Lesson 1

**Few-shot learning aims to replicate the remarkable human capacity to identify novel objects within a category of similar items using only a limited number of examples. In contrast, conventional learning models rely on extensive datasets for training, which can be costly to assemble on a large scale. Additionally, these models excel at capturing insights from infrequent data instances, where only a handful of examples are accessible.**

**Zero-shot learning** refers to the capability of a model to acquire knowledge from data without direct exposure to it, effectively achieving learning without any prior access to it.

**Google Colab** is a cloud-based platform provided by Google that allows users to write, execute, and collaborate on Python code using Jupyter notebooks. It provides free access to GPU and TPU resources, making it a popular choice for machine learning and data science projects.

**Co:Here** is a company that provides natural language processing models to help improve human-machine interactions.

Machine learning can be classified into broad categories:

1. The type of supervision during training can vary, including supervised, unsupervised, semi-supervised, self-supervised, and other methods.
2. Some machine learning systems can learn incrementally on the fly through online learning, while others require batch learning.
3. Machine learning systems can work by comparing new data points to known data points (instance-based learning) or by detecting patterns in the training data and building a predictive model (model-based learning).

Lesson 2

* It’s important to take into account the business context of a machine learning application.
* Before starting, select an appropriate performance measure.
* Use descriptive statistics to gain an understanding of the data structure.
* Utilize visualizations and correlations to explore relationships within the data.
* Set aside a test set before beginning to work with the data.
* Combine attributes to uncover stronger correlations.
* Data cleaning and transformation often require the most effort.
* Numerical attributes should be transformed to the same scale before applying machine learning algorithms through standardization or min-max scaling.
* Categorical attributes can be transformed using one-hot encoding or numerical transformation.
* Skewness in the data can be addressed by applying a logarithmic transformation.
* Cross-validation can help in selecting the best model.
* Hyperparameters can be tuned using grid search or random search.
* Monitor the performance of the system and its inputs regularly.
* We know there’s an overfitting problem because the training error is low (actually zero) while the validation error is high.
* The SKLearn (Sci-kit learn) framework provides powerful tools for data transformations, metrics, and data pipelines.
* Sci-Kit Learn is remarkably well designed and it’s principles are summarized in this paper: [[1309.0238] API design for machine learning software: experiences from the scikit-learn project (arxiv.org)](https://arxiv.org/abs/1309.0238)
* Stratified sampling can help preserve the characteristics of the dataset when splitting it into training and test sets.
* Missing values can be imputed using methods such as median imputation or by dropping columns or rows.
* The Mathplotlib framework provides powerful visualization tools for data exploration.
* Evaluate multiple models to select the best one using validation techniques.
* The Joblib framework can be used to save and retrieve models.

Summary of chapter II from Hands on Machine learning book:

Steps for ML project:

1. Look at the big picture.

2. Get the data.

3. Explore and visualize the data to gain insights.

4. Prepare the data for machine learning algorithms.

5. Select a model and train it.

6. Fine-tune your model.

7. Present your solution.

8. Launch, monitor, and maintain your system.

• Popular open data repositories:

—OpenML.org

—Kaggle.com

—PapersWithCode.com

—UC Irvine Machine Learning Repository

—Amazon’s AWS datasets

—TensorFlow datasets

• Meta portals (they list open data repositories):

—DataPortals.org

—OpenDataMonitor.eu

• Other pages listing many popular open data repositories:

—Wikipedia’s list of machine learning datasets

—Quora.com

—The datasets subreddit

1. Frame problem:

* What is the business objective of it all?
* What the current solution looks like?

A sequence of data processing components is called a data pipeline

This is clearly a typical supervised learning task, since the model can be trained with labeled examples.

Examples are labeled

This is a univariate(single dependent variable), multiple (mutiple indip. variables), regression(predicting) task

Performance measures:

* Root mean square error (RMSE): A picture containing font, text, handwriting, line

  Description automatically generated
  + Used if not many outliers
  + *m = number of instances*
  + **xi** = Vector af all features of instanacei (excl. label)
  + **X** = matrix of all feature of all instances. = Matrix composed of *ith* row = transpose of **xi**
  + *h*(ypothesis) is your system’s prediction function --> predicted value ŷ(i) = h(x(i))
  + RMSE(X,h) is the cost function
* mean absolute error:
* If many outliers A picture containing text, font, white, handwriting

  Description automatically generated

the ℓk norm of a vector v containing n elements is defined as ∥v∥k = (|v1|k + |v2|k + ... + |vn|k)^1/k. ℓ0 gives the number of nonzero elements in the

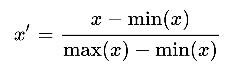
vector, and ℓ∞ gives the maximum absolute value in the vector.

The higher the norm index, the more it focuses on large values and neglects small ones.

1. Check the assumptions

Recheck if the assumptions are right. E.g. check if you are really working on a regression task, and not something that later gets classified in categories (classification task)

1. Get the Data
2. Create train set
   1. Before the brain detects a pattern in the data -> *data snooping* bias = overly optimistic estimation using the test set
   2. Need to always generate the same shuffled train set and not iterate over whole data
      1. Common solution --> use instance identifier as hash (set same number for random generator), always select same 20% from new information
      2. Alternative--> Save train set and load when needed
      3. Neither working if data updated
   3. Better solution: use each instance’s identifier if unique, if new data added => 20% of it is put in training\_set
   4. *train\_test\_split()* most common function in Scikit-learn to split test set => shorter function
   5. Purely random sampling can lead to non-representative sample
      1. => Stratified sampling => Dividing in categories, then sample from each category (should not have too many strata, and each stratum should be large enough!!)
   6. ScikitLearn provides a number of splitter classes in the *sklearn.model\_selection* package => various splitting strategies
3. Look for correlations
   1. .corr() function => Pearson's correlation coefficient
   2. scatter\_matrix() => matrix of correlations plots
   3. Build variables combinations (E.g. rooms / house)
4. Prepare data for ML
   1. Write functions to transform the data
   2. If missing values: Delete row, delete column, impute => SimpleImputer from Scikit
   3. Scikit OrdinalEncoder => Convert categories to numbers
   4. Scikit OneHotEncoder => Convert categorical => one-hot vectors
      1. Sparse matrix: containing only 1 and their position. Not 0s per se, they are assumed where no 1s. More efficient…
   5. Feature scaling and transformation
      1. ML Algorithms don't work well with data different scales
         1. Min-max (*normalization*) {0,1} => *MinMaxScaler* (Scikit)



* + - 1. Standardization, not restricted to range, much less affected by outliers => StandardScaler(Scikit)
    1. Fit scalers to training\_set only
    2. Training\_set values will always be scaled to range , if new data contains outliers => Might end outside range
       1. To avoid this => parameter *clip* = True
    3. ML doesn't work well with heavy tails => shrink tail beforehand / make distribution symmetrical
       1. Apply log / power of ^{0-1}
       2. Bucketizing feature: Chopping distrib. Into equal sized buckets => replace each feature with [index]bucket it belongs to
          1. If multi-modal => bucketize into categories, not values

Add feature for each mode => radial basis function = lower value if further from data. Scikit rbf\_kernel()

* + - * 1. If we transform target data => needs to be reverted back to normal: Scikit *inverse\_transform()*
        2. Faster => *TransformedTargetRegressor*
      1. Custom transformers
         1. No training required => write custom functions
      2. Pipelines:
         1. *Pipeline class*
         2. *make\_pipeline()*
         3. You can index the estimator[i] in the pipeline
         4. *ColumnTransformer()* applies the appropriate transform to each column as defined by u (categorical / numerical)

make\_column\_selector() => select all features of given type

* + - 1. Train & evaluate a model:
         1. On the training set
         2. Cross validation => *train\_ test\_split()* OR better: Scikit-Learn’s *k\_-fold cross-validation* => Split train\_set into 10-folds, train on 9 eval on 1 test. Iterate 10 times => return 10 evaluation scores
         3. Test different models and measure their performance
      2. Fine-tune model
         1. *GridSearchCV* class => automatically tune hyperparameters u choose => Takes a while
         2. *RandomizedSearchCV* => Better with many hyperparameters
         3. *HalvingRandomSearchCV* & *HalvingGridSearchCV* => Only test entire hyperparameter space on small training set ("Limited resources") => most significant hyperparameters go to next round on larger resources training\_set => Best hyperparameters evaluated on "*Full resources*"
         4. Ensemble methods => Combine models => better prediction (especially if very different types of errors from each model)
      3. Analyzing best models & errors => Get insights by analyzing the best models
         1. *sklearn.feature\_selection.SelectFromModel* => Automatically drop least useful features
         2. Look at type of error model makes
         3. Ensure model works well on all categories of the training\_set, not only on average
      4. Evaluate system on test-set
         1. Calculate confidence intervals
         2. Often performance slightly lower than in cross-validation => Don't retrain model on entire set
         3. Monitor performance in different ways: inputs, evaluate different versions of model
         4. Backup old models & able to revert quickly if needed
         5. Backup datasets versions, too

Lesson 3

* Kaggle is a popular online platform for data scientists and machine learning practitioners. It hosts competitions, provides datasets, and facilitates a community where users can collaborate, share insights, and showcase their skills in solving data-related challenges.
* Stratified sampling is a sampling technique in statistics where the population is divided into homogeneous subgroups or strata based on certain characteristics. A representative sample is then drawn from each stratum in proportion to its size, ensuring that each subgroup is adequately represented in the final sample, which helps reduce bias and increase the accuracy of the analysis.
* SimpleImputer is a class from the Scikit-learn library in Python that provides a straightforward way to handle missing values in a dataset. It offers various strategies, such as filling missing values with a constant, the mean, median, or most frequent value of the respective feature.

Lesson 4

The recent advancements in artificial neural networks (ANNs) can be attributed to several key factors:

1. The availability of vast amounts of data (commonly referred to as big data) has significantly contributed to the training of ANNs.

2. There has been a remarkable surge in computational performance, aligning with the predictions of Moore's law.

3. The gaming industry's development of powerful GPU cards has proven beneficial for parallel computations involved in training ANNs.

4. Minor adjustments to the training algorithms have resulted in substantial improvements.

5. The practical implications of certain theoretical limitations of ANNs, such as convergence to local optima rather than global optima, have been found to be less problematic than previously assumed.

6. ANNs have entered a positive feedback loop, where successful outcomes attract funding, leading to further advancements, and the cycle continues.

**Perceptron**: The perceptron serves as a fundamental component within a neural network. It encompasses the reception of input signals, assignment of weights and bias, and application of an activation function to generate an output. It constitutes the foundational element in the architecture of neural networks.

**Threshold Logic Unit (TLU)**: The TLU represents a specific variation of the perceptron, characterized by a distinct activation function. Unlike complex functions such as sigmoid or ReLU, the TLU employs a binary step function to produce outputs of either 0 or 1, contingent upon the comparison of the weighted sum of inputs with a predefined threshold. This simplicity of operation yields effective binary classification outcomes.

**Fully Connected Layer**: A fully connected layer, often referred to as a dense layer, establishes comprehensive interconnections between all neurons in the previous layer and the current layer. It achieves this by associating distinct weights with each connection. The layer then performs a weighted summation of inputs, subsequently employing an activation function. This layer constitutes a vital locus of computational activity within neural networks.

**Output Layer**: The output layer denotes the terminal stratum of a neural network, responsible for generating the final predicted outputs. The choice of activation function in this layer hinges upon the nature of the problem at hand, encompassing options such as softmax for multi-class classification or sigmoid for binary classification. It represents the ultimate stage in the processing and inference capabilities of the neural network.

**XOR Problem**: Perceptrons, namely single-layer neural networks, face limitations in solving the XOR problem due to its non-linearly separable nature. The XOR function necessitates a non-linear decision boundary, which cannot be achieved by single-layer perceptrons restricted to linear separations. Consequently, this problem remained beyond the capacity of such network architectures.

**Deep Neural Network (DNN)**: A deep neural network constitutes an advanced form of neural network architecture, distinguished by the presence of multiple hidden layers sandwiched between the input and output layers. These stacked layers empower the network to acquire complex and hierarchical representations of data, enabling it to tackle intricate and intricate problems more effectively.

**Backpropagation**: Backpropagation serves as a pivotal learning algorithm employed in training neural networks. It comprises two fundamental phases. First, the forward propagation stage involves the passage of input data through the network, producing corresponding predictions. Subsequently, the backward propagation phase commences, wherein the error between the predictions and the actual target values is computed and propagated backward through the network. This process enables the update of weights and biases using gradient descent optimization, thereby iteratively minimizing the error and enhancing the network's performance.

Lesson 5

It’s better to use the sigmoid function for gradient descent, instead of the heavyside function, because it is differentiable everywhere.

The **MNIST database** is a comprehensive repository of handwritten digit images. This extensive dataset comprises a substantial collection of thousands of grayscale images representing numerical digits ranging from 0 to 9. It serves as a widely recognized benchmark in the domains of computer vision and pattern recognition, enabling researchers and practitioners to train and evaluate machine learning models for accurate recognition and classification of handwritten digits.

**Mini-batches** are subsets of the training data that are used to update the parameters of the model during training. Instead of using the entire dataset to perform a single update, the dataset is divided into smaller, equally-sized batches. These batches are processed, and the prameters of the model are updated based on the average loss over the mini-batch.  
Using mini-batches has several advantages over using the entire dataset. First, it is computationally more efficient on GPUs because we can use parallelization to process multiple batches at the same time. Second, mini-batches are a compromise betweeen measuring only one instance of loss function and every instance for each pair of output and predictions. Choosing a good batch size is one of the decisions you need to make as a deep learning practitioner to train your model quickly and accurately.  
It is important to shuffle the mini-batches during training because it ensures that each mini-batch is representative of the entire dataset. If the data is not shuffled, the model may see similar examples in each mini-batch, which can lead to poor generalization and overfitting. Shuffling the mini-batches ensures that the model sees a diverse range of examples in each iteration, which can help the model to generalize better.

In machine learning, an **optimizer** refers to an algorithm or technique employed to enhance the performance of a machine learning model during the training phase. Its main purpose is to iteratively adjust the model's parameters, such as weights and biases, based on an error or loss function. This iterative process aims to minimize the error and improve the model's ability to make accurate predictions. By determining the direction and magnitude of parameter updates, the optimizer guides the learning process, helping the model converge towards an optimal solution.  
FastAi provides the SGD class, that does just that.