The usage of Least Squares in Machine Learning

MadeByHumans.ai research
Author: Luis Román

Fall 2021

Abstract— In this paper I investigate the usage of least squares in machine learning. The primary question: are there any algorithms used in machine learning today that use least squares, will be answered. The aim of this paper is to get insight in the usage of least squares in the field of machine learning. I give an answer to the secondary questions; What is machine learning? What algorithms are used in machine learning? What different types of learning do exist? The method used is an investigation on the usage of least squares in traditional machine learning algorithms. By answering the secondary questions, I get closer to answering the primary question. I investigate the objective function used by the algorithms to determine if there is any usage of least squares. There is an overview of the used algorithms used in machine learning, and I show that least squares as an objective function is only used in linear regression.

List of Diagrams

Table of Abbreviations

Abbreviation	**Explanation**
SVM	Support Vector Machine
kNN	k-Nearest Neighbors
MSE	Mean Squared Error

1. Introduction

The rationale for choosing the usage of least squares in machine learning as a topic for this paper is twofold.

First, least squares is one of the oldest and most popular methods for fitting data. The method dates to 1805 and was officially discovered and published by Adrien-Marie Legendre (Merriman, 1877).

Second, “the least squares method is one of the most fundamental methods in Statistics to estimate correlations among various data.” (Fujii, 2018, p. 1).

Since machine learning is quite a modern field, in that sense, it gained lots of popularity the last decade. It’s interesting to see if such a traditional method as the least squares is used in the more modern field of machine learning today.

We think above reasons and the central question; are there any algorithms used in machine learning today that use least squares? Make an interesting subject for this paper.

With this paper the aim is to get an insight in the usage of least squares in the field of machine learning.

To narrow down the scope of this paper we will only address the fundamental algorithms used in traditional machine learning. With traditional machine learning we mean; we won’t be addressing deep learning.

As stated, the central question we can ask ourselves is, are there any algorithms used in machine learning today that use least squares. Other questions that come to mind are. What is machine learning? What algorithms are used in machine learning? What different types of learning do exist? After answering these questions, we will conclude this paper with the answer on the central question.

2. What is machine Learning

“Machine Learning is a subfield of computer science that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon” (Burkov, 2019, p. 1). Based on examples of phenomena in the natural world we let machines learn. In the next section we will address the types of learning machines can use to learn.

2.1 Types of Learning

There are different types of learning defined in machine learning. Supervised, unsupervised, semi-supervised and reinforcement learning. What follows is a short description of each type.

In **supervised learning**, like the name already states, humans supervise the learning. Humans will label the examples of some phenomenon, which then will be fed to an algorithm. For instance, if we have a list of houses, the price of an individual house can be called a feature. In supervised learning these features typically are labeled by humans.

Unsupervised learning contains a collection of unlabeled examples (Burkov, 2019). In contrast to supervised learning, the algorithm auto-discovers the labels of the dataset.

Semi-supervised learning is a combination of the two methods stated earlier. It contains both labeled and unlabeled examples (Burkov, 2019).
Humans label a part of the dataset, and the rest of the dataset is also auto discovered by the algorithm.

According to (Burkov, 2019) **reinforcement learning** is a subfield of machine learning, where the machine lives in a defined environment and can perceive this environment.
With the use of policies, the algorithm gets rewards and tries to optimize the behavior by maximizing the rewards it gets (Burkov, 2019).

Above learning methods are the main types of learning in the traditional sense of machine learning. There exist other methods of learning like deep learning with neural nets, although this is out of the scope of this paper, we thought it was worth mentioning it.

We have given answer to the question. What different types of learning do exist? After addressing the types of learning now it’s time address the fundamental algorithms used in machine learning in the next section.

2.2 Fundamental algorithms

In the previous section we have seen what different types of learning are used in machine learning. Now, to get closer to answering our central question, which is. What algorithms used in machine learning make use of least squares? We first need to answer the following question. What algorithms are the fundamental algorithms used in machine learning. This is what the next section will cover.

2.3 Linear Regression

In the real world we try to find a linear relationship between two or more variables (Miller, n.d.). For example, the temperature of the day in Celsius degrees, and the number of visitors to a certain theme park. We may want to know if the number of visitors change when the temperature changes. Or, if I know the temperature of tomorrow, how many visitors can I expect? This is where linear regression comes in handy.

According to (Miller, n.d.) the basic problem is to find the best fit straight-line $y = ax + b$ given that, for $n \in 1, \ldots, N$, the pairs (xn, yn) are observed. With best fit is meant, the line that is the closest to all observed variables.

In linear regression there can be a positive relationship or a negative relationship between variables. When there is a negative relationship, the line pointing downwards, the plus sign in the equation changes into a minus sign.

This is the formula for linear regression:

$$y = ax + b$$

Where, y is the dependent variable, in our example the number of visitors. a is the slope, x is the independent variable, in our example the temperature and b is the y-intercept. To fill in the equation we need to calculate the slope of the line and we need to calculate the y-intercept of the line.

To calculate the slope:

$$a = \frac{n\left(\sum xy\right) - \left(\sum x\right)\left(\sum y\right)}{n\left(\sum x^2\right) - \left(\sum x\right)^2}$$

To calculate the y-intercept:

$$b = y - ax$$

Having calculated these values, we can fill in the formula for linear regression, when new we do this by hand. In Python we can use the function linear regression from the package Scikit Learn or code the formula by hand.
The objective of this model is to minimize the residual, that is the distance between a ground truth value y and the predicted value y. The residual is sometimes called the squared error loss or mean squared error (MSE).

We went over the main points in linear regression and have seen the real-world usability of this method. In the next section, we will talk about another important algorithm used in machine learning called logistic regression.

2.4 Logistic Regression

According to Burkov (2019), logistic regression is not a regression, but it's a classification algorithm. The name is used in statistics and because the mathematical formula is similar to that of linear regression it's called that way.
Logistic regression is often used in binary classification, for example if the depend y_i can only be 0 or 1. It's also suitable for multiclass classification (Burkov, 2019). In multi-class classification you would want to have multiple classes like, airplane, bicycle, autobus et cetera. What logistic regression does is basically calculating the probability of y_i being of a certain class.

To calculate the probability of an email being spam 1, or no spam 0 we could use the logistic regression formula like this:

$$P(Y=1) = \frac{1}{1+e^{-(ax+b)}}$$

The outcome will be a value between 0 and 1, if the value is higher or equal then a certain threshold, let's say 0.8, the email is classified as spam. In the denominator of the formula the resemblance with the linear regression model is visible. The difference between the two models is that in linear regression we minimize the mean squared error of our training data.
In logistics regression we maximize the likelihood of our training data according to the model (Burkov, 2019).

We have got a basic understanding of logistic regression. The next learning method we will describe is decision tree learning.

2.5 Decision Tree Learning

Decision trees make use of an acyclic graph that is used to make decisions. In each branching node within the graph, a specific feature j of the feature vector is examined (Burkov, 2019). When the value of the specific feature is lower than a certain threshold, the left branch is followed, else, the right branch is followed. The decision is made when the leaf node is reached (Burkov, 2019).

One of the formulations of decision tree algorithms is called ID3:

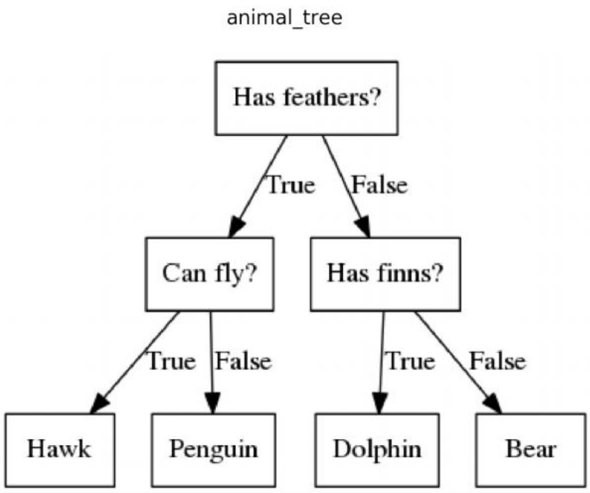
$$\frac{1}{N}\sum_{i=1}^{N}\left[y_i \ln f_{ID3}(\mathbf{x}_i) + \left(1-y_i\right)\ln\left(1-f_{ID3}(\mathbf{x}_i)\right)\right],$$

Here f_{ID3} is the decision tree.

Decision trees are used in various applications. For example, the detection of anomalies in data. An advantage of decision trees is that the graphical notation makes it easy to understand. Which favors the explicability of the algorithm.

Figure 1

Example of a decision tree for animals



Note. Towards AI, Animal Tree.
From https://towardsai.net, by Iriondo, 2018.
(https://towardsai.net/p/machine-learning/differences-between-ai-and-machine-learning-1255b182fc6)

In this section we have seen what decision tree learning can be used for. The next section is going to be about another important algorithm in traditional machine learning. The Support Vector Machine (SVM).

2.6 Support Vector Machine

One of the most influential approaches to supervised learning is the support vector machine (Boser et al., 1992; Cortes and Vapnik, 1995).
To stick to the example of the spam filter, a human will label the messages with label either “spam” or “no-spam”. Since computers in the end only understand binary, we need to convert the words in the message containing the features we are looking for into a machine-readable format. This is done using the so-called “bag of words”. Basically, we count the number of occurrences a word appears in the text of the email. The presence of a certain word, for example “casino” in an email will make the feature 1, so the word “casino” is a feature with a value of 1.

This is the model for Support Vector Machine (SVM):

$$f(x) = sign\left(w^{\cdot} x - b^{\cdot}\right),$$

Where the sign function is a mathematical operation that converts every positive input to +1 and any negative input becomes –1. Here $w^{\cdot}$ and $b^{\cdot}$ are the optimal values for parameters w and b.

SVM is used in classification and regression problems it is a very popular algorithm. In the next section we will cover k-Nearest Neighbors.

2.7 k-Nearest Neighbors

k-Nearest Neighbors (kNN) is a so-called non-parametric learning algorithm. Where other learning algorithms allow discarding the training data after the model is build, kNN keeps all training data in memory (Burkov, 2019).

In the example of our spam filter, if an unseen email comes in, the kNN algorithm finds k training examples closest to the email and returns the corresponding label. k-Nearest Neighbors makes use of a distance function to calculate the closeness between two examples. There are other distance functions kNN can use. For the scope of this investigation, we will only look at kNN with cosine similarity.

Here is a kNN algorithm using cosine similarity as a distance function:

$$s(x_i, x_k) = \cos(\angle(x_i, x_k)) = \frac{\sum_{j=1}^{D} x_i^{(j)} x_k^{(j)}}{\sqrt{\sum_{j=1}^{D}(x_i^j)^2}\sqrt{\sum_{j=1}^{D}(x_k^{(j)})^2}},$$

Cosine similarity measures the directions of two vectors. When the angle between two vectors is equal to 0 degrees, the algorithm sets the cosine similarity to 1. In the case of orthogonal vectors, the cosine similarity is equal to 0 degrees. If vectors point in opposite directions the cosine similarity is equal to –1.

k-Nearest Neighbors is widely used in machine learning. In this chapter kNN is the last algorithm we will cover; we have now explained the most fundamental algorithms in machine learning today. And answered one of the secondary questions of this paper; What algorithms are used in machine learning?

The next chapter will cover the algorithms that use least squares, if any, and conclude this paper by giving the answer to our central question. What algorithms used in machine learning today use least squares?

3. Conclusion

In chapter 2 we have seen what fundamental algorithms are used in machine learning today. We have got an introduction to the mathematical formulas behind them and have developed an intuition for the way they work. This chapter will focus on answering the question which of the fundamental algorithms used in machine learning make use of least squares if there are any of course.

The first algorithm we covered is linear regression. The objective function, or simply put the objective of linear regression is to minimize the distance between a ground truth value y and the predicted value $\hat{y}$.

The calculation of the residual r_i is done by subtracting the predicted variable $\hat{y}$ from observed variable y_i,

$$r_i = y_i - \hat{y}_i,$$

Then we can do a summation over all values in the vector r_i squared, which we than should minimize,

$$S = \sum_{i=1}^{n} r_i^2.$$

The least-squares method finds the optimal parameter values by minimizing the least-squares method (*Dekking & Michel, 1946*).

Concluding we can say that linear regression, one of the fundamental algorithms in machine learning does indeed use the method of least squares.

The second of the fundamental algorithms we covered is logistics regression. Although the names are similar, and we can see the resemblance in the mathematical notation between the two algorithms. The objective of both algorithms is not the same. Put another way, the optimization criterion of both functions is different. Where linear regression minimizes the average squared errors, logistics regression uses maximum likelihood. Therefore, we can conclude that logistic regression does not use least squares.

Decision tree learning is the third of the machine learning algorithms we have covered in this paper. We used the ID3 implementation. Here we see that the optimization criterion used is average log likelihood. The mathematical formula does not show any least squares optimization. The conclusion we can make is that decision trees do not make use of least squares.

Support vector machine is the fourth algorithm we have seen. Where least squares will try to minimize the distance between the regression line and the points on the hyperplane, the objective of SVM is to try and maximize the margin, between the two classes in the hyperplane, to create a so-called maximum-margin hyperplane. Therefore, we can conclude that SVM does not use least squares.

The fifth and last of the algorithms we have seen is k-Nearest Neighbors. The objective of this algorithm is to find the minimum distance between examples, put another way, the closeness between two examples. In this paper we have examined KNN with the distance function cosine similarity which makes use of calculating the distance in the directions of two vectors. Examining the formula of KNN we can conclude that there is no use of least squares.

From the five algorithms we have examined only one algorithm uses least squares. It would be interesting for further research to investigate if there are any other algorithms used in machine learning or deep learning that under the hood use least squares.

List of Appendices

Output Plots

Figure 1. Chosen ideal functions

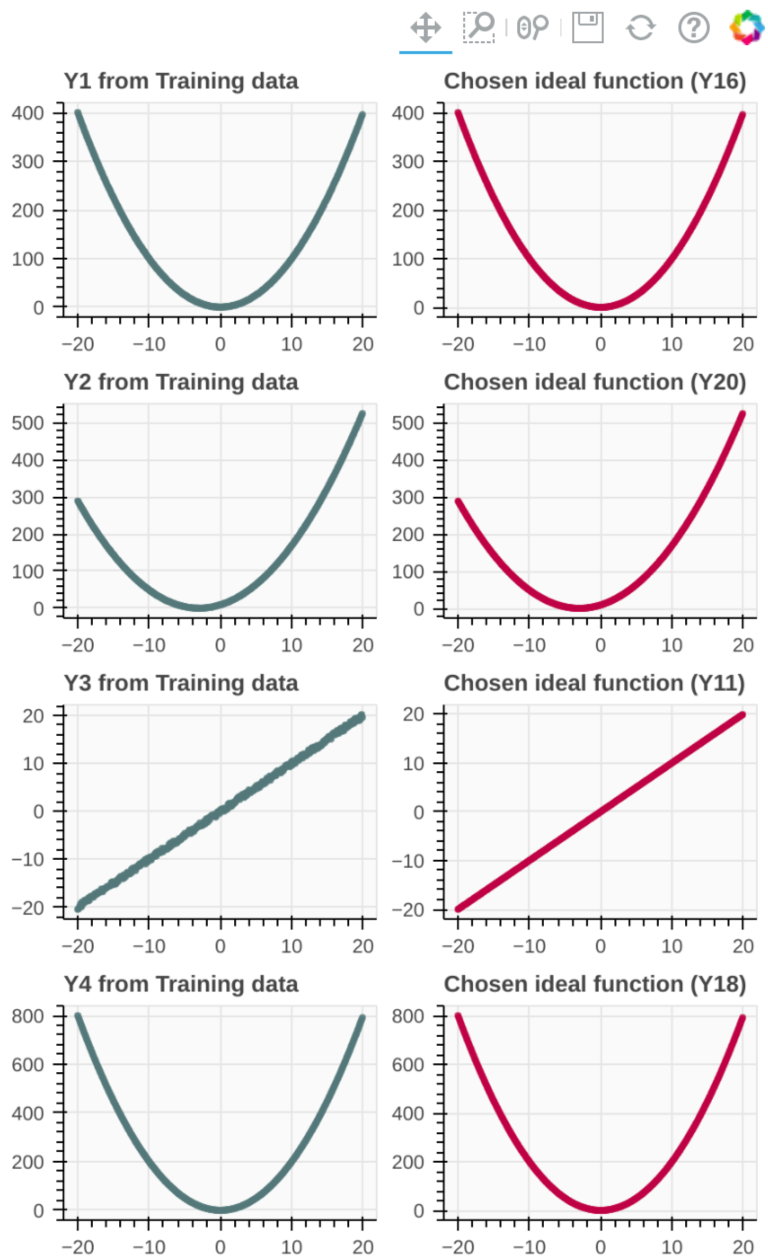
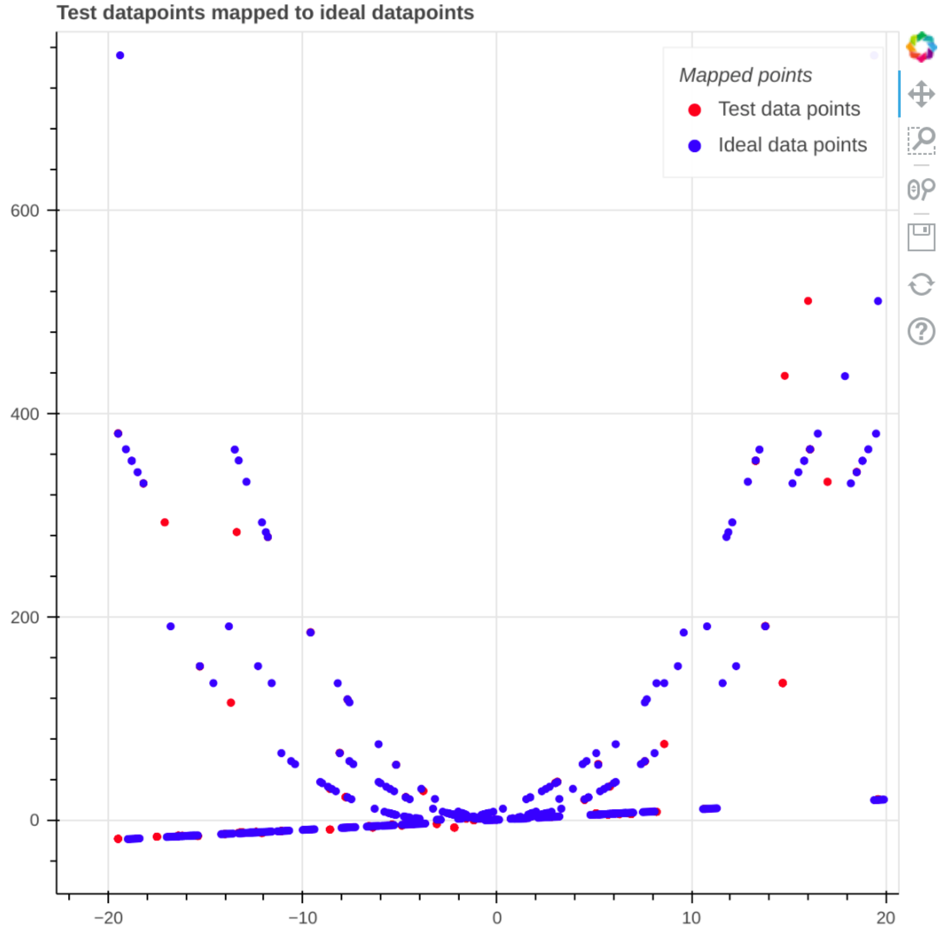


Figure 2. Test datapoints mapped to ideal datapoints



Read me

To use the software provided with the written assignment of this paper please copy/paste the code in each code appendix to separate files and install the requirements.txt with PIP. Create two folders “data” and “output” in the directory where the Python files are and add the data CSV files to the “data” folder. The output folder is where the plots are saved with Bokeh. I recommend using a new virtual environment to keep this software separate from the system OS Python version. The software is tested on the Python version 3.6.9. If copy/pasting the code results in format problems, since sharing code through appendices isn’t a best practice, you can clone the repository which is described in appendix 10: Git Commands.

Create Virtual environment
pip install virtualenv virtualenvwrapper

Update ~/.bashrc
Use vim or nano to open ~/.bashrc and paste the next three line, save the file when done:

export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh

Source ~/.bashrc for changes to take place
source ~/.bashrc

Create virtual environment for the project
mkvirtualenv nameofvirtualenv -p python3

Install requirements.txt
pip install -r requirements.txt

Run main program
python3 run_all.py

Requirements (requirements.txt)

```
absl-py==0.13.0
appdirs==1.4.3
apturl==0.5.2
argcomplete==1.8.1
argh==0.26.2
asn1crypto==0.24.0
astor==0.8.1
Babel==2.4.0
beautifulsoup4==4.6.0
blinker==1.4
Brlapi==0.6.6
Brotli==1.0.4
cached-property==1.5.2
certifi==2018.1.18
chardet==3.0.4
click==6.7
cliff==2.11.0
cmd2==0.7.9
colorama==0.3.7
command-not-found==0.3
ConfigArgParse==0.11.0
construct==2.8.16
cryptography==2.1.4
cssutils==1.0.2
cupshelpers==1.0
debtcollector==1.13.0
decorator==4.1.2
defer==1.0.6
deprecation==1.0.1
distlib==0.3.2
distro-info===0.18ubuntu0.18.04.1
docutils==0.14
dogpile.cache==0.6.2
feedparser==5.2.1
filelock==3.0.12
Flask==0.12.2
funcsigs==1.0.2
gast==0.4.0
google-pasta==0.2.0
grpcio==1.38.0
h2==3.0.1
h5py==3.1.0
hpack==3.0.0
html2text==2018.1.9
html5lib==0.999999999
httplib2==0.9.2
hyperframe==5.1.0
idna==2.6
importlib-metadata==4.5.0
importlib-resources==5.1.4
```

```
iso8601==0.1.11
itsdangerous==0.24
Jinja2==2.10
jmespath==0.9.3
jsbeautifier==1.6.4
jsonpatch==1.16
jsonpointer==1.10
kaitaistruct==0.7
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.2
keyring==10.6.0
keyrings.alt==3.0
keystoneauth1==3.4.0
language-selector==0.1
launchpadlib==1.10.6
lazr.restfulclient==0.13.5
lazr.uri==1.0.3
louis==3.5.0
lxml==4.2.1
macaroonbakery==1.1.3
Mako==1.0.7
Markdown==3.3.4
MarkupSafe==1.0
mitmproxy==2.0.2
monotonic==1.0
msgpack==0.5.6
munch==2.2.0
netaddr==0.7.19
netifaces==0.10.4
numpy==1.19.5
oauth==1.0.1
oauthlib==2.0.6
olefile==0.45.1
openstacksdk==0.11.3
os-client-config==1.29.0
os-service-types==1.1.0
osc-lib==1.9.0
oslo.config==5.2.0
oslo.i18n==3.19.0
oslo.serialization==2.24.0
oslo.utils==3.35.0
passlib==1.7.1
pathtools==0.1.2
pbr==3.1.1
pexpect==4.2.1
Pillow==5.1.0
positional==1.1.1
prettytable==0.7.2
protobuf==3.17.3
pyasn1==0.4.2
PyAudio==0.2.11
pycairo==1.16.2
pycrypto==2.6.1
```

```
pycups==1.9.73
Pygments==2.2.0
pygobject==3.26.1
pyinotify==0.9.6
PyJWT==1.5.3
pymacaroons==0.13.0
PyNaCl==1.1.2
pyOpenSSL==17.5.0
pyparsing==2.2.0
pyperclip==1.6.0
pyRFC3339==1.0
python-apt==1.6.5+ubuntu0.7
python-dateutil==2.6.1
python-debian==0.1.32
python-keystoneclient==3.15.0
python-magic==0.4.16
python-neutronclient==6.7.0
pytz==2018.3
pyxdg==0.25
PyYAML==3.12
reportlab==3.4.0
requests==2.18.4
requests-unixsocket==0.1.5
requestsexceptions==1.3.0
rfc3986==0.3.1
roman==2.0.0
ruamel.yaml==0.15.34
s3cmd==2.0.1
scour==0.36
SecretStorage==2.3.1
simplejson==3.13.2
six==1.11.0
sortedcontainers==1.5.7
stevedore==1.28.0
system-service==0.3
systemd-python==234
tensorboard==1.14.0
tensorflow==1.14.0
tensorflow-estimator==1.14.0
termcolor==1.1.0
tornado==4.5.3
typing-extensions==3.10.0.0
ubuntu-advantage-tools==27.2
ubuntu-drivers-common==0.0.0
ufw==0.36
unattended-upgrades==0.1
urllib3==1.22
urwid==2.0.1
usb-creator==0.3.3
virtualenv==20.4.7
wadllib==1.3.2
watchdog==0.8.3
webencodings==0.5
```

```
Werkzeug==0.14.1
wrapt==1.12.1
xkit==0.0.0
zipp==3.4.1
```

Database class (database.py)

```
from sqlalchemy import create_engine, MetaData, Table, Column, Float,
insert
import pandas as pd
from  collections import defaultdict
import sqlalchemy as db
from sqlalchemy.sql import text
import math
import numpy as np


class DataBase(object):

    '''
    This class is responsible for all data manipulation tasks e.g.:
reading from CSV, writing to SQLite, updating data etc., also
    the database connection and the creation of tables is done in this
class.
    '''

    def __init__(self):

        '''

        Here we initiate the class with the parameters needed.
        For convenience the creation of the SQLite engine for connecting
to the database, the creation of
        the table with test-data, with mapping and y-deviation is done
here also.

        '''

        self.training_data_df =  pd.read_csv('data/train.csv') #create
Pandas dataframe from training csv file
        self.table_name_training = "training"
        self.ideal_data_df =  pd.read_csv('data/ideal.csv') #create
Pandas dataframe from ideal data csv file
        self.table_name_ideal = "ideal"
        self.test_data_df =  pd.read_csv('data/test.csv') #create Pandas
dataframe from test csv file
        self.database_url = "sqlite:///data/linear-regression.db"
        self.metadata = db.MetaData()
        self.trainingdatamatrix = np.empty(shape=(400,5)) #create ndarray
with training data, self.trainingdatamatrix is a (K x L matrix), where K
= 400, and L is 5
        self.idealdatamatrix = np.empty(shape=(400,51)) #create ndarray
with ideal data, self.idealdatamatrix is a (K x L matrix), where K = 400,
and L is 51
```

```
        # create SQLite engine and create table three to save test data,
deviations
        # and choosen ideal functions later
        try:
            self.engine = create_engine(self.database_url, echo = False)

            table_three = Table(
            'table_three', self.metadata,
            Column('x_test', Float),
            Column('y_test', Float),
            Column('delta_y', Float),
            Column('ideal_n_y', Float),)
            self.metadata.create_all(self.engine)

        except Exception as e:
            print("This error occurred during the creation of the SQLite
engine:")
            print(e)



    def insert_training_data(self):

        '''

        This function inserts the training data from a CSV file into the
SQLite database.

        '''

        self.training_data_df.to_sql( #convert Pandas dataframe with
training data to SQL
        self.table_name_training,
        self.engine,
        if_exists='replace',
        index=False,
        chunksize=500,
        dtype={
            "x": Float,
            "y1": Float,
            "y2": Float,
            "y3": Float,
            "y4": Float,
        }
    )

    def insert_ideal_data(self):

        '''

        This function inserts the ideal data from a CSV file into the
SQLite database.
```

```
        '''

        self.ideal_data_df.to_sql( # convert Pandas dataframe with ideal
data to SQL
        self.table_name_ideal,
        self.engine,
        if_exists = 'replace',
        index = False,
        chunksize = 500,
        dtype = self.create_ideal_data_dict() # create ideal data
dictionary, so we don't get 50 lines of y-value declarations
    )

    def create_ideal_data_dict(self):

        '''

        This function creates a dictionary to pass to the "dtype"
argument in the "to_sql" function
        of Pandas. Without this we would have 50 lines of column
declarations in our code.

        Returns:
        dict: with columnnames as keys and "float" as value for each pair

        '''

        self.ideal_data_dict = {"x": Float}
        for i in range(1,51):
            self.ideal_data_dict['y'+str(i)]=Float

        return self.ideal_data_dict


    def read_training_data(self):

        '''

        This function reads the x and y-values from the SQLite database
tables and creates
        ndarray with shape (400, 5)

        Raises:
        A custom exception if there is an error reading the database

        Returns:
        ndarray with shape (400, 5)

        '''

#         if not 1 <= y_column < 5:
#             raise CustomException(y_column, "There are only four y-
columns in the training dataset, please provide 1, 2, 3 or 4 as values
```

```
{}".format(y_column))

        self.training_data = db.Table('training', self.metadata,
autoload=True, autoload_with=self.engine)
        self.query = db.select([self.training_data.columns.x,
self.training_data.columns.y1, self.training_data.columns.y2,
self.training_data.columns.y3, self.training_data.columns.y4])
        self.results = self.engine.execute(self.query).fetchall()

        start = 0
        for value in self.results:
            self.trainingdatamatrix[start] = self.results[start] #add
values from SQLite to ndarray
            start += 1

        return self.trainingdatamatrix # (self.trainingdatamatrix is (K x
L) matrix, where K = 400, and L is 5)


    def read_ideal_data(self):

        '''

        This function reads the x and y-values from the SQLite database
tables (Ideal data ) and creates
        ndarray with shape (400, 51)

        Raises:
        A custom exception if there is an error reading the database

        Returns:
        ndarray with shape (400, 51)

        '''
        self.ideal_data = db.Table('ideal', self.metadata, autoload=True,
autoload_with=self.engine)
        self.query = db.select([self.ideal_data.columns.x,
self.ideal_data.columns.y1, self.ideal_data.columns.y2,
self.ideal_data.columns.y3, self.ideal_data.columns.y4,
self.ideal_data.columns.y5, self.ideal_data.columns.y6,
self.ideal_data.columns.y7, self.ideal_data.columns.y8,
self.ideal_data.columns.y9, self.ideal_data.columns.y10,
self.ideal_data.columns.y11, self.ideal_data.columns.y12,
self.ideal_data.columns.y13, self.ideal_data.columns.y14,
self.ideal_data.columns.y15, self.ideal_data.columns.y16,
self.ideal_data.columns.y17, self.ideal_data.columns.y18,
self.ideal_data.columns.y19, self.ideal_data.columns.y20,
self.ideal_data.columns.y21, self.ideal_data.columns.y22,
self.ideal_data.columns.y23, self.ideal_data.columns.y24,
self.ideal_data.columns.y25, self.ideal_data.columns.y26,
self.ideal_data.columns.y27, self.ideal_data.columns.y28,
self.ideal_data.columns.y29, self.ideal_data.columns.y30,
self.ideal_data.columns.y31, self.ideal_data.columns.y32,
```

```
self.ideal_data.columns.y33, self.ideal_data.columns.y34,
self.ideal_data.columns.y35, self.ideal_data.columns.y36,
self.ideal_data.columns.y37, self.ideal_data.columns.y38,
self.ideal_data.columns.y39, self.ideal_data.columns.y40,
self.ideal_data.columns.y41, self.ideal_data.columns.y42,
self.ideal_data.columns.y43, self.ideal_data.columns.y44,
self.ideal_data.columns.y45, self.ideal_data.columns.y46,
self.ideal_data.columns.y47, self.ideal_data.columns.y48,
self.ideal_data.columns.y49, self.ideal_data.columns.y50])
        self.results = self.engine.execute(self.query).fetchall()

        start = 0
        for value in self.results:
            self.idealdatamatrix[start] = self.results[start] # add
values from SQLite to ndarray
            start += 1

        return self.idealdatamatrix # (self.idealdatamatrix is (K x L)
matrix, where K = 400, and L is 51)

    def read_test_data(self):

        '''

        This function reads the x and y-values from the test data CSV
file and creates
        ndarray with shape (100, 2)

        Raises:
        A custom exception if there is an error reading the CSV file

        Returns:
        ndarray with shape (100, 2)

        '''
        return (self.test_data_df.to_numpy(copy=False))

    def insert_mapped_test_data(self, x_test, y_test, delta_y,
ideal_n_y):

        '''

        This function saves the mapped testdata to the database

        Raises:
        A custom exception if there is an error saving the data

        Returns:
        Boolean: success or failure

        Parameters:
        x_test: decimal
        y_test: decimal
```

```
        delta_y: decimal
        ideal_n_y: decimal

        '''

        with self.engine.connect() as con:
            self.rs = con.execute('INSERT INTO table_three (x_test,
y_test, delta_y, ideal_n_y) VALUES (?, ?, ?, ?)', (x_test, y_test,
delta_y, ideal_n_y))
            print (self.rs)
```

Custom Exception class (custom_exception.py)

```
class CustomException(Exception):

    '''
    In this class we created our own exceptions to raise when necessary.
    '''

    def __init___(self, exception_parameter, exception_message):

        super().__init__(self, exception_parameter, exception_message)
```

Plotting class (plot.py)

```
from math import e
from sqlalchemy import create_engine, MetaData, Table, Column, Float
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from bokeh.plotting import figure
from bokeh.io import show, output_notebook, output_file
from bokeh.layouts import gridplot, row
from database import *
from stats import *
from bokeh.plotting import figure, show
from bokeh.sampledata.iris import flowers
from bokeh.models import Circle, ColumnDataSource, Grid, LinearAxis, Plot



class Plot:

    '''
    This class is responsible for all plotting tasks e.g.: plotting
charts with training/test data etc.
    '''

    def plot_training_and_ideal(self):

        output_file("output/chosen-ideal-functions.html")

        # Get data from database class
        data_actions = DataBase()
        td = data_actions.read_training_data()
        td_ideal = data_actions.read_ideal_data()

        x = [row[0] for row in td]

        y1 = [y1[1] for y1 in td]
        i_y1 = [i_y1[16] for i_y1 in td_ideal]

        # create plot with training data y1
        s1 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Y1 from Training data")
        s1.circle(x, y1, size=3, color="#53777a", alpha=0.8)

        # create plot with ideal data y16
        s2 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Chosen ideal function (Y16)")
        s2.circle(x, i_y1, size=3, color="#c02942", alpha=0.8)

        y2 = [y2[2] for y2 in td]
        i_y2 = [i_y2[20] for i_y2 in td_ideal]
```

```
        # create plot with training data y2
        s3 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Y2 from Training data")
        s3.circle(x, y2, size=3, color="#53777a", alpha=0.8)

        # create plot with ideal data y20
        s4 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Chosen ideal function (Y20)")
        s4.circle(x, i_y2, size=3, color="#c02942", alpha=0.8)

        y3 = [y3[3] for y3 in td]
        i_y3 = [i_y3[11] for i_y3 in td_ideal]

        # create plot with training data y2
        s5 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Y3 from Training data")
        s5.circle(x, y3, size=3, color="#53777a", alpha=0.8)

        # create plot with ideal data y3
        s6 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Chosen ideal function (Y11)")
        s6.circle(x, i_y3, size=3, color="#c02942", alpha=0.8)

        y4 = [y4[4] for y4 in td]
        i_y4 = [i_y4[18] for i_y4 in td_ideal]

        # create plot with training data y2
        s7 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Y4 from Training data")
        s7.circle(x, y4, size=3, color="#53777a", alpha=0.8)

        # create plot with ideal data y3
        s8 = figure(width=215, height=170,
background_fill_color="#fafafa", title="Chosen ideal function (Y18)")
        s8.circle(x, i_y4, size=3, color="#c02942", alpha=0.8)

        # make a grid
        grid = gridplot([[s1, s2], [s3, s4], [s5, s6], [s7, s8]])
        show(grid)

    def plot_test_ideal_data_points(self):

        # output to static HTML file
        output_file("output/testdata-mapped-to-idealdata.html")

        # Get data from stats class
        data = Stats()

        mapped_test_data_point_x, mapped_ideal_data_point_x,\
        mapped_test_data_point_y, mapped_ideal_data_point_y\
        = data.map_test_data()

        p = figure(plot_width = 600, plot_height=600, title = "Test
```

```
datapoints mapped to ideal datapoints") #, x_range=(-1000, 1000),
y_range=(-1000, 1000)

        circles1 = p.circle(mapped_test_data_point_x,
mapped_test_data_point_y, size=5, color="red", line_color=None,
legend_label="Test data points")
        circles1.selection_glyph    = Circle(fill_color="blue",
line_color=None)
        circles1.nonselection_glyph =
Circle(fill_color="red",  line_color=None)

        circles2 = p.circle(mapped_ideal_data_point_x,
mapped_ideal_data_point_y, size=5, color="blue", line_color=None,
legend_label="Ideal data points")
        circles2.selection_glyph    = Circle(fill_color="blue",
line_color=None)
        circles2.nonselection_glyph =
Circle(fill_color="red",  line_color=None)

        # display legend in top right corner
        p.legend.location = "top_right"

        # give title to legend
        p.legend.title = "Mapped points"

        show(p)
```

Statistics class (stats.py)

```
from numpy.lib.function_base import append
from database import *
from plot import *


class Stats():

    '''
    This class is responsible for all statistics needed for the
assignment like,
    choosing the Ideal function, calculating least squares, mean squared
error etc.
    '''

    def choose_ideal_functions(self):

        '''

        This function chooses the ideal functions from the ideal table in
the SQLite database table.
        For each Y-value in the training data columns it computes the
Total Least Squares deviation and
        compares this with the computed Y-value deviations from the ideal
data columns.
        The one column from the ideal data that has the smallest
difference in
        Total Least Squares deviation with the training data is then
choosen as ideal.


        Raises:
        ..

        Returns:

        Tuple with choosen ideal functions for each training function and
the value of the deviation

        '''
        np.set_printoptions(suppress=True) #Disable scientific notation
in Numpy, so we can
        #more easily see what our data looks like
        data_actions = DataBase()
        training_data = data_actions.read_training_data()
        ideal_data = data_actions.read_ideal_data()

        ty = np.delete(training_data, 0, axis=1) #remove column with x
values from matrix, since we don't need it to choose Ideal function
        iy = np.delete(ideal_data, 0, axis=1) #remove column with x
values from matrix, since we don't need it to choose Ideal function
```

```
        #Create dictionaries with mean sqaured errors
        mse_y_1 = {}
        mse_y_2 = {}
        mse_y_3 = {}
        mse_y_4 = {}

        for t_yn in range(0, 4):
            for i_yn in range(0, 50):
                mse = np.square(np.subtract(ty[:, t_yn],iy[:,
i_yn])).mean() # calculate mean squared errors between training-y and
ideal-y functions
                if (t_yn == 0):
                    mse_y_1[i_yn] = mse
                if (t_yn == 1):
                    mse_y_2[i_yn] = mse
                if (t_yn == 2):
                    mse_y_3[i_yn] = mse
                if (t_yn == 3):
                    mse_y_4[i_yn] = mse

        # Create tuples y1..yn with the choosen ideal function minimum
deviation and the value
        # of the deviation
        y_1 = min(mse_y_1.items(), key=lambda x: x[1])
        y_2 = min(mse_y_2.items(), key=lambda x: x[1])
        y_3 = min(mse_y_3.items(), key=lambda x: x[1])
        y_4 = min(mse_y_4.items(), key=lambda x: x[1])

        return y_1, y_2, y_3, y_4


    def map_test_data(self):

        '''

        This function will map the test data to the ideal data and save
the deviation at hand
        when the maximum deviation of the calculated regression does not
exceed the largest deviation
        between training dataset (A) and the ideal function (C) chosen
for it by more than
        factor sqrt(2)

        Parameters:
        ..

        Raises:
        ..

        Returns:
        ..
```

```
        '''
        np.set_printoptions(suppress=True) #Disable scientific notation
in Numpy, so we can
        #more easily see what our data looks like
        data_actions = DataBase()
        test_data = data_actions.read_test_data()
        ideal_data = data_actions.read_ideal_data()

        #Create lists of the ideal function columns from the matrix with
ideal data
        i_y1 = [i_y1[16] for i_y1 in ideal_data]
        i_y2 = [i_y1[20] for i_y1 in ideal_data]
        i_y3 = [i_y1[11] for i_y1 in ideal_data]
        i_y4 = [i_y1[18] for i_y1 in ideal_data]
        ideal_data_point_x = [i_x[0] for i_x in ideal_data]



        ty = test_data
        # max deviation
        max_dev = 0.08778705256534793

        max_dev_square_root = math.sqrt(max_dev) #error band, not exactly
sure what teacher means, keep both, and ask Lino
        devsquared = max_dev * math.sqrt(2)

        # create lists of mapped points to pass to plotter class
        mapped_test_data_point_x = []
        mapped_test_data_point_y = []
        mapped_ideal_data_point_x = []
        mapped_ideal_data_point_y = []

        for idt, t_yn in enumerate(ty):
            for idi, ideal_data_point in enumerate(i_y1):
                mse =
np.square(np.subtract(t_yn[1],ideal_data_point)).mean() # calculate mean
squared errors between training-y and ideal-y functions
                if mse <= devsquared:
                    data_actions.insert_mapped_test_data(t_yn[0],
t_yn[1], mse, 16)
                    mapped_test_data_point_x.append(t_yn[0])
                    mapped_test_data_point_y.append(t_yn[1])
                    mapped_ideal_data_point_x.append(ideal_data_point_x[i
di])
                    mapped_ideal_data_point_y.append(ideal_data_point)

            for idi2, ideal_data_point2 in enumerate(i_y2):
                mse =
np.square(np.subtract(t_yn[1],ideal_data_point2)).mean() # calculate mean
squared
                if mse <= devsquared:
                    data_actions.insert_mapped_test_data(t_yn[0],
t_yn[1], mse, 20)
```

```
                    mapped_test_data_point_x.append(t_yn[0])
                    mapped_test_data_point_y.append(t_yn[1])
                    mapped_ideal_data_point_x.append(ideal_data_point_x[i
di2])
                    mapped_ideal_data_point_y.append(ideal_data_point2)


            for idi3, ideal_data_point3 in enumerate(i_y3):
                mse =
np.square(np.subtract(t_yn[1],ideal_data_point3)).mean()
                if mse <= devsquared:
                    data_actions.insert_mapped_test_data(t_yn[0],
t_yn[1], mse, 11)
                    mapped_test_data_point_x.append(t_yn[0])
                    mapped_test_data_point_y.append(t_yn[1])
                    mapped_ideal_data_point_x.append(ideal_data_point_x[i
di3])
                    mapped_ideal_data_point_y.append(ideal_data_point3)

            for idi4, ideal_data_point4 in enumerate(i_y4):
                mse =
np.square(np.subtract(t_yn[1],ideal_data_point4)).mean()
                if mse <= devsquared:
                    data_actions.insert_mapped_test_data(t_yn[0],
t_yn[1], mse, 18)
                    mapped_test_data_point_x.append(t_yn[0])
                    mapped_test_data_point_y.append(t_yn[1])
                    mapped_ideal_data_point_x.append(ideal_data_point_x[i
di4])
                    mapped_ideal_data_point_y.append(ideal_data_point4)

        return mapped_test_data_point_x, mapped_ideal_data_point_x,
mapped_test_data_point_y, mapped_ideal_data_point_y
```

Unit test (unit_test.py)

```
import unittest
import pathlib as pl

# Subclass unit test testcase
class TestCaseBase(unittest.TestCase):
    def assertIsFile(self, path):
        if not pl.Path(path).resolve().is_file():
            raise AssertionError("File does not exist: %s" % str(path))

# Subclassing our own created TestCaseBase class
class DataTest(TestCaseBase):
    def test_test_data(self):
        path = pl.Path("data/test.csv")
        self.assertIsFile(path)

    def test_ideal_data(self):
        path = pl.Path("data/ideal.csv")
        self.assertIsFile(path)

    def test_train_data(self):
        path = pl.Path("data/train.csv")
        self.assertIsFile(path)

if __name__ == '__main__':
    unittest.main()
```

Main (run_ouput.py)

```
from database import *
from plot import *
from stats import *
from unit_test import *

def my_suite():
    suite = unittest.TestSuite()
    result = unittest.TestResult()
    suite.addTest(unittest.makeSuite(DataTest))
    runner = unittest.TextTestRunner()
    print(runner.run(suite))

def main():

    # run tests to check if datafiles are in place
    my_suite()

    data_actions = DataBase()
    # Create Tables
    data_actions.insert_training_data()
    data_actions.insert_ideal_data()

    plot_actions = Plot()
    plot_actions.plot_training_and_ideal()

    plot_actions.plot_test_ideal_data_points()

# run the program
if __name__ == "__main__":
    main()
```

Git commands

To clone the project
git clone https://github.com/MadebyhumansAI/least-squares-ideal-functions.git

After editing a file
git commit -am "Edited file”

To push it again to the repository
git push

After adding files to the repository
git add filename

Or to add all files that are changed
git add .

To check the status of the branche
git status

To pull the changes a colleague made
git pull

Bibliography

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In COLT *'92: Proceedings of the fifthannual workshop on Computational learning theory*, pages 144-152, New York, NY, USA. ACM.

Burkov, A. (2019). *The Hundred-page Machine Learning Book*.

Dekking, & Michel. (1946). *A Modern Introduction to Probability and Statistics.* Springer Science & Business Media.

Fujii, K. (2018). Least Squares Method from the View Point of Deep Learning. *Advances in Pure Mathematics*, *05*, 485–493. https://doi.org/10.4236/apm.2018.85027

Merriman, M. (1877). *A List of Writings Relating to the Method of Least Squares*.

Miller, S. (n.d.). The Method of Least Squares. *Department of Mathematics and Statistics Williams College*. https://web.williams.edu/Mathematics/sjmiller/public_html/probabilitylifesaver/MethodLeastSquares.pdf