

Unit 01 - Machine Learning Basics (Technical Article)

Arya Sanjay - Interactive Intelligence

1 Introduction to Machine Learning

Machine Learning (ML) is a powerful subset of artificial intelligence (AI) that empowers computers to learn from data and make predictions or decisions without the need for explicit programming. Rather than following static, rule-based instructions, machine learning models identify patterns, structures, and relationships within data. These models generalize from known data to make predictions about new, unseen data, much like how humans learn from past experiences.

Consider the task of recognizing handwritten digits, like those in postal addresses or bank checks. We can train a machine learning model using a dataset of labeled images, where each image corresponds to a digit from 0 to 9. The model learns to recognize the unique patterns of each digit by analyzing features such as curves, edges, and line intersections. Over time, the model refines its ability to correctly identify digits it has never encountered before, thus demonstrating its ability to generalize from the data it was trained on.

2 Data Splitting: Train and Test Sets

Train/Test Split refers to this method of dividing the dataset, typically using an 80-20 or 70-30 ratio. For example, in an 80-20 split, 80% of the data is used for training, and the remaining 20% is held back for testing

The reason we need a test set is to prevent **overfitting**, a common problem where a model becomes too tailored to the training data, performing well on it but poorly on new data. By evaluating on the test set, we can gauge whether the model has generalized well beyond the data it was trained on.

In practice, the following steps are often taken when working with train/test splits:

- **Step 1: Data Splitting.** Split the data into training and test sets before training the model. This prevents any information from the test set from leaking into the model.

- **Step 2: Model Training.** Use the training set to build the machine learning model by adjusting weights, minimizing errors, or finding patterns.
- **Step 3: Model Evaluation.** Once the model is trained, evaluate it on the test set. Common evaluation metrics include accuracy, precision, recall, and mean squared error, depending on the type of model.

In some cases, a third subset called a **validation set** is also used. The validation set helps tune hyperparameters and prevent overfitting before final testing on the test set. This train/test split method is fundamental to machine learning as it ensures that models can be assessed on their ability to generalize, avoiding bias toward the data they were trained on.

2.1 Regression and Classification Tasks

Machine learning generally tackles two major types of problems: regression and classification.

Classification is the task of categorizing a set of items into predefined classes. For example, classifying an image as either a “cat” or a “dog.” The output is typically a discrete label, such as “yes” or “no,” or in this case, “cat” or “pig.”

On the other hand, **regression** is about predicting a continuous value. For instance, predicting a person’s weight based on their height is a regression task, where height is the input feature and weight is the predicted continuous value. In multiple regression, multiple features (like height, age, etc.) are used to predict an output, such as house prices or stock market trends.

3 Linear Regression: Regression

3.1 Fundamentals of Logistic Regression

In machine learning, linear regression is one of the most fundamental algorithms. It tries to model the relationship between input features and the target output by fitting a straight line to the data points. The general form of a linear regression model is:

$$\hat{y} = b + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Where:

- \hat{y} is the predicted output,
- x_1, x_2, \dots, x_n are the input features,
- w_1, w_2, \dots, w_n are the weights (coefficients) assigned to each feature,

- b is the intercept.

The goal of linear regression is to find the values of w and b that minimize the difference between the predicted values and the actual target values in the training data. This is often done by minimizing the Sum of Squared Errors (SSE).

3.2 Understanding Sum of Squared Errors (SSE)

The **Sum of Squared Errors (SSE)** is a commonly used metric in regression analysis to quantify how well a model fits the observed data. It measures the total difference (or error) between the actual values (from the dataset) and the predicted values (generated by the model). Specifically, SSE sums the squared differences between each actual value and its corresponding predicted value. By squaring these differences, the metric ensures that all deviations are counted positively (i.e., large errors in either direction, above or below, contribute significantly to the total error). The formula for SSE is:

$$SSE = \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

Where:

- y_i represents the **actual value** for data point i ,
- \hat{y}_i represents the **predicted value** from the model for the same data point,
- m is the **number of data points** in the dataset.

The concept of squaring the errors is essential because it penalizes larger deviations more than smaller ones. In other words, if a prediction is far off from the actual value, its contribution to the SSE will be disproportionately larger, making it easier to identify large errors.

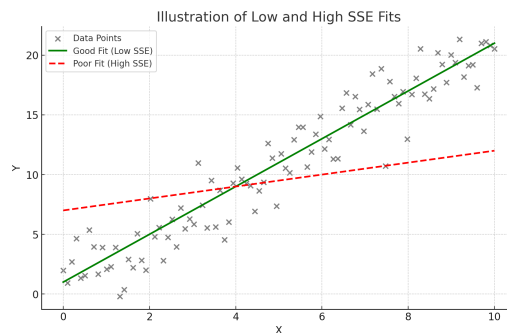


Figure 1: An illustration of two lines of fit that would produce a low and high SSE respectively. The line that fits the data better has a low SSE

The goal in regression is to minimize SSE, because a lower SSE value indicates a better fit of the model to the data. A model with a small SSE suggests that its predictions are close to the actual observed values, indicating that the model is accurate and effectively capturing the patterns in the data.

3.3 Minimizing SSE: Simple Case

Least Squares Regression is one of the most widely used methods in statistics for finding the best-fitting line through a set of data points. The term “least squares” refers to the fact that the method seeks to minimize the **sum of squared differences** (i.e., the SSE) between the observed values and the values predicted by the regression line. The result is a line (or, in higher dimensions, a hyperplane) that minimizes the overall prediction error across all data points.

In simple linear regression, the model is expressed as:

$$\hat{y} = w_0 + w_1x$$

Where:

- w_0 is the **intercept** of the line, representing the predicted value when the input $x = 0$,
- w_1 is the **slope** or weight, representing the change in the predicted value \hat{y} for a one-unit change in the input x ,
- x is the input (or feature) variable, and
- \hat{y} is the predicted value based on the input.

In the least squares method, the goal is to find the optimal values for w_0 and w_1 that minimize the SSE, ensuring that the line fits the data as closely as possible.

The same concept can be extended to multiple regression, where we aim to minimize the SSE by adjusting multiple weights w_1, w_2, \dots, w_n .

Mathematically, this optimization process requires taking the partial derivatives of SSE with respect to the weights (including the intercept) and setting them to zero. This gives us the **normal equations**:

$$w_0 = \frac{\sum y - w_1 \sum x}{n}$$
$$w_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

These equations provide the values of the intercept (w_0) and the slope (w_1) that minimize SSE. Once these values are calculated, the resulting regression line represents the best-fitting line for the data according to the least squares criterion.

4 Logistic Regression: Classification

4.1 Fundamentals of Logistic Regression

Logistic regression, despite its name, is a classification algorithm used to predict a binary outcome—often represented as a “yes/no” or “1/0.” Unlike linear regression, which predicts continuous values, logistic regression is designed to output the probability that a given input belongs to a certain class. For instance, it can help us predict if an email is spam or not based on various features.

The fundamental idea behind logistic regression is to use a **sigmoid function** to transform the output of a linear equation (a combination of input variables) into a probability between 0 and 1. The probability tells us how likely the input belongs to one of two possible classes, usually denoted as 1 (positive class) or 0 (negative class). The sigmoid function is defined as:

$$P(y = 1|x) = \frac{1}{1+e^{-(b+w^T x)}}$$

Where:

- w is the **weight vector**, which defines the importance of each feature in the input.
- b is the **bias term**, which adjusts the overall output of the function.
- e is Euler’s number, and the exponent ensures that the output is squeezed between 0 and 1.

4.2 The Sigmoid Function and Its Role

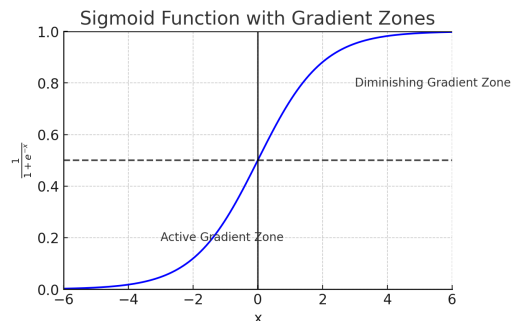


Figure 2: A graph showing the sigmoid function.

The sigmoid function plays a crucial role in logistic regression because it takes any input from the linear equation and converts it into a value between 0 and 1. This

value can be interpreted as a probability, helping us decide which class the input most likely belongs to. Here's how it works:

- If the result of the linear equation ($b + w^\top x$) is a large positive number, the sigmoid function outputs a value close to 1, indicating that the input most likely belongs to class 1.
- If the result is a large negative number, the sigmoid function outputs a value close to 0, indicating class 0.
- If the result is near 0, the sigmoid outputs a value close to 0.5, meaning the input is equally likely to belong to either class.

This probability can then be used to classify the input: if the probability is greater than 0.5, we predict class 1, and if it is less than 0.5, we predict class 0.

4.3 Training with the Log-Likelihood Loss

To make logistic regression work effectively, we need to find the best values for the weights w and bias b . This is done by minimizing the **log-likelihood loss function**. This function measures how well the model's predicted probabilities match the actual class labels. By adjusting the weights and bias to minimize this loss, we ensure that the model produces the most accurate predictions.

During training, the model learns to minimize this loss by using optimization techniques like gradient descent. This ensures that the weights are fine-tuned to give the highest probability for the correct class as often as possible.

4.4 Extensions to Multiclass Classification

While logistic regression is primarily used for binary classification, it can be extended to handle multiple classes through methods such as **one-vs-rest (OvR)**. In this approach, we train multiple logistic regression models, each one responsible for distinguishing whether an input belongs to one particular class or any of the others. The model with the highest probability ultimately decides the final class label.

4.5 Why It's Called "Regression"

The term "logistic regression" comes from its similarity to **linear regression**. Both methods start by calculating a linear combination of the input features. However, while linear regression outputs continuous values, logistic regression uses the sigmoid function to turn these values into probabilities, which are used for classification tasks. This difference is crucial to understanding why logistic regression is a classification algorithm despite having "regression" in its name.

Synthesis Questions:

1. How does the sigmoid function play a role in making Logistic Regression a classification algorithm?
2. For the algorithm to say that the input is equally likely to be from either class, what must $b + w^T x$ have been?

5 K-Means Clustering: Unsupervised Learning

5.1 Introduction to Unsupervised Learning

In contrast to supervised learning, which learns from labeled data, **unsupervised learning** operates on **unlabeled data** and aims to find hidden patterns or structures within it. One powerful method used in unsupervised learning is **k-means clustering**. This algorithm attempts to group data points into k clusters, where each cluster contains points that are similar to each other. Unlike supervised methods where there is a clear “right answer”, the objective here is to let the algorithm uncover these clusters on its own.

5.2 How K-Means Clustering Works

At its core, k-means clustering is about finding the best way to group data points. This process involves iteratively assigning data points to clusters and then recalculating the center of each cluster, known as the **centroid**. The idea is to minimize the distance between data points and their corresponding centroids, so points within a cluster are more similar to each other than to points in other clusters.

The process of k-means clustering can be broken down into the following steps:

- **Step 1: Initialization.** The algorithm begins by randomly selecting k initial cluster centroids from the dataset. These centroids act as the starting points for the clusters.
- **Step 2: Assignment.** Each data point is assigned to the nearest centroid based on a distance metric, typically Euclidean distance. This forms k clusters of data points, with each point belonging to the cluster whose centroid is closest.
- **Step 3: Update.** Once the points are assigned to clusters, the centroids of the clusters are recalculated. The new centroid is the mean of all the data points in the cluster.

- **Step 4: Repeat.** Steps 2 and 3 are repeated until the centroids no longer move significantly, or the cluster assignments do not change. This indicates that the algorithm has converged.

The goal of this process is to minimize the sum of squared distances between each data point and its cluster’s centroid. This ensures that points in the same cluster are as close to each other as possible.

5.3 Minimizing Within-Cluster Distance

In k-means clustering, the objective is to reduce the **within-cluster variance**, or in simpler terms, the total distance between the data points and the centroid of their assigned cluster. The function to minimize is called the **sum of squared errors** (SSE), which measures how far the data points are from their respective cluster centroids:

$$SSE = \sum_{j=1}^k \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

Where:

- x_i represents a data point,
- μ_j is the centroid of cluster j ,
- C_j represents the set of points assigned to cluster j ,
- $\| \cdot \|^2$ is the squared Euclidean distance.

By minimizing the SSE, k-means clustering ensures that data points within the same cluster are tightly packed around their centroid, making the clustering meaningful.

5.4 Choosing the Number of Clusters k

A critical decision in k-means clustering is determining the value of k , or how many clusters the data should be divided into. Often, this is not known in advance, so techniques such as the **elbow method** are used. In the elbow method, the SSE is calculated for various values of k , and the results are plotted. The “elbow” point on the plot represents the optimal number of clusters—beyond this point, increasing k provides diminishing returns in terms of reducing SSE.

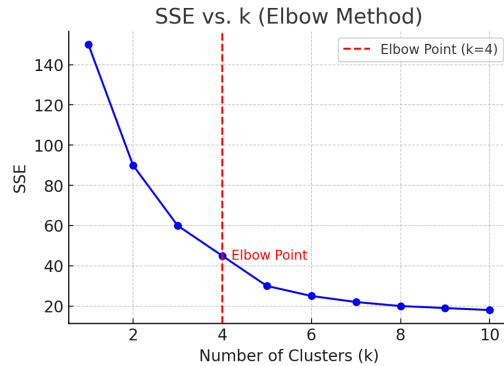


Figure 3: An example of a graph showing SSE vs. k , and the elbow that can be used to pick the optimal k .

Another method to determine k is by using the **silhouette score**, which evaluates how similar a data point is to its assigned cluster compared to other clusters. Higher silhouette scores indicate better-defined clusters.

5.5 Limitations of K-Means Clustering

While k-means is an effective clustering algorithm, it comes with several limitations:

- **Sensitivity to Initialization.** The algorithm is highly sensitive to the initial placement of the centroids. Poor initial choices can result in suboptimal clusters or even cause the algorithm to converge to a local minimum. To address this, a more refined initialization method, **K-Means++**, is often used to improve the initial centroids.
- **Fixed Number of Clusters.** K-means requires that k be defined ahead of time, which might not always be clear from the dataset. This adds complexity to its usage.
- **Sensitivity to Outliers.** K-means clustering can be significantly impacted by outliers. Since the algorithm seeks to minimize distances, outliers (data points far from the rest of the data) can disproportionately affect the cluster centroids.

Despite these limitations, k-means remains a widely used and powerful clustering technique for discovering patterns in data, particularly when the data is well-structured and the number of clusters is approximately known.

Synthesis Questions:

1. Define “Unsupervised Learning” and how it is different than “Supervised Learning”.
2. What is a centroid? How is it calculated?
3. If the calculated centroids do not move very much after a few iterations, what does this indicate?
4. If you have n points to cluster, what value of k will give the lowest SSE no matter how the points are formed? (**Hint:** It’s kind of a cheap, degenerate case!)
5. Why do you think k-means clustering is sensitive to centroid initialization?

6 K-Means vs. K-Medians Clustering

6.1 Introduction to K-Medians Clustering

While **k-means** is a widely used method for clustering, an alternative approach is **k-medians clustering**, which offers a different way of calculating cluster centers. Instead of using the mean of the data points to define the cluster center as k-means does, k-medians calculates the center as the **median** of the data points in the cluster. This key difference makes k-medians more robust, particularly in the presence of **outliers**, as the median is less sensitive to extreme values. In k-medians, the objective is to minimize the sum of **absolute differences** between data points and the cluster center, rather than the sum of squared distances. This approach results in cluster centroids that are less influenced by anomalous data points, providing a more stable representation of the typical cluster member in datasets with noisy or skewed distributions.

6.2 When to Use K-Medians

K-medians clustering is especially useful in situations where the data contains significant **outliers** or when using non-Euclidean distance metrics. Unlike k-means, which can be overly influenced by extreme values due to its reliance on the mean, k-medians offers a more **robust** solution by focusing on the median. For example, if a dataset contains several unusually large or small values that could distort the cluster centroid under k-means, k-medians will provide a more accurate and representative clustering result.

K-medians is typically used when minimizing the **sum of absolute differences** between points is more meaningful than minimizing squared differences, such as in scenarios where data points vary widely in scale or when outliers are frequent. While k-medians may be less common than k-means in practice, it proves highly

effective when the assumptions behind k-means (such as normal distribution of data or minimal outliers) do not hold.

6.3 Key Differences Between K-Means and K-Medians

The fundamental difference between **k-means** and **k-medians** lies in how they compute the cluster centroids and the distance metrics they use. K-means calculates the centroid as the **mean**, minimizing the sum of squared Euclidean distances. In contrast, k-medians uses the **median**, minimizing the sum of absolute distances (often referred to as Manhattan distance).

Here are key distinctions between the two methods:

- **K-Means:** More sensitive to outliers because the mean is influenced by extreme values. It is best suited for datasets where the distance between points is naturally measured using Euclidean distances, and when the data does not contain many outliers.
- **K-Medians:** More robust to outliers, as the median remains stable even in the presence of extreme values. It is better suited for datasets with skewed distributions or where minimizing absolute differences is a priority.

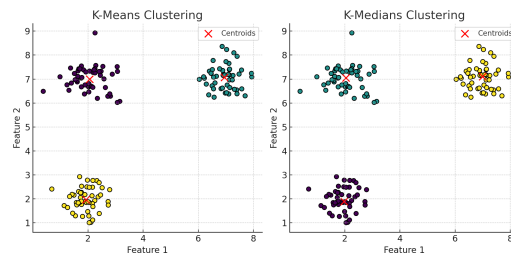


Figure 4: An image comparing and contrasted the clusters created from k-means and k-medians methods.

The choice between the two methods depends heavily on the nature of the dataset. K-means is computationally more efficient but may produce distorted clusters in datasets with many outliers, whereas k-medians offers a more resilient solution for datasets with irregularities or noise.

Synthesis Questions:

1. Give a concrete, real-world example of a situation where *k-means* clustering is a better choice than *k-medians* clustering
2. Give a concrete, real-world example of a situation where *k-medians* clustering is a better choice than *k-means* clustering
3. Give a concrete, real-world example of a situation where *neither* of these algorithms will work well

7 Other ML Concepts

7.1 Bias-Variance Tradeoff

This is a concept that doesn't really fit anywhere else, but is incredibly important to talk about. The bias-variance tradeoff heavily rates to over and underfitting. **Overfitting** is when you have a model that fits the data it is trained on *too* well. This could come about by, for example, trying to fit a very high-order polynomial curve to data that is linearly correlated. Instead of creating a line of best fit through the data, a sufficiently complex polynomial could just weave through every point in the graph. This would result in 0 SSE, but terrible performance on real-world data after the model is deployed. This is because real data is noisy, and a sufficiently complex model will fit to that noise because doing that technically minimizes SSE. **Underfitting** is the exact opposite problem. Say you had some synthetic data that took the shape of a parabola, but tried to fit a straight line to it. No matter how many training examples you provide, your model is not complex enough (you can also say it is not *expressive* enough) to fit to the curve well.

An example of a very basic, not-so-expressive linear model:

$$y = w_1x_1 + b$$

An example of a very expressive 5th-degree polynomial model:

$$y = w_1x_1 + w_2x_1^2 + w_3x_1^3 + w_4x_1^4 + w_5x_1^5 + b$$

A rarer, but still useful sinusoidal model:

$$y = w_1\sin(w_2x_1 - w_3) + b$$

When people talk about the bias-variance tradeoff, they are essentially referring to over and under fitting, but are being a little more specific. **Bias** refers to how close

the model line is to the “true” underlying curve. You can generate synthetic data for a line by generating points from some $y = mx + b$, but then adding a noise term $\epsilon \sim N(0, 1)$ to each point. If your underlying model fits to the generating $y = mx + b$ but ignores the noise introduced with ϵ , then it has *low bias*. **Variance** is a bit more complicated, but it essentially concerns the robustness of the model. If you were to add or remove a few points from a dataset, then a model with the appropriate level of expressivity should not change its learned coefficients by much. For example, if you had data correlating age with height, you shouldn’t expect removing one or two points to affect the underlying correlation. A overly expressive model would, however, completely refit to swerve through all the points in a new way which drastically changes the coefficients. Variance measures how inconsistent the model is across different training subsets. When fitting ML models, you want a model that minimizes both bias *and* variance. To do this, you must pick a ML model that has enough expressive power to capture underlying trends, but is not so expressive that it suffers from high variance errors.

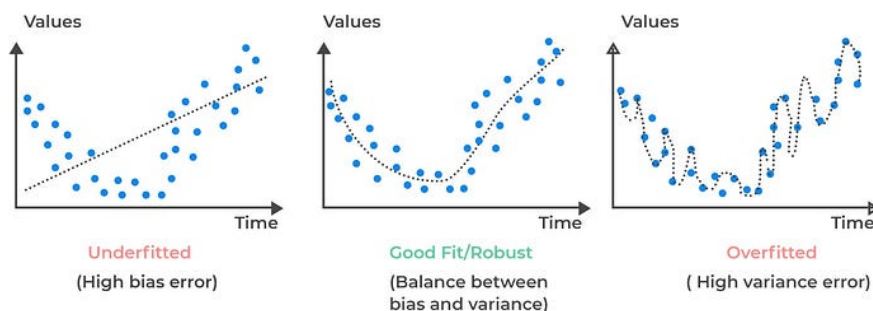


Figure 5: An illustration visualizing different balances between bias and variance for model fitting.

7.2 Convexity

Convexity is important for ML models as well. So far, the linear models we have shown you are all **convex**. What this means is that we can guarantee we are able to converge upon an optimal set of weights if we train enough. You can think of optimizing a linear model like rolling down a valley. The lower you are on this valley, the lower the SSE. You take steps by fitting the model to training examples. Once you reach the bottom of this valley, you are satisfied as you cannot get any lower. Moving in any direction either increases your SSE or keeps it the same. *If you are sure that the function you are optimizing is convex, then you can rest assured that you have found an optimal set of weights.* However, if the function you are optimizing on is non-convex, there is a chance that a lower valley exists

somewhere on this SSE landscape, but you can never be 100% certain!

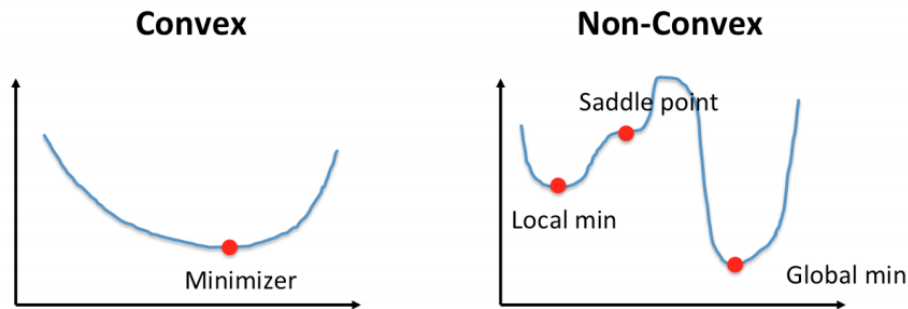


Figure 6: An illustration visualizing a convex function and a non-convex function. Also shown is the problem that comes with optimizing non-convex functions: getting stuck at a local minima.

Figure 6 illustrates this point. For the non-convex function, once you have reached a minimum, you don't know if it is a local minimum or a global minimum. However, in a convex function, you can be certain you have found a global minimum. Knowing that you are working with a convex optimization problem (i.e. a linear model) gives you the peace of mind in knowing that with enough training you will find weights that fit the data as well as possible for the model's complexity class.

Synthesis Questions:

1. What does “expressivity” of a function mean?
2. How do over and underfitting relate to bias and variance?
3. If a model gives low bias and variance while fitting to a dataset, is it a good choice?
4. Why is knowing that your optimization problem is convex useful/good?
5. **Bonus:** Are either k-means or k-medians convex optimization problems? Give reasoning (does not need to be in proof form)

8 When to Use Machine Learning Algorithms

Machine learning models are widely used in applications like image recognition, natural language processing, and predictive analytics. However, selecting the right algorithm for a task is essential. Simple models like linear regression may suffice for tasks with linear relationships, while more complex tasks (like image classification) often require more advanced models, such as neural networks (discussed in Unit 2).

Furthermore, unsupervised learning methods like k-means clustering are valuable when you want to find hidden patterns in the data without labeled outcomes.

9 Conclusion

Unit 1 covered the fundamental concepts of machine learning, with a focus on supervised learning (linear regression) and unsupervised learning (k-means clustering). We also introduced important concepts like regularization (L1 and L2), the bias-variance tradeoff, and convexity. Additionally, we discussed the differences between k-means and k-medians clustering and clarified why logistic regression is a classification algorithm.

In future units, you will explore deeper concepts like neural networks, support vector machines, and further techniques in model optimization and regularization. Understanding these basics is crucial as machine learning continues to influence more of our lives.

Unit 1 Project Specs

Technical Project Spec:

The project for this *Basics* section will **have you finish a code template through Google Colab**. Please ask questions as you work through this project. Be sure to discuss with others in your group if you have one! Share your answers as you like, the goal is to learn and we're not holding grades over your head.

This project will be going over k-means clustering (unsupervised ML). We will be using the Scikit-Learn library.

A few general helpful tips (if applicable):

- Be sure to appropriately make a copy of the Colab template before starting to save your progress!
- Renaming your copy to something that contains your name is a good idea, it will make it easier for us to review your submissions.
- Leave comments to cement your understanding. Link syntax to ideas.

Check out this handy image that gives popular sk-learn clustering algorithms and their usages:

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large n_{samples} , medium n_{clusters} with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters, inductive	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_{samples}	Many clusters, uneven cluster size, non-flat geometry, inductive	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with n_{samples}	Many clusters, uneven cluster size, non-flat geometry, inductive	Distances between points
Spectral clustering	number of clusters	Medium n_{samples} , small n_{clusters}	Few clusters, even cluster size, non-flat geometry, transductive	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large n_{samples} and n_{clusters}	Many clusters, possibly connectivity constraints, transductive	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large n_{samples} and n_{clusters}	Many clusters, possibly connectivity constraints, non Euclidean distances, transductive	Any pairwise distance
DBSCAN	neighborhood size	Very large n_{samples} , medium n_{clusters}	Non-flat geometry, uneven cluster sizes, outlier removal, transductive	Distances between nearest points
OPTICS	minimum cluster membership	Very large n_{samples} , large n_{clusters}	Non-flat geometry, uneven cluster sizes, variable cluster density, outlier removal, transductive	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation, inductive	Mahalanobis distances to centers
BIRCH	branching factor, threshold, optional global clusterer.	Large n_{clusters} and n_{samples}	Large dataset, outlier removal, data reduction, inductive	Euclidean distance between points
Bisecting K-Means	number of clusters	Very large n_{samples} , medium n_{clusters}	General-purpose, even cluster size, flat geometry, no empty clusters, inductive, hierarchical	Distances between points

Figure 7: Popular sk-learn clustering algorithms and their usages

Also this image visualizing the clustering algorithms:

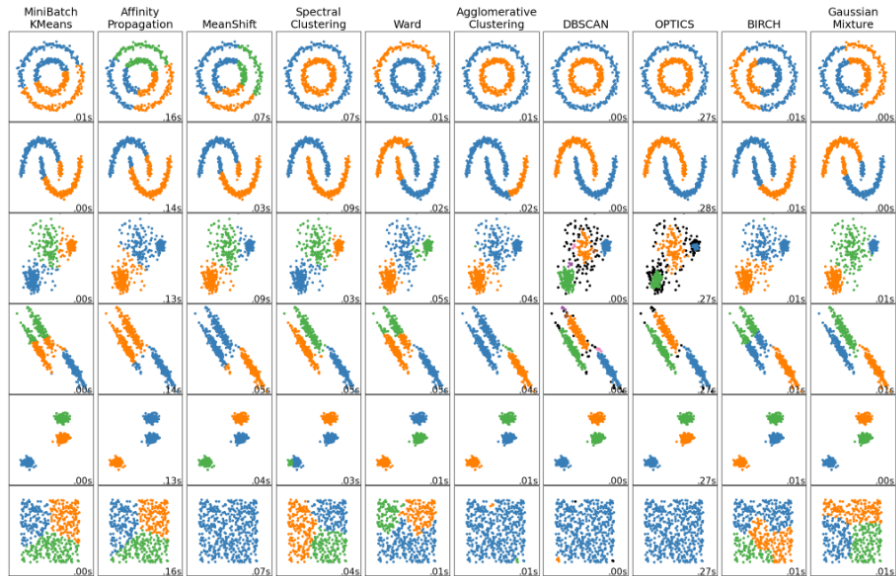


Figure 8: Visualization of clustering algorithms

Read up on k-means clustering in the provided link (Images provided above also contained here). Feel free to check out the other algorithms as well: SK-Learn Clustering

Now, follow the instructions on this Jupyter notebook (hosted on Google Colab) to implement some of the things we talked about! The notebook contains a link to the answers for this project. To use it, you will need to import the '.ipynb' file to a new Colab project yourself. It is highly recommended that you only use this to check your answers after you are done completing the project yourself. This is a trust-based system!

Colab Link: Unit 1 Colab Template (30 min)

When you are finished with your code, independently verify that it works and have fun with it! You could try this method on different datasets, such as this one for example. If you add any additional functionality be sure to talk about it with others and give them ideas.

Remember that this is all for your learning, so do your best and don't stress!

Congratulations! You now understand the basics of Clustering and PCA!