Taxi Demand Prediction for Regions with Sparse Demand

Chenhao Wang **Philip Cho** **Logan Born**
Zhuo Cen **Yijing Xue**
School of Computing Science
Simon Fraser University
{cwa217,pycho,loborn,cenzhuoc,yijingx}@sfu.ca

Abstract

This work considers the problem of predicting demand for taxis and ride hailing services in areas with relatively low demand. We show that a state-of-the-art demand prediction model performs poorly when demand is low and spread over a wide geographic area. We identify the factors which contribute to this poor performance, and investigate alternative models which do not suffer from the same shortcomings. We show that training a hierarchical prediction model with a new loss function can significantly improve the performance in low-demand regions.

1 Introduction

Taxi and ride hailing services provide an important service in urban settings, where they offer a more direct transportation option than is available on public transit. The ability to accurately predict when and where users will require taxi service allows these companies to provide faster service by dispatching drivers to high-demand areas ahead of time. Users thus receive prompt service, and companies which are better at predicting demand may be able to outperform their competitors by arriving at high demand areas more quickly.

[6] develops a model for taxi demand prediction which provides state-of-the-art performance on a dataset containing taxi orders from Guangzhou, China. The datapoints are densely packed in a region of 14km $\times$ 14km, and the overall demand is very high, averaging 300,000 orders per day. The authors show that their model outperforms other architectures on this particular dataset, including former state of the art models [1, 7] and a variety of non-neural baselines. They do not compare the model's performance on other datasets. We propose to study this model's performance on a new dataset where demand is typically low and sparsely distributed; we hypothesize that the model will not be able to make accurate predictions in this setting, due to the fact that demand will be zero in most of the training examples and consistently predicting zero will be a valid strategy for the model to minimize the loss.

We consider two techniques to prevent the model from overfitting to the training examples with zero demand. A hierarchical model first predicts whether demand will be zero or nonzero; the regression is only trained on examples which are predicted to have nonzero demand, in order to make the training data more balanced. We also train with an alternative loss function which is designed to push the model towards making more balanced predictions. We find that both approaches improve the accuracy of the model, but best results are obtained by using both at the same time.

2 Methodology

2.1 Model Architecture

As we are interested in evaluating an existing model on a novel dataset, we use the same architecture described by [6]. We summarize the structure of the network below.

At each timestep t, we construct a heatmap $\mathbf{Y}_t$ where each cell corresponds to some geographic location, and the value in a cell records the number of taxi reservations made at the corresponding location at time t. To predict the demand in location i at the next timestep $t+1$, we use information from i and its neighbors over the previous h timesteps, where h is a fixed sequence length.

We first extract an $S \times S$ square centered on i from the heatmaps $\mathbf{Y}_{t-h+1}$, $\mathbf{Y}_{t-h+2}$, ... $\mathbf{Y}_t$. Let $\mathbf{Y}_t^i \in \mathbb{R}^{S \times S \times 1}$ refer to the square centered on location i extracted from the heatmap for time t. We pass each $\mathbf{Y}_t^i$ through a CNN with 3 layers. Let $\mathbf{Y}_t^{i,0} = \mathbf{Y}_t^i$ denote the original input to the CNN; then the output from layer k of the network is given by

$$\mathbf{Y}_t^{i,k} = f(\mathbf{Y}_t^{i,k-1} * \mathbf{W}_t^k + \mathbf{b}_t^k)$$

where $*$ denotes convolution and we take $f(\cdot)$ to denote a ReLU. For each of the h preceding timesteps t and each layer of the CNN k there is a unique weight matrix $\mathbf{W}_t^k$ and bias $\mathbf{b}_t^k$. These are shared across all locations i to reduce model size.

The top layer of the CNN is flattened to create a row vector and passed through a dense layer to reduce its dimensionality. Let $\mathbf{e}_t^i$ denote the resulting vector which represents the CNN features from location i at time t.

We next construct a series of feature vectors $\hat{\mathbf{s}}_t^i$ representing other information about location i at time t. These features include weather information, flags specifying whether time t represents a holiday, and location features specifying the geographic coordinates of location i. The vectors $\hat{\mathbf{s}}_t^i$ are concatenated with the flattened outputs from the CNN to create a final series of vectors $\mathbf{g}_t^i = \hat{\mathbf{s}}_t^i \oplus e_t^i$.

The vectors $\mathbf{g}_{t-h+1}^i$, $\mathbf{g}_{t-h+2}^i$, ..., $\mathbf{g}_t^i$ are passed as input to an LSTM to produce an overall representation $\mathbf{h}_t^i$ of location i and its neighborhood in the hours leading up to time t.

Prior to training, we learn a low-dimensional representation $\mathbf{m}^i$ of each location i in the training data. This representation is obtained by computing the average weekly demand pattern for each location and using dynamic time warping [3] to compute the similarity between the demand patterns in each region. We construct a graph where the edge weight between two nodes i and j equals the similarity between the demand patterns in locations i and j. We use LINE [4] to reduce the dimensionality of this graph, producing a small embedding $\mathbf{m}^i$ for each location i. The intuition is that distant areas may have similar demand patterns: for example, all residential areas will likely be busy at similar times. This locational embedding allows the model to capture these similarities even when the map provided to the CNN does not contain any regions similar to the area being predicted.

This embedding is passed through a dense layer to produce a hidden representation $\hat{\mathbf{m}}^i$ and concatenated with the final hidden state of the LSTM to produce an overall representation $\mathbf{q}_t^i = \mathbf{h}_t^i \oplus \hat{\mathbf{m}}^i$. At this point $\mathbf{q}_t^i$ contains information about i, its neighborhood, and external features such as the weather over the past h time steps. A fully connected layer with sigmoid activation converts $\mathbf{q}_t^i$ into a predicted demand value $\hat{y}_{t+1}^i$ between 0 and 1, which can be denormalized to give an overall prediction of the demand in location i at time $t+1$. The overall architecture is summarized in Figure 1.

Since we have much less data than the original authors of [6], we reduce the dimension of the hidden layers to prevent overfitting to the training data. Our CNN outputs have 8 dimensions; the LSTM outputs have 128 dimensions; and the locational embedding has 32 dimensions.

2.2 Data

Our training data comprises one year of reservations from Kabu, a company located in Richmond, BC that offers ride hailing service to its clients. There are approximately 500,000 total orders from the year 2018, giving a daily average of about 1370 orders. These orders span a region of more than

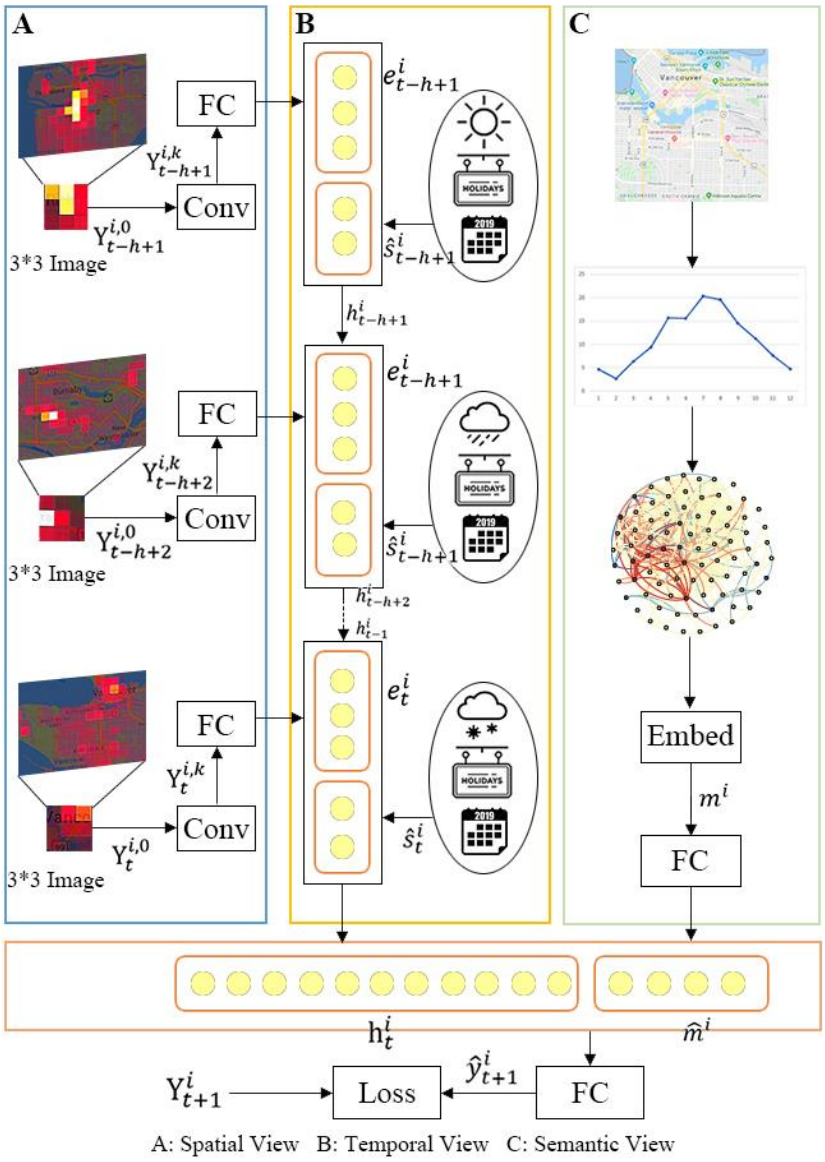

Figure 1: Model architecture re-implemented from [6].

960 km^2, which is more than four times larger than the area considered by [6]. Our test set consists of 187,000 orders spanning the first three months of 2019.

We prepared our data following the process described in [6] by grouping orders into discrete chunks of time and discrete locations on a grid. Due to time constraints it was not feasible to train a model with exactly the same parameters as the original paper. We used hour-long chunks rather than 30 minute chunks to reduce the number of training examples; this allowed us to train models more quickly and to compare a greater variety of model architectures against one another. The original paper used cells of size 800m$\times$800m; our training data span a region which cannot be evenly divided into cells of this size, so we instead used cells of size 700m$\times$700m. We used a 3$\times$3 neighborhood to extract heatmaps for the CNN, and we used a sequence length of 6 for both the CNN and LSTM inputs. This choice of kernel size and sequence length were also motivated by the need to train our model in a reasonable length of time; the original paper used kernels of size $\geq$5$\times$5 and sequences of length 8.

As in the original paper, we filtered out locations with extremely low demand (in our case, this meant < 25 orders per year). Despite this filtering, the training data are still extremely skewed, with only about 6.5% of the output labels being nonzero.

3 Experiments

3.1 Experimental Design

We consider two ideas for improving model performance when overall demand is low: (i) alternative loss functions, and (ii) hierarchical model architectures.

3.1.1 Loss Functions

The original model due to [6] is optimized with respect to mean absolute percentage error (MAPE),

$$\text{MAPE} = \frac{100\%}{N} \sum_{n=1}^{N} \left| \frac{t_n - y_n}{t_n} \right|$$

where y_n denotes the predicted demand for the nth training example and t_n denotes the true demand. Note that MAPE is asymmetric: underestimates can incur an error of at most 100%, but overestimates can incur arbitrarily large errors. Past work [5] has observed that regressions optimized using MAPE tend to make predictions which are too low, since the penalty for underestimating is lower on average than the penalty for overestimating. This may be acceptable on data with high overall demand, but in a context where demand is often zero we predict that this will further bias models in favor of predicting zero demand.

For this reason we consider an alternative loss function which we design to discourage models which underestimate the data:

$$\text{MSLE} = \frac{1}{N} \sum_{n=1}^{N} (\log(1 + y_n) - \log(1 + t_n))^2$$

This loss function is based on minimizing the squared error of the accuracy ratio y_n/t_n, as suggested by [5]. We take the logarithm to ensure that the error grows faster for small values than for large ones, and we add 1 to each term to prevent issues from zeros in the data.[1] This error penalizes underestimates more than overestimates, meaning that models trained to optimize this function should be biased against predicting zero.

3.1.2 Hierarchical Models

To make the regression model's training data less skewed, we consider a two-stage hierarchical model. In the first stage, a classifier predicts whether the demand at the current location and timestep is going to be zero or nonzero; the regression is only trained on datapoints where the classifier predicts nonzero demand. This helps to unbias the training data, and only relies on features that are available to the model at test time.

We consider two approaches to training a hierarchical model. As a baseline, we use a simple heuristic classifier. If there is an order in location i at time t, then on average there are slightly more than 3 orders at the same location over the preceding 6 timesteps. Based on this value, we construct a heuristic classifier which predicts nonzero demand if there are more than 3 orders in the last 6 timesteps, and zero demand otherwise.

We also construct an end-to-end neural model by adding a second dimension to the network's output layer. The overall loss function then becomes a sum of two terms, $\lambda \mathcal{L}_R + (1 - \lambda)\mathcal{L}_C$, where $\mathcal{L}_R$ is the regression loss and $\mathcal{L}_C$ is the classifier loss. Regression loss is calculated as before using either MAPE or MSLE, except that we only compute the regression loss on training examples which have nonzero demand. The classifier component uses weighted binary cross-entropy loss (weighted to

[1] Following our "invention" of this loss function, we discovered that it already exists in the literature, where it is referred to as mean squared logarithmic error, or MSLE [2]

account for the imbalance between classes). In the case where we use MSLE loss for the regression, the overall loss can be expressed as

$$\mathcal{L} = \lambda \mathcal{L}_R + (1-\lambda)\mathcal{L}_C = \frac{\lambda}{|\mathcal{N}|} \sum_{n \in \mathcal{N}} MSLE(y_{n,1}, t_{n,1}) + \frac{1-\lambda}{N} \sum_{n=1}^{N} WCE(y_{n,2}, t_{n,2})$$

where $y_{n,1}$ and $t_{n,1}$ are the predicted and actual demand for the nth training example; $y_{n,2}$ and $t_{n,2}$ are the predicted and actual classes for the nth training example; $\mathcal{N}$ is the set of examples where $y_{n,2}$ is nonzero, that is, the set of examples for which the classifier predicts nonzero demand; and MSLE and WCE represent mean squared logarithmic error and weighted cross-entropy error respectively. We perform a small grid search over $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$ and find that $\lambda = 0.75$ minimizes validation loss for our data.

3.2 Results

Table 1 shows the performance of four models as measured using mean absolute percentage error (MAPE), mean squared logarithmic error (MSLE), and root mean squared error (RMSE). The baseline model is the architecture described by [6]; baseline+MSLE is the baseline model trained to minimize MSLE; hierarchical is the hierarchical model with a heuristic classifier; and hierarchical+MSLE is the hierarchical model with a heuristic classifier, trained to minimize MSLE. Figure 2 gives a visual comparison of the demand predicted by each of the models.

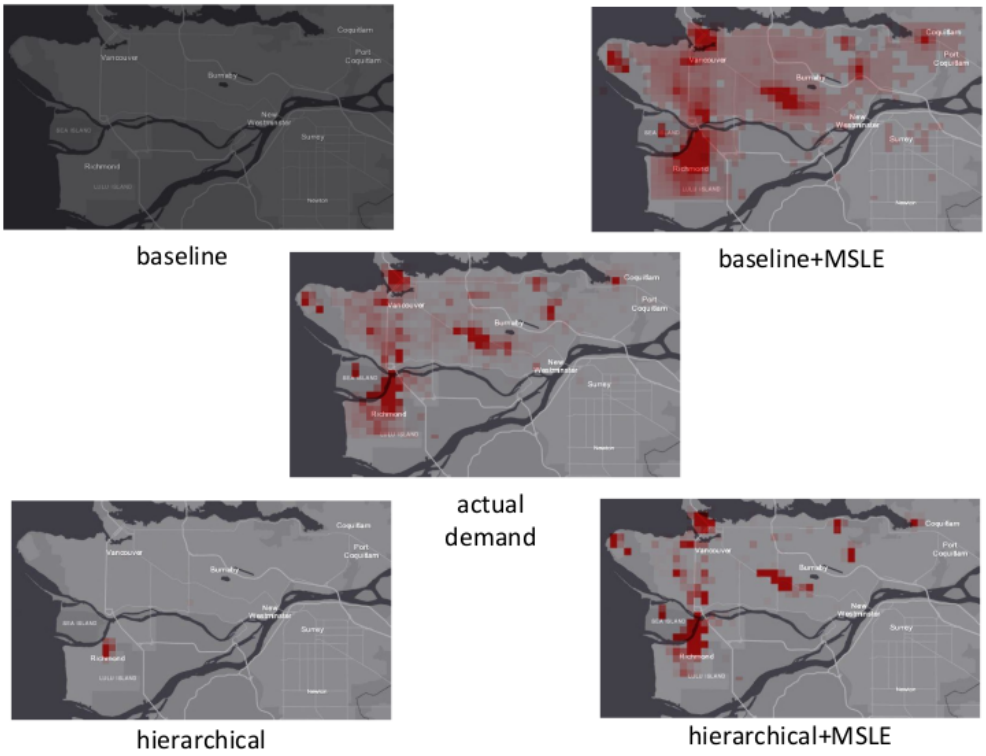

Figure 2: Comparison of the models from [6].

As predicted, the baseline model consistently predicts very low demand. We find that training to optimize MSLE improves prediction accuracy as measured by the MSLE and RMSE metrics, but for the baseline architecture it results in poorer accuracy as measured by MAPE. We posit that this is due to the asymmetry of MAPE and MSLE. MAPE assigns lower error to predictions which underestimate the true value; MSLE assigns lower error to predictions which overestimate. Since there are many zeros in our training data, our model will overestimate on most of the training examples, causing MAPE to grow even though the predictions may be closer to the truth in cases where demand is nonzero.

Model	MAPE	MSLE	RMSE
baseline	0.1114	0.0620	0.5890
baseline+MSLE	0.1399	0.0398	0.4501
hierarchical	0.1101	0.0597	0.5744
hierarchical+MSLE	**0.0499**	**0.0239**	**0.3128**

Table 1: Comparison of approaches for handling regions with low demand.

A hierarchical architecture improves accuracy as measured by all three of our metrics. We posit that this is because the hierarchical structure reduces the skew in the training data. It is no longer possible for the model to optimize the loss by always predicting zero, so the model's predictions move closer to the true values.

Optimal results are obtained using both the hierarchical structure and training to optimize MSLE. Optimizing MSLE helps the regression component ignore the imbalance in the training data, while the classifier component allows the model to ignore the regression and predict zero when appropriate. Figure 2 above also demonstrates the fact that hierarchical model combined with MSLE yields the result closest to the actual demand, which is represented in the middle of the figure.

3.2.1 Jointly Trained Models

Since the joint classifier-and-regression model has more parameters than the other models, time constraints prevented us from training this model on the full dataset.[2] We instead trained on a subset of 250,000 datapoints sampled from the original training data. We report results for this model separately, in Table 2; to ensure a fair comparison we compare against a version of the heuristic model that is trained on the same reduced dataset. Both use MSLE loss to train the regression portion of the network.

Model	MAPE	MSLE	RMSE
hierarchical+MSLE	**0.0977**	**0.0398**	**0.4319**
joint+MSLE	0.1691	0.0746	0.7299

Table 2: Comparison of heuristic classifier vs. jointly trained model.

To our surprise, joint training reduces performance relative to using a heuristic classifier. We posit that this may be due to poor hyperparameter settings. In principle joint training tends to improve the performance of neural models. In our case we only performed a small grid search to determine how to weight the terms $\mathcal{L}_R$ and $\mathcal{L}_C$ in our loss function; we predict that a more thorough search will reveal a value for λ which provides better performance than the heuristic classifier. However, given the added parameters involved in the joint model, and the fact that training this model is much slower, the overall gain may not be worthwhile. The heuristic classifier has the advantage of simplicity and faster training.

4 Conclusion

We have shown that the state-of-the-art demand prediction model due to [6] does not perform well on data where the average demand is very small. We consider two techniques for improving its performance in this setting. Training to optimize MSLE prevents the model from underestimating the average demand. Using a classifier to predict whether demand with be zero prior to regression reduces the degree to which the regression's training data are skewed. A heuristic classifier is sufficient to ensure good performance on our dataset, and does not increase training time relative to the baseline model. Jointly training a classifier and a regression is slightly less effective than using a heuristic classifier, possibly due to poor hyperparameter settings. The techniques discussed here are also applicable to improving other models for which the training data is heavily skewed.

[2]The joint model requires approximately twice as long per epoch relative to the baseline model.

Our code is available at `https://csil-git1.cs.surrey.sfu.ca/loborn/726-kabu-prediction`. If desired, please feel free to contact us by email with your user id and we can give you developer access.

5 Contributions

Chenhao Acquired raw data from local company. Data cleaning and pre-processing to get heatmaps, data generation for CNN and topological component, hierarchical+MSLE model training. Modified outputs to predict heatmaps.

Philip Participated in the research and aided the model refinement by training and calculating the losses in the hierarchical models. Supported the drafting of the final report and poster by generating heat-map images.

Logan Trained the baseline model; proposed and implemented the hierarchical and end-to-end models. Suggested MSLE as a means of unbiasing the model. Generated LSTM training data.

Zhuo Participated in research, generated the grid index. Figured out our model architecture and created the architecture figure. Combined the maps and heatmaps to get final images.

Yijing Participated in the research. Transformed the geographic coordinates into heatmap index. Trained and calculated the losses in baseline+MSLE models.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., 1st edition, 2017.

[3] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, February 1978.

[4] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. *CoRR*, abs/1503.03578, 2015.

[5] Chris Tofallis. A better measure of relative prediction accuracy for model selection and model estimation. *Journal of the Operational Research Society*, 66(8):1352–1362, Aug 2015.

[6] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. Deep multi-view spatial-temporal network for taxi demand prediction. 2018.

[7] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1655–1661. AAAI Press, 2017.