Machine Learning

Andrea Sofia Vallejo Budziszewski

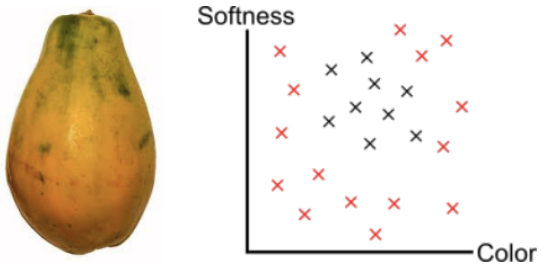
2023

Class 1

Understanding Machine Learning

Features

In the realm of machine learning, features are distinctive attributes or characteristics of the data that serve to describe or quantify the input. Think of features as the crucial details that define your data. In our example with papayas, the features we're interested in are "softness" and "color." Features play the vital role of representing input data in a structured and understandable way.



Tastiness Label: Papayas are categorized into two groups: they are either "tasty" (x) or "not tasty" (red x).

Observations

Observations, also known as data points or examples, are the tangible instances we gather for our machine learning endeavor. These observations encompass the values of the features (softness and color) and the corresponding tastiness label (either tasty or not tasty). In essence, these observations form the training data that our machine learning algorithm utilizes to discern underlying patterns.

The goal

The central objective of machine learning is to unearth hidden patterns or relationships within the data. In the context of our papaya example, the aim is to uncover patterns that explain how the features (softness and color) of papayas relate to their tastiness label (tasty or not tasty).

Prediction

When presented with a previously unseen papaya (x), our goal is to predict whether it falls into the "tasty" category or not. To do this, we record the features (softness and color) and employ a pattern model to make the prediction. It's important to note that the quality of predictions hinges on the chosen model, so we need a method to evaluate how well our models are performing.

Areas of Machine Learning

Supervised Learning

In supervised learning, our data consists of pairs of (feature, label), and the primary objective is to predict the label for new, unseen examples. Think of it as having a teacher guiding you through the

learning process.

Unsupervised Learning

Unsupervised learning, on the other hand, deals with data that only includes feature examples. Here, the aim is to identify inherent patterns or structures within the features themselves, without the guidance of labeled examples.

Reinforcement Learning

Reinforcement learning involves algorithms that can take actions that modify features. The ultimate goal is to take actions that maximize long-term rewards. It's akin to learning by trial and error, where the agent seeks to maximize its cumulative reward over time.

Domain Set

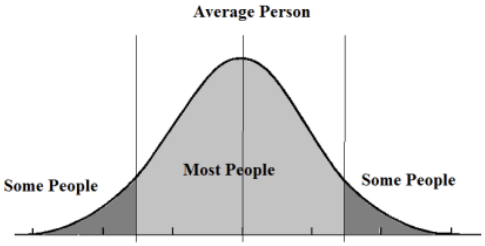
Domain Set Notation

Let's assume the existence of a set of features, denoted as $X_1, \ldots, X_d$. These features collectively describe the objects or concepts we're interested in.

Input Representation

Objects or concepts are effectively portrayed by feature vectors, expressed as $(x_1, \ldots, x_d)$. An input can be defined as $x = (x_1, \ldots, x_d) \in X_1 \times \ldots \times X_d$, which can be simplified as $x \in X$. In our example with papayas, d signifies the number of dimensions, which is 2, representing softness and color.

Input Distribution



Input Probability Distribution

It's essential to recognize that not all inputs are equally likely in the real world. We often assume that inputs follow a certain probability distribution, represented as D. To understand this, imagine that we can draw an input x from this distribution, denoted as $x \sim D$. It underscores the fact that different inputs may occur with varying probabilities.

Target Set

Target Set Notation

The target set, or label space, denoted as Y, signifies what we aim to predict through machine learning. The specific form of Y significantly influences the nature of the machine learning problem.

Classification

In the case of classification, the target set is $Y = c_1, \ldots, c_k$, which essentially means that our goal is to classify the input into one of several predefined classes.

Regression

On the other hand, if $Y = \mathbb{R}$, we're dealing with a regression problem. In regression, the aim is to predict a real number as the target, providing a continuous output.

Logistic Regression

For some problems, like our papaya example, Y may take on the form $Y = [0, 1]$. This corresponds to logistic regression, where the target is a probability, often used for binary classification problems. In our case, $Y = \text{tasty}, \text{not tasty}$ represents this scenario.

Labelling Function

Mathematical Representation

The elusive pattern we strive to learn is mathematically captured by a labelling function, typically expressed as $f : X \to Y$. What this means is that for any given input $x \in X$, the corresponding label is determined by $y = f(x) \in Y$. However, it's important to bear in mind that in real-world applications, labeling can be noisy, and identical inputs may yield different labels due to uncertainties or errors in the labeling process.

Training Set

Data is collected through observations and is labeled. This labeled data forms a training set denoted as S, which contains pairs of input and corresponding labels: $S = ((x_1, y_1), \ldots, (x_m, y_m))$. Each data point (x_i, y_i) consists of an input x_i sampled from a distribution D and a label y_i generated by an unknown function $f(x_i)$. Notably, the training set is a sequence, not a set, because it can contain identical data points (x_i, y_i) and (x_j, y_j) when $i < j$.

Supervised Learning Problem

A supervised learning problem comprises the following elements:

- A domain set X, which represents the possible input values. For example, $X = X_1 \times \ldots \times X_d$ indicates a d-dimensional input space.
- An unknown probability distribution D on X.
- A target set Y, which represents the possible output labels.

- An unknown labeling function $f : X \to Y$, which assigns labels to inputs.
- A training set $S = ((x_1, y_1), \ldots, (x_m, y_m))$ sampled from D and labeled by f.

Hypothesis Class

Given the vast space of possible functions, learners typically limit the set of candidate functions through a hypothesis class denoted as H. The hypothesis class H consists of functions $h : X \to Y$, and each function h is a potential model. As an example, consider a papaya detection problem, where H could be the set of all possible rectangles in two dimensions.

Loss Function

To assess the performance of a hypothesis $h \in H$, a loss function, denoted as ℓ, is needed. The loss function $\ell : Y \times Y \to \mathbb{R}$ measures the error between a predicted value $\hat{y}$ and the true label y. Loss is a cost, and lower values indicate better performance. Various loss functions can be employed, including:

- Classification or 0-1 loss: Given by $\ell(\hat{y}, y) = 1$ if $\hat{y} \neq y$, and 0 otherwise.
- Squared loss: Given by $\ell(\hat{y}, y) = (\hat{y} - y)^2$.

True Loss vs. Training Loss

The loss is used to evaluate the performance of a hypothesis $h \in H$. The true loss $L_{D,f}(h)$, also known as generalization error or risk, quantifies the mistakes made by h on the entire domain set X. It is defined as:

$$L_{D,f}(h) = E[x \sim D]\{\ell(h(x), f(x))\}.$$

The training loss $L_S(h)$, also called empirical risk, quantifies the mistakes of h on the training set S. It is calculated as the average loss over the training set:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(x_i), y_i).$$

Learning Algorithm

A learning algorithm aims to produce a hypothesis $h \in H$ that closely approximates the unknown labeling function f. The goal is to minimize the true loss $L_{D,f}(h)$. However, this true loss is unobservable because D and f are unknown.

Empirical Risk Minimization

Empirical risk minimization (ERM) is a fundamental principle in supervised learning. Given a hypothesis class H and a training set S, ERM selects the hypothesis that minimizes the empirical risk, also known as the training loss. Formally, this is expressed as $h_S = \mathrm{ERM}_H(S) \in \arg\min_{h \in H} L_S(h)$.

It's important to note that if the true labeling function f is in the hypothesis class H, then the training loss $L_S(h_S)$ is 0 because h_S perfectly matches f.

Supervised Learning

In supervised learning, the learner makes several key choices, including:

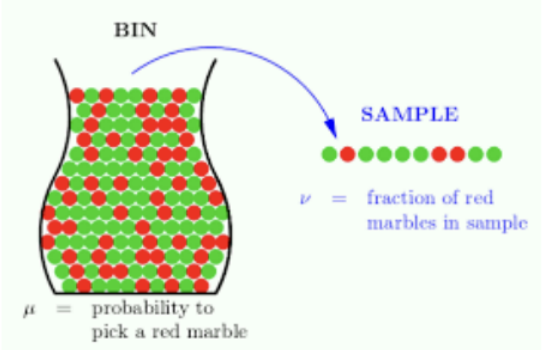
- The hypothesis class H, which contains candidate labeling functions.
- The loss function ℓ, which measures the error between predicted and true labels.
- An algorithm A that minimizes the empirical risk.

Generalization Properties

Generalization is a critical concept in supervised learning, as it relates to how well a model trained on a specific dataset performs on unseen data. Trivially, the true loss $L_{D,f}(h)$ can be decomposed into the training loss $L_S(h)$ and a term $L_{D,f}(h) - L_S(h)$. The accuracy of the approximation between $L_S(h)$ and $L_{D,f}(h)$ for a fixed hypothesis $h \in H$ depends on the quality of the training set $S = ((x_1, y_1), \ldots, (x_m, y_m))$.

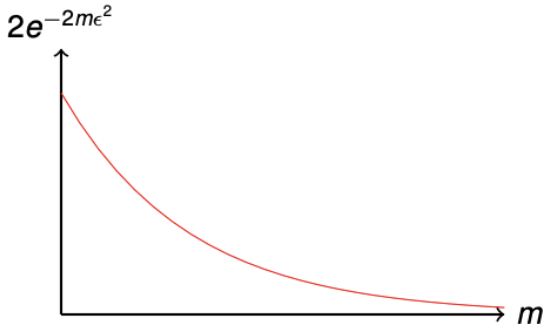
With high probability, $L_S(h)$ provides a good approximation of $L_{D,f}(h)$.

Bin Analogy



Imagine a large bin containing red and green marbles, with the proportion of red to green marbles being μ. If you randomly select m marbles from the bin and record the proportion ν of red marbles, Hoeffding's inequality can be used to analyze the probability of $|\nu - \mu|$ differing significantly from μ. Hoeffding's inequality states: $P\left[|\nu - \mu| > \varepsilon\right] \leq 2e^{-2m\varepsilon^2}$.

Growth of Upper Bound



The probability of your training set S being unrepresentative and having a significant difference between $L_S(h)$ and $L_{D,f}(h)$ decreases as a function of m, the number of samples in your training set.

Generalization

ERM returns a hypothesis $h_S = \text{ERM}_H(S)$ that minimizes the training loss $L_S(h_S)$. The question is how well this hypothesis h_S generalizes to the true loss $L_{D,f}(h_S)$. There is a bias due to the selection of h with the smallest training loss, and the quality of generalization depends on the dataset and the characteristics of the hypothesis class H.

Coin Flipping

Consider K people, each flipping a coin 10 times. On average, each person will get 5 heads and 5 tails, but the probability of getting 10 heads (or 10 tails) is approximately 0.1

Bounding the Probability of S Being Bad

For each hypothesis h in the hypothesis class H, you can bound the probability of the training set S being unrepresentative using Hoeffding's inequality. The sum of the probabilities of S being "bad" for all hypotheses can be bounded using the union bound on Hoeffding's inequality. This results in the following bound: $P[|L_S(h_S) - L_{D,f}(h_S)| > \varepsilon] \leq 2|H|e^{-2m\varepsilon^2}$, where $|H|$ is the size of the hypothesis class.

Noisy Labels

In practice, it's common for two identical inputs to have different labels due to noise. Instead of using a labeling function f, we can model this noise as a conditional probability distribution $P(y|x)$. On input x, the label is y with probability $P(y|x)$. This can be even more compactly represented as a joint probability distribution D on $X \times Y$, and data points (x, y) are sampled from D.

Naïve Empirical Risk Minimization

In empirical risk minimization (ERM), you compute the loss $L_S(h)$ for each hypothesis h in H and return the hypothesis with the smallest loss. This process requires iterating over all hypotheses in H and all data points in S, which can be computationally expensive.

Online Learning

Online learning involves processing data points one at a time and updating the candidate hypothesis after each data point is observed. The order of data points matters in online learning. In the context of $|Y| = 2$, $f \in H$, and a finite $|H|$, online learning is often implemented.

Consistent Learner

A consistent learner is an online learning algorithm that aims to predict data points correctly. The algorithm initializes H_1 with the entire hypothesis class H and proceeds to update hypotheses based on observed data points. The theorem states that the consistent learner will make at most $|H| - 1$ mistakes.

Halving Learner

The halving learner is another online learning algorithm. It starts with H_1 as the entire hypothesis class and, at each step, selects the majority vote of the hypotheses in H_t. The theorem suggests that the halving learner will make at most $\log_2(|H|)$ mistakes.

Algorithmic Properties

It's important to acknowledge that f may not necessarily be in H, and $|H|$ might not be small or finite. Algorithms should be computationally efficient, especially as minimizing the empirical risk is often computationally expensive (e.g., NP-hard). Typically, algorithms are specialized for the hypothesis class H and the chosen loss function ℓ.

k Nearest Neighbors

This approach assumes the existence of a distance function $\rho : X \times X \to \mathbb{R}$. To make predictions, k nearest neighbors are considered, where k is a parameter. The majority label among the k-nearest neighbors is returned as the prediction. This method does not require a hypothesis class or a specific loss function.

Theorem for k Nearest Neighbors:

The theorem states that the expected true loss $L_{D,f}(h_S)$ for k nearest neighbors is bounded by 2 times the optimal loss $L_{D,f}(h^*)$ plus a constant term that depends on the data and distance function.

Proving the Consistent Learner's Mistakes

The consistent learner updates its hypothesis set H_t based on its past mistakes. If the algorithm makes a mistake in round t, the hypothesis h used for prediction will not be part of H_{t+1}. It follows that $|H_{t+1}| \leq |H_t| - 1$ after each mistake. After k mistakes, $|H_t| \leq |H| - k$. Since $f \in H$ and f never makes mistakes, $|H_t| \geq 1$ for each t. Therefore, $|H| - k \geq |H_t| \geq 1$, which is equivalent to $k \leq |H| - 1$, indicating that the consistent learner makes at most $|H| - 1$ mistakes.

Proving the Halving Learner Makes at Most $\log_2(|H|)$ Mistakes

The halving learner updates its hypothesis set H_t based on its past mistakes. If the algorithm makes a mistake in round t, the hypotheses used for prediction will not be part of H_{t+1}. This implies that $|H_{t+1}|$ is reduced by at least half after each mistake. After k mistakes, $|H_t| \leq |H|/2^k$. Since $|H_t| \geq 1$ for all t, we have $|H|/2^k \geq |H_t| \geq 1$. Solving for k yields $k \leq \log_2(|H|)$, which indicates that the halving learner makes at most $\log_2(|H|)$ mistakes.

Proving the Memorizing Predictor

The memorizing predictor returns the label y_i if there exists (x_i, y_i) in S such that $x = x_i$, and returns 0 if no such (x_i, y_i) exists. To show the existence of a polynomial p_S that determines the memorizing predictor by defining $h(x) = 1$ whenever $p_S(x) \geq 0$, make p_S equal to 0 in each data point (x_i, y_i) such that $y_i = 1$, and less than 0 elsewhere. For a single data point (x_1, y_1) such that $y_1 = 1$, $p_S(x) = -k \cdot |x - x_1|^2$. For multiple data points, $p_S(x) = -\sum_{i \in [m] \text{ where } y_i = 1} |x - x_i|^2$.

Expected Training Loss

Fix some hypothesis $h \in H$ and show that the expected value of $L_S(h)$ equals $L_{D,f}(h)$, i.e., $E[S \sim D[L_S(h)]] = L_{D,f}(h)$. The expected training loss can be written as $E[S \sim D[L_S(h)]] = E[S \sim D[(1/m)\sum_{i=1}^{m} \ell(h(x_i), f(x_i))]]$. This is equal to $(1/m)\sum_{i=1}^{m} E[x_i \sim D[\ell(h(x_i), f(x_i))]]$. Each term in the summation is the expected loss for a single data point, and therefore, it is equal to $L_{D,f}(h)$. Thus, the expected training loss is the average of the true loss $L_{D,f}(h)$ over the training set and is, therefore, equal to $m/m \cdot L_{D,f}(h) = L_{D,f}(h)$.

Class 2

Perceptron

Binary Classification

Binary classification involves categorizing data into one of two classes, typically represented as $Y = \{-1, +1\}$. We assume the input features as $x = (x_1, x_2, \ldots, x_d)$, where d is the number of numerical features.

Hypothesis Function $h(x)$

We compute a weighted score for the input features and output either +1 or -1. The hypothesis function is defined as: $h(x) = 1$ if $\sum(w_i \cdot x_i) > \theta$ (the threshold) and -1 otherwise.

Sign Function (sign(x))

The sign function outputs +1 if $x > 0$ and -1 otherwise.

Hypothesis $h(x)$ in terms of weights

The hypothesis $h(x)$ can be expressed in terms of weights $w = (w_0, w_1, w_2, \ldots, w_d)$ and features x, where $x_0 = 1$ is a dummy feature: $h(x) = \text{sign}(\sum(w_i \cdot x_i) - \theta)$. Here, w_0 is the bias weight and is equal to $-\theta$.

Classification Loss (ℓ)

The classification loss is typically denoted as $\ell(h(x_i), y_i)$ and equals J if $h(x_i) \neq y_i$ and 0 if they are equal. $L_S(h)$ represents the total classification loss over the entire dataset, defined as: $L_S(h) = (1/m)\sum(\ell(h(x_i), y_i))$ for $i = 1$ to m.

Perceptron Learning Algorithm

1. Initialize the weight vector w_0 as 0. 2. Find a data point (x_i, y_i) that is misclassified, i.e., $h(x_i) \neq y_i$. 3. Update the weights as follows: $w_1 \leftarrow w_0 + y_i \cdot x_i$. 4. Repeat steps 2 and 3 for weight vectors w_t, incrementing t for each iteration.

Convergence of Perceptron Learning Algorithm

If the data is linearly separable (meaning there exists a hyperplane that separates the two classes), the Perceptron Learning Algorithm (PLA) is guaranteed to converge to a hypothesis h_S with zero classification loss, i.e., $L_S(h_S) = 0$. However, if the data is not linearly separable, the PLA may never converge.

Variants of PLA

Instead of running PLA until convergence, you can choose to run it for a fixed number of iterations (T) and stop when $t > T$. The Pocket algorithm is another variant. It updates the weight vector w only when the total number of mistakes decreases. This is useful in noisy datasets where you want to keep the best-performing weights seen so far.

Linear Regression

Regression Problem

In regression problems, the output variable Y is a real number ($Y = \mathbb{R}$), not a binary class. We have a hypothesis function $h(x)$ that is defined as the sum of weighted features, where $x = (x_1, x_2, \ldots, x_d)$.

Hypothesis $h(x)$

The hypothesis $h(x)$ is defined as: $h(x) = \sum(w_i \cdot x_i) = w^\intercal x$, where i goes from 0 to d.

Squared Loss

The squared loss measures the square of the difference between the predicted value $h(x_i)$ and the actual value y_i. $\ell(h(x_i), y_i) = (h(x_i) - y_i)^2$. The mean squared error (MSE) is represented as $L_S(h)$: $L_S(h) = (1/m) \sum((h(x_i) - y_i)^2)$ for $i = 1$ to m.

Minimizing $L_S(w)$

We aim to find the weight vector $w = (w_0, w_1, w_2, \ldots, w_d)$ that minimizes $L_S(w)$. $L_S(w)$ is a continuous, differentiable, and convex function of the weights.

Matrix Representation of $L_S(w)$

$L_S(w)$ can be expressed as: $L_S(w) = (1/m)\|\|Xw - y\|\|^2$, where X is an $m \times (d + 1)$ matrix of inputs, and y is an $m \times 1$ vector of labels.

Minimizing $L_S(w)$ by Setting Gradient to 0

To minimize $L_S(w)$, we set the gradient of $L_S(w)$ with respect to w to 0: $\nabla_w L_S(w) = (2/m)(X^\intercal X w - X^\intercal y)$.

Analytical Solution with Invertible $X^\intercal X$

If $X^\intercal X$ is invertible, there's an analytical solution for w called w_{lin}: $w_{\text{lin}} = (X^\intercal X)^{-1} X^\intercal y$.

Approximate Labels with Hat Matrix H

We can approximate labels for inputs $x_1, x_2, \ldots, x_m$ using the hat matrix H: $y = Hy$, where $H = X(X^\intercal X)^{-1} X^\intercal$.

Linear Regression for Classification

Linear regression can be used for classification by predicting labels as $\text{sign}(w_{\text{lin}}^\intercal x)$. For label +1, the squared loss bounds the 0-1 loss (classification error), and the same holds for label -1. This approach sacrifices bound tightness for computational efficiency.

Loss Optimized by PLA

The Perceptron Learning Algorithm (PLA) implicitly optimizes the hinge loss, $\ell_i(w) = \max(0, -y_i w^\intercal x_i)$, as it updates weights to minimize this loss when there is a misclassification.

Logistic Regression

Soft Classification

In soft classification, the goal is to estimate the probability of belonging to a class. The hypothesis function $h(x)$ is defined as $\theta(w^\intercal x)$, where $\theta(s)$ is the logistic function.

Logistic Function (θ)

The logistic function is defined as $\theta(s) = 1/(1 + e^{-s})$.

Two Classes: $\{-1, +1\}$

The problem involves two classes, $\{-1, +1\}$. Ideally, data would be in the form $((x_1, 0.8), \ldots, (x_m, 0.1))$, representing the probability of belonging to class $+1$. However, data is often in the form $((x_1, +1), \ldots, (x_m, -1))$, which can be viewed as hard probabilities (0 or 1).

Minimizing $L_S(w)$

The goal is to find the weight vector w that minimizes the loss function $L_S(w)$:

$$L_S(w) = \frac{1}{m} \sum \ell(h(x_i), y_i)$$

The cross-entropy loss $\ell(h(x_i), y_i)$ is defined as

$$\ell(h(x_i), y_i) = \ln(1 + e^{-y_i w^\intercal x_i})$$

$L_S(w)$ is continuous, differentiable, and convex.

Likelihood of a Biased Coin Flip

The likelihood of a biased coin flip is expressed as $p(y|x, w) = \theta(yw^\intercal x)$. The logistic loss is the logarithm of the likelihood: $\log p(y|x, w) = \log \theta(yw^\intercal x) = -\log(1 + e^{-yw^\intercal x})$. This provides a probabilistic interpretation of losses within the Bayesian formalism.

Minimizing $L_S(w)$ with Gradient Descent

To minimize $L_S(w)$, you need to find w such that the gradient $\nabla_w L_S(w)$ is equal to 0. The gradient is calculated as: $\nabla_w L_S(w) = (1/m) \sum \theta(-y_i w^\intercal x_i)(-y_i x_i)$. Unfortunately, there's no analytic solution to $\nabla_w L_S(w)$, so gradient descent is used to find the optimal weights.

Handling More Than Two Classes ($k > 2$)

If there are more than two classes ($2 < k$), two main approaches can be used:

One-versus-all (OVA)

Perform binary classification of one class vs. the rest for each class and return the most probable class.

One-versus-one (OVO)

Perform binary classification for pairs of classes and return the most probable class.

Properties of OVA:

OVA involves a linear number of classifiers but can result in imbalanced data.

Properties of OVO:

OVO requires a quadratic number of classifiers but often results in more balanced data.

In summary, soft classification involves estimating the probability of class membership using logistic regression and the logistic loss function. When dealing with more than two classes, OVA and OVO are two common approaches to extend binary classification to multiclass classification.

Non-Linear Transforms

Non-Linear Transforms

In many real-world scenarios, data is not linearly separable, making linear models less effective. One approach to tackle this problem is to derive non-linear transformations to make data more amenable to linear models.

Circular Hypothesis

Circular hypothesis is an example of a non-linear hypothesis. For instance, you can have a hypothesis like $h(x) = \text{sign}(0.6 - x_1^2 - x_2^2)$. This hypothesis is non-linear in the quadratic terms x_1^2 and x_2^2.

Linear in Quadratic Terms

To work with non-linear hypotheses like the circular one, we can introduce additional non-linear terms. A quadratic transform could be defined as $z = \Phi(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. This transformation introduces non-linear terms to capture the circular pattern within the data.

Transforming Data

For each data point (x_i, y_i), we transform it to $(z_i = \Phi(x_i), y_i)$. We then apply a linear algorithm in the transformed space to find a weight vector $w^{\sim}$. The hypothesis for input x is proportional to $w^{\sim} \intercal \Phi(x)$, allowing us to use a linear model in the transformed feature space.

Linear Models

Linear models are widely used in supervised learning and include: Perceptron, Linear regression, and Logistic regression. These models can be effective when combined with non-linear transforms.

Hypothesis Class of q-th Order Polynomials

Let H_q be the hypothesis class of q-th order polynomials. $H_1 \subset H_2 \subset \ldots \subset H_q$, meaning higher-order polynomials include the lower-order ones. However, it's important to note that as the polynomial order increases, the number of features increases as well. The q-th order polynomial has $O(d^q)$ dimensions, which can lead to increased memory requirements and longer running times for algorithms.

In summary, non-linear transforms allow linear models to handle non-linear data. Circular hypotheses, as an example, can be made linear with the introduction of non-linear terms. The use

of non-linear transforms can greatly increase the expressiveness of linear models, but it's essential to be mindful of the increased computational demands as the dimensionality of the feature space grows.

Exercises

One-Versus-All (OVA) Multiclass Classification:

In OVA, we create a binary classifier for each class against the rest. In this case, we need to build 10 such classifiers for a 10-class problem. Each of these 10 classifiers uses all the data, which is N data points. Since algorithm A has a running time of N^3, the total running time for OVA is $10 \cdot N^3$, which simplifies to $10N^3$.

One-Versus-One (OVO) Multiclass Classification:

In OVO, we build binary classifiers for each pair of classes. For a 10-class problem, this requires $\binom{10}{2} = 45$ classifiers in total, one for each combination of two classes. Each of these 45 classifiers uses data points from just 2 classes, which amounts to $\frac{2N}{10}$ data points. The running time for each classifier using algorithm A is $\frac{8N^3}{1000}$ (given that N^3 is the running time for N data points, and we have $\frac{2N}{10}$ data points for each classifier). The total running time for OVO is $45 \cdot \left(\frac{8N^3}{1000}\right)$, which simplifies to $\frac{9}{25}N^3$.

Explanation of Hat Matrix (H) Properties:

The hat matrix H, defined as $H = X(X^TX)^{-1}X^T$, has several key properties: 1. Symmetry:

H is symmetric, meaning $H^T = H$. This property arises from the fact that the transpose of the product of two matrices, $(AB)^T$, is equal to B^TA^T for any matrices A and B. 2. Idempotent:

$H^2 = H$, indicating that applying the hat matrix twice is the same as applying it once. This is a result of the idempotent nature of H, meaning that H times H results in H. 3. Complement:

$(I - H)^2 = (I - H)$, where I is the identity matrix. This property is related to the complement matrix $(I-H)$, which, when squared, retains its complement form. The complement matrix provides information about the portion of the dependent variable not explained by the independent variables used in the model represented by H.

Class 3

Generalization and VC Dimension

True Loss and Training Loss

True Loss (LD,f(h)): Measures the mistakes made by hypothesis (h) on the entire domain set X, with distribution D and labeling function f. Training Loss (LS(h)): Measures the mistakes made by hypothesis (h) on the training set S, which consists of (x, y) pairs.

Goal

We aim for a hypothesis (h) with a small true loss (LD,f(h)), but we can only directly measure the training loss (LS(h)).

Generalization

Generalization is the key objective in machine learning. We seek to minimize the difference between true loss (LD,f(h)) and training loss (LS(h)). This can be expressed as LD,f(h) - LS(h).

How Well Does LS(h) Approximate LD,f(h)?

Hoeffding's inequality provides insight into this approximation.

Hoeffding's Inequality

For a single, fixed hypothesis h, it estimates the probability that the absolute difference between $L_S(h)$ and $L_{D,f}(h)$ exceeds a value ϵ. The probability is bounded by $2 \cdot e^{-2m\epsilon^2}$, where m is the number of training samples.

Challenges with Unbounded Hypothesis Set (—H—)

In practice, the hypothesis set (H) is often infinite, leading to an unbounded —H—. Vapnik-Chervonenkis (VC) dimension (DVC) quantifies the effective size of H.

VC Dimension and Hoeffding's Inequality

For hypothesis h_S that minimizes empirical risk, the probability that the absolute difference between $L_S(h_S)$ and $L_{D,f}(h_S)$ exceeds ϵ is bounded by $2D_{VC}e^{-2m\epsilon^2}$. In linear models, —H— may be infinite, but DVC can be finite (e.g., d + 1, where d is the model's complexity).

Model Complexity

Model complexity refers to the number of model parameters (e.g., weights) and is often proportional to DVC. A more complex model may generalize less effectively due to a larger DVC.

Alternative Formulation of Hoeffding's Inequality

It introduces Ω(m, DVC) as a measure that decreases as a function of m. Training loss (LS) typically increases as a function of m, while true loss (LD,f) decreases as a function of m.

No Universal Algorithm

No algorithm performs well on all learning problems. Theorem: Given any binary classification algorithm A on domain X, and a training set size $m \leq \frac{|X|}{2}$, there exist distributions D and labeling functions f such that, with high probability, $L_{D,f}(A(S)) \geq \frac{1}{8}$. This theorem highlights the challenge of achieving a universal solution in machine learning.

Bias and Variance

Restricting Hypothesis Classes

It's essential to limit the class of hypothesis functions ($\mathcal{H}$) to ensure manageable model complexity. However, excessive restriction can hinder the ability to approximate the true labeling function f.

Bias

Bias measures how far the labeling function f is from the hypothesis class $\mathcal{H}$. A larger hypothesis class is more likely to include f but makes it challenging to pinpoint the correct f.

Variance

Variance measures how far the empirical risk minimization (ERM) hypothesis h_S is, on average, from f. It determines how far the true loss $L_{D,f}$ is from its theoretical limit. Variance typically decreases as the size of the training set (m) increases.

Tradeoff: Bias vs. Variance

Less complex models (lower $|\mathcal{H}|$) have more bias but less variance. More complex models (higher $|\mathcal{H}|$) have less bias but more variance. It is impossible to achieve both zero bias and zero variance simultaneously.

Regression Task and Squared Error (ERM hypothesis h_S)

Consider a regression task with squared error as the loss function for the ERM hypothesis h_S.

Decomposition of Expected True Loss

$\mathbb{E}_{S\sim D,f}\{L_{D,f}(h_S)\}$ can be decomposed into components: $\mathbb{E}_{x\sim D}\{(h_S(x) - f(x))^2\}$ $\mathbb{E}_{S\sim D,f}\{(h_S(x) - h(x))^2\}$ $\mathbb{E}_{S\sim D,f}\{(h_S(x) - h(x))(h(x) - f(x))\}$

This decomposition includes variance and bias terms.

Decomposition of Expected True Loss (Cont'd)

The decomposition simplifies to: $\mathbb{E}_{x\sim D}\{\text{variance}(x) + \text{bias}(x)\}$ Where $h(x)$ is the average ERM hypothesis on input x.

An Example with Hypothesis Classes

Assume that the true labeling function f follows a sine curve. Two hypothesis classes: $\mathcal{H}_0$ (constant hypotheses) and $\mathcal{H}_1$ (linear hypotheses). A small training set with $m = 2$ data points. Which hypothesis class is better for approximating the sine curve?

Averaging ERM Hypotheses

Define $g(x) = h(x)$: the average ERM hypothesis on input x.

The class notes illustrate the tradeoff between bias and variance when selecting a hypothesis class, especially in regression tasks. While a more complex hypothesis class may reduce bias, it increases variance, and vice versa. Achieving a balance between these two factors is essential for successful model generalization. In the example, averaging ERM hypotheses can provide a more robust approximation to the true function.

Overfitting

Overfitting and Model Complexity

More complex models often lead to smaller training loss ($L_S(h)$). Overfitting occurs when models sacrifice true loss ($L_{D,f}(h)$) for a smaller training loss. Higher model complexity results in smaller training loss but poor generalization properties, leading to larger true loss.

Finding the Optimal Model Complexity

There exists a theoretical optimum model complexity. Increasing model complexity beyond this point causes the loss to increase dramatically. In practice, it's often better to start with simpler models.

Techniques to Combat Overfitting

Linear models: Introduce constraints on the weight vector w. Constrained optimization: minimize $L_S(w)$ subject to the constraint $\sum(w_i^2) \leq C$. It can be challenging to optimize (NP-hard). Alternative approach: add an extra term to the loss function ($L_{aug}(w) = L_S(w) + \lambda \sum(w_i^2)$). λ is a hyperparameter controlling regularization. Linear regression example: $w_{\text{reg}} = (X^T X + \lambda I)^{-1} X^T y$.

Model Selection and Validation

Overfitting makes selecting by $L_S(h)$ not always a good idea. Validation: A better approach to approximate $L_{D,f}(h)$ but tends to be optimistic. Use a validation set (V) sampled independently of the training set (S). Compute the validation loss ($L_V(h)$), which is a better estimate of $L_{D,f}(h)$. In practice, divide the dataset into a training set and a validation set. Train multiple alternative models on the training set (S) and compute validation loss for each ($L_V(h_S)$). Select the model with the smallest validation error and retrain it on the entire dataset ($S \cup V$).

Cross-Validation for Model Selection

Partition the training set (S) into k subsets ($S_1, \ldots, S_k$), each of size m/k. In each experiment, train on $S \setminus S_i$ and validate on S_i. The cross-validation loss is the average of losses across experiments: $L_{\text{cv}}(\theta) = (1/k) \sum(L_{S_i}(h_i))$. A commonly used practice is to choose $k = 5$ or $k = 10$ for cross-validation.

In summary, the class notes emphasize the importance of addressing overfitting by finding an optimal model complexity and employing techniques like regularization and model selection. Validation and cross-validation are valuable tools to estimate and compare the true loss of different models, helping in the selection of models with better generalization performance.

Class 4

Optimization

Empirical Risk Minimization

The objective in supervised learning is to find a hypothesis, denoted as h, from a hypothesis class H that minimizes the empirical risk, $LS(h)$. This is essentially an optimization problem, where we aim to discover the hypothesis that makes the training data fit the best. Supervised learning is fundamentally intertwined with the field of optimization, as it involves searching for the best hypothesis.

Properties of Functions

Non-Differentiable: A function is considered non-differentiable at a point if it lacks a well-defined derivative at that specific point. These functions often exhibit abrupt changes or corners, which can pose challenges for traditional gradient-based optimization methods. Specialized optimization techniques, such as subgradient methods, are necessary to handle non-differentiable functions effectively.

Convex: A function is deemed convex when, in simple terms, it exhibits an upward curvature and lacks local minima. This property greatly simplifies the optimization process because it guarantees that any local minimum is also a global minimum. Convex optimization problems can be efficiently solved, and convex functions find widespread use in machine learning due to their advantageous characteristics.

Smooth: A smooth function is one that possesses a continuous and differentiable gradient. Smoothness enhances the tractability of optimization tasks, as gradient-based methods excel when dealing with such functions. In contrast, non-smooth functions may exhibit discontinuities or sharp changes in gradient, making optimization considerably more challenging.

Differentiable function and Convex set

A function, denoted as $f : C \to \mathbb{R}$, is considered differentiable if it is continuous and possesses a finite gradient at each point within its domain. Even for continuous functions, it is possible to compute a sub-gradient, which provides an approximation of the gradient. A set C is convex if, for any two points x_1 and x_2 in C, the entire line segment between them remains within C for any value of α in the range $[0, 1]$. A function $f : C \to \mathbb{R}$ is considered convex if, for all x_1 and x_2 in C and α within $[0, 1]$, it satisfies the convexity condition.

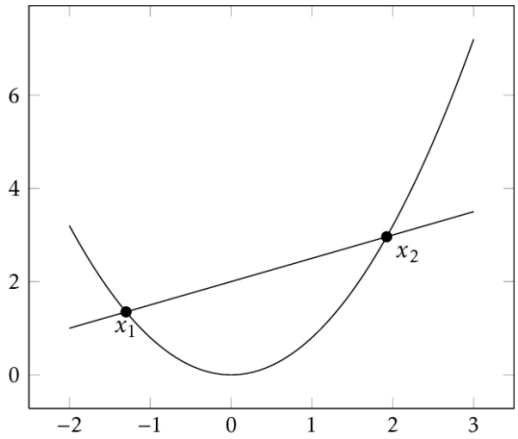
Convex function

This set is called a convex set if it has a particular property. A function, denoted as f, which maps points from this convex set to real numbers ($\mathbb{R}$), is considered a convex function if it satisfies the following rule:

For any two points, x_1 and x_2, within the convex set C, and for any value of α between 0 and 1 (inclusive), the following must hold:

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2)$$

The Convexity Formula:



Tangent and Convexity

For a function $f : C \to \mathbb{R}$ to be convex, it must satisfy the following inequality:

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha)f(x_2)$$

Detailed explanation of the formula:

$f(x_1)$ and $f(x_2)$ are the values of the function f at points x_1 and x_2, respectively.

α is a parameter that ranges between 0 and 1, inclusive.

$(1-\alpha)$ is complementary to α, such that the sum of α and $(1-\alpha)$ equals 1.

$\alpha x_1 + (1-\alpha)x_2$ represents a convex combination of x_1 and x_2. It's a weighted average where α represents the weight of x_1, and $(1-\alpha)$ represents the weight of x_2.

The formula states that for a function to be convex, the value of the function at the weighted average of any two points in the convex set $\alpha x_1 + (1-\alpha)x_2$ should be less than or equal to the weighted average of the function values at those two points $\alpha f(x_1) + (1-\alpha)f(x_2)$.

Local and Global Minimum for Convex Functions

Given a convex set C, a function $f : C \to \mathbb{R}$, and a point x_0 in C, x_0 is considered a local minimum if there exists a positive number δ such that for every point x within a δ-radius ball (neighborhood) around x_0, the function value at x_0, $f(x_0)$, is less than or equal to the function value at x, i.e., $f(x_0) \leq f(x)$.

Essential Theorem for Convex Functions: An essential theorem states that every local minimum of a convex function is also a global minimum. This means that in the context of convex functions, if you find a point x_0 where the function has a minimum value within a neighborhood, that minimum value is not just a local minimum; it's a global minimum.

Tangent at a Point in a Convex Set: Now, let's talk about the tangent at a point x_0 within a convex set C. The tangent at a point x_0 is represented as the linear function $l(x) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle$, where $\nabla f(x_0)$ is the gradient of the function at x_0. This tangent essentially serves as a linear approximation of the function near x_0.

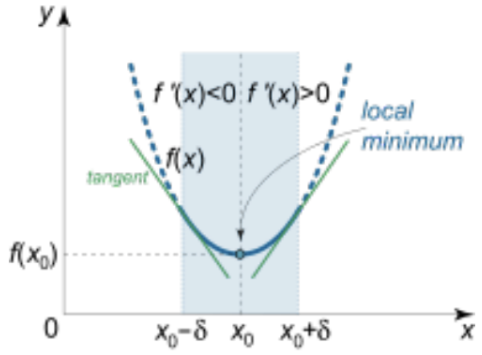
Crucial Theorem for Convex Functions (Tangent Below Function): Another important theorem for convex functions is that the tangent at any point x_0 in C lies below the function itself.

In other words, for any x in the convex set C, the value of the tangent function $l(x)$ is less than or equal to the value of the original function $f(x)$: $l(x) \leq f(x)$.

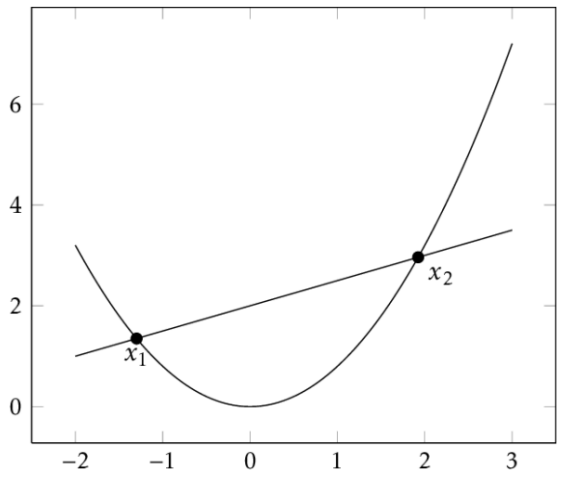
Equivalence of Convexity and Monotonic Non-Decreasing First Derivatives: For univariate, twice-differentiable functions, several properties are equivalent to convexity. One of these properties is that the function has monotonic non-decreasing first derivatives. This means that the slope (derivative) of the function is non-decreasing as you move along the domain of the function. Additionally, non-negative second derivatives (i.e., the curvature of the function is not "concave") is another equivalent property. For convex functions, the second derivative is non-negative, indicating that the function curves upward.

These theorems and concepts illustrate the power and importance of convex functions in optimization and other mathematical and computational problems. Convexity guarantees nice properties that simplify the search for optimal solutions.

Local Minimum: A local minimum is a point x_0 in the convex set C where, for some positive number δ, the function value at x_0, $f(x_0)$, is less than or equal to the function value at any other point x within a δ-radius ball around x_0.



Global Minimum: A global minimum is the lowest possible value of the function over the entire convex set C, not just within a neighborhood. It is the minimum value that the function can attain over its entire domain.



Tangent:

In the context of convex sets and functions, the tangent at a specific point x_0 plays a significant role. This tangent is represented as a linear function denoted as $l(x) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle$, where $\nabla f(x_0)$ is the gradient of the function at the point x_0. This tangent serves as a linear approximation of the function f near the point x_0. The formula for the tangent reflects how the function behaves locally around this point.

Theorem: The Tangent at x_0 Lies Below the Convex Function f

This theorem is a crucial result in the theory of convex functions, emphasizing an essential property:

Convex Functions: A function $f : C \to \mathbb{R}$ is considered convex if, for any two points x_1 and x_2 in the convex set C, and for any value of α between 0 and 1, the following inequality holds:

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2)$$

Explanation:

This theorem states that for a convex function f defined on a convex set C, the tangent at any point x_0 in C lies below the function itself. In other words, for any x in the convex set C, the value of the tangent function $l(x)$ is less than or equal to the value of the original function $f(x)$:

$$f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle \leq f(x)$$

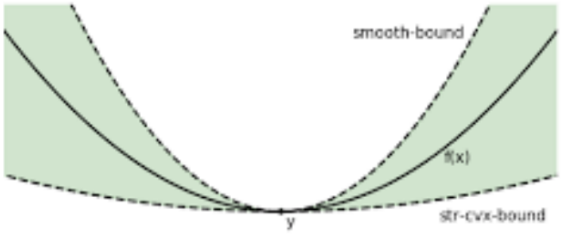
Key Concepts:

Tangent at x_0: The tangent represents a linear approximation of the function near a specific point x_0. It helps capture the local behavior of the function around that point.

Implication:

This theorem underscores one of the key characteristics of convex functions. It guarantees that at any point x_0 within the convex set C, the tangent line provides a lower bound for the function values. This property is fundamental in convex optimization because it simplifies the analysis and search for optimal solutions. It ensures that the linear approximation underestimates the function's values, making it easier to find the minimum of the convex function without exploring the entire domain. Convex optimization relies on this property to efficiently locate global minima.

Smoothness



A differentiable function $f : C \to \mathbb{R}$ is considered β-smooth if, for any pair of points x_1 and x_2 in C, the gradient difference is bounded by β times the Euclidean distance between x_2 and x_1. This smoothness property leads to the inequality, which implies that for β-smooth convex functions, the difference between function values at two points is bounded. Combining convexity and smoothness leads to specific conditions for learnability in supervised learning.

Smoothness and convexity

For a β-Smooth Function:

A function $f : C \to \mathbb{R}$ is considered β-smooth if it satisfies the following inequality for any x_1 and x_2 within the convex set C:

$$f(x_2) \leq f(x_1) + \langle \nabla f(x_1), x_2 - x_1 \rangle + \frac{\beta}{2}\|x_2 - x_1\|^2$$

Explanation:

This inequality relates the function values at two points, x_1 and x_2, in a convex set C. It states that for a β-smooth function, the function value at x_2 is upper-bounded by the function value at x_1, the directional gradient $\langle \nabla f(x_1), x_2 - x_1 \rangle$, and a quadratic term $\frac{\beta}{2}\|x_2 - x_1\|^2$. The quadratic term captures the curvature and smoothness of the function. It essentially bounds the difference in function values between x_1 and x_2 based on the gradient and the distance between these points.

For a Convex β-Smooth Function:

If the function f is also convex, we can obtain the following:

$$0 \leq f(x_2) - f(x_1) - \langle \nabla f(x_1), x_2 - x_1 \rangle \leq \frac{\beta}{2}\|x_2 - x_1\|^2$$

Explanation:

When the function is both convex and β-smooth, this inequality shows that the difference between the function values at x_2 and x_1, as well as the directional gradient term, is bounded by a quadratic term related to the β-smoothness of the function and the distance between x_1 and x_2.

Key Concepts:

β-Smooth Function: A β-smooth function is one that has a smoothness property where the function's values are upper-bounded by a linear term (the directional gradient) and a quadratic term, with $\frac{\beta}{2}$ being the coefficient of the quadratic term.

Convexity: Convexity is a property of functions defined on convex sets. A convex function satisfies the inequality $f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2)$ for any x_1 and x_2 in the convex set and any α in the range [0, 1].

Implication:

The inequality for β-smooth functions is important in optimization because it characterizes the behavior of functions in terms of smoothness and convexity. When a function is both β-smooth and convex, this inequality provides a bound on the difference in function values between two points, making it useful for analyzing and optimizing such functions. It helps in understanding the behavior of gradient-based optimization methods, particularly in convex optimization.

Convex learning problem

In a supervised learning problem $\langle X, D, Y, f, S\rangle$, the learner selects $\langle H, \ell, A\rangle$ components. The learning problem is considered convex if the hypothesis class H is a convex set, and the loss function ℓ and empirical risk function L_S are also convex. Convexity has a profound impact on learnability conditions in supervised learning.

Properties of Convex Functions

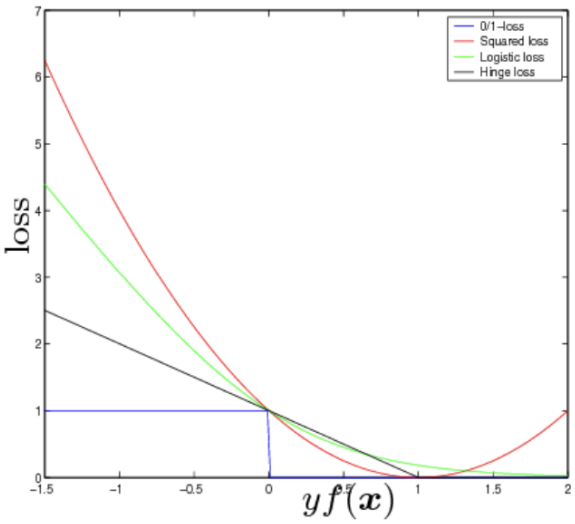
Convexity has some interesting properties. For instance, all norms, scaling a function by a positive constant, summing two convex functions, taking the maximum of two convex functions, and applying an affine transformation to a convex function result in convex functions. This insight can help in constructing and understanding convex functions in various learning problems.

Convexity and Learnability

Even if the hypothesis class H is bounded, there exist learning problems where an algorithm may fail. But if the loss function is both convex and β-smooth, the convex learning problem becomes learnable, subject to certain conditions. The learnability conditions involve convexity, boundedness, and smoothness.

Approximating Non-Convex Problems

When dealing with non-convex problems in supervised learning, it is possible to approximate them by using a convex loss function that provides an upper bound for the original loss. An example of this is the 0-1 loss function, which can be used as an approximation for non-convex problems.



Gradient Descent

The core idea in optimization with a convex, differentiable loss function is to descend in the opposite direction of the gradient.

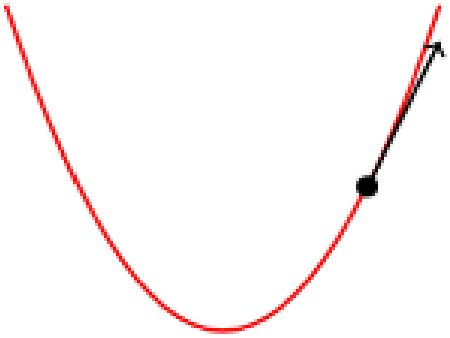


Figure 1: Gradient Descent

Optimization with Convex, Differentiable Loss Functions

The core idea in optimization with a convex, differentiable loss function is to descend in the opposite direction of the gradient.

Algorithm

1. Initialize the weight vector as $w_0 = 0$.

2. Compute the gradient $\nabla w \mathcal{L}_S(w_0)$.
3. Update the weights as $w_1 \leftarrow w_0 - \eta \nabla w \mathcal{L}_S(w_0)$.
4. Repeat the process from step 2 for weight vector w_t, where $t = 1, 2, \ldots$

Learning Rate

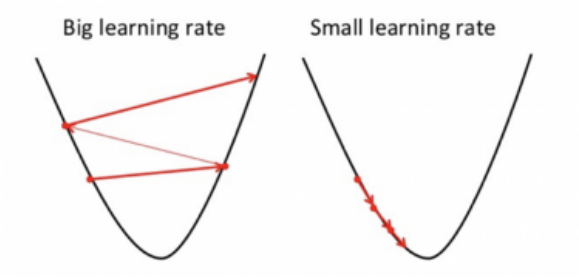


Figure 2: Learning Rate

The learning rate η is a crucial parameter that determines the rate of descent. It should be chosen carefully. If η is too large, learning can oscillate and may not reach the minimum. If η is too small, learning is slow and may not converge.

Convergence of Gradient Descent

Consider a convex and ρ-smooth function f.

Let H be the set of weight vectors satisfying $\|w\| \leq B$, and let w^* be the solution that minimizes $f(w)$ within H.

Running gradient descent for T iterations with a learning rate $\eta = \frac{B}{\rho\sqrt{T}}$ results in the average weight vector $\bar{w}$.

Theorem: The output vector $\bar{w}$ satisfies $f(\bar{w}) - f(w^*) \leq \frac{B\rho}{\sqrt{T}}$.

For a desired accuracy ϵ, the number of required iterations T is at least $T \geq \left(\frac{B}{2\rho}\right)^2 / \epsilon^2$.

Function Properties

f is a convex and ρ-smooth function.

A function is convex if, for any two points x_1 and x_2 in its domain and any value of t between 0 and 1, the following inequality holds:

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

A function is ρ-smooth if its gradient (or derivative) is Lipschitz continuous with a Lipschitz constant ρ. In other words, for all x_1 and x_2 in its domain, the following inequality holds:

$$\|\nabla f(x_1) - \nabla f(x_2)\| \leq \rho \cdot \|x_1 - x_2\|$$

H is a set of weight vectors w that satisfy the constraint $\|w\| \leq B$. In other words, it's a bounded region in which the optimization is taking place.

w^* is the solution that minimizes the function $f(w)$ within the set H. It represents the optimal solution within the bounded region.

Gradient descent is run for T iterations with a learning rate η given by $\frac{B}{\rho\sqrt{T}}$.

The algorithm outputs the average weight vector $\bar{w}$, which is the average of the weight vectors obtained in all T iterations.

Theorem:

The output vector $\bar{w}$ satisfies the following inequality:

$$f(\bar{w}) - f(w^*) \leq \frac{B\rho}{\sqrt{T}}$$

$f(\bar{w})$ represents the value of the function f evaluated at the average weight vector $\bar{w}$ obtained by gradient descent.

$f(w^*)$ is the value of the function at the optimal solution w^*.

B is the bound on the Euclidean norm of the weight vectors in H.

ρ is the smoothness constant.

T is the number of iterations.

$\sqrt{T}$ represents the square root of T.

The theorem tells us that the difference between the function value at the average weight vector $\bar{w}$ and the optimal value w^* is bounded by $\frac{B\rho}{\sqrt{T}}$. This means that as you run gradient descent for more iterations, the difference between the average weight vector and the optimal solution becomes smaller, and it scales inversely with the square root of the number of iterations.

Additionally, the theorem provides guidance on selecting the number of iterations T needed to achieve a desired accuracy ϵ. For a given accuracy ϵ, you need at least T iterations, where:

$$T \geq \left(\frac{B}{2\rho}\right)^2 / \epsilon^2$$

This result can help you determine how many iterations are required to reach a specific level of accuracy in the optimization process.

Basically, the theorem demonstrates the convergence behavior of gradient descent when applied to convex and smooth functions in a bounded region, providing insights into how the difference between the average solution and the optimal solution decreases with more iterations. It also offers a guideline for choosing the number of iterations to meet a desired accuracy.

Stochastic Gradient Descent (SGD)

Standard gradient descent can be inefficient when dealing with large training sets (S) because it requires iterating over the entire dataset in each iteration. SGD addresses this by computing a partial gradient on a subset of data points in each iteration. While each partial gradient may not perfectly match the full gradient, in expectation, these partial gradients lead to a descent toward the minimum.

Steps in SGD

1. Initialize the weight vector as $w_0 = 0$.
2. Compute a partial gradient v_0 such that $E[v_0|w_0] = \nabla wLS(w_0)$, where $LS(w_0)$ is the loss function associated with the current weight vector w_0.
3. Update the weights as $w_1 \leftarrow w_0 - \eta v_0$, where η is the learning rate.
4. Repeat the process from step 2 for the weight vector w_t, where t increments for each iteration.

SGD Theorem and Projection

Similar to standard gradient descent, SGD also has a convergence theorem that relates to the properties of the algorithm. Run SGD for T iterations with a learning rate $\eta = \frac{B}{\rho\sqrt{T}}$, and the average weight vector $\bar{w}$ satisfies: $E[f(\bar{w})] - f(w^*) \leq \frac{B\rho}{\sqrt{T}}$. The number of required iterations to achieve a desired accuracy ϵ is given by: $T \geq \left(\frac{B}{2\rho}\right)^2 / \epsilon^2$. An issue with SGD is that the updated weight vector may not remain bounded within the set H (where H defines the constraint $\|w\| \leq B$). To address this issue, projection onto H is applied without affecting convergence. This process is known as "SGD with projection."

SGD Convergence Theorem

The convergence theorem for SGD essentially states that as you run SGD for more iterations, the average weight vector $\bar{w}$ converges to a solution with an expected value close to the optimal solution w^*. The difference between the expected value of the function at $\bar{w}$ and w^* is bounded by $\frac{B\rho}{\sqrt{T}}$, where B is the bound on the norm of weight vectors in H, ρ is the smoothness constant, and T is the number of iterations. This result is helpful in understanding the convergence properties of SGD.

Number of Required Iterations for Accuracy

The theorem also provides a guideline for determining the number of iterations (T) required to reach a desired accuracy (ϵ). To achieve an accuracy of ϵ, you need at least T iterations, where $T \geq \left(\frac{B}{2\rho}\right)^2 / \epsilon^2$. This helps you estimate the computational effort needed to achieve a specific level of accuracy with SGD.

SGD with Projection

To ensure that the weight vector remains bounded within the set H, projection onto H is applied. This means that if the weight vector goes outside the bounds defined by H, it is projected back onto H. This projection step ensures that the algorithm remains within the feasible region while still benefiting from the convergence properties described in the convergence theorem.

Strong Convexity

A function $f : C \to \mathbb{R}$ is α-strongly convex if it satisfies the strong convexity condition. Strong convexity is a stricter property than convexity, implying that the function has a bowl-like shape around its minimum. Strongly convex functions exhibit faster convergence in optimization.

Function Properties

f is a λ-strongly convex and ρ-smooth function. These terms mean: A function is λ-strongly convex if, for all pairs of points x_1 and x_2 in its domain, the following inequality holds: $\frac{\lambda}{2}\|\|x_2 - x_1\|_2^2 \leq f(x_2) - f(x_1) - \langle \nabla f(x_1), x_2 - x_1 \rangle$

A function is ρ-smooth if its gradient (or derivative) is Lipschitz continuous with a Lipschitz constant ρ. In other words, for all x_1 and x_2 in its domain, the following inequality holds: $\|\nabla f(x_1) - \nabla f(x_2)\| \leq \rho\|\|x_1 - x_2\|_2$

H is a set of weight vectors w such that the Euclidean norm ($\|w\|$) is less than or equal to B. In other words, it's a bounded region in which the optimization is taking place.

w^* is the argument (weight vector) that minimizes the function f within the set H. In other words, it's the optimal solution within the bounded region.

The SGD algorithm is run for T iterations with a learning rate η given by $\frac{1}{\lambda t}$, where t is the iteration index.

The algorithm outputs the average weight vector $\bar{w}$, which is the average of the weight vectors obtained in all T iterations.

The output vector $\bar{w}$ satisfies the following inequality:

$$E[f(\bar{w})] - f(w^*) \leq \frac{\rho}{2}\left(\frac{1}{\lambda T}\right)(1 + \log(T))$$

$E[f(\bar{w})]$ represents the expected value (average) of the function f evaluated at the average weight vector $\bar{w}$ obtained by SGD. $f(w^*)$ is the value of the function at the optimal solution w^*. ρ is the smoothness constant. λ is the strong convexity constant. T is the number of iterations. $\log(T)$ is the natural logarithm of T.

It basically tells us that the expected value of the function at the average weight vector $\bar{w}$ is close to the optimal solution w^*. The difference is bounded by a term that depends on the smoothness of the function (ρ), the strong convexity of the function (λ), the number of iterations (T), and a logarithmic term $\log(T)$

Constrained Optimization

In the realm of optimization, we often encounter problems where we need to minimize a function while adhering to certain constraints. These constrained optimization problems can be represented as:

$$\text{Minimize } f(w) \tag{1}$$

$$\text{Subject to: } \begin{cases} g_i(w) = 0, & \text{for all } i = 1, \ldots, n \text{ (equality constraints)} \\ h_j(w) \geq 0, & \text{for all } j = 1, \ldots, k \text{ (inequality constraints)} \end{cases} \tag{2}$$

Equality Constraints and Lagrangian

When dealing with equality constraints, a powerful technique is to use the Lagrangian. For a problem with equality constraints $g_i(w) = 0$, the Lagrangian is defined as:

$$L(w, \lambda) = f(w) + \sum_{i=1}^{n} \lambda_i g_i(w) \tag{3}$$

Here, λ represents the Lagrange multipliers, one for each equality constraint. To find the optimal solution w^*, we set the gradient of the Lagrangian with respect to w to zero:

$$\nabla_w L(w^*, \lambda) = 0 \tag{4}$$

The dual optimization problem is then defined as maximizing $g(\lambda) = L(w^*, \lambda)$, and to obtain λ^*, we must solve this dual problem.

Inequality Constraints and KKT Conditions

For problems involving inequality constraints, the Karush–Kuhn–Tucker (KKT) conditions are commonly used. When dealing with inequality constraints $h_j(w) \geq 0$, the Lagrangian is:

$$L(w, \beta) = f(w) + \sum_{j=1}^{k} \beta_j h_j(w) \tag{5}$$

The KKT conditions are as follows:

$$1. \nabla_w L(w, \beta) = 0 \text{ (Gradients of the Lagrangian with respect to } w \text{ are zero)} \tag{6}$$

$$2. \beta_j h_j(w) = 0, \text{ for all } j = 1, \ldots, k \text{ (Complementary slackness: either the constraint is active, or the Lagrange multipli} \tag{7}$$

Linear and Quadratic Optimization

In linear optimization, both the objective function $f(w)$ and the constraints $g_i(w)$ and $h_j(w)$ are linear. The problem can be expressed as:

$$\text{Minimize } f(w) = c^T w \tag{8}$$

$$\text{Subject to: } \begin{cases} Gw = 0 \\ Hw \geq 0 \end{cases} \tag{9}$$

In quadratic optimization, the objective function is quadratic in the form:

$$f(w) = w^T A w + c^T w \tag{10}$$

Exercises

Demonstrating Convexity of the Univariate Function $f(x) = x^2$

In this example, we will prove that the univariate function $f(x) = x^2$ is convex. Convexity in this context implies that the function's second derivative, $f''(x)$, is non-negative for all x.

We start with the function $f(x) = x^2$.

The first step is to compute the first derivative, denoted as $f'(x)$. For this function, it is calculated as: $f'(x) = 2x$.

Next, we compute the second derivative, $f''(x)$. It is the derivative of the first derivative, and for $f(x) = x^2$, we have: $f''(x) = 2$.

To check the convexity of the function, we examine the second derivative.

In the given example, the second derivative is equal to 2, which is a positive constant.

Since the second derivative $f''(x)$ is non-negative for all x, this confirms that the function $f(x) = x^2$ is convex.

The general criterion for convexity in univariate functions is that if the second derivative is non-negative over the entire domain of the function, then the function is convex.

In conclusion, we have shown that the univariate function $f(x) = x^2$ is indeed convex by demonstrating that its second derivative, $f''(x)$, is always non-negative. This is a fundamental concept in optimization and calculus that helps us understand the behavior of functions in relation to their convexity.

Demonstrating Convexity of the Univariate Function $f(x) = \ln(1 + \exp(x))$

In this example, we will prove that the univariate function $f(x) = \ln(1+\exp(x))$ is convex. Convexity in this context implies that the function's second derivative, $f''(x)$, is non-negative for all x.

We start with the function $f(x) = \ln(1 + \exp(x))$.

The first step is to compute the first derivative, denoted as $f'(x)$. For this function, it is calculated as: $f'(x) = \frac{1}{1+\exp(x)} \cdot \exp(x)$.

Next, we compute the second derivative, $f''(x)$. It is the derivative of the first derivative. For $f(x) = \ln(1 + \exp(x))$, we have: $f''(x) = \frac{\exp(-x)}{(1+\exp(-x))^2}$.

To check the convexity of the function, we examine the second derivative.

In the given example, the second derivative is expressed as $\frac{\exp(-x)}{(1+\exp(-x))^2}$.

The denominator $(1 + \exp(-x))^2$ is always positive because the exponential function $\exp(-x)$ is always positive.

The numerator $\exp(-x)$ is also always positive since the exponential function is positive and raised to a negative power.

The result is the ratio of two positive values, which is itself positive.

The general criterion for convexity in uni variate functions is that if the second derivative is non-negative over the entire domain of the function, then the function is convex.

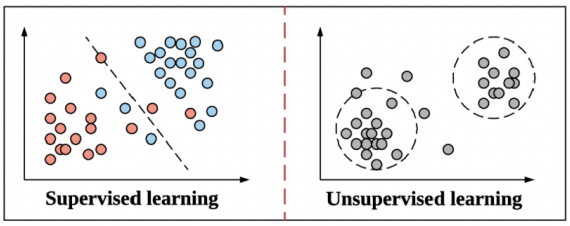
Therefore, in this specific case, since the second derivative $f''(x)$ is always positive, we conclude that the function $f(x) = \ln(1 + \exp(x))$ is indeed convex.

Class 5

Class 6

Introduction - Unsupervised Learning

Unsupervised learning is a fundamental branch of machine learning that deals with problems where we don't have access to labeled data. In this context, unsupervised learning seeks to uncover the inherent structure and patterns within an unlabeled dataset. Let's break down the key components of unsupervised learning to gain a comprehensive understanding of the concept.



Domain Set (X)

Unsupervised learning operates within a domain set, denoted as X. This set is comprised of d different dimensions or features, represented as $X_1, X_2, \ldots, X_d$. Each dimension can be thought of as a characteristic or attribute that describes the data. For instance, in a dataset of images, the dimensions could correspond to pixel values, color channels, or other image features.

Unknown Probability Distribution (D) on X

In unsupervised learning, we work with an unknown probability distribution, often denoted as D, that characterizes the data in our domain set X. This probability distribution represents the likelihood of observing different values across the dimensions in X. Understanding this distribution is crucial as it guides our exploration of the data.

Unlabeled Training Set (S)

The unlabeled training set, represented as S, consists of m data points $(x_1, x_2, \ldots, x_m)$. These data points are randomly sampled from the underlying probability distribution D. The absence of labels means that we do not have predefined categories or target values associated with the data points. This lack of supervision is a distinguishing feature of unsupervised learning.

The Objective of Unsupervised Learning

The primary goal of unsupervised learning is to uncover structure within the data. This structure could take various forms, such as clusters, patterns, or relationships between data points. By identifying these hidden structures, unsupervised learning algorithms aim to provide insights and understanding about the underlying data distribution.

Clustering

Introduction

Clustering is a technique in unsupervised learning that involves grouping sets of similar objects into clusters. It is widely used for various applications, particularly in data analysis and pattern recognition. Clustering, however, is not as straightforward as it may seem, and it comes with its own set of advantages and disadvantages. Let's explore the key aspects of clustering.

Key Components of Clustering

- **Distance Metric:** Clustering relies on a metric that measures the distances between input data points. This metric is essential for determining the similarity or dissimilarity between data points.
- **Classification:** Clustering can also be used for classification, where new inputs are assigned to existing clusters. This can help categorize new data based on the patterns observed in the training data.

Advantages of Clustering

- Most Common Unsupervised Learning Technique: Clustering is one of the most commonly used techniques in unsupervised learning, making it widely applicable.
- Identification of General Patterns: It helps in discovering general patterns within complex datasets, allowing for insights into the underlying data structure.
- Consideration of All Features: Clustering takes all features or dimensions into account, making it versatile for a wide range of datasets.

Disadvantages of Clustering

- Difficulty in Defining Clusters: Precisely defining what constitutes a "cluster" can be challenging, as the concept of a cluster varies from one dataset to another.
- Determining the Number of Clusters: Deciding how many clusters are needed in a dataset can be a complex task, often requiring heuristic approaches.
- Parameter Tuning: Many clustering algorithms have parameters that need to be fine-tuned, which can be a time-consuming process.
- Non-Separability of Data Points: In some cases, data points may not be easily separable into distinct clusters, making clustering challenging.

Cluster Definitions

Clusters can be defined in several ways, including:

- Small distances between cluster members.
- Specific intervals or statistical distributions.
- Dense areas within the state space.

Clustering can be viewed as a multi-objective optimization problem.

The K-Means Algorithm

The K-Means algorithm, also known as Lloyd's algorithm, is a widely used clustering technique. It is a centroid-based algorithm and is often considered the most popular clustering algorithm. In K-Means, "k" represents the number of desired clusters. The algorithm assumes a distance metric "d" that measures the distance between data points. The objective of K-Means is to minimize the K-Means cost, which quantifies the sum of squared distances between data points and their respective cluster centroids.

K-Means Algorithm Steps

- Input: Training set $S = (x_1, \ldots, x_m)$, integer k.
- Randomly choose k points as initial centroids.
- Iterate until convergence:
  - Recompute clusters $C_1, \ldots, C_k$ given the centroids.
  - Recompute centroids $\mu_1, \ldots, \mu_k$ given the clusters.
- Convergence occurs when clusters do not change between consecutive iterations.

Properties of K-Means

- K-Means is a greedy algorithm that is very fast compared to other clustering techniques.
- It tends to find clusters of comparable extent, and the resulting regions are Voronoi cells.
- The algorithm follows an expectation-maximization approach, interleaving assignment and update steps.

K-Means Disadvantages

- K-Means heavily depends on the choice of the number of clusters (k) and the initial centroids.
- It cannot find clusters with complex shapes.
- There are no guarantees on the magnitude of error compared to the optimal solution.
- The resulting cluster may not even be a local minimum, which makes interpretation challenging.

Variants of K-Means

K-Means can be improved using techniques like random restarts, random partition, and k-medoids. These variants address some of the limitations of the basic K-Means algorithm.

How to Choose the Number of Clusters (k)

Several methods, including G-Means, the elbow method, and the silhouette method, can be used to determine the optimal number of clusters.

Gaussian Mixture Models (GMMs)

GMMs are distribution-based clustering algorithms and are considered generative models. Each cluster in GMM is modeled as a Gaussian distribution with parameters, including mean (μ) and covariance (Σ). GMM assumes that data points are generated according to a combination of Gaussian components.

GMMs Optimization

The objective of GMMs is to maximize the (log-)likelihood for the entire dataset, which involves finding the optimal parameters for each Gaussian component. Direct maximization is challenging due to the presence of latent variables (unobserved component assignments), leading to the use of the Expectation-Maximization (EM) algorithm.

GMMs Expectation-Maximization (EM) Algorithm

- Input: Training set $S = (x_1, \ldots, x_m)$, integer k.
- Randomly initialize parameters for k Gaussian components.
- Iterate until convergence:
  - Expectation (E-Step): Compute the probability of each data point belonging to each cluster.
  - Maximization (M-Step): Update the parameters for each cluster based on the responsibilities computed in the E-Step.

GMMs vs. K-Means

- GMMs provide a soft assignment, meaning data points have probabilities of belonging to different clusters.
- K-Means, in contrast, offers a hard assignment, where data points belong to only one cluster.
- GMMs can capture complex cluster shapes due to their use of covariance matrices.

Dimensionality Reduction

Dimensionality reduction is a crucial technique in the field of data analysis and machine learning, particularly when dealing with high-dimensional data. This technique is used to discover patterns, reduce data complexity, and visualize data in lower-dimensional spaces. Let's delve into the fundamental concepts and techniques related to dimensionality reduction.

Preliminaries

- High-dimensional data is represented as x_i in $\mathbb{R}^D$, where $i = 1, \ldots, N$.
- Not all possible vectors appear in the dataset; data often lie on a low-dimensional manifold, which is a subset of possible values.
- The goal of dimensionality reduction is to find latent variables z_i in $\mathbb{R}^Q$ (where $Q < D$) that parametrize the location of x_i on the manifold.

Uses of Dimensionality Reduction

- Structure Discovery: It helps in discovering the underlying structure and patterns within the data.
- Visualization: Dimensionality reduction is useful for visualizing high-dimensional data in a lower-dimensional space.
- Pre-processing: It is employed as a pre-processing step for improving the efficiency of machine learning algorithms.

Classical Methods for Dimensionality Reduction

- Principal Component Analysis (PCA)
- Multidimensional Scaling (MDS)

Nonlinear Methods

- Isomap
- Locally Linear Embeddings (LLE)
- Maximum Variance Unfolding (MVU)
- Uniform Manifold Approximation and Projection (UMAP)
- Stochastic Neighbor Embedding (SNE)

Principal Component Analysis (PCA)

Preliminaries

- Given a dataset $X = (x_1, \ldots, x_N)^\top$ in $\mathbb{R}^{N \times D}$, where x_i is an observation in $\mathbb{R}^D$.
- The goal of PCA is to learn a linear bidirectional mapping, X to Z, such that as much information of X is retained in Z.
- We want to encode x_i into z_i such that if we decode z_i into $\hat{x}_i$, then $\hat{x}_i$ is a good approximation of the original x_i.

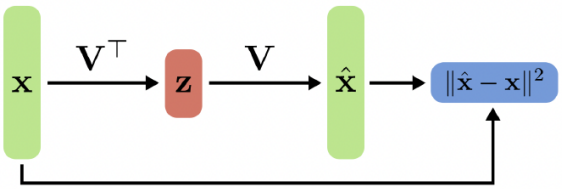
Derivation as Loss Minimization

- The encoding and decoding process in PCA involves linear mappings, which can be described by:
  - Decoder: $\hat{x}_i = \bar{x} + \sum_j (z_{ij} v_j)$, where $\bar{x}$ is the data mean and $V = (v_1, \ldots, v_Q)$ is an orthonormal basis.
- The goal is to minimize the L2 reconstruction loss with respect to Z and V, where the loss is defined as the difference between the original x_i and the reconstructed $\hat{x}_i$.

Variance Maximization Perspective

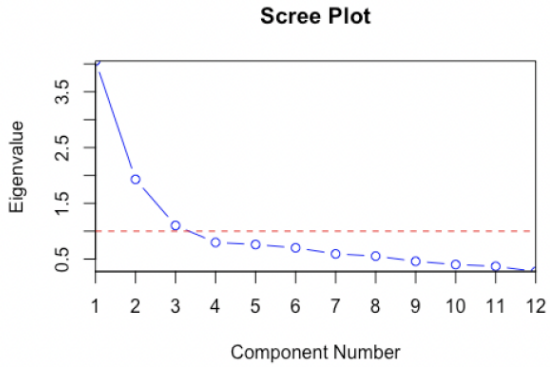
PCA can also be viewed as a variance maximization technique in latent space. The goal is to maximize the variance of the latent points z, captured by the covariance matrix S.

PCA Algorithm Steps



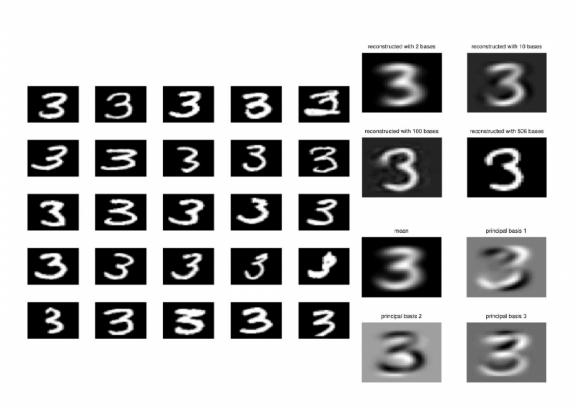
- Compute the data mean $\bar{x}$ and the scatter matrix S.
- Compute the eigen-decomposition of S.
- Select the top Q eigenvectors corresponding to the Q largest eigenvalues to form the basis V.

Eigenspectrum



The eigenspectrum represents how variance is distributed across dimensions. The Scree plot shows the eigenvalues, which monotonically decrease.It provides an indication of the proportion of variance explained by each component.

Example: MNIST Digits



PCA is applied to the MNIST dataset to reduce the dimensionality and visualize handwritten digits.

Example: Images

PCA can also be used for dimensionality reduction and visualization of images.