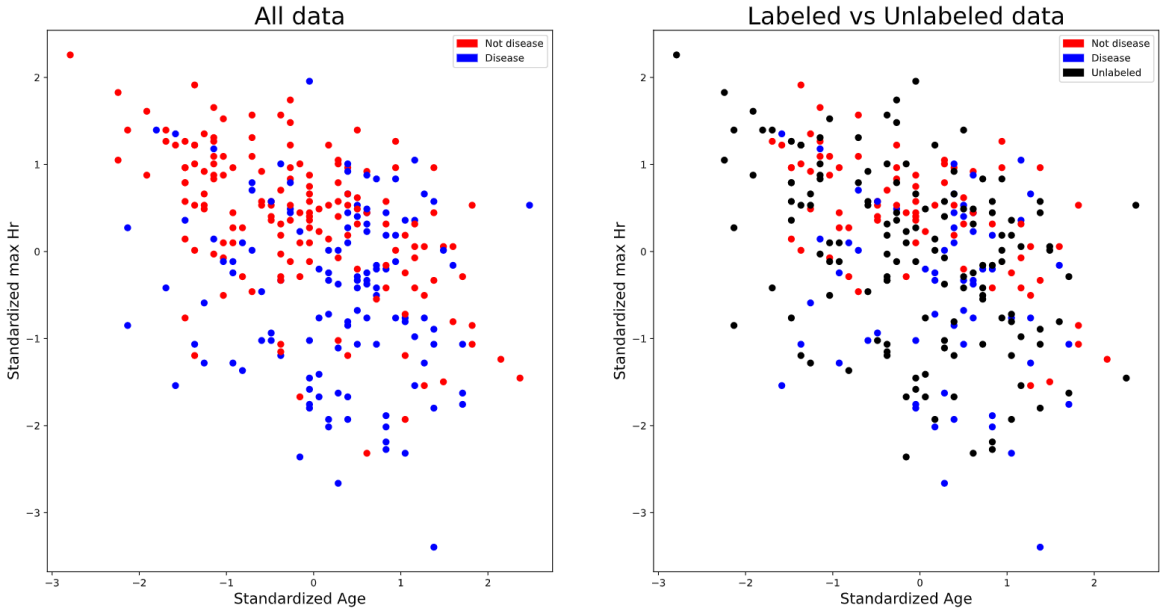
We standardized the features by removing the mean of each column and dividing the result by the column standard deviation. We marked the two classes as follows:

· 1 for presence of heart disease;

· -1 for absence of heart disease.

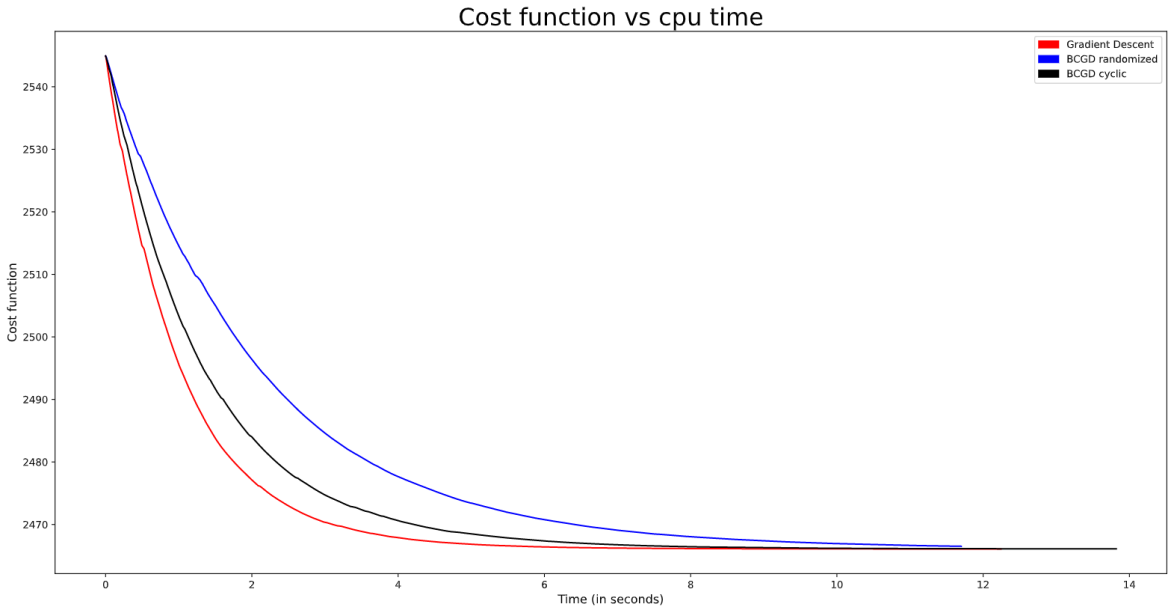
We computed the similarity measure as defined before. We then split the dataset and considered 50% of the samples as unlabeled.

By plotting the samples according to two of the features, *Standardize age* and *Standardize maximum heart rate*, we can distinguish the two clusters.



3.4 Algorithms testing on heart disease data

We tested the algorithms and compared their performances by plotting the descent of the Cost Function as a function of CPU time.



As for the previous datasets *Gradient descent* performs the best, followed by *Block coordinate gradient descent cyclic* and *Block coordinate gradient descent randomized*. The final accuracy is around 80% .