Optimizing Pokémon Po

Tyler Dae Devlin
Daniel Kunin
Daniel Xiang

November 6, 2016

Contents

1 Nontechnical Summary

Pokémon are on the loose, harassing pedestrians, and spreading mischief in Pallet Town, a 10 by 10 grid measuring 4 miles by 4 miles. One seems to appear every 30 minutes or so, and they range from being worthless to very valuable, on a scale of 1 (common/worthless) to 20 (rare/valuable). Once a Pokémon appears at a location in our town, it remains there for 15 minutes and then disappears. Our goal was to create the perfect Pokémon trainer, one to catch 'em all, who plays efficiently and optimally.

Our first step was to formulate a probabilistic model that would allow us to make predictions about when and where Pokémon might appear in the town. We used this model as well as data on past Pokémon appearances to construct an algorithm that would allow our trainer to make optimal decisions about where to travel to at any given moment. The main idea is as follows: if there is a Pokémon nearby, the player should go catch it; otherwise, the player should move to a location where Pokémon are likely to appear in the future.

The concept is straightforward but effective. We tested our algorithm against a simpler algorithm in which the trainer does not strategically move to Pokémon hot spots. Our algorithm achieved scores that were more than five times higher than the simple algorithm on average! We created a dynamic visualization to illustrate the logic of our algorithm and its advantages over the simpler approach. Feel free to explore the visualization at this link: `pokemonpo.getforge.io`.

In summary, we developed a mathematical model for the distribution of Pokémon, an algorithm for optimizing the total point value of Pokémon caught, and an interactive simulation of our trainer's behavior. The trainer equipped with this mathematical toolset is indeed prepared to catch 'em all!

2 Introduction

Pokémon appear randomly in time across a 10 by 10 grid representing the city. Some are worth more than others, with the more valuable ones appearing less frequently than the lower valued Pokémon. Once a Pokémon appears, it is capturable for 15 minutes, and then it will disappear. What this means for our algorithm is that we should only chase Pokémon that spawn within 15 minutes of travel time of our current location.

Since the values of realized Pokémon and their positions are inherently random, our objective is to maximize the number of points we get on average. In other words, we would like our algorithm to incorporate the likelihoods of certain regions that yield Pokémon more frequently and of higher value.

Our project involves three main pieces, a mathematical model, an algorithm that employs the model, and a visualization that tests and validates the algorithm. Using the data file provided, we infer distributions on the inter-arrival times between Pokémon appearances. These distributions allowed us to evaluate locations in the map and make more optimal decisions in our algorithm. Although we developed a mathematically precise model to contribute to the algorithm, we implemented a slightly less complex version due to time constraints, but whose main objectives were in line with the mathematical modeling.

Using visualizations we were able to test our algorithm, making sure it was differentiating between high and low probability regions. We created an in time D3 visualization of our algorithms as they ran, and put it online at `pokemonpo.getforge.io` and we encourage the reader to check it out. It is user interactive, so one can modify the movement speeds as well as the strategy the algorithm uses in order to easily grasp the difference between our algorithm and a much more basic one.

Some background research involved finding average walking speeds of pedestrians in an urban setting. Using this information, we set our algorithm to traverse the city at the speed of an average human, thus more closely modeling the true behavior of a Pokémon Po player. We also research testing techniques, such as k fold cross validation, in order to effectively evaluate our algorithm.

3 The Model

We model the 10 by 10 grid by a square lattice graph $G = (V, E)$ with

$$V = \{1, \ldots, 10\} \times \{1, \ldots, 10\},$$
$$E = \{(m, n) \in V : m, n \text{ are adjacent}\}$$

The ij^{th} node in this 10 by 10 lattice graph is associated with a random variable X_{ij} representing the "value" of a node, relative to our current position on the lattice. We lay down the following set up in order to eventually define the "value" X_{ij} of a node.

As we move throughout the lattice, we keep track of our current position, which we define as $(i_{\text{curr}}, j_{\text{curr}})$, where i_{curr} and j_{curr} are the row index and column index of our position in the graph, respectively.

We also note that since the grid is 10 blocks by 10 blocks, and 4 miles by 4 miles, each block is $\frac{4}{10}$ miles. Since the average human walking speed is 3.1 miles per hour,[1] we conclude that the average number of minutes it takes to traverse an edge on our graph is

$$\frac{4}{10} \text{ miles} \cdot \frac{60 \text{ minutes}}{3.1 \text{ miles}} = 7.7 \text{ minutes}.$$

Thus our average speed s in terms of blocks per minute is

$$s = \frac{1 \text{ block}}{7.7 \text{ minutes}} = 0.129 \text{ blocks/min}.$$

Here we make our first basic assumption.

Assumption 1: Our Pokémon trainer walks at a constant speed of .129 blocks/min throughout the 12 hours.

Comment. This doesn't seem that realistic in terms of a human needing to eat and rest, but ignoring these factors to begin with is necessary in order to set up a baseline model. □

Fix a node (i, j). If t is the time it takes to travel from $(i_{\text{curr}}, j_{\text{curr}})$ to (i, j), the number of Pokémon N_t that appear during this time interval t can be written as

$$N_t = \sum_{k=1}^{\infty} \mathbb{1}\{T_k \in [0, t)\},$$

where T_k is the time at which the k^{th} Pokémon appears. For example, if two Pokémon appear before time t, then the indicators would be 0 for all $k \geq 3$, and the sum would evaluate to 2, as desired.

Assumption 2: The inter-arrival times $\{T_{k+1} - T_k\}$ are distributed i.i.d. $N(\mu, \sigma^2)$.

Comment. We initially thought that the Pokémon appeared in time according to a Poisson Process. But if this were true, then the inter-arrival times would be distributed i.i.d. exponential, which was clearly not the case. We visualized the histogram in Figure 1 and saw that it did not resemble an exponential distribution with rate $\lambda = 30$ minutes, a contradiction to the Poisson Process assumption. As a result, we reworked the distribution of Poké appearances subject to normally distributed inter-arrival times (as suggested by Figure 1).

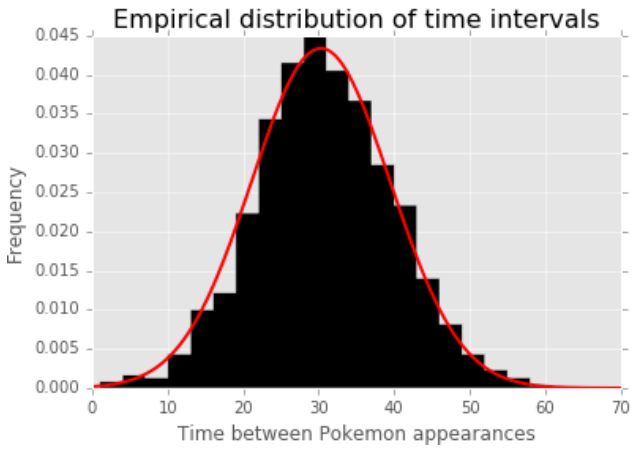

Figure 1: A histogram showing the empirical distribution of inter-arrival times and the pdf of a normal distribution parametrized by the sample mean and sample standard deviation.

We estimate the parameters μ, σ^2 by the sample mean and sample variance,

$$\hat{\mu} \doteq \frac{1}{n-1} \sum_{k=1}^{n-1} (T_{k+1} - T_k) \approx 30.27 \text{ minutes},$$
$$\hat{\sigma}^2 \doteq S^2 \approx 84.69 \text{ minutes}^2.$$

Throughout the rest of the document, when μ and σ^2 are used to specify a distribution, we write them to mean the estimates given above. □

We can write the arrival time T_k of Pokémon k as the sum of the previous k inter-arrival times

$$T_k = \sum_{j=0}^{k-1} (T_{j+1} - T_j),$$

where $T_0 \doteq 0$. Since T_k is the sum of k i.i.d. normal random variables, it is also a normal random variable with distribution

$$T_k \sim N(k\mu, k\sigma^2).^{[2]}$$

Let P be the 10×10 matrix where P_{ij} is the empirical frequency with which Pokémon appear at the ij^{th} spot in our lattice based on the given data file. We assume that there is some true underlying pmf over the nodes in the 10×10 grid that determines where the next Pokémon will appear. Rather than assume some parametric form for this distribution, we simply estimate the point mass of each node according to the empirical frequencies in the matrix P.

It then follows that the number of Pokémon K_{ij} appearing at node (i, j) in a time interval t is distributed according to the following two-stage hierarchical model:

$$\begin{aligned} K_{ij} &\sim \text{Binom}(N_t, P_{ij}), \\ N_t &= \sum_{k=1}^{\infty} \mathbb{1}\{T_k \in [0, t)\}, \qquad T_k \sim \text{Normal}(k\mu, k\sigma^2) \end{aligned}$$

This mixture distribution describes the number of Pokémon N_t that appear in a time interval t, and the number of Pokémon which appear at the ij^{th} node as $\text{Binom}(N_t, P_{ij})$.

We let the random variable S_m denote the point value of the m^{th} Pokémon that appears at the ij^{th} node. The overall value X_{ij} that we seek is then simply the sum of the point values of the K_{ij} Pokémon at that node:

$$X_{ij} = \sum_{m=1}^{K_{ij}} S_m$$

Assumption 3: The random variables $\{S_i\}$ are distributed i.i.d. according to the empirical pmf shown below.

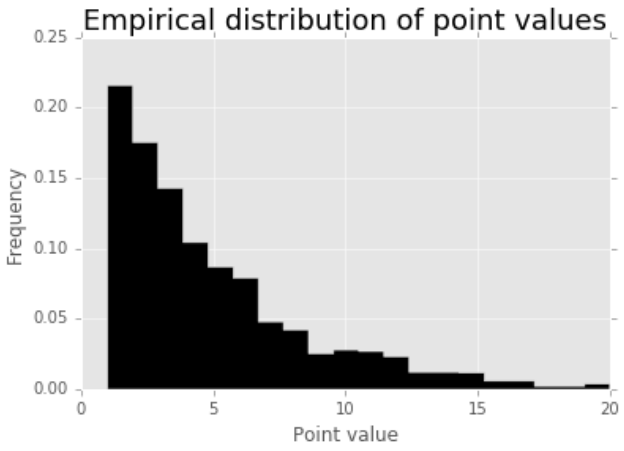

Figure 2: The histogram showing the empirical distribution of Pokémon point values. Observe that higher-value Pokémon appear less frequently.

Comment. The empirical pmf resembles that of a Geometric distribution, but the interpretation of counting the number of failures until success doesn't seem to fit the problem here. The only way in which we use this distribution in the following pages is in calculations involving the mean of this pmf, so it isn't necessary infer a parametric model (e.g. via maximum likelihood) when that model will give precisely the same mean as the empirical pmf. □

We can thus compute the expected value at each node $E(X_{ij})$ as

$$\begin{aligned} E(X_{ij}) &= E\left(\sum_{m=1}^{K_{ij}} S_m\right) \\ &= E\left(E\left(\sum_{m=1}^{k} S_m \mid K_{ij} = k\right)\right) \\ &= E\left(\sum_{m=1}^{K_{ij}} E(S_m)\right). \end{aligned}$$

where the second equality follows from the tower property of conditional expectations. Since we assumed $\{S_m\}$ to be i.i.d., the above is equal to

$$\begin{aligned} E(X_{ij}) &= E\left(\sum_{m=1}^{K_{ij}} E(S_1)\right) \\ &= E(S_1)E(K_{ij}). \end{aligned}$$

Again applying the tower property and replacing $E(S_1)$ with the sample mean of the Pokémon scores, $\bar{S}$, we have

$$\begin{aligned} E(X_{ij}) &= \bar{S}E(E(K_{ij} \mid N_t)) \\ &= \bar{S}E(\text{Binom}(N_t, P_{ij})) \\ &= \bar{S}E(N_t P_{ij}) \\ &= \bar{S}P_{ij}E(N_t). \end{aligned}$$

It remains to compute $E(N_t)$. By definition,

$$E(N_t) = E\Big(\sum_{k=1}^{\infty} \mathbb{1}\{T_k \in [0,t)\}\Big).$$

We wish to justify bringing this expectation into the infinite series. We do so by rewriting the above expression as

$$E(N_t) = \int_{\mathbb{R}} \int_{\mathbb{N}} \mathbb{1}\{T_k \in [0,t)\}\, d\nu\, d\mathbb{P},$$

where ν is the counting measure and $\mathbb{P}$ is the probability measure of N_t. By Theorem 2.37 (a) of reference [3], since the integrand is nonnegative and measurable, we can swap the order of integration to get

$$\begin{aligned} E(N_t) &= \int_{\mathbb{N}} \int_{\mathbb{R}} \mathbb{1}\{T_k \in [0,t)\} \, d\mathbb{P} \, d\nu \\ &= \sum_{k=1}^{\infty} E(\mathbb{1}\{T_k \in [0,t)\}). \end{aligned}$$

Since the expectation of an indicator random variable is simply the probability of that event, this is equal to

$$E(N_t) = \sum_{k=1}^{\infty} \mathbb{P}(T_k \in [0,t)) = \sum_{k=1}^{\infty} [\Phi_{k\mu,k\sigma^2}(t) - \Phi_{k\mu,k\sigma^2}(0)],$$

where $\Phi_{k\mu,k\sigma^2}(x)$ is the cdf of a Normal$(k\mu, k\sigma^2)$ distribution. Plugging this back in for $E(N_t)$, we get

$$E(X_{ij}) = \bar{S} P_{ij} \sum_{k=1}^{\infty} [\Phi_{k\mu,k\sigma^2}(t) - \Phi_{k\mu,k\sigma^2}(0)]. \tag{1}$$

We then compute a 10 by 10 utility matrix U based on these expected values. We define the ij^{th} element in U to be

$$U_{ij} = \frac{V_{ij} + E(X_{ij})}{t},$$

where V_{ij} is defined as

$$V_{ij} = \begin{cases} 0 & \text{if no Pokémon at position } i,j \\ \text{Value of Poké}_{ij} & \text{otherwise} \end{cases}$$

Assumption 4: It is plausible to scale the value of a node by the inverse of the time it would take to reach that node.

Comment. We scale the values at the nodes by $1/t$ in our utility matrix U, since nodes that are farther away will take more time to get to, and are thus more costly to pursue. There are of course other choices of functions of t to scale by (e.g. $1/\log t, 1/t^2$, etc.) but for the sake of simplicity we choose $1/t$. If more time were to be spent on this model, a sensible experiment might be to compare the performances of the algorithm when using different scale factors. □

Now that we've computed the utility matrix U, we feed it into the algorithm to help make decisions on where to move, as described in the following section.

4 The Algorithm

We show the heat maps of the total number of Pokémon that appear in the grid, as well as the total values of those Pokémon below

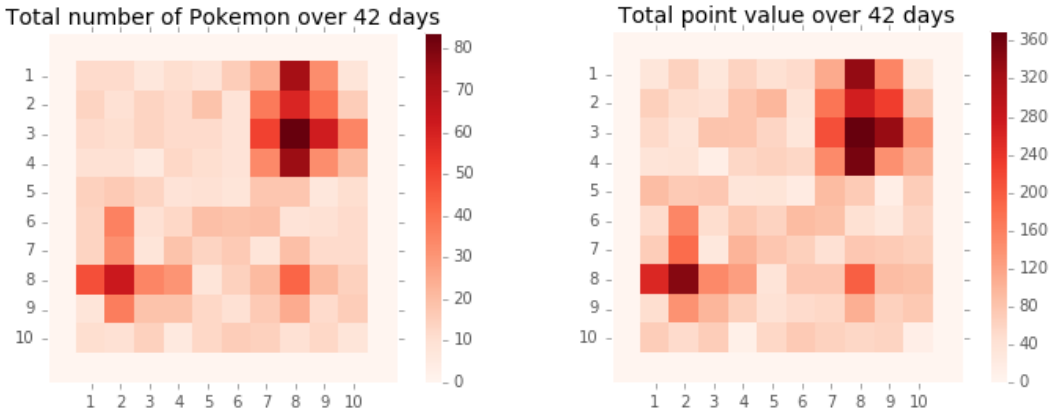

Figure 3: Heat maps to visualize the distribution of the number of Pokémon appearances and the cumulative scores over the playing grid. The entire 42-day dataset was used to create these plots.

Observe that there are 2 main "hotspots" where Pokémon are likely to appear and whose overall values over the 42 days were highest. These hotspots are centered at (8,2) and (3,8). We would like our algorithm to take into account these positions as favorable places to be when there are no Pokémon immediately in range. Our utility matrix is constructed such that we are biased to hover in such regions of high probability mass.

We take the distance between two positions $(i_1, j_1), (i_2, j_2)$ in our matrix to be the Manhattan distance, i.e.

$$d\left((i_1, j_1), (i_2, j_2)\right) = |i_1 - i_2| + |j_1 - j_2|$$

We define the set of nodes $\mathcal{N}^{(t)}(i, j)$ to be

$$\mathcal{N}^{(t)}(i, j) = \left\{(m, n) \in V : \frac{d((i, j), (m, n))}{s} \leq t\right\}$$

In words, $\mathcal{N}^{(t)}(i, j)$ is the set of nodes in our graph to which we can travel within t minutes at a speed of s.

In particular, the set of nodes to which we restrict our attention when catching Pokémon is $\mathcal{N}^{(15)}(i_{\text{curr}}, j_{\text{curr}})$; since Pokémon only last 15 minutes, all Pokémon appearing outside of this 15 minute neighborhood are uncatchable.

Assumption 5: Whenever we encounter a Pokémon, we immediately catch it with 100% probability.

Comment. This is a simplifying assumption that doesn't adhere very strictly to the actual gameplay of Pokémon. In fact, a more complex and accurate model would assign lower capture-success rates to the more rare Pokémon compared to the more common ones. However, it is a reasonable assumption that we found necessary to make for when first writing the algorithm. Given more time, we may have introduced more complexity into the model that would take these issues into consideration. □

Our algorithm is described below in pseudocode.

Algorithm 1 Local Pokémon Po Algorithm

Result: a total score representing the sum of the point values of the Pokémon caught in a 12 hour period.

while current time $<$ 12 hours **do**
  if $\exists$ a Poké at the player's current position **then**
    Catch it!
  end
  for $(m,n) \in \mathcal{N}^{(15)}(i_{\text{curr}}, j_{\text{curr}})$ **do**
    if $\exists$ a Poké at node (m,n) **then**
      Append (Value(Poké),(m,n)) to the list of Pokémon that are catchable from the current position. Call this list $L(i_{\text{curr}}, j_{\text{curr}})$.
    end
  end
  $t \leftarrow$ time required to traverse an edge
  if $L(i_{\text{curr}}, j_{\text{curr}}) \neq \varnothing$ **then**
    $(i^*, j^*) \leftarrow \underset{(m,n)}{\text{argmax}}\{\text{Value(Poké)} : (\text{Value(Poke)}, (m,n)) \in L(i_{\text{curr}}, j_{\text{curr}})\}$
  else
    Compute utility matrix U
    $(i^*, j^*) \leftarrow \underset{(i,j)\in\mathcal{N}^{(t)}(i_{\text{curr}}, j_{\text{curr}})}{\text{argmax}} U_{ij}$
  end
  Take a step towards (i^*, j^*)
  current time $\leftarrow$ current time $+ t$
end

In words, our algorithm keeps track of the current time, and performs a sequence of computations in order to decide where to move next, as long as our current time is less than 12 hours. We loop through the set of nodes to which we can travel within 15 minutes and check to see if there is a Pokémon sitting at one of them. We keep a list of all positions and point values of Pokémon in this neighborho od and traverse an edge in the direction of the most valuable one if this list isn't empty. If the list

is empty, then we compute the utility matrix U, and take a step in the direction of the node with the largest utility value. We repeat this process until the 12 hours are up.

An important feature of this algorithm is that when no Pokémon are within range, the player moves to the highest utility node *among the four neighboring nodes*. In this sense, the algorithm is a **local** hill-climbing algorithm. We were also curious to see whether **global** hill-climbing might be a better strategy. By making a few small modifications to Algorithm 1, we can obtain the following global hill-climbing algorithm.

Algorithm 2 Global Pokémon Po Algorithm

Result: a total score representing the sum of the point values of the Pokémon caught in a 12 hour period.

while current time $<$ 12 hours **do**
  if $\exists$ a Poké at the player's current position **then**
    Catch it!
  end
  for $(m,n) \in \mathcal{N}^{(15)}(i_{\text{curr}}, j_{\text{curr}})$ **do**
    if $\exists$ a Poké at node (m,n) **then**
      Append (Value(Poké),(m,n)) to the list of Pokémon that are catchable from the current position. Call this list $L(i_{\text{curr}}, j_{\text{curr}})$.
    end
  end
  if $L(i_{\text{curr}}, j_{\text{curr}}) \neq \varnothing$ **then**
    $(i^*, j^*) \leftarrow \underset{(m,n)}{\text{argmax}}\{\text{Value(Poké)} : (\text{Value(Poke)}, (m,n)) \in L(i_{\text{curr}}, j_{\text{curr}})\}$
  else
    Compute utility matrix U
    $(i^*, j^*) \leftarrow \underset{(i,j)}{\text{argmax}}\, U_{ij}$
  end
  Take a step towards (i^*, j^*)
  current time $\leftarrow$ current time + time required to traverse an edge
end

The only difference between the global and local algorithms is that in the global algorithm we take the argmax of U_{ij} over all nodes in the graph, rather than merely the four neighboring nodes.

5 Evaluation/Results

In addition to the global and local algorithms specified above, we also created a basic algorithm to serve as a baseline for comparison. The pseudocode for the naïve algorithm is given below.

Algorithm 3 Naïve Algorithm

Result: a total score representing the sum of the point values of the Pokémon caught in a 12 hour period.

while current time < 12 hours **do**
  for $(m, n) \in \mathcal{N}^{(15)}(i_{\text{curr}}, j_{\text{curr}})$ **do**
    if $\exists$ a Poké at node (m, n) **then**
      $(i^*, j^*) \leftarrow (m, n)$
    end
  end
  $t \leftarrow d((i_{\text{curr}}, j_{\text{curr}}), (i^*, j^*))/s$ (i.e., the time to travel to node (i^*, j^*) at speed s)
  Go to (i^*, j^*)
  current time $\leftarrow$ current time $+ t$
end

This algorithm merely scans its 15 minute neighborhood for Pokémon and captures the first one that appears in range, ignoring everything in its path until it does so. Note that this naïve algorithm makes no use of the 42-day dataset.

We implemented all three of the above algorithms—local, global, and naïve—in Python. The implementations are collected in a Jupyter notebook and are accessible at this link: `https://github.com/tylerdevlin/BMCM/blob/master/pokemonPo.ipynb`. (This Jupyter notebook also contains the code used to generate all figures in this report and the predictions we discuss below.)

We used k-fold cross validation, with $k = 10$, to determine how well our algorithms perform on unseen data. In other words, we designated 90% of the dataset as training data used to construct the utility matrix U, and then ran our algorithms on the remaining 10%, i.e. the test data; we repeated this procedure 10 times in such a way that each data point is used exactly once for testing. After normalizing for the time duration of our test data, we obtained scores summarized by the following box plots.

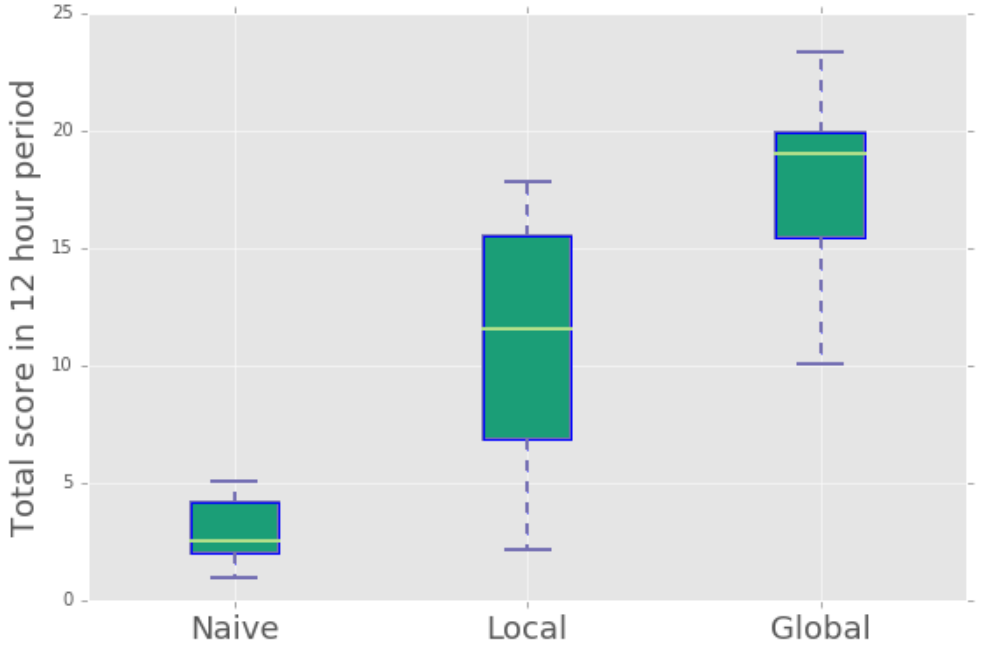

Figure 4: Box plots illustrating the performance of our three algorithms as measured by the total point-value of Pokémon caught in a 12-hour period. The scores were computed via a 10-fold cross validation procedure. Our algorithm used a walking speed of 3.1 miles/hour and node (5,5) as the starting location.

We can see that the global algorithm achieves the highest scores among the three, and both the global and local algorithms significantly outperform the naïve algorithm. We predict that our global algorithm will achieve an expected score of around 17.4 in 12-hour period using a novel unseen dataset.

In addition to the quantitative evaluation of our algorithms' performance via cross validation, we also visually evaluated performance and correctness through our dynamic D3 visualization (`pokemonpo.getforge.io`). The visualization allowed us to test and debug our algorithms on many test cases and verify that our theoretical model led to practical algorithmic results.

6 Conclusion

In summary, we modeled the inter-arrival times of Pokémon as normally distributed. We used the sample mean and variance (i.e. the maximum likelihood estimators for the parameters of a normal distribution) in all of our computations.

Defining the "value" at each node as the sum of the values of the Pokémon that would appear in the time it takes to travel there meant that this "value" was a function of several random variables, each with a different distribution. To compute the expected value, we iterated the functional over conditional expectations to end up with the final expression given by (1). Using this final expression, we computed a utility matrix U whose entries indicated the overall scaled priority of nodes relative to how far they were from the current position.

The second portion of the project was to implement an algorithm that incorporated the model in decision making. We wrote python scripts to implement two algorithms. We first implemented a naive strategy, one which immediately pursues the first Pokémon to appear within a 15 minute radius and ignores all other factors until the Pokémon is caught. The second algorithm takes into account the heat maps shown in Figure 3 and the utility matrix defined in the Modeling section. Instead of staying put when no Pokémon are around, we naturally gravitate to more valuable nodes, where the notion of average value over time is captured by the utility matrix. As a result, we tend towards higher density areas, i.e. "hotspots", so the improved algorithm is more likely to encounter Pokémon in its 15 minute radius, since its 15 minute radius is strategically placed.

We also created a visualization of the game being played in real time. A screen shot of the animation is shown in the figure below.

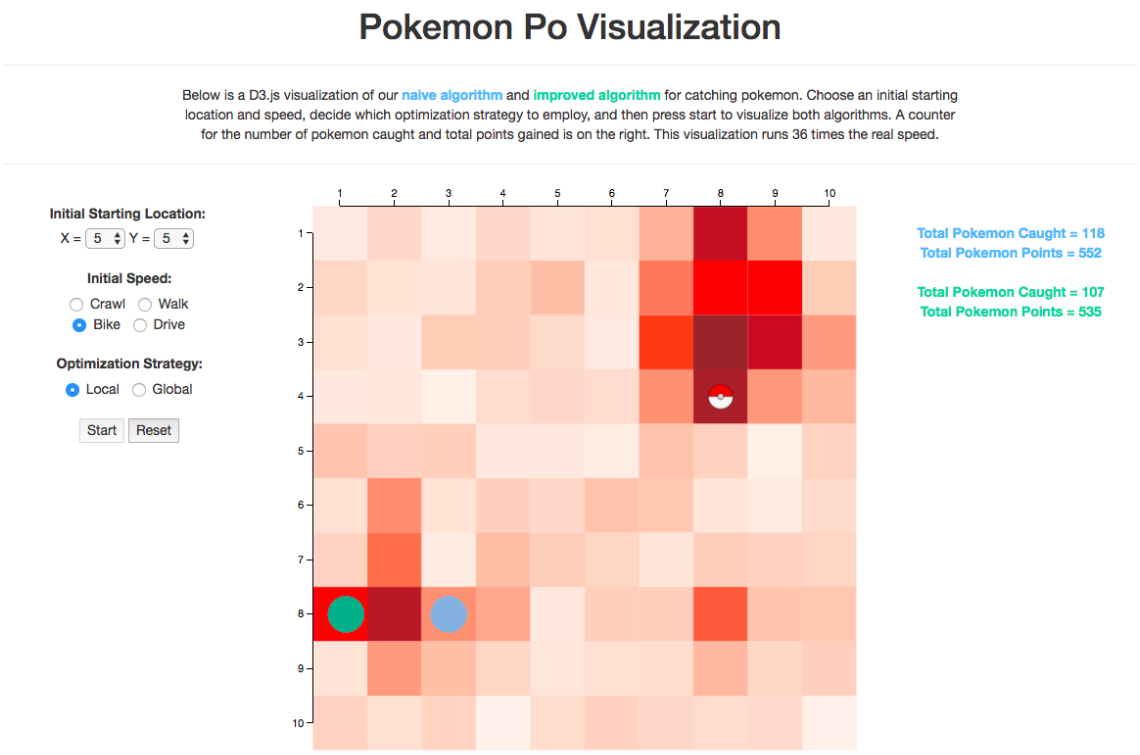

Figure 5: A screen shot of a visualization of the naive algorithm. The URL at which you can access and play with the visualization is `pokemonpo.getforge.io`

6.1 Strengths and Weaknesses

One of the virtues of our model is its adaptability. The mixture model depends on several distributions, but when we found out that one of the distributions we assumed was incorrect, we were easily able to rectify the model with just a few adjustments and calculations. By exploiting the probabilistic nature of the problem, our model is able to assign long term average values to different areas, based on the data file provided, and thus inform our algorithm to make smarter decisions.

The weaknesses of our model lie in the underlying assumptions. In order to move forward in the modeling process, we needed to make simplifying assumptions. Some of them were more justified than others, though all cause the model to deviate from reality, in exchange for plausibility.

Our first assumption was that we move throughout the map at a constant rate of .129 blocks/min. This can easily be adjusted in our code, in fact the visualization on the website allows the user to travel at 4 different rates. To make the model more realistic, one might consider requiring pauses to simulate breaks that a player would need, since most people don't play Pokémon for 12 hours straight.

Our second assumption is a reasonable one, based on the data. We assume that the inter-arrival times are distributed normally. The depicted histogram fits a normal distribution with the sample mean and variance very closely. As a result we are quite confident in this assumption.

Our third assumption was to use the empirical pmf for the distribution on Pokémon point values. Any parametric distribution we infer from this data using maximum likelihood estimation will share the same mean as the empirical pmf (i.e. the sample mean). The only characteristic of the distribution we use in the rest of the project is the mean, so we don't even bother inferring another distribution.

Our fourth assumption is on the definition of our utility matrix. We determine a scaling factor which agrees with our intuition. If there were more time, we could implement the algorithm for several choices of scaling factors, and pick the one which maximizes the overall score.

Our fifth assumption was on the capturability of Pokémon. For simplicity's sake, we modeled all Pokémon as having the same 100% capture rate, and assumed that it took no time to capture the Pokémon. This isn't reflected in the true gameplay, and given more time we would have incorporated more complexity in the model to take these issues into account. However, with time being a limited resource, we found it necessary to make this assumption in constructing our algorithm.

6.2 Further Improvements

There are many places in our strategy that could be experimented with and improved had we more time. An idea we had toyed with but decided would take too long to implement involved the possible paths we could take toward realized Pokémen. If the target node is m horizontal steps and n vertical steps away, then there are

$$\binom{m+n}{n}$$

possible paths to get there. Some of these paths go through higher density areas, and thus are more favorable than the others. This seemed a bit too complicated to implement well in the time we were given, so we decided not to pursue it.

Another improvement we thought of while observing our visualization was a possible alternative objective function. Another plausible way to measure the value of the neighboring nodes could be to simply sum the probability masses that lie on the nodes within a 15 minute radius of our current location. This would lead us away from edges and would also capture the bias toward high density regions that an optimal algorithm should prefer. Since we realized this quite late into the process, we weren't able to modify our implementation. But given more time, this improvement is one of the most important ones to consider.

In conclusion, there are places in both our model and our algorithm that could be improved upon, but would have taken too much time to do so in just a couple of days. Our improved algorithm did indeed outperform the naive one we set up as a baseline, and this can be easily observed using our D3 visualization at `pokemonpo.getforge.io`. In fact, we encourage the reader to do so!

References

[1] Levine, R. V. & Norenzayan, A. (1999). "The Pace of Life in 31 Countries". Journal of Cross-Cultural Psychology. 30 (2): 178–205.

[2] Casella, G., and Berger, R. L. "Statistical Inference". Duxbury Press, 2002. Print.

[3] Folland, Gerald B. "Real Analysis: Modern Techniques and Their Applications". 2d ed. New York: Wiley, 1999. Print.

[4] Hoel, P.G., S. C. Port, and C.J. Stone. "Introduction to Stochastic Processes". Boston: Houghton Mifflin, 1972. Print.

[5] Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. "Learning from Data: A Short Course.". United States: AMLBook.com, 2012. Print.