Report on Soft Computing Techniques

Topic

Design & Implement Neural network models (Perceprton,SVM,LVQ,SOM) on Real Estate Valuation



Submitted to

Pooja Rana

Submitted by

Subhajit Payra

Reg. No. 11801422

GitHub Link: <https://github.com/subhajitpayra/MLProject.git>

**Abstract**

Humans and Animals have ability to continuously acquire and transfer knowledge throughout their lifespan & for this task only, we are known as intelligent systems. From mid 1950s, scientists like Warren McCulloch, Walter Pitts and D.O Hebb had opened the subject of computational neural network. Since then, many advancements had happened like creation of perceptron by Rosenblatt (1958), functional multilayer network were published by Lapa (1965) and continuous backpropagation using Dynamic Programming (1961). ANNs began as an attempt to duplicate the structure of human brain to perform tasks that conventional algorithms could do. ANN replicate the brain using artificial neurons, use learning and predict the result like what a human brain does. Nowadays, ANN is used in many industries for function approximation or regression analysis, data processing and classification purpose. Its ability to process nonlinear model is extensively used in System Identification and control (vehicle control, trajectory prediction), image processing, pattern and sequence recognition (handwriting) and 3D reconstruction. ANNs have been used to accelerate reliability analysis of infrastructures subject to natural disasters and to predict foundation settlements.

ANNs have been employed in cybersecurity, with the objective to discriminate between legitimate activities and malicious ones. For example, machine learning has been used for classifying Android malware, for identifying domains belonging to threat actors and for detecting URLs posing a security risk.

**Introduction**

Human brain is the most advanced and intelligent among all the creatures. It generally recognizes the pattern of everything and make some inferences based on it. Nowadays, Machines are also able to recognize the patterns in a dataset and approximate the results using different learning techniques.

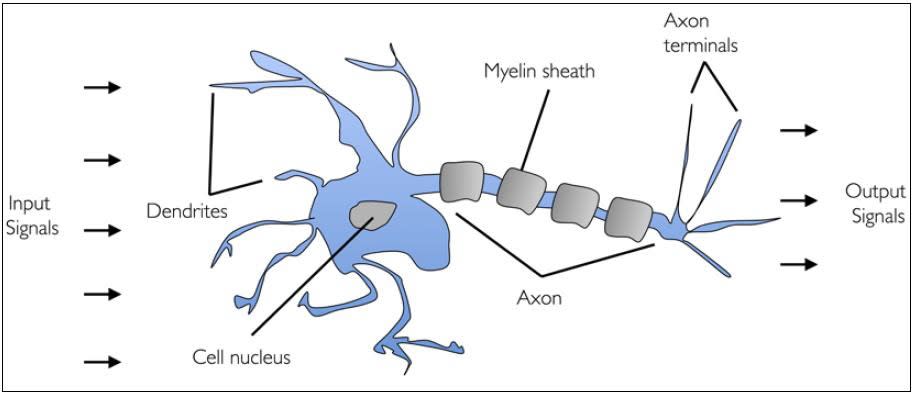
One of the best features of Machine Learning Algorithms are, they can be trained both on labelled (Supervised Learning) and unlabeled data (Unsupervised Learning). Unsupervised learning is a technique of model grouping the data without knowing what the actual objects are.

Neural Networks are highly efficient in classification (a supervised technique), clustering (unsupervised techniques). It can also be trained to process, identify and predict the image using Convolutional Neural Network. For classification and clustering, we have different algorithms in Neural Network which can determine the classes and clusters based on training data. From a structural perspective, ANNs can be divided into two main categories, feed-forward networks, in which the computation is performed in a layer-by-layer fashion from the input to the output of the network; and recurrent networks which have an interconnected network structure including cycles.

All these neural networks take a fixed amount of input and output data to be trained, in a fixed format (array, data frame etc.). Before the data is fed to network, weights are randomly initialized corresponding to each neuron. These weights are further updated based on the error (difference between actual output and obtained output) and these weights are again fed to neurons and iteration continues, to get the most appropriate results. Once the data is fed, our network is trained by iterative learning method. We use these ANN algorithms in different dataset to classify and clustering of data.

## **Biological Neuron**

A human brain has billions of neurons. Neurons are interconnected nerve cells in the human brain that are involved in processing and transmitting chemical and electrical signals. Dendrites are branches that receive information from other neurons.



Cell nucleus or Soma processes the information received from dendrites. Axon is a cable that is used by neurons to send information. Synapse is the connection between an axon and other neuron dendrites.

Let us discuss the rise of artificial neurons in the next section.

What are you waiting for? Interested in taking up a Deep Learning Course?

## **Rise of Artificial Neurons**

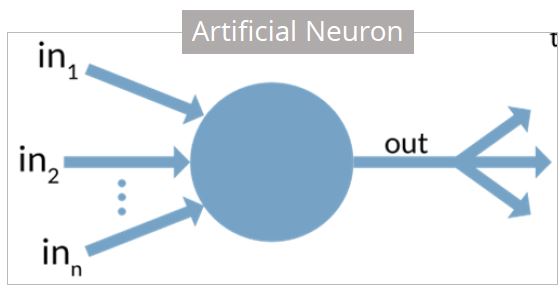
Researchers Warren McCullock and Walter Pitts published their first concept of simplified brain cell in 1943. This was called McCullock-Pitts (MCP) neuron. They described such a nerve cell as a simple logic gate with binary outputs.

Multiple signals arrive at the dendrites and are then integrated into the cell body, and, if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

In the next section, let us talk about the artificial neuron.

## **Artificial Neuron**

An artificial neuron is a mathematical function based on a model of biological neurons, where each neuron takes inputs, weighs them separately, sums them up and passes this sum through a nonlinear function to produce output.



In the next section, let us compare the biological neuron with the artificial neuron.

## **Biological Neuron vs. Artificial Neuron**

The biological neuron is analogous to artificial neurons in the following terms:

|  |  |
| --- | --- |
| **Biological Neuron** | **Artificial Neuron** |
| Cell Nucleus (Soma) | Node |
| Dendrites | Input |
| Synapse | Weights or interconnections |
| Axon | Output |

## **Artificial Neuron at a Glance**

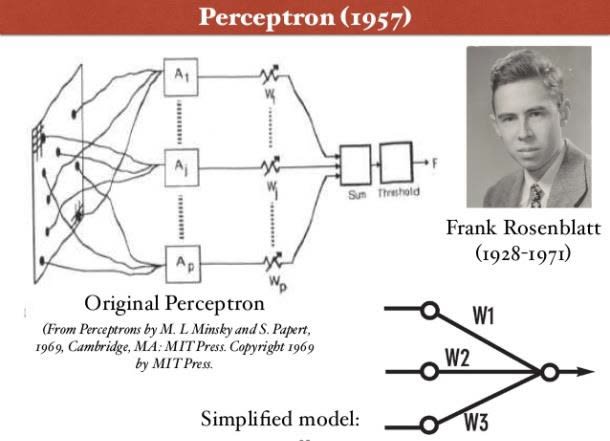
The artificial neuron has the following characteristics:

* A neuron is a mathematical function modeled on the working of biological neurons
* It is an elementary unit in an artificial neural network
* One or more inputs are separately weighted
* Inputs are summed and passed through a nonlinear function to produce output
* Every neuron holds an internal state called activation signal
* Each connection link carries information about the input signal
* Every neuron is connected to another neuron via connection link

In the next section, let us talk about perceptron.

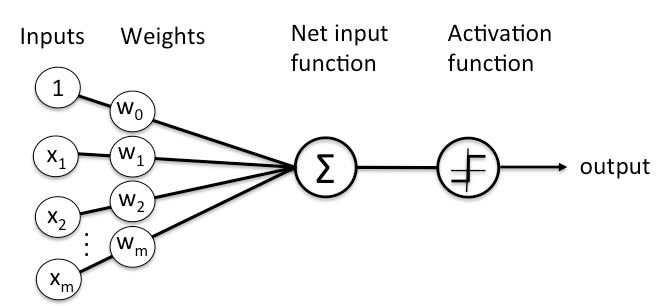
## **Perceptron**

A perceptron is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data.



Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron.

A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.

There are two types of Perceptrons: Single layer and Multilayer.

Single layer Perceptrons can learn only linearly separable patterns.

Multilayer Perceptrons or feedforward neural networks with two or more layers have the greater processing power.

The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.

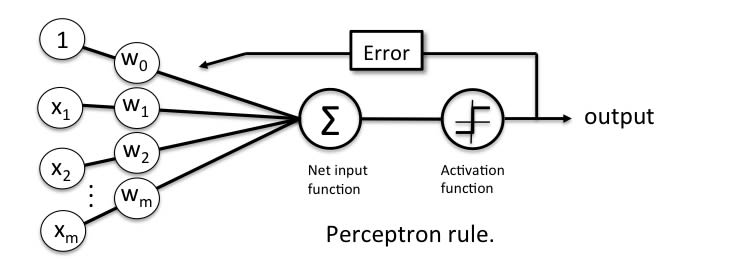
This enables you to distinguish between the two linearly separable classes +1 and -1.

Note: Supervised Learning is a type of Machine Learning used to learn models from labeled training data. It enables output prediction for future or unseen data.

Let us focus on the Perceptron Learning Rule in the next section.

## **Perceptron Learning Rule**

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.

The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output. In the context of supervised learning and classification, this can then be used to predict the class of a sample.

In the next section, let us focus on the perceptron function.

## **Perceptron Function**

Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value ”f(x)”is generated.



In the equation given above:

“w” = vector of real-valued weights

“b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)

“x” = vector of input x values



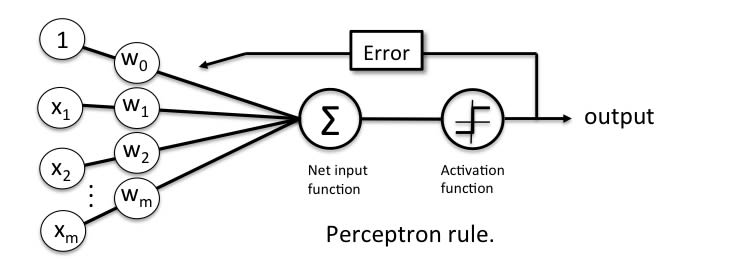
“m” = number of inputs to the Perceptron

The output can be represented as “1” or “0.”  It can also be represented as “1” or “-1” depending on which activation function is used.

Let us learn the inputs of a perceptron in the next section.

## **Inputs of a Perceptron**

A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result. The above below shows a Perceptron with a Boolean output.



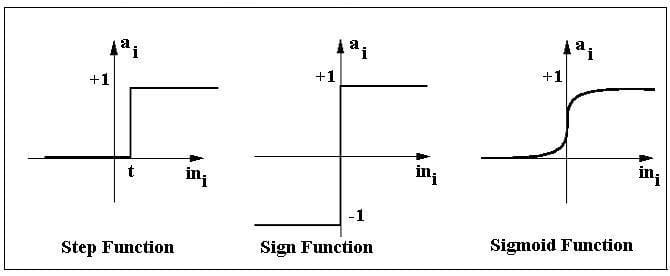
A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function “∑” multiplies all inputs of “x” by weights “w” and then adds them up as follows:



In the next section, let us discuss the activation functions of perceptron.

## **Activation Functions of Perceptron**

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.

For example:

If ∑ wixi> 0 => then final output “o” = 1 (issue bank loan)

Else, final output “o” = -1 (deny bank loan)

Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

## **Output of Perceptron**

Perceptron with a Boolean output:

Inputs: x1…xn

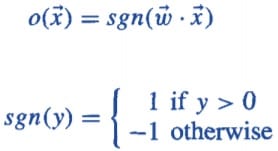
Output: o(x1….xn)



Weights: wi=> contribution of input xi to the Perceptron output;

w0=> bias or threshold

If ∑w.x > 0, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.



An output of +1 specifies that the neuron is triggered. An output of -1 specifies that the neuron did not get triggered.

“sgn” stands for sign function with output +1 or -1.

Want to check the Course Preview of Deep Learing? [Click here to watch!](https://www.simplilearn.com/deep-learning-course-with-tensorflow-training?source=GhPreviewCTAText#/course-preview)

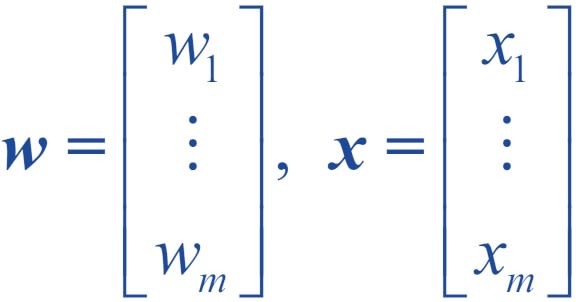
## **Error in Perceptron**

In the Perceptron Learning Rule, the predicted output is compared with the known output. If it does not match, the error is propagated backward to allow weight adjustment to happen.

Let us discuss the decision function of Perceptron in the next section.

## **Perceptron: Decision Function**

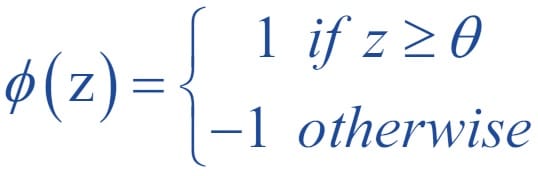
A decision function φ(z) of Perceptron is defined to take a linear combination of x and w vectors.



The value z in the decision function is given by:



The decision function is +1 if z is greater than a threshold θ, and it is -1 otherwise.



This is the Perceptron algorithm.

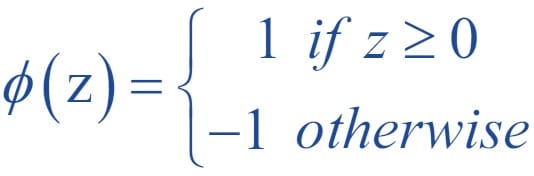
**Bias Unit**

For simplicity, the threshold θ can be brought to the left and represented as w0x0, where w0= -θ and x0= 1.



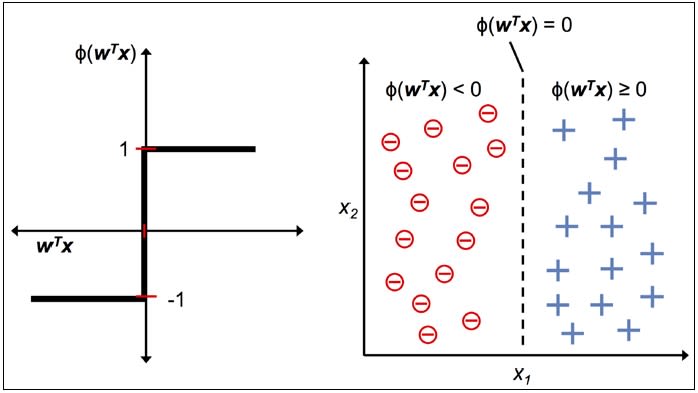
The value w0  is called the bias unit.

The decision function then becomes:



**Output**

The figure shows how the decision function squashes wTx to either +1 or -1 and how it can be used to discriminate between two linearly separable classes.



## **Perceptron at a Glance**

Perceptron has the following characteristics:

* Perceptron is an algorithm for Supervised Learning of single layer binary linear classifier.
* Optimal weight coefficients are automatically learned.
* Weights are multiplied with the input features and decision is made if the neuron is fired or not.
* Activation function applies a step rule to check if the output of the weighting function is greater than zero.
* Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1.
* If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.

Types of activation functions include the sign, step, and sigmoid functions.

What is Support Vector Machine?

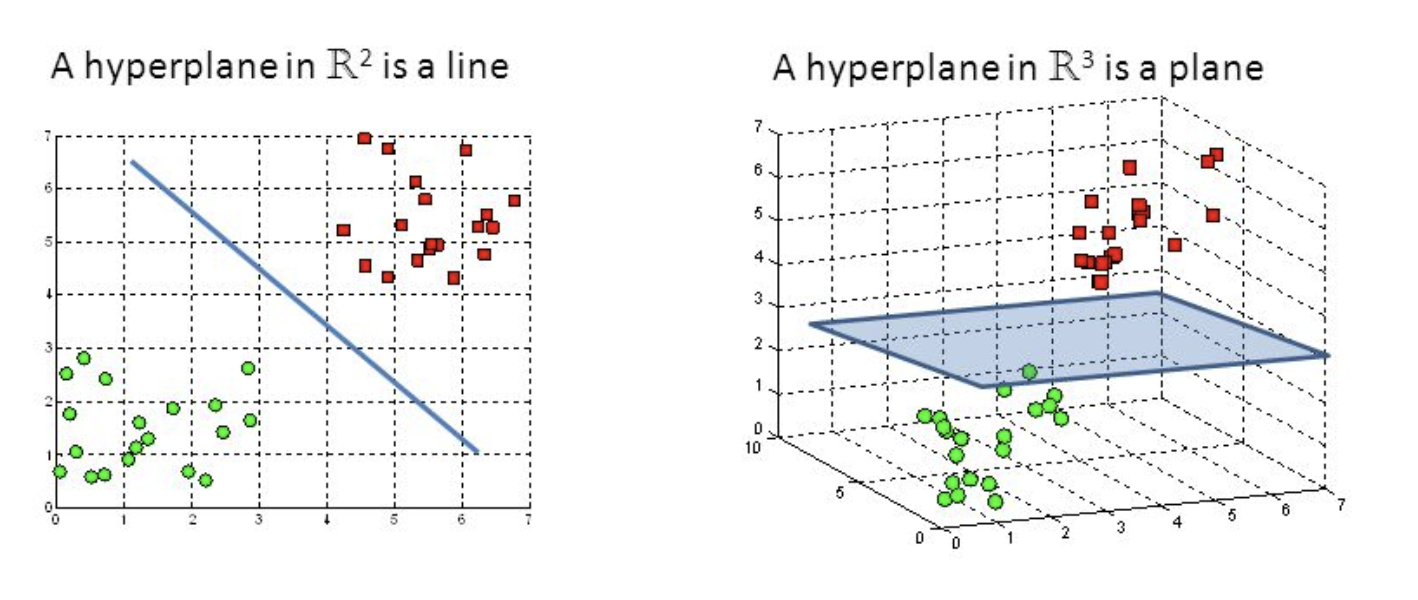
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes and Support Vectors



Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support Vectors

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

**Dataset**

Dataset plays the most important role in any machine learning algorithms. Just like humans needs their eyes and sense organs to observe the things happening around and recognizing the pattern, similarly dataset is fed to networks and machine recognizes the pattern.

**Limitations of Artificial Neural Network**

* **Black Box Nature:** When it comes to output then Neural Network predicts the output with a great accuracy but the process going in networks (hidden layers) are still unclear. We are still not able to understand by which means it is concluding or inference.
* **Duration of development:** Although libraries like ‘keras’ and ‘tensorflow’ makes the computational task a lot easier still there exists a major drawback in duration required by neural network to build and train the data. It’s easier to work on the task which has already been done but it’s a lot difficult to make efficient algorithms when we are solving a new problem in machine learning.
* **Amount of Data:** ANN require a very large data set to train our model and obtain great accuracy in results. With few algorithms like Naïve Bayes, network works well with small dataset too. But, in most of the cases, we require a good amount of labelled/unlabeled data for building and training our model.
* **Computationally Expensive:** Neural networks are also more computationally expensive than traditional algorithms. As neural networks require much larger dataset, it also require more time to train the data and similarly takes larger time for computations.

**Proposed Methodology**

For classification and clustering of Real Estate dataset, I have implemented different neural networks algorithms like Perceptron [Figure 1], MLP using Backpropagation, SVM [Figure 2], SOM [Figure 3] and LVQ.

Data is given to the input of neural network then neurons adjust weights corresponding to every neuron (if any error is identified), the most appropriate result is taken as output after comparing it to the target dataset.

In Supervised learning like Perceptron, LVQ etc. algorithms continue with weight update.

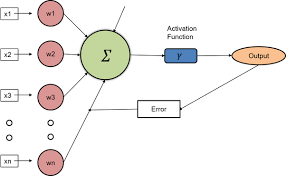


Figure 1: Perceptron Model Figure 2: Support Vector Machine

Diagram

Description automatically generated

Figure 3: Self Organizing Map (Unsupervised)

**Results and Discussion**

Real Estate dataset is trained using different neural network techniques.I have used libraries like keras, tensorflow, sklearn and Minisom to implement MLP using Backpropagation and Self-Organizing Maps.

Also, for Support Vector Machine and Perceptron, I have used svm and sklearn library.

In classification using MLP (backpropagation), the accuracy obtained was very low because it requires a large dataset and, in our case, dataset contains just 498 entries. Accuracy occurred using perceptron is also quite low(15%) because of the small dataset and since the values (number of reviews published) are very variable.

In Support Vector Machine, I obtained a 60% accuracy on testing data. The model worked so perfect because I have made data linear separable using kernel trick. Also, I have obtained the support vector indices in order to track those vectors lying on the classifier boundary.

In Self-Organizing map and Linear Vector Quantization, weight are getting updated and learning rate parameter decreases to 80% after every iteration. The results obtained using these two algorithms are also accurate.

**References**

[1] S. Renjith and C. Anjali, ‘A personalized mobile travel recommender system using hybrid algorithm,’ 2014 First International Conference on Computational Systems and Communications (ICCSC), Trivandrum, 2014, pp. 12–17. doi: 10.1109/COMPSC.2014.7032612

[2] Feature Discovery by Competitive Learning, D.E. Rumelhalt and D. Zipser – 1985 Cognitive Science, 1985, 9, 75-112, 1985

[3] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. **5** (4): 115–133. [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier)):[*10.1007/BF02478259*](https://doi.org/10.1007%2FBF02478259)

[4] Hebb, Donald (1949). [*The Organization of Behavior*](https://books.google.com/books?id=ddB4AgAAQBAJ). New York: Wiley. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [978-1-135-63190-1](https://en.wikipedia.org/wiki/Special:BookSources/978-1-135-63190-1)

[5] Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization In The Brain". *Psychological Review*. **65** (6): 386–408

[6] J. Weng, N. Ahuja and T. S. Huang, "[Cresceptron: a self-organizing neural network which grows adaptively](http://www.cse.msu.edu/~weng/research/CresceptronIJCNN1992.pdf)," *Proc. International Joint Conference on Neural Networks*, Baltimore, Maryland, vol I, pp. 576–581, June, 1992.

[7] J. Weng, N. Ahuja and T. S. Huang, "[Learning recognition and segmentation of 3-D objects from 2-D images](http://www.cse.msu.edu/~weng/research/CresceptronICCV1993.pdf)," *Proc. 4th International Conf. Computer Vision*, Berlin, Germany, pp. 121–128, May, 1993.

[8] J. Weng, N. Ahuja and T. S. Huang, "[Learning recognition and segmentation using the Cresceptron](http://www.cse.msu.edu/~weng/research/CresceptronIJCV.pdf)," *International Journal of Computer Vision*, vol. 25, no. 2, pp. 105–139, Nov. 1997

[9] Derrick Mwiti, “Introduction to Self-Organizing Maps”, heartbeat.com, July 2018

[10] Y.-S. Park, S. Lek [Ecological Model Types](https://www.sciencedirect.com/science/article/pii/B9780444636232000074) in [Developments in Environmental Modelling](https://www.sciencedirect.com/science/bookseries/01678892), 2016

[11] C. Gajendran, P. Thamarai, “Intelligent Prediction Modelling of Water Quality using ANN” in [GIS and Geostatistical Techniques for Groundwater Science](https://www.sciencedirect.com/book/9780128154137), Tamilnadu, 2019

[12] Studies from Department of Computer Science & Engineering in the Area of Artificial Neural Networks Described (A Technique to Stock Market Prediction Using Fuzzy Clustering and Artificial Neural Networks), Atlanta, 2015

[13] Emny Harna Yossy in “Mango Fruit Sortation System using NN and Computer Vision” et al. 2017

[14] Koklu, Murat, Tutuncu, K PY - 2019/11/15, SP – 012032, T1 - Tree based classification methods for occupancy detection, VL – 675, DO - 10.1088/1757-899X/675/1/012032, IOP Conference Series: Materials Science and Engineering

[15] Jay Shah, Neural Networks for Beginners: Popular Types and Applications, ‘blogs.statsbot.com’, Nov 2017

CODE

QUESTION

Design & Implement Neural network models (Perceprton,SVM,LVQ,SOM) on Real Estate Valuation

https://github.com/subhajitpayra/MLProject.git

#####################################################################################################################

#####################################################################################################################

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression, Lasso, Ridge

from sklearn.neighbors import KNeighborsRegressor

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score, cross\_val\_predict, KFold

import xgboost as xgb

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.pipeline import Pipeline

plt.style.use('seaborn')

file = pd.read\_csv('Realestate.csv', sep=';')

df = pd.DataFrame(file)

def preprocessing(data):

#Do first stage pre-processing (i.e. exclude non-numeric prices and convert prices to numeric variables)

data = data[data.price\_new.str.contains("aanvraag") == False].dropna()

data[['price\_new']] = data[['price\_new']].apply(pd.to\_numeric)

# exclude everything with a price above or below 3 standard deviations (i.e. outliers)

data = data[np.abs(data["price\_new"]-data["price\_new"].mean())<=(3\*data["price\_new"].std())]

# Set x and y (dropping zipcode and rooms as latitude, longitude and surface pretty much capture the former)

y = data.price\_new

X = data.drop('price\_new', axis = 1).drop('zipcode\_new', axis = 1).drop('rooms\_new', axis = 1)

return X, y

def model(pipeline, parameters, X\_train, y\_train, X, y):

grid\_obj = GridSearchCV(estimator=pipeline,

param\_grid=parameters,

cv=3,

scoring='r2',

verbose=2,

n\_jobs=1,

refit=True)

grid\_obj.fit(X\_train, y\_train)

'''Results'''

results = pd.DataFrame(pd.DataFrame(grid\_obj.cv\_results\_))

results\_sorted = results.sort\_values(by=['mean\_test\_score'], ascending=False)

print("##### Results")

print(results\_sorted)

print("best\_index", grid\_obj.best\_index\_)

print("best\_score", grid\_obj.best\_score\_)

print("best\_params", grid\_obj.best\_params\_)

'''Cross Validation'''

estimator = grid\_obj.best\_estimator\_

'''

if estimator.named\_steps['scl'] == True:

X = (X - X.mean()) / (X.std())

y = (y - y.mean()) / (y.std())

'''

shuffle = KFold(n\_splits=5,

shuffle=True,

random\_state=0)

cv\_scores = cross\_val\_score(estimator,

X,

y.values.ravel(),

cv=shuffle,

scoring='r2')

print("##### CV Results")

print("mean\_score", cv\_scores.mean())

'''Show model coefficients or feature importances'''

try:

print("Model coefficients: ", list(zip(list(X), estimator.named\_steps['clf'].coef\_)))

except:

print("Model does not support model coefficients")

try:

print("Feature importances: ", list(zip(list(X), estimator.named\_steps['clf'].feature\_importances\_)))

except:

print("Model does not support feature importances")

'''Predict along CV and plot y vs. y\_predicted in scatter'''

y\_pred = cross\_val\_predict(estimator, X, y, cv=shuffle)

plt.scatter(y, y\_pred)

xmin, xmax = plt.xlim()

ymin, ymax = plt.ylim()

plt.plot([xmin, xmax], [ymin, ymax], "g--", lw=1, alpha=0.4)

plt.xlabel("True prices")

plt.ylabel("Predicted prices")

plt.annotate(' R-squared CV = {}'.format(round(float(cv\_scores.mean()), 3)), size=9,

xy=(xmin,ymax), xytext=(10, -15), textcoords='offset points')

plt.annotate(grid\_obj.best\_params\_, size=9,

xy=(xmin, ymax), xytext=(10, -35), textcoords='offset points', wrap=True)

plt.title('Predicted prices (EUR) vs. True prices (EUR)')

plt.show()

# Pipeline and Parameters - MLP Regression

pipe\_neural = Pipeline([('scl', StandardScaler()),

('clf', MLPRegressor())])

param\_neural = {'clf\_\_alpha': [0.001, 0.01, 0.1, 1, 10, 100],

'clf\_\_hidden\_layer\_sizes': [(5),(10,10),(7,7,7)],

'clf\_\_solver': ['lbfgs'],

'clf\_\_activation': ['relu', 'tanh'],

'clf\_\_learning\_rate' : ['constant', 'invscaling']}

# Execute preprocessing & train/test split

X, y = preprocessing(df)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=0)

# Execute model hyperparameter tuning and crossvalidation

model(pipe\_neural, param\_neural, X\_train, y\_train, X, y)

#############################################################################################

#############################################################################################

#############################################################################################

import pandas as pd

data = pd.read\_csv("Realestate.csv")

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

training\_set, test\_set = train\_test\_split(data, test\_size = 0.2, random\_state = 1)

X\_train = training\_set.iloc[:,0:2].values

Y\_train = training\_set.iloc[:,2].values

X\_test = test\_set.iloc[:,0:2].values

Y\_test = test\_set.iloc[:,2].values

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random\_state = 1)

classifier.fit(X\_train,Y\_train)

Y\_pred = classifier.predict(X\_test)

test\_set["Predictions"] = Y\_pred

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(Y\_test,Y\_pred)

accuracy = float(cm.diagonal().sum())/len(Y\_test)

print("\nAccuracy Of SVM For The Given Dataset : ", accuracy)

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

Y\_train = le.fit\_transform(Y\_train)

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random\_state = 1)

classifier.fit(X\_train,Y\_train

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

plt.figure(figsize = (7,7))

X\_set, y\_set = X\_train, Y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('black', 'white')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'orange'))(i), label = j)

plt.title('Apples Vs Oranges')

plt.xlabel('Weight In Grams')

plt.ylabel('Size in cm')

plt.legend()

plt.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

plt.figure(figsize = (7,7))

X\_set, y\_set = X\_test, Y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('black', 'white')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],c = ListedColormap(('red', 'orange'))(i), label = j)

plt.title('Apples Vs Oranges Predictions')

plt.xlabel('Weight In Grams')

plt.ylabel('Size in cm')

plt.legend()

plt.show()

#######################################################################################################################

#######################################################################################################################

#######################################################################################################################

# LVQ for the Ionosphere Dataset

from random import seed

from random import randrange

from csv import reader

from math import sqrt

# Load a CSV file

def load\_csv(filename):

dataset = list()

with open(filename, 'r') as file:

csv\_reader = reader(file)

for row in csv\_reader:

if not row:

continue

dataset.append(row)

return dataset

# Convert string column to float

def str\_column\_to\_float(dataset, column):

for row in dataset:

row[column] = float(row[column].strip())

# Convert string column to integer

def str\_column\_to\_int(dataset, column):

class\_values = [row[column] for row in dataset]

unique = set(class\_values)

lookup = dict()

for i, value in enumerate(unique):

lookup[value] = i

for row in dataset:

row[column] = lookup[row[column]]

return lookup

# Split a dataset into k folds

def cross\_validation\_split(dataset, n\_folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / n\_folds)

for i in range(n\_folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

# Calculate accuracy percentage

def accuracy\_metric(actual, predicted):

correct = 0

for i in range(len(actual)):

if actual[i] == predicted[i]:

correct += 1

return correct / float(len(actual)) \* 100.0

# Evaluate an algorithm using a cross validation split

def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

folds = cross\_validation\_split(dataset, n\_folds)

scores = list()

for fold in folds:

train\_set = list(folds)

train\_set.remove(fold)

train\_set = sum(train\_set, [])

test\_set = list()

for row in fold:

row\_copy = list(row)

test\_set.append(row\_copy)

row\_copy[-1] = None

predicted = algorithm(train\_set, test\_set, \*args)

actual = [row[-1] for row in fold]

accuracy = accuracy\_metric(actual, predicted)

scores.append(accuracy)

return scores

# calculate the Euclidean distance between two vectors

def euclidean\_distance(row1, row2):

distance = 0.0

for i in range(len(row1)-1):

distance += (row1[i] - row2[i])\*\*2

return sqrt(distance)

# Locate the best matching unit

def get\_best\_matching\_unit(codebooks, test\_row):

distances = list()

for codebook in codebooks:

dist = euclidean\_distance(codebook, test\_row)

distances.append((codebook, dist))

distances.sort(key=lambda tup: tup[1])

return distances[0][0]

# Make a prediction with codebook vectors

def predict(codebooks, test\_row):

bmu = get\_best\_matching\_unit(codebooks, test\_row)

return bmu[-1]

# Create a random codebook vector

def random\_codebook(train):

n\_records = len(train)

n\_features = len(train[0])

codebook = [train[randrange(n\_records)][i] for i in range(n\_features)]

return codebook

# Train a set of codebook vectors

def train\_codebooks(train, n\_codebooks, lrate, epochs):

codebooks = [random\_codebook(train) for i in range(n\_codebooks)]

for epoch in range(epochs):

rate = lrate \* (1.0-(epoch/float(epochs)))

for row in train:

bmu = get\_best\_matching\_unit(codebooks, row)

for i in range(len(row)-1):

error = row[i] - bmu[i]

if bmu[-1] == row[-1]:

bmu[i] += rate \* error

else:

bmu[i] -= rate \* error

return codebooks

# LVQ Algorithm

def learning\_vector\_quantization(train, test, n\_codebooks, lrate, epochs):

codebooks = train\_codebooks(train, n\_codebooks, lrate, epochs)

predictions = list()

for row in test:

output = predict(codebooks, row)

predictions.append(output)

return(predictions)

# Test LVQ on dataset

seed(1)

# load and prepare data

filename = 'Realestate.csv'

dataset = load\_csv(filename)

for i in range(len(dataset[0])-1):

str\_column\_to\_float(dataset, i)

# convert class column to integers

str\_column\_to\_int(dataset, len(dataset[0])-1)

# evaluate algorithm

n\_folds = 50

learn\_rate = 0.001

n\_epochs = 500

n\_codebooks = 200

scores = evaluate\_algorithm(dataset, learning\_vector\_quantization, n\_folds, n\_codebooks, learn\_rate, n\_epochs)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

#########################################################################################

##########################################################################################

###########################################################################################

#As usual we start importing a number of libraries that will be come in handy later on

import numpy as np

import pandas as pd

import seaborn as sns

from imageio import imwrite

#from scipy.misc import imsave

import matplotlib

import matplotlib.pyplot as plt

from matplotlib.offsetbox import OffsetImage, AnnotationBbox

from matplotlib.cbook import get\_sample\_data

%matplotlib inline

import plotly.offline as py

py.init\_notebook\_mode(connected=True)

import plotly.graph\_objs as go

import plotly.tools as tls

from sklearn.preprocessing import StandardScaler

from PIL import Image, ImageChops

np.random.seed(0)

#We load here the data from the provided training set, we randomly select 500 landmark points and separate the labels.

train = pd.read\_csv('Realestate.csv')

train = train.sample(n=500, random\_state=0)

labels = train['label']

train = train.drop("label",axis=1)

#Let's plot the distribution and see if the distribution is uniform

sns.distplot(labels.values,bins=np.arange(-0.5,10.5,1))

#Then we normalize the data, a crucial step to the correct functioning of the SOM algorithm

trainSt = StandardScaler().fit\_transform(train.values)