Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Departman za matematiku i informatiku

Hunor Tot-Bagi

Inexact Restoration method for solving Hinge loss problems

Master thesis

2022, Novi Sad

Contents

I would like to thank and express my deepest gratitude to my mentor Nataša Krklec Jernikić for her guidance, support, patience and encouragement throughout the whole project. It was a real pleasure working with her. I am grateful to professor Nataša Krejić and professor Dušan Jakovetić for the valuable feedback and being in the commission team. Also, many thanks to Tijana Ostojić for sending the necessary data and Luka Rutešić for helpful advices. In addition, I would like to thank my mother Ana who kept always pushing me through rough times, and to my lovely Katarina for being my oasis.

Chapter 1

Introduction

Hinge Loss are problems of binary classification which are very common in the field of machine learning. The Hinge Loss function is non-smooth, thus in solving these kind of problems one has to use subgradient methods. Regardless of non-differentiability, recent research [21] shows us that if we include information about second order derivatives we can notice an improvement of the algorithm performances compared to first order subgradient methods. Because of that in this thesis we will take a closer look on BFGS variants of algorithms adjusted to non-differentiable functions.

On the other hand, the problems of machine learning mostly fall into so-called big data problems. The big amount of data generated by us humans is constantly growing (online learning, stochastic optimization) which additionally complicates the application process of classical methods, so it is necessary to apply algorithms with variable sample size. Even if we are talking about problems with finite amount of data - problems of finite sums, classical methods are often not efficient and expensive because they are considering all individual (local) functions. Because of this it is necessary to apply algorithms which will on the most sophisticated way determine the sample size, i.e., the number of local functions which will be considered in each iteration and thus reduce the computational costs of the whole optimization process. One of the most efficient methods for this is the Inexact Restoration method [17]. The strength of this method lies in adaptive decision making, that is updating the sample sizes through iterations. The main idea is to work with as small as possible sample size, but in such way that the optimization process and the convergence of the algorithm is intact.

The adaptation of the Inexact Restoration method on problems which are non-smooth is dealt with in a recent paper [15] and this will be the backbone of this thesis. However, the main focus will be implementing, testing and the applications of the method on real world problems, including the ones from IoT sector [1]. First we will test the algorithms on three fairly similar datasets. The first being a mushroom dataset, where we are interested in determining whether a specific mushroom is edible or not. After that we will take a look on a splice dataset which are DNA sequences to recognize two types of junctions exon/intron or intron/exon. Next in the adult dataset we will tell if the person is making more than some fixed amount of money on a yearly basis. Lastly we have the IoT dataset which is a bit different from others in a sense that we don't have labels on the training data so it will be an unsupervised task. Here we will try to predict is the given sample an anomaly or not. In industry, detecting anomalies within the operating systems can be beneficial, as we can predict the malfunctioning of the equipment and mitigate the potential losses. Beside these classification tasks our goal is to measure the performances of these algorithms especially in terms of computational cost measured by FEV - the number of scalar products. The algorithm will be implemented and tested in the programming language Python.

Chapter 2

Machine learning problems

Machine learning (ML) is defined as a discipline of artificial intelligence (AI) and computer science that provides machines the ability to automatically learn from data and past experiences to identify patterns and make predictions with minimal human intervention. Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, uncovering key insights within data mining projects. These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase, requiring them to assist in the identification of the most relevant business questions and subsequently the data to answer them. But how does machine learning works? First there is a decision process where in general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labelled or unlabeled, your algorithm will produce an estimate about a pattern in the data, and the error function serves to evaluate the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model. Lastly there is a model optimization process. If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this evaluate and optimize process, updating weights autonomously until a threshold of accuracy has been met.

Let us mention a few applications of machine learning. In today day and age a vast majority of people around the world are using social media platforms such as Facebook and Instagram. These companies are actively using artificial intelligence tools to process and analyze photos, videos and text to enhance the user experience buy the data which is collected by our actions on these platforms. They are showing us what is the most valued and relevant for each user to create a personalised experience. Also we must not forget the targeted advertising which brings a company the biggest profit. By assessing the search preferences and engagement insights from its users, Instagram can sell advertising to companies who want to reach that particular customer profile and who might be most interested in receiving a particular marketing message. Let us move to the medical diagnosis and healthcare because we can make use of these tools in this field also. Machine Learning incorporates techniques and tools to deal with the diagnostic and prognostic issues in the diverse medical realms. They are highly used for, the analysis of medical data for detecting regularities in data, handling inappropriate data, explaining data generated by medical units, also for effective monitoring of patients. Machine learning also helps in estimating disease breakthroughs, driving medical information for outcomes research, planning and assisting therapy, and entire patient management. Machine learning classifiers fall into three primary categories supervised learning, unsupervised learning and semi-supervised learning.

Supervised learning, also known as supervised machine learning, is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model is fitted appropriately. This occurs as part of the cross validation process to ensure that the model avoids overfitting or underfitting. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. Some methods used in supervised learning include neural networks, naive Bayes, linear regression, logistic regression, random forest, support vector machine (SVM), and more.

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, image and pattern recognition. It is also used to reduce the number of features in a model through the process of dimensionality reduction - principal component analysis (PCA) and singular value decomposition (SVD) are two common approaches for this. Other algorithms used in unsupervised learning include neural networks, k-means clustering, probabilistic clustering methods, and more.

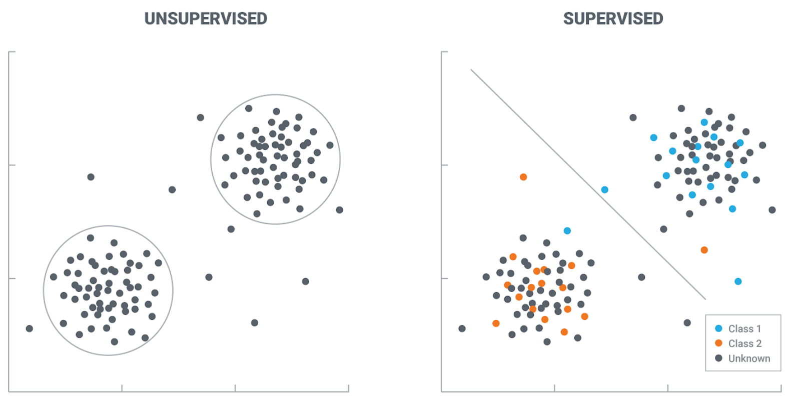

Figure 2.1: Unsupervised and Supervised approach, Source: [5]

Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labeled dataset to guide classification and feature extraction from a larger, unlabeled dataset. Semi-supervised learning can solve the problem of having not enough labeled data (or not being able to afford to label enough data) to train a supervised learning algorithm.

2.1 L_2 - Regularized binary hinge loss

Hinge loss is cost function which is used for training binary classification models. It is mostly used in the support vector machines (SVM). The main idea in the SVM approach is to find the most optimal hyperplane in the n-dimensional space, where n represents the number of features. That hyperplane will separate the data points into two classes. It is clear that this hyperplane is a $n-1$ dimensional subspace of the n-dimensional space.

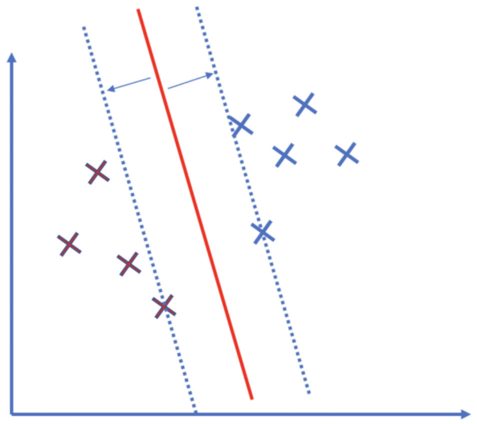

Figure 2.2: Representation of the hyperplanes in $\mathbb{R}^2$, Source: [2]

We can see that there are more available hyperplanes, however we are interested in the one that has the widest margin. Because with that one the separation between the two classes will be the best. More precisely, the best margin is the minimal distance from the hyperplane to the dataset points. The desired maximized hyperplane is the red one in Figure 2.2.

Let x be the coefficient vector of the hyperplane, then the width of the margin can be calculated as $\frac{2}{||x||}$. If we want to maximize it, we need to minimize $||x||$. So we end up with the following optimization problem

$$\min_x \frac{1}{2}||x||^2, \text{ subject to } z_i x^T \omega_i \geq 1, i = 1, \ldots, N, \tag{2.1}$$

where N is the number of samples, $w_i, i = 1, \ldots, N$ are the attribute vectors, and $z_i \in \{-1, 1\}$ is the target value of the sample w_i. Our problem 2.1 has a unique solution, since it is a convex function with affine constraint so the problem is convex. For this type of problem we know that a local solutions are also global.

This kind of a problem is used only in linearly separable datasets. As we know in reality datasets often needs to be separated by nonlinear classifiers.

In order to cope with this nonlinearity, one can apply a penalty function which is "softening" the constrains by moving them into the objective function. We end up with hinge loss problem with regularization. In particular, hinge loss with L_2 regularization objective function is defined as follows

$$f(x) := \frac{\lambda}{2}||x||^2 + \frac{1}{N}\sum_{i=1}^{N} l(\omega_i, z_i, x), \tag{2.2}$$

where $N \in \mathbb{N}$ is the number of samples, $\lambda > 0$ a regularization parameter, $w_i \in \mathbb{R}^n, i = 1, \ldots, N$ are the feature vectors of each sample, and $z \in \mathbb{R}^N$ is the vector of corresponding true labels, with $z_i \in \{-1, 1\}$ and x is the decision variable which represents a vector of coefficients for the classifier. The function $l(\omega_i, z_i, x)$ is the hinge loss defined as

$$l(\omega_i, z_i, x) := \max(0, 1 - z_i x^T \omega_i).$$

The loss l is a non-negative convex function of x which measure the discrepancy between w_i and the predictions arising from using x.

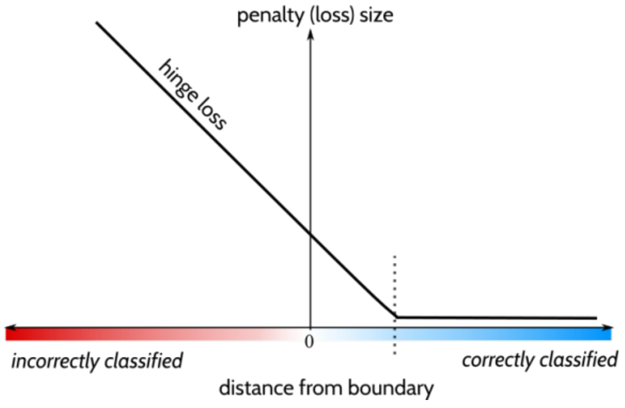

Figure 2.3: Hinge loss function, Source: [3]

On Figure 2.3 the x-axis represents the distance from the boundary, and the y-axis represents the penalty that the function will incur depending on its distance. The dotted line on the x-axis represents the number 1. This means that when the distance from the boundary is greater than or at 1, our loss size is 0. If the distance from the boundary is 0, meaning that the instance is exactly on the boundary, then we incur a loss size of 1. We can see that correctly classified points will have a small loss size, while incorrectly classified instances will have a high loss size. A negative distance from the boundary is resulting a high hinge loss value. This essentially means that we are on the wrong side of the boundary, and that the instance will be classified incorrectly. On the other hand, a positive distance from the boundary yields a low hinge loss, or no hinge loss at all. The further we are away from the boundary to the right side, the lower our hinge loss will be.

Let us remind ourselves what was our initial goal. It was to find the coefficient vector x, which we achieve by solving the following optimization problem

$$\min_x f(x) := \min_x \left(\frac{\lambda}{2} ||x||^2 + \frac{1}{N} \sum_{i=1}^{N} l(\omega_i, z_i, x) \right).$$

The objective function that we consider is nonsmooth and usually very costly to evaluate since N can be very large. Thus, the Variable Sample Size (VSS) is commonly used and the function f is usually approximated by the Sample Average Approximation (SAA) function. For a given sample size N, the SAA approximate objective function for $N_k \leq N$ is defined as

$$f_{N_k}(x) = \frac{1}{N_k} \sum_{i=1}^{N_k} f_i(x) = \frac{\lambda}{2} ||x||^2 + \frac{1}{N_k} \sum_{i=1}^{N_k} \max(0, 1 - z_i x^T \omega_i). \tag{2.3}$$

In order to determine the subgradient of the SAA function, we define the set of points which are in error ($\mathcal{E}_k$), on the margin ($\mathcal{M}_k$) and well classified ($\mathcal{W}_k$) are respectively

$$\begin{aligned} \mathcal{E}_k &:= \{i \in \{1, \ldots, N_k\} : 1 - z_i x^T \omega_i > 0\} \\ \mathcal{M}_k &:= \{i \in \{1, \ldots, N_k\} : 1 - z_i x^T \omega_i = 0\} \\ \mathcal{W}_k &:= \{i \in \{1, \ldots, N_k\} : 1 - z_i x^T \omega_i < 0\}. \end{aligned}$$

The subgradient is defined as

$$\partial f_{N_k}(x) = \lambda x - \frac{1}{N_k} \sum_{i=1}^{N_k} \beta_i z_i \omega_i = \overline{x} - \frac{1}{N_k} \sum_{i \in \mathcal{M}_k} \beta_i z_i \omega_i, \tag{2.4}$$

where $\overline{x} := \lambda x - \frac{1}{N_k} \sum_{i \in \mathcal{E}_k} z_i \omega_i$ and

$$\beta_i := \begin{cases} 1, & i \in \mathcal{E}_k \\ [0,1], & i \in \mathcal{M}_k \\ 0, & i \in \mathcal{W}_k. \end{cases}$$

One thing that is necessary to mention is that the `descentDirection` algorithm that will be explained in detail in section 3.2 requires an oracle that provides $\arg\sup_{g \in \partial f_{N_k}(x_k)} g^T p$ for a given direction p. For this problem we can implement such an oracle at a computational cost of $\mathcal{O}(n|\mathcal{M}_k|)$, where n is dimension of direction p and $|\mathcal{M}_k|$ the number of current margin points. Using the subgradient from 2.4 we get

$$\sup_{g \in \partial f_{N_k}(x_k)} g^T p = \sup_{\beta_i, i \in \mathcal{M}_k} \left(\overline{x}_k - \frac{1}{N_k} \sum_{i \in \mathcal{M}_k} \beta_i z_i \omega_i \right)^T p$$

$$= \overline{x}_k^T p - \frac{1}{N_k} \sum_{i \in \mathcal{M}_k} \inf_{\beta_i \in [0,1]} (\beta_i z_i \omega_i^T p),$$

where $\overline{x}$ is the same as above mentioned. Since, for a given direction p the first term of the right-hand side is a constant, the supremum is attained when we set $\beta_i, \forall_i \in \mathcal{M}_k$ in the following way

$$\beta_i := \begin{cases} 0, & z_i \omega_i^T p \geq 0, \\ 1, & z_i \omega_i^T p < 0. \end{cases}$$

2.2 Hinge loss for anomaly detection

Datasets often have outliers. Detecting such points can be of great importance. For example we can prevent some unwanted malfunctioning or mitigate some potential losses in systems. There are various method for achieving this goal. One such method is the "one class SVM", which is an unsupervised machine learning method because we don't have any information about training labels. As in standard binary classification SVM are also trying to find the best separating hyperplane with the largest margin, but we will have to make some adjustiment to this problem.

To find the optimal hyperplane we have to solve the following minimization problem

$$\min_{x,r} \frac{1}{2}||x||^2 - r, \text{ subject to } x^T\omega_i \geq |r|, i = 1, \ldots, N$$

The samples ω_i for which $x^T\omega_i \leq |r|$ is true, will be considered as anomalies. Adding this regularization parameter and the penalization component, we can rewrite the problem as

$$\min_{x,r} f(x,r) = \min_{x,r} \left(\frac{\lambda||x||^2}{2} - \lambda r + \frac{1}{N}\sum_{i=1}^{N} \max\{0, r - x^T\omega_i\} \right)$$

The subgradients have the following form

$$\partial_x f(x,r) = \lambda x - \frac{1}{N}\sum_{i=1}^{N} \beta_i\omega_i = \lambda x - \frac{1}{N}\sum_{i\in\mathbb{E}'} \omega_i - \frac{1}{N}\sum_{i\in\mathbb{M}'} \beta_i\omega_i$$

$$\partial_r f(x,r) = -\lambda + \frac{1}{N}\sum_{i\in\mathbb{E}'} 1 + \frac{1}{N}\sum_{i\in\mathbb{M}'} \beta_i$$

$$\partial f(x,r) = \begin{bmatrix} \partial_x f(x,r) \\ \partial_r f(x,r) \end{bmatrix}$$

where,

$$\mathbb{E}' = \{i \in \{1, \ldots, N\}, r - x^T\omega_i > 0\}$$

$$\mathbb{M}' = \{i \in \{1, \ldots, N\}, r - x^T\omega_i = 0\}$$

$$\mathbb{W}' = \{i \in \{1, \ldots, N\}, r - x^T\omega_i < 0\}$$

and β_i is

$$\beta_i := \begin{cases} 1, & i \in \mathbb{E}' \\ [0,1], & i \in \mathbb{M}' \\ 0, & i \in \mathbb{W}'. \end{cases}$$

Same as in the L_2 - regularized binary hinge loss case, here for the anomaly detection we are also in the need for the the $\sup_{g \in \partial f(x,r)} g^T p$. The only difference that occurs is that we are optimizing for two unknown vectors x and r, so we have the following slightly complicated form of the desired quantity

$$\sup_{g \in \partial f(x,r)} g^T p = \lambda p_x^T x - \frac{1}{N}\sum_{i \in \mathbb{E}} p_x^T \omega_i - \lambda p_r + p_r \frac{|\mathbb{E}|}{N} + \frac{1}{N}\sum_{i \in \mathbb{M}} \beta_i^*(p_r - p_x^T\omega_i)$$

with

$$\beta_i^* := \begin{cases} 1, & p_r - p_x^T\omega_i > 0, \\ 0, & p_r - p_x^T\omega_i \leq 0. \end{cases}$$

where $\beta_i^*(p_r - p_x^T\omega_i) = \sup_{\beta_i \in [0,1]} \beta_i(p_r - p_x^T\omega_i)$. We can replace sup with max and get

$$\arg\max_{g \in \partial f(x,r)} g^T p = \begin{bmatrix} -\lambda + \frac{|\mathbb{E}|}{N} + \frac{1}{N}\sum_{i \in \mathbb{M}} \beta_i^* \\ \\ \lambda x - \frac{1}{N}\sum_{i \in \mathbb{E}} \omega_i - \frac{1}{N}\beta_i^*\omega_i \end{bmatrix}$$

Chapter 3

The algorithm

In this chapter, we are going to present the application of the method on the real world problems. This method has three main ingredients: 1) it uses BFGS update rule adapted to the VSS framework to utilize some second order information; 2) it uses descent directions so that the line search is well defined; 3) it uses IR approach to update the sample size used for the SAA estimators. We describe these ingredients in the sequel.

3.1 BFGS update

In numerical optimization Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is an iterative method for solving unconstrained nonlinear optimization problems and it belongs to Quasi-Newton methods. It was named after Charles George Broyden, Roger Fletcher, Donald Goldfarb and David Shanno who each came up with the algorithm independently in 1970. The algorithm determines the descent direction by preconditioning the gradient with curvature information. It does so by gradually improving an approximation to the hessian matrix of the loss function. Since the updates of the BFGS curvature matrix do not require matrix inversion, its computational complexity is only $\mathcal{O}(n^2)$ compared to $\mathcal{O}(n^3)$ in Newton's method [4]. The main advantage of this algorithm is that it converges fast to a solution if the objective function is convex. Let us denote by H_k the approximation of the Hessian matrix. Then, quasi-Newton direction d^k satisfies

$$H_k d^k = -\nabla f(x^k)$$

Assume that we have an approximation H_k and that we performed the iteration to obtain x_{k+1}. Then we need to update the Hessian approximation H_{k+1}. One requirement is that H_{k+1} satisfies the secant equation $H_{k+1}s^k = y^k$

where $s^k = x^{k+1} - x^k$ and $y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$ and f is the objective function.

The secant equation does not determine an unique H_{k+1}. Therefore, besides the symmetry which is a natural requirement for Hessian approximation, other conditions are imposed. The least-change update condition is the most successful one and it states that the next approximation matrix should be as close as possible to the current approximation. Therefore H_{k+1} is a solution of the following optimization problem

$$\min ||H - H_k|| \text{ subject to } H^T = H \ , \ Hs^k = y^k$$

Clearly, the solution of the above problem depends on the norm we use in the objective function. The BFGS formula is given as

$$H_{k+1} = H_k + \frac{y^k (y^k)^T}{(y^k)^T s^k} - \frac{H_k s^k (s^k)^T H_k}{(s^k)^T H_k s^k}$$

By using the SMW (Sherman-Morrison-Woodbury) formula, one can also get update for the inverse Hessian matrix approximation, i.e.,

$$B_{k+1} = B_k - \frac{B_k y^k (y^k)^T B_k}{(y^k)^T B_k y^k} + \frac{s^k (s^k)^T}{(y^{kT}) s^k}$$

Above we described the basic case where the function is smooth, but our problem requires to work with subradients and the approximate functions, so we present the BFGS update that can be used in the considered environment. To get the descent direction p_k we need the corresponding BFGS matrix B_k.

$$p_k = -B_k(x_k) g_k.$$

Let $y_k := g_{k+1} - g_k$ and $s_k = x_{k+1} - x_k$ where $g_k \in \partial f_{N_k}(x_k), g_{k+1} \in \partial f_{N_{k+1}}(x_{k+1})$ are the subgradients. The matrices have to be uniformly positive definite so, we skip the BFGS update if $s_k^T y_k \geq 10^{-4} ||y_k||^2$ and we start with B_0 as an identity matrix. The BFGS update can be represented by:

$$B_{k+1} = (I - \rho_k s_k y_k^T) B_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \tag{3.1}$$

where

$$\rho_k = \frac{1}{y_k^T s_k}.$$

3.2 Descent directions

The main procedure uses line search method, and thus it requires a descent direction. As for the start, let us define what is a descent direction for nonsmooth case. We say that p is descent direction for a function ψ at the point $x \in \mathbb{R}^n$ if

$$g^T p < 0 \text{ for all } g \in \partial\psi(x),$$

or equivalently, if the following holds

$$\sup_{g \in \partial\phi(x)} g^T p < 0. \tag{3.2}$$

The pseudo-quadratic model of ψ at $x \in \mathbb{R}^n$ is given by

$$Q_k(p) = \phi(x) + Y(p),$$

where

$$Y(p) = \frac{1}{2} p^T B^{-1} p + \sup_{g \in \partial\phi(x)} g^T p, \tag{3.3}$$

and $B \in \mathbb{R}^{n \times n}$ is a nonsingular matrix. The algorithm below will return a decent direction p for $\psi(x)$.

The input parameters are a subgradient $\tilde{g}_0 \in \partial(x)$, tolerance $\epsilon \geq 0$, iteration bound $i_{\max} \in \mathbb{N}$ and the BFGS matrix. It is assumed that $\tilde{g}_{i+1} = \arg\sup_{g \in \partial\psi(x)} g^T p_i$ is provided, and the first subgradient $\tilde{g}_0 \in \partial\psi(x)$ is arbitrarily chosen. In general, providing $\tilde{g}_{i+1} = \arg\sup_{g \in \partial\psi(x)} g^T p_i$ is extremely hard, but it can be determined analytically for the problems that we consider such as Hinge loss and IoT as explained in the previous section. Below we present the algorithm which returns a descent direction p at the point x.

Algorithm 1 [21]: $p =$`descentDirection` $(\tilde{g}_0 \in \partial\psi(x), \epsilon, i_{max}, B)$

```
S0 Initialize $i = 0, \bar{g}_0 = \tilde{g}_0, p_0 = -B\tilde{g}_0$

S1 Calculate the next subgradient $\tilde{g}_1 = \arg\sup_{g \in \partial\psi(x)} g^T p_0$

S2 Compute $\epsilon_0 := p_0^T \tilde{g}_1 - p_0^T \bar{g}_0$
```

```
S3 While  $(\tilde{g}_{i+1}^T p_i > 0$ or $\epsilon_0 > \epsilon)$ and $\epsilon_i > 0$ and $i < i_{max}$ do
      $\mu^* := \min \left[1, \frac{(\bar{g}_i - \tilde{g}_{i+1})^T B \bar{g}_i}{(\bar{g}_i - \tilde{g}_{i+1})^T B (\bar{g}_i - \tilde{g}_{i+1})}\right]$
      $\bar{g}_{i+1} = (1-\mu^*)\bar{g}_i + \mu^* \tilde{g}_{i+1}$
      $p_{i+1} = (1-\mu^*)p_i - \mu^* B \tilde{g}_{i+1}$
      $\tilde{g}_{i+2} = \arg \sup_{g \in \partial\psi(x)} g^T p_{i+1}$
      $\epsilon_{i+1} = \min_{j \le i+1} \left[p_j^T \tilde{g}_{j+1} - \frac{1}{2}(p_j^T \bar{g}_j + p_{i+1}^T \bar{g}_{i+1})\right]$
      $i := i + 1$
End

S4 Compute $p = \arg\min_{j \le i} Y(p_j)$

S5 If $\sup_{g \in \partial\psi(x)} g^T p < 0$ then return $p$, else return failure.
```

Let us briefly explain some key points of this algorithm. For nonsmooth functions the direction $p_k = -B_k g_k$ may not always fulfill the descent condition 3.2, where B_k is a BFGS matrix that will be explained in section below. If the desired direction is not descent the process in line search algorithm will not be able to find a suitable step size $\alpha > 0$. For this aim the iterative approach for finding this direction was employed, namely the **Algorithm 1**. Our goal is to minimize the speudo-quadratic model 3.3. This problem can be solved using quadratic programming, but it would be computationally expensive, instead we adopt an alternative approach which does not solve this problem to optimality. The key idea is to write the proposed direction at ireration $k+1$ as a convex combionation of p_k and $-B_k g_{k+1}$, more precisely $p_{i+1} = (1-\mu^*)p_i - \mu^* B \tilde{g}_{i+1}$ where the coefficient μ^* is a dual variable of the subproblem. More details about this algorithm can we found in [21]. It can be be proven that at a non-optimal iterate a direction finding tolerance $\epsilon \ge 0$ exists such that the search direction produced by **Algorithm 1** is a descent direction. Also that the algorithm converges to a solution with precision ϵ in $O(1/\epsilon)$ iterations.

3.3 Inexact Restoration approach

In this section we are putting together the above presented `descentDirection` and the BFGS algorithm in action.

Algorithm 2: IRNS

S0 Given $x_0 \in \mathbb{R}^n, N_0 \in \mathbb{N}, \theta_0 \in (0,1), m, \beta, \gamma, \overline{\gamma} > 0, \epsilon \geq 0, i_{\max} \in \mathbb{N}$.
Set $k = 0, B_0 = I$ and compute $g_0 \in \partial f_{N_0}(x_0)$.

S1 Restoration phase. Find $\tilde{N}_{k+1} \geq N_k$ such that

$$h(\tilde{N}_{k+1}) \leq r h(N_k), \tag{3.4}$$

$$f_{\tilde{N}_{k+1}}(x_k) - f_{N_k}(x_k) \leq \beta h(N_k). \tag{3.5}$$

S2 If

$$\Phi(x_k, \tilde{N}_{k+1}, \theta_k) - \Phi(x_k, N_k, \theta_k) \leq \frac{1-r}{2}(h(\tilde{N}_{k+1}) - h(N_k)) \tag{3.6}$$

set $\theta_{k+1} = \theta_k$. Else

$$\theta_{k+1} := \frac{(1+r)(h(N_k) - h(\tilde{N}_{k+1}))}{2[f_{\tilde{N}_{k+1}}(x_k) - f_{N_k}(x_k) + h(N_k) - h(\tilde{N}_{k+1})]}. \tag{3.7}$$

S3 Optimization Phase.

a) Choose $N_{k+1} \leq \tilde{N}_{k+1}$ such that for
$p_k =$ `descentDirection` $(\tilde{g}_0 \in \partial f_{N_{k+1}}(x_k), \epsilon, i_{max}, B_k)$

and for $\alpha \in (0, \tau_k]$ for some $\tau_k > 0$ we have

$$f_{N_{k+1}}(x_k + \alpha p_k) - f_{\tilde{N}_{k+1}}(x_k) \leq -\gamma \alpha ||p_k||^2, \tag{3.8}$$

$$h(N_{k+1}) \leq h(\tilde{N}_{k+1}) + \overline{\gamma} \alpha^2 ||p_k||^2. \tag{3.9}$$

b) Find $\alpha_k \in (0,1]$ as large as possible such that (3.8) and (3.9) hold for $\alpha = \alpha_k$ and in addition

$$\Phi(x_k + \alpha_k p_k, N_{k+1}, \theta_{k+1}) - \Phi(x_k, N_k, \theta_{k+1}) \leq \frac{1-r}{2}(h(\tilde{N}_{k+1}) - h(N_k)). \tag{3.10}$$

S4 Set $s_k = \alpha_k p_k$ and $x_{k+1} = x_k + s_k$.

S5 Choose a subgradient $g_{k+1} \in \partial f_{N_{k+1}}(x_{k+1})$ and compute $y_k := g_{k+1} - g_k$.

If $s_k^T y_k \geq m||y_k||^2$, update B_{k+1} by (3.1).
Else $B_{k+1} = B_k$.

S6 Set $k := k + 1$ and go to S1.

Now that we have seen all parts of the main algorithm, let us dive deep into and give a detailed explanations of each step.

S0: Before the first step we have to initialize a number of parameters in orders to start the algorithm. Although some other choices are possible, we opted for following ones in the applications presented in the following chapter: x_0 is a vector with random numbers generated from uniform distribution of length n which is number of features, N_0 is the first sample size which is obtained by taking 10% of the full sample size. Other parameters are: the penalty parameter $\theta_0 = 0.9$, regularization parameter $l = 10^{-5}, m = 10^{-4}, \gamma = 10^{-4}, \gamma_k = 1, r = 0.95$, the initial BFGS matrix $B_0 = I$, iteration limit $i_{\max} = 100$, direction finding tolerance $\epsilon = 10^{-4}$. Notice that ϵ and $i_{\max}$ are used for Algorithm 1 which is called within the main IRNS Algorithm.

S1: Step one is the restoration phase where the feasibility is improved. Here we want to find the next sample size denoted by $\tilde{N}_{k+1}$, it has to be larger or equal then the previous sample size i.e. $\tilde{N}_{k+1} \geq N_k$. We obtain it in the following way

$$N(1 - rh(N_k)) \leq \tilde{N}_{k+1} \tag{3.11}$$

for $r < 1$, where we choose h to be $(N - N_k)/N$. Inequality 3.11 is just the explicit expression of 3.4. The value of our objective function $f_{\tilde{N}_{k+1}}(x_k)$ can increase with respect to f_{N_k} but only by at most $\beta h(N_k)$ which is told by the second inequality in this step 3.5. We can look at this condition in a way that the optimality can deteriorate with respect to the previous iteration, but it is controlled by the function h. In other words, if the approximation of the objective function is looser for smaller sample sizes the deterioration of optimality can be relatively large.

S2: In the second step, we update the penalty parameter $\theta_k \in (0, 1)$ with the help of merit function

$$\Phi(x, N, \theta) := \theta f_N(x) + (1 - \theta)h(N)$$

This parameter plays the role in giving different weights to the objective function. If inequality 3.6 is not satisfied this parameter is updated by 3.7. It can be proved that θ_k is bounded away from zero under some standard conditions.

S3: Step S3 tries to decrease the computation costs if such decrease is relevant. So, we choose the sample size $N_{k+1} \leq \tilde{N}_{k+1}$. Then we calculate the descent direction p_k with help of function `descentDirection` for the function $f_{N_{k+1}}$ at the current iteration x_k. There is no predetermined rule for finding N_{k+1} only that it has to be less or equal then $\tilde{N}_{k+1}$. The most efficient choice is unknown and it is most likely dependent on a given problem. When S3 a) is done we perform a line search backtracking for the merit function ϕ along the descent search direction p_k. The backtracking process is performed in the following way. We estimate the sample size lower bound N_{k+1}^{trial} derived from (3.10). The value of N_{k+1}^{trial} is calculated as follows:

$$N_{k+1}^{trial} := N_k + \frac{1-r}{2} \cdot \frac{\tilde{N}_{k+1} - N_k}{1 - \theta_{k+1}} - \hat{\theta}_{k+1}(\gamma\alpha||p_{k-1}||^2 - f_{\tilde{N}_{k+1}}(x_k) + f_{N_k}(x_k)), \tag{3.12}$$

where $\hat{\theta}_{k+1} = N \cdot \frac{\theta_{k+1}}{1-\theta_{k+1}}$. Than we get three candidates for N_{k+1} which are:

$$N_{k+1} \in \{\left\lceil N_{k+1}^{trial} \right\rceil, \left\lceil (N_{k+1}^{trial} + \tilde{N}_{k+1})/2 \right\rceil, \tilde{N}_{k+1}\}. \tag{3.13}$$

In the first iteration we set $\alpha_k = 0.5^j$ for counter $j = 0$, we try all three candidates above mentioned and when all three inequalities namely 3.8, 3.9 and 3.10 are satisfied we return N_{k+1} and α_k. If none of three candidates satisfies the condition we increase j by one and repeat the whole process until we find one that satisfies the conditions.

We implemented also a safeguard which ensures that the number of samples can't be less than the initial sample size. This was necessary because without this safeguard sometimes N_{k+1}^{trial} tend to be negative number which is not acceptable and makes no sense for the algorithm.

S4: Here we perform a quick update to our solution iterate x_{k+1} with obtained step size α_k and descent direction p_k from previous steps.

S5: The goal of step S5 is to calculate the next g_{k+1} and also y_k which is necessary to update the BFGS matrix by 3.1. If the current B_k is not uniformly positive definite, in other words $s_k^T y_k \geq m||y_k||^2$ is not satisfied we skip the BFGS update and use the same one for the next iteration. Than we update our iteration counter k by one and go to S1.

At this point we are familiar with the details and the functionality of the algorithm, but what about some theoretical facts? The following assumption describes some properties of our problem and are necessary for the corollary which states the convergence result for the finite sum case.

Assumption 3.1. *For any give N, x and B such that $mI \preceq B(x) \preceq MI$, for some positive and bounded constants $m \leq M$ we can compute a direction $p_N \in \mathbb{R}^n$ such that*

$$p_N(x) = -B(x)\bar{g}_N(x) \quad \textit{and}$$

$$\sup_{g \in \partial f_N(x)} g^T p_N(x) \leq -\frac{m}{2}||\bar{g}_N(x)||^2, \quad \bar{g}_N(x) \in \partial f_N(x).$$

Parameter β which is used in inequality 3.5 can be arbitrary large, but has to be finite number. In our case, finite sums, we can prove that under the standard conditions such β exists. Since we do not impose differentiablity of the objective function nor any other special property, the following assumption is necessary.

Assumption 3.2. *Suppose that there exists β such that inequality 3.5 holds for each k.*

Corollary 3.1. *Let Assumptions 3.1 and 3.2 hold and assume $\sum_k \alpha_k = \infty$. If $f = f_{N_{\max}}$ and $f_i, i = 1, \ldots, N_{\max}$ are continuous and strongly convex, then $\lim_{k\to\infty} x_k = x^*$, where x^* denotes the solution of the finite sum problem. Moreover, if $\alpha_k \geq \bar{\alpha} > 0$ for all $k \in \mathbb{N}$, then the worst-case complexity is of order $\mathcal{O}(\epsilon^{-2})$.*

Chapter 4

Numerical results

Here we test the presented IRNS algorithm with variable sample size scheme on a non-smooth convex finite sum problems i.e., bounded sample size with real world data. We are interested in quantifying the computation cost savings obtained from this approach.

In this chapter we will test the performance of the algorithms on three supervised problems namely: Mushroom, Adult, Splice and an unsupervised IoT problem. Beside this, we are interested in computation saving which is measure by FEV - the number of scalar products. It will be tested against the heuristic and the full sample counterpart, showing the advantages of the adaptive sample size IRNS scheme. In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

True positive (TP) - Correctly indicates the presence of a condition or characteristic. True negative (TN) - Correctly indicates the absence of a condition or characteristic. False positive (FP) - Wrongly indicates that a particular condition or attribute is present. False negative (FN) - Wrongly indicates that a particular condition or attribute is absent.

From these, we derive the standard measures:

$$\text{Precision (P)} = \frac{TP}{TP + FP}$$

$$\text{Recall (R)} = \frac{TP}{TP + FN}$$

$$\text{Accuracy (A)} = \frac{TP + TN}{FP + FN + TN + TP}$$

$$\text{F1 score (F1)} = \frac{2 \cdot P \cdot R}{P + R}$$

4.1 Binary classification

This is a machine learning framework and considers L_2-regularized binary hinge loss functions for binary classification. The problem is of the form

$$f(x) := \frac{\lambda}{2}||x||^2 + \frac{1}{N}\sum_{i=1}^{N} l(\omega_i, z_i, x)$$

The columns of ω are different attributes, and the samples are represented by rows. The vector $x \in \mathbb{R}^n$ is the vector of weight coefficient, that we are searching for in our algorithm. The data [6] and [18] did not need any additional preprocessing, that means after checking for various properties like missing data everything was ready to pass on to the algorithm. The parameters of the algorithms are $\theta_0 = 0.9, r = 0.95, \tilde{\gamma} = 1$ and $\gamma = 10^{-4}$. The function h is defined as $h(N_k) = (N - N_k)/N$ with $N_0 = \lceil 0.1N \rceil$. The computation cost is measured by FEV which is is the number of scalar products. The algorithms are stopped when the maximum Max_{FEV} value is attained. The regularization constant is set to $\lambda = 10^{-5}$ and the initial point x_0 is random vector from uniform distribution. Other parameters are $i_{\max}$=100, $\epsilon = 10^{-4}$, $\overline{\gamma} = 1$, $\gamma = 10^{-4}$. The experiments were conducted on three datasets commonly used in binary classification studies, namely SPLICE, MUSHROOMS and ADULT9. Table 4.1 summarizes the properties of the datasets, where N is the number of data points including both training and testing sets and n is the dimension of the decision variable. We use the classical split, 80% of data as training set and the remaining 20% as testing set, where N_{train} and N_{test} are the cardinal numbers of those sets respectively.

	Dataset	N	n	N_{train}	N_{test}	Max_{FEV}
1	SPLICE [6]	3175	60	2540	635	10^5
2	MUSHROOMS [18]	8124	112	6500	1624	10^5
3	ADULT9 [6]	32561	123	26049	6512	10^6

Table 4.1: Properties of the datasets used in the experiments

Each dataset has been split into training and test set. The algorithm is using the training set to find the optimal solution, and it validates the predictions after each iteration with the test set.

4.1.1 Mushroom dataset

In Mushroom dataset we are given attributes of a mushrooms with 8124 samples, and 112 features, all of which only consists of 1 and -1 values. Our goal is to predict whether a given mushroom is edible (1) or not (-1). The algorithms are stopped when the maximum number of scalar products, Max_{FEV} is reached. We also track the value of the true relevant objective function $f_{Ntrain}(x_k)$ or $f_{Ntest}(x_k)$. If the predicted value was negative it was classified as (-1) edible and (1) if its poisonous.

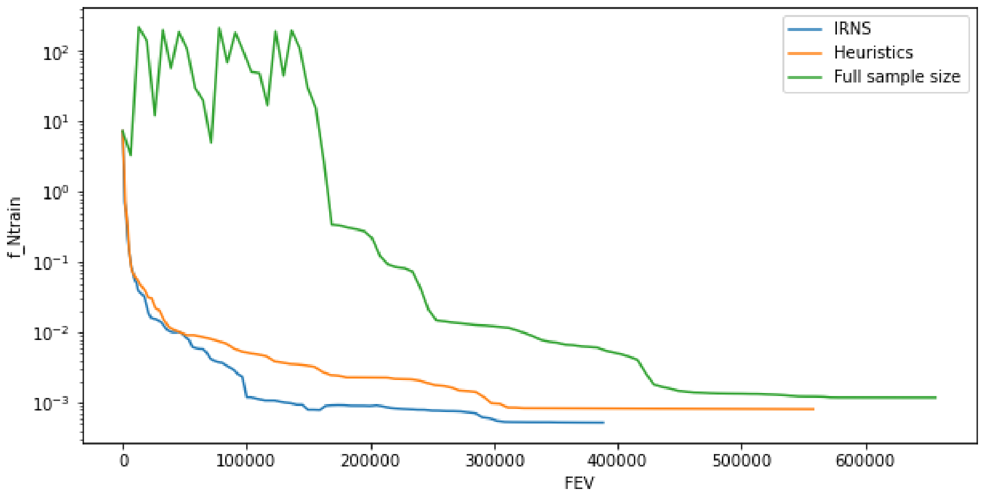

Figure 4.1: Mushrooms - Training loss versus FEV

It appears that the full sample method is unstable in the beginning, but we can notice as the iterations increase the value is decreasing to the levels of other two algorithms.

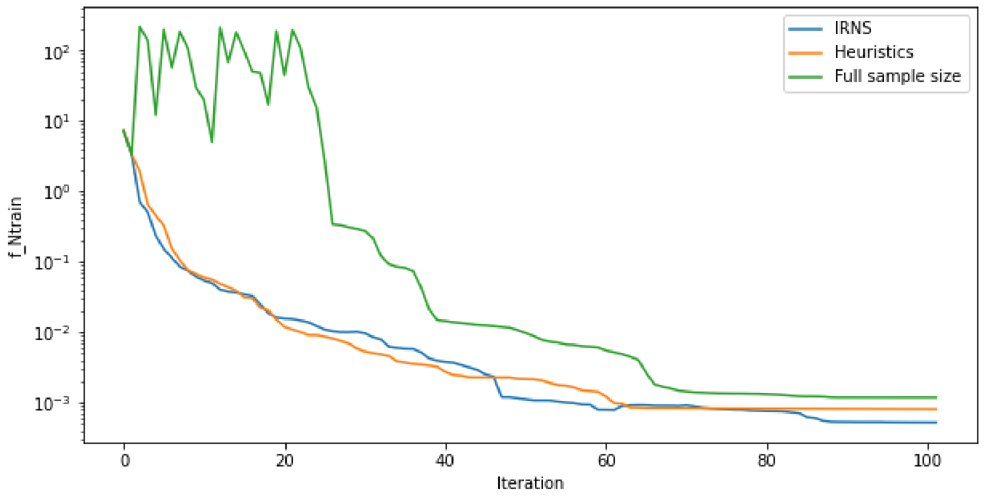

Figure 4.2: Mushrooms - Training loss versus iteration

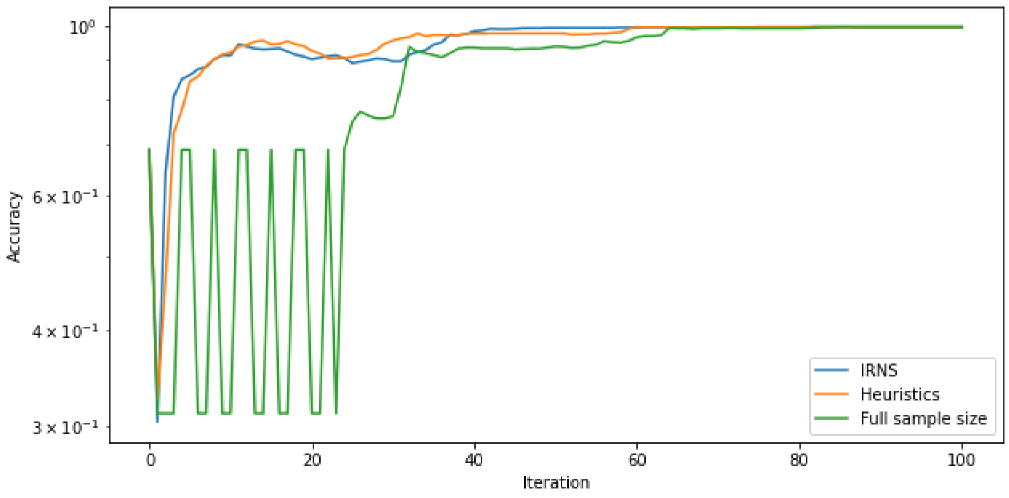

Figure 4.3: Mushrooms - Accuracy versus iteration

As mentioned above, the instability in the beginning of the full sample algorithm can be seen in the Accuracy and F1 Score metrics. But in the end all three algorithms achieves very good performance

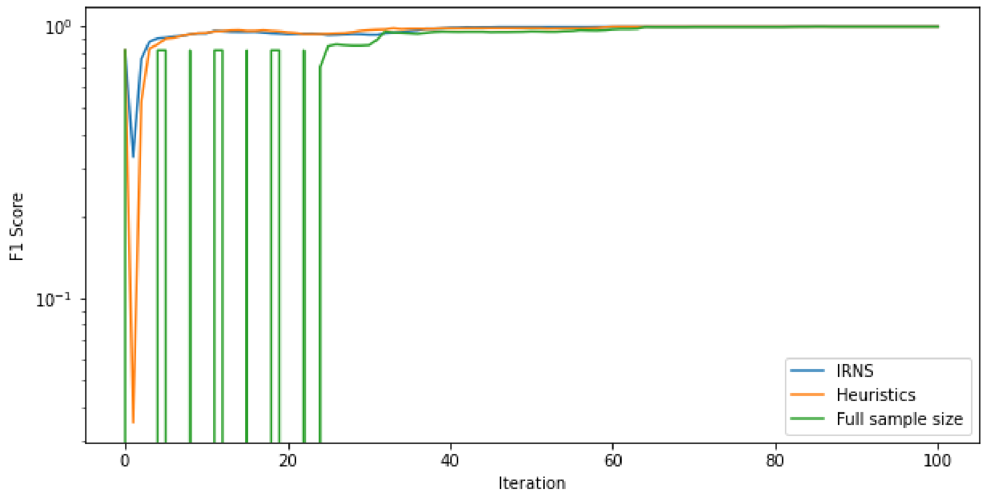

Figure 4.4: Mushrooms - F1 Score versus iteration

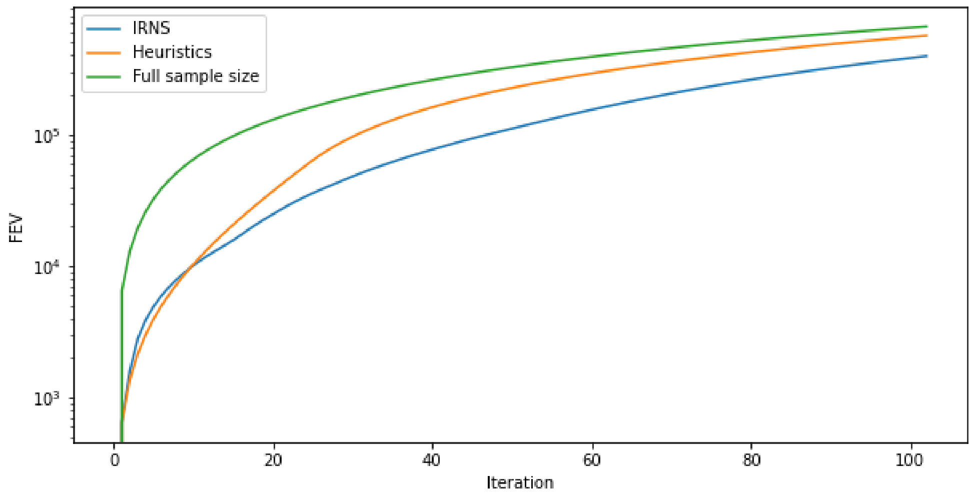

Figure 4.5: Mushrooms - FEV versus iteration

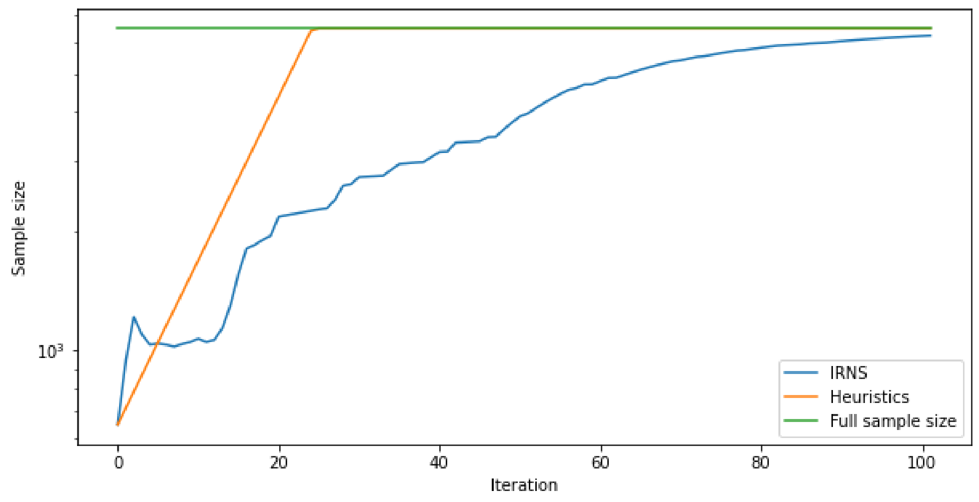

Figure 4.6: Mushrooms - Sample size representation

In Figure 4.6 we can see the sample sizes picked over the iteration. $\tilde{N}_{k+1}$ is represented with blue, and N_{k+1} with green and the heuristics i.e. when $N_{k+1} = \min\{\lceil 1.1 N_k \rceil, N\}$ with orange. We can see the advantages of this method from the figures, because it provides savings in the computational costs and achieving the same amount of accuracy as in the heuristic and full sample size approach.

4.1.2 Splice dataset

This dataset comes from the UCI repository of machine learning databases. The task is to recognize two types of splice junctions in DNA sequences; exon/intron (EI) or intron/exon (IE) sites. A splice junction is a site in a DNA sequence at which "superflous" DNA is removed during protein creation. Intron refers to the portion of the sequence spliced out while exon is the part of the sequence retained. We have 3175 samples and 60 features.

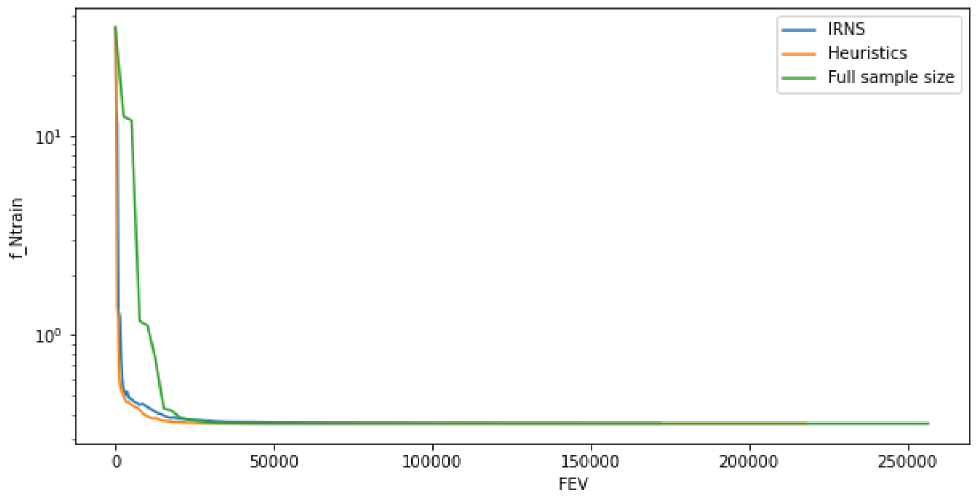

Figure 4.7: Splice - Training loss versus FEV

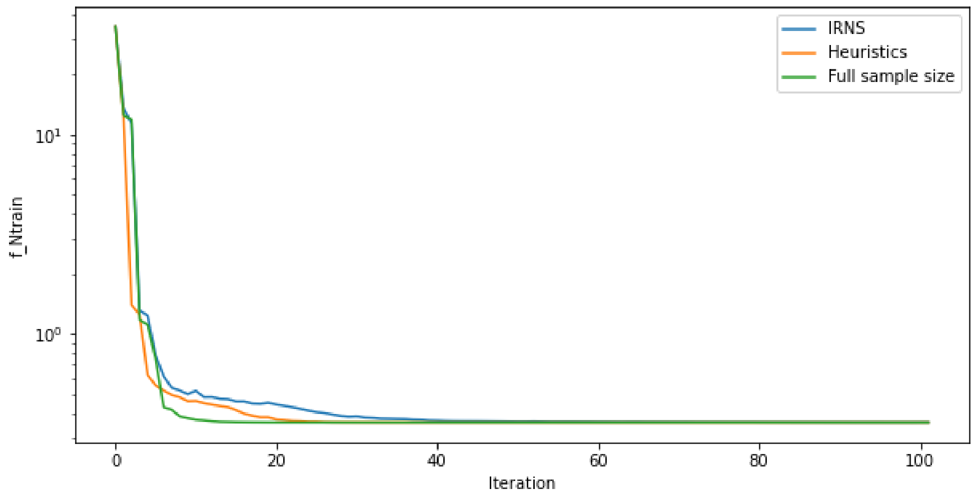

Figure 4.8: Splice - Training loss versus iteration

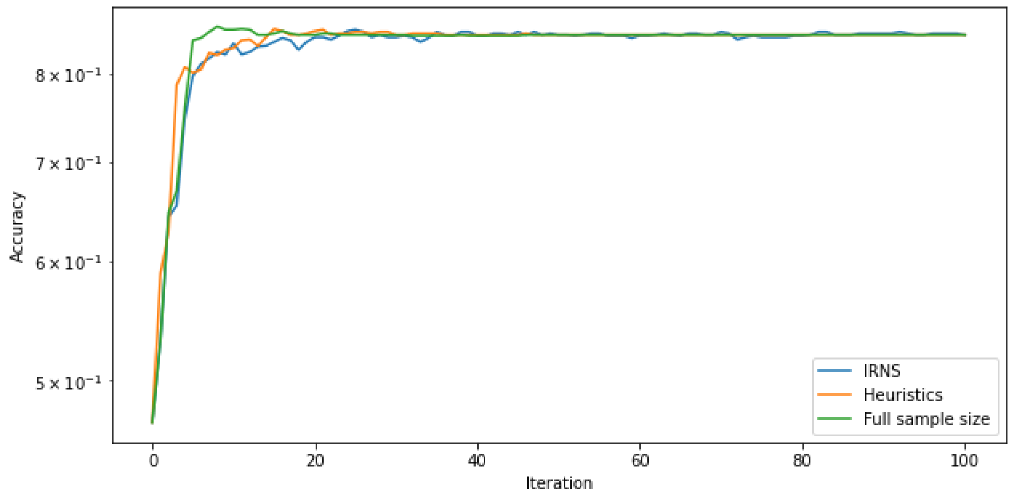

Figure 4.9: Splice - Accuracy versus iteration

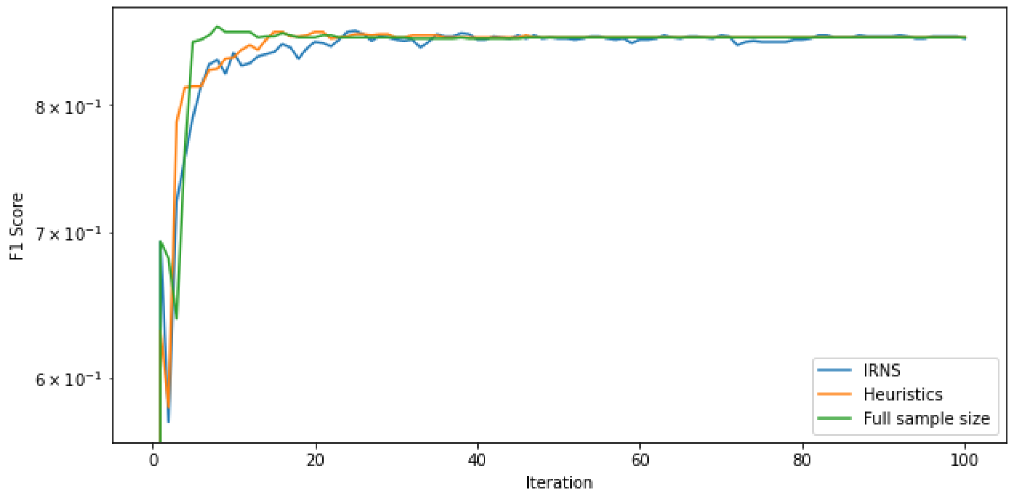

Figure 4.10: Splice - F1 Score versus iteration

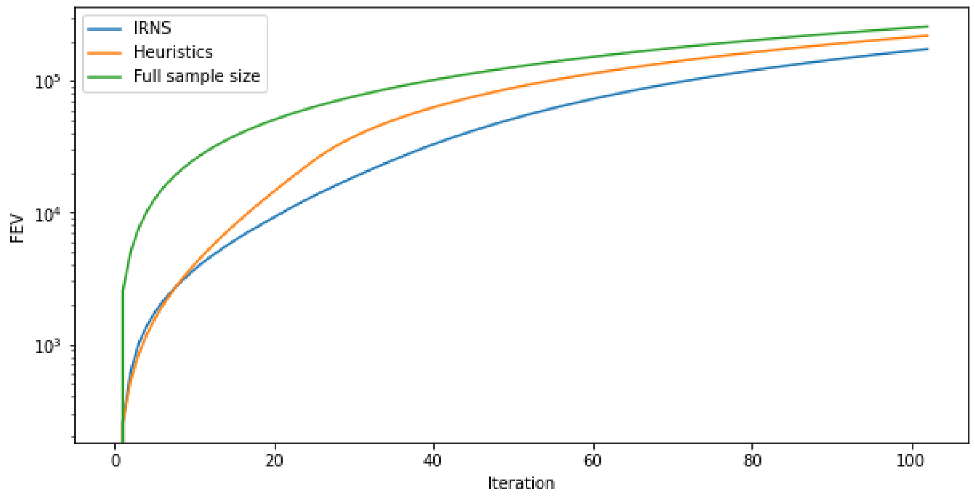

Figure 4.11: Splice - FEV versus iteration

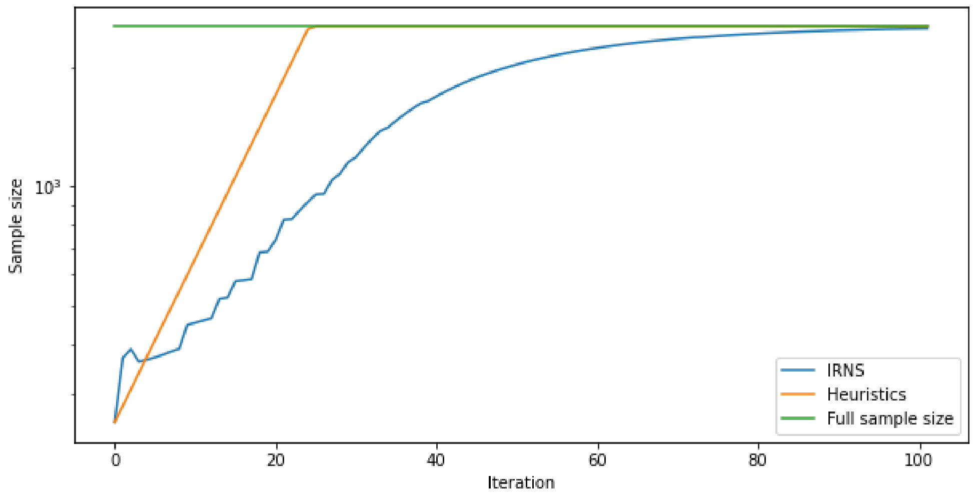

Figure 4.12: Splice - Sample size representation

4.1.3 Adult dataset

In Adult dataset, we want to determine whether the person makes more or less than 50000$ a year. We have 32561 samples and 123 features which are: age, working class, education etc.

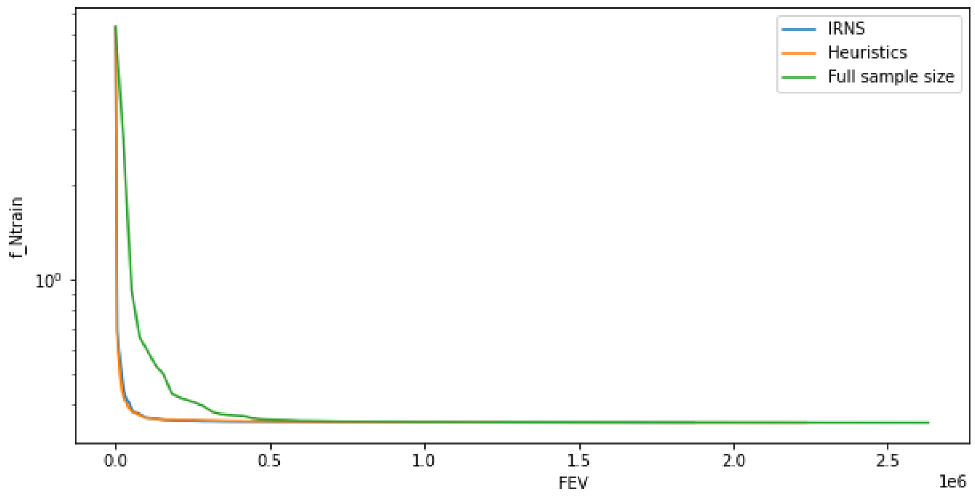

Figure 4.13: Adult - Training loss versus FEV

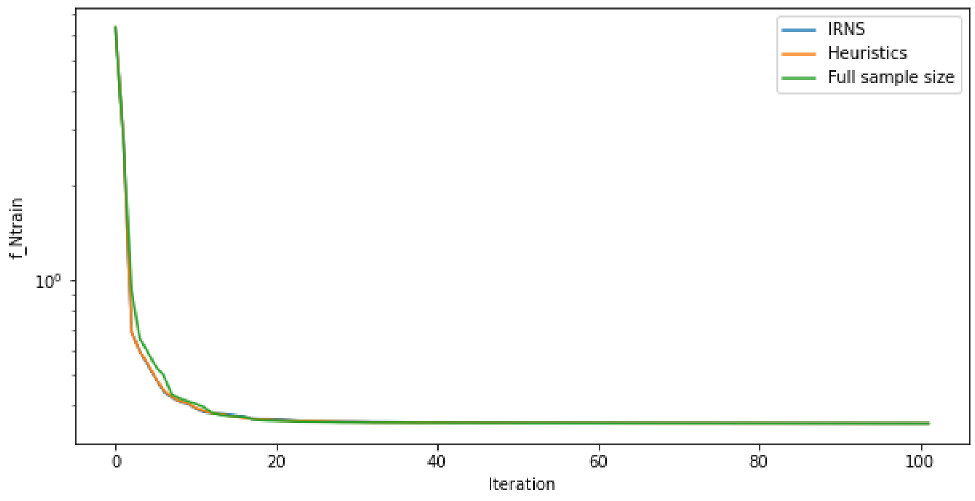

Figure 4.14: Adult - Training loss versus iteartion

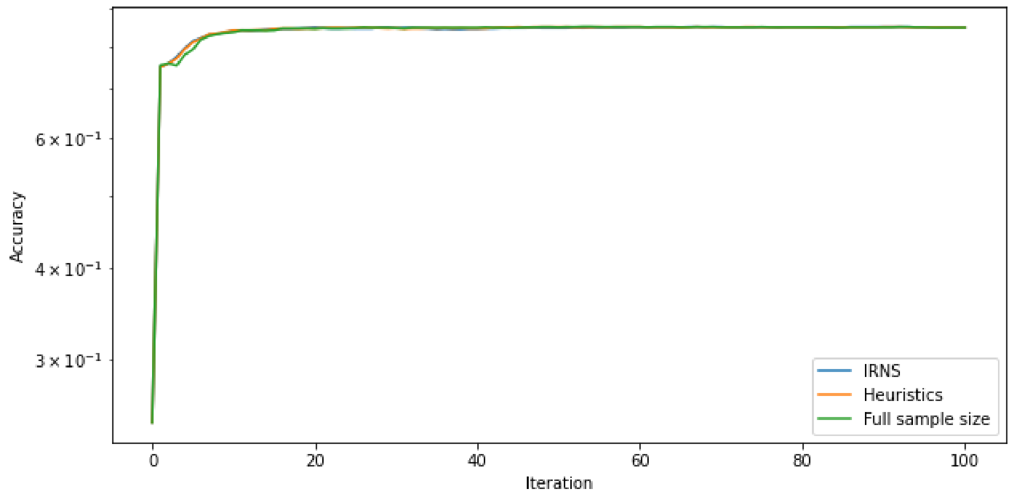

Figure 4.15: Adult - Accuracy versus iteration

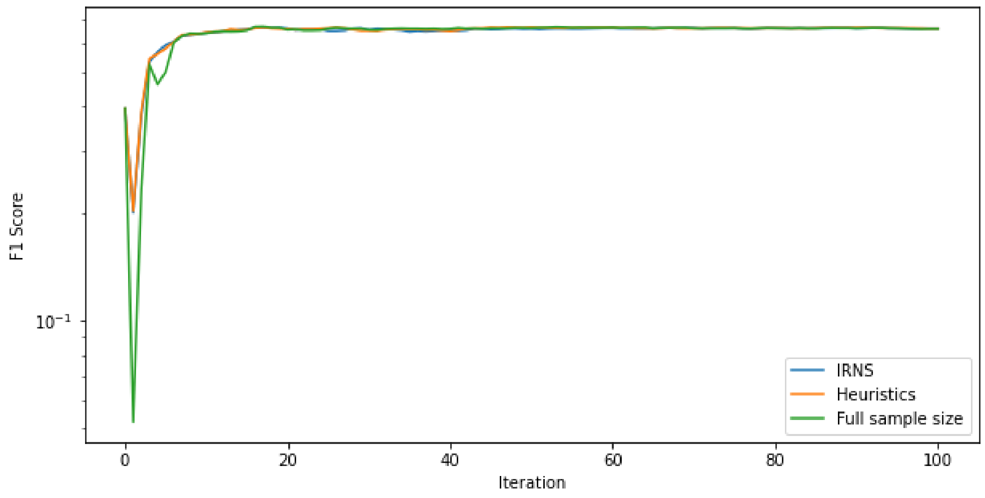

Figure 4.16: Adult - F1 Score versus iteration

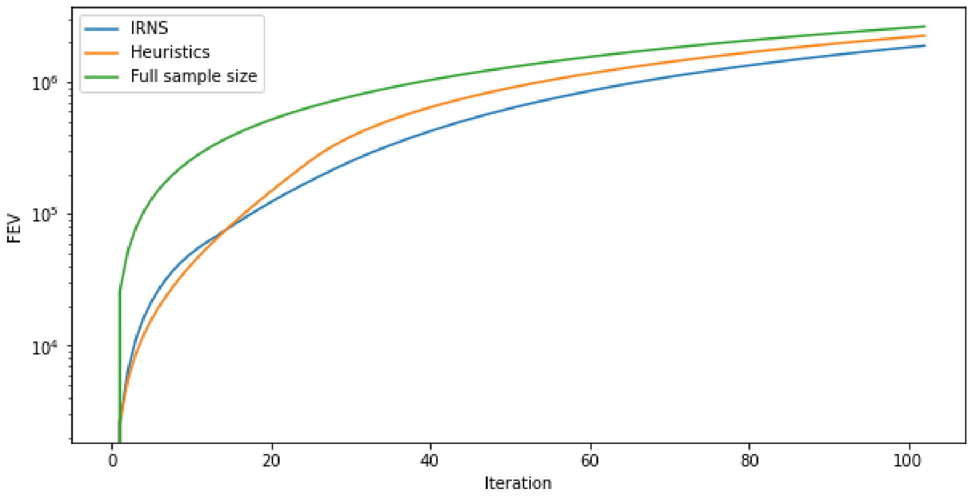

Figure 4.17: Adult - FEV versus iteration

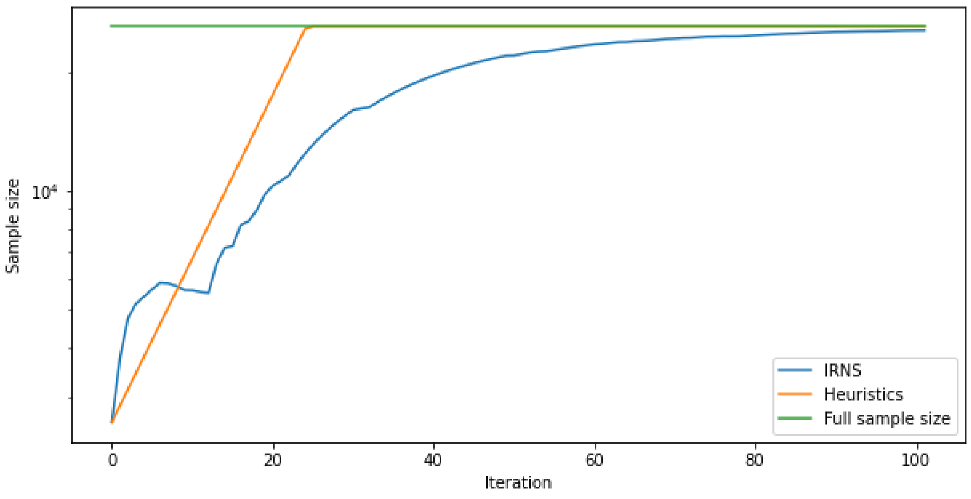

Figure 4.18: Adult - Sample size representation

4.2 Anomaly Detection for Cellular IoT

So far we have seen the application of the IRNS algorithm on three dataset, in this section we will consider an industrial problem. Finding anomalies is of great importance in industry because we can predict malfunctioning of some infrastructures, and thus prevent unnecessary losses. The goal of this section is to implement the algorithms on the dataset from [1] in order to find the anomalies. The research is part of the H2020 C4IIoT project-Cyber security 4.0: protecting the Industrial Internet of Things.

Lets talk more about how the data was acquired. It was generated using NB-IoT edge nodes and created a setup where an edge node has been attached to a box-shaped container inside a transport vehicle moving throught the city of Novi Sad. The devices were initially connected to the NB-IoT network, and they had the uninterrupted connectivity along their paths. The positions data was collected from GNSS module (timestamp, latitude, longitude, altitude, speed and number of satellites in range), as well as the outputs of the IMU (acceleration and magnetic field along the 3 spatial axes). The time resolution (sampling period) of the GNSS samples was approximately 10 s. The sampling period of the IMU was approximately 15 ms.

The dataset is arranged by timestamps, and it had been split to training and test sets, they have 12678 and 1571 samples respectively. There are 13 attributes which are numerical. For the test data ground truth anomalies are given. In the first few experiments the results that the algorithm were producing was not acceptable, so there had to be done some transformations on the data. Namely, normalization and standardisation. The normal scaling scales and translates each feature such that it is in the given range on the training set. We used $(0, 1)$ range, and it is calculated as: $z = (x - x_{\min})/(x_{\max} - x_{\min})$. The standard scaling is calculated as: $z = (x - u)/s$ where u is the mean of the training samples and s is the standard deviation of the training samples. Also various values for the parameter λ was tested for the both data transformations. We present here only the best performance which was obtained by standardisation the data and $\lambda = 0.005$. Test were concluded when the features that represent location where present and omitted. We get slightly better results without information about the location. Although the classification results are not great, we can see that the IRNS approach achieves the best results on train loss figure 4.19. In the following tables we can observe the classification results. On the test dataset there were 166 samples labelled as anomalies and 1405 which are not.

	IRNS	Heuristics	Full sample size
TP	95	118	88
FP	336	395	189
FN	71	48	178
TN	1069	1010	1216

Table 4.2: Confusion matrix for IoT data

	IRNS	Heuristics	Full sample size
Accuracy	0.741	0.718	0.830
Precision	0.220	0.230	0.318
Recall	0.572	0.711	0.530
F1 score	0.318	0.348	0.397

Table 4.3: Classification results

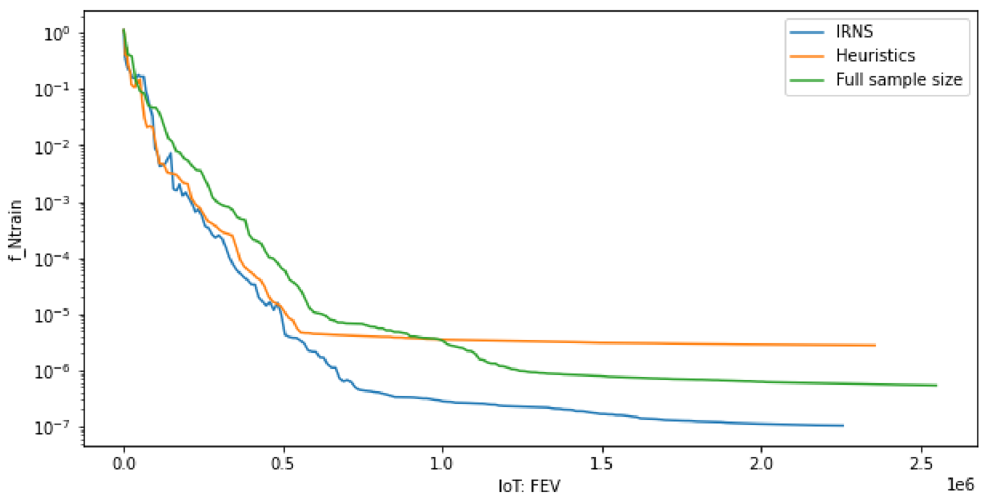

Figure 4.19: IoT - Training loss versus FEV

Figures 4.19 and 4.20 show the training loss of the algorithms with respect to FEV and number of iterations. It is clear that the IRNS approach achieves better result and is faster than other two, so it appears to be the most efficient.

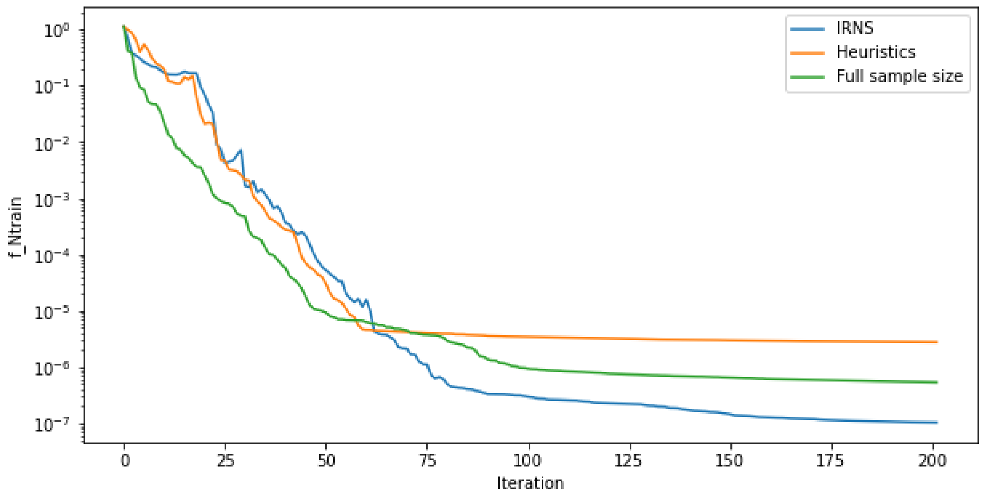

Figure 4.20: IoT - Training loss versus iteartion

From a recent master thesis *The gradient sampling algorithm for solving binary classification problems* [19] we can make a comparison how well our model performed on a training data regarding the loss function. The main purpose of the thesis was to test the SVM gradient sampling models on similar data sets and IoT data. Gradient sampling methods had been tested therein, as well as Nonnormalized grandiant sampling, Limiting ls gradient sampling and Trust region gradient sampling.

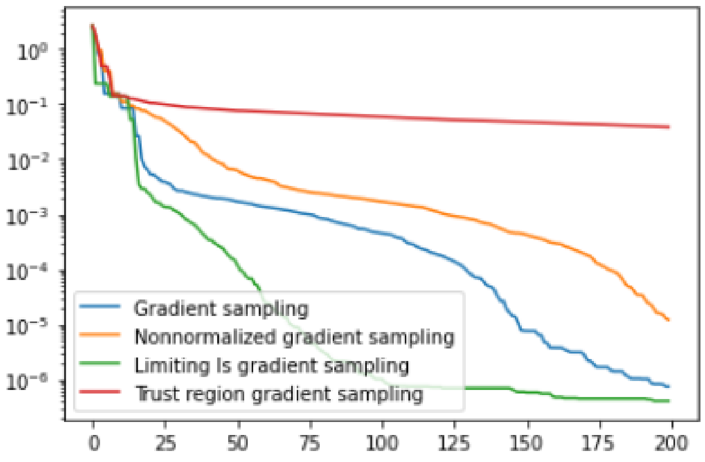

Figure 4.21: Figure from [19] representing the training loss of the IoT dataset with location

On a training set SVM models achieves decrease of magnitude of 10^{-6} over

200 iterations, and as we can see on figures 4.19 and 4.20 the IRNS approach have yield better results of order 10^{-7}. The main reason for this performance increase is the inclusion of the second order derivaties.

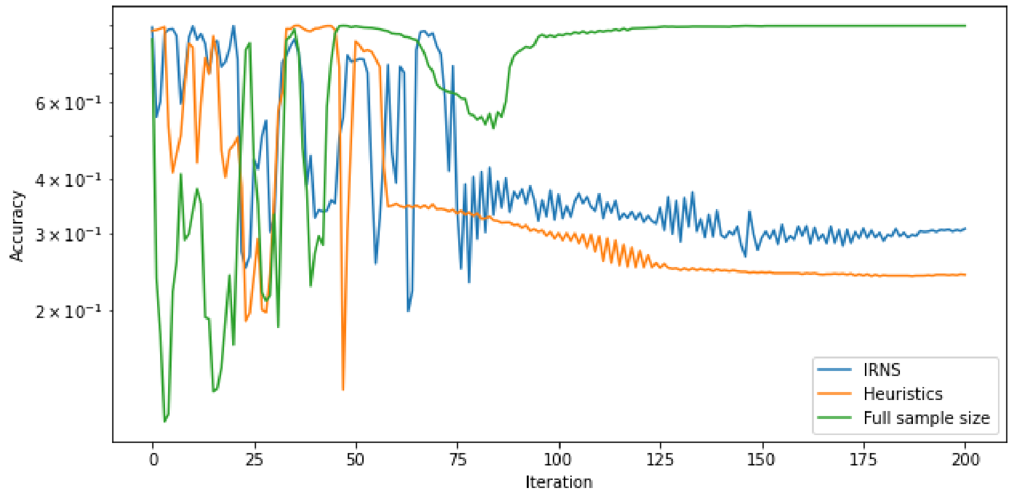

Figure 4.22: IoT - Accuracy versus iteration

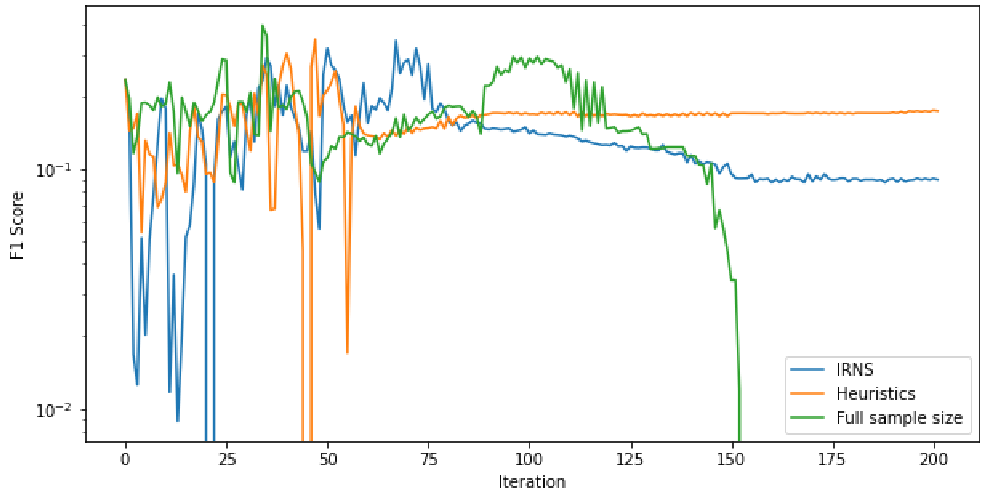

Figure 4.23: IoT - F1 Score versus iteration

We have to mention that the performance metrics are less satisfying as the gradient sampling methods. We can clearly see this on Figures 4.22 and 4.23. The reason for this is yet unknown and are open for further investigation in the space of parameters, maybe with application of some powerful grid search algorithm.

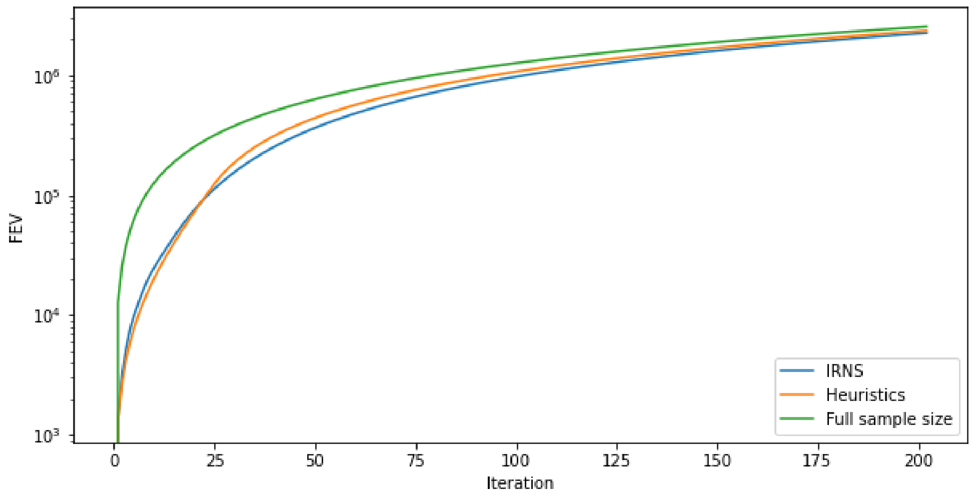

Figure 4.24: IoT - FEV versus iteration

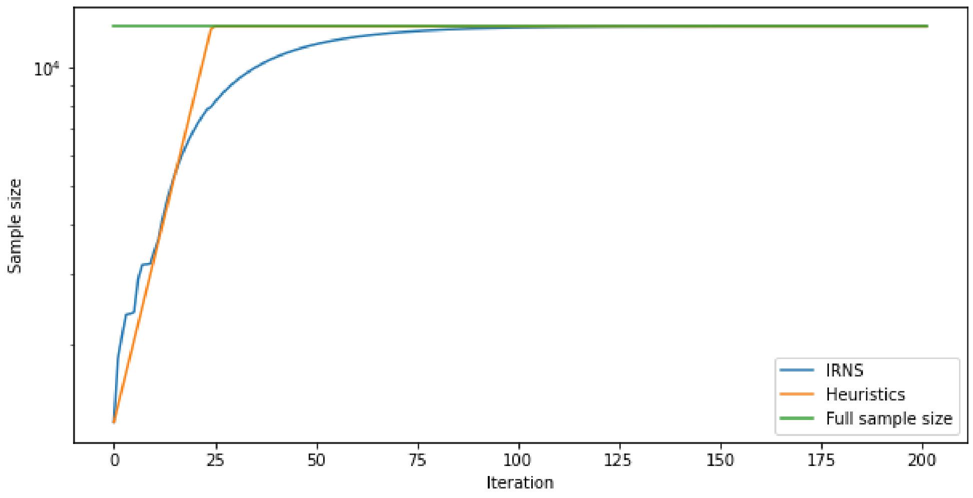

Figure 4.25: IoT - Sample size representation

Chapter 5

Conclusion

Withing this thesis first we have revised some of the fundamental principles of Machine learning, the Inexact Restoration method and the Hinge loss binary classification method for the usual case and for the anomaly extraction. Than we deep dived into the details of the algorithm. We successfully implemented IRNS algorithm in Python. The method was tested on a set of real data problems. It was tested against the heuristic and the full sample counterpart, showing the advantages of the adaptive sample size scheme, especially in terms of computational cost measured by FEV - the number of scalar products. Furthermore, the method was tested on IoT problem. The dataset had been generated for a C4IIoT project. Namely, this dataset had been created using NB-IoT edge nodes which were attached to the transport vehicle moving through the city of Novi Sad. The positioning data had been collected from the GNSS and IMU modules respectively.

Again, adaptive sample size strategy proved to be beneficial with respect to the core optimization problem which falls into the domain of support vector machine approach (SVM). It also shows the advantage of using second order information when the results about the training objective function's value are considered. More precisely, when the same number of iterations are considered, there can be seen that IRNS achieves better vicinity of the solution than the first order (Gradient Sampling) method. However, the results on the test set are not as expected since the metrics such as precision and accuracy are not at the satisfactory level. This indicates the potential problem with overfitting. Resolving this issue could be a subject of the future work.

Appendix A

The Python code

Code for Mushrooms, Adult and Splice datasets.

```
def aimh(N_k):
    return (N_max-N_k)/N_max

def hinge_loss(x,N):
    prediction = train.iloc[:N,:].dot(x)
    maximums = np.maximum(0,1-labels[:N]*prediction)
    return l/2*LA.norm(x)**2+1/N*maximums.sum()

def hl_der(x,N):
    y=train.iloc[:N,:].dot(x)*labels[:N]
    maximums=1-y
    all_der = (train.iloc[:N,:].T*labels[:N]).T
    greater_der = all_der[maximums>0]
    zero_der = all_der[maximums==0]
    beta=np.random.uniform(0,1,np.size(zero_der))
    if len(zero_der)!=0:
        return np.array(l*x-greater_der.sum()/N-np.dot(zero_der,beta)/N)
    elif len(zero_der)==0:
        return np.array(l*x-(greater_der.sum())/N)

def new_sample(N_k):
    return math.ceil(N_max*(1-r*aimh(N_k)))

def merit_function(x_k, N_k, theta_k):
    return theta_k*hinge_loss(x_k, N_k)+(1-theta_k)*aimh(N_k)

def find_theta(x, N_tilda, N, theta):
    sum1 = merit_function(x, N_tilda, theta)-merit_function(x, N, theta)
    sum2 = (1-r)/2*(aimh(N_tilda)-aimh(N))
    if sum1 <= sum2:
        return theta
    else:
        sum3 = (1+r)*(aimh(N)-aimh(N_tilda))
        sum4 = hinge_loss(x, N_tilda)-hinge_loss(x, N)+aimh(N)-aimh(N_tilda)
        return sum3/(2*sum4)

def sup_gp(x, p, N):
    sum1 = 0
    sum2 = 0
    for i in range(N):
        check = 1-labels[i]*np.matmul(train.iloc[i,:],x)
        if check == 0:
            inf = labels[i]*np.matmul(train.iloc[i,:],p)
            if inf < 0:
                sum1 += inf
        if check > 0:
            sum2 += labels[i]*train.iloc[i,:]
    return np.matmul(l*x - (sum2/N),p) - sum1/N

def model_Y(x, p, N, B):
    return np.matmul(np.matmul(p,np.linalg.inv(B)),p)/2 + sup_gp(x, p, N)

def argsup_gp(x, p, N):
    sum1 = np.zeros(x.shape)
    sum2 = np.zeros(x.shape)
    for i in range(N):
        check = 1-labels[i]*np.matmul(train.iloc[i,:],x)
        if check > 0:
            sum1 += labels[i]*train.iloc[i][:]
        if check == 0:
            if labels[i]*np.matul(train.iloc[i,:],p) < 0:
                sum2 += lables[i]*train.iloc[i][:]
    return np.array(l*x-sum1/N-sum2/N)

def descentDirection(g0_tilda, e, i_max, B, x, N):
    G_tilda = []
    G_bar = []
    P = []
    E = []

    #Step 0
    g0_bar = g0_tilda
    G_bar.append(g0_bar)
    G_tilda.append(g0_tilda)
    P.append(np.matmul(-B, G_tilda[0]))

    #Step 1
    G_tilda.append(argsup_gp(x, P[0], N))

    #Step 2
    E.append(np.matmul(P[0],G_tilda[1])-np.matmul(P[0],G_bar[0]))

    #Step 3
    i = 0
    while (np.matmul(G_tilda[i+1],P[i])>0 or E[0]>e) and E[i]>0 and i<i_max:
        first = np.matmul(G_bar[i]-G_tilda[i+1],np.matmul(B,G_bar[i]))
        second = np.matmul(G_bar[i]-G_tilda[i+1],np.matmul(B,G_bar[i]-G_tilda[i+1]))
        ni = min(1,first/second)
        G_bar.append((1-ni)*G_bar[i]+ni*G_tilda[i+1])
        P.append((1-ni)*P[i]-ni*np.matmul(B,G_tilda[i+1]))
        G_tilda.append(argsup_gp(x, P[i+1], N))
        e_list = []
        j = 0
        while j <= i+1:
            sum1 = np.matmul(P[j],G_tilda[j+1])
            sum2 = np.matmul(P[j],G_bar[j])+np.matmul(P[i+1],G_bar[i+1])
            e_list.append(sum1-sum2/2)
            j += 1
        E.append(min(e_list))
        i += 1

    #Step 4
    list_p = []
    j = 0
    while j <= i:
        list_p.append(model_Y(x, P[j], N, B))
        j += 1
    p_final = P[np.argmin(list_p)]

    #STEP5
    if sup_gp(x, p_final, N) < 0:
        return p_final
    else:
        print("Failed to find direction")

def check1(x, alpha, p, N_trial, N_tilda):
    sum1 = hinge_loss(x+alpha*p, N_trial)
    sum2 = hinge_loss(x, N_tilda)
    sum3 = -gama*alpha*(LA.norm(p,2)**2)
    return (sum1-sum2)<=sum3

def check2(N_trial, N_tilda, alpha, p):
    sum1 = aimh(N_trial)
    sum2 = aimh(N_tilda)
    sum3 = gama_k*alpha*alpha*(LA.norm(p,2)**2)
    return sum1<=(sum2+sum3)

def check3(x, alpha, p, N_trial, theta, N_start, N_tilda):
    sum1 = merit_function(x+alpha*p, N_trial, theta)
    sum2 = merit_function(x, N_start, theta)
    sum3 = (1-r)*(aimh(N_tilda)-aimh(N_start))/2
    return (sum1-sum2)<=sum3

def BFGS(B, g_new, g_old, x_new, x_old):
    I = np.identity(g_new.size)
    y = g_new-g_old
    s = x_new-x_old
    sT = np.reshape(s,(1,-1))
    s = np.reshape(s,(-1,1))
    yT = np.reshape(y,(1,-1))
    y = np.reshape(y,(-1,1))
    ro = 1/(yT@s)
    sum1 = I-ro*(s@yT)
    sum2 = I-ro*(y@sT)
    sum3 = ro*(s@sT)
    return (np.matmul(np.matmul(sum1,B),sum2)+sum3)

def theta_k(theta):
    return N_max*theta/(1-theta)

def p_N_trial(N_start, N_tilda, theta, theta_k, alpha, p ,x):
    sum1 = N_start + ((1-r)/2)*(N_tilda-N_start)/(1-theta)
    sum2 = theta_k*(gama*alpha*(LA.norm(p,2)**2) - hinge_loss(x, N_tilda)\
    + hinge_loss(x, N_start))
    return round(math.floor(sum1-sum2))

def first_N_trial(N_start, N_tilda, theta, theta_k, alpha, x):
    sum1 = N_start+(1-r)/2*(N_tilda-N_start)/(1-theta)
    sum2 = theta_k*(gama*alpha-hinge_loss(x, N_tilda)+hinge_loss(x, N_start))
    return math.ceil(sum1-sum2)

def candidates(N_trial, N_tilda):
    storage = []
    storage.append(int(N_trial))
    storage.append(int(math.ceil((N_trial+N_tilda)/2)))
    storage.append(N_tilda)
    return storage

def backtracking_IRNS(B, x, N_trial, N_tilda, N_start, theta):
    alpha_start = 0.5
    degree = 0
    g0 = hl_der(x, N_trial[0])
    g1 = hl_der(x, N_trial[1])
    g2 = hl_der(x, N_trial[2])

    p0 = descentDirection(g0, e, i_max, B, x, N_trial[0])
    p1 = descentDirection(g1, e, i_max, B, x, N_trial[1])
    p2 = descentDirection(g2, e, i_max, B, x, N_trial[2])

    while True:
        alpha = alpha_start**degree

        first = check1(x, alpha, p0, N_trial[0], N_tilda)
        second = check2(N_trial[0], N_tilda, alpha, p0)
        third = check3(x, alpha, p0, N_trial[0], theta, N_start, N_tilda)
        if first and second and third:
            print("Alpha: %s | Trial value: %s" %(alpha,0))
            return alpha, p0, N_trial[0]
            break
        else:
            first = check1(x, alpha, p1, N_trial[1], N_tilda)
            second = check2(N_trial[1], N_tilda, alpha, p1)
            third = check3(x, alpha, p1, N_trial[1], theta, N_start, N_tilda)
            if first and second and third:
                print("Alpha: %s | Trial value: %s" %(alpha,1))
                return alpha, p1, N_trial[1]
                break
            else:
                first = check1(x, alpha, p2, N_trial[2], N_tilda)
                second = check2(N_trial[2], N_tilda, alpha, p2)
                third = check3(x, alpha, p2, N_trial[2], theta, N_start, N_tilda)
                if first and second and third:
                    print("Alpha: %s | Trial value: %s" %(alpha,2))
                    return alpha, p2, N_trial[2]
                    break
                else:
                    if degree >= 10:
                        print("forced to stop")
                        return alpha, p2, N_trial[2]
                        break
                    degree += 1
                    print(degree)

def backtracking_normal(B, x, N_trial, N_tilda, N_start, theta):
    alpha_start = 0.5
    degree = 0
    g = hl_der(x, N_trial)
    p = descentDirection(g,e,i_max,B,x,N_trial)
    while True:
        alpha = alpha_start**degree
        first = check1(x, alpha, p, N_trial, N_tilda)
        third = check3(x, alpha, p, N_trial, theta, N_start, N_tilda)
        if first and third:
            return alpha, p
            break
        else:
            if degree >= 10:
                return alpha, p
                break
            degree += 1

def algorithm(method):
    start = time()

    max_fev = 0
    MAX_FEV = []
    MAX_FEV.append(max_fev)
    f_Ntrain = []
    N_tilda = []
    trial_N = []
    ITER = []
    ACC = []
    F1 = []
    T = []
    X = []
    P = []
    B = []
    G = []
    C = []
    A = []
    N = []

    X.append(x0)
    if method == "IRNS":
```

```
    N0 = math.ceil(0.1*N_max)
    N.append(N0)
    g0 = hl_der(x0, N[0])
    G.append(g0)
    max_fev += N[0]
    MAX_FEV.append(max_fev)
    f_Ntrain.append(hinge_loss(X[0],N_max))

if method == "HEUR":
    N0 = math.ceil(0.1*N_max)
    N.append(N0)
    g0 = hl_der(x0, N[0])
    G.append(g0)
    max_fev += N[0]
    MAX_FEV.append(max_fev)
    f_Ntrain.append(hinge_loss(X[0],N_max))

if method == "FSS":
    N0 = train.shape[0]
    N.append(N0)
    g0 = hl_der(X[0], N[0])
    G.append(g0)
    max_fev += N[0]
    MAX_FEV.append(max_fev)
    f_Ntrain.append(hinge_loss(X[0],N_max))

N_tilda.append(0)
T.append(theta0)
B.append(B0)

k = 0
ITER.append(0)
#while max_fev <= 10**6:
while k <= 100:
    if method == "IRNS":
        N_tilda.append(new_sample(N[k]))
        T.append(find_theta(X[k], N_tilda[k+1], N[k], T[k]))

        if k == 0:
            N_trial = first_N_trial(N[0], N_tilda[1], T[1],\
            theta_k(T[1]), 1, X[0])
        else:
            N_trial = p_N_trial(N[k], N_tilda[k+1], T[k+1],\
            theta_k(T[k+1]), A[-1], P[k-1] ,X[k])
        trial_N.append(N_trial)
        three_canditates = candidates(N_trial, N_tilda[k+1])
        C.append(three_canditates)
        print(C[k])
        alpha, p, N_new = backtracking_IRNS(B[k],X[k],three_canditates,\
        N_tilda[k+1],N[k],T[k+1])
        N.append(N_new)
        max_fev += N[-1]
        MAX_FEV.append(max_fev)

    if method == "HEUR":
        N_tilda.append(new_sample(N[k]))
        T.append(find_theta(X[k], N_tilda[k+1], N[k], T[k]))
        N_trial = round(min(1.1*N[k],N_max))
        N.append(N_trial)
        alpha, p = backtracking_normal(B[k],X[k],N_trial,\
        N_tilda[k+1],N[k],T[k+1])
        max_fev += N[-1]
        MAX_FEV.append(max_fev)

    if method == "FSS":
        N_tilda.append(train.shape[0])
        T.append(find_theta(X[k], N_tilda[k+1], N[k], T[k]))
        N_trial = train.shape[0]
        alpha, p = backtracking_normal(B[k], X[k], N_trial,\
        N_tilda[k+1], N[k], T[k+1])
        N.append(train.shape[0])
        max_fev += N[-1]
        MAX_FEV.append(max_fev)

    P.append(p)
    A.append(alpha)

    #Step 4
    sk = alpha*P[k]
    X.append(X[k]+sk)
    f_Ntrain.append(hinge_loss(X[-1],N_max))

    F, Accuracy = scores(k, X[-1])
    F1.append(F)
    ACC.append(Accuracy)

    #Step 5
    G.append(hl_der(X[k+1], N[k+1]))
    yk = G[k+1]-G[k]
```

```
        if np.matmul(sk,yk) >= m*LA.norm(yk,2)**2:
            B.append(BFGS(B[k], G[k+1], G[k], X[k+1], X[k]))
        else:
            B.append(B[k])

        #Step 6
        k += 1
        ITER.append(k)
        print("-------------------------------------")
    end = time()
    b = end - start
    print(' ')
    print('Time elapsed: %s minutes and %s seconds.\
    '%(np.floor(b / 60), np.round(b % 60)))
    return F1, ACC, MAX_FEV, f_Ntrain, N, ITER
```

Code for the anomaly detection is the same as above except for these four functions:

```
def hinge_loss(y, N):
    x = y[:-1]
    r = y[-1]
    prediction = train.iloc[:N,:].dot(x)
    maximums = np.maximum(0,r-prediction)
    return (l*LA.norm(x,2)**2)/2-(l*r)+(np.sum(maximums)/N)

def hl_der(y, N):
    ro=y[-1]
    x=y[:-1]
    y=train.iloc[:N,:].dot(x)
    maximums=ro-y
    all_der=train.iloc[:N,:]
    greater_der=all_der[maximums>0]
    zero_der=all_der[maximums==0]
    beta=np.random.uniform(0,1,np.size(zero_der))
    if len(zero_der)!=0:
        new_x = l*x-greater_der.sum()/(N)-np.dot(zero_der,beta)/(N)
        new_r = np.array([sum(beta)/(N)+len(greater_der)/(N)-l])
        return np.concatenate(new_x,new_r)
    elif len(zero_der)==0:
        return np.concatenate((l*x-greater_der.sum()/N,\
        np.array([len(greater_der)/(N)-l])))

def sup_gp(y, p, N):
    x = y[:-1]
    r = y[-1]
    px = p[:-1]
    pr = p[-1]

    y=train.iloc[:N,:].dot(x)
    decision = r - y
    all_der = train.iloc[:N,:]
    greater_der = all_der[decision>0]
    zero_der = all_der[decision==0]
    h = zero_der.dot(px)
    h1 = pr - h
    check = zero_der[h1>0]
    a = l*(px.dot(x))
    b = (greater_der.dot(px)).sum()
    c = l*pr
    d = pr*len(greater_der)
    e = pr-check.dot(px)
    if len(e)==0:
        return a-b/N-c+d/N
    else:
        return a-b/N-c+d/N+e/N

def argsup_gp(y, p, N):
    x = y[:-1]
    r = y[-1]
    px = p[:-1]
    pr = p[-1]

    sum1 = 0
    sum2 = 0
    sum3 = np.zeros(x.shape[0])
    sum4 = np.zeros(x.shape[0])
    for i in range(N):
        decision1 = r - np.matmul(x,train.iloc[i,:])
        if decision1 > 0:
            sum1 += 1
            sum3 += train.iloc[i,:]
        if decision1 == 0:
            if pr-np.matmul(px,train.iloc[i,:]) > 0:
                sum2 += 1
                sum4 += train.iloc[i,:]
    new_x = np.array(l*x-sum3/N-sum4/N)
    new_r = -l+sum1/N+sum2/N
    return np.concatenate((new_x, new_r), axis=None)
```

Bibliography

[1] https://zenodo.org/record/4686782#.Yis6NNXMKUl.

[2] https://programmathically.com/understanding-hinge-loss-and-the-svm-cost-function/

[3] https://miro.medium.com/max/1150/1*PGqpYm7o5GCbDXxXErr2JA.png.

[4] https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm.

[5] https://www.informatec.com/en/machine-learning.

[6] https://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/binary.html.

[7] A. ASL and M. L. OVERTON. Analysis of limited-memory bfgs on a class of nonsmooth convex functions. *IMA Journal of Numerical Analysis*, 41:1–27, 2021.

[8] A. BAGIROV, N. KARMITSA A, and M. MÄKELÄ. Introduction to nonsmooth optimization. *Springer*, 2014.

[9] S. BELLAVIA, N. KREJIĆ, and N. KRKLEC JERINKIĆ. Subsampled inexact newton methods for minimizing large sums of convex function. *IMA Journal of Numerical Analysis*, 40:2309–2341, 2018.

[10] S. BELLAVIA, N. KREJIĆ, and B. MORINI. Inexact restoration with subsampled trust-region methods for finite-sum minimization. *Computational Optimization and Applications*, 76:701–736, 2020.

[11] V. CEVHER, S. BECKER, and M. SCHMIDT. Convex optimization for big data. *IEEE Signal Processing Magazine*, 31:32–43, 2014.

[12] F. E. CURTIS and X. QUE. An adaptive gradient sampling algorithm for non-smooth optimization. *Optimization Methods and Software*, 28:1302–1324, 2013.

[13] A. P. GEORGE and W. B. POWELL. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Mach. Learn.*, 65:167–198, 2006.

[14] E. A. PILOTTA J. M. MARTINEZ. Inexact restoration algorithms for constrained optimization. *Journal of Optimization Theory and Applications*, 104:135–163, 2000.

[15] N. KREJIĆ, N. KRKLEC JERINKIĆ, and T. OSTOJIĆ. Minimizing nonsmooth convex functions with variable accuracy. 2020.

[16] N. KREJIĆ and N. KRKLEC JERNIKIĆ. Nonmonotone line search methods with variable sample size. *Numerical Algorithms*, 68:711–739, 2015.

[17] N. KREJIĆ and J. M. MARTINEZ. Inexact restoration approach for minimization with inexact evaluation of the objective function. *Mathematics of Computation*, 85:1775–1791, 2016.

[18] M. LICHMAN. Uci machine learning repository. `https://archive.ics.uci.edu/ml/index.php`, 2013.

[19] L. RUTEŠIĆ. The gradient sampling algorithm for solving binary classification problems. `https://matematika.pmf.uns.ac.rs/wp-content/uploads/2021/06/LukaRutesic.pdf`, 2021, Master thesis.

[20] D. YAN and H. MUKAI. Optimization algorithm with probabilistic estimation. *Journal of Optimization Theory and Applications*, 64:345–371, 1993.

[21] J. Yu, S. VISHWANATHAN, S. GUENTER, and N. SCHARAUDOLPH. A quasi-newton approach to nonsmooth convex optimization problems in machine learning. *Journal of Machine Learning Research*, 11:1145–1200, 2010.

List of Figures

Biography

I was born on the 2nd of October, 1992 in Novi Sad, Serbia where I have finished elementary school "Jožef Atila". Guided by my friends I enrolled in electrical high school "Mihajlo Pupin". After graduation in 2011, my growing interest in mathematics led me to the Faculty of Sciences in Novi Sad. There I have graduated the Applied Mathematics - Technomathematics Bachelor course in 2018 and in the same year I continued my master studies in the field of Data Science at the same faculty.

Appendix B

Key documentation

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:
RBR

Identifikacioni broj:
IBR

Tip dokumentacije: Monografska dokumentacija
TZ

Tip zapisa: Tekstualni štampani materijal
TZ

Vrsta rada: Master rad
VR

Autor: Hunor Tot-Bagi
AU

Mentor: dr Nataša Krklec Jerinkić
ME

Naslove rada: Metod netačne restauracije za rešavanje Hinge Loss problema
NR

Jezik publikacije: Engleski
JP

Jezik izvoda: Srpski / Engleski
JI

Zemlja publikovanja: Republika Srbija
ZP

Uže geografsko područje: Vojvodina
UGP

Godina: 2022.
GO

Izdavač: Autorski reprint
IZ

Mesto i adresa: Prirodno-matematički fakultet, Trg Dositeja Obradovića 4, Novi Sad
MA

Fizički opis rada: (5 / 57 / 21 / 3 / 3 / 25 / 0)(broj poglavlja / broj strana / broj citata / broj tabela / broj slika / broj grafika / broj priloga)
FO

Naučna oblast: Matematika
NO

Naučna disciplina: Mašinsko učenje i numerička analiza
ND

Predmetna odrednica / ključne reči: binarna klasifikacija, hinge loss, mašinsko učenje, SVM, deterkcija anomalija, IoT, BFGS, netačna restauracija
PO, UDK

Čuva se: U biblioteci Departmana za matematiku i informatiku, Prirodno-matematički fakultet, Univerzitet u Novom Sadu
ČU

Važna napomena:
VN

Izvod: Rad se može podeliti u tri glavne celine. U prvom delu objašnjavaju se osnovni principi mašinskog učenja, Hinge Loss funkcija gubitka za binarnu klasifikaciju i traženje anomalija korišćenjem pomoćne vektorske mašine. U drugom delu se može naći opis BFGS metoda, metoda netačne restauracije, kao i glavnog algoritma. Poslednja celina je posvećena numeričkim rezultatima.

Prikazani su grafici konvergencije i dati su rezultati predviđanja. Algoritmi su testirani na četiri skupa podataka gde je naglasak bio na traženju anomalija na skupu podataka iz c4IIot projekta.

IZ

Datum prihvatanja teme od NN veća:
DP

Datum odbrane:
DO

Članovi komisije:
ČK

Predsednik: dr Nataša Krejić, redovni profesor Prirodno-matematičkog fakulteta u Novom Sadud

Član: dr Dušan Jakovetić, vanredni profesor Prirodno-matematičkog fakulteta u Novom Sadu

Mentor: dr Nataša Krklec Jerinkić, vanredni profesor Prirodno-matematičkog fakulteta u Novom Sadu

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
KEY WORD DOCUMENTATION

Accession number:
ANO

Identification number:
INO

Document type: Monograph type
DT

Type of record: Printed text
TR

Content code: Master thesis
CC

Author: Hunor Tot-Bagi
AU

Mentor: dr Nataša Krklec Jerinkić
MN

Title: Inexact Restoration method for solving Hinge loss problems
TI

Language of text: English
LT

Language of abstract: Serbian / English
LA

Country of publication: Republic of Serbia
CP

Locality of publication: Vojvodina
LP

Publication year: 2022
PY

Publisher: Author's reprint
PU

Publication place: Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4
PP

Physical description: (4 / 50 / 21 / 3 / 36 / 0 / 0)(chapters / pages / literature / tables / pictures / graphics / appendices)
PD

Scientific field: Mathematics
SF

Scientific discipline: Machine learning and numerical analysis
SD

Subject / Key words: binary classification, hinge loss, machine learning, SVM, anomaly detection, IoT, BFGS, inexact restoration
SKW

Holding data: The Library of the Department of Mathematics and Informatics, Faculty of Science and Mathematics, University of Novi Sad
HD

Note:
N

Abstract: This thesis can be separated in three main parts. In first we explain the basic principles of machine learning, Hinge loss function for binary classification and detecting anomalies using SVM. Second section will be devoted for description of BFGS algorithm, algorithm of Inexact restoration and for the main algorithm. The last section is reserved for numerical results. We present graphs with convergence results and prediction. Algorithms were tested on four different datasets, where the emphasis was on finding the anomalies on dataset from the c4IIot project.
AB