

Sajad Gholamzadehrizi

CSC 44700 Machine Learning

Professor Dr. Erik Grimmelmann

Final Project Report

How much is my dream home?

Regression using TensorFlowJS

Introduction

Finding one's next place to live in can involve many difficulties. However, understanding the right budget for one's desired house is the first step in the process. This can involve a lot of guessing and searching on house listings to come up with a number for the rent one will need to afford to be able to move. In this project I have tried to aim to make this easier for people who want to find their next place in New York City, to be able to see a prediction of their desired place rent. I have provided a user interface for the user with a simplistic and intuitive user experience design to do so.

My goal out of this project was to build a web application that utilizes Tensorflow in JavaScript to give the user the ability to train, test and predict the rent based on user's criteria for their next NYC house. The user is also able to view visualization of the datapoints and prediction line of rents using Tensorflow JS library on the web environment. As a Full Stack developer intern, I am very interested in learning how we can incorporate Machine Learning models in web applications. Tensorflow in JavaScript provides web developers the advantages of running on the browser smoothly without the need to run that on the server side, giving access to device features

such as microphone, camera and locations, all of which have made this tool very popular among web developers.



Data Preparation

The first step in the process was to acquire the dataset needed for the purpose of the application. After looking for datasets on Kaggle (1), I found a suitable dataset providing a very large USA houses listings for rent that was originally scraped from Craigslist. This dataset was quite up to date, being updated 5 months ago, and it included columns that would help the prediction of rent based on an average person's criteria such as square footage of the houses, number of bedrooms and etc. As you can see below, this dataset provides 384,977 of houses listings which is beyond the scale of this application. Therefore, I decided to focus on the New York City housing market. New York City which is the largest city in the US, would give me enough number of data, 763 houses, to be able to build a reliable model, and it would bring down the scale of dataset to make it easier to process the information.

Data Explorer

532.57 MB

housing.csv

< housing.csv (532.57 MB)				
Detail	Compact	Column	10 of 22 columns	
id	url	region	region_url	price
listing id	listing URL	craigslist region	region URL	rent per month
	384977 unique values	jacksonville 1% columbus 1% Other (376993) 98%	413 unique values	
	ist.org/apa/d/sparks-961-marble-hills-cir/7849819568.html		ist.org	
7849819739	https://reno.craigslist.org/apa/d/reno-1-beds-coming-up-patio-with-every/7849819739.html	reno / tahoe	https://reno.craigslist.org	1000
7849819563	https://reno.craigslist.org/apa/d/reno-2nd-floor-1-bed-1-bath-fully/7849819563.html	reno / tahoe	https://reno.craigslist.org	950
7849818229	https://reno.craigslist.org/apa/d/reno-	reno / tahoe	https://reno.craigslist.org	950

After inspecting the dataset, I realized that there are 40 rows of houses listings which are invalidly scraped. For an expensive city like NYC, it would be quite impossible to find a place that costs less than \$400 a month. I also filtered out the houses that are too small to count as a habitable house. Houses of smaller than 200 sqft in size are decarded from the dataset. I also simplified the dataset by discarding columns that included less meaningful information for the purpose of our regression model such as web URL of the listing and description.

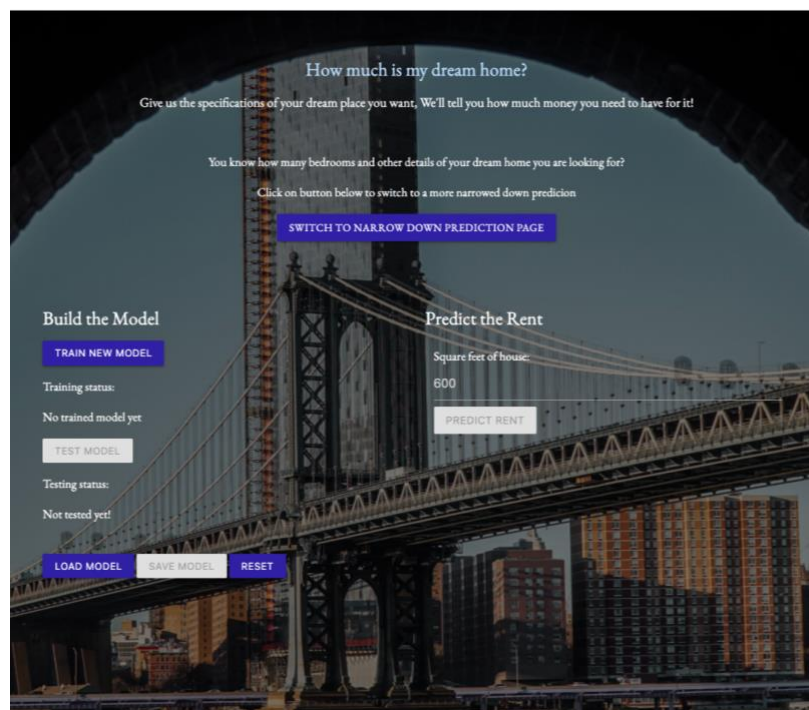
```
processing...
Skipping an out of range house with a price of 2500 and SQFT of 20
Skipping an out of range house with a price of 16 and SQFT of 735
Skipping an out of range house with a price of 2552 and SQFT of 100
Skipping an out of range house with a price of 2 and SQFT of 2300
Skipping an out of range house with a price of 25 and SQFT of 1100
Skipping an out of range house with a price of 4300 and SQFT of 20
Skipping an out of range house with a price of 1850 and SQFT of 20
Skipping an out of range house with a price of 3600 and SQFT of 20
Skipping an out of range house with a price of 3000 and SQFT of 20
Skipping an out of range house with a price of 3150 and SQFT of 20
Skipping an out of range house with a price of 3900 and SQFT of 20
Skipping an out of range house with a price of 4000 and SQFT of 20
Skipping an out of range house with a price of 3231 and SQFT of 20
Skipping an out of range house with a price of 3025 and SQFT of 20
Skipping an out of range house with a price of 2500 and SQFT of 20
Skipping an out of range house with a price of 2150 and SQFT of 20
Skipping an out of range house with a price of 1200 and SQFT of 0
Skipping an out of range house with a price of 1899 and SQFT of 20
Skipping an out of range house with a price of 0 and SQFT of 1100
Skipping an out of range house with a price of 3150 and SQFT of 20
Skipping an out of range house with a price of 17 and SQFT of 780
Skipping an out of range house with a price of 2695 and SQFT of 20
Skipping an out of range house with a price of 2000 and SQFT of 20
Skipping an out of range house with a price of 16 and SQFT of 525
Skipping an out of range house with a price of 1 and SQFT of 900
Skipping an out of range house with a price of 2 and SQFT of 2300
Skipping an out of range house with a price of 1895 and SQFT of 20
Skipping an out of range house with a price of 1895 and SQFT of 20
Skipping an out of range house with a price of 1850 and SQFT of 20
Skipping an out of range house with a price of 2300 and SQFT of 20
Skipping an out of range house with a price of 17 and SQFT of 780
Skipping an out of range house with a price of 2 and SQFT of 1000
Skipping an out of range house with a price of 25 and SQFT of 1100
Skipping an out of range house with a price of 16 and SQFT of 525
Skipping an out of range house with a price of 6995 and SQFT of 75
Skipping an out of range house with a price of 1900 and SQFT of 70
Skipping an out of range house with a price of 980 and SQFT of 75
Skipping an out of range house with a price of 2 and SQFT of 2300
Skipping an out of range house with a price of 1 and SQFT of 900
Skipping an out of range house with a price of 25 and SQFT of 1100
NYC Lisitings processed, number of listings: 723
```

The result of processing our dataset is 723 New York City houses that are listed for rent, which we will use to train our model. Next, I picked input and output features to map them to feature values and label values that I will explain in more details in the training section. Afterwards, I shuffled the data to avoid any possible patterns in order of data or features values, so no split of datasets has more of one category of values over another. Then, I split this dataset into 50% of training dataset and 50% of testing dataset.

User Interface – First Page

As discussed above, we should get interactions from the user. I built a simple web page as a start for the users who are only interested to quickly predict the rent of their desired house by providing a single attribute in this case just the size of the house in sqft. Additionally, on this page the user is able to build the model. They can view the training error and validation error once the training is done, and they are able to test the model to see how well the model is tested by viewing the testing loss. Moreover, in order to make the web app more efficient and less time consuming for multiple runs, the user is able to save the model and load the previously saved models. There is also a reset button to reset the page and the clear the model.

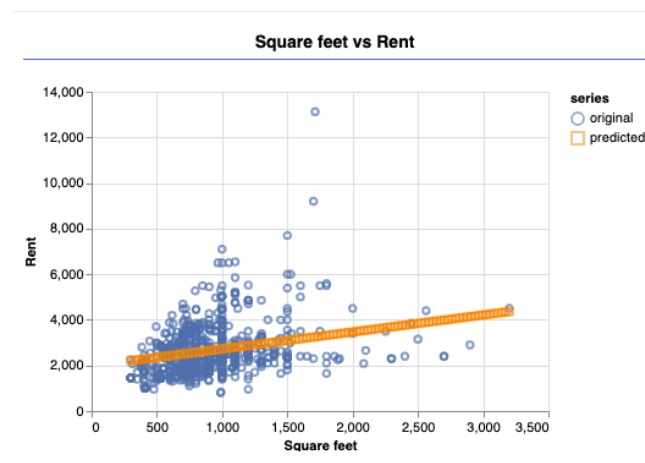
Additionally, there is a visor toggle button that allows the user to view the data visualization for their dataset and the training performance.



Training the Model

To get started and for this use case, I took a simplistic approach to build the model with lower computation cost. Initially, on page load, using TensorFlow JS Visor library, the visualization of datapoints for the two main features of sqft vs rent is shown. As we can see in a sample snapshot of graph below, there is a linear trend in the distribution of datapoints when comparing the features of size in sqft and rent is US dollars. This can be seen in the prediction line as well, that is generated after the user trains the model. On the visor, it can be seen that over epochs, the prediction line gets generated and updated over epochs and shows how the model predicts the rent for one hundred data points of normalized sqft.

On the graph below, you can view a snapshot of the original datapoints for two features of rent vs. sqft and the prediction line in orange which in this case has been updated on the last epoch.



Model is built using the Layers API in TensorFlowJS. I built a sequential model with dense layers that are connected together. Although for this simple model, I decided to start by using just one dense layer to see how it works out. With a linear activation function and one input dimension and an optional bias set to true.

```

// sequential model, output of a layer is the input of next layer
model = tf.sequential();

// single layer, single node, linear
// dense, all inputs and outputs are connected
model.add(tf.layers.dense({
  units: 1,
  useBias: true,
  activation: 'linear',
  inputDim: 1,
}));

```

Additionally, I set the loss function to the built-in Mean Squared Error in the TensorflowJS library. Since our data is already normalized (using min max normalization method (2)), this is a pretty reasonable loss function and choosing other loss functions such as RMSE might not be needed. As for the optimizer, I chose the built-in Stochastic Gradient Descent with a learning rate of 0.01. This learning rate gave me the best performance overall.

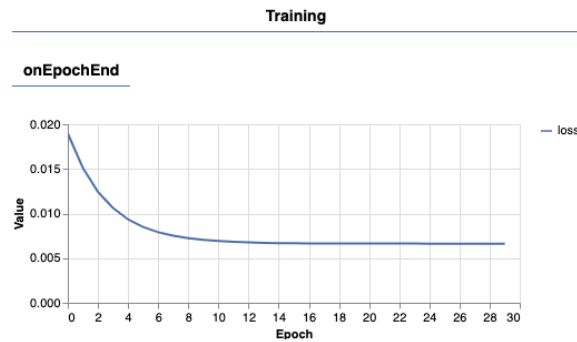
Next, I fit the model with the training feature and training label. I chose 32 batch size and 30 epochs as it seemed pretty reasonable for the loss I observed as the training was progressing on the graph. For a safeguard against overfitting, we also specify a validation split of 20% so we can later on check the loss of validation set as well independently from training dataset.

```

// fit method to train for a certain number of epochs, used 20 first, then 30
// depends on training dataset and learning rate
return model.fit(trainingFeatureTensor, trainingLabelTensor, {
  // in each epoch, there'll be number batches of data to be optimized
  batchSize: 32,
  epochs: 30,
  // 20% of dataset for validation for safeguard to check overfitting
  // calculating loss independently from training data
  validationSplit: 0.2,
  callbacks: {
    onEpochEnd,
    // to log loss values on each epochs
    // onEpochEnd: (epoch, log) => console.log(`Epoch ${epoch}: loss = ${log.loss}`)
    onEpochBegin: async function () {
      await plotPredictionLine();
      const layer = model.getLayer(undefined, 0);
      tfvis.show.layer({ name: "Layer 1" }, layer);
    }
  }
});

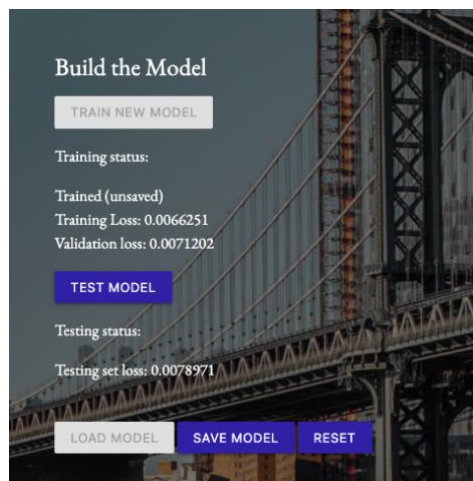
```

And the graph below shows how training performed on each epoch for 30 epochs. The Training loss is 0.0066251 and validation loss is 0.0071202. We can see from the result that the model has done pretty well, and the results are quite accurate.

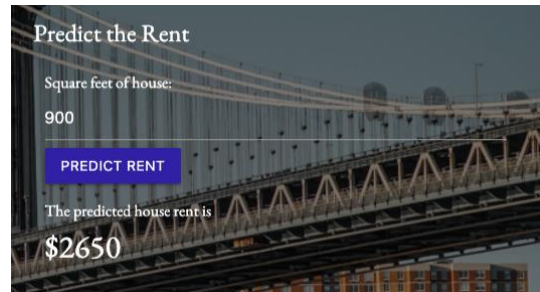


Test and Results

When the user is done training the model, they can test the model. As shown below, testing set loss is 0.0078971 which is very close to training set and validation set loss values. We have observed that the model has been successfully trained and tested even with a simple linear regression model of one dense layer.



Now the user is finally able to use the model to predict the rent of their desired house with the provided size in sqft. As seen below, the prediction in this case is \$2,650 for a house of 900 sqft.



Discussion

Even though the model turned out pretty well, I decided to try out some other optimizer leaning rates or structures of a model. First, keeping the same structure, I tried using different optimizers or learning rates. When I used 0.1 as learning rate, it gave a slightly bigger loss, but it was constant in fewer epochs. Using 0.001, it took more epochs to get to the minimum loss. Using 0.5 the training did not perform as well, it tended to get bigger and smaller periodically over epochs and in the end, the loss was larger than desired. Using 1.0 as learning rate, the loss became larger and larger on each epoch and the training did not do well at all. Lastly, I tried using `adam()` the built-in optimizer that does not need any learning rate, this optimizer gave me a similar result as when I used `sgd(0.01)` meaning Stochastic Gradient Decent with 0.01 as our learning rate.

```
// optimizer
// stochastic gradient descent builtin optimizer, with a learning rate of 0.01
// 0.1: good starting point
// 0.01: reduced loss, but slightly more number of epochs to train
// 0.001: taking longer to min
// 0.5: wobbly moving away from min, bit too large
// 1.0: very much too large, diverging away from min
// adam: different optimizer algorithm, works effectively without learning rate, the algo adapts learning rate over epochs
const optimizer = tf.train.sgd(0.01);

// prepares model for training and testing, chose loss func from builtin tf.loss.meanSquaredError
// RMSE with already normalized data is overkill, so we just use MSE
model.compile({
  loss: 'meanSquaredError',
  optimizer,
});
```


Next, I decided to make the dense layers more complex. I also I added more layers but did not see any major improvement in performance. I also tried to use a non-linear activation function. This time I used sigmoid activation function (3) and added multiple layers to the model as well, again I did not see any major improvement in training performance that would justify making the computations more costly.

```
// do it non-linearly with sigmoid activ func
// model.add(tf.layers.dense({
//   units: 1,
//   useBias: true,
//   activation: 'sigmoid',
//   inputDim: 1,
// }));

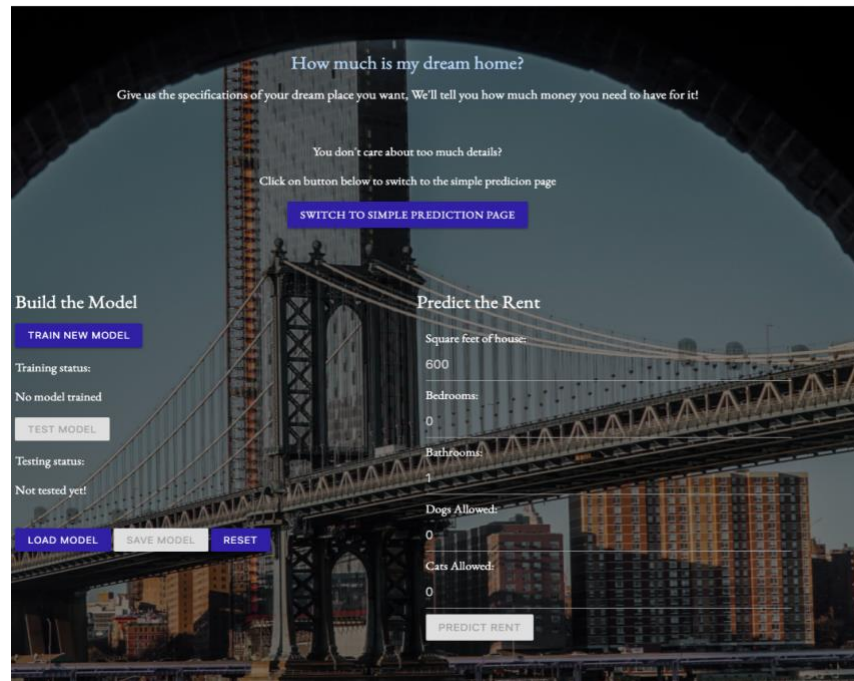
// add multiple layers to model
// speeds up training process, but more vulnerable to overfit
// model.add(tf.layers.dense({
//   units: 10,
//   useBias: true,
//   activation: 'sigmoid',
//   inputDim: 1,
// }));

// model.add(tf.layers.dense({
//   units: 1,
//   useBias: true,
//   activation: 'linear',
// }));
// model.add(tf.layers.dense({
//   units: 1,
//   useBias: true,
//   activation: 'linear',
// }));

// model.add(tf.layers.dense({
//   units: 10,
//   useBias: true,
//   activation: 'sigmoid',
// }));
// model.add(tf.layers.dense({
//   // only one label
//   units: 1,
//   useBias: true,
//   activation: 'sigmoid',
// }));
```

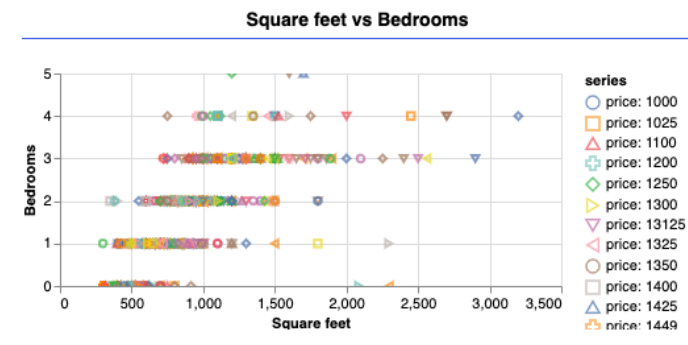
User Interface – Second Page

Now, to make my web application more interesting, I provided a second user page for a more narrowed down prediction criteria for the user. This time, in addition to the same train and test functionalities, the user is able to provide five features: rent of their desired house, number of bedrooms, bathrooms, whether or not they want the place to allow dogs or cats.



Training the Model

The user is to have a look at the data visualization on page load. The graph shows how the distribution of houses are based on two features of sqft vs. bedrooms with various rents.



Once the user trains the model, they can view how their model performs on the graph below, on each epoch.

Model is built using the Layers API in TensorFlowJS again. To build our Artificial Neural Network, I built a sequential model with three dense layers that are connected together. It has a linear activation function and five input dimensions and an optional bias set to true. Unit of the output layer is one. Additionally, the data will be normalized again, and I also set the loss function to the built-in Mean Squared Error in the TensorflowJS library. As for the optimizer, this time I chose the adam() built-in optimizer that does not need an explicitly defined learning rate. The algorithm adapts the learning rate over epochs.

```
model = tf.sequential();

// using linear
model.add(tf.layers.dense({
  units: 10,
  useBias: true,
  activation: 'linear',
  inputDim: 5,
}));
model.add(tf.layers.dense({
  units: 10,
  useBias: true,
  activation: 'linear',
}));
model.add(tf.layers.dense({
  units: 1,
  useBias: true,
  activation: 'linear',
}));

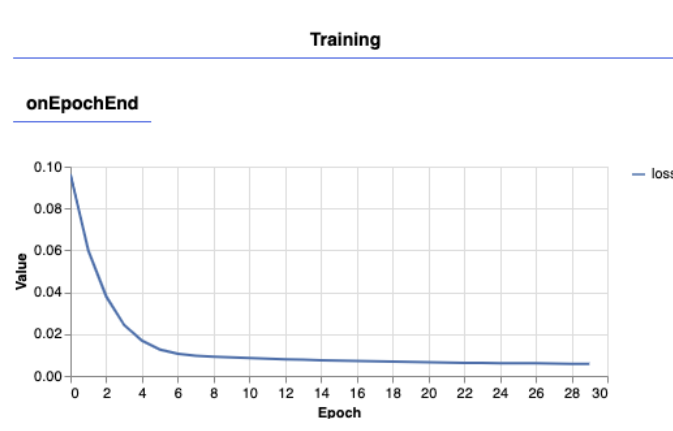
const optimizer = tf.train.adam();
model.compile({
  // using MSE
  loss: 'meanSquaredError',
  optimizer,
});
```

Next, I fit the model with the five training features: rent, bedrooms, bathrooms, dogs_allowed, cats_allowed and training label: price. Again, I chose 32 batch size and 30 epochs as it seemed pretty reasonable for the training performance I observed as the training was progressing on the graph. And following the same standard, we also specify a validation split of 20% so we can later on check the loss of validation set as well independently from training dataset.

```
// function to train our model
async function trainModel(model, trainingFeatureTensor, trainingLabelTensor) {
  const { onBatchEnd, onEpochEnd } = tfvis.show.fitCallbacks(
    { name: "Training" },
    ['loss']
  );
  // return the fit model
  return model.fit(trainingFeatureTensor, trainingLabelTensor, {
    batchSize: 32,
    epochs: 30,
    validationSplit: 0.2,
    callbacks: {
      onEpochEnd,
      onEpochBegin: async function () {
        const layer = model.getLayer(undefined, 0);
        tfvis.show.layer({ name: "Layer 1" }, layer);
      }
    }
  });
}
```

sejad1993gh, 5 days ago • minor changes

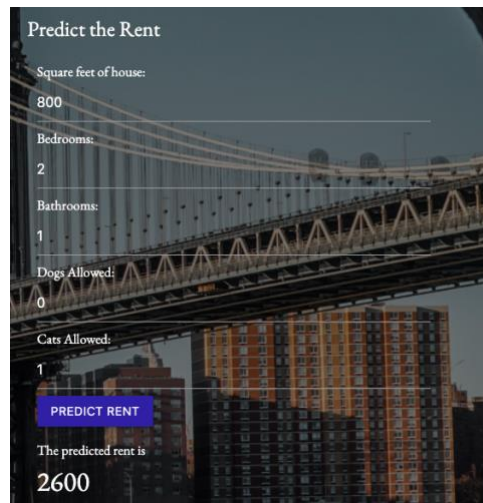
And the graph below shows how training performed on for 30 epochs. The Training loss is 0.0056970 and validation loss is 0.0070074. We can see from the results that the model has done pretty well, and the results are quite consistent. The training loss is also not much different from the previous model in the first page.



Test and Results

When the training is done, the user is able to test the model. As shown below, testing set loss is 0.0068991 which is close to training set and validation set loss values. We have observed that the model has been successfully trained and tested even. Now the user is able to finally use the model to predict the rent of their desired house with the provided size in sqft, number of

bedrooms and bathrooms, if need to allow dogs or cats. As seen below, the prediction in this sample case is \$2,600 for a house of 800 sqft, 2 bedrooms, 1 bathrooms, no dogs, but cats allowed.



Predict the Rent

Square feet of house:
800

Bedrooms:
2

Bathrooms:
1

Dogs Allowed:
0

Cats Allowed:
1

PREDICT RENT

The predicted rent is
2600

Final Thoughts

From this project, I learned how one can incorporate Machine Learning in a practical way for any web application without even the need for coding on the server side. TensorFlowJS uses a backend powered by WebGL to take advantage of the GPU. I also learned how important it is to acquire and clean the data properly. Data preparations took more time than I expected as the amount of data was very large and inspecting and preparing was challenging at first. I also learned how linear and non-linear regression works in TensorFlow JavaScript. Data Visualization in tf-vis library helped me understand the overview insights of my data and draw the prediction line and training performance of my models. Trying out different use cases and different input features and various model structures helped me have a better understanding of regression in TensorFlow. Although this web application was not as complex as a typical application in the market, I believe learning how to design and implement a small but intuitive

application that can utilize Machine Learning and work as a smaller piece of a larger application, inspired me to further expand my skills in utilizing Machine Learning in a practical sense in the future.

Works Cited

- 1- <https://www.kaggle.com/austinreese/usa-housing-listings>
- 2- [https://en.wikipedia.org/wiki/Feature_scaling#Rescaling_\(min-max_normalization\)](https://en.wikipedia.org/wiki/Feature_scaling#Rescaling_(min-max_normalization))
- 3- <https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8>
- 4- Udemy, Machine Learning in JavaScript Course