AM 221: Advanced Optimization	Spring 2018
Kojin Oshiba	*Problem Set 8 — Due Wednesday, April. 4th at 23:59*

Instructions:

- All your solutions should be prepared in LaTeX and the PDF and .tex should be submitted to Canvas. Please submit all your files as ONE archive of filetype zip, tgz, or tar.gz.
- Name the file [your-first-name]_[your-last-name].[filetype]. For example, I would call my submission rasmus_kyng.zip.
- INCLUDE your name in the submisson pdf and any files with code.
- If the TFs cannot easily deduce your identity from your files alone, they may decide not to grade your submission.
- For each question, a well-written and correct answer will be selected a sample solution for the entire class to enjoy. If you prefer that we do not use your solutions, please indicate this clearly on the first page of your assignment.

1. Gradient descent with a noisy oracle. In this problem, we will show that the gradient descent algorithm can be used when optimizing a strongly convex function given an approximate oracle for its gradient.

Let us consider a twice-differentiable strongly convex function $f : \mathbb{R}^n \to \mathbb{R}$, *i.e* we have:

$$mI_n \preceq \nabla^2 f(\mathbf{x}) \preceq MI_n,\ \mathbf{x} \in \mathbb{R}^n$$

for some constants $m, M > 0$. The function f is unknown to us. Instead, for any $\mathbf{x} \in \mathbb{R}^n$, we can query an oracle for the value of the gradient of f at $\mathbf{x} \in \mathbb{R}^n$. The oracle is erroneous in the following sense: let us denote by $\tilde{\nabla} f(\mathbf{x})$ the value returned by the oracle at point $\mathbf{x}$, then we have:

$$||\tilde{\nabla} f(\mathbf{x}) - \nabla f(\mathbf{x})|| \leq \delta ||\nabla f(\mathbf{x})||$$

for some $\delta > 0$. Such an oracle is called δ-erroneous.

We now consider the gradient descent algorithm from Lecture 9 where the update at each iteration is computed using the erroneous oracle: denoting by $\mathbf{x}^{(k)}$ the solution at iteration k, the solution at iteration $k+1$ is given by:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t\tilde{\nabla} f(\mathbf{x}^{(k)})$$

where the step size is constant set to $t = \frac{1}{M}$.

We say that a solution $\mathbf{x}'$ has accuracy ε for f if $f(\mathbf{x}') - f(\mathbf{x}^*) \leq \varepsilon$, where $\mathbf{x}^*$ is the minimizer of f. By adapting the analysis of the gradient descent algorithm from Lecture 9, prove the following statement:

Theorem. *For any $\varepsilon > 0$, the gradient descent algorithm using a δ-erroneous oracle with $\delta \leq 0.1$ computes a solution of accuracy ε in at most 10 times as many iterations as we proved are sufficient for the standard gradient descent algorithm* i.e *the one which uses the exact gradient, i.e. show that* $10\frac{M}{m}\log\left(\frac{f(\mathbf{x}^{(0)})-f(\mathbf{x}^*)}{\epsilon}\right)$ *iterations suffice.*

Remark 1. The constant 10 was chosen rather arbitrarily, it is not tight.

First, we have

$$\begin{aligned}
&||\tilde{\nabla} f(\mathbf{x}) - \nabla f(\mathbf{x})|| \leq \delta||\nabla f(\mathbf{x})|| \\
\Rightarrow &||\tilde{\nabla} f(\mathbf{x}) - \nabla f(\mathbf{x})||^2 \leq \delta^2||\nabla f(\mathbf{x})||^2 \\
\Rightarrow &\nabla f(\mathbf{x}^{(k)})^\intercal \tilde{\nabla} f(\mathbf{x}^{(k)}) \geq \frac{||\tilde{\nabla} f(\mathbf{x})||^2 + (1-\delta^2)||\nabla f(\mathbf{x})||^2}{2}
\end{aligned}$$

Hence,

$$\begin{aligned}
f(\mathbf{x}^{(k+1)}) &= f(\mathbf{x}^{(k)} - t\tilde{\nabla} f(\mathbf{x}^{(k)})) \\
&\leq f(\mathbf{x}^{(k)}) + \nabla f(\mathbf{x}^{(k)})^\intercal(-t\tilde{\nabla} f(\mathbf{x}^{(k)})) + \frac{M}{2}|| - t\tilde{\nabla} f(\mathbf{x}^{(k)})||^2 \\
&\leq f(\mathbf{x}^{(k)}) - \frac{t}{2}(||\tilde{\nabla} f(\mathbf{x})||^2 + (1-\delta^2)||\nabla f(\mathbf{x})||^2) + \frac{Mt^2}{2}||\tilde{\nabla} f(\mathbf{x}^{(k)})||^2 \\
&= f(\mathbf{x}^{(k)}) - \frac{1-\delta^2}{2M}||f(\mathbf{x}^{(k)})||^2 \quad (\because t = \frac{1}{M})
\end{aligned}$$

Analogous to the proof of Lecture 9 on page 6, we thus have

$$k \geq \frac{\log\left(\frac{f(x^0)-\alpha^*}{\epsilon}\right)}{\log\left(\frac{1}{1-\frac{m(1-\delta^2)}{M}}\right)}$$

What we have left to show is

$$\frac{1}{\log\left(\frac{1}{1-\frac{m(1-\delta^2)}{M}}\right)} \geq 10\frac{M}{m} \Leftrightarrow 1 - \frac{m(1-\delta^2)}{M} \leq e^{-\frac{m}{10M}}$$

This can be shown easily because,

$$e^{-\frac{m}{10M}} \geq 1 - \frac{m}{10M} = 1 - \frac{0.1m}{M}$$

$$1 - \frac{m(1-\delta^2)}{M} \leq 1 - \frac{0.99m}{M} \leq 1 - \frac{0.1m}{M}$$

This concludes the proof.

2. Duality and SVMs. In this problem, we will refine our analysis of the primal and dual formulations of the support vector machine optimization problem of Lecture 13. Remember that in this problem we have a dataset consisting of two clusters of data points in $\mathbb{R}^d$: positively labeled data points $\{\mathbf{x}_i,\ i \in I\}$ and negatively labeled data points $\{\mathbf{x}_j,\ j \in J\}$. The goal is to find an affine hyperplane $(\mathbf{a}, b) \in \mathbb{R}^d \times \mathbb{R}$ such that:

$$\mathbf{a}^\intercal \mathbf{x}_i + b > 0,\ i \in I$$
$$\mathbf{a}^\intercal \mathbf{x}_j + b < 0,\ j \in J$$

The primal problem is written as:

$$\begin{aligned}\min_{\mathbf{a}\in\mathbb{R}^d,\, b\in\mathbb{R}} \quad & \frac{\|\mathbf{a}\|^2}{4} \\ \text{s.t. } & \mathbf{a}^\intercal \mathbf{x}_i + b \geq 1,\ i \in I \\ & \mathbf{a}^\intercal \mathbf{x}_j + b \leq -1,\ j \in J\end{aligned}$$

and we computed the dual in class:

$$\begin{aligned}\max_{\lambda\in\mathbb{R}^{|I|},\, \mu\in\mathbb{R}^{|J|}} \quad & \frac{1}{\left\| \sum_{i\in I} \lambda_i \mathbf{x}_i - \sum_{j\in J} \mu_j \mathbf{x}_j \right\|^2} \\ \text{s.t. } & \sum_{i\in I} \lambda_i = \sum_{j\in J} \mu_j = 1 \\ & \lambda \geq 0,\ \mu \geq 0\end{aligned}$$

We assume that Slater's condition holds so that we have strong duality.

a. Let us denote by (λ, μ) a dual-optimal solution and by $(\mathbf{a}, b)$ a primal-optimal solution. Show that:

$$\text{either } \lambda_i = 0 \quad \text{or} \quad \mathbf{a}^\intercal \mathbf{x}_i + b = 1, \quad i \in I$$

similarly, show that:

$$\text{either } \mu_j = 0 \quad \text{or} \quad \mathbf{a}^\intercal \mathbf{x}_j + b = -1, \quad j \in J$$

Give a geometric interpretation.

b. Using part a., explain how a primal-optimal solution could be computed from a dual-optimal solution.

a. Since strong duality holds, KKT conditions are necessary. Hence, from condition (8) in Lecture 14, we have

$$\lambda_i(1 - \mathbf{a}^\intercal \mathbf{x}_i - b) = 0, \forall i \in I$$

$$\mu_j(1 + \mathbf{a}^\intercal \mathbf{x}_j + b) = 0, \forall j \in J$$

This implies what we want to show.

b. From equation (14) in Lecture 13,

$$\frac{\mathbf{a}}{2} + \sum_{j\in J} \mu_j \mathbf{x}_j - \sum_{i\in I} \lambda_i \mathbf{x}_i = 0 \Leftrightarrow \mathbf{a} = 2(\sum_{j\in J} \mu_j \mathbf{x}_j - \sum_{i\in I} \lambda_i \mathbf{x}_i)$$

Once we know $\mathbf{a}$, we can find b as follows:

$$b = -\frac{\min_{i\in I} \mathbf{a}^\intercal \mathbf{x}_i + \max j \in J \mathbf{a}^\intercal \mathbf{x}_i}{2}$$

This is becasue b is half the difference between the point in I that is on the support vector, which is $\operatorname{argmin}_{i\in I} \mathbf{a}^\intercal \mathbf{x}_i$ and point in J that is on the other support vector, which is $\operatorname{argmax}_{j\in J} \mathbf{a}^\intercal \mathbf{x}_j$.

3. Revisiting the Maximum Coverage Problem Remember the Maximum Coverage Problem from Section 1. In this problem there is a universe of elements $\mathcal{U} = \{1, \ldots, m\}$ and you are given as input a collection of n subsets $S_1, \ldots, S_n$ of $\mathcal{U}$ ($S_i \subseteq \mathcal{U}$) and a budget $k \in \mathbb{N}$. The goal is select a collection $\mathcal{S}$ of at most k of the sets $S_1, \ldots, S_n$ such as to maximize the number of elements of $\mathcal{U}$ contained in the union of the sets in $\mathcal{S}$. In other words, the goal is to solve:

$$\max_{|\mathcal{S}|\leq k} \left| \bigcup_{S_i\in\mathcal{S}} S_i \right|$$

One of the relaxations of this problem we considered in Section 1 was the following:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{j=1}^{m} \min\left\{1, \sum_{i:j\in S_i} x_i\right\} \\ \text{s.t} \quad & x_i \geq 0 \quad 1 \leq i \leq n \\ & \sum_{i=1}^{n} x_i \leq k \end{aligned} \tag{P}$$

a. Sow that the objective function in problem (P) is concave. How could you use an algorithm for minimizing a convex function to solve this relaxation?

b. Show that problem (P) can be reformulated as a linear program.

a. Let the objective be $f(\mathbf{x})$.

$$\begin{aligned} f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) &= \sum_{j=1}^{m} \min(1, \sum_{i:j\in S_i} \lambda\mathbf{x}_{1,i} + (1-\lambda)\mathbf{x}_{2,i}) \\ &\geq \sum_{j=1}^{m} \min(1, \sum_{i:j\in S_i} \lambda\mathbf{x}_{1,i}) + \sum_{j=1}^{m} \min(1, \sum_{i:j\in S_i} (1-\lambda)\mathbf{x}_{2,i}) \\ &\geq \lambda \sum_{j=1}^{m} \min(1, \sum_{i:j\in S_i} \mathbf{x}_{1,i}) + (1-\lambda) \sum_{j=1}^{m} \min(1, \sum_{i:j\in S_i} \mathbf{x}_{2,i}) \\ &= \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2) \end{aligned}$$

Hence, $f(\mathbf{x})$ is concave. The second line follows from the first line because $\lambda\mathbf{x}_{1,i}, (1-\lambda)\mathbf{x}_{2,i}$ are both positive and hence summing the minimums separately as in the second line would result in a lower value overall.

The third line follows from the second line because $\min(1, \sum_{i:j\in S_i} \lambda\mathbf{x}_{1,i})$ is either 1 or $\sum_{i:j\in S_i} \lambda\mathbf{x}_{1,i}$, whereas $\lambda \min(1, \sum_{i:j\in S_i} \mathbf{x}_{1,i})$ is either λ or $\sum_{i:j\in S_i} \lambda\mathbf{x}_{1,i}$. Since $\lambda \in [0,1]$, we have that the former is always greater than or equal to the latter. We can similarly compare $\min(1, \sum_{i:j\in S_i}(1-\lambda)\mathbf{x}_{2,i})$ and $(1-\lambda)\min(1, \sum_{i:j\in S_i} \mathbf{x}_{2,i})$.

Therefore, we can negate the objective to make it an objective to minimize a convex function and apply gradient descent, or other optimization methods we learned in class.

b. We can replace $\min\left\{1, \sum_{i:j\in S_i} x_i\right\}$ with y_j's and add linear constraints on y_j's. The reformulated LP is thus,

$$\begin{aligned}
\max_{\mathbf{x}} \quad & \sum_{j=1}^{m} y_j \\
\text{s.t} \quad & y_j \le 1, y_j \le \sum_{i:j\in S_i} x_i \\
& x_i \ge 0 \quad 1 \le i \le n \\
& \sum_{i=1}^{n} x_i \le k
\end{aligned}$$

4. Barrier method. Let us briefly review the barrier method seen in section 8. Consider the following optimization problem with m inequality constraints:

$$\begin{aligned}
\min_{\mathbf{x}\in\mathbb{R}^n} \; & f(\mathbf{x}) \\
\text{s.t. } & f_i(\mathbf{x}) \le 0, \; 1 \le i \le m
\end{aligned}$$

This problem is transformed into the following unconstrained problem using the barrier function $\hat{I}_t(u) = -\frac{1}{t}\log(-u)$:

$$\min_{\mathbf{x}\in\mathbb{R}^n} tf(\mathbf{x}) - \sum_{i=1}^{m} \log\big(-f_i(\mathbf{x})\big)$$

Let us denote by $\mathbf{x}^*(t)$ the optimal solution to this problem. Then the barrier method simply consists in solving the transformed problem for increasing values of t:

Algorithm 1 Barrier method with parameter $\mu > 1$

1: $t \leftarrow 1$, $\mathbf{x} \leftarrow$ feasible solution
2: **while** $\frac{m}{t} \ge \varepsilon$ **do**
3: $\quad\mathbf{x} \leftarrow \mathbf{x}^*(t)$
4: $\quad t \leftarrow \mu t$
5: **end while**
6: **return** $\mathbf{x}$

In this problem we will use the barrier method to compute the support vector machine classifier for the forged banknotes dataset available at: http://rasmuskyng.com/am221_spring18/psets/hw7/banknotes.data. In each line, the first four columns contain measurements from a banknote (real numbers) and the last column is a binary (0 or 1) variable indicating if the banknote was forged. Denoting by $\mathbf{x}^i \in \mathbb{R}^4$ the measurements from banknote i, the goal is to construct a classifier which takes $\mathbf{x}^i$ as input and predicts the last column $y^i \in \{0, 1\}$.

First, convert the labels to $\hat{y}^i \in \{-1, 1\}$, i.e. $\hat{y}^i = 2y^i - 1$. As seen in class, finding a separating hyperplane now amounts to finding $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that $\hat{y}^i(\mathbf{w}^\intercal \mathbf{x}_i + b) \geq 1$, for $1 \leq i \leq n$, where $\mathbf{x}_1, \dots \mathbf{x}_n$ are the (modified) data points.

We consider a "soft margin" version of the SVM optimization problem (also seen in the previous homework). The optimization problem is the following

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \ & \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{i=1}^{n} \xi_i \\ \text{s.t } & \hat{y}^i(\mathbf{w}^\intercal \mathbf{x}_i + b) + \xi_i \geq 1,\ 1 \leq i \leq n \\ & \xi_i \geq 0,\ 1 \leq i \leq n \end{aligned} \tag{1}$$

a. Write code to implement the barrier method described in Algorithm 1 for the optimization problem (1). For the internal optimization $\mathbf{x} \leftarrow \mathbf{x}^*(t)$, you are free to reuse either your implementation of the gradient descent algorithm or of Newton's method from previous problem sets. In fact you are encouraged to experiment with both!

b. Report the accuracy (fraction of correctly classified data points) for the optimal hyperplane found by the code you wrote in part a. For the penalty parameter λ, reuse the optimal value you found in Problem Set 7 (or experiment with different values if you haven't done Problem Set 7). What is the impact of the parameter μ on the convergence of your implementation?

a. The code is under hw8.py. Basically, I converted the SVM optimization problem to barrier method optimization problem. Then, I implemented gradient descent using PyTorch.

b. For the sake of time, I run each μ for 25 iterations and report the final accuracy. The lower μ is the higher the accuracy because the barrier method is more careful in searching the optimal solution. But since t increases slowly, the convergence is slower.

   The optimal accuracy is about 88.48% is consistent across different μ's (assuming you run the method for sufficient number of iterations. Below, I provide the row outputs for some of the μ values. See the code for the optimal hyperparameters chosen.

```
Mu: 1.1
0
t: 1 | objective: 136865.62103624135 | classification accuracy: 0.37244897959183676
1
t: 1.1 | objective: 136529.417169686 | classification accuracy: 0.5342565597667639
2
t: 1.2100000000000002 | objective: 136192.2125845217 | classification accuracy: 0.639941690
```

```
3
t: 1.3310000000000004 | objective: 135854.28418061612 | classification accuracy: 0.70043731
4
t: 1.4641000000000006 | objective: 135515.7755352815 | classification accuracy: 0.741982507
5
t: 1.6105100000000008 | objective: 135176.77923031317 | classification accuracy: 0.76384839
6
t: 1.771561000000001 | objective: 134837.36406482928 | classification accuracy: 0.780612244
7
t: 1.9487171000000014 | objective: 134497.58434500627 | classification accuracy: 0.79810495
8
t: 2.1435888100000016 | objective: 134157.48430887505 | classification accuracy: 0.80685131
9
t: 2.357947691000002 | objective: 133817.1008436097 | classification accuracy: 0.8192419825
10
t: 2.5937424601000023 | objective: 133476.4652601572 | classification accuracy: 0.827988338
11
t: 2.853116706110003 | objective: 133135.6044981562 | classification accuracy: 0.8309037900
12
t: 3.1384283767210035 | objective: 132794.54197736568 | classification accuracy: 0.84183673
13
t: 3.4522712143931042 | objective: 132453.29822176174 | classification accuracy: 0.84693877
14
t: 3.797498335832415 | objective: 132111.89133174712 | classification accuracy: 0.853498542
15
t: 4.177248169415656 | objective: 131770.33735120686 | classification accuracy: 0.855685131
16
t: 4.594972986357222 | objective: 131428.65055941133 | classification accuracy: 0.860058309
17
t: 5.054470284992944 | objective: 131086.84370767354 | classification accuracy: 0.863702623
18
t: 5.559917313492239 | objective: 130744.92821437202 | classification accuracy: 0.867346938
19
t: 6.115909044841463 | objective: 130402.91432789678 | classification accuracy: 0.869533527
20
t: 6.72749994932561 | objective: 130060.81126439998 | classification accuracy: 0.8717201166
21
t: 7.400249944258172 | objective: 129718.62732541031 | classification accuracy: 0.873177842
22
t: 8.140274938683989 | objective: 129376.36999910967 | classification accuracy: 0.874635568
23
t: 8.954302432552389 | objective: 129034.04604817541 | classification accuracy: 0.876093294
24
t: 9.849732675807628 | objective: 128691.66158644333 | classification accuracy: 0.876093294


Mu: 1.5
```

```
0
t: 10.834705943388391 | objective: 128349.22214617081 | classification accuracy: 0.87755102
1
t: 16.252058915082586 | objective: 128006.59674384505 | classification accuracy: 0.87755102
2
t: 24.37808837262388 | objective: 127663.84731847458 | classification accuracy: 0.877551020
3
t: 36.56713255893582 | objective: 127321.0150638374 | classification accuracy: 0.8775510204
4
t: 54.85069883840373 | objective: 126978.12744488848 | classification accuracy: 0.877551020
5
t: 82.2760482576056 | objective: 126635.2027984988 | classification accuracy: 0.87827988338
6
t: 123.41407238640839 | objective: 126292.25337927244 | classification accuracy: 0.87827988
7
t: 185.12110857961258 | objective: 125949.28738480997 | classification accuracy: 0.87827988
8
t: 277.68166286941886 | objective: 125606.31030662653 | classification accuracy: 0.87827988
9
t: 416.52249430412826 | objective: 125263.32583610328 | classification accuracy: 0.87827988
10
t: 624.7837414561924 | objective: 124920.33648554892 | classification accuracy: 0.878279883
11
t: 937.1756121842886 | objective: 124577.34408019487 | classification accuracy: 0.878279883
12
t: 1405.763418276433 | objective: 124234.35051672532 | classification accuracy: 0.879008746
13
t: 2108.6451274146493 | objective: 123891.35685854408 | classification accuracy: 0.87973760
14
t: 3162.967691121974 | objective: 123548.36238182496 | classification accuracy: 0.881195335
15
t: 4744.451536682961 | objective: 123205.36730142016 | classification accuracy: 0.881195335
16
t: 7116.677305024441 | objective: 122862.37181748544 | classification accuracy: 0.881195335
17
t: 10675.015957536662 | objective: 122519.3760639305 | classification accuracy: 0.881195335
18
t: 16012.523936304991 | objective: 122176.38013028956 | classification accuracy: 0.88119533
19
t: 24018.785904457487 | objective: 121833.38407643854 | classification accuracy: 0.88119533
20
t: 36028.17885668623 | objective: 121490.38794242527 | classification accuracy: 0.881924198
21
t: 54042.26828502935 | objective: 121147.39175503905 | classification accuracy: 0.881195335
22
t: 81063.40242754403 | objective: 120804.39553220168 | classification accuracy: 0.881195335
23
```

```
t: 121595.10364131605 | objective: 120461.39928590394 | classification accuracy: 0.88192419
24
t: 182392.65546197406 | objective: 120118.40302417018 | classification accuracy: 0.88265306


Mu: 2
0
t: 273588.9831929611 | objective: 119775.40675236995 | classification accuracy: 0.882653061
1
t: 547177.9663859222 | objective: 119432.4104694656 | classification accuracy: 0.8826530612
2
t: 1094355.9327718443 | objective: 119089.41418229898 | classification accuracy: 0.88265306
3
t: 2188711.8655436886 | objective: 118746.41789338531 | classification accuracy: 0.88338192
4
t: 4377423.731087377 | objective: 118403.42160399725 | classification accuracy: 0.884110787
5
t: 8754847.462174755 | objective: 118060.42531469913 | classification accuracy: 0.884839650
6
t: 17509694.92434951 | objective: 117717.42519839175 | classification accuracy: 0.884839650
7
t: 35019389.84869902 | objective: 117374.4250817147 | classification accuracy: 0.8848396501
8
t: 70038779.69739804 | objective: 117031.42496508779 | classification accuracy: 0.884839650
9
t: 140077559.39479607 | objective: 116688.42484848999 | classification accuracy: 0.88483965
10
t: 280155118.78959215 | objective: 116345.42473177887 | classification accuracy: 0.88483965
11
t: 560310237.5791843 | objective: 116002.42461507906 | classification accuracy: 0.884839650
12
t: 1120620475.1583686 | objective: 115659.42449840144 | classification accuracy: 0.88483965
13
t: 2241240950.316737 | objective: 115316.42438174725 | classification accuracy: 0.884839650
14
t: 4482481900.633474 | objective: 114973.42426511573 | classification accuracy: 0.884839650
15
t: 8964963801.266949 | objective: 114630.42414850737 | classification accuracy: 0.884839650
16
t: 17929927602.533897 | objective: 114287.42403192236 | classification accuracy: 0.88483965
17
t: 35859855205.067795 | objective: 113944.42391536062 | classification accuracy: 0.88483965
18
t: 71719710410.13559 | objective: 113601.42379882222 | classification accuracy: 0.884839650
19
t: 143439420820.27118 | objective: 113258.4236823071 | classification accuracy: 0.884839650
20
```

```
t: 286878841640.54236 | objective: 112915.42356581529 | classification accuracy: 0.88483965
21
t: 573757683281.0847 | objective: 112572.42344934672 | classification accuracy: 0.884839650
22
t: 1147515366562.1694 | objective: 112229.42333290147 | classification accuracy: 0.88483965
23
t: 2295030733124.339 | objective: 111886.4232164795 | classification accuracy: 0.8848396501
24
t: 4590061466248.678 | objective: 111543.42310008076 | classification accuracy: 0.884839650


Mu: 5
0
t: 9180122932497.355 | objective: 111200.42298370531 | classification accuracy: 0.884839650
1
t: 45900614662486.78 | objective: 110857.42286735312 | classification accuracy: 0.884839650
2
t: 229503073312433.9 | objective: 110514.4227510242 | classification accuracy: 0.8848396501
3
t: 1147515366562169.5 | objective: 110171.42263471855 | classification accuracy: 0.88483965
4
t: 5737576832810848.0 | objective: 109828.42251843613 | classification accuracy: 0.88483965
5
t: 2.868788416405424e+16 | objective: 109485.42240217693 | classification accuracy: 0.88483

6
t: 1.434394208202712e+17 | objective: 109142.42228594101 | classification accuracy: 0.88483
7
t: 7.17197104101356e+17 | objective: 108799.4221697283 | classification accuracy: 0.8848396
8
t: 3.58598552050678e+18 | objective: 108456.42205353882 | classification accuracy: 0.884839
9
t: 1.79299276025339e+19 | objective: 108113.42193737258 | classification accuracy: 0.884839
10
t: 8.96496380126695e+19 | objective: 107770.42182122957 | classification accuracy: 0.884839
11
t: 4.482481900633475e+20 | objective: 107427.42170510975 | classification accuracy: 0.88483
12
t: 2.2412409503167375e+21 | objective: 107084.42158901317 | classification accuracy: 0.8848
13
t: 1.1206204751583688e+22 | objective: 106741.42147293978 | classification accuracy: 0.8848
14
t: 5.603102375791843e+22 | objective: 106398.42135688958 | classification accuracy: 0.8848
15
t: 2.8015511878959216e+23 | objective: 106055.4212408626 | classification accuracy: 0.88483
16
t: 1.4007755939479608e+24 | objective: 105712.4211248588 | classification accuracy: 0.88483
```

```
17
t: 7.003877969739804e+24 | objective: 105369.42100887818 | classification accuracy: 0.88483
18
t: 3.501938984869902e+25 | objective: 105026.42089292077 | classification accuracy: 0.88483
19
t: 1.750969492434951e+26 | objective: 104683.42077698653 | classification accuracy: 0.88483
20
t: 8.754847462174755e+26 | objective: 104340.42066107546 | classification accuracy: 0.88483
21
t: 4.377423731087377e+27 | objective: 103997.42054518756 | classification accuracy: 0.88483
22
t: 2.1887118655436885e+28 | objective: 103654.42042932284 | classification accuracy: 0.8848
23
t: 1.0943559327718443e+29 | objective: 103311.42031348129 | classification accuracy: 0.8848
24
t: 5.471779663859221e+29 | objective: 102968.42019766285 | classification accuracy: 0.88483


Mu: 10
10
0
t: 2.735889831929611e+30 | objective: 102625.42008186759 | classification accuracy: 0.88483
1
t: 2.735889831929611e+31 | objective: 102282.41996609548 | classification accuracy: 0.88483
2
t: 2.735889831929611e+32 | objective: 101939.4198503465 | classification accuracy: 0.884839
3
t: 2.7358898319296114e+33 | objective: 101596.4197346207 | classification accuracy: 0.88483
4
t: 2.735889831929611e+34 | objective: 101253.41961891798 | classification accuracy: 0.88483
5
t: 2.735889831929611e+35 | objective: 100910.41950323842 | classification accuracy: 0.88411
6
t: 2.735889831929611e+36 | objective: 100567.41938758196 | classification accuracy: 0.88411
7
t: 2.7358898319296115e+37 | objective: 100224.41927194865 | classification accuracy: 0.8841
8
t: 2.7358898319296115e+38 | objective: 99881.41915633842 | classification accuracy: 0.88411
9
t: 2.7358898319296115e+39 | objective: 99538.41904075131 | classification accuracy: 0.88411
10
t: 2.7358898319296115e+40 | objective: 99195.4189251873 | classification accuracy: 0.884110
11
t: 2.7358898319296115e+41 | objective: 98852.4188096464 | classification accuracy: 0.884110
12
t: 2.7358898319296114e+42 | objective: 98509.4186941286 | classification accuracy: 0.884110
13
```

```
t: 2.7358898319296114e+43 | objective: 98166.41857863391 | classification accuracy: 0.88411
14
t: 2.7358898319296113e+44 | objective: 97823.41846316229 | classification accuracy: 0.88338
15
t: 2.7358898319296113e+45 | objective: 97480.41834771374 | classification accuracy: 0.88338
16
t: 2.7358898319296115e+46 | objective: 97137.41823228828 | classification accuracy: 0.88338
17
t: 2.7358898319296115e+47 | objective: 96794.41811688589 | classification accuracy: 0.88338
18
t: 2.7358898319296115e+48 | objective: 96451.41800150656 | classification accuracy: 0.88338
19
t: 2.7358898319296116e+49 | objective: 96108.41788615032 | classification accuracy: 0.88338
20
t: 2.7358898319296115e+50 | objective: 95765.41777081713 | classification accuracy: 0.88338
21
t: 2.7358898319296116e+51 | objective: 95422.41765550697 | classification accuracy: 0.88338
22
t: 2.735889831929612e+52 | objective: 95079.41754021989 | classification accuracy: 0.883381
23
t: 2.7358898319296115e+53 | objective: 94736.41742495586 | classification accuracy: 0.88338
24
t: 2.7358898319296115e+54 | objective: 94393.41730971485 | classification accuracy: 0.88338
```