INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM)
VOLUME: 05 ISSUE: 07 | JULY - 2021 ISSN: 2582-3930

House Price Detection-TensorFlow

Sachit Mendiratta , Guide: Mr ML Sharma
Information Technology
Maharaja Agrasen Institute of Technology, Rohini, Sec 22
New Delhi, India

Abstract

Many students have already discovered the joy of learning programming using the Python programming language. Now it's time to take Python to the next level. This internship introduced Machine Learning with Data Science, concept to unify statistics, data analysis and their related methods. It's aimed at Computer Science students who want to learn techniques and theories drawn from many fields within the context of mathematics, statistics, computer science, domain knowledge and information science Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data. Data science practitioners apply machine learning algorithms to numbers, text, images, video, audio, and more to produce Artificial Intelligence systems to perform tasks that ordinarily require human intelligence. In turn, these systems generate insights which analysts and business users can translate into tangible business value. Python is used throughout, even for settings, files, and data models. Topics that will be covered during the workshop include: setup and configuration, template language, and database integration through object-relational mapping.

Keyword – TensorFlow, Neural Networks, Pandas, Matplotlib, NumPy

I. INTRODUCTION

House Price Detection is built using NumPy, Pandas, Scikit-Learn and Matplotlib, belonging to domain of Machine Learning and Data Science Concepts in order to detect house prices by training and deploying model given explanatory variables that cover many aspects of residential houses.

Real estate is known as one of the most important sectors of the economy. They contribute to balancing the economy of a country in as much as it boosts the income of people. It plays a crucial role in the lives of many especially those who own land. It gives space for the businesses to operate and it gives of a reputation that the country is doing well.

Provides Job Opportunities: Real estates are forerunners

when it comes to giving jobs to many people. From construction to utilization, real estate provides job to many. Such jobs during construction include the engineers, architects, laborer's while jobs available after construction greatly depends on how the real estate will be utilized.

Ecommerce The project can also be included with government of India in economy and housing sector which comprises of housing, retail, hospitality and commercial sector overall to help predict the house prices accurately and with efficiency

When deployed correctly, the scope of the House Price Predictor is extended to various Taxation domains like Direct tax, Indirect tax, Capital tax, Income tax and Sales tax.

II. MODULES AND LIBRARIES

A. Neural Network

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. A neural network evaluates price data and unearths opportunities for making trade decisions based on the data analysis. The networks can distinguish subtle nonlinear interdependencies and patterns other methods of technical analysis cannot. According to research, the accuracy of neural networks in making price predictions for stocks differs. Some models predict the correct stock prices 50 to 60 percent of the time while others are accurate in 70 percent of all instances. Some have posited that a 10 percent improvement in efficiency is all an investor can ask for from a neural network.

B. TensorFlow

TensorFlow is a very powerful and open-source library for implementing and deploying large-scale machine learning models. This makes it perfect for research and production. Over the years it has become one of the most popular libraries for deep learning.
The goal of this post is to build an intuition and understanding for how deep learning libraries work under the hood, specifically TensorFlow. To achieve this goal, we will mimic its API and implement its core building blocks from scratch. This has the neat little side effect that, by the end of this post, you will be able to use TensorFlow with confidence, because you'll have a deep conceptual

understanding of the inner workings. You will also gain further understanding of things like variables, tensors, sessions or operations. TensorFlow is a framework composed of two core building blocks — a library for defining computational graphs and a runtime for executing such graphs on a variety of different hardware. A computational graph has many advantages but more on that in just a moment. In a nutshell, a computational graph is an abstract way of describing computations as a directed graph. A directed graph is a data structure consisting of nodes (vertices) and edges. It's a set of vertices connected pairwise by directed edges. TensorFlow uses directed graphs internally to represent **computations,** and they call this **data flow graphs** (or computational graphs). While nodes in a directed graph can be anything, nodes in a computational graph mostly represent **operations**, **variables**, or **placeholders**.
Operations create or manipulate data according to specific rules. In TensorFlow those rules are called Ops, short for operations. Variables on the other hand represent shared, persistent state that can be manipulated by running Ops on those variables. The edges correspond to data, or multidimensional arrays (so-called Tensors) that flow through the different operations. In other words, edges carry information from one node to another. The output of one operation (one node) becomes the input to another operation and the edge connecting the two nodes carry the value.

C. Pandas

Pandas has been one of the most popular and favorite data science tools used in Python programming language for data wrangling and analysis. Data is unavoidably messy in real world. And Pandas is seriously a game changer when it comes to cleaning, transforming, manipulating and analyzing data. Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

D. Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "Pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter, since then it has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

Matplotlib 2.0.x supports Python versions 2.7 through 3.6. Python 3 support started with Matplotlib 1.2. Matplotlib 1.4 is the last version to support Python 2.6. Matplotlib has pledged not to support Python 2 past 2020 by signing the Python 3 Statement.

E. Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays .The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops, using NumPy. Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[18] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

F. Scikit-Learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

G. Scipy

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and SciLab. The NumPy stack is also sometimes referred to as the SciPy stack. SciPy is also a family of conferences for users and developers of these tools: SciPy (in the United

States), Euro SciPy (in Europe) and SciPy.in (in India).[6] Enthought originated the SciPy conference in the United States and continues to sponsor many of the international conferences as well as host the SciPy website.

The SciPy library is currently distributed under the BSD license, and its development is sponsored and supported by an open community of developers. It is also supported by Nymphicus, a community foundation for supporting reproducible and accessible science.

III. MACHINE LEARNING

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks .A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

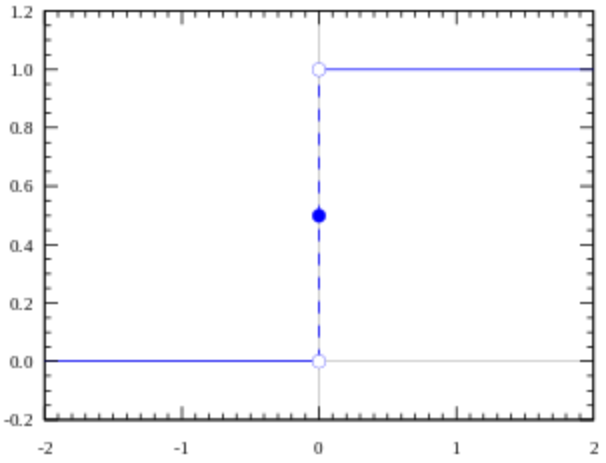
IV. DESIGN APPROACH

A. Activation Function:

An activation function is a very important feature of an artificial neural network, they basically decide whether the neuron should be activated or not. In artificial neural networks, the activation function defines the output of that node given an input or set of inputs. Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron. The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output.

B. The Step Function

The first thing that comes to our minds is how about a threshold-based activation function? If the value of Y is above a certain value, declare it activated. If it's less than the threshold, then say it's not. Hmm great. This could work!
Activation function A = "activated" if Y > threshold else not
Alternatively, A = 1 if y> threshold, 0 otherwise
Well, what we just did is a "step function", see the below figure.



Its output is 1 (activated) when value > 0 (threshold) and outputs a 0 (not activated) otherwise.
Great. So, this makes an activation function for a neuron. No confusions. However, there are certain drawbacks with this. To understand it better, think about the following. Suppose you are creating a binary classifier. Something which should say a "yes" or "no" (activate or not activate). A Step function could do that for you! That's exactly what it does, say a 1 or 0. Now, think about the use case where you would want multiple such neurons to be connected to bring in more classes. Class1, class2, class3 etc. What will happen if more than 1 neuron is "activated". All neurons will output a 1 (from step function). Now what would you decide? Which class is it? Hmm hard, complicated.
You would want the network to activate only 1 neuron and others should be 0 (only then would you be able to say it classified properly/identified the class). Ah! This is harder to train and converge this way. It would have been better if the activation was not binary and it instead would say "50% activated" or "20% activated" and so on. And then if more than 1 neuron activates, you could find which neuron has the "highest activation" and so on (better than max, a SoftMax, but let's leave that for now).
In this case as well, if more than 1 neuron says "100% activated", the problem still persists. I know! But. Since there are intermediate activation values for the output, learning can be smoother and easier (less wiggly) and chances of more than 1 neuron being 100% activated is lesser when compared to step function while training (also depending on what you are training and the data).
Ok, so we want something to give us intermediate (analogy) activation values rather than saying "activated" or not (binary).
The first thing that comes to our minds would be Linear function.

C. Linear Function

A = cx
A straight-line function where activation is proportional to input (which is the weighted sum from neuron). This way, it gives a range of activations, so it is not binary activation. We can definitely connect a few neurons together and if more

than 1 fires, we could take the max (or SoftMax) and decide based on that. So that is ok too. Then what is the problem with this? If you are familiar with gradient descent for training, you would notice that for this function, derivative is a constant.

A = cx, derivative with respect to x is c. That means, the gradient has no relationship with X. It is a constant gradient and the descent is going to be on constant gradient. If there is an error in prediction, the changes made by back propagation is constant and not depending on the change in input delta(x)

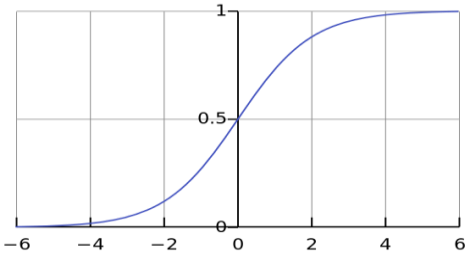
This is not that good! (Not always, but bear with me). There is another problem too. Think about connected layers. Each layer is activated by a linear function. That activation in turn goes into the next level as input and the second layer calculates weighted sum on that input and it in turn, fires based on another linear activation function.

No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer! Pause for a bit and think about it.

That means these two layers (or N layers) can be replaced by a single layer. Ah! We just lost the ability of stacking layers this way. No matter how we stack, the whole network is still equivalent to a single layer with linear activation (a combination of linear functions in a linear manner is still another linear function).

D. Sigmoid Function

$$A = \frac{1}{1+e^{-x}}$$



Well, this looks smooth and "step function like". What are the benefits of this? Think about it for a moment. First things first, it is nonlinear in nature. Combinations of this function are also nonlinear! Great. Now we can stack layers. What about non binary activations? Yes, that too! It will give an analogy activation unlike step function. It has a smooth gradient too.

And if you notice, between X values -2 to 2, Y values are very steep. Which means, any small changes in the values of X in that region will cause values of Y to change significantly. Ah, that means this function has a tendency to bring the Y values to either end of the curve.

Looks like it's good for a classifier considering its property? Yes! It indeed is. It tends to bring the activations to either side of the curve (above x = 2 and below x = -2 for example). Making clear distinctions on prediction.

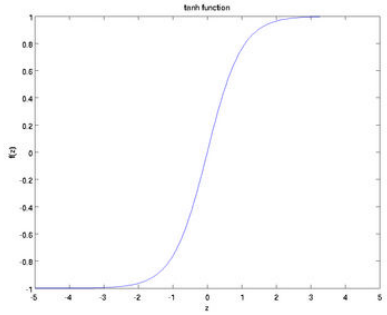
Another advantage of this activation function is, unlike linear function, the output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. So, we have our activations bound in a range. Nice, it won't blow up the activations then.

This is great. Sigmoid functions are one of the most widely used activation functions today. Then what are the problems with this? If you notice, towards either end of the sigmoid function, the Y values tend to respond very less to changes in X. What does that mean? The gradient at that region is going to be small. It gives rise to a problem of "vanishing gradients". Hmm. So, what happens when the activations reach near the "near-horizontal" part of the curve on either side?

Gradient is small or has vanished (cannot make significant change because of the extremely small value). The network refuses to learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits). There are ways to work around this problem and sigmoid is still very popular in classification problems.

E. Tanh Function

Another activation function that is used is the tanh function.



$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

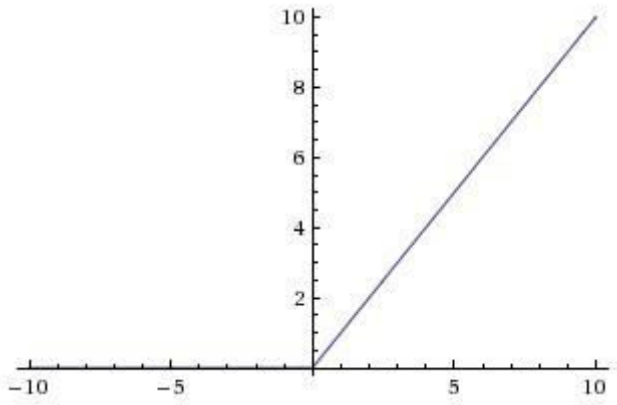
$$tanh(x) = 2\,sigmoid(2x) - 1$$

Ok, now this has characteristics similar to sigmoid that we discussed above. It is nonlinear in nature, so great we can stack layers! It is bound to range (-1, 1) so no worries of activations blowing up. One point to mention is that the gradient is stronger for tanh than sigmoid (derivatives are steeper). Deciding between the sigmoid or tanh will depend on your requirement of gradient strength. Like sigmoid, tanh also has the vanishing gradient problem.

F. Relu

A(x) = max (0, x)

The ReLu function is as shown above. It gives an output x if x is positive and 0 otherwise.



At first look this would look like having the same problems of linear function, as it is linear in positive axis. First of all, ReLu is nonlinear in nature. And combinations of ReLu are also nonlinear! (In fact, it is a good approximator. Any function can be approximated with combinations of ReLu). Great, so this means we can stack layers. It is not bound though. The range of ReLu is [0, inf). This means it can blow up the activation. Another point that I would like to discuss here is the sparsity of the activation. Imagine a big neural network with a lot of neurons. Using a sigmoid or tanh will cause almost all neurons to fire in an analog way (remember?). That means almost all activations will be processed to describe the output of a network. In other words, the activation is dense. This is costly. We would ideally want a few neurons in the network to not activate and thereby making the activations sparse and efficient.

ReLu give us this benefit. Imagine a network with random initialized weights (or normalized) and almost 50% of the network yields 0 activation because of the characteristic of ReLu (output 0 for negative values of x). This means fewer neurons are firing (sparse activation) and the network is lighter. Woah, nice! ReLu seems to be awesome! Yes, it is, but nothing is flawless. Not even ReLu.

Because of the horizontal line in ReLu (for negative X), the gradient can go towards 0. For activations in that region of ReLu, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). This is called dying ReLu problem. This problem can cause several neurons to just die and not respond making a substantial part of the network passive. There are variations in ReLu to mitigate this issue by simply making the horizontal line into non-horizontal component. for example, y = 0.01x for x<0 will make it a slightly inclined line rather than horizontal line. This is leaky ReLu. There are other variations too. The main idea is to let the gradient be non-zero and recover during training eventually.

ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

Now, which activation functions to use. Does that mean we just use ReLu for everything we do? Or sigmoid or tanh? Well, yes and no. When you know the function, you are trying to approximate has certain characteristics, you can choose an activation function which will approximate the function faster leading to faster training process. For example, a sigmoid works well for a classifier (see the graph of sigmoid, doesn't it show the properties of an ideal classifier?) because approximating a classifier function as combinations of sigmoid is easier than maybe ReLu, for example. Which will lead to faster training process and convergence. You can use your own custom functions too! If you don't know the nature of the function you are trying to learn, then maybe i would suggest start with ReLu, and then work backwards. ReLu works most of the time as a general approximator!

IV.METHODOLOGY & RESULTS

A. *Dataset*

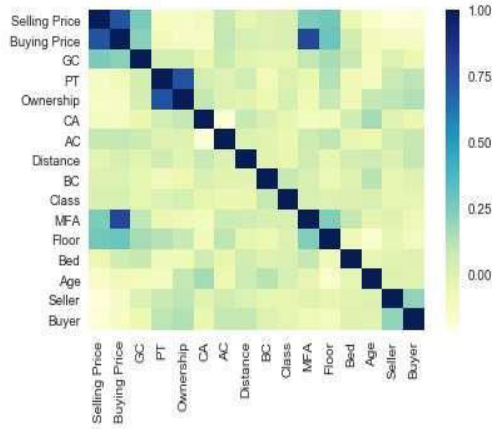
The dataset is a collection of housing prices in 2016 with some attributes or factor variables. As this paper uses machine learning predictions, these variables are called features. Table 2 shows the set of features to develop the prediction model. This study uses 19 attributes or features as independent variables for predicting house prices. Table 2:

Features	*Description*
CRIM	per capita crime rate by town
ZN	*proportion of residential land zoned for lots over 25,000 sq.ft.*
INDUS	*proportion of non-retail business acres per town*
CHAS	*Charles River dummy variable*
NOX	*nitric oxides concentration (parts per 10 million)*
RM	*average number of rooms per dwelling*
AGE	*proportion of owner-occupied units built prior to 1940*
DIS	*weighted distances to five Boston employment centres*
RAD	*index of accessibility to radial highways*
TAX	*full-value property-tax rate per $10,000*
PTRATIO	*pupil-teacher ratio by town*
LSTAT	*% lower status of the population*
MEDV	*Median value of owner-occupied homes in $1000's*

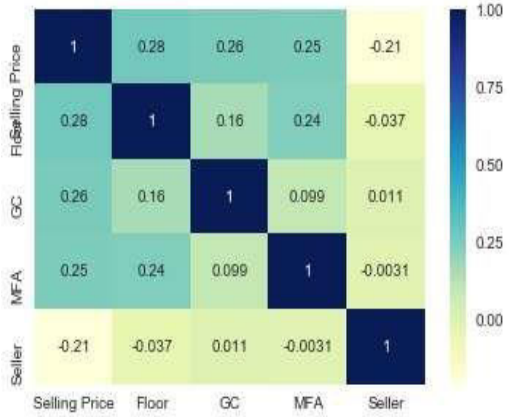
Features in the dataset

B. Features selection

Feature's selection is an important step of machine learning prediction. In this paper, features selection is divided into four groups. First group used all the independent parameters in the training dataset. It is a combination of variables with very weak, weak and strong relationships on the dependent variable sale price. In this paper, the level of relationship is defined as Strong if the coefficient correlation value is between 0.51 to 1.00 and moderate if the value is between 0.3 to 0.5. Otherwise, weak level is between 0.2 to 0.29 and very weak level is between 0.1 to 0.19. Fig. 1 shows the Python heatmap plot of all variables in the dataset.

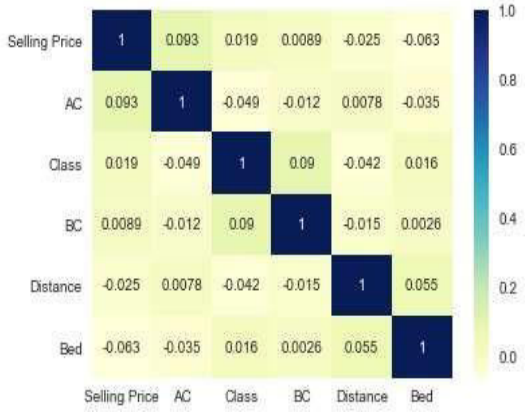


I. FIG. 1: CORRELATION LEVEL OF ALL FEATURESTHE FOLLOWING FIG. 2 SHOWS THE HEATMAP PLOT OF WEAK RELATIONSHIP VARIABLES WITH THE SELLING PRICES.



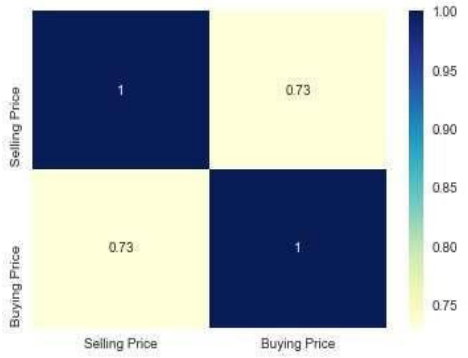
II. FIG. 2: CORRELATION LEVEL OF FEATURES WITH VALUES BETWEEN 0.20-0.29 (WEAK)

Subsequently, Fig. 3 shows another five variables with very weak relationship with the selling prices. These variables are:



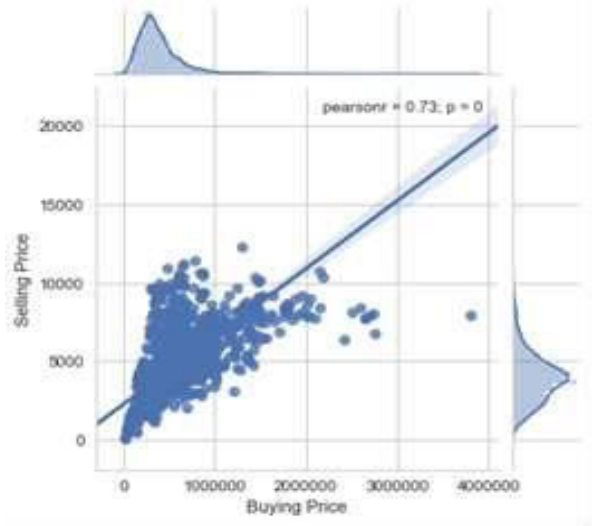
III. FIG. 3: CORRELATION LEVEL OF FEATURES WITH VALUES BETWEEN 0.1 TO 0.19 (VERY WEEK)

Only the buying price variable is found to have a strong relationship with the selling prices with coefficient value 0.73, as presented in the following Fig. 4.



IV. FIG. 4: THE CORRELATION LEVEL BETWEEN SELLING PRICE AND BUYING PRICE

It is interesting to observe the data distribution between the selling price and buying price, thus presented in the following Fig. 5.



V. FIG. 5: DATA DISTRIBUTION BETWEEN SELLING PRICES AND BUYING PRICES

V. RESULTS AND DISCUSSION

The Study on Neural Network Model used TensorFlow, NumPy, Pandas, SciPy, Scikit-Learn and Matplotlib involving Python and Data Science to predict accurate house prices was done successfully.

VI.CONCLUSION

This paper presents the reviews and findings of using neural network algorithms for real data of housing prices. The researchers demonstrate that feature selection is an important component of machine learning prediction. Two important performances of machine learning prediction are accuracy of prediction, and averaging of errors or fitness, which may be affected according to the feature's selection with different groups of relationship levels. However, these findings are limited to the tested dataset and therefore requires further investigations for different types of problems.

VI. ACKNOWLEDGMENT

I would like to extend my gratitude and my sincere thanks to my honourable, esteemed supervisor Mr ML Sharma. He is not only a great trainer with deep vision but also and most importantly knowledgeable and warm hearted as well. I sincerely thanks for his exemplary guidance and encouragement. His trust and support inspired me in the important moments of my making right decisions and I am glad to work with him.

VIII. REFERENCES

1. A. Marandi, A. Juuso, and E. Kazimir's, "Expert systems with applications fuzzy multiple criteria decision-making techniques and applications – Two decades review from 1994 to 2014," Expert Syst. Appl., 42(8), 2015, pp. 4126–4148.
2. F. Zahedi, "The analytic hierarchy process-A survey of the method and its applications," Interfaces, 16(4), 1986, pp. 96–108.
3. G. Magesh and P. Saratha, "Attribute reduction and cost optimization using machine learning methods to predict breast cancer," International Journal of Recent Technology and Engineering, 7(6), 2019, pp. 306–308.
4. C. Rajinikanth and S. A. Lincoln, "A semi supervised based Hyper Spectral Image (HSI) classification using machine learning approach," International Journal of Recent Technology and Engineering, 7(5S2), 2019, pp. 13–16.
5. S. Singh, M. Kaushik, A. Gupta, and A. K. Malviya, "Weather forecasting using machine learning techniques," SSRN Electron. J., 6, 2019, pp. 38–41.
6. S. Nasiriya and T. Deepa, "Dual Edge Classifier Based Support Vector Machine (DESVM) classifier for clinical dataset," International Journal of Recent Technology and Engineering, 7(6), 2019, pp. 331–338.
7. C. R. Rao and H. Totenberg, "Linear models," in Linear Models, New York: Springer, 1995, pp. 3–18.
8. A. Liwa, M. Wiener, "Classification and regression by random Forest," R News, 2(3), 2002, pp. 18–22.
9. S. Borden, A. Rane, G. Shende, and S. Shetty, "Real estate investment advising using machine learning," Int. Res. J. Eng. Technol., 4(3), 2017, pp. 1821–1825.
10. D. Diatonic and L. Sivan, "The multicollinearity illusion in moderated regression analysis," Mark. Lett., 27(2), 2016, pp. 403–4