Lily Bernard au collège Learngame-HC

Contexte

Ludo Bernard est directeur au collège Learngame-HC et souhaite les meilleurs résultats à sa fille Lily Bernard, étudiante de dernière année au collège. Ici, tout ce qui compte est la cote z, l'une des mesures les plus courantes en statistiques pour mesurer et comparer des performances entre élèves. C'est la cote z qui compte pour les demandes d'admissions à l'université.

Pour une matière, si n est le nombre d'étudiants et $x_1, x_2 \dots x_n$ sont les notes finales des étudiants, la moyenne et l'écart-type de la cohorte sont calculés comme suit:

$$\begin{aligned}\mu =& \frac{1}{n}\sum_{i=1}^{n} x_i \\ \sigma =& \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2} \\ =& \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \mu^2}\end{aligned}$$

Puis la cote z d'un étudiant A est calculée comme suit:

$$z_A = \frac{x_A - \mu}{\sigma}$$

Dans le cours de mathématiques, les notes finales de tous les étudiants ont été enregistrées. Lily a eu une bonne note, si bien que sa cote z est aussi très bonne. Par contre, puisqu'elle souhaite étudier en médecine, son père Ludo est inquiet et se demande si sa cote z sera suffisante.

Son père étant directeur, il décide de trafiquer les notes des étudiants afin d'améliorer la cote z de Lily. Par contre, pour ne pas éveiller les soupçons, il ne doit pas modifier les notes n'importe comment. Il se donne 4 règles afin d'éviter d'être démasqué:

1. **Ne pas modifier la note de Lily.** En effet, comme c'est sa fille (conflit d'intérêt), sa note risque d'être vérifiée par une tierce personne.
2. **Ne baisser la note d'aucun étudiant.** En effet, un étudiant surpris négativement par sa note a plus de chance de demander une révision, ce qui exposerait la magouille.
3. **Si un étudiant A a une note supérieure à un étudiant B avant la tricherie, alors l'étudiant A doit avoir une note supérieure ou égale à l'étudiant B après tricherie.** Si deux amis A et B qui se parlent beaucoup ont l'impression que A a mieux fait lors des évaluations, mais que B termine pourtant avec une meilleure note, ils risquent de soupçonner une magouille.
4. **Les notes doivent demeurer des multiples de 0.01%.** En effet, le format des notes ne permet pas une précision supérieure à un centième de pourcentage.

Bref, le directeur se permet de majorer les notes des étudiants de son choix, tant que l'ordre des notes est préservé et que la note de sa fille reste inchangée.

Votre mission

Le fichier data.txt contient les notes vierges des étudiants. Votre mission consiste à modifier les notes en respectant les 4 contraintes précédentes, de manière à optimiser la cote z de Lily.

Flag

Le ficher 'flag-steps-fake.txt' sert d'exemple pour vous indiquer comment créer le FLAG. À une exception près, reproduisez exactement les étapes dans 'flag-steps-fake.txt'.

L'exception est évidemment que vous devez changer les notes des étudiants avant d'appliquer les étapes.

Solutionnaire

Théorie

Observons l'expression suivante qui donne la cote z de Lily:

$$z_L = \frac{x_L - \mu}{\sigma}$$

D'un point de vue intuitif, la manière la plus simple de l'améliorer est d'augmenter x_L, mais on n'a pas le droit. Une autre manière simple est de diminuer μ, mais c'est impossible, car les règles nous empêchent de baisser la note d'un étudiant. Notre dernier moyen est donc de faire baisser σ. Comment? En augmentant les notes des étudiants les plus faibles. Par contre, il ne faut pas augmenter les notes des étudiants faibles infiniment car il ne faut pas trop faire augmenter μ en même temps.

D'un point de vue plus technique, il y a trop de possibilités pour tenter un bruteforce maintenant. Il faut encore réduire théoriquement le nombre de possibilités.

Principe de base en optimisation: si une amélioration est possible, c'est qu'on n'est pas dans un état optimal. Dans notre cas, s'il est possible de faire baisser l'écart-type sans modifier la moyenne, alors ceci est une amélioration, donc l'état actuel n'est pas optimal.

La question est: comment diminuer l'écart-type sans modifier la moyenne? Voici comment:

En rapprochant deux notes. Cela veut dire, prendre 2 notes a **et** b **telles que** $a + 0.02 \leq b$**, puis, changer les notes par** $a' = a + 0.01$ **et** $b' = b - 0.01$. On va appeler cette opération un **rapprochement**.

Il est évident que cette opération ne change pas la moyenne, mais la preuve qu'elle diminue l'écart-type sera laissée à la fin pour ne pas casser le rythme de lecture.

Cependant, il n'est pas toujours légal de faire un rapprochement. En effet, aucune note ne doit baisser. Par contre, si on décide de majorer deux notes, alors la différence entre les deux nouvelles notes doit être de 0 ou 0.01, car si la différence dépasse 0.01, il est possible de faire un rapprochement entre les deux notes, ce qui améliorerait z_L!

Bref, on arrive au résultat suivant: **La différence entre la plus grande et la plus petite de toutes les notes majorées ne doit pas dépasser** 0.01. Bref, notre nouvel ensemble de notes majorées devra ressembler à ceci: on fixe un plancher p. Toutes les notes

inférieures p seront majorées à p ou $p+0.01$. Les notes qui étaient déjà supérieures à $p+0.01$ resteront inchangées.

Plus en détail, parmi les notes majorées, le nombre de notes k à majorer à p plutôt que $p+0.01$ reste à déterminer, mais une fois ce nombre déterminé, les notes à majorer à p sont celles qui étaient les k plus faibles avant majoration. Ainsi, les seuls paramètres à choisir pour optimiser notre cote z est le plancher p et ce nombre k. Cela réduit beaucoup notre espace de recherche!

Code

Voir le code python dans les fichiers fournis. Ce code trouve p et k qui maximisent z_L. Note: p et k sont presque toujours nommés "floor" et "jump_idx" dans le code, respectivement.

Finalement, après exécution du code, il s'avère que la solution finale est beaucoup plus élégante que prévu: en fait, toutes les notes majorées doivent être égales et il n'y a donc pas d'écart de 0.01 entre elles. Le k optimal est donc littéralement le nombre de notes majorées. On n'a pas trouvé de preuve théorique de la raison pour laquelle c'est comme tel.

Bref, le code trouve que **majorer à** 84.11% **toutes les notes inférieures à cette valeur** maximise z_L.

Commentaires à propos du code

Comment travailler uniquement avec des entiers

Une des questions qui risque d'être posée par les participants est comment éviter de travailler avec des nombres flottants, afin d'éviter les imprécisions. Et bien, voici comment.

Le code travaille avec les notes multipliées par 100. Cela ne change pas la cote z puisque ça multiplie la note de Lily, la moyenne ainsi que l'écart-type tous par 100. Donc, le dénominateur et le numérateur seront tous deux multipliés par 100 ci-dessous:

$$z_L = \frac{x_L - \mu}{\sigma}$$

Ce qui ne change donc pas la valeur du ratio.

Ensuite, l'autre source de nombres à virgules laids est qu'il y a une racine carrée dans le calcul de l'écart-type. L'astuce, ici, est d'optimiser z_L^2 au lieu de z_L, c'est équivalent. Si on note Q la somme des carrés des notes:

$$z_L = \frac{x_L - \mu}{\sqrt{\frac{Q}{n} - \mu^2}}$$
$$z_L^2 = \frac{(x_L - \mu)^2}{\frac{Q}{n} - \mu^2}$$

Et voilà, fini les nombres irrationnels. Mais on a encore une forme non-entier sur non-entier. Pour régler ça, on peut le numérateur et le dénominateur de la fraction par n^2. Si on note P la somme des notes:

$$z_L^2 = \frac{(x_L - \mu)^2}{\frac{Q}{n} - \mu^2}$$
$$= \frac{(Nx_L - P)^2}{NQ - P^2}$$

Maintenant z_L^2 est sous la forme entier/entier. Pour l'optimisation, il est facile de comparer deux fractions a et b représentant une cote z pour trouver la plus grande:

(a.num * b.denom) versus (b.num * a.denom)

L'erreur la plus fréquente

Cette section a été rajoutée le lendemain des CSGames 2024.

Certaines équipes ont soumis exactement le même flag, qui n'était pas le bon. Vous êtes arrivé à une réponse presque pareille, qui était aussi de majorer les notes en dessous d'un certain plancher à ce plancher, sans modifier les autres notes... sauf qu'elles ont trouvé un plancher de 84.08% au lieu de 84.11%.

Le problème est que votre algorithme essayait sûrement de faire de modifications de notes une par une, jusqu'à ce qu'aucune note ne puisse être modifiée individuellement en améliorant la cote z. Le problème est que ce genre d'algorithme ("local search") peut s'arrêter dans un maximum local qui n'est pas un maximum global. C'est ce qui vous est arrivé. À partir d'un plancher de 84.08%, tenter de modifier une note individuellement empire la cote z, alors votre algorithme ne voulait plus bouger.

Le code fourni possède l'algo ci-dessus. Il n'est pas appelé par défaut; il faut changer MINISTEPSALGO à True dans cst. Si vous le faites, vous verrez l'erreur sur la console.

Réponse

Pour obtenir le flag, il faut générer le même string que dans data.txt, mais en remplaçant par 84.11% les notes qui y sont inférieures. Ensuite, calculez sha1(string-modifié)

FLAG{70dd5eb985dca345fae297712542e61bf0ee7cfa}

Annexe

On avait remis à plus tard la preuve qu'un rapprochement diminue l'écart-type. La voici:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \mu^2}$$

En reprenant nos deux notes a et b telles que $a + 0.02 \leq b$, on se demandait si les rapprocher en les changeant pour $a' = a + 0.01$ et $b' = b - 0.01$ diminuait l'écart-type. Et bien, dans la formule ci-dessus, la contribution à la sommation des anciennes notes est $a^2 + b^2$, tandis qu'elle est $a'^2 + b'^2$ après rapprochement. Le reste de la formule est inchangé. Pour prouver que l'écart-type diminue, il suffit de prouver que $a'^2 + b'^2 < a^2 + b^2$.

$$
\begin{aligned}
&(a'^2 + b'^2) - (a^2 + b^2) \\
=&(a'^2 - a^2) + (b'^2 - b^2) \\
=&(a' - a)(a' + a) + (b' - b)(b' + b) \\
=&0.01(a' + a) - 0.01(b' + b) \\
=&0.01(a' + a - b' - b) \\
<&0
\end{aligned}
$$

CQFD

Lien pour le code

https://github.com/desharnc27/incoming-csg-2024-python