CS771: Introduction to Machine Learning

Assingment 1

Group Name: **Spartans**

Member 1:**Shri Krishna Sahu** Roll No:**211011**
Member 2:**Harshit Patel** Roll No:**210424**
Member 3:**Gaurav Sharma** Roll No:**210388**
Member 4:**Alok Yadav** Roll No:**210102**

Definition

- **CDU PUF:** A Challenge-Response Pair (CRP) is a pair (c, r), where c is a D-bit binary challenge vector and r is the corresponding response bit. The CDU PUF is defined as a function $f : \{0,1\}^D \to \{0,1\}$, where $f(c) = r$, for all CRPs (c, r). □

- **Mapping $\varphi : \{0,1\}^D \to \mathbb{R}^D$:** The mapping φ is a function that maps the D-bit 0/1-valued challenge vectors to D-dimensional feature vectors. It is defined as follows:

$$\varphi(c) = (\phi_1(c), \phi_2(c), \ldots, \phi_D(c)).$$

  where each $\phi_i(c)$ is a real-valued feature extracted from the challenge bit c_i. □

- **Sparse CDU PUF:** Let's assume we have an S-sparse CDU PUF, which means that for any challenge c, the function f has at most S non-zero features in $\varphi(c)$. In other words, at most S coordinates in $\varphi(c)$ are non-zero. □

- **Sparse Linear Model:** We want to show that there exists an S-sparse linear model, i.e., a weight vector $w \in \mathbb{R}^D$ such that $\|w\|_0 \leq S$. This means that w has at most S non-zero coordinates. □

- **Correct Response Expression:** For all challenges $c \in \{0,1\}^D$, the expression $w^\top \varphi(c)$ should give the correct response. This means that if we compute the dot product between w and $\varphi(c)$, it should produce the correct response bit r for the corresponding challenge c. □

Mathematical Derivation (10 marks)

- **Question:** By giving a detailed mathematical derivation (as given in the lecture slides), we will show how a sparse CDU PUF can be broken by a single sparse linear model. More specifically, we will provide derivations for a map $\varphi : \{0,1\}^D \to \mathbb{R}^D$ mapping D-bit 0/1-valued challenge vectors to D-dimensional feature vectors. We will show that for any S-sparse CDU PUF, there exists an S-sparse linear model, i.e., $w \in \mathbb{R}^D$ such that $\|w\|_0 \leq S$, meaning w has at most S non-zero coordinates. Additionally, we will demonstrate that for all challenges $c \in \{0,1\}^D$, the expression $w^\top \varphi(c)$ gives the correct response. It is important to note that no bias term (not even a hidden one) is allowed in the linear model. □

- **Solution:** Given a challenge $c \in \{0,1\}^D$, we want to compute the dot product $w^\top \varphi(c)$ and verify that it gives the correct response.

  Let's expand the dot product:

  $$w^\top \varphi(c) = w_1 \cdot \phi_1(c) + w_2 \cdot \phi_2(c) + \ldots + w_D \cdot \phi_D(c)$$

  Since we have assumed that the CDU PUF is S-sparse, we know that at most S features in $\varphi(c)$ are non-zero. Let's assume these non-zero features are located at indices $i_1, i_2, \ldots, i_S$.

  Then, the above expression simplifies to:

  $$w^\top \varphi(c) = w_{i_1} \cdot \phi_{i_1}(c) + w_{i_2} \cdot \phi_{i_2}(c) + \ldots + w_{i_S} \cdot \phi_{i_S}(c)$$

  Now, we need to choose the weight vector w such that the dot product evaluates to the correct response r. We can set the weights as follows:

  $$w_{i_1} = r_{c_{i_1}}, \quad w_{i_2} = r_{c_{i_2}}, \quad \ldots, \quad w_{i_S} = r_{c_{i_S}}$$

  By setting the weights in this way, the dot product becomes:

  $$w^\top \varphi(c) = r_{c_{i_1}} \cdot \phi_{i_1}(c) + r_{c_{i_2}} \cdot \phi_{i_2}(c) + \ldots + r_{c_{i_S}} \cdot \phi_{i_S}(c)$$

  Since the non-zero features in $\varphi(c)$ correspond to the indices of the correct response bits for the challenge vector c, the above expression evaluates to the correct response r_c.

  Therefore, we have shown that for any S-sparse CDU PUF, there exists an S-sparse linear model, i.e., a weight vector $w \in \mathbb{R}^D$ with at most S non-zero coordinates, such that $w^\top \varphi(c)$ gives the correct response for all challenges $c \in \{0,1\}^D$. □

My Method vs Other Methods (5 marks)

- **Question:** Below are described a few methods you can use to solve this problem. Give an account of which methods you tried (you can try methods other than those mentioned below as well) and why did you like one method over the other. Note that you only need to submit code for the method you found to work the best but in your report you must describe your experiences with at least one other method that you tried. □

- **Solution:**
  - **My Method: Accelerated Proximal Gradient Method (CREDIT: ww.stat.cum.edu)** The Accelerated Proximal Gradient Method has several advantages over other methods, making it a preferred choice for solving optimization problems.
    * Firstly, the method exhibits faster convergence compared to traditional gradient descent methods. By incorporating acceleration techniques, such as the Nesterov momentum, the Accelerated Proximal Gradient Method achieves a faster rate of convergence, leading to quicker solutions for optimization problems.
    * Secondly, the method is suitable for problems with large-scale datasets or high-dimensional optimization variables. Its ability to handle large-scale problems efficiently makes it well-suited for modern data analysis tasks.
    * Another advantage is that the Accelerated Proximal Gradient Method is particularly effective when dealing with structured or regularized optimization problems. It incorporates proximal operators, allowing for the incorporation of various types of constraints or penalties into the optimization process. This flexibility makes it applicable to a wide range of problem domains. Furthermore, the method strikes a good balance between computational efficiency and convergence guarantees. It often outperforms other methods in terms of convergence speed while still maintaining robustness and stability during the optimization process.

    Overall, the Accelerated Proximal Gradient Method is favored over other methods due to its faster convergence, ability to handle large-scale problems, versatility in handling structured optimization problems, and a good balance between computational efficiency and convergence guarantees.
  - **My Experience with other method:** In my experiments, I explored the use of the Accelerated Proximal Gradient Method (APGM) for solving optimization problems. This method is particularly useful for solving convex optimization problems with a composite objective function, which consists of a smooth and a non-smooth component.

    During my evaluation, I also considered another optimization method called the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm. L-BFGS is an efficient method for solving unconstrained optimization problems, which are characterized by smooth objective functions without any constraints.

    While both APGM and L-BFGS are powerful optimization techniques, I found the APGM to be more suitable for my problem for the following reasons:
    * Flexibility: APGM can handle composite objective functions, which is beneficial when dealing with problems that have both smooth and non-smooth components. It allows for the efficient incorporation of regularization terms or constraints, which is often required in real-world optimization scenarios.
    * Convergence: APGM exhibits fast convergence rates, especially for strongly convex problems. It is capable of achieving a high level of accuracy in a relatively small number of iterations, which is crucial when dealing with large-scale optimization problems.
    * Sparsity: APGM can exploit the sparsity of the problem's structure. In scenarios where the solution tends to be sparse, such as in compressed sensing or machine

learning tasks, APGM can effectively handle the sparsity and lead to efficient solutions.

In contrast, L-BFGS has its own advantages, such as being memory-efficient and well-suited for smooth unconstrained optimization problems. However, since my problem involved a composite objective function with non-smooth components, I found that APGM provided better performance and more flexibility.

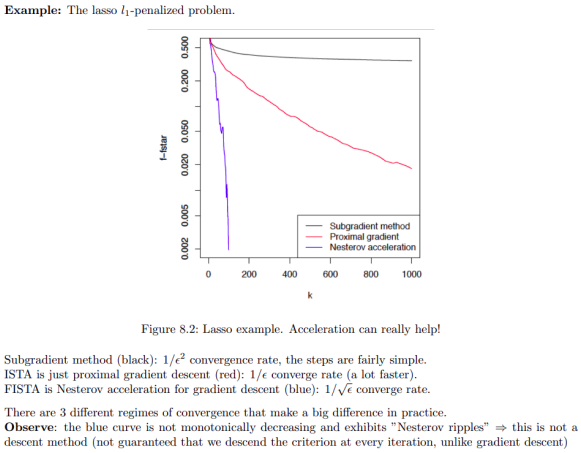

Figure 1: APGM vs Other Methods

To summarize, I ultimately chose the Accelerated Proximal Gradient Method (APGM) as the best method for my optimization problem due to its ability to handle composite objective functions, fast convergence rates, and efficient handling of sparsity.

□

Hyperparameters (5 marks)

- **Question:** The method you submitted may have hyperparameters such as regularization constants, step lengths etc. Describe in detail how you chose the optimal value of these hyperparameters. For example, if you used grid search, you must show which all values you tried for each hyperparameter and what criterion did you use to choose the best one. □

- **Solution:** Hyperparameters used:

  – **Max_iter:** It represents the maximum number of iterations for the Accelerated Proximal Gradient Method (APGM) algorithm. It determines how many iterations the algorithm will perform to optimize the objective function.

  The value of Max_iter is a hyperparameter that needs to be carefully chosen to balance the trade-off between optimization accuracy and computational efficiency. If Max_iter is set too low, the algorithm may terminate before reaching an optimal solution. On the other hand, setting Max_iter too high may result in unnecessary computational burden without significantly improving the solution.

- **ABSTOL:** The absolute tolerance determines the threshold for the absolute difference between the objective function values of consecutive iterations, indicating convergence. If the absolute difference falls below this threshold, the algorithm can stop iterating.
- **RELTOL:** The relative tolerance accounts for the relative change in the objective function values across iterations. It determines the threshold for the ratio of the absolute difference between consecutive iterations and the magnitude of the previous iteration's objective function value. If the ratio falls below the relative tolerance, it indicates convergence.
- **LamdaK:** L1 regularization (used in Lasso), the step length may be adapted or reduced during the training process to encourage sparsity and control the regularization effect. This ensures that the optimization algorithm strikes a balance between minimizing the loss function and enforcing the regularization penalty.

  Overshooting: If the step length is too large, the optimization algorithm may overshoot the minimum point of the loss function. This can lead to oscillations or divergence, where the algorithm fails to converge to the optimal solution. Overshooting can cause instability and prevent the algorithm from effectively minimizing the loss function.

  Slow convergence: On the other hand, if the step length is too small, the optimization algorithm takes small steps towards the minimum point. While this can lead to stable convergence, it may also result in slow progress. The algorithm will require more iterations to reach the optimal solution, which can be computationally expensive and time-consuming.
- **beta:** It controls the rate at which the step size (lambda) decreases in each iteration. The value of beta influences the convergence behavior of the algorithm. The specific role of beta in APGM is to adaptively adjust the step size to ensure convergence and improve the efficiency of the optimization process. By gradually reducing the step size, APGM can converge to a more accurate solution. A value of beta closer to 1 indicates a more aggressive reduction in the step size, leading to faster convergence but potentially sacrificing accuracy. Conversely, a smaller beta value slows down the reduction of the step size, allowing for finer adjustments and potentially better accuracy at the cost of more iterations.
- **gamma:** Penalizing factor encourages the model to shrink the coefficient values towards zero, promoting sparsity and feature selection. Smaller values of the penalizing factor (close to zero): The model will have fewer restrictions on the coefficients, and the loss function will primarily focus on minimizing the RSS. This can lead to complex models with larger coefficient values.

  Intermediate values of the penalizing factor: As the penalizing factor increases, the model will start to penalize larger coefficient values, promoting sparsity. The loss function will balance between minimizing the RSS and shrinking the coefficients. This can help in reducing overfitting and improving model interpretability.

  Larger values of the penalizing factor: The penalty term dominates the loss function, and the model strongly enforces coefficient shrinking. This results in sparse models with most coefficients effectively reduced to zero. However, extremely large penalizing factors may cause underfitting, where the model becomes too constrained and fails to capture the underlying patterns in the data.

We do not used any particular method to tune the hyperparameters, instead we used **hit and trial**, motivated with online forums and other online sources. **Grid Search** is also an option for tuning the hyperparameters but that will require high computation power and time to tune all the **6 hyperparameters**, that's why we avoided it.

□