Comparison of Approaches to Large Scale Regression

Till Bungert
Mat-Nr.: 3302154
bungert@stud.uni-heidelberg.de

October 8, 2018

Abstract

Gradient boosting descision trees are the most popular model for tabular data regression on kaggle, however they require careful feature engineering. We investigate the performance of different methods on such a problem with zero feature engineering and find that neural networks perform well for this task when paired with embeddings. We also investigate differen Gaussian process based approaches, but find that they are harder to train than neural networks.

1 Introduction

Deep neural networks have become the de facto standard tool in many computer vision tasks, but other methods prevail for different tasks. The most popular and succesful method for regression on tabular data, at least on Kaggle, is gradient boosting decision trees [5].

In this report we look at different and less popular approaches to large scale regression. We try to estimate a function mapping taxi ride parameters to the corresponding price for $55M$ records.

We will first investigate neural networks, using entity embeddings for the categorical inputs as is common practice in neural language processing and

has been applied to a similar problem to ours by De Brébisson et al. [3].

We will then move on to investigate a more uncommon method and try different methods of applying Gaussian processes to the problem.

Finally we compare all our methods to gradient boosting decision trees.

2 Methods

2.1 Entity Embeddings

We map categorical variables with C categories represented by indices $c \in \{0, 1, \ldots, C-1\}$ to real-numbered vectors $\mathbf{x}_c \in \mathbb{R}^n$

$$\textit{Embedding} : \{0, 1, \ldots, C-1\} \to \mathbb{R}^n, c \mapsto \textit{Embedding}(c) = \mathbf{x}_c. \tag{1}$$

These embedding layers are implemented as lookup tables. The vector associated with each index is a parameter of the model and is learnd jointly with the rest of the model.

If the input to our model is a mixture of continuous and categorical variables as is the case here, we learn one embedding layer for each of the categorical variables and concatenate the vector components of each embedding output together with the continuous variables to one vector. This concatenated vector then serves as the input to the rest of the model.

2.2 Neural Networks

Feed-forward neural networks, sometimes called multilayer perceptrons, are one of many machine learning models designed to approximate some function f^*. They define a mapping $y = f(x, \theta)$ where θ is learned to result in the best approximation.

The name feed-forward neural network stems from the fact that they consist of intermediary functions called layers, that are chained together.

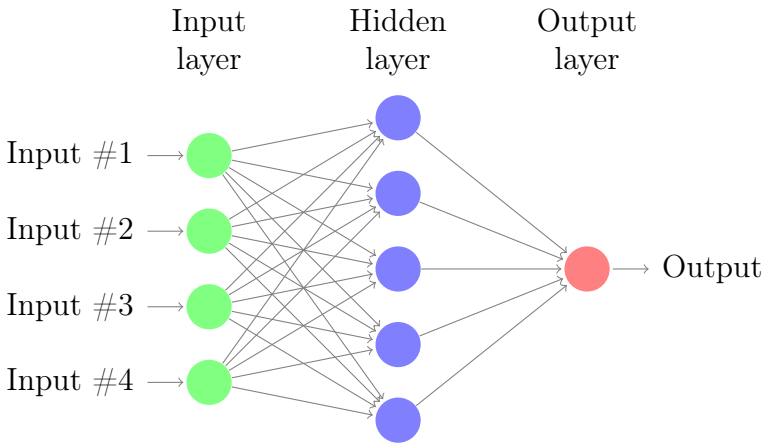


Figure 1: 2-Layer neural network

The length of the chain of these intermediary functions gives the depth of the network. As f^*, f and all intermediary functions are vertor valued, the dimensionality of the vector gives the width of the layer. If a layer is not the input or output layer it is called hidden. By depicting each vector component as a node, neural networks can be described by directed acyclic graphs as in Figure 1 [4].

In most cases, each layer consists of a linear function $y = w^T x$, where w are called the weights of that layer and are part of θ. As a chain of linear functions is still a linear function, we need something to make the neural network nonlinear to learn general functions. To acomplish this, each layer is associated with a nonlinear function called the activation function, that is applied to the result of the linear function. Popular choices for activation functions are the rectified linear unit (ReLU)

$$ReLU(x) = \max(0, x) \tag{2}$$

and the softmax [4].

2.3 Gaussian Processes

If we want to estimate some unobserved function $f = f(\mathbf{x})$ responsible for generating some set of observed variables $\mathbf{Y} \in \mathbb{R}^N$ from a corresponding set of input variables $\mathbf{X} \in \mathbb{R}^{N \times D}$, Gaussian Processes can be used to obtain nonparametric prior distributions over the latent function f.

Formally, each datapoint y_n is generated from $f(\mathbf{x}_n)$ by adding Gaussian noise

$$y_n = f(\mathbf{x}_n) + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \sigma_\epsilon^2 \mathbf{I}) \tag{3}$$

and f is drawn from a zero-mean Gaussian process

$$f \sim \mathcal{GP}(\mathbf{0}, k(x, x')) \tag{4}$$

defined by its covariance function k operating on the inputs $\mathbf{X}$. To obtain a flexible model only very general assumptions are made when choosing the covariance function. The popular radial basis function $k(x, x') = \sigma^2 \exp^{-\frac{(x-x')^2}{2l^2}}$ for instance only makes an assumption about the smoothness of f. We denote kernel function hyperparameters as θ. The marginal likelihood for a set of outputs given a set of inputs can now be analytically computed as [2]

$$p(\mathbf{Y}|\mathbf{X}) = \mathcal{N}(\mathbf{Y}|\mathbf{0}, k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon \mathbb{1}). \tag{5}$$

If want now to predict the output y^* for some novel input $\mathbf{x}^*$, we have to condition the joint distribution of the outputs

$$p(\mathbf{Y}, y^*|\mathbf{X}, \mathbf{x}^*) = \mathcal{N}(\mathbf{Y}, y^*|\mathbf{0}, \mathbf{K}^+), \quad \mathbf{K}^+ = \begin{pmatrix} \mathbf{K}_{\mathbf{XX}} & \mathbf{K}_{\mathbf{X}\mathbf{x}^*} \\ \mathbf{K}_{\mathbf{x}^*\mathbf{X}} & \mathbf{K}_{\mathbf{x}^*\mathbf{x}^*} \end{pmatrix} \tag{6}$$

on the known outputs to obtain [1]

$$p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \mathcal{N}(y^*|\ \mathbf{K}_{\mathbf{x}^*\mathbf{X}}\mathbf{K}_{\mathbf{XX}}^{-1}\mathbf{Y},\ \mathbf{K}_{\mathbf{x}^*\mathbf{x}^*} - \mathbf{K}_{\mathbf{x}^*\mathbf{X}}\mathbf{K}_{\mathbf{XX}}^{-1}\mathbf{K}_{\mathbf{X}\mathbf{x}^*}). \tag{7}$$

This computation, however, has complexity $\mathcal{O}(N^3)$.

To be able to use Gaussian processes for large datasets we apply stochastic variational inference to the model. For this we introduce a set of inducing variables $\mathbf{U}$ that represent the values of f at the points $\mathbf{Z} \in \mathbb{R}^{M \times D}$ living in $\mathbf{X}$ space, where $M < N$. Predictions then take the form

$$p(y^*|\mathbf{x}^*, \mathbf{Z}, \mathbf{U}) = \mathcal{N}(y^*|\mathbf{K}_{\mathbf{x}^*\mathbf{Z}}\mathbf{K}_{\mathbf{Z}\mathbf{Z}}^{-1}\mathbf{U},\ \mathbf{K}_{\mathbf{x}^*\mathbf{x}^*} - \mathbf{K}_{\mathbf{x}^*\mathbf{Z}}\mathbf{K}_{\mathbf{Z}\mathbf{Z}}^{-1}\mathbf{K}_{\mathbf{Z}\mathbf{x}^*}) \tag{8}$$

We estimate the inducing points $\mathbf{Z}$ using stochastic variational interference by using a variational distribution $q(\mathbf{U}) = \mathcal{N}(\mathbf{U}|\mathbf{m}, \mathbf{S})$ to place a lower bound on $p(\mathbf{Y}|\mathbf{X})$:

$$\log p(\mathbf{Y}|\mathbf{X}) \geq \sum_{i=1}^{N} \mathcal{L}_i - KL(q(\mathbf{U})||p(\mathbf{U})) \tag{9}$$

where $\mathcal{L}_i$ only depends on one input-output pair $\mathbf{x}_i, y_i$. The derivation and exact definition of $\mathcal{L}$ in out of scope for this report and not importantn. It can be found in [6]. The important property of this lower bound is that it is a sum of terms where each only corresponds to one input-output pair, allowing us to use stochastic gradient descent to train the model.

3 Experiments

The experiments are implemented in python using the popular PyTorch framework [1] and the probabilistic programing language Pyro [2]. Code for our experiments can be found on GitHub [3].

[1] `https://pytorch.org/`
[2] `http://pyro.ai/`
[3] `https://github.com/tbung/nyc-taxi-fare-challenge`

3.1 Data Set

The dataset consists of roughly $55M$ rows. We are given date and time of the taxi ride, GPS coordinates of start and end point and passenger count. The goal is to learn the price in dollars of each of those rides. An exerpt from the dataset can be seen in Table 1.

The first step to approach this challenge is analyzing the dataset. For that we first look at the distribution of the location data as shown in Figure 3. We find that the distributions are centered around New York, but reach very far. We also find nonsensical points in the water, unreachable by taxi, but choose to leave them in. We constrain the dataset to only the red boxed area, corresponding to the area in which the test set lies.

Next we take a look at the fare amount. A histogram and Gaussian kernel density estimate are given in Figure 2. As we can see there is one large spike and several smaller one. Other contestants on kaggle identified thos smaller spikes as corresponding do fixed price fares to airports and took distance to airport as an additional feature. We choose to ignore this and do no feature engineering. However we purge `NaN` entries and entries greater than 250$. Lastly we purge all entries with more than 6 passengers and normalize all our data.

We follow the approach of De Brébisson et al. [3] and treat time data as categorical, divided in the subcategories year, month, day of the week and quaterhour of the day.

Because of time constraints we limit the dataset to $8M$ randomly selected entries and split it in 90% training and 10% test data. Kaggle provides a validation set of about $10K$ entries.

Table 1: Excerpt from the dataset

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:00	-73.844315	40.721317	-73.841614	40.712276	1
1	16.9	2010-01-05 16:52:00	-74.016045	40.711304	-73.979271	40.782005	1
2	5.7	2011-08-18 00:35:00	-73.982735	40.761269	-73.991241	40.750561	2
3	7.7	2012-04-21 04:30:00	-73.987129	40.733143	-73.991570	40.758091	1
4	5.3	2010-03-09 07:51:00	-73.968094	40.768009	-73.956657	40.783764	1

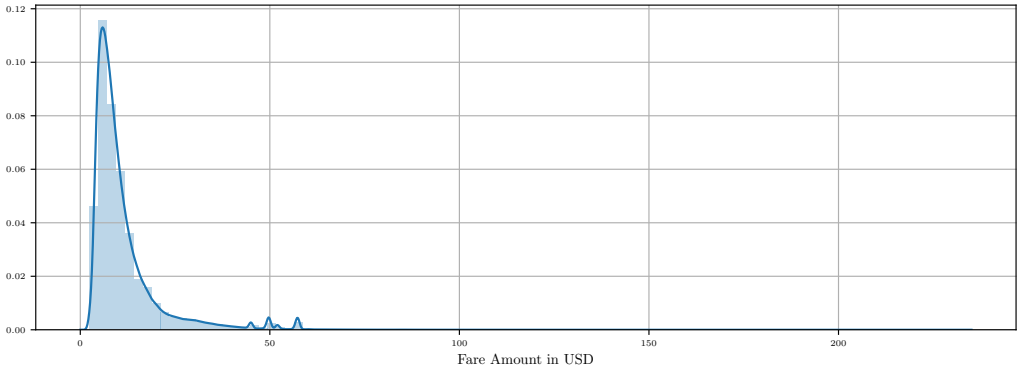


Figure 2: Distribution of fare amounts and Kernel Density Estimate. One can see an initial spike well estimated by a Gaussian distribution and few smaller spikes, most likely pertaining to fixed-price rides, e.g. to airports.

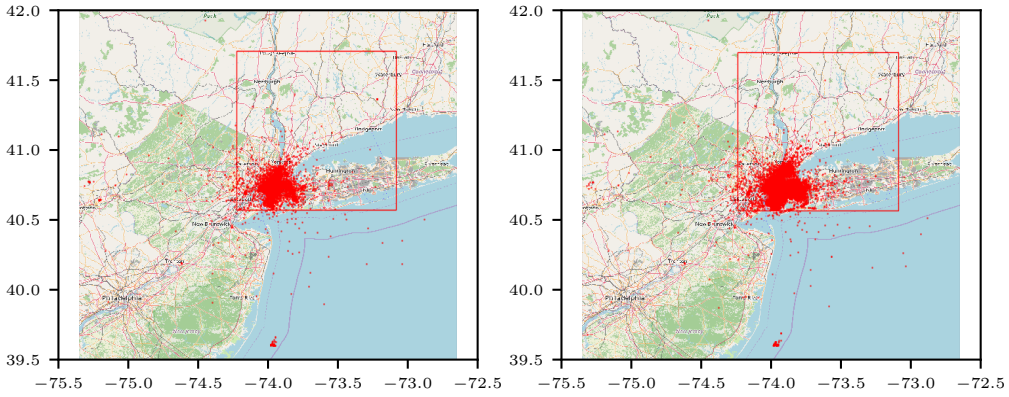


Figure 3: Distribution of the given location data in degrees. The area to which we constrain the data is displayed as a red rectangle. **Left:** Taxi ride pickup locations **Right:** Taxi ride dropoff locations

3.2 Model Setups

3.2.1 Deep Feed-Forward Network

We construct a fully-connected network consisting of three layers with 1024, 256 and 64 outputs with rectified linear units as activation functions and batch normalization applied after each layer. A final linear layer maps to the output space.

Following De Brébisson et al. [3], we apply embeddings to each categorical component in the inputs and learn the embedding vectors jointly with the network weights. The number of embedding dimensions for each variable can be found in Table 2.

Table 2: Dimension of embedding vectors

Variable	Number of Categories	Number of Dimensions
Passenger Count	6	3
Year	7	4
Month	12	6
Weekday	7	4
Quaterhour	96	50

Furthermore, again inspired by De Brébisson et al. [3], we apply K-means clustering to the targets in the data set and use the clusters to place a prior on our outputs. We do this by letting the network output a score for each cluster, applying a softmax function to place the scores in the $(0, 1)$ range. We then sum over all cluster centers weighted by the network scores to obtain the final output. The full process can be described by 1 where N is the sum of the embedding dimensions and number of continuous values, D is the number of outputs (either the number of clusters, or 1) and $w_{n\times m}$ and b_m are $n \times m$-dimensional weight matrix and m-dimensional bias vector of the respective layer. BN denotes batch normalization, $ReLU$ the rectified linear unit and $SoftMax$ the softmax function. The cluster centers are denoted by

Algorithm 1 Fully-connected Deep Neural Network architecture, inspired by [3]

function TAXINET($(x_{\text{categorical}}, x_{\text{continuous}})$)
  $e \leftarrow Embedding(x_{\text{categorical}})$
  $x \leftarrow (e, x_{\text{continuous}})$
  $y \leftarrow BN(ReLU(w^T_{N\times 1024} \cdot x + b_{1024}))$
  $y \leftarrow BN(ReLU(w^T_{1024\times 256} \cdot y + b_{256}))$
  $y \leftarrow BN(ReLU(w^T_{256\times 64} \cdot y + b_{64}))$
  $y \leftarrow SoftMax(w^T_{64\times D} \cdot y + b_{64})$
  $z \leftarrow \sum_{i=0}^{D} c_i y_i$
  return z
end function

c_i.

3.2.2 Gaussian Processes

We investigate two different Gaussian process approaches: deep kernel learning and deep gaussian processes. Both are learned using stochastic variational inference as described above.

In the deep kernel learning, the kernel function is prepended by a neural network, whos weights are learnt jointly with the inducing points[7]. For the kernel we choose the above mentioned radial basis function and use the same network as in the experiments above, but with two outputs and without computing cluster scores.

In deep Gaussian processes (DGP) we simply chain multiple Gaussian processes so that the input of one Gaussian process is the output of another Gaussian process. Here we investigate a 5-layer DGP using radial basis functions as the kernels.

3.2.3 Gradient Boosting

For our gradient boosting experiments we use LightGBM [4]. Please refer to our code for the exact setup, which is inspired by a kaggle kernel by user Sylas [5].

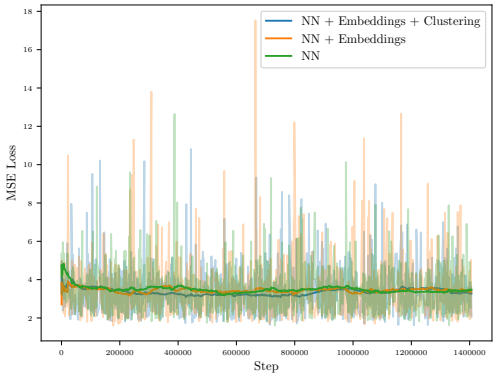
3.3 Results

Table 3: Obtained accuracy for each approach

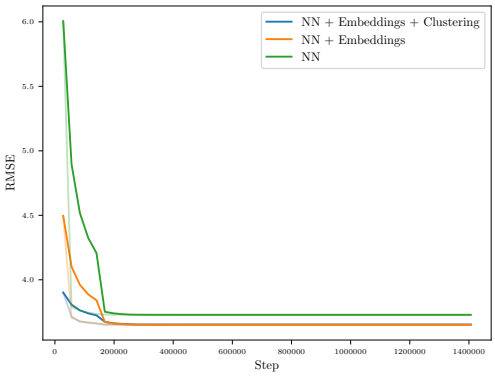
Approach	Validation Accuracy	Test Accuracy
Deep Neural Network with Embeddings and Clusters	3.65	**3.08**
DGP	6.34	5.98
Deep Kernel GP	3.66	3.29
Gradient Boosting	3.72	3.20

The results can be seen in Table 3. As we can see the neural network with embeddings and clusters performs best. Taking a look at the training curves in Figure 4 we can see, that the loss for the neural network is very noisy and the test accuracy converes quickly. The deep kernel approach seems to converge similarly while the DGP approach seems to converge much more slowly.

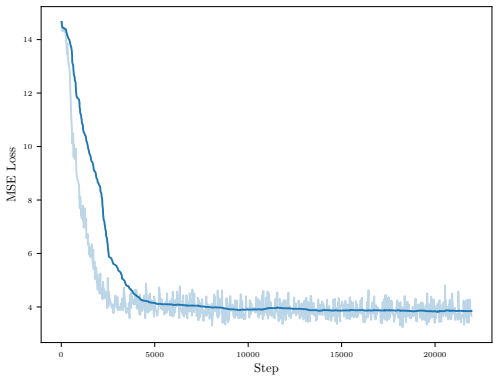
[4] `https://github.com/Microsoft/LightGBM`
[5] `https://www.kaggle.com/jsylas/python-version-of-top-ten-rank-r-22-m-2-88`



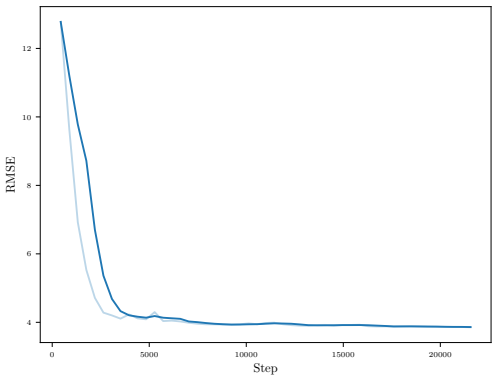
a Train loss for the neural network approach



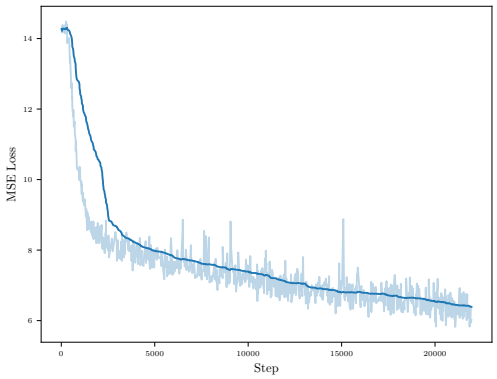
b Test accuracy for the neural network approach



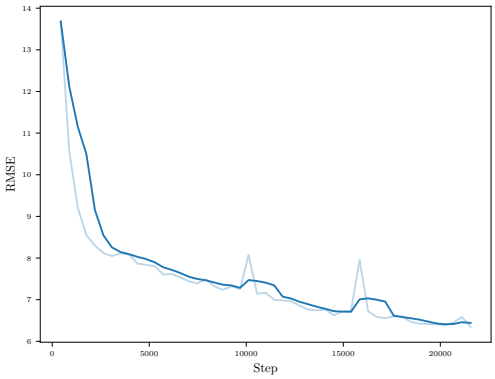
c Train loss for the deep kernel learning



d Test accuracy for the deep kernel learning



e Train accuracy for the DGP



f Test accuracy for the DGP

Figure 4: Training curves for all methods. Solid lines are smoothe by using a running mean, transparent lines are the true data.

References

[1] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[2] Andreas Damianou and Neil Lawrence. "Deep gaussian processes". In: *Artificial Intelligence and Statistics*. 2013, pp. 207–215.

[3] Alexandre De Brébisson et al. "Artificial neural networks applied to taxi destination prediction". In: *arXiv preprint arXiv:1508.00021* (2015).

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[5] Cheng Guo and Felix Berkhahn. "Entity embeddings of categorical variables". In: *arXiv preprint arXiv:1604.06737* (2016).

[6] James Hensman, Nicolo Fusi, and Neil D Lawrence. "Gaussian processes for big data". In: *arXiv preprint arXiv:1309.6835* (2013).

[7] Andrew G Wilson et al. "Stochastic variational deep kernel learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2586–2594.