Machine Learning Final Project Report

November 11, 2020  
INFR 3700  
Reese Daniel & Shayan Saif  
University of Ontario Institute of Technology

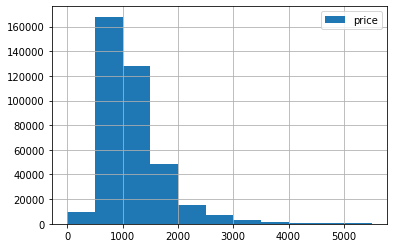
## Introduction

The United States of America is a large country with a diverse range of places to live. There are urban and rural environments, big mansions and shoebox apartments, and prices that range dramatically depending on location (location, location!). In this project, we aim to use recent, existing data to train machine learning models to accurately predict how much a given housing unit should cost per month, depending on a variety of factors.

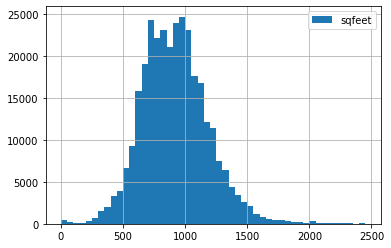
The USA Housing Listings dataset, which is the dataset we elected to use for this project, scrapes data from Craigslist related to apartments, condos, houses, and other housing types available for rent across the United States. The dataset includes around 385,000 observations and 22 columns. Columns include price per month, type of property (apartment, house, townhouse, condo, etc.), square footage, number of beds, number of bathrooms, latitude, longitude, US state, whether cats/dogs are allowed, whether smoking is allowed, if the property is wheelchair accessible, if there is electric vehicle charging, and if the property comes furnished. The data were first scraped from Craigslist in December 2019, and were most recently updated in June 2020.

## Data Graphs & Insights

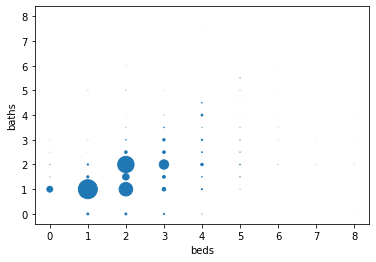
Frequencies of housing types. Apartments constituted the hefty majority of the dataset. For the sake of simplicity and to avoid the need to do one hot encoding, all housing types besides apartments were removed. Thus, the ‘type’ column was removed as well.



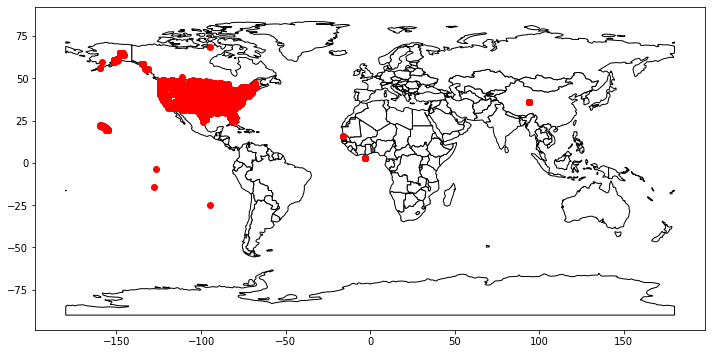
Histogram of housing prices. Most housing listings in the dataset are between $500 and $4,000, indicating that our model will most likely be much better at predicting prices for apartments that cost only a few thousand dollars per month. To prevent outliers from negatively affecting our models, we removed all apartments with prices greater than $10,000. There were also 1070 rows with a price of $0. We assumed that these are outliers because the listers probably just did not want to share online the price of their units for rent. These rows were removed.



Histogram of housing square feet. Most house listings in the dataset are less than 1500 square feet. 665 rows indicated a square footage of less than 100 sq. ft., so these rows were removed, since they were likely errors.



Scatter showing number of beds vs. number of bathrooms. Larger markers indicate a higher occurrence of that bed/bathroom combination. By far, the most common units have 1 bed/1 bathroom, 2 beds/2 bathrooms, or 2 beds/1 bathroom. The range of beds is [0,8] and the range of bathrooms is also [0,8], giving us 64 different combinations.



When the data are plotted on a world map, we can see that a number of points have incorrect latitudes and longitudes, since this dataset is supposed to only reflect the United States. All points outside of the continental United States were removed.

If we zoom in on the continental US and color each point by state, we see that the dataset covers close to the entire country, with only a little bit of sparseness in the west. This means that our models should be able to predict pricing accurately irrespective of state.

## Data Preparation

The following columns were removed from the dataset, as they took up valuable space and were not needed for the machine learning algorithms:

* ID (a unique ID for each row)
* URL (the URL for the Craigslist listing)
* Region (the name of the region the listing was in)
* Region\_URL (the URL for the region’s website)
* Image\_URL (the URL for the listing’s image)
* Description (some text describing the property)
* Laundry\_Options (an enum of 5 different types of laundry)
* Parking\_Options (an enum of 7 different types of parking)

1,920 rows were dropped because they contained missing values. 2 housing units were listed as having 1000 beds and 2 other units were listed as having 1100 beds, so these 4 outliers were removed. The state column was removed as this information can be deduced from the longitude and latitude. This left us with the following columns: price; sqfeet; beds; baths; cats\_allowed; dogs\_allowed; smoking\_allowed; wheelchair\_access; electric\_vehicle\_charge; comes\_furnished; latitude; longitude. The data were finally split into 80% training, 20% testing sets. From each set, the price and state columns were removed.

## Machine Learning Algorithms

### Linear Regression

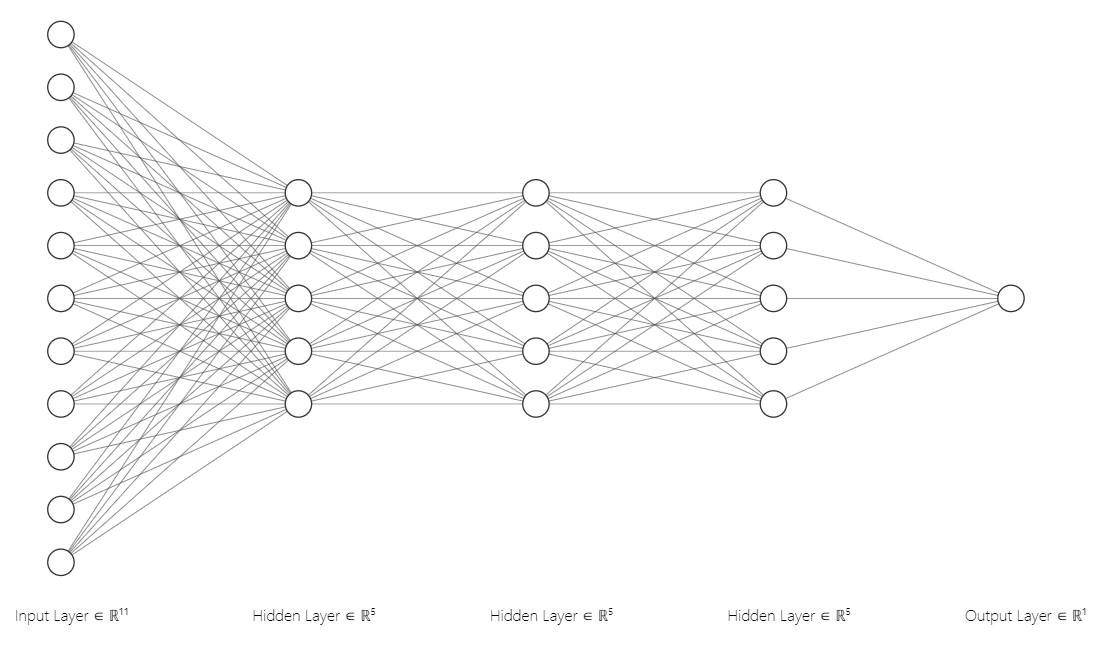
Using the linear regression class from the sklearn.linear\_model library, we fitted the model with the training features and the prices for those features. Afterwards, the algorithm was able to determine the coefficients of the features and the intercept. Furthermore, the model made predictions based on the test data and had a root mean squared error of 511.68.

### Decision Tree Regressor

We imported the Decision Tree Regressor class from the sklearn.tree library. We initialized it with a max\_depth of 4, a minimum leaf sample of 0.1, and random\_state set to 3. Afterwards, we fitted the model with the training dataset and then predicted the test dataset. Afterwards, we extracted the root mean squared error, which compared to the linear regression model. It was 464.74, making it the better algorithm in this case.

### Deep Neural Network

We scaled the training and testing features using the StandardScaler from the preprocessing scikit learn library. This ensured that the algorithm could easily distinguish between the different values it was given. Afterwards, we created the model using the following layers:



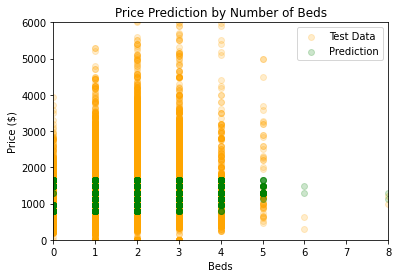
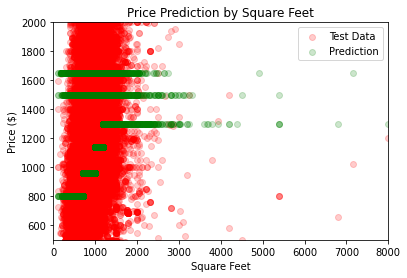
The model takes in 11 features (sqfeet, beds, baths, etc.) and processes it through three hidden relu layers of size 5, and outputs 1 value. The model was compiled using mean squared error as the loss, adam as the optimizer, and mean squared error as the metric. Finally, the model was run using the training and testing dataset. The results were very poor and the model improved very slightly through 10 epochs. We believe this may be because the optimizer had found a local minimum for the loss and could not be improved further.

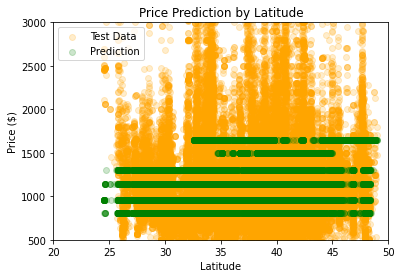
### Performance

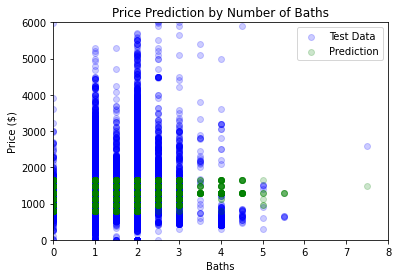
|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Root Mean Squared Error | Data Mean | Accuracy (%) |
| Linear Regression | 511.68 | 1151.66 | 55.57% |
| Decision Tree Regressor | 464.74 | 59.65% |
| Deep Neural Network | 516.16 | 44.82% |

### Graphs

Decision Tree Regressor was the most accurate regression model for this dataset. Here are its predictions for different features as the x-axis.







## Appendix

### Link to Dataset

<https://www.kaggle.com/austinreese/usa-housing-listings>

### Source Code

"""

# Machine Learning Final Project

Reese Daniel & Shayan Saif | November 2020 | INFR 3700 | University of Ontario Institute of Technology

## Data Graphs & Insights

"""

#import all the things that we need here

!pip install -q geopandas

import pandas as pd

import numpy as np

import seaborn as sn

from matplotlib import pyplot as plt

import geopandas as gpd

from shapely.geometry import Point, Polygon

#import the csv from S3, header=0 means the first row contains col names, index\_col=False means pandas should come up with its own row IDs, usecols only includes the desired columns

data = pd.read\_csv('https://ml-final-project.s3.amazonaws.com/housing\_exported.csv',header=0, index\_col=False,

usecols=['price','type','sqfeet','beds','baths','cats\_allowed','dogs\_allowed','smoking\_allowed','wheelchair\_access','electric\_vehicle\_charge','comes\_furnished','latitude','longitude','state'])

#Show the first 5 rows to get a feel for what the dataframe looks like

data.head()

#Drop all rows that have 1 or more missing values

print("Dropping",len(data)-len(data.dropna()),"rows because they contain missing values")

data.dropna(inplace=True)

#Look at the distribution of number of beds

data.beds.groupby(data.beds).describe().sort\_values(by='count')

# There are 4 outliers with >= 1000 beds; let's remove them

data = data[data.beds <= 8]

#Histogram of prices

data.price.hist(bins=np.arange(start=0,stop=6000,step=500), legend=True, figsize=(6,4))

#Seems like most housing units cost less than $4000

#Let's remove all rows where the price is > $10,000

print("Removing",len(data[data.price > 10000]),"rows because their price is an outlier")

data = data[data.price <= 10000]

#Remove any rows with a price of $0; this is probably just what the lister does when they don't want to share the price online

print("Removing",len(data[data.price == 0]),"rows because the price is $0 and that's gotta be untrue")

data = data[data.price > 0]

#Histogram of type column

data.type.groupby(data.type).hist(figsize=(6,4), xrot=45)

#Seems like the hefty majority of housing units are apartments; to simplify our models, let's just only keep apartments and drop the column

print("Removing",len(data[data.type != 'apartment']),"rows because they aren't apartments")

data = data[data.type == 'apartment']

data.drop(columns=['type'], inplace=True)

print("The data frame now has",len(data),"rows")

#Histogram of sqfeet

data.sqfeet.hist(bins=np.arange(start=0,stop=2500,step=50), legend=True, figsize=(6,4))

#Let's remove any rows where the sqfeet is less than 100 since this probably isn't true

print("Removing",len(data[data.sqfeet < 100]),"rows since they are listed as under 100 sq ft")

data = data[data.sqfeet >= 100]

#Plot a scatter with x-axis being number of beds and y-axis being number of baths

bedbath = data.groupby(['beds','baths']).size().reset\_index(name='freq')

bedbath.plot.scatter(x='beds',y='baths',s=bedbath.freq/300)

#Correlation matrix and heatmap

sn.heatmap(data.corr(), annot=True)

data.corr()['price'].sort\_values()

#Remove any rows where longitude falls outside of [-180,180]

#Remove any rows where latitude falls outside of [-90,90]

data = data[data.longitude <= 180]

data = data[data.longitude >= -180]

data = data[data.latitude <= 90]

data = data[data.latitude >= -90]

#Plot on a world map to find outliers

gdf = gpd.GeoDataFrame(data, geometry=gpd.points\_from\_xy(data.longitude, data.latitude))

world = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_lowres'))

ax = world.plot(color='white', edgecolor='black')

gdf.plot(ax=ax, color='red')

#A number of rows have incorrect latitudes and longitudes

#Let's restrict our data to the continental US for simplicity

data = data[data.longitude <= -64]

data = data[data.longitude >= -144]

data = data[data.latitude <= 49]

data = data[data.latitude >= 24]

#Plot on a map of North America

gdf = gpd.GeoDataFrame(data, geometry=gpd.points\_from\_xy(data.longitude, data.latitude))

world = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_lowres'))

ax = world[world.continent == 'North America'].plot(color='white', edgecolor='black')

gdf.plot(ax=ax, color='red')

#Scatter plot, longitude vs. latitude, colored by state

color\_labels = data['state'].unique()

rgb\_values = sn.color\_palette("Set2",50)

color\_map = dict(zip(color\_labels, rgb\_values))

data.plot.scatter(x='longitude',y='latitude', c=data['state'].map(color\_map))

"""## Data Preparation"""

from sklearn.model\_selection import train\_test\_split

#Split data into 80% training and 20% testing data

train, test = train\_test\_split(data, test\_size=0.2, random\_state=69)

#train\_target is only the price column, since that's what we're trying to predict

train\_target = train['price']

#Same thing with test\_target

test\_target = test['price']

#train\_features and test\_features will hold all the columns we will use to train and test the models.

#Price is removed because that's what we're predicting. State is removed because it's a class and we aren't doing one hot encoding. Geometry is removed because it's an abstract object type.

train\_features = train.drop(columns=['price','state','geometry'])

test\_features = test.drop(columns=['price','state','geometry'])

"""## Machine Learning Algorithms"""

#All our machine learning algorithms have the same goal: use train\_features (all our interesting columns) to predict train\_target (price)

#We can gauge the performance of these models using the test\_features to predict test\_target, and analyzing the Mean Square Error or Mean Absolute Error

"""Linear Regression Model"""

from sklearn.linear\_model import LinearRegression

reg = LinearRegression()

reg.fit(train\_features, train\_target)

reg.coef\_

reg.intercept\_

data.describe()

from sklearn.metrics import mean\_squared\_error as MSE

reg\_pred = reg.predict(test\_features)

reg\_rmse = MSE(test\_target, reg\_pred) \*\* (1/2)

print("The root mean squared error:\t", reg\_rmse)

"""Decision Tree Regressor"""

from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(max\_depth=4,

min\_samples\_leaf=0.1,

random\_state=3)

dt.fit(train\_features, train\_target)

dt\_pred = dt.predict(test\_features)

dt\_rmse = MSE(test\_target, dt\_pred) \*\* (1/2)

print("The root mean squared error:\t", dt\_rmse)

axes = plt.gca()

axes.set\_xlim([0,8000])

axes.set\_ylim([500, 2000])

plt.scatter(test\_features['sqfeet'], test\_target, alpha=0.2, c='red', label="Test Data")

plt.scatter(test\_features['sqfeet'], dt\_pred, alpha=0.2, c='green', label="Prediction")

plt.xlabel("Square Feet")

plt.ylabel("Price ($)")

plt.legend()

plt.title("Price Prediction by Square Feet")

axes = plt.gca()

axes.set\_xlim([20,50])

axes.set\_ylim([500, 3000])

plt.scatter(test\_features['latitude'], test\_target, alpha=0.2, c='orange', label="Test Data")

plt.scatter(test\_features['latitude'], dt\_pred, alpha=0.2, c='green', label="Prediction")

plt.xlabel("Latitude")

plt.ylabel("Price ($)")

plt.legend()

plt.title("Price Prediction by Latitude")

axes = plt.gca()

axes.set\_xlim([0,8])

axes.set\_ylim([0, 6000])

plt.scatter(test\_features['baths'], test\_target, alpha=0.2, c='blue', label="Test Data")

plt.scatter(test\_features['baths'], dt\_pred, alpha=0.2, c='green', label="Prediction")

plt.xlabel("Baths")

plt.ylabel("Price ($)")

plt.legend()

plt.title("Price Prediction by Number of Baths")

axes = plt.gca()

axes.set\_xlim([0,8])

axes.set\_ylim([0, 6000])

plt.scatter(test\_features['beds'], test\_target, alpha=0.2, c='orange', label="Test Data")

plt.scatter(test\_features['beds'], dt\_pred, alpha=0.2, c='green', label="Prediction")

plt.xlabel("Beds")

plt.ylabel("Price ($)")

plt.legend()

plt.title("Price Prediction by Number of Beds")

"""Deep Neural Network"""

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

train\_scale = scaler.fit\_transform(train\_features)

test\_scale = scaler.transform(test\_features)

from tensorflow.keras.layers.experimental import preprocessing

normalizer = preprocessing.Normalization()

normalizer.adapt(np.array(train\_features))

import tensorflow as tf

from tensorflow import keras

dnn = keras.models.Sequential([

normalizer,

keras.layers.Dense(11, input\_dim=11, activation="relu", kernel\_initializer='normal'),

keras.layers.Dense(5, activation="relu", kernel\_initializer='normal'),

keras.layers.Dense(5, activation="relu", kernel\_initializer='normal'),

keras.layers.Dense(5, activation="relu", kernel\_initializer='normal'),

keras.layers.Dense(1, kernel\_initializer='normal')

])

dnn.compile(loss="mean\_squared\_error",

optimizer='adam')

history = dnn.fit(train\_scale, train\_target, epochs=10, validation\_data=(test\_scale, test\_target), verbose=1)