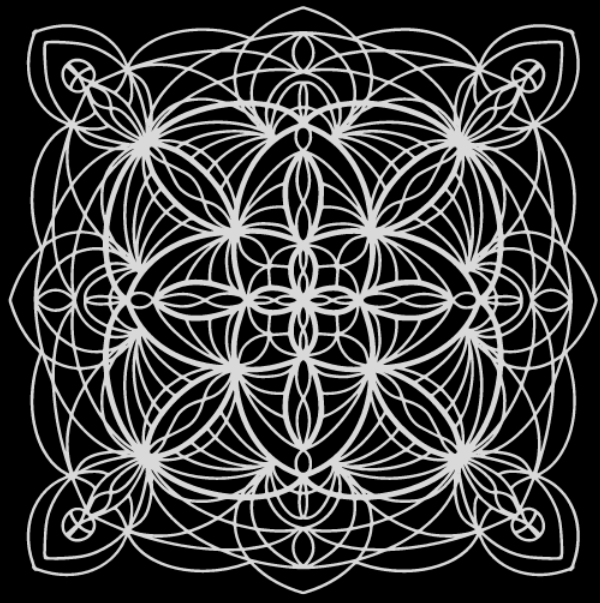
Murat Durmus

A PRIMER TO THE 42 MOST COMMONLY USED MACHINE LEARNING ALGORITHMS

***

(With Code Samples)



Cover design:

Murat Durmus (Mandala: ilona illustrations - canva.com)

About the Author

Murat Durmus is CEO and founder of AISOMA (a Frankfurt am Main (Germany) based company specializing in AI-based technology development and consulting) and Author of the books "Mindful AI - Reflections on Artificial Intelligence".& "INSIDE ALAN TURING"

You can get in touch with the author via:

- LinkedIn: https://www.linkedin.com/in/ceosaisoma/

- E-Mail: murat.durmus@aisoma.de

Note:

The code examples and their description in this book were written with the support of ChatGPT (OpenAI).

“All models are wrong,

but some are useful.”

George E. P. Box

INTRODUCTION

Machine learning refers to the development of AI systems that can perform tasks due to a "learning process" based on data. This is in contrast to approaches and methods in symbolic AI and traditional software development, which are based on embedding explicit rules and logical statements in the code. ML is at the heart of recent advances in statistical AI and the methodology behind technological achievements such as computer programs that outperform humans in tasks ranging from medical diagnosis to complex games. The recent surge of interest in AI is largely due to the achievements made possible by ML. As the term "statistical AI" suggests, ML draws on statistics and probability theory concepts. Many forms of ML go beyond traditional statistical methods, which is why we often think of ML as an exciting new field. However, despite the hype surrounding this technological development, the line between ML and statistics is blurred. There are contexts in which ML is best viewed as a continuum with traditional statistical methods rather than a clearly defined separate field. Regardless of the definitional boundaries, ML is often used for the same analytical tasks that conventional statistical methods have been used for in the past. ML Approaches.

ML is a very active area of research that encompasses a broad and ever-evolving range of methods. Three primary approaches can be distinguished at a high level: **supervised learning**, **unsupervised learning**, and r**einforcement learning**.

Supervised Learning

In supervised learning, the task of the ML algorithm is to infer the value of a predefined target variable (or output variable) based on known values of feature variables (or input variables). The

presence of labeled data (i.e., data with known values for the target in question) is a prerequisite for supervised learning. The learning process consists of developing a model of the relationship between feature and target variables based on labeled training data. This process is also referred to as "model training." After a successful training phase (which is confirmed by a testing phase also based on labeled data), the resulting model can be applied to unlabeled data to infer the most likely value of the target variable. This is referred to as the inference phase.

Supervised learning can solve two main types of analytic problems:

- **Regression problems** where the target variable of interest is continuous. Examples include predicting future stock prices or insurance costs.
- **Classification problems**, where the target of interest is a categorical variable. These include issues where the target variable is binary (e.g., whether a financial transaction is fraudulent or non-fraudulent) and multi-class problems that involve more than two categories. For example, classification can be used to assess the likelihood that customers will default on loan repayments.

Unsupervised Learning

Unsupervised learning involves identifying patterns and relationships in data without a predefined relationship of interest. Unlike supervised learning, this approach does not rely on labeled training data. Therefore, unsupervised learning can be more exploratory, although the results are not necessarily less meaningful.

Unsupervised learning is beneficial when labeled data is unavailable or expensive to produce. This approach can be used to solve problems such as the following:

Cluster analysis involves grouping units of observations based on similarities and dissimilarities between them. Examples of tasks where cluster analysis can be helpful include customer segmentation exercises.

Association analysis, where the goal is to identify salient relationships among variables within a data set. Association rules (i.e., formal if-then statements) typically describe such relationships. These rules can lead to findings such as "customers interested in X are also interested in Y and Z." Association analysis is used for product recommendation and customer service management tasks.

Reinforcement Learning

Reinforcement learning is based on the concept of an "agent" exploring an environment. The agent's task is to determine an optimal action or sequence of steps (the goal of interest) in response to its environment. The learning process does not rely on examples of "correct responses." Instead, it depends on a reward function that provides feedback on the actions taken. The agent strives to maximize its reward and thus improve its performance through an iterative process of trial and error.

Reinforcement learning is practical when the optimal actions (i.e., the correct responses) are unknown. In such situations, labeled training data are not available or risk producing suboptimal results when analysts use supervised learning. The conceptual structure of the approach also makes it relevant for problem types that have a sequential or dynamic nature. Examples include problems in robotics or games.

Much work on reinforcement learning is taking place in the context of basic research. This includes research in general AI. Compared to other ML approaches, reinforcement learning is less common in business. The most noted business applications are outside of financial services and include autonomous vehicles and other forms of robotics. Potential applications in financial services include trading or trade execution and dynamic pricing.

These three approaches include a variety of ML methods such as linear regression, decision trees, support vector machines, artificial neural networks, and ensemble methods. However, two general points about methodological differences are worth noting.

First, ML methods differ significantly in complexity. Discussions of ML often focus on practices with a high degree of complexity. For example, neural networks, a family of techniques that search for patterns and relationships in data sets using network structures similar to those found in the biological brain, receive considerable attention. However, ML also includes fewer complex methods such as ordinary least squares regression and logistic regression. These more straightforward methods have long been used in statistics and econometrics and were established before ML emerged in its current form. We will return to the issue of complexity and its practical implications in later chapters. It should be noted that ML as a field encompasses specific, highly complex methods but is not limited to them.

Second, ML methods can be used to design static or dynamic systems. For static systems, ML is used to develop models that do not evolve once they are deployed unless a new model intentionally replaces them. In dynamic systems, on the other hand, models continue to adapt after deployment based on new data that becomes available during operation.

Such dynamic (or incremental) learning can greatly benefit situations where the data available during development is limited or where models capture phenomena with rapidly changing characteristics.

The Taxonomy used in this book

Main Domain and Data Types

Main Domain	Data Type	Definition
Computer Vision	Image	Visual representation of a pixel matrix consisting of one channel for black and white images, three elements for color images (RGB), or four elements for color images with opacity (RGBA).
	Video	A succession of images (frames), sometimes grouped with a time series (a sound).
NLP / Speech Processing	Text	A succession of characters (e.g., a tweet, a text field).
	Time Series	A series of data points (e.g., numerical) indexed in time order.
Classic Data Science	Structured Data	Data is organized in a predefined array model with a specific column for each characteristic (e.g., text, numeric data, date). To be more precise, structured data refers to organized data found, for example, in a relational database (which, as mentioned, may contain columns of text). Quantitative data can be distinguished from qualitative data. Quantitative data correspond to numeric data that can support some arithmetic operations, while qualitative data are usually used as categorical data to classify data according to their similarities.

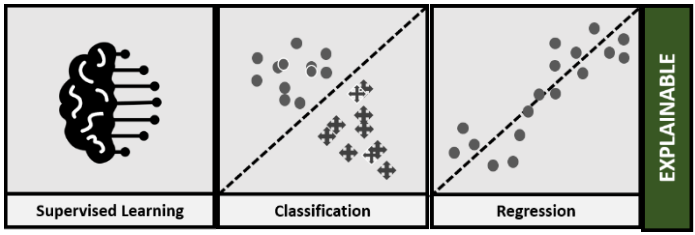
Learning paradigms with subtypes.

Learning Paradigm	Subtype	Definition
Supervised Learning	Classification	Classification is the process of predicting the class of given data points. (Is the picture a cat or a dog?)
	Regression	Regression models are used to predict a continuous value. (Predict the price of a house based on its features).
Unsupervised Learning	Clustering	Clustering is the task of dividing data points into multiple groups so that data points in the same groups are more similar to each other than the data points in the other groups.
	Dimensionality Reduction	Dimensionality reduction refers to techniques for reducing the number of input variables in the training data.
Reinforcement Learning	Rewarding	The reward is an area of ML that deals with how intelligent agents should act in an environment to maximize the notion of cumulative reward by learning from their experiences through feedback.

Explainability

An important aspect of AI security is explainability. Understanding the algorithms and making them explainable makes them accessible to as many people as possible. In addition, explainability helps increase the trustworthiness of AI and supports forensics and analysis of decisions.

ADABOOST



Definition	**AdaBoost uses multiple iterations to create a single composite strong learner by iteratively adding weak learners. In each training phase, a new weak learner is added to the ensemble and a weight vector is adjusted to focus on examples that were misclassified in previous rounds.**
Main Domain	**Classic Data Science**
Data Type	**Structured Data**
Data Environment	**Supervised Learning**
Learning Paradigm	**Classification, Regression**
Explainability	**Explainable**

AdaBoost (Adaptive Boosting) is an ensemble learning algorithm used to improve the accuracy of weak classifiers by combining them into a strong classifier. A classifier is a model that can predict the class or category of input, and a weak classifier is a model that performs better than random guessing but not as well as a strong classifier.

The AdaBoost algorithm works by iteratively training a series of weak classifiers on the data and adjusting the weights of the samples in training set at each iteration. The algorithm assigns higher weights to the samples misclassified by the previous classifiers and lower weights to the samples correctly classified. This process is repeated for a fixed number of iterations or until a stopping criterion is met.

At the end of the process, the algorithm combines the outputs of all the weak classifiers into a final strong classifier. The combination is done by assigning a weight to each weak classifier based on its accuracy. The last strong classifier assigns a class or category to the input by taking a weighted majority vote of the outputs of all the weak classifiers.

AdaBoost is a powerful algorithm that has been used in various applications, including image and speech recognition, object detection, and bioinformatics. It is beneficial when the data is noisy or has multiple features and is resistant to overfitting.

One of the main advantages of AdaBoost is that it can be used with a variety of weak classifiers, including decision trees, neural networks, and support vector machines. It's also simple to implement and computationally efficient. However, it is sensitive to outliers and noise in the data, and it can be affected by choice of weak classifier and the number of iterations.

Example:

Imagine we have a dataset with 100 observations, each with two features (x1 and x2) and a binary label (1 or -1). We want to train a classifier that can predict the label of a new observation based on its features.

1. The algorithm starts by training a weak classifier on the data, for example, a decision stump (a one-level decision tree) that splits the data based on a threshold value of one of the features. This classifier correctly classifies 80 of the observations.
2. Next, the algorithm assigns a weight to each observation based on whether it was correctly or incorrectly classified. The weight of the correctly classified observations is reduced, and the weight of the incorrectly classified observations is increased.
3. The algorithm then trains a second weak classifier on the data using the updated weights. This classifier may be different from the first one; for example, it could use an additional feature or another threshold value. This classifier correctly classifies 85 of the observations.
4. The algorithm assigns new weights to the observations and repeats the process for a fixed number of iterations or until a stopping criterion is met.
5. At the end of the process, the algorithm has trained several weak classifiers on the data, assigning a weight to each classifier based on its accuracy. The final strong classifier is a weighted majority vote of the outputs of all the weak classifiers.

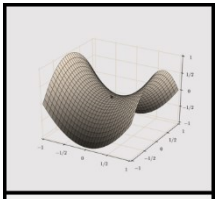
An example of how to use the Adaboost algorithm in Python using the scikit-learn library:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification

# Generate some example data
X, y = make_classification(n_features=4, n_informative=2,
                           n_redundant=0, random_state=0)

# Create an instance of the Adaboost classifier
clf = AdaBoostClassifier(random_state=0)

# Fit the model to the data
clf.fit(X, y)

# Make predictions on new data
predictions = clf.predict(X)
```

In this example, we first import the **AdaBoostClassifier** class from the **ensemble** module of scikit-learn. Then, we use the **make_classification** function to generate example data for the model. Next, we create an instance of the classifier, setting the random state to 0 for reproducibility. Then, we use the **fit** method to train the model on the data and the **predict** method to make predictions on new data.

It's worth noting that the **AdaBoostClassifier** can be used for classification problems. If you want to use Adaboost for regression, you can use the **AdaBoostRegressor** class instead.

ADAM OPTIMIZATION



Optimization

Definition	**Adam optimization is an extension of stochastic gradient descent. It can be used instead of classical stochastic gradient descent to update the network weights more efficiently thanks to two methods: adaptive learning rate and momentum.**
Main Domain	**Classic Data Science**
Data Type	**Structured Data**
Data Environment	**-**
Learning Paradigm	**Optimization**
Explainability	**-**

Adam (Adaptive Moment Estimation) is an optimization algorithm used to update the parameters of a machine learning model during training. It is a popular algorithm used in deep learning and neural networks.

Adam is an extension of the stochastic gradient descent (SGD) algorithm, which is a method to optimize the parameters of a model by updating them in the direction of the negative gradient of the loss function. The Adam algorithm, like SGD, uses the gradients of the loss function concerning the model parameters to update the parameters. In addition, it also incorporates the concept of "momentum" and "adaptive learning rates" to improve the optimization process.

The "momentum" term in Adam is similar to the momentum term used in other optimization algorithms like SGD with momentum. It helps the optimizer to "remember" the direction of the previous update and continue moving in that direction, which can help the optimizer to converge faster.

The "adaptive learning rates" term in Adam adapts the learning rate for each parameter based on the historical gradient information. This allows the optimizer to adjust the learning rate for each parameter individually so that the optimizer can converge faster and with more stability.

Adam is widely used in deep learning because it is computationally efficient and can handle sparse gradients and noisy optimization landscapes. But it requires more memory to store the historical gradient information, and it may be sensitive to the choice of hyperparameters, such as the initial learning rate.

In summary, Adam is a powerful optimization algorithm that can improve the convergence speed and stability of the model during training by incorporating the concepts of momentum and

adaptive learning rates; it's widely used in deep learning and neural networks as it is computationally efficient and can handle noisy optimization landscapes.

Example:

Imagine we have a neural network with two layers, the first layer has four neurons and the second layer has one neuron; the network is used for binary classification. The goal is to find the optimal values for the weights and biases of the neurons that minimize the loss function.

1. The algorithm starts by initializing the weights and biases of the neurons randomly.
2. Next, the algorithm performs a forward pass of the data through the network to calculate the output of the neurons and the loss function.
3. The algorithm then calculates the loss function's gradients for the neurons' weights and biases.
4. The algorithm uses the gradients to update the weights and biases of the neurons using Adam optimization. The update step includes calculating the moving average of the gradients and the squared gradients, which are used to adjust the learning rate for each weight and bias individually.
5. The algorithm repeats steps 2-4 for a fixed number of iterations or until a stopping criterion is met.
6. At the end of the process, the algorithm has found the optimal values for the weights and biases of the neurons that minimize the loss function.

The example I provided is a simplified version of the process; in practice, the neural network may have more layers and neurons, and the dataset may be much more significant. Also, the example

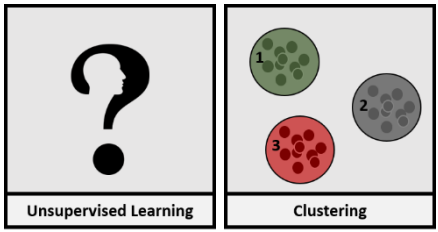
shows the process for a binary classification problem, but the Adam optimization algorithm can be used to optimize any differentiable loss function.

Code example of how to use the Adam optimization algorithm in Python with the Keras library:

```
from keras.optimizers import Adam

# Create a model
model = ...

# Compile the model with Adam optimizer
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=False)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

In this example, the Adam optimizer is being used with the specified learning rate (lr) and beta values, and the model is being compiled with binary crossentropy loss and accuracy metrics. The model is then trained on X_train and y_train data with 10 epochs and a batch size of 32. It's worth noting that this is just one of many ways to code Adam Optimization in Keras and the specific hyperparameter values can be adjusted based on the dataset and the problem at hand.

AGGLOMERATIVE CLUSTERING



Definition	**Agglomerative clustering is a "bottom-up" approach to hierarchical clustering. Each observation starts in its cluster, and cluster pairs are merged as they move up the hierarchy.**
Main Domain	**Classic Data Science**
Data Type	**Structured Data**
Data Environment	**Unsupervised Learning**
Learning Paradigm	**Clustering**
Explainability	**-**

Agglomerative Clustering is a type of hierarchical clustering algorithm. Hierarchical clustering algorithms are a class of algorithms that create a hierarchy of clusters, where each cluster is a subset of the previous one. In contrast, other clustering algorithms like k-means make flat clusters where each point belongs to exactly one cluster.

Agglomerative Clustering starts with each point as an individual cluster, then iteratively merges the closest pair of clusters until all points belong to a single cluster or a stopping criterion is met. The main idea behind agglomerative Clustering is that similar topics are more likely to be in the same cluster, and therefore, the algorithm starts with a large number of small clusters and ends with a small number of large clusters.

One of the critical parameters in Agglomerative Clustering is the linkage criteria, which determines the distance between clusters. Common linkage criteria include:

- Single linkage (the distance between the closest points in each cluster).
- Complete connection (the distance between the farthest points in each cluster).
- Average link (the average distance between all points in each cluster).
- Ward linkage (the minimum variance of distances between all points in each cluster).

Agglomerative Clustering is an efficient and flexible algorithm for clustering data. However, it has some limitations. For example, it does not scale well to large datasets, is sensitive to the linkage criteria, and needs to provide a way to determine the optimal number of clusters. Despite these limitations, it's a widely used

algorithm, and it is used in many applications such as image analysis, bioinformatics, and customer segmentation.

Example:

Imagine we have a dataset with 6 points, represented by the coordinates (x, y) in a two-dimensional space: (1,2), (2,4), (3,5), (4,4), (5,2), (6,1)

1. The algorithm starts by treating each point as an individual cluster. So, we have 6 clusters, each containing one point.
2. Next, the algorithm finds the closest pair of clusters and merges them into a new cluster. The linkage criteria used in this example is "single linkage," which means the algorithm finds the minimum distance between the closest points of each cluster. For example, the closest pair of clusters is (1,2) and (6,1), with a distance of 1.
3. The process is repeated, and the algorithm finds the next closest pair of clusters. In this example, the closest pair is (2,4) and (5,2), with a distance of 2.
4. The algorithm continues to merge clusters until all points are in the same cluster. In this case, the final cluster contains all 6 points and forms a triangle shape.

In this example, the number of clusters is determined by the stopping criterion, which is merging all the points in one cluster. However, in practice, one can use other stopping criteria, such as a maximum number of clusters or a threshold for the linkage distance.

Keep in mind that this is a simple example, and the process can be different depending on the linkage criteria, stopping criteria, and the shape of the data. Also, this example uses a 2-dimensional

space, but the algorithm can work with any dimensions, and the linkage criteria can be adjusted accordingly.

Code example of how to use the scikit-learn library to perform agglomerative clustering in Python:

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs

# Create some sample data
X, y = make_blobs(n_samples=100, centers=3,
random_state=42)

# Initialize the agglomerative clustering model
agg_clustering = AgglomerativeClustering(n_clusters=3)

# Fit the model to the data
agg_clustering.fit(X)

# Predict the cluster labels for each data point
agg_clustering_labels = agg_clustering.labels_

print(agg_clustering_labels)
```

In this example, we first generate a sample dataset of 100 points in 3 clusters using the make_blobs function from the sklearn.datasets module. Then, we initialize an AgglomerativeClustering object with 3 clusters and fit it to the data using the fit() method. Finally, we predict the cluster labels for each data point using the labels_ attribute of the fitted model.