PREDICTING GREEN LINE TRAVEL TIMES

FINAL REPORT FOR CS 136 SPR AT TUFTS IN FALL '25

Nicole Falicov **Angela Shen**

Sharing permission statement: Instructors can share this report to future students of similar classes

1 INTRODUCTION

Travel times on the Green Line are notoriously inconsistent, to the extent that there is an unofficial six mile course known as the Charlie Card Challenge, in which runners seek to out-race the T as it travels from Boston College to Park Street. However, the speed at which runners must run can vary from a 6:40 to 10:00 minute per mile pace. Inspired by this race, we wanted to see if probabilistic modeling could accurately predict the travel time between the two stations.

We will use univariate regression to train a model that given the date and time of departure, will produce a single estimate for the travel time from Boston College to Park Street via the Green Line during peak use hours (7AM - 7PM). We will be training on data recorded from January through May and evaluating the quality of predictions on trips during June.

Probabilistic modeling is suitable for this task because the Green Line is subject to stochastic variables such as car traffic, weather conditions, differing user demand, and skipping low demand stations when delays are especially high. Using a probabilistic approach will better account for this uncertainty and ideally produce more reliable predictions.

2 DATA AND EXPERIMENTAL PLAN

2.1 DATA DESCRIPTION

Our dataset comes from the MBTA Rapid Transit Travel Times (Massachusetts Bay Transit Authority) dataset from 2024, which is officially released by the MBTA. The dataset contains information about trips occurring on each of the main train lines. A limitation we observed within the dataset was that ride information was not recorded every day. However, each month contained a reasonable amount of observations spread across each month making the data still suitable for our task. No incomplete entries were found during pre-processing. After filtering the data to only include trips that fit our criteria, we had 17,668 instances of the trip from Boston College to Park Street via the Green Line occurring between 7AM to 7PM. We further limited our dataset to trips occurring in January through June, which reduced the number of instances to 7,671.

The dataset contains 15 features, but we use four relevant features for this project. One feature is used directly as provided from the dataset and the remaining three were created using the service date feature (a string in YYYY-MM-DD format). Our features are the following:

- **Number of days since January 1st, 2024:** positive integer, calculated from service date feature of original dataset.
- **Departure time:** positive integer indicating seconds since midnight, provided directly from original dataset.
- **Day of the week:** positive integers 0-6 representing Monday through Sunday, obtained from service date feature of original dataset.
- **Holiday status:** boolean represented as 0 or 1, obtained from service date feature of original dataset and holidays python library (Vacanza Team).

We normalized all raw features to have a mean of 0 and a standard deviation of 1 to speed up computation. After prediction, we undo the scaling to obtain travel time predictions.

Our label is the total travel time in seconds, which is always a positive value that represents the difference between the seconds from midnight of the start and end of the trip. Though time is continuous, the dataset only provides travel times rounded to full seconds. However, we do regression assuming time is continuous and predict float values.

We have observed that the train times generally follow a bell shaped curved that is slightly skewed right. Observing train times at different times of day (Figure 4), we observe that travel times generally increase around peak usage during morning and evening commute times.

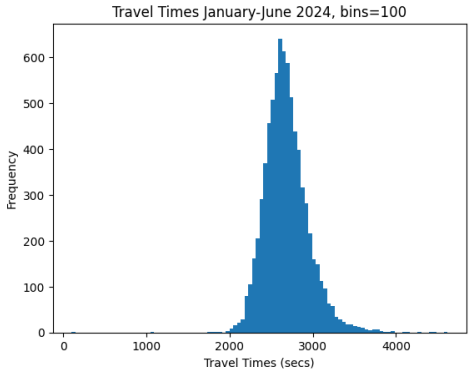

Figure 1: Histogram showcasing the distribution of travel times that occur during January-June 2024. Travel times in the dataset generally follow a bell curve, with slightly more data points on the right side than the left.

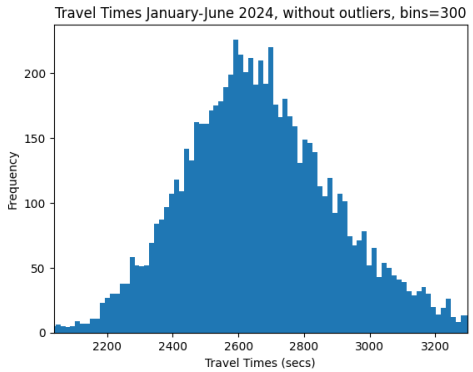

Figure 2: Focusing only on points within 1.5 $\times$ the IQR, this histogram with 500 bins similarly shows a bell curve with a slightly fatter and longer tail on the right side.

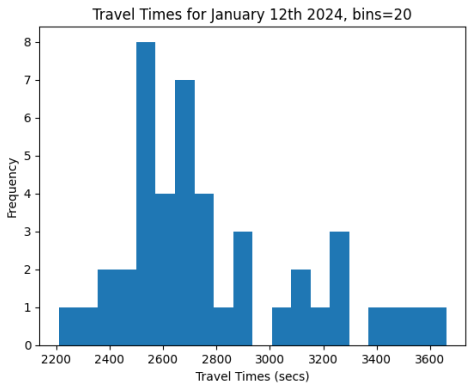

Figure 3: Distribution of total travel times on a single day (January 12, 2024). There is no discernible underlying pattern.

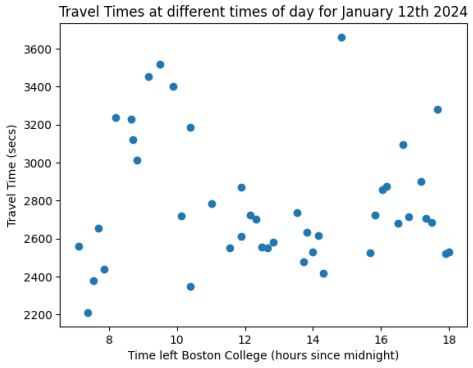

Figure 4: Total travel times at different times of the day for January 12, 2024. The travel times were larger during typical morning and evening commute times.

2.2 Performance Metric

We used average log likelihood as our primary performance metric. Likelihood is the probability of a label given the observations and learned model parameters. Taking the average log likelihood of all test data allows us to assess the accuracy of predictions on previously unseen data using the weights (baseline) and mixture parameters (upgrade) tuned on training data. Average log likelihood on the test set will therefore allow us to measure how well training on trips from January through May can help us predict trips in June.

2.3 Dataset Split

We consider all data points from January through May as our training data and all data points from June as our test data. This gives us 6,049 training points and 1,622 test points which results in an approximately 79/21 train/test split.

Split	Data points	Percentage
Training	6049	78.86
Test	1622	21.14

Table 1: Summary info about splits into train/test.

3 Baseline Method

Model. For our baseline method, we pursued linear regression. Our goal was to predict travel time given known data points $(x_1, \dots x_N)$ for departures from Boston College. Our model consists of the following random variables:

- $t_1, \dots t_N$: (observable) travel times at each date-time value of interest in dataset of size N
- w: (hidden) weight vector that determines the mean function for predicting t from x

The variance will be a fixed value, based on the likelihood precision hyperparameter β, and we conduct a grid search with K-fold cross-validation to determine the optimal value. We chose to fix the variance to reduce the number of parameters that we had to estimate.

Our raw features x_n are 4-dimensional. From our dataset, we know that each t_n random variable comes with a known, fixed date-time feature x_n.

To simplify our analysis, we assumed $x_{1:N}$ are independent and identically distributed. In real life, past trains can delay future trains, but to make our problem tractable, we assumed each journey had no impact on other journeys.

3.0.1 Prior Model

We place a Multivariate Normal prior over the weight vector $w \in \mathbb{R}^M$ since we expect our weights to be normally distributed.

$$p(w) = \mathcal{N}(w \mid 0, \alpha^{-1} I_M)$$

Here, the scalar $\alpha > 0$ is the prior precision hyperparameter.

3.0.2 Likelihood Model

We model the N observed outputs $t = \{t_n\}_{n=1}^N$ as follows:

$$p(t \mid x, w) = \prod_{n=1}^{N} \mathcal{N}(t_n \mid w^T \phi(x_n), \beta^{-1})$$

Here, the scalar $\beta > 0$ is the likelihood precision hyperparameter.

3.0.3 Feature Transform

We use polynomial features for the feature transform (subscript of x_n indicates the feature index):

$\phi^{(0)}(x_n) = [1]^T$

$\phi^{(1)}(x_n) = [x_{n_0},\ x_{n_1},\ x_{n_2},\ x_{n_3}]^T$

$\phi^{(2)}(x_n) = [x_{n_0},\ x_{n_1},\ x_{n_2},\ x_{n_3},\ x_{n_0}^2,\ x_{n_0}x_{n_1},\ x_{n_0}x_{n_2},\ x_{n_0}x_{n_3},\ x_{n_1}^2,\ x_{n_1}x_{n_2},\ x_{n_1}x_{n_3},\ x_{n_2}^2,\ x_{n_2}x_{n_3},\ x_{n_3}^2]^T$

Where we get to pick the order of the polynomial as a hyperparameter. Too high of an order may result in overfitting and too low of an order may result in underfitting, thus we will find the optimal order via grid search.

Estimation. To form a prediction for new t_* given an unseen data point x_*, we use MAP point estimation to get an estimate for the mean vector w, which can be solved as follows:

$$\begin{aligned} w_{MAP} &= \text{argmax}_w \log p(w \mid \{x_n, t_n\}_{n=1}^N, \alpha, \beta) \\ &= (\Phi^T \Phi + \frac{\alpha}{\beta} I)^{-1} \Phi^T t \end{aligned}$$

Thus, given the MAP estimate of w, we can compute the likelihood of a new data point t_* given new input point x_* via the following:

$$p(t_* \mid x_*, w = w_{MAP}, \beta) = \mathcal{N}(t_* \mid w_{MAP}^T \phi(x_*), \beta^{-1})$$

We select the optimal value of w via grid search with K-fold cross-validation by selecting the hyperparameters for w that yield the highest validation-set log likelihood.

Implementation. From prior course assignments (Hughes), we had access to code for fitting data and making predictions using the MAP, and for generating visualizations. For the polynomial feature transformations, we used the Polynomial Features package from sklearn.preprocessing (scikit (c)). For normalization of our features, we used StandardScaler from scikit-learn (scikit (d)). Any remaining code was written from scratch.

Hyperparameter Plan. We use a grid search with K-fold cross-validation to find the best α (prior precision hyperparameter), β (likelihood precision hyperparameter), polynomial order, step size, and number of iterations. Since MAP has a closed form, step size and number of iterations will only apply to our upgrade method. Log likelihood on the validation set was used as the evaluation metric. We search over the following values:

- α: 10 values evenly spaced from 10^{-8} to 10^1.
- β: 27 values evenly spaced between 10^{-1} to 10^1
- Polynomial order: 1-9

4 Upgrade Method

Motivation and Hypothesis. We expect that train travel times can be broadly categorized into a variety of groups such as: early, on-time, delayed, extremely delayed, and express. Each group will likely will follow a different distribution. Delayed train travel times likely follow a different distribution than on-time trains because delays propagate as a train moves through multiple stations, possibly resulting in the right skew we observe in our dataset. As a result of these multiple categories, we do not anticipate that travel times will follow a standard Gaussian distribution and thus be adequately modeled by our baseline. Because Gaussians are symmetrical, they won't be able to properly account for the skew and will result in an increased variance that doesn't accurately reflect the underlying patterns in the data. It is also possible that early, extremely delayed, and express trains follow different distributions that we cannot observe from our exploratory data analysis.

To better model this behavior, we use a Gaussian mixture model (GMM) for our likelihood and experiment with between 2-5 clusters. This method is more robust for modeling data that is composed of multiple distributions, and will perform better because we anticipate each cluster will be Gaussian. We hypothesize that compared to the univariate Gaussian, regression using a GMM should improve log likelihood on our test set, especially when accurately modeling delayed travel times, because it is more robust when modeling data with multiple underlying distributions.

Model. Our upgrade model is a GMM that models the joint distribution of inputs (x_n) and outputs (t_n). We then use Gaussian mixture regression to predict travel times. This model consists of the following random variables:

- x_n: (observable) feature vector for the nth trip
- t_n: (observable) travel time for the nth trip
- z_n: (hidden) mixture component assignment

Our raw features x_n are 10-dimensional because day of the week was one-hot encoded for the upgrade. Each t_n comes with a known, fixed date-time feature x_n. Again we assume that $x_{1:N}$ are independent and identically distributed for the same reasons previously stated.

The joint distribution $p(x_n, t_n)$ is defined as a mixture of K Gaussian distributions:

$$p(x_n, t_n) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\begin{bmatrix} x_n \\ t_n \end{bmatrix} \,\middle|\, \mu_k, \Sigma_k \right)$$

Where parameters π_k, μ_k, and Σ_k are defined as:

- π_k: mixture weights that sum to 1
- μ_k: component means
- Σ_k: full covariance matrices

Estimation. The objective for maximum likelihood estimation of the GMM parameters is defined as follows:

$$\pi_k^*, \mu_k^*, \Sigma_k^* = \text{argmax}_{\{\pi_k, \mu_k, \Sigma_k\}} \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\begin{bmatrix} x_n \\ t_n \end{bmatrix} \,\middle|\, \mu_k, \Sigma_k \right)$$

Expectation Maximization (EM) is the algorithm used to find the best $\pi_k^*, \mu_k^*, \Sigma_k^*$. EM alternates between updating posterior responsibilities $p(z_n = k \mid x_n, t_n)$ during the E-step and updating π_k, μ_k, and Σ_k during the M-step to estimate the best π_k^*, μ_k^* and Σ_k^*.

Implementation. Scaling of data was done similarly to the baseline with the exception of one hot encoding the day of the week using OneHotEncoder (scikit (b)). Using scikit-learn's GaussianMixture, we fit a GMM and learned mixture weights π_k, component means μ_k, and covariances Σ_k via EM implemented by scikit-learn (scikit (a)). Polynomial features were applied to continuous inputs only.

Because scikit-learn's GaussianMixture's default score is joint log-likelihood $p(x_n, t_n)$ and does not have the capability to do regression, we implemented functionality to calculate the log likelihood of labels given data and parameters and the predicted mean and variance from scratch using the following methods.

First we partition each component mean and covariance:

$$\mu_k = \begin{bmatrix} \mu_{x,k} \\ \mu_{t,k} \end{bmatrix}, \quad \Sigma_k = \begin{bmatrix} \Sigma_{xx,k} & \Sigma_{xt,k} \\ \Sigma_{tx,k} & \Sigma_{tt,k} \end{bmatrix}$$

Then we compute the predicted mean and variance using equations 2.81 and 2.82 from Bishop (2006):

$$\mu_{t|x,k} = \mu_{t,k} + \Sigma_{tx,k}\Sigma_{xx,k}^{-1}(x - \mu_{x,k}), \quad \Sigma_{t|x,k} = \Sigma_{tt,k} - \Sigma_{tx,k}\Sigma_{xx,k}^{-1}\Sigma_{xt,k}$$

The posterior responsibility for each component is calculated as:

$$r_k(x) \propto \pi_k \mathcal{N}(x \mid \mu_{x,k}, \Sigma_{xx,k})$$

Then the log likelihood we use as our evaluation metric is computed as the mixture of the Gaussians weighted by $r_k(x)$:

$$\frac{1}{N} \log p(t_n \mid x_n, \pi_k, \mu_k, \Sigma_k) = \frac{1}{N} \sum_{k=1}^{K} \log r_k(x) \mathcal{N}(t \mid \mu_{t|x,k}, \Sigma_{t|x,k})$$

Hyperparameter Plan. We tuned hyperparameters using 3-fold cross validation and selected values that maximize the average log-likelihood of seeing the labels given the data and parameters on validation splits. We searched over the following values:

- Number of mixture components K: 2, 3, 4, 5
- Covariance regularization: 0.01, 0.001, 0.0001
- Polynomial feature order: 1, 2, 3

The number of max iterations was fixed at 100, tolerance was fixed at 10^{-4}, and covariance type was fixed at $full$.

5 Results and Analysis

Both models were trained on scaled data from January through May of 2024 and evaluated on the June 2024 observations. 3-fold cross validation was used to select hyperparameters for both the baseline and upgrade. For the baseline we were unable to evaluate methods above order 8 or 9, due to computational constraints. For the GMM any order over 1 caused dramatic overfitting to the training data. We found that our GMM performed better than our baseline by achieving higher validation and test log likelihood scores.

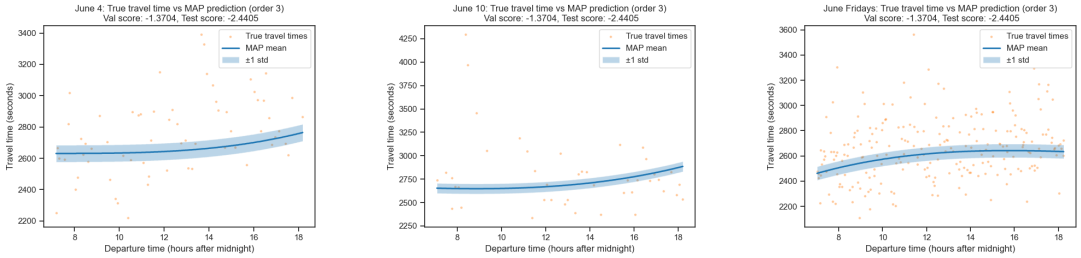

Figure 5: **Baseline Results.** Subsets of predicted travel time versus true travel time for baseline model. The best performance came from an order 3 model, with an average log probability score on the test set of -2.4405. MAP predictions with a 1 standard deviation boundary at each time are indicated by the solid blue line and shaded blue zone.

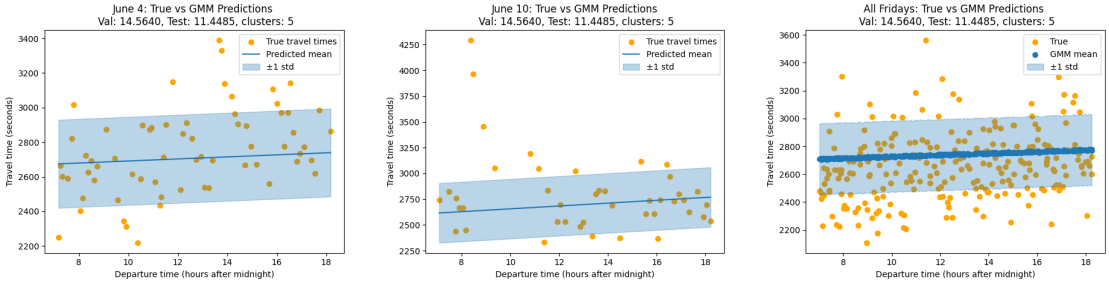

Figure 6: **Upgrade Results.** Subsets of predicted travel time versus true travel times for the upgrade model. The best performance came from a 5-component model with an average log probability on the test set of 11.4485.

Method	Validation log probability	Test log probability
Baseline	-1.3704	-2.4405
GMM (2 components)	0.7165	-1.7566
GMM (5 components)	14.5640	11.4485

Figure 7: **Comparison of methods.** Validation and test set average log likelihood score for each method

In Figure 7, we can observe that the upgrade method clearly outperforms the baseline method on average log likelihood, even with 2 clusters. This supports our hypothesis of our baseline model being unable to properly model data consisting of multiple separate distributions, and of the upgrade method performing better. Increasing the number of clusters also improves our results but likely caused some overfitting to the training data. On the whole, we can see a clear improvement in the quality of predictions between our baseline and upgrade method. However, our figures show that the GMM was not able to make perfect predictions. Considering the variability in travel times on the Green Line, this would make sense. Our model can't predict factors that also affect trip length like construction, station closures, mechanical failures, or car traffic.

6 Discussion, Reflection, and Outlook

The main strength of our study came from our familiarity with the data source. We are frequent users of the T due to its convenience and proximity to Tufts. Our knowledge of the T helped inform the models we selected for our study, and provided context for our results. However, computational costs limited the scope of our study. We would have liked to investigate more complex models, worked with more data, and included more features, but we quickly realized that complex models were not feasible with our current computing setup.

Through our project, we gained a deeper understanding of using probabilistic methods in the real world, and the difficulties in implementing theoretical models to produce a concrete result. We also learned how important it is to be familiar with the data that you use, so you can have context for your results.

Given more time, we would have liked to see whether the difficulty in predicting Green Line travel times extends to the other transit lines, which are not affected by street traffic and can travel at higher speeds below ground.

REFERENCES

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006. ISBN 978-0-387-31073-2. https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

Michael Hughes. Statistical pattern recognition. https://www.cs.tufts.edu/cs/136/2025f/.

Massachusetts Bay Transit Authority. Mbta blue book open data portal. https://mbta-massdot.opendata.arcgis.com/datasets/0b4dc16b8b984836962229865d5b573b/about.

scikit, a. https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture.

scikit, b. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

scikit, c. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html.

scikit, d. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

Vacanza Team. Holidays 0.87. https://pypi.org/project/holidays/.

A CODE APPENDIX

```
def conditional_log_likelihood(gmm, X, y):
    """
    Compute per sample conditional log-likelihoods log p(y_i|x_i)
    Parameters:
    gmm : sklearn.mixture.GaussianMixture
        A fitted GMM modeling the joint distribution p(x, y)
    X : array with shape (N, D)
        Input features
    y : array with shape (N,)
        Target values
    Returns:
    log_lik : array with shape (N,)
        Conditional log-likelihood for each data point
    """
    # Number of data points
    n_samples = X.shape[0]
    # Number of mixture components in the GMM
    n_components = gmm.weights_.shape[0]
    log_lik = np.zeros(n_samples)

    # Loop through each data sample
    for i in range(n_samples):
        x_i = X[i] # input
        y_i = y[i] # target
        cond_probs = np.zeros(n_components)
         # Loop through each component
        for k in range(n_components):
            # Get mu from learned params
            mu = gmm.means_[k]
            # Get sigma from learned params
            Sigma = gmm.covariances_[k]
            # Split mean of input and mean of target
            mu_x, mu_y = mu[:-1], mu[-1]
            # Split covariances
            Sigma_xx, Sigma_xy = Sigma[:-1, :-1], Sigma[:-1, -1]
            Sigma_yx, Sigma_yy = Sigma[-1, :-1], Sigma[-1, -1]

            # Compute p(x_i|k)
            px = multivariate_normal.pdf(x_i, mean=mu_x, cov=Sigma_xx)
            # Compute p(k) * p(x_i|k)
            gamma_k = gmm.weights_[k] * px
            # Compute inverse of covariance of x
            Sigma_xx_inv = np.linalg.inv(Sigma_xx)
            # Compute E[y|x, k] (bishop eq 2.81)
            mu_cond = mu_y + Sigma_yx @ Sigma_xx_inv @ (x_i - mu_x)
            # Compute Var[y|x, k] (bishop eq 2.82)
            var_cond = Sigma_yy - Sigma_yx @ Sigma_xx_inv @ Sigma_xy
            # Compute p(k) * p(x_i|k) * p(y_i|x_i, k)
            cond_probs[k] = gamma_k * norm.pdf(y_i,
                                               loc=mu_cond, scale=np.sqrt(var_cond))

        # Sum to compute p(y_i|x_i), add small epsilon for numerical stability
        log_lik[i] = np.log(cond_probs.sum() + 1e-12)
    return log_lik
```