Predicting Running Times from Race History

Matt Millett and TJ Melanson

***Abstract*—In this project, we explored two different methods to predict runners' times by using their race histories. Baseline prediction used Peter Riegel's method of distance ratios, and the oracle was the Stanford Running Club coach, each which achieved $\approx$ 10% error on randomly chosen races from a preliminary dataset of 16 runners. On the real data set of 103 runners, the Riegel predictor had 4.1% overall error. We implemented locally weighted linear regression, which achieved down to 4% average error with leave-one-out-cross-validation on 100 different runners. However, the weighted regression failed to predict any value at all for approximately 10% of the data. We also built a Hidden Markov Model (HMM) modeling fitness states, which achieves 3.8% overall error, and also offers more insight into a runner's actual state of fitness.**

***Keywords*—Hidden Markov Model, Riegel, Athletics, Running, Times, Fitness, LOOCV, Locally Weighted Linear Regression, Stochastic Gradient Descent, EM algorithm, Baum-Welch, Expectation Maximization.**

I. Introduction

Many runners like to think about what race times they ”could have run” or could run on a particular date. Our models attempt to predict runners' race times and fitness based on their race histories. Right now, there are only a few reliable methods to predict race ability, but they are generalized for large groups of (often elite) runners. This is partly because none of them take advantage of a runner's entire race history to individualize predictions by only focusing on a single performance. For example, VO2 max testing, one method to detect fitness, involves strapping a respirator to one's face and running on a treadmill- obviously impractical. Coaches can also hazard a guess at what an athlete might run, but they can be prohibitively expensive for the average runner. In this project, we attempted to find a way to predict race times based on race history.

In our method, we use data from a runner's race history that could give them a much better idea of how they will perform. Our first method is locally weighted linear regression on individual runners to directly predict their race times for given distances. Our second method uses a Hidden Markov Model (HMM) to predict an athlete's fitness. His predicted fitness can then be used to predict his race time for a given distance with a particular probability. The input to our algorithm is a runner's race history and/or a series of other runner race histories. We use our locally weighted linear regression and our HMM to output predicted race times (future or hypothetical).

Matt Millett (SUNetID millett) is in CS 221 and CS 229. The work for the locally weighted linear regression was done particularly for CS 229. The work done on the HMM was mainly for CS221, but since it's so difficult to implement it was also partly done for CS 229. Discussion of error and comparison, however, is so interlinked it's hard to separate them, so really the discussion applies to both classes.

TJ Melanson (SUNetID melanson) is in CS 221.

A. Related Work

There is limited research on analytically predicting a runner's future race time using full race histories. Most models rely on a single performance to guess at one's racing abilities. The Vo2Max test is a test where the subject breathes through a respirator and runs at maximum effort on a treadmill. Although accurate, the equipment needed to perform the test is expensive and cumbersome. The famous coach Jack Daniels attempts to measure Vo2Max fitness from a single race, but it parameterizes fitness with a single value and we didn't use it as a baseline for this project [1]. Peter Riegel, an engineer, wrote in a 1981 paper how to predict a new race time based on a single recent result [2]. But no analytical model takes in a full history of past race times to predict future times.

B. Evaluating Results

We evaluate our results in two ways. The first way is with leave-one-out cross-validation (LOOCV) for the locally weighted linear regression, leaving out one race and using the rest of a runners' races to predict the race excluded from training data. For the HMM, there is no objective measure of fitness states, so we only evaluate the times the model outputs. If the regression is unable to make a prediction at all (due to matrix underflow errors), we count the number of failed predictions as well. This corresponds to the question, ”What could I have run for this distance at this time?”

For the HMM, we evaluate error by doing forward prediction: predicting one's next race based on all the previous race history. We cannot use LOOCV directly because it is too slow and impractical for this method. Instead, we leave 10 athletes out of our training set to use as our test set. The HMM prediction corresponds to the question, ”What will I run in my next race?” To normalize across distances, we actually predict the error in m/s instead of seconds. We calculate our relative error as:

$$\frac{\text{Predicted speed} - \text{True speed}}{\text{True speed}}$$

Unfortunately, our oracle was limited on time, so she was only able to predict one randomly selected race per athletes (for 16 athletes), but the rest of our models have full error analyses.

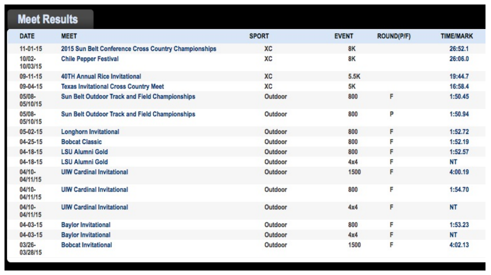
C. Gathering Data

Flotrack.com provides a large database of race histories for many athletes. However, it provides the data in human-readable rather than machine-readable form, so we had to utilize a web-scraping tool to extract the needed data. We gathered this data from random athletes on tfrrs.org [3]. Each file of data is one athlete's race history downloaded from the website. We have 2 datasets, one where we found athletes who raced in Cross-Country meets, and one where we found the athletes who raced in Track meets. People who raced in the Cross-Country meets tended to race the 5000m to the 10000m. Many Cross-Country runners also run longer events in track, so we ended up with lots of 3000 and 1600 data from them as well. We also found athletes from track meets who ran everything from the 60m to the 5000 on the track. To scrape this data, we used the BeautifulSoup library for web-scraping in Python [4].

We ended up with data for 103 athletes. Each athlete has approximately 20-40 races in their history, where a race consists of the race date, the race time, the race distance, and the name of the meet where the race was. Note that the races stored in this website only store history of *college* races. Part of the requisite for data selection was that the athlete had at least 10 races, because otherwise it would be difficult to have enough data to work with, particularly for our locally-weighted linear regression.

II. Preliminary Approaches: Baseline and Oracle

To get a good feel for the problem, we used a small data set for our initial predictions. We scraped race data for 16 top NCAA athletes from flotrack.org [5]. We used this data to find initial estimates for error. Both the baseline and oracle predicted for the same races. Note that while these errors may seem large, (10% error corresponds to more than a 1 minute difference in 5k time, or 12 seconds in the 800!), there are a couple reasons for this: firstly, courses can vary vastly in difficulty and we had no way to featurize this, and secondly our initial scraping failed to differentiate hurdle races and normal races, which have large time differences.



Meet Results

DATE	MEET	SPORT	EVENT	ROUND(P/F)	TIME/MARK
11-01-15	2015 Sun Belt Conference Cross Country Championships	XC	8K		26:52.1
10/02-10/03/15	Chile Pepper Festival	XC	8K		26:06.0
09-11-15	40TH Annual Rice Invitational	XC	5.5K		19:44.7
09-04-15	Texas Invitational Cross Country Meet	XC	5K		16:58.4
05/08-05/10/15	Sun Belt Outdoor Track and Field Championships	Outdoor	800	F	1:50.45
05/08-05/10/15	Sun Belt Outdoor Track and Field Championships	Outdoor	800	P	1:50.94
05-02-15	Longhorn Invitational	Outdoor	800	F	1:52.72
04-25-15	Bobcat Classic	Outdoor	800	F	1:52.19
04-18-15	LSU Alumni Gold	Outdoor	800	F	1:52.57
04-18-15	LSU Alumni Gold	Outdoor	4x4	F	NT
04/10-04/11/15	UIW Cardinal Invitational	Outdoor	1500	F	4:00.19
04/10-04/11/15	UIW Cardinal Invitational	Outdoor	800	F	1:54.70
04/10-04/11/15	UIW Cardinal Invitational	Outdoor	4x4	F	NT
04-03-15	Baylor Invitational	Outdoor	800	F	1:53.23
04-03-15	Baylor Invitational	Outdoor	4x4	F	NT
03/26-03/28/15	Bobcat Invitational	Outdoor	1500	F	4:02.13

Fig. 1: *This is what the data on the website looks like before scraping. It has the date, meet, event, and time listed. Ultimately, we only used the date, event, and time.*

A. Baseline

Peter Riegel, an engineer, developed a widely used time predictor based on a single race [2]. Take the race you want to run of distance d_{new} and the most recent time you have run t_{old} for a race of distance d_{old}. The time you run in the, t_{new} can be calculated as:

$$t_{new} = t_{old} * \left(\frac{d_{new}}{d_{old}}\right)^{1.06}$$

The baseline we use bases the new race off of the most recent race run, indirectly setting a to whatever would make the most recent distance be proportional at the most recent time. While this is a fast method, it fails to incorporate entire race history. On our preliminary data set, Riegel's method averaged 11.71% error in predicting times, where it took the athlete's most recent race and plugged it into the equation to predict the new time. (In this case, LOOCV and forward prediction are the same.)

B. Oracle

Pattisue Plumer is the coach of the Stanford Running Club and a former Olympian. For our oracle, we asked her to predict the times of 16 NCAA athletes of random races in their career. She was allowed to look at the entire rest of the race history. Unfortunately, this doesn't perfectly model a coach who personally knows the athlete and can make more precise predictions. She achieved 9.73% error on her predictions.

III. Approaches

Once we gathered our final dataset, the Riegel predictor predicted race times with an average of 4.1% error. This is a machine learning task, where given the inputs of race history we output an expected race time. There is some error that even the best predictor will never overcome, which has to do with human variability across races regardless of fitness. There is also variability of difficulty of cross-country courses and things like the weather on the day of the meet that throw a lot of variance into the data. In our dataset we were only able to scrape meet names, and this made it difficult to actually find that sort of data that introduces this variability. We used a locally weighted linear regression first to model this problem, and then a Hidden Markov Model, which we thought could better model these uncertainties of athlete's state of mind, the weather, and a whole host of other factors that affect race performance.

A. Locally Weighted Linear Regression

The first thing we did was to use locally weighted linear regression to predict race times. The features of the data we used were the race distance and the race date. Race date was featurized as number of days since the athlete's first recorded race. We used the Normal Equation to find the value of θ for the regression:

$$\theta = (X^T W X)^{-1} W Y$$

Where W is a weight vector corresponding to the weight of each date. [6]

We tried two different kernels to predict runners' times. The first was simple: if the distance of what we were predicting matched a test data point, we'd give that data point a weight of 1. Otherwise it'd be of weight 0. This gave 6.4% error on the track and cross country datasets, with failures to predict 339/2921 races, and an average root mean squared error of 0.28 m/s for cross-country data and 0.35 m/s for track data. For a few examples, our model averaged about 1 second off for the 800 and 25 seconds off the 8k. Our model fails to predict race times when the date or the distance is too far off from anything the model has seen before; the kernel sets the weights of everything to 0 or tiny numbers that make taking the inverse of the matrix for the Normal Equation impossible or so small it causes underflow. We left these out of our percentage error calculation and treated them as their own statistic, because any human looking at the outputs of the predictor would understand, for example, it's ludicrous to run 5 kilometers in 43.02 seconds (and yes, this was one of our outputs). (Without removing these failures, the error shoots up to 250% on average.) We also tested a Gaussian kernel to predict weights, which ultimately ended up giving us the best error. We chose a Gaussian kernel because it's commonly used for Locally Weighted Linear Regression [6], and our intuition tells us as runners that as distance and date drop off, the importance of each weight could be approximately normal. The Gaussian kernel sets the weight w of an observation $r = (r_{distance}, r_{date})^T$ while predicting at point $p = (p_{distance}, p_{date})^T$ as:

$$w = e^{(\frac{Q}{-2c^2})}$$

where

$$Q = (r-p) \begin{bmatrix} distanceWeight & 0 \\ 0 & dateWeight \end{bmatrix} \dot{} (r-p)$$

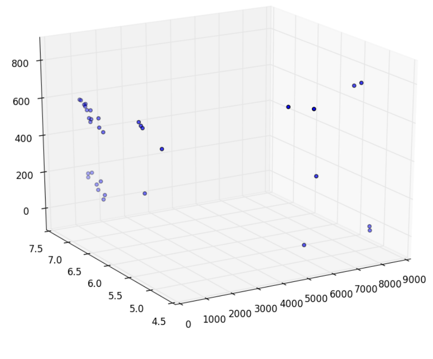


Fig. 2: *This is a graph of the data for one of our runners. On the x axis is distance (in m), on the y axis is speed (in $\frac{m}{s}$), and on the z axis is the number of days since the athlete's first race.*

and c as the bandwidth parameter.

Because of the lack of prior literature on this topic, we plugged in different values for the respective weights and bandwidth parameters to see what would work. In our modeling assumptions, we decided that the distance was a more important factor than the date, but to get the best results we would need to use both. Distance gave specificity to the race and date gave us an idea of how the runner improved over time.

One of the main issues with locally weighted linear regression is that it fails to predict races that are too far out from an event horizon unless the bandwidth value is large enough to turn it into simple linear regression. This makes it difficult to extrapolate to predict in the future or distances outside the data set originally provided while retaining the useful locality this technique provides.

Using these assumptions, we plugged in different weight and bandwidth values to see what gave the lowest error values. Setting $dateWeight = 0.01$ and $distanceWeight = 1$ and $bandwidth = 40$ gave 1.48% error on the track-seeded data set with 138/1377 failures, but actually didn't do that much better on the Cross-Country data set than simply looking at the distance, giving 6.11% error with $163/1544$ failures to predict times. Interestingly, this was about as good of a value as we could get, even after adjusting all the parameters.

The fact that locally weighted linear regression performs better on track data shouldn't be surprising. On the track, there is much less uncertainty about if the course is measured properly, or if there will be any hills (because there won't be). Most tracks are virtually identical. It is intriguing that introducing a weight for the date drives down the error so much, but that suggests that overall in our dataset, people tend to vary more in their race times from week to week, and incorporating the date allows us to better capture this variability.

Although we can somewhat model this variability with locally weighted linear regression, we decided to use a different model that allows us to model this uncertainty more directly and has more explanatory power: the Hidden Markov Model.

B. *Hidden Markov Model*

The second approach we took was to build a Hidden Markov Model as shown below.

Runners have hidden states of fitness, and from those states they emit race times with certain probabilities. There is no literature on how to model any of these probabilities, so we chose what distributions made the most sense to us. States of fitness are parameterized by 2 values, s and E, which roughly correspond to raw speed and endurance ability. Recall that Riegel's formula tells us that one's race speed drops off as a power function of distance. Roughly based off of Riegel's work, we decided to model this as:

$$t_{avg} = sd^E$$

where t_{avg} is the expected time that one will run given a certain value of s and E. Respectively, s and E are proxies for one's sprint and endurance abilities. Smaller values for both s and E correspond to faster times. We decided that race times would be normally distributed around t_{avg}, and we used standard deviation as a fixed hyperparameter value σ with respect to distance (see *Hyperparameters* for a more detailed discussion.) Thus, we can say:

$$t|(d, s, E) \sim \mathcal{N}(sd^E, \sigma d)$$

where d is the race distance σ is the standard deviation in $\frac{\mathrm{s}}{\mathrm{m}}$, s is the speed value, and E is the endurance value. We also initialized all our transition probabilities between fitness states to be uniform. While this is certainly not true in practice, it worked well enough for initialization. We discretized our continuous state space by initializing our s and E values for each state as linearly spaced values. We trained our HMM with a version of Expectation Maximization known as the Baum-Welch algorithm. The expectation step is a forward-backward algorithm used to find the probability of the sequence landing in any of the particular states. The forward part determines the probability that a given sequence will land on a given state, and the backward determines the probability that the remaining values will occur after that state. The distribution of time given a distance and fitness (parameterized by s and E) is normally distributed about the mean of sd^E, as described above.

The maximization step must find the values of θ that maximize the probability of a given sequence. Because s and E are dependent on each other (with respect to distance and time), determining the maximum likelihood that a given s and E would give a sequence is impossible, as multiple values of s and E can result in the same datapoint. Because of this, rather than using a deterministic algorithm to find the most likely pairs of s and E, we used Stochastic Gradient Descent (SGD) to adjust the parameters to the best value.

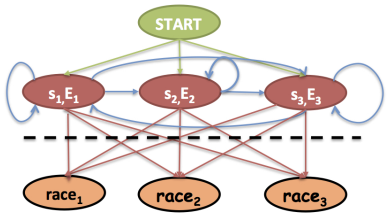


Fig. 3: *Our HMM (shown if it had 3 states). Hidden states are parameterized by s and E values and emissions are races, which consist of a time and distance. Transition probabilities were calculated using the Baum-Welch algorithm and emission probabilities were assumed to be normally distributed about the mean value of sd^E, where d is the input race distance. We did Stochastic Gradient Descent to maximize the expected probability of the s and E values for a training input sequence.*

We used the python library "yahmm" ("Yet Another Hidden Markov Model Library") [7] to build our HMM. The library allowed us to run the Baum-Welch EM algorithm for HMM'S to maximize the probability of particular parameters for us by training on the date-ordered race sequences from 93/103 of our athletes. On the remaining 10 athletes, we ran a forward inference for their final races, using their entire race histories up to that point. We then found the most likely state of fitness for the runner to be in and then calculated sd^E as our estimated race time. LOOCV (training on 102 athletes and test the 103^{rd}'s final race time based on his history) would be infeasible, because creating and training the HMM each time would take prohibitively long. Instead, 10 runners from the original dataset became the test set, while the other 93 were used to train the data. Note that the error function here is slightly different than that used for the locally weighted linear regression (LWLR): the LWLR interpolates over all of a single athlete's races, but the HMM extrapolates from an athlete's race history up to a point to predict his/her future time. (For the locally weighted linear regression, it doesn't make much sense to try to do future prediction because the locally weighted regression will tend to do much better on *interpolating* data.)

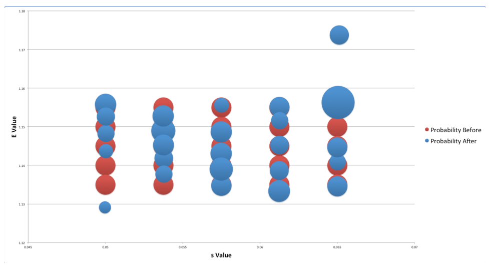
C. Hyperparameters

We set σ, the standard deviation of predicted time for a particular distance, equal to 0.005 $\frac{\mathrm{s}}{\mathrm{m}}$. We chose this value because on the surface, it makes sense: On a bad day, for example, one could run up to $(0.005\frac{\mathrm{s}}{\mathrm{m}} * 800m) = 4$ seconds slower in the 800, or 25 seconds slower in the 5k. Because of runtime limitations, we also only had 25 different states of fitness to choose from.

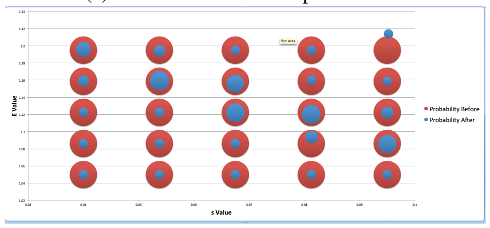
Additionally, there are hyperparameters for the EM algorithm: the η of the SGD, the edge inertia, and transition pseudocount. Edge inertia is the inverses of step size η for the model training. After some manual tuning, we found an η of 0.05 to be the optimal value: any smaller value didn't change the s or E values, while a much larger caused the gradient descent to diverge. When testing data, we found that the number of edges between fitness states far exceeded the number of transitions in the sequences, so we had to add a transition "pseudocount" to smooth out the transition probabilities (similar to Laplace smoothing). As pseudocount smoothing did not always provide stable answers, we also set the edge inertia to be 0.9 so only small changes in transition probabilities occurred.

D. Initialization

Neither the Baum-Welch nor the SGD maximization guarantee a global minimum. Because s and E depend on each other, both algorithms could converge to a nonoptimal solution. An extreme example is a value of E=1.0 and s=0.15 fit the data point of 2:00 for an



(a) Narrow initialization parameters



(b) Wide initialization parameters

Fig. 4: *Training the HMM with different initialization states with values of s and E. The size of each bubble indicates prior probability. Red indicates the initialization and blue indicates the final parameter values.*

800 (a reasonable value), it fails to scale for longer distances (it assumes that the runner never tires, which is highly unlikely). This means the HMM is sensitive to how it's initialized, so we had to feed it sensible original parameters. E probably varies from person to person, but Riegel's formula told us $E \approx 1.06$. However, we played with values of s and E to create pairs of 800m and 5k times that made sense to us based on our own running experience (e.g. a 1:57 800m corresponding to about 16:00 or 16:30 for a 5k, depending on one's endurance). We played with the initialization states to see how we the error changed. As expected, training on a narrower range of states brought prediction error from 6% to 4%, whereas training on a narrower range brought overall error from 3.5% to 3.3%: a much smaller improvement.

IV. Discussion

One advantage of HMM's is that they tend to be more descriptive about the data they analyze than regression. By parameterizing these states as states of fitness and assigning each race to a single fitness state, we are able to actually find out some numbers that give one an idea about his/her sprint ability and endurance, accounting for one's entire race history up to that particular point of inference. HMM's may also be able to extrapolate better with less data. As a generative, not discriminative model, the HMM builds a probability model for extrapolation, which we hoped would work well with fewer races per athlete. And this turned out to be true: for our models, on average the HMM performed slightly better relative to baseline.

A. Accuracy

The most important output for each of these models is the raw accuracy. But it's important to note the error values we calculate for the HMM and for the locally weighted linear regression have a subtle difference: the HMM error is for **extrapolation** to future races, whereas the locally weighted linear regression error is for **interpolation**. Despite these differences, we can compare each technique's error to our baseline error: since Riegel's baseline predictor only takes one race as input instead of many, it technically performs both predictions.

The errors are summed up in the table below. The Locally Weighted Linear Regression (LWLR) error in the table is the interpolation error on LOOCV, listed along with the number of failures to predict it had. The HMM error in the table is the error on predicting an athlete's final race given his/her total history up to that point.

	XC Error	Track Error	Overall
Baseline	3.6%	5.0%	4.1%
LWLR	6.1% ($\frac{163}{1544}$ fails)	1.5% ($\frac{138}{1377}$ fails)	3.9%
HMM	3.5%	5.9%	3.8%

Again LWLR to HMM error measure slightly different things, so we have to be careful when comparing their error values. Respectively, they predict the times an athlete "could have run" vs. times an athlete "will run". However, we can compare both the Riegel prediction error to both of them, since the Riegel prediction only looks at a single race.

V. Conclusion

Our predicted models actually didn't perform as well as we expected them to compared to our baseline. The main exception to that was locally weighted linear regression on track data. These models seemed promising, especially the HMM, because the HMM is more of a descriptive model of whole states of fitness. With more rich and complete race histories, our models might be able to perform even better, but because each runner only had about 20-40 races, it was much harder to adapt our models to the data, particularly with the locally weighted regression, which failed to produce a reasonable prediction on 10% of attempts. That said, we also were able to optimize to slightly beat the baseline, which is exciting.

REFERENCES

[1] Daniels, Jack. ”VDOT Running Calculator.” Jack Daniels’ VDOT Running Calculator. The Run S.M.A.R.T. Project, n.d. Web. 11 Dec. 2015. https://runsmartproject.com/calculator/.

[2] Riegel, Peter S. ”Athletic Records and Human Endurance: A time-vs.-distance equation describing world-record performances may be used to compare the relative endurance capabilities of various groups of people.” American Scientist (1981): 285-290.

[3] TFRRS — Track Field Results Reporting System. Web. 11 Dec. 2015. https://tfrrs.org/.

[4] ”Beautiful Soup.” Last updated 28 Sept. 2015. Python library. 11 Dec. 2015. http://www.crummy.com/software/BeautifulSoup/.

[5] ”Top Rankings, Men, XC, 2015, NCAA Division 1.” FloTrack. N.p., 20 Oct. 2015. Web. 11 Dec. 2015. http://www.flotrack.org/rankings/subjective/1302.

[6] Ng, Andrew. ”CS229 Problem Set 1: Semi-Supervised Learning.” 14 Oct. 2015. Web. 11 Dec. 2015. http://cs229.stanford.edu/materials/ps1.pdf.

[7] ”Yahmm: Yet Another Hidden Markov Model.” Jmschrei. Last updated 2 Dec. 2015. Python library. 11 Dec. 2015. https://github.com/jmschrei/yahmm.

ACKNOWLEDGMENT

The authors would like to thank Pattisue Plumer for all her expertise as an oracle. We’d also like to thank the CS221 and CS229 TA’s who helped us out on this project, especially Bryan McCann, who promptly responded to all our emails.