Automatizing stint assigning process

The "Vegas" Team:
Cezary Klimczuk, William Louth, Gustavs Zilgalvis

March 17, 2019

1 Introduction

To automatize the process of assigning stints to students, we need a universal measure of how good a certain student is at performing certain type of tasks. To do so, all jobs have to have an assigned category. Ideally, we would be looking at a measure j_q that would describe the ability of student j to perform a type of work q. However, because different businesses are characterized by various distribution of grades (i.e. some jobs are easier than others) the grades they give need to be normalized with respect to that distribution. Once we have normalized the grades, we can implement the algorithm that for every free stint $s \in S$ assigns a student j so that we maximize the expected grade for all stints.

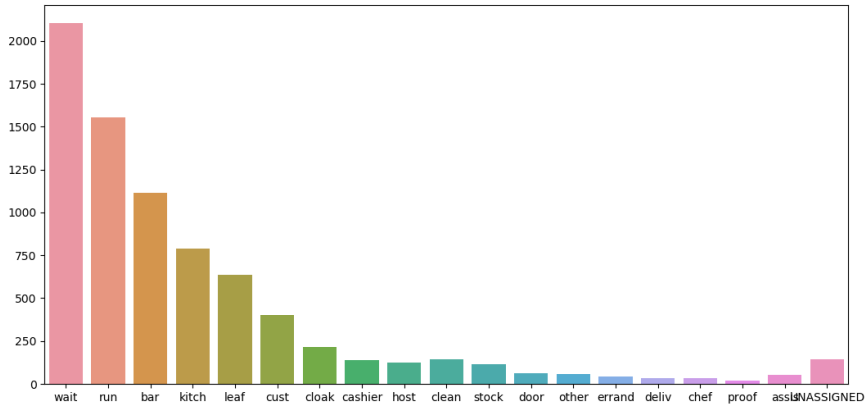
2 Job Types

We created a code that based on the job name assigns it to a certain type. Having n job types, we use historical data to calculate the average score (the concept of *score* will be introduced in the next section) that each student has in a certain type of job. E.g. student X has an average score of 0.965 at running, 0.988 at waiting and so on. Trying to assess whether a student is fit for the job, we first determine to which category does the job belong and then look what's the student's average *score* for this type of job.

3 Creating different types of jobs

We noticed that the Pareto principle takes place when it comes to dividing jobs into categories. A significant proportion of jobs appearing on the system belongs to a certain type. We created a piece of code that assigns a job to one of the 17 categories. These 17 categories cover over 98.1% of all stints.

Figure 1: The distribution of stints by type



4 Grades Normalization. Scores.

Each business may have its own 'way' of rating the student. Some might be very harsh, the other ones can give straight 5's. Our task is to normalize the grade given by a business to some measure we called a *score*. We keep track of businesses' reviews and store it in the database. Every time a grade is given the algorithm checks the distribution of grades of a certain business and calculates the percentile score from the distribution according to the formula

$$\frac{\sum_{i=1}^{k} i \times d_i}{\sum_{i=1}^{5} i \times d_i}$$

Where k is the grade given by a company, and d_i is the density of grade i given by the company. Using this formula we make sure that getting a certain grade reflects the performance in comparison to the wider population. However, it has some limitations for newer businesses, so we came up with an arbitrary limit of > 20 stints when this formula comes into place. For businesses with < 20 stints, the global distribution of grades is assumed.

Figure 2: Business A review distribution (harsh)

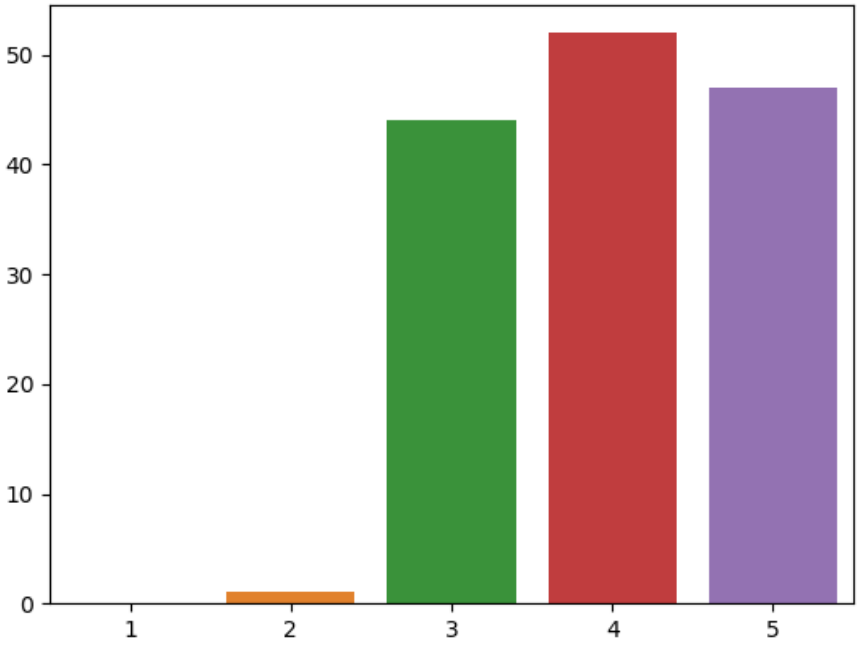
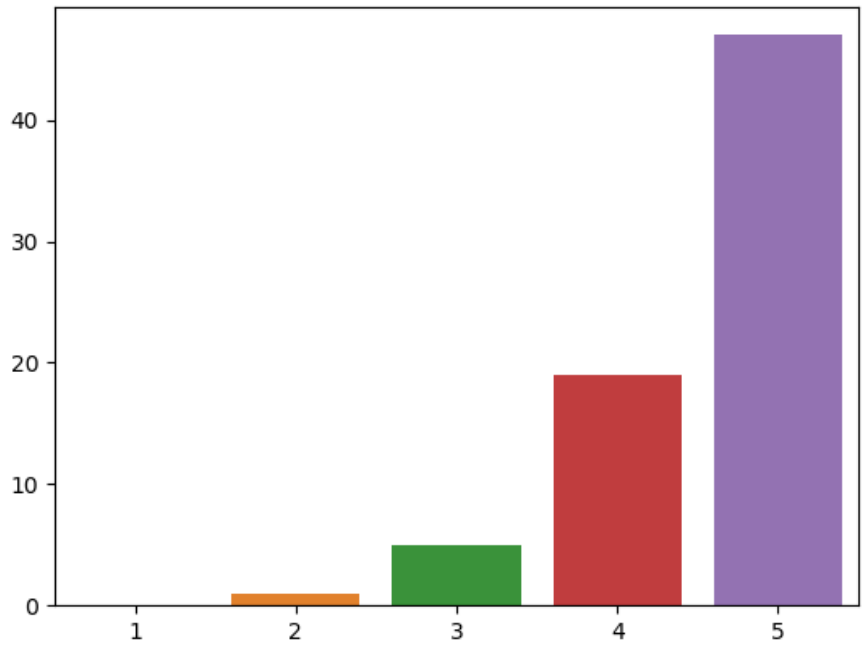


Figure 3: Business B review distribution (generous, similar to global)

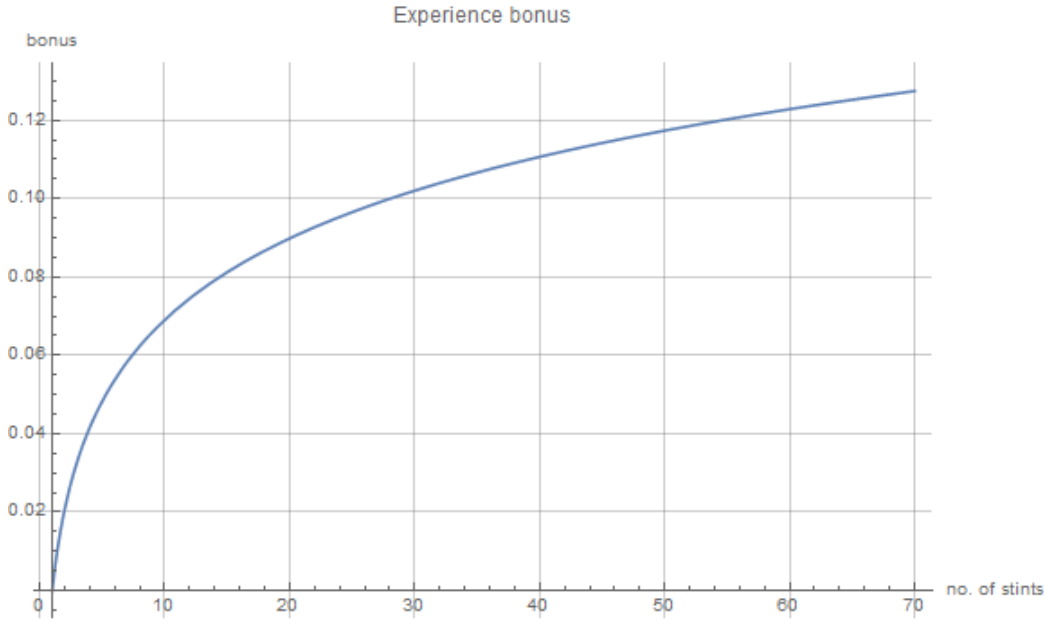


After the normalized score is given, we use it to calculate the average score in each type of work for every student. Students, that have not been assigned a stint from a certain type of work, will have a score of equal to the average of their scores from all other types. If a student is new, their score for each type of work will be 0.6. The 0.6 should give encourage the system to pick the new students to try them out. This value should be easily customizable.

We also included other factors like the experience factor e depending on the number of stints done in the type of work q.

$$e_q = 0.03 \ln (n_q)$$

Figure 4: Bonus function with respect to experience



Other factor taken into account is the mean score of the student from all stints divided by 7.5, which is added to the scores from particular types.

$$j_q + \frac{\bar{j}}{7.5}$$

Another factor is the distance d, that is being calculated using the haversine formula and longitude & latitude data. The distance bonus delta is calculated based on the formula

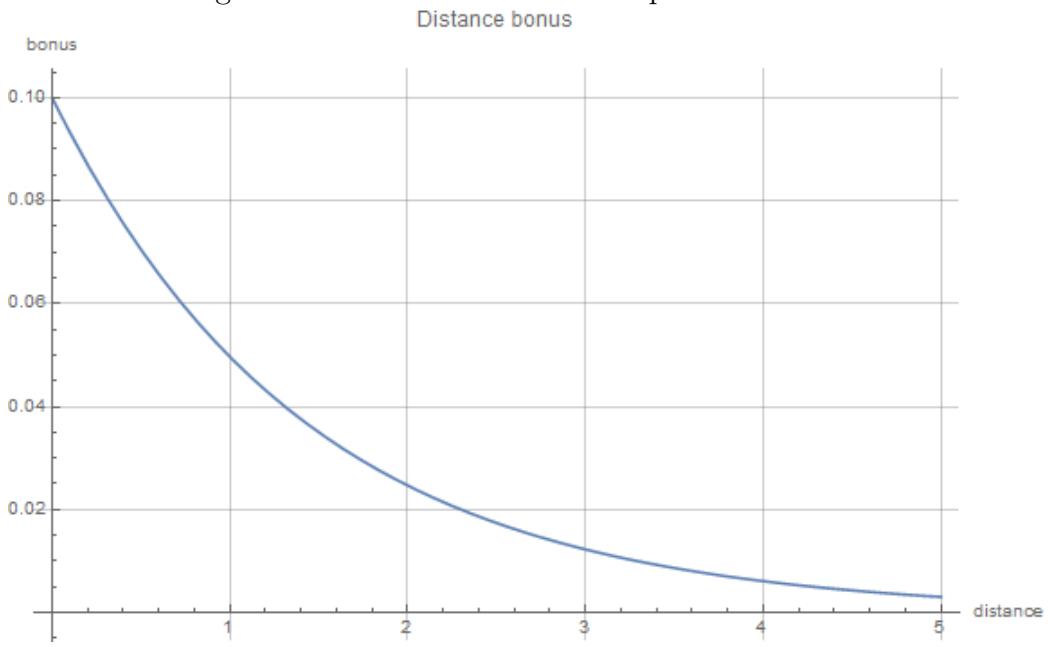
$$\delta = 0.1e^{-0.7d}$$

The distance d is calculated using the formula

$$d = 2r \arcsin\left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\text{hav}(\lambda_2 - \lambda_1)}\right)$$

Where (φ_i, λ_i) is the latitude and longitude of point i

Figure 5: Bonus function with respect to distance



5 The Matching Algorithm

The algorithm takes available students and free stints at any given time. It creates a matrix with rows representing students and columns representing available stints. The entries represent student j's score at type of stint s.

$$M = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1s} \\ r_{21} & r_{22} & \dots & r_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ r_{j1} & r_{j2} & \dots & r_{js} \end{pmatrix}$$

If students are not available within the time frame given by the company, their entry automatically becomes NaN. We built a tool resolves the problem of overlapping student's availability. Also, if the job is coming from the business of rank 4, the entry of a student with no experience becomes NaN as well.

The algorithm looks for the smallest value in the entire matrix and crosses it out. Let's say its r_{j2}

$$M = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1s} \\ r_{21} & r_{22} & \dots & r_{2s} \\ \vdots & \vdots & \vdots & \vdots \\ r_{j1} & & \dots & r_{js} \end{pmatrix}$$

The following process continues until the stint is left with only one available student. Then we cross out (practically turning all student values to NaN) all entries for this student and the process continues.This ensures that all stints will be assigned to the best students available at any given instance.

If the algorithm finds itself in a situation where there is equal or smaller number of students than available stints, it will also assign a job to a student, which is left with no other options to maximize the number of stints filled.

6 Future Optimization

This automation brings an opportunity to test which parameters are effective and which ones need further tuning. The code can be easily adjusted if we decide to put more emphasis on experience/distance. The pairing algorithm is the most effective one in terms of maximizing the expected performance of students, but it is yet to investigate whether it can run faster. For now, it can handle matching 1000 students with 20 stints within a reasonable time. The number of times this algorithm runs needs to be adjusted according to the demand at any given day.