Bump Matching Through Logistic Regression
CS 229 Final Project

Stanislav Moreinis

Dec. 10, 2010

1 Introduction

This project will focus on applying machine learning to model bump matching for the mobile application Bump. The application aims to connect two networked users through the physical act of bumping their phones together, which sends two separate messages to a central server which are then ideally paired together. This is possible since both phones send the signal at essentially the same time (once they bump), with similar geolocation data, if available, as well some other identifying features. The current approach performs the pairing using fixed thresholds across all populations of bumps, something which may be modeled more accurately by a statistical classification algorithm. Because each handset cannot be assumed to be tightly synchronized to the correct time, the server can only construct a server-local time range during which the bump could have occurred. For a given bump, the set of the potential candidates for a pairing can then be significantly reduced by only considering those whose time intervals intersect (or fail to intersect by some small threshold) the time range of the given bump. For each potential candidate, we can then form a binary classification problem on the two bumps to determine if their features are close enough to have been generated from the same physical event.

2 Classification

To match bumps, we train a model that, given a pair of bumps, can predict whether or not they are close enough to each other to have been caused by the same interaction. To represent the different dimensions of one bump being close to another, we can use a vector of various similarity metrics between the two bumps as the input for an L1-regularized L2-loss support vector classification model (implemented by liblinear), which outputs 1 if the two bumps should match and 0 otherwise. Once a model assigns a label to all of the potential pairs, we scan the list once to require that if the pair (A, B) is matched, (B, A) must also be matched (enforcing symmetry on the matching). We then scan the list again to only match a bump if it has no more than one predicted match pair (which can help us make enforce strict matching on more general criteria). Finally, we perform the first scan again to ensure that symmetry is maintained.

2.1 Location Matching

To measure the geographical distance between bumps, we can use a binary variable which is 1 only if the given bump is the closest among all considered bumps in combination with a continuous measure of $dist/\sqrt{acc}$, where $dist$ is the distance between the two reported GPS locations, and acc is the maximum allowed accuracy given by either of the devices. Since we can't confidently match bump A with B if a third bump C is closer to A than B is, our discrete measure is very effective at filtering out a majority of far away bumps. The distance measure then prevents us from falsely matching the closest bump if it is too far away (i.e. if the original bump has no good match). The exact formula was derived through cursory experimentation on a smaller data set, but if we treat the reported accuracy as a variance of our estimate and assume that the distance between pairs of bumps is drawn from a normal distribution (with errors proportional to the accuracy, or variance), then the probability that a given distance measure was sampled from a distribution with mean 0 can be given by calculating $Z = \frac{X-\mu}{\sigma} = dist/\sqrt{acc}$. Since the matches currently being made have a specific type which shows which matching criteria it fits, we can limit both the training and the testing step to only those bumps which were matched using location information. Since all of these bumps have reported location, a clearer examination of how well the location closeness metric works is to only test it on those bumps which have reported location.

2.2 IP Matching

Since the mobile devices Bump is running on are often connected to wireless networks, we can test whether or not the two bumps came from the same IP address (assuming it's not a loopback address), which would imply that both devices are on the same Wi-Fi network and are much more likely to be paired. While simply adding an indicator variable on whether or not both bumps had the same non-loopback address was fairly simple to allow us to also match bumps with reported location using IP, a big advantage of IP matching was its ability to identify bumps without a reported location. Since the two sets of features (IP and location) were fairly orthogonal to one another, we introduce a binary variable for whether or not both of the bumps had reported their location, so non-matching location metrics would not greatly decrease the chances of predicting an IP-matching bump. If the location was available, then this would 'activate' the location-matching part, and if it had not, then the IP information could still be used to make a match. The addition of this last variable resulted in the success rate across the increased population of bumps rising back to the level consistent with that seen in the previous section.

2.3 Other Features

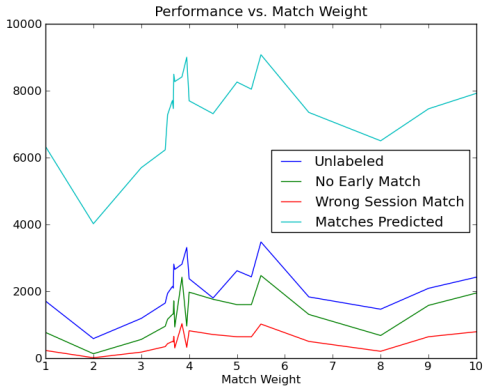
To compare the machine learning solution with the one currently in-place, 2 other metrics considered by the existing process were also added to the feature set, along with an additional indicator variable for each signifying whether or not each one could be

computed. In this stage, we no longer filter the bumps considered by match type or any other criteria, and therefore closely approximate the typical stream of bumps at a given time (albeit with the added luxury of having knowledge of all of the bumps at once instead of receiving it one by one).

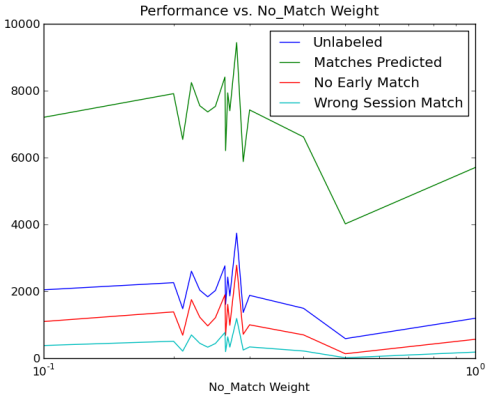
3 Analysis

Using the framework obtained in our last step (with a total of 12 features per each pair of bumps being trained on the entire set), we train a model on an hour of bumps and then test it on the next hour of bumps. To vary the importance of each pair of bumps being matched or not matched, we trained and tested the model on the same pair of hours for a number of different weights for both positive and negative labels. We can see the accuracy of the model shown below. As expected, increasing the weight of positive labels (or decreasing the weight of the negative labels) causes the model to identify more matches, which also tends to increase the amount of server matches that it agrees on. Much as expected, the number of negatively labeled matches grows as well. Some of these can be argued to still be correct matches, and instead labeled incorrectly by the existing system - as in the case of matching A's bump with B's if A's next match this session is B. There is also a significant portion of bumps matched that have no next session match - a symptom of an oversight by the existing matching algorithm, as we assume that most bumps produced had a matching bump that was also transmitted to the server. For those bumps matched which do not fall into either of these categories, we can consider them to be fairly correct false positives.

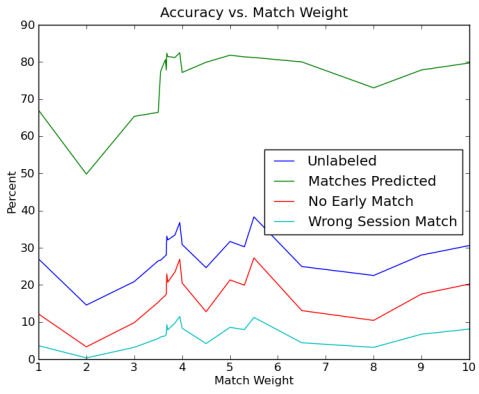
Although we can improve the amount of agreed matches between the machine learning implementation and the existing system, it comes at a cost of more and more false positives. Using 5% of all of the matches created as the maximum false positive rate we tolerate yields us fairly similar optimal results whether it be through adjusting the weights of positively or negatively labelled bump pairs. Using a weight of 0.264 for each pair of bumps that wasn't matched (and a weight of 1 for those that were), the model predicts a total of 7425 matches after all the constraint filters had been applied. Of these, 5546 (80.44%) matches agree with those made by the server. 881 of the unlabeled matches are the next ones in the session for our users, and 661 of the unlabeled ones have no next match in their session.This leaves us with 337 matches that were not labeled by the existing system, nor have any probable cause to believe they should have been labelled otherwise, for a false positive rate of 4.54%. This leaves us with 7088 matches that we can still argue were correct, compared to the 6895 labelled matches. A final note is that only 14 of the matches were rejected by the actual server for having too many potential match candidates, so the machine learning implementation does well to err on the side of caution and not match too eagerly in densely populated areas of the feature space.



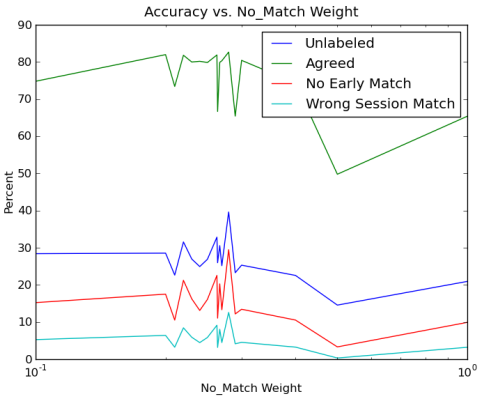
(a) Positive Weight



(b) Negative Weight



(c) Positive Weight



(d) Negative Weight

4 Future Direction

Although we have seen adequate results from training our model on only one hour of bumps, we can achieve a greater accuracy by increasing our training sample to more hours. It is also not clear that the analysis presented above to filter the unlabeled matches into actual false positives and those that are wrongly labeled is exhaustive and entirely accurate, as a user with no next matches during his session could've simply been fidgeting with his phone, or his bump partner's match never made it to the bump server. Therefore, a more refined labeling of matches would not only facilitate gauging the performance of the algorithm, but will also allow for more clear training data that can then result in a more accurate model. Finally, although the current accuracy does not suggest that the machine learning implementation can be an effective enough replacement for the system currently in place, there is enough evidence to suggest that it could provide a useful supplementary system for it. Combining both systems will allow us to match the users that are currently being matched, as well as to take advantage of the additional unlabeled matches, therefore improving the Bump users' experience without harming the false positive rate.