MACHINE LEARNING

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

What is Machine Learning?

To start talking about machine learning we should first go through the history, how the term came up, and all the processes. Machine learning history starts in 1943 with the first mathematical model of neural networks presented in the scientific paper "A logical calculus of the ideas immanent in nervous activity" by Walter Pitts and Warren McCulloch. Different calculations brought into different solutions where in the book is discussed how many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time, where this statement tells us that in particular conditions movements of situations are connected to each other in logical solutions from where we can build new software which knows how to apply the logical calculation to predict the new situation or give new solutions, also that we can train that software with new data we insert. In 1950 Alan Turing created the Turing Test if a computer has real intelligence, in order to pass the test computer has to fool the human that the computer is also a real human.

Machine Learning is a branch of artificial intelligence(AI) and computer science that focuses on the use of data and algorithms to imitate the way how humans learn, gradually improving its accuracy. Different companies are early focused on machine learning which helps automatize the world and these late years we can see it taking place in every industry. One of these companies is IBM where the first-ever computer learning program was written in 1952 by Arthur Samuel. The program was the game of checkers, and the IBM computer improved at the game the more it played, studying which moves made up winning strategies and incorporating those moves into its program which explains really well the term machine learning (machine is learning more and more while using it, collecting data, analyzing them based on given algorithms and applying the better and better solutions day to day).

Over time machine learning shifted from a knowledge-driven to a data-driven approach, where scientists began creating programs for computers to analyze large amounts of data and draw conclusions based on algorithms used to analyze. These days in our daily use of our intelligent devices we get into sites or programs that use machine learning to have a better performance like Netflix and Spotify suggestions where most of us spent most of our free time since the algorithm has gotten so good that we keep thinking how does this program know, or seeing self-driving cars. Combining mathematical solutions with the computing power of our personal computers and the servers that are used to host most of nowadays powerful programs algorithms are trained to make classifications or predictions and get key insights in data mining projects.

Machine learning algorithms are typically created using frameworks like PyTorch or TensorFlow where PyTorch is written in Python and TensorFlow in C++ but there is a huge similarity between those basing on the fact that Python is written in C.

As time passed machine learning got better and better where there is introduced a new term Deep Learning. But what is the difference between those two? Deep learning can use labeled datasets also known as supervised learning to inform its algorithms but it's not necessarily labeled, and can also take unstructured data in its raw form and it can automatically determine the set of features that distinguish different categories of data from one another a case that eliminates some human interventions where in the other hand the classical(non-deep) machine learning is more dependent on human interventions to learn. A neural network that consists of more than 3 layers can be considered a deep learning algorithm or a deep neural network. Deep learning and neural network are credited with accelerating progress in areas such as computer vision.

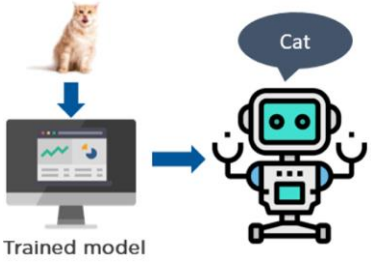
How does machine learning work?

We can break it down into three main steps:

1. A decision process: This describes the main use of the algorithm to predict or classify based on some input data. This step includes collecting data, preparing data, choosing a model, and training it.
2. An error function: Evaluate the prediction of a model. If there are known examples an error function can make a comparison to assess the accuracy of the model.
3. A model optimization process: To check if the model can fit better to the data points in the training set. This is done by tuning the parameters in your model. At a particular value of the parameter, the accuracy will be at maximum where finding these values helps in model optimization.

There are different types of machine learning

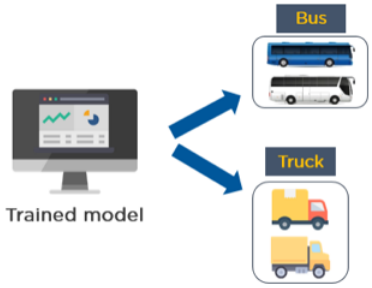
Supervised Machine learning: Supervised learning is a type of machine learning that uses labeled data to train machine learning models. In labeled data, the output is already known. The model just needs to map the inputs to the respective outputs. As input data is given to the model, the model adjusts its weights until it has been fitted. This helps solve real-world problem as classifying spam emails, identifying figures



Most popular supervised algorithms: Linear Regression, Logistic regression, Support vector machine, K nearest neighbor, Decision tree, Random Forest, Naïve bayes.

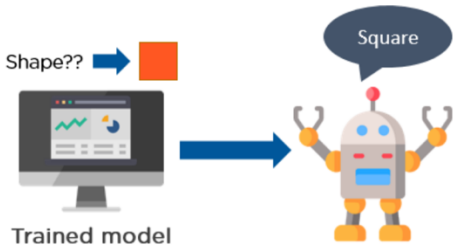
Few of the top supervised learning applications are weather prediction, sales forecasting, stock price analysis.

Unsupervised learning: Unsupervised learning is a type of machine learning that uses unlabeled data to train machines. Unlabeled data doesn't have a fixed output variable. The model learns from the data, and discovers hidden patterns or data groupings without the need for human intervention. This method's ability to discover similarities and differences in information makes it ideal for exploratory data analysis, and cross-selling strategies. Most popular algorithms: K means clustering, Hierarchical clustering, DBSCAN, and Principal Component Analysis. Unsupervised learning finds patterns and understands the trends in the data to discover the output. So, the model tries to label the data based on the features of the input data. The training process used in unsupervised learning techniques does not need any supervision to build models. They learn on their own and predict the output.



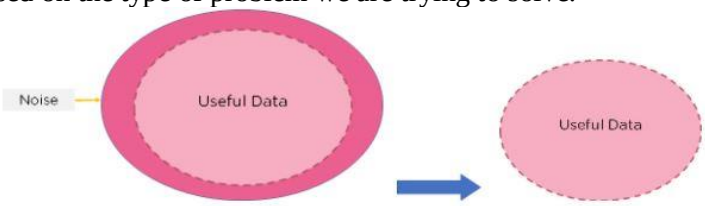
Semi-supervised learning: Is a medium between supervised and unsupervised learning. During training it uses a smaller labeled dataset to guide the classification and feature extraction from a larger unlabeled dataset.

Reinforcement machine learning: similar to supervised but the algorithm isn't trained using sample data. This model learns as it goes by using trial and error. It uses an agent and an environment to produce actions and rewards.



The agent has a start and an end date.
The most used algorithms: Q-learning, Sarsa, Monte Carlo, Deep Q network. After completing the task the agent receives an award.. Do not need any external supervision to train the model.

Machine learning models follow a simple rule: whatever goes in comes out. If we put garbage into our model we can expect the output to be garbage too. In this case, garbage refers to noise in our data. To train a model we collect enormous quantities of data to help the machine learn better. Usually, a good portion of the data collected is noise while some of the columns of our dataset might not contribute significantly to the performance of our model. Further having a lot of data can slow down the training process and cause the model to be slower. The model may also learn from this irrelevant data and be inaccurate
To reduce this noise data in machine learning is used feature selection, which tries to get rid of these irrelevant data. It is the process of automatically choosing relevant features for our machine-learning model based on the type of problem we are trying to solve.



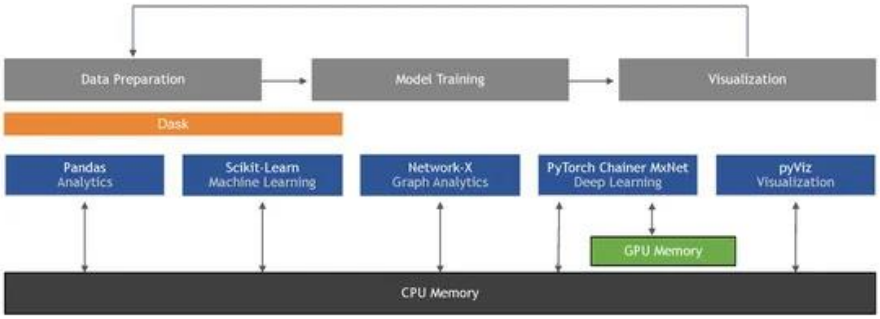
We do this by including or excluding important features without changing them. 2 types of feature selection models: Supervised and unsupervised.

To be a good developer in machine learning mathematics is one of the most crucial prerequisites for becoming an expert in this field. It is a fundamental skill that is needed to be possessed for working with machine learning algorithms. From choosing the right algorithm to selecting the correct parameter, it all uses mathematical concepts in every step of the machine learning process. Some of the key important skills needed in mathematics for machine learning are: Statistics and probability, linear algebra, and calculus where most of the algorithms applied are based on these concepts. By these days machine learning has a really wide market and it is getting wider and wider with the latest development in technology and how the world is going through Artificial intelligence and getting the latest development which is ChatGPT, the reason that pushed me to take this subject as a case study to talk about it and learn more while studying. It was always great to see how to handle big data, how programmers develop different frameworks to make it all easy to work with data, how to choose the right algorithm, and all the stuff while searching I saw that the most preferred programming language for machine learning, analyses, big data and artificial intelligence is python, a reason that pushed me immediately to write about machine learning.

Machine Learning with Python

Scientific Computing and Machine Learning in Python

Python is one of the most popular programming languages for data science and therefore enjoys a large number of useful add-on libraries developed by the community. Although the performance of interpreted language such as Python for the computation-intensive task is inferior to lower-level programming languages, extension libraries such as NumPy and SciPy have been developed that build upon lower-layer Fortran and C implementations for fast and vectorized operations on multidimensional arrays.
What helps in programming is formulating also the arithmetic operations as a sequence of actions instead of performing a set of operations for each element at a time we can make better use of our modern CPU architectures with Single Instruction, Multiple Data(SIMD) support. A library that helps us to do this is NumPy which uses highly optimized linear algebra libraries, that's why we can see using Numpy easily translated in pure Python functions. Python language has seen a tremendous growth of popularity within the scientific computing community within the last decade, most recent machine learning and deep learning libraries are now Python-based. Aside from the benefits of the language itself, the community around the available tools and libraries make Python particularly attractive for workloads in data science, machine learning, and scientific computing.
CPU-bound code in a single thread, and its multiprocessing packages come with other significant performance trade-offs. An alternative to the CPython implementation of the Python language is PyPy which it runs code four times faster than CPython on average. NumPy is a multidimensional array library with basic linear algebra routines, and the SciPy library adorns NumPy arrays with many important primitives, from numerical optimizers and signal processing to statistics and sparse linear algebra. As of 2019, SciPy was found to be used in almost half of all machine learning projects on GitHub.



Optimizing Python's Performance for Numerical Computing and Data Processing

Special optimized instruction sets for the CPU, like Intel's Streaming SIMD Extensions (SSE) and IBM's AltiVec, are being used underneath many low-level library specifications, such as the Binary Linear Algebra Subroutines in order to take full advantage of these modern processors.
When numerical libraries such as NumPy and SciPy receive a substantial performance boost, for example, through hardware-optimized subroutines, the performance gains automatically extend to higher-level machine learning libraries, like Scikit-learn, which primarily use NumPy and SciPy. The aforementioned CPU instruction sets enable vectorization by making it possible for the processors to schedule a single instruction over multiple data points in parallel, rather than having to schedule different instructions for each data point.
Modin's DataFrame features the same API as the Pandas' equivalent, but it can leverage external frameworks for distributed data processing in the background, such as Ray.

Scikit-learn, the Industry Standard for Classical Machine Learning

Scikit-learn has become the industry standard Python library used for feature engineering and classical ML modeling on small to medium-sized datasets in no small part because it has a clean, consistent, and intuitive

API. In addition, with the help of the open-source community, the Scikit-learn developer team maintains a strong focus on code quality and comprehensive documentation. In addition to its numerous classes for data processing and modeling, referred to as *estimators*, Scikit-learn also includes a first-class API for unifying the building and execution of machine learning pipelines: the pipeline API.

a

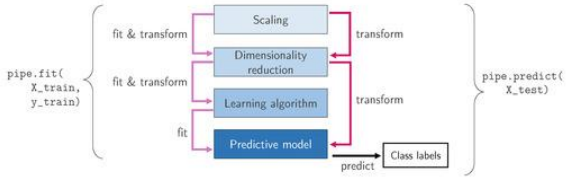
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn import datasets
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.3,
                     random_state=42, stratify=y)

pipe = make_pipeline(StandardScaler(),
                     PCA(n_components=2),
                     SVC(kernel='linear'))

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
print('Test Accuracy: %.3f' % pipe.score(X_test, y_test))
```

b



To find the right balance between providing useful features and the ability to maintain high-quality code, the Scikit-learn development team only considers well-established algorithms for inclusion into the library.

Ensemble Learning: Gradient Boosting Machines and Model Combination

Combinations of multiple machine learning algorithms or models, which are known as ensemble techniques, are widely used for providing stability, increasing model performance, and controlling the bias-variance trade-off. In contrast to bagging, the boosting meta-algorithm is iterative in nature, incrementally fitting weak learners such as pre-pruned decision trees, where the models successively improve upon poor predictions (the leaf nodes) from previous iterations. Gradient boosting improves upon the earlier adaptive boosting algorithms, such as AdaBoost. One major performance challenge of gradient boosting is that it is an iterative rather than a parallel algorithm, such as bagging. Another time-consuming computation in gradient boosting algorithms is to evaluate different feature thresholds for splitting the nodes when constructing the decision trees. Because of the significant performance drawbacks in Scikit-learn's implementation, libraries like XGBoost and LightGBM have emerged, providing more efficient alternatives. Currently, these are the two most widely used libraries for gradient boosting machines, and both of them are largely compatible with Scikit-learn. Both XGBoost and LightGBM support categorical features. While LightGBM can parse them directly, XGBoost requires categories to be one-hot encoded because its columns must be numeric. Both libraries include algorithms to efficiently exploit sparse features, such as those which have been one-hot encoded, allowing the underlying feature space to be used more efficiently.

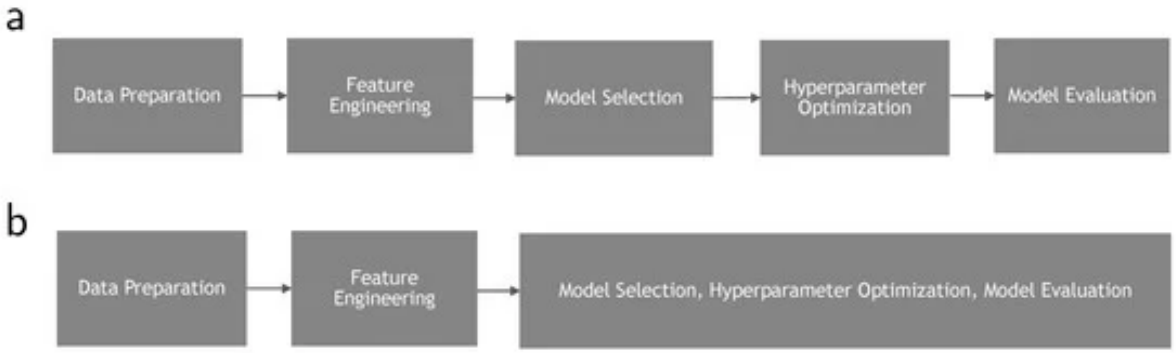
The model combination is a subfield of ensemble learning, which allows different models to contribute to a shared objective irrespective of the algorithms from which they are composed. In model combination algorithms, for example, a logistic regression model could be combined with a k-nearest neighbors classifier and a random forest.

Scalable Distributed Machine Learning

While Scikit-learn is targeted for small to medium-sized datasets, modern problems often require libraries that can scale to larger data sizes. Using the Joblib API, a handful of algorithms in Scikit-learn are able to be parallelized through Python's multiprocessing. Many classical ML algorithms are concerned with fitting a set of parameters that are generally assumed to be smaller than the number of data samples in the training dataset. In distributed environments, this is an important consideration since model training often requires communication between the various workers as they share their local state in order to converge at a global set of learned parameters.
Hyperparameter tuning is a very important use case in machine learning, requiring the training and testing of a model over many different configurations to find the model with the best predictive performance.

Automatic Machine Learning (AutoML)

Libraries like Pandas, NumPy, Scikit-learn, PyTorch, and TensorFlow, as well as the diverse collection of libraries with Scikit-learn-compatible APIs, provide tools for users to execute machine learning pipelines end-to-end manually. Tools for automatic machine learning (AutoML) aim to automate one or more stages of these machine learning pipelines, making it easier for non-experts to build machine learning models while removing repetitive tasks and enabling seasoned machine learning engineers to build better models faster.

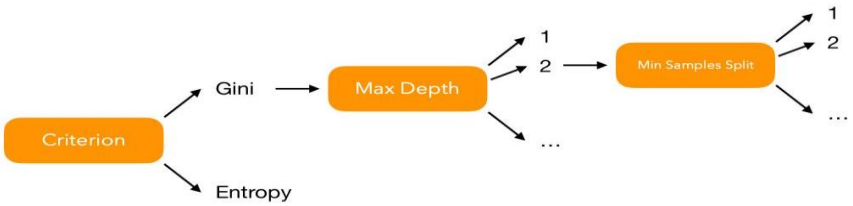


Auto-sklearn's API is directly compatible with Scikit-learn while H2O-AutoML, TPOT, and auto-keras provide Scikit-learn-like APIs. Each of these three tools differs in the collection of provided machine learning models that can be explored by the AutoML search strategy.

Machine learning pipelines often begin with a data preparation step, which typically includes data cleaning, mapping individual fields to data types in preparation for feature engineering, and imputing missing values. Some libraries, such as H2O-AutoML, attempt to automate the data-type mapping stage of the data preparation process by inferring different data types automatically. In the feature extraction stage, the fields are often transformed to create new features with improved signal-to-noise ratios or to scale features to aid optimization algorithms. Libraries like the TPOT model the end-to-end machine learning pipeline directly so they can evaluate variations of feature engineering techniques in addition to selecting a model by predictive performance.

Hyperparameter Optimization and Model Evaluation

Hyperparameter optimization (HPO) algorithms form the core of AutoML. The most naïve approach to finding the best-performing model would exhaustively select and evaluate all possible configurations to ultimately select the best-performing model. The goal of HPO is to improve upon this exhaustive approach by optimizing the search for hyperparameter configurations or the evaluation of the resulting models, where the evaluation involves cross-validation with the trained model to estimate the model's generalization performance.



If we now take into account continuous variables such as a learning rate factor, we rapidly realize that testing all the different hyperparameter combinations is impossible. Grid search and Randomized search are the two most popular methods for hyper-parameter optimization of any model. In both cases, the aim is to test a set of parameters whose range has been specified by the users and observe the outcome in terms of the performance of the model. However, the way the parameters are tested is quite different between Grid Search and Randomized Search. Related to grid search, *random search* is a brute-force approach. However, instead of evaluating all configurations in a user-specified parameter range exhaustively, it chooses configurations at random, usually from a bounded area of the total search space.

Several libraries use a formalism of BO, known as sequential model-based optimization (SMBO), to build a probabilistic model through trial and error. The Hyperopt library brings SMBO to Spark ML, using an algorithm known as the tree of Parzen estimators.

This approach can, however, be long to run and should be used if the model you are tuning does not have too many parameters, or if you don't have too much training data. Grid search is implemented in scikit-learn under the name of GridSearchCV (for cross-validation):

```
from sklearn.model_selection import GridSearchCV

param_grid = {
'penalty': ['l1', 'l2'],
'C': [0.8, 1, 1.2],
'tol': [0.00005, 0.0001, 0.00015, 0.0002],
'fit_intercept' : [True, False],
'warm_start' : [True, False]
}

lr = LogisticRegression()
grid_search = GridSearchCV(lr, param_grid, cv=5, scoring='f1_macro',
return_train_score=True)
grid_search.fit(X_val, y_val)
estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='warn',
n_jobs=None, penalty='l2', random_state=None, solver='warn',
tol=0.0001, verbose=0, warm start=False)
```

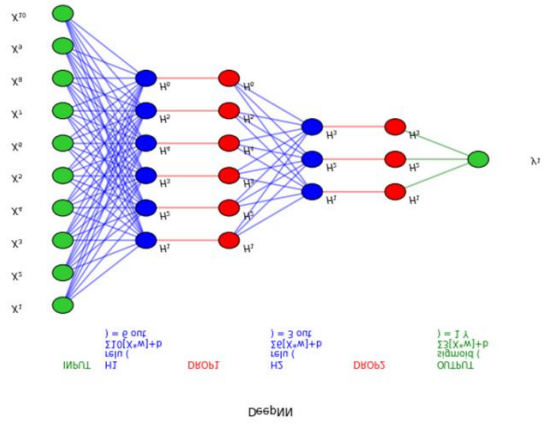
Since all of the above-mentioned search strategies can still be quite extensive and time-consuming, an important step in AutoML and HPO involves reducing the search space, whenever possible, based on any useful prior knowledge. All of the libraries referenced accept an option for the user to bind the amount of time to spend searching for the best model. Auto-sklearn makes use of meta-learning, allowing it to learn from previously trained datasets while both Auto-sklearn and H2O-AutoML provide options to avoid parameters that are known to cause slow optimization.

Neural Architecture Search

The overarching theme in the development of architecture search algorithms is to define a search space, which refers to all the possible network structures, or hyperparameters that can be composed. A search strategy is an HPO over the search space, defining how NAS algorithms generate model structures. Like HPO for classical ML models, neural architecture search strategies also require a model evaluation strategy that can produce an objective score for a model when given a dataset to evaluate. Neural search leverages deep neural networks to intelligently search through all sorts of data, including images, videos, and PDFs. This innovative approach provides a much more comprehensive and contextual search than traditional text-based search engines.

Neural search spaces can be placed into one of four categories, based on how much of the neural network structure is provided beforehand:

1. Entire structure: Generates the entire network from the ground-up by choosing and chaining together a set of primitives, such as convolutions, concatenations, or pooling. This is known as *macro search*.
2. Cell-based: Searches for combinations of a fixed number of hand-crafted building blocks, called cells. This is known as *micro search*.
3. Hierarchical: Extends the cell-based approach by introducing multiple levels and chaining together a fixed number of cells, iteratively using the primitives defined in lower layers to construct the higher layers. This combines macro and micro search.
4. Morphism-based structure: Transfers knowledge from an existing well-performing network to a new architecture



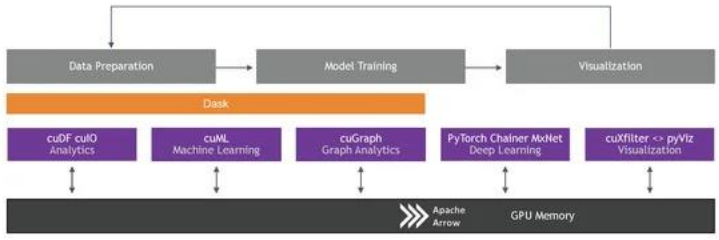
The progressive neural architecture search (PNAS) investigated the use of the Bayesian optimization strategy SMBO to make the search for CNN architectures more efficient by exploring simpler cells before determining whether to search more complex cells.

General Purpose GPU Computing for Machine Learning

Even when efficient libraries and optimizations are used, the amount of parallelism that can be achieved with CPU-bound computation is limited by the number of physical cores and memory bandwidth. Additionally, applications that are largely CPU-bound can also run into contention with the operating system.

Research into the use of machine learning on GPUs predates the recent resurgence of deep learning. Ian Buck, the creator of CUDA, was experimenting with 2-layer fully-connected neural networks in 2005, before joining NVIDIA. Shortly thereafter, convolutional neural networks were implemented on top of GPUs, with a dramatic end-to-end speedup observed over highly-optimized CPU implementations. The release of the first CUDA Toolkit gave life to general-purpose parallel computing with GPUs. Initially, CUDA was only accessible via C, C++, and Fortran interfaces, but in 2010 the PyCUDA library began to make CUDA accessible via Python as well. PyCUDA's numpy interaction code has automatically allocated space on the device, copied numpy arrays a and b over, launched a 400x1x1single block grid, and copied

dest back. GPUs enable the single instruction multiple threads (SIMT) programming paradigms, higher throughput, and more parallel model compared to SIMD, with high-speed memory spanning several multiprocessors (blocks), each containing many parallel cores (threads).
RAPIDS provides unmatched speed with familiar APIs that match the most popular PyData libraries. Built on the shoulders of giants including NVIDIA CUDA and Apache Arrow, it unlocks the speed of GPUs with code you already know. RAPIDS allows fluid, creative interaction with data for everyone from BI users to AI researchers on the cutting edge. GPU acceleration means less time and less cost moving data and training models.



RAPIDS core libraries include near drop-in replacements for the Pandas, Scikit-learn, and Network-X libraries named cuDF, cuML, and cuGraph, respectively. Other components fill gaps that are more focused, while still providing a near drop-in replacement for an industry-standard Python API where applicable.
The Numba library provides just-in-time (JIT) compilation, enabling vectorized functions to make use of technologies like SSE and AltiVec. This separation of describing the computation separately from the data also enables Numba to compile and execute these functions on the GPU. In addition to JIT, Numba also defines a DeviceNDArray, providing GPU-accelerated implementations of many common functions in NumPy's NDArray. Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

```
@njit(parallel=True)
def logistic_regression(Y, X, w, iterations):
    for i in range(iterations):
        w -= np.dot(((1.0 /
              (1.0 + np.exp(-Y * np.dot(X, w)))
              - 1.0) * Y), X)
    return w
```

The TensorFlow and PyTorch libraries define Tensor objects, which are multidimensional arrays. These libraries, along with Chainer, provide APIs similar to NumPy, but build computation graphs to allow sequences of operations on tensors to be defined separately from their execution
CuPy is an open-source array library for GPU-accelerated computing with Python. CuPy utilizes CUDA Toolkit libraries including cuBLAS, cuRAND, cuSOLVER, cuSPARSE, cuFFT, cuDNN and NCCL to make full use of the GPU architecture.
The figure shows CuPy speedup over NumPy. Most operations perform well on a GPU using CuPy out of the box. CuPy speeds up some operations more than 100X.

```
>>> import cupy as cp
>>> x = cp.arange(6).reshape(2, 3).astype('f')
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]], dtype=float32)
>>> x.sum(axis=1)
```

```
array([  3.,  12.], dtype=float32)
```

CuPy's interface is highly compatible with NumPy and SciPy; in most cases, it can be used as a drop-in replacement. All you need to do is just replace numpy and scipy with cupy and cupyx.scipy in your Python code. The Basics of CuPy tutorial is useful to learn the first steps with CuPy.

TensorFlow

In 2016 Google released TensorFlow, which followed a similar approach to Theano by using a static graph model. While this separation of graph definition from execution still does not allow for real-time interaction, TensorFlow reduced compilation times, allowing users to iterate on the different ideas more quickly. TensorFlow also focused on distributed computing, which not many DNN libraries were providing at the time.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x test, y test)
```

TensorFlow allows you to create dataflow graphs that describe how data moves through a graph. The graph consists of nodes that represent a mathematical operation. A connection or edge between nodes is a multidimensional data array. It takes inputs as a multi-dimensional array where you can construct a flowchart of operations that can be performed on these inputs.

While static computation graphs are attractive for applying code optimizations, model export, and portability in production environments, the lack of real-time interaction still makes them cumbersome to use in research environments. The separation between declaration and execution makes static graphs cumbersome for many research contexts, which often require introspection and experimentation. It is important that this section is concluded by noting that all the major deep learning frameworks are now Python-based. According to elaborate analyses of major publishing venues, social media, and search results, many researchers are abandoning TensorFlow in favor of PyTorch. TensorFlow has always provided a direct path to production. Whether it's on servers, edge devices, or the web, TensorFlow lets you train and deploy your model easily, no matter what language or platform you use. Use TFX if you need a full production ML pipeline. For running inference on mobile and edge devices, use TensorFlow Lite. Train and deploy models in JavaScript environments using TensorFlow.js. TensorFlow also supports an ecosystem of powerful add-on libraries and models to experiment with, including Ragged Tensors, TensorFlow Probability, Tensor2Tensor and BERT

Pytorch

PyTorch is an open-source machine learning framework based on the Python programming language and the Torch library. Torch is an open-source ML library used for creating deep neural networks and is written in the Lua scripting language. It's one of the preferred platforms for deep learning research. The framework is built to speed up the process between research prototyping and deployment.

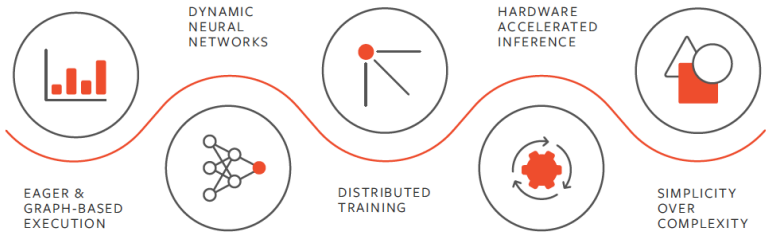
PyTorch is pythonic in nature, which means it follows the coding style that uses Python's unique features to write readable code. Python is also popular for its use of dynamic computation graphs. It enables developers, scientists and neural network debuggers to run and test a portion of code in real-time instead of waiting for the entire program to be written. Transition seamlessly between eager and graph modes with TorchScript, and accelerate the path to production with TorchServe. Both TensorFlow and PyTorch appear to be inspiring each other and are converging on their respective strengths and weaknesses. PyTorch added static graph features (recently enabled by TorchScript) for production and mobile deployment while TensorFlow added dynamic graphs to be more friendly for research. Both libraries are expected to remain popular choices in the upcoming years.

ChatGPT uses PyTorch library.

```
import torch
def fn(x, y):
   a = torch.cos(x).cuda()
   b = torch.sin(y).cuda()
   return a + b
new_fn = torch.compile(fn, backend="inductor")
input_tensor = torch.randn(10000).to(device="cuda:0")
a = new_fn(input_tensor, input_tensor)
```

This example will not actually run faster. Its purpose is to demonstrate the torch.cos() and torch.sin() features which are examples of pointwise ops as in they operate element by element on a vector. A more famous pointwise op you might want to use would be something like torch.relu(). Pointwise ops in eager mode are suboptimal because each one would need to read a tensor from memory, make some changes, and then write back those changes. The single most important optimization that the inductor does is fusion. So back to our example, we can turn 2 reads and 2 writes into 1 read and 1 write which is crucial, especially for newer GPUs where the bottleneck is memory bandwidth (how quickly you can send data to a GPU) rather than compute (how quickly your GPU can crunch floating point operations).

PyTorch is the work of developers at Facebook AI Research and several other labs. The framework combines the efficient and flexible GPU-accelerated backend libraries from Torch with an intuitive Python frontend that focuses on rapid prototyping, readable code, and support for the widest possible variety of deep learning models. Pytorch lets developers use the familiar imperative programming approach, but still output to graphs. It was released to open source in 2017, and its Python roots have made it a favorite with machine learning developers.



Some important properties:

- A large and vibrant community at Pytorch.org with excellent documentation tutorials.
- Integrated with python libraries like NumPy,SciPy, Cython.
- Well supported by major cloud platforms
- Scripting language TorchScript is easy to use and flexible when in eager mode

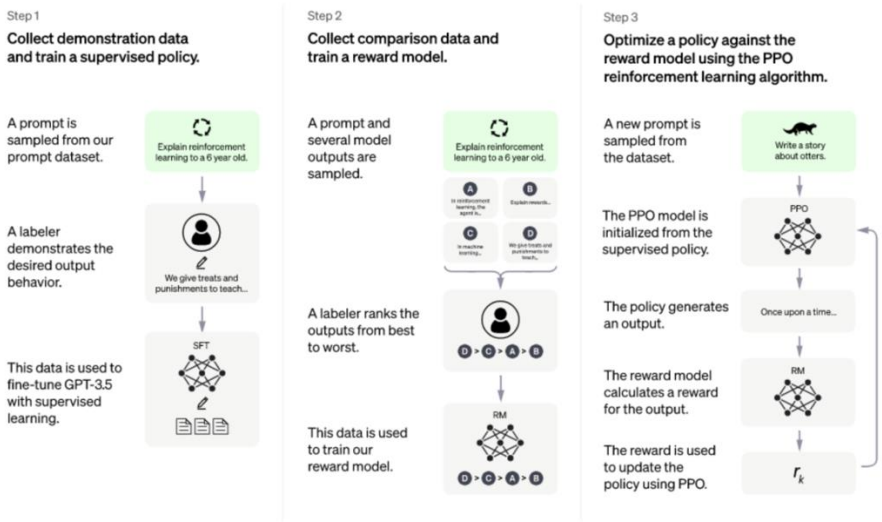
- Supports CPU, GPU and parallel processing
- Supports dynamic computational graphs
- Well regarded set of APIs that can be used to extend core functionality

Both PyTorch and TensorFlow have developed so quicky over their relatively short lifetimes that the debate landscape is ever-evolving. Outdated or incomplete information is abundant, and further obfuscates the complex discussion of which framework has the upper hand in a given domain.
While TensorFlow has a reputation for being an industry-focused framework and PyTorch has a reputation for being a research-focused framework, we'll see that these notios stem partially from outdated information.
In the arena of model availability, PyTorch and TensorFlow diverge sharply. Both have their own official model repositories, but practitioners may want to utilize models from other sources.
When we compare HuggingFace model availability for PyTorch vs TensorFlow, the number of models available for use exclusively in PyTorch absolutely blows the competition where almost 92% of models are PyTorch exclusive up from 85% last year. In contrast only about 8% are TensorFlow exclusive with only 14% of all models available.
For research practitioners, having access to models from recently-published papers is critical. Attempting to recreate new models that you want to explore in a different framework wastes valuable time, so being able to clone a repository and immediately start experimenting saves time.
Currently, TensorFlow wins on deployment. Serving and TFLite are more robust than the PyTorch competitors, and the ability to use tFLite for local AI in conjuction with Google Coral devices is a must-have for many industries. In cocntrast PyTorch Live focuses in mobile only, and TorchServe is still in its infancy. While regularization can increase discernibility in linear models, nonlinear models can introduce correlations among the input variables, which can make it difficult to predict the cause and effect relationship between the inputs and outputs.

ChatGPT

"Hello! I am ChatGPT, a Large Language Model (LLM) developed by OpenAI. I am an artificial intelligence language model that uses deep learning to understand and generate natural language responses to user queries. My purpose is to engage in natural language conversations with users and provide helpful and informative responses to their questions."
As an AI language model, ChatGPT is powered by OpenAI's architecture, which is state-of-the-art language model based on deep learning techniques. GPT is trained using a variant of the Transformer architecture, a type of neural networking that excels at processing sequential data like natural language.
Specifically, GPT utilizes a massive neural network with 175 billion parameters, making it one of the largest language models created.During the training process it learns from a diverse range of text data sources, including books, articles, websites, and other textual materials, available on internet.
The training process involves exposing the model to a vast amount of text data and having it predict the next word or phrase in a given context.By repeatedly training on this task, the models learns the statistical patterns and semantic relationship present in the training data.

Technology behind ChatGPT:
The model was trained using a method called Reinforcement Learning from Human Feedback (RLHF). The training process involved a fine-tuned initial model using supervised learning, with human trainers playing both the role of the user and an AI assistant.These trainers were provided with model-generated suggestions to help them craft their responses. The resulting dialogue dataset was combined with the InstructGPT dataset and transformed into a dialogue format.



To create a reward model for reinforcement learning, the researchers needed to collect comparison data by having human trainers rank different responses generated by the model. They randomly selected a model-written message and sampled several alternative completions, which human trainers then ranked. Using these reward models, the model was fine-tuned using Proximal Policy Optimization, and several iterations of this The fine-tuning process leveraged both supervised learning as well as reinforcement learning in a process called reinforcement learning from the human feedback process performed. Both approaches use human trainers to improve the model's performance. In the case of supervised learning, the model was provided with conversations in which the trainers played both sides: the user and the AI assistant.

ChatGPT initially used a Microsoft Azure supercomputing infrastructure, powered by Nvidia GPUs, that Microsoft built specifically for OpenAI and that reportedly cost "hundreds of millions of dollars". Following the success of ChatGPT, Microsoft dramatically upgraded the OpenAI infrastructure in 2023. ChatGPT is an extrapolation of a class of machine learning Natural Language Processing models known as Large Language Model (LLMs). LLMs digest huge quantities of text data and infer relationships between words within the text. These models have grown over the last few years as we've seen advancements in computational power. LLMs increase their capability as the size of their input datasets and parameter space increase. All GPT models leverage the transformer architecture, which means they have an encoder to process the input sequence and a decoder to generate the output sequence. Both the encoder and decoder have a multi-head self-attention mechanism that allows the model to differentially weight parts of the sequence to infer meaning and context. In addition, the encoder leverages masked-language modeling to understand the relationship between words and produce more comprehensible responses.

Behind the scenes, OpenAI the organization behind ChatGPT, utilizes PyTorch for training and fine-tuning language models like GPT 3.5. PyTorch's capabilities enable efficient computation on GPU's dynamic computation graphs, and a wide range of tools that GPT takes advantage of. Tesla uses PyTorch for Autopilot, their self-driving technology. The company uses PyTorch to train networks to complete tasks for their computer vision applications, including object detection and depth modeling. Microsoft is also a major PyTorch user. It's the primary framework the company uses to develop machine learning solutions to improve Bing search and Xbox, among other products. What's more, Microsoft also owns and develops PyTorch for Windows applications and runs a cloud option of PyTorch through Microsoft Azure. One of the most exciting aspects of ChatGPT is the newly released ChatGPT API, which allows companies to take advantage of the capabilities of artificial intelligence without having to invest significant resources

in developing their own models. This innovation has the potential to transform various industries and create new opportunities for innovation. Companies can now build on top of ChatGPT to develop new tools and services that leverage its powerful language processing capabilities. Chat GPT differs from a search engine in that it generates human-like responses to text prompts, whereas a search engine provides a list of relevant web pages based on keywords. Chat GPT can also engage in natural language conversations, whereas a search engine is not. Chat GPT is a powerful natural language processing tool that has become popular for its ability to generate human-like responses to text prompts. It can be used for various applications, including answering questions, completing tasks, and conversing. By understanding how Chat GPT works and how to use it, users can use its capabilities to enhance their productivity and creativity. Since ChatGPT now offers its own API a lots of projects are poping out related with this subject and there are coming different Machine learning projects related with text processing, image processing and game changing stuffs.

How it connects with the project

The reason that I chose this subject is that we can see a clear connection between the project and machine learning, where in the project we had the training data, the test data from where we can conclude that on the train data, we had x and possible y values which sends us immediately to supervised machine learning. Also, the mathematical formulas that had to be used to take the possible solutions which is a key knowledge that a person needs to have in order to be a machine learning engineer. The analysis the structure and the solutions that are needed to predict are directly related with machine learning techniques, and while coding in the project and searching I got myself stuck on the ways of analyzing data, the ways how solutions are provided, the way how you train a data model and how you handle big data.

References:

https://www.ibm.com/topics/machine-learning
https://www.lightsondata.com/the-history-of-machine-learning/#:~:text=Machine%20learning%20history%20starts%20in,by%20Donald%20Hebb%20is%20published.
https://www.simplilearn.com/tutorials/machine-learning-tutorial/how-to-become-a-machine-learning-engineer
https://www.google.com/books/edition/Python_Machine_Learning/GOVOCwAAQBAJ?hl=en&gbpv=1&printsec=frontcover
https://www.mdpi.com/2078-2489/11/4/193
https://blog.amphy.com/machine-learning-why-use-python/
https://towardsdatascience.com/starting-your-journey-to-master-machine-learning-with-python-d0bd47ebada9
https://maelfabien.github.io/machinelearning/Explorium4/#what-is-hyperparameter-optimization
https://medium.com/jina-ai/build-neural-text-search-engine-in-10-minutes-12d27c51dde1
https://github.com/topics/neural-search
https://documen.tician.de/pycuda/

https://rapids.ai/
https://numba.pydata.org/
https://cupy.dev/
https://www.tensorflow.org/
https://www.techtarget.com/searchenterpriseai/definition/PyTorch#:~:text=PyTorch%20is%20an%20open%20source,platforms%20for%20deep%20learning%20research.
https://pytorch.org/
https://pytorch.org/tutorials/intermediate/scaled_dot_product_attention_tutorial.html?utm_source=whats_new_tutorials&utm_medium=scaled_dot_product_attention_tutorial
https://www.nvidia.com/en-us/glossary/data-science/pytorch/#:~:text=PyTorch%20is%20a%20fully%20featured,developers%20to%20learn%20and%20use.
https://www.tokioschool.com/en/news/machine-learning-fundamentals-python/
https://github.com/tensorflow/tensorflow
https://www.tensorflow.org/about
https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/
https://chat.openai.com/
https://openai.com/blog/chatgpt
https://en.wikipedia.org/wiki/ChatGPT#:~:text=ChatGPT%20is%20an%20artificial%20intelligence,supervised%20and%20reinforcement%20learning%20techniques.
https://techcrunch.com/2023/05/12/chatgpt-everything-you-need-to-know-about-the-ai-powered-chatbot/#:~:text=ChatGPT%20is%20a%20general%2Dpurpose,to%20produce%20human%2Dlike%20text.
https://towardsdatascience.com/how-chatgpt-works-the-models-behind-the-bot-1ce5fca96286#:~:text=ChatGPT%20is%20an%20extrapolation%20of,Large%20Language%20Model%20(LLMs).
https://www.pentalog.com/blog/tech-trends/chatgpt-fundamentals/
https://www.scalablepath.com/data-science/chatgpt-architecture-explained
https://www.awesomescreenshot.com/blog/knowledge/what-is-chat-gpt

Need to be mentioned: Since machine learning is a hot topic right now, and it is a really complicated one, for what there is an organized 2-3 years dedication fields of study, most of the paragraphs are taken from internet for what I have the sections references, from links where I took parts off, to make a well organized document about this subject.

Here comes the assignment:

```python
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
import os
from sqlalchemy.orm import declarative_base


BASE_DIR = os.path.dirname(os.path.realpath(__file__))
connection_string = "sqlite:///" + \
    os.path.join(BASE_DIR, 'database', 'python_assignment.db')
Base = declarative_base()


class Connection:
    def __init__(self):
        '''
        Initializes database using sqlalchemy
        No params passed on this constructor
        Takes datas declared up
        '''
        self.engine = create_engine(connection_string, echo=True)
        self.Session = sessionmaker(bind=self.engine)
        self.session = self.Session()
        self.base = Base

    def dropTable(self):
        '''
        Drops all tables from database
        No parameters provides
        Void method
        '''
        self.base.metadata.drop_all(self.engine)

    # model methode for overriding
    def createTable():
        '''
        The method used for override
        '''
        print('create table')

    # model methode for overriding
    def initializeDatas():
        '''
        The method used for override
        '''
        print('initialize data')
```

```
import sys
from connection import Connection
import pandas as pd
import unittest


class IdealData(Connection):
    def __init__(self):
        '''
        Constructor of this class
        It initialises base class Connection
        '''
        super().__init__()

    def createTable(self):
        '''Create table method overided from Connection'''
        self.base.metadata.create_all(self.engine)

    def initializeDatas(self):
        '''
        Initializes ideal datas on ideal_data table
        It replaces them if it exists
        No params needed
        Void method
        Reads data from the file ideal.csv under dataset folder
        '''
        try:
            df = pd.read_csv('dataset/ideal.csv')
            self.createTable()
            df.to_sql('ideal_data', self.engine,
                      if_exists='replace', index=False)
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))

    def loadDatas(self):
        '''
        Used to load datas from database
        No params provided
        Returns all rows from ideal_data table
        '''
        try:
            return pd.read_sql_table(
                'ideal_data', self.engine.connect())
        except:
```

```
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))


class UnitTestIdealData(unittest.TestCase):
    def test_loadDatas(self):
        idealData = IdealData()
        result = idealData.loadDatas()

        self.assertEqual(result.iloc[0]['y1'], -0.9129453)
```

```
import sys
import pandas as pd
from trainData import TrainData
from idealData import IdealData
import numpy as np
import unittest


class IdealFunctions:
    def __init__(self):
        '''This constructor does not initializes anything'''
        pass

    def initializeDatas(self):
        '''
        Method to initialize datas on database and read from database
        No params are provided
        Returns trainData and ideal datas as dataframes loaded from database
tables
        '''
        try:
            trainData = TrainData()
            trainData.initializeDatas()
            idealData = IdealData()
            idealData.initializeDatas()

            trainDf = trainData.loadDatas()
```

```
            idealDf = idealData.loadDatas()

            return trainDf, idealDf
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))

    def findFunctions(self):
        '''
        Finds the ideal functions comparing training data and ideal datas as
best fit on how they minimize the sum of all y deviation squared
        No arguments provided
        Returns founded functions as frame, trainDf (train dataframe) and
idealDf(ideal dataframe)
        '''
        try:
            result = self.initializeDatas()
            trainDf, idealDf = result

            # Extract y values from training dataset
            train_y1 = trainDf['y1'].values
            train_y2 = trainDf['y2'].values
            train_y3 = trainDf['y3'].values
            train_y4 = trainDf['y4'].values

            ideal_x = idealDf['x'].values

            # Extract y values from ideal functions dataset
            ideal_y = idealDf.iloc[:, 1:].values

            deviations = np.zeros(50)

            # find deviations based on 50 functions
            for i in range(50):
                ideal_yi = ideal_y[:, i]
                A = np.vstack([train_y1, train_y2, train_y3, train_y4]).T
                b = ideal_yi
                x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)
                deviations[i] = np.sum((np.dot(A, x) - b)**2)

            # Choose the four ideal functions with the smallest deviation
            idx = np.argsort(deviations)[:4]
            ideal_chosen = ideal_y[:, idx]

            frame = pd.DataFrame(ideal_chosen, columns=[
```

```
                                'y1', 'y2', 'y3', 'y4'])
            frame['x'] = ideal_x

            return frame, idealDf, trainDf
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))


class UnitTestIdealFunctions(unittest.TestCase):
    def test_initializeDatas(self):
        idealFunctions = IdealFunctions()
        trainDf, idealDf = idealFunctions.initializeDatas()

        self.assertEqual(trainDf.iloc[0]['x'], -20)
        self.assertEqual(idealDf.iloc[0]['y1'], -0.9129453)

    def test_findFunctions(self):
        idealFunctions = IdealFunctions()
        frame, trainDf, idealDf = idealFunctions.findFunctions()

        self.assertEqual(trainDf.iloc[0]['x'], -20)
        self.assertEqual(idealDf.iloc[0]['y1'], 0.052657526)
        self.assertEqual(frame.iloc[0]['y1'], -0.008333334)
```

```
import sys
import pandas as pd
import math
from connection import Connection
import unittest
from idealFunctions import IdealFunctions


class TestData(Connection):
    def __init__(self, idealFunctions):
        '''
        Constructor of this class
        idealFunctions: Ideal functions dataframe provided from previous
```

```
calculation found best 4
        '''
        self.idealFunctions = idealFunctions
        super().__init__()

    def createTable(self):
        '''
        Creates database if not exists
        No argument provided
        Type void
        '''
        self.base.metadata.create_all(self.engine)

    def initializeDatas(self, data):
        '''
        Initializes datas on database as table test_data_deviation
        data: Data to be inserted after calculation are made(dataframe)
        Type void
        '''
        data.to_sql('test_data_deviation', self.engine,
                    if_exists='replace', index=False)

    def compare(self):
        '''
        Load test datas row by row and compares with idealFunctions to find
if it passes
        No parameters provided
        Returns dataframe with the functions that passed the conditions
        '''
        try:
            testData = pd.read_csv('dataset/test.csv')
            datasPassed = []
            for index, row in testData.iterrows():
                found = self.idealFunctions[self.idealFunctions['x'] ==
row['x']]
                maxValue = found.filter(like='y').iloc[0].max()
                if row['y'] / maxValue <= math.sqrt(2):
                    rowAppended = {'x': row['x'], 'y': row['y'], 'delta_y':
maxValue,
                                   'number_of_function': found.index[0]}
                    datasPassed.append(rowAppended)

            df = pd.DataFrame(datasPassed)
            return df
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,
```

```
exception_value))


class UnitTestTestData(unittest.TestCase):
    def test_compare(self):
        idealFunctions = IdealFunctions()
        frame, idealDf, trainDf = idealFunctions.findFunctions()
        testData = TestData(frame)
        result = testData.compare()

        self.assertEqual(result.iloc[0]['x'], -5)
```

```
# python-turnitin

Clone Project
`git clone git@github.com:enisbe1/python-turnitin.git` using SSH
`git clone https://github.com/enisbe1/python-turnitin.git` using HTTP

Move to develop branch
`git checkout develop`

Get all the datas
`git pull origin develop`
All ready for developing on your local personal computer

Now assume that you want to add something new to the code
First we create a branch based on what we want to add if new feature or a fix
The branch names are prefered to follow the structure `feat/feature-name`,
`fix/fix-name`
After creating the new branch we start to code, when finished
use
`git add .` to add all files in the stage
`git commit -m "Message typed here"` to commit the changes, message should be
short and meaningfull
And in the end we use
`git push origin branch-name` to push the changes to the repository
We create a pull request with the branch develop which should be merged after
being reviewed by colleagues
```

```
import sys
from connection import Connection
import pandas as pd
import unittest


class TrainData(Connection):
    def __init__(self):
        '''
        Constructor of this class
        It initialises base class Connection
        '''
        super().__init__()

    def createTable(self):
        '''
        Creates database if not exists
        No parameter provided
        Type void
        '''
        self.base.metadata.create_all(self.engine)

    def initializeDatas(self):
        '''
        Initializes datas on database under table train data by reading
train.csv under dataset folder
        No parameter provided
        Void method
        '''
        try:
            df = pd.read_csv('dataset/train.csv')
            super().dropTable()
            self.createTable()
            df.to_sql('train_data', self.engine,
                      if_exists='replace', index=False)
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))

    def loadDatas(self):
        '''
        Loads data from train_data table
        No parameter provided
        Returns train_data table with the rows it contains as dataframe
        '''
```

```
        try:
            return pd.read_sql_table(
                'train_data', self.engine.connect())
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))


class UnitTestTrainData(unittest.TestCase):
    def test_loadDatas(self):
        trainData = TrainData()
        trainData.initializeDatas()
        result = trainData.loadDatas()

        self.assertEqual(result.iloc[0]['x'], -20)









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

{
  "folders": [
      {
            "path": "."
      }
],
  "settings": {}
}
```

```
import sys
from bokeh.plotting import figure, output_file, show
from bokeh.layouts import column
from bokeh.models import ColumnDataSource
import unittest
from idealFunctions import IdealFunctions
from testData import TestData

output_file("layout.html")
colors = ['red', 'green', 'blue', 'yellow', 'purple']


class Vizualization:
    def __init__(self, trainDf, idealDf, frame, datasToSave):
        '''
        Constructor of this class, initializes the parameters provided when
called
        trainDf: Train dataset loaded from database
        idealDf: Ideal dataset loaded from database
        frame: Datas calculated as maximum deviation
        datasToSave: Compared with test data and only the fitted ones in
calculations
        '''
        self.trainDf = trainDf
        self.idealDf = idealDf
        self.frame = frame
        self.datasToSave = datasToSave
        pass

    def visualizeFrame(self, dtFrame, title):
        '''
        It visualizes a single frame using bokeh
        dtFrame: Dataframe for visualization with circles
        title: Title for the dataframe
        Return the graph created by given dataframe
        '''
        try:
            source = ColumnDataSource(dtFrame)
            p = figure(title=title,
                       x_axis_label='X', y_axis_label='Y')
            for i, col in enumerate(dtFrame.columns[1:]):
                p.circle('x', col, source=source,
                         color=colors[i % 5], size=1)

            return p
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
```

```
{}".format(exception_type,

exception_value))

    def vizualize(self):
        '''
        Vizualizes the graphs in column
        No parameters provided
        Type void
        '''
        try:
            trainGraph = self.visualizeFrame(self.trainDf, 'Train Data')
            idealGraph = self.visualizeFrame(self.idealDf, 'Ideal Data')
            frameGraph = self.visualizeFrame(self.frame, 'Ideal Functions')
            datasToSaveGraph = self.visualizeFrame(self.frame, 'Ideal Data')

            show(column(trainGraph, idealGraph, frameGraph,
datasToSaveGraph))
        except:
            exception_type, exception_value, exception_traceback =
sys.exc_info()
            print("Exception Type: {}\nException Value:
{}".format(exception_type,

exception_value))


class UnitTestVizualize(unittest.TestCase):
    def test_visualizeFrame(self):
        idealFunctions = IdealFunctions()
        frame, idealDf, trainDf = idealFunctions.findFunctions()
        testData = TestData(frame)
        datasToSave = testData.compare()
        vizualization = Vizualization(trainDf, idealDf, frame, datasToSave)
        result = vizualization.visualizeFrame(trainDf, 'trainData')

        self.assertIsNotNone(result)
```

```python
from idealFunctions import IdealFunctions
from testData import TestData
from visualization import Vizualization


def main():
    '''
    Main method to start the application
    Calls classes with logical order to execute each one of them
    On program execution this method is called
    No return on this method
    '''
    idealFunctions = IdealFunctions()
    frame, idealDf, trainDf = idealFunctions.findFunctions()
    testData = TestData(frame)
    datasToSave = testData.compare()

    visualization = Vizualization(trainDf, idealDf, frame, datasToSave)
    visualization.vizualize()


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

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos into
it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
```

```
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
#   For a library or package, you might want to ignore these files since the
code is
#   intended to run in multiple environments; otherwise, check them in:
# .python-version

# pipenv
#   According to pypa/pipenv#598, it is recommended to include Pipfile.lock
in version control.
#   However, in case of collaboration, if having platform-specific
dependencies or dependencies
#   having no cross-platform support, pipenv may install dependencies that
don't work, or not
```

```
#   install all needed dependencies.
#Pipfile.lock

# poetry
#   Similar to Pipfile.lock, it is generally recommended to include
poetry.lock in version control.
#   This is especially recommended for binary packages to ensure
reproducibility, and is more
#   commonly ignored for libraries.
#   https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-
to-version-control
#poetry.lock

# pdm
#   Similar to Pipfile.lock, it is generally recommended to include pdm.lock
in version control.
#pdm.lock
#   pdm stores project-wide configurations in .pdm.toml, but it is
recommended to not include it
#   in version control.
#   https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-
project/pdm
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject
```

```
# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
#  JetBrains specific template is maintained in a separate
JetBrains.gitignore that can
#  be found at
https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
#  and can be added to the global gitignore or merged into this file.  For a
more nuclear
#  option (not recommended) you can uncomment the following to ignore the
entire idea folder.
#.idea/
```

https://github.com/enisbe1/python-turnitin