# Machine Learning

Adair Antonio da Silva Neto

January 3, 2025

**Disclaimer**

This text is a work in process. It is based on Stanford's "CS229 - Machine Learning" course and the book [HTF17].

# Contents

# Chapter 1

# Supervised Learning

## 1.1 Linear Regression and Gradient Descent

### 1.1.1 Linear Regression

Given a set of data $x$, our goal is to predict $y$, i.e., to find a **hypothesis** h such that $h(x) \approx y$.

In linear regression, we suppose that the relationship is linear:

$$h_\theta(x) = \theta_0 + \sum_{i=1}^{d} \theta_i x_i = \sum_{i=0}^{d} \theta_i x_i = \theta^{\mathrm{T}} x$$

with $x_0 = 1$.

So our goal is to find the parameters $\theta_i$ that minimize the **cost function**

$$\mathrm{J}(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

where $n$ is the number of elements in the training set.

### 1.1.2 Batch / Stochastic Gradient Descent

Our first approach is the **batch gradient descent**:

1. Initialize $\theta_0$ (equal to 0 or 1, for example);

2. For each $j$, update

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathrm{J}(\theta)$$

Notice that, for a single training example, we have

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x)-y)^2\\ &= (h_\theta(x)-y)\frac{\partial}{\partial\theta_j}(h_\theta(x)-y)\\ &= (h_\theta(x)-y)\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{d}\theta_i x_i - y\right)\\ &= (h_\theta(x)-y)x_j\end{aligned}$$

So, our algorithm becomes to repeat, until convergence, the update

$$\theta_j := \theta_j + \alpha\sum_{i=1}^{n}(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$$

for all $j$.

Implementing in Python, we have the Listing 1.

```
def batchGradientDescent(x, y, theta, alpha, numIterations):
    m = len(y)  # Number of examples
    for iteration in range(numIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y
        gradient = np.dot(x.T, loss) / m  # Compute gradient over all examples
        theta = theta - alpha * gradient  # Update theta
    return theta
```

Listing 1: Batch Gradient Descent.

A less costly alternative is the **stochastic gradient descent**, in which we repeat, for each $i = 1,\ldots,n$,

$$\theta_j := \theta_j + \alpha(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$$

for all $j$.

Again in Python, we have the Listing 2.

```
def stochasticGradientDescent(x, y, theta, alpha, numIterations):
    m = len(y)  # Number of examples
    for iteration in range(numIterations):
        for i in range(m):
            xi = x[i].reshape(1, -1)  # Single example
            yi = y[i]
            hypothesis = np.dot(xi, theta)
            loss = hypothesis - yi
            gradient = xi.T * loss  # Gradient for this example
            theta = theta - alpha * gradient.flatten()  # Update theta
    return theta
```

Listing 2: Stochastic Gradient Descent.

### 1.1.3 Normal Equations

To rewrite this procedure in matricial form, we let the **design matrix** X formed by the training inputs in each row, i.e. $(\mathrm{X}_{ij}) = x_j^{(i)}$.

Since $h_\theta(x^{(i)}) = (x^{(i)})^\mathrm{T}\theta$ and using that $z^\mathrm{T}z = \sum_i z_i^2$, it follows that

$$\frac{1}{2}(\mathrm{X}\theta - y)^\mathrm{T}(\mathrm{X}\theta - y) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 = \mathrm{J}(\theta)$$

Differentiating with respect to $\theta$,

$$\begin{aligned}
\mathrm{D}\mathrm{J}(\theta) &= \mathrm{D}\left(\frac{1}{2}(\mathrm{X}\theta - y)^\mathrm{T}(\mathrm{X}\theta - y)\right) \\
&= \frac{1}{2}\mathrm{D}((\mathrm{X}\theta)^\mathrm{T}\mathrm{X}\theta - (\mathrm{X}\theta)^\mathrm{T}y - y^\mathrm{T}(\mathrm{X}\theta) + y^\mathrm{T}y) \\
&= \frac{1}{2}\mathrm{D}(\theta^\mathrm{T}\mathrm{X}^\mathrm{T}\mathrm{X}\theta - \theta^\mathrm{T}\mathrm{X}^\mathrm{T}y - y^\mathrm{T}\mathrm{X}\theta) \\
&= \frac{1}{2}\mathrm{D}(\theta^\mathrm{T}\mathrm{X}^\mathrm{T}\mathrm{X}\theta - y^\mathrm{T}\mathrm{X}\theta - y^\mathrm{T}\mathrm{X}\theta) \\
&= \frac{1}{2}\mathrm{D}(\theta^\mathrm{T}\mathrm{X}^\mathrm{T}\mathrm{X}\theta - 2(\mathrm{X}^\mathrm{T}y)^\mathrm{T}\theta) \\
&= \frac{1}{2}(2\mathrm{X}^\mathrm{T}\mathrm{X}\theta - 2\mathrm{X}^\mathrm{T}y) \\
&= \mathrm{X}^\mathrm{T}\mathrm{X}\theta - \mathrm{X}^\mathrm{T}y
\end{aligned}$$

Setting this derivative to zero, we obtain the **normal equations**

$$\mathrm{X}^\mathrm{T}\mathrm{X}\theta = \mathrm{X}^\mathrm{T}y$$

Hence, the value of $\theta$ that minimizes $\mathrm{J}(\theta)$ is

$$\theta = (\mathrm{X}^\mathrm{T}\mathrm{X})^{-1}\mathrm{X}^\mathrm{T}y$$

## 1.2 Locally Weighted and Logistic Regression

### 1.2.1 Locally Weighted Regression

**Parametric** learning algorithm: fit a fixed set of parameters $\theta_i$ to data.

**Non-parametric** learning algorithm: the amount of data/parameters needed grows with the size of the training set.

In locally weighted regression, we fit $\theta$ to minimize

$$\sum_{i=1}^{n} w^{(i)}(y^{(i)} - \theta^\mathrm{T}x^{(i)})^2$$

where $w^{(i)}$ is a **weighting function**, for example:

$$w^{(i)} = \exp\left(\frac{-(x^{(i)} - x)^2}{2\tau^2}\right)$$

and $\tau$ is called the **bandwidth** parameter.

Notice that, if $|x^{(i)} - x|$ is small, then $w^{(i)} \approx 1$. If $|x^{(i)} - x|$ is large, then $w^{(i)} \approx 0$. Here, $x$ is the point where we want to predict.

### 1.2.2 Probabilistic Interpretation

Why do we use least squares?

Suppose that $y^{(i)} = \theta^{\mathrm{T}} x^{(i)} + \varepsilon^{(i)}$, where $\varepsilon$ represents the unmodelled effects and random noise. We assume that the $\varepsilon^{(i)}$ are i.i.d and $\varepsilon^{(i)} \sim \mathrm{N}(0, \sigma^2)$.

This means that

$$p(y^{(i)} \mid x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y^{(i)} - \theta^{\mathrm{T}} x^{(i)})^2}{2\sigma^2}\right)$$

i.e. $(y^{(i)} \mid x^{(i)}; \theta) \sim \mathrm{N}(\theta^{\mathrm{T}} x^{(i)}, \sigma^2)$. The ";" means "parameterized by".

The **likelihood** of $\theta$ is

$$\mathrm{L}(\theta) = p(y \mid x; \theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

And the **log-likelihood** is simply

$$\begin{aligned} l(\theta) = \log(\mathrm{L}(\theta)) &= \sum_{i=1}^{m} \left( \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \log \exp\left(\frac{-(y^{(i)} - \theta^{\mathrm{T}} x^{(i)})^2}{2\sigma^2}\right)\right) \\ &= m \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \sum_{i=1}^{m} \frac{-(y^{(i)} - \theta^{\mathrm{T}} x^{(i)})^2}{2\sigma^2} \end{aligned}$$

The **maximum likelihood estimation (MLE)** is (obviously!) to choose $\theta$ to maximize $\mathrm{L}(\theta)$, which is the same as choosing $\theta$ to minimize

$$\frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^{\mathrm{T}} x^{(i)})^2 = \mathrm{J}(\theta)$$

This proves that minimizing the least square errors is finding the maximum likelihood estimate.

### 1.2.3 Logistic Regression

We consider a binary classification problem, i.e., $y \in \{0, 1\}$. Thus, $h_\theta(x) \in [0, 1]$.

In logistic regression, we choose

$$h_\theta(x) = g(\theta^{\mathrm{T}} x) = \frac{1}{1 + e^{-\theta^{\mathrm{T}} x}}$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is the **sigmoid** or **logistic** function.

Since

$$p(y=1 \mid x;\theta) = h_\theta(x)$$

it follows that

$$p(y=0 \mid x;\theta) = 1-h_\theta(x)$$

Hence,

$$p(y \mid x;\theta) = h_\theta(x)^y(1-h_\theta(x))^{1-y}$$

In this case, the likelihood is

$$\mathrm{L}(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) = \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}}(1-h_\theta(x^{(i)}))^{1-y^{(i)}}$$

and the log likelihood,

$$l(\theta) = \sum_{i=1}^{m}[y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

To choose $\theta$ to maximize $l(\theta)$, we use **batch gradient ascent**:

$$\theta_j := \theta_j + \alpha\frac{\partial}{\partial\theta_j}l(\theta)$$

Computing this partial derivative, we obtain

$$\theta_j := \theta_j + \alpha\sum_{i=1}^{m}(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$$

### 1.2.4 Newton's Method

Suppose we have a function $f$ and want to find $\theta$ such that $f(\theta)=0$. In our case, we have $f=l'$. It is simply to iterate

$$\theta^{(t+1)} := \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

When $\theta$ is a vector,

$$\theta^{(t+1)} := \theta^{(t)} - \mathrm{H}^{-1}\mathrm{D}l(\theta^{(t)})$$

where H is the Hessian matrix.

The Newton's method has quadratic convergence.

## 1.3 Perceptron and Generalized Linear Model

### 1.3.1 Perceptron

The **perceptron** is similar to logistic regression, but instead of the sigmoid function, we use

$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Again, we have $h_\theta(x) = g(\theta^{\mathrm{T}}x)$ and the update rule is

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta^{(i)}(x))x_j^{(i)}$$

Notice that $y^{(i)} - h_\theta^{(i)}(x)$ will be 0 if the algorithm got it right, or $\pm 1$ if it got it wrong.

### 1.3.2 Exponential Family

The **exponential family** is a class of probability distributions with p.d.f.

$$p(y;\eta) = b(y)\exp(\eta^{\mathrm{T}}\mathrm{T}(y) - a(\eta))$$

where $y$ is the data, $\eta$ is the **natural parameter**, $\mathrm{T}(y)$ is a sufficient statistic, $b(y)$ is the **base measure**, and $a(\eta)$ is the **log-partition function**.

Remark that $e^{-a(\eta)}$ is a normalization constant to make sure that $p(y;\eta)$ sums, over $y$, to 1.

**Example (Bernoulli Distribution).** Let $\varphi$ be the probability of the event. Then,

$$\begin{aligned} p(y;\varphi) &= \varphi^y(1-\varphi)^{(1-y)} \\ &= \exp(\log(\varphi^y(1-\varphi)^{(1-y)})) \\ &= \exp\left[\log\left(\frac{\varphi}{1-\varphi}\right)y - \log(1-\varphi)\right] \end{aligned}$$

We have that $b(y) = 1$, $\mathrm{T}(y) = y$, $\eta = \log\left(\frac{\varphi}{1-\varphi}\right)$ (i.e. $\varphi = \frac{1}{1+e^{-\eta}}$), and $a(\eta) = -\log(1-\varphi)$ (i.e. $-\log(1-\frac{1}{1+e^{-\eta}}) = \log(1+e^{\eta})$).

**Example (Gaussian).** Assume that $\sigma^2 = 1$. Then,

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}}\exp\left(\frac{-(y-\mu)^2}{2}\right) \\ &= \frac{1}{\sqrt{2\pi}}e^{\frac{-y^2}{2}}\exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

We have $b(y) = \frac{1}{\sqrt{2\pi}}e^{\frac{-y^2}{2}}$, $\mathrm{T}(y) = y$, $\eta = \mu$, and $a(\eta) = \frac{\mu^2}{2} = \frac{\eta^2}{2}$.

**Properties of Exponential Families.**

1. The MLE with respect to $\eta$ is concave, i.e., the negative log-likelihood is convex.
2. The expectation of $y$ is
$$\mathrm{E}[y;\eta] = \frac{\partial}{\partial\eta}a(\eta)$$
3. The variance is
$$\mathrm{Var}[y;\eta] = \frac{\partial^2}{\partial\eta^2}a(\eta)$$

In applications, we use the Gaussian distribution for real-valued data, Bernoulli for binary data, Poisson for counting, gamma or exponential distributions for positive real numbers, and beta and Dirichlet distributions.

### 1.3.3 Generalized Linear Models

In **generalized linear models**, we do the following assumptions (design choices):

1. $y \mid x; \theta$ is a member of exponential family of $\eta$.
2. $\eta = \theta^{\mathrm{T}} x$, with $\theta, x \in \mathbf{R}^n$.
3. Test time: output will be $h_\theta(x) = \mathrm{E}[y \mid x; \theta]$.

The idea is that given $x$, we have a linear model $\theta^{\mathrm{T}} x$, which yields our parameter $\eta$. Then, based on our problem, we choose an exponential family and output the expectation.

During learning (training), we do gradient ascent on

$$\max_\theta \log p(y^{(i)}; \theta^{\mathrm{T}} x^{(i)})$$

The learning update rule is the same for all distributions:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta^{(i)}(x))x_j^{(i)}$$

**Terminology:**

1. $\eta$ is the **natural parameter**.
2. The expectation $\mathrm{E}[y; \eta] = g(\eta)$ is the **canonical response function**.
3. The inverse $\eta = g^{-1}(\mu)$ is the **canonical link function**.

For the logistic regression, we use the Bernoulli distribution, whence

$$h_\theta(x) = \mathrm{E}[y \mid x; \theta] = \varphi = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-\theta^{\mathrm{T}} x}}$$

For the linear regression, we use the Gaussian distribution. Then,

$$h_\theta(x) = \mathrm{E}[y \mid x; \theta] = \mu = \eta = \theta^{\mathrm{T}} x$$

### 1.3.4 Softmax Regression (Multiclass Classification)

Consider a classification problem in which the response variable $y$ can take $k \in \mathbf{N}$ possible values. For example, we may classify e-mails into three classes: spam, personal and work.

The labels $y$ are of the form $\{0, 1\}^k$ (i.e. a vector with $k$ entries, where all are 0 except for one which is equal to 1).

Each class $k$ has its own set of parameters $\theta_{\text{class}} \in \mathbf{R}^n$.

Given $x$, we compute $\theta_{\text{class}}^{\mathrm{T}} x$, exponentiate it and then normalize to obtain a probability distribution $\hat{p}$:

$$p(y = i \mid x; \theta) = \frac{e^{\theta_i^{\mathrm{T}} x}}{\sum_{i=1}^k e^{\theta_j^{\mathrm{T}} x}}$$

To obtain the real distribution $p$, we need to minimize the distance between the distributions, i.e., to minimize the cross-entropy between the two distributions.

The **cross-entropy** is

$$\text{CrossEnt}(p,\hat{p}) = -\sum_{y=1}^{k} p(y)\log\hat{p}(y) = -\log\hat{p}(y_0) = -\log\frac{e^{\theta_i^{\mathrm{T}}x}}{\sum_{i=1}^{k} e^{\theta_j^{\mathrm{T}}x}}$$

We treat this as the loss function and do the gradient descent.

## 1.4 Generative Learning Algorithms

A **discriminative learning algorithm** (e.g. logistic regression) learns $p(y \mid x)$, i.e., it learns

$$h_\theta = \begin{cases} 0 \\ 1 \end{cases}$$

directly.

A **generative learning algorithm** learns $p(x \mid y)$, i.e., it looks at the features $x$ given the class $y$. It also learns $p(y)$, the **class prior**. That means that we look at each class at a time.

Using Bayes rule, given a new test example, we compute

$$p(y=1 \mid x) = \frac{p(x \mid y=1)p(y=1)}{p(x)}$$

where $p(x) = p(x \mid y=1)p(y=1) + p(x \mid y=0)p(y=0)$.

### 1.4.1 Gaussian Discriminant Analysis (GDA)

Suppose $x \in \mathbf{R}^n$ (drop $x_0 = 1$ convention). The key assumption is that $p(x \mid y)$ is Gaussian. We model

$$\begin{aligned} y &\sim \text{Bernoulli}(\varphi) \\ x \mid y=0 &\sim \mathrm{N}(\mu_0, \Sigma) \\ x \mid y=1 &\sim \mathrm{N}(\mu_1, \Sigma) \end{aligned}$$

Our goal is to maximize the joint likelihood

$$\mathrm{L}(\varphi,\mu_0,\mu_1,\Sigma) = \prod_{i=1}^{m} p(x^{(i)},y^{(i)};\varphi,\mu_0,\mu_1,\Sigma) = \prod_{i=1}^{m} p(x^{(i)} \mid y^{(i)})p(y^{(i)})$$

Using MLE, we compute

$$\max_{\varphi,\mu_0,\mu_1,\Sigma} l(\varphi,\mu_0,\mu_1,\Sigma)$$

and obtain, for $\varphi$,

$$\varphi = \frac{\sum_{i=1}^{m} y^{(i)}}{m} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}}}{m}$$

For $\mu_0$ and $\mu_1$, we obtain the averages

$$\mu_0 = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} x^{(i)}}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}}}$$

and

$$\mu_1 = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}} x^{(i)}}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}}}$$

For $\Sigma$, we have

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^{\mathrm{T}}$$

Having fitted these parameters, to make a prediction, we choose

$$\arg\max_y p(y \mid x) = \arg\max_y \frac{p(x \mid y)p(y)}{p(x)} = \arg\max_y p(x \mid y)p(y)$$

### 1.4.2 Generative and Discriminative Comparison

In the discriminative model, we maximize the conditional likelihood

$$\mathrm{L}(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

For fixed parameters, if we plot $p(y = 1 \mid x; \varphi, \mu_0, \mu_1, \Sigma)$ as a function of $x$, we have the sigmoid function.

When is the GDA superior and when is the discriminative model superior?

The GDA implies logistic regression, but the converse is not true. This means that the GDA has stronger assumptions and thus it does better (if the assumptions hold).

We can also prove that if

$$\begin{aligned} y &\sim \text{Bernoulli}(\varphi) \\ x \mid y = 0 &\sim \text{Poisson}(\lambda_0) \\ x \mid y = 1 &\sim \text{Poisson}(\lambda_1) \end{aligned}$$

then $p(y = 1 \mid x)$ is also logistic.

This means that logistic regression works fine for Gaussian and Poisson data. However, if we assume our data is Poisson and it is Gaussian, for example, the model will work poorly.

GDA is also more computationally efficient.

### 1.4.3 Naive Bayes

Consider the e-mail spam classification problem. How can we represent it as a feature vector?

Take all the words in the dictionary (or the top 10000 most used words) and define a vector $x$ by putting $x_i = 1$ if the $i$th word of the dictionary appears in the e-mail and $x_i = 0$ otherwise. I.e., $x_i = \mathbf{1}_{\{\text{word } i \text{ appears in e-mail}\}}$. This is called the **multi-variate Bernoulli event model**.

We want to model $p(x \mid y)$ and $p(y)$. If we have 10000 words, there are $2^{10000}$ possible values of $x$.

Assume that the $x_i$'s are conditionally independent given $y$. This means that

$$\begin{aligned} p(x_1,\ldots,x_{10000} \mid y) &= p(x_1 \mid y)p(x_2 \mid x_1,y)\cdots p(x_{10000} \mid x_1,x_2,\ldots,y) \\ &= p(x_1 \mid y)p(x_2 \mid y)\cdots p(x_{10000} \mid y) \end{aligned}$$

The parameters of this model are:

- If the word $j$ is spam, what is the chance of it appearing?

$$\varphi_{j|y=1} = p(x_j = 1 \mid y = 1)$$

- If the word $j$ is not spam, what is the chance of it appearing?

$$\varphi_{j|y=0} = p(x_j = 1 \mid y = 0)$$

- What is the chance that the next e-mail you receive is spam?

$$\varphi_y = p(y = 1)$$

The joint likelihood is

$$\mathrm{L}(\varphi_y, \varphi_{j|y}) = \prod_{i=1}^{m} p(x^{(i)},y^{(i)}; \varphi_y, \varphi_{j|y})$$

The MLE yields

$$\varphi_y = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}}}{m}$$

and

$$\varphi_{j|y=1} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{x_j^{(i)}=1,\, y^{(i)}=1\}}}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}}}$$

At prediction time, we compute

$$p(y = 1 \mid x) = \frac{p(x \mid y = 1)p(y = 1)}{p(x \mid y = 1)p(y = 1) + p(x \mid y = 0)p(y = 0)}$$

### 1.4.4 Laplace smoothing

Notice that if we have a new word, the probability will be zero divided by zero. To deal with this problem, we use Laplace smoothing.

In Laplace smoothing, we compute

$$p(x = j) = \frac{\sum_{j=1}^{m} \mathbf{1}_{\{x^{(i)}=j\}} + 1}{m + k}$$

which then yields, in the previous example,

$$\varphi_{j|y=0} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{x_j^{(i)}=1,\, y^{(i)}=0\}} + 1}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} + 2}$$

### 1.4.5 Event models

Instead of representing $x$ as a vector of zeroes and ones, we can describe it as a vector in $\mathbf{R}^{n_i}$, where $n_i$ is the length of the e-mail $i$, and each entry $x_j \in \{1,\dots,10000\}$ represents the $j$-th word of the e-mail. For example, if the first word of the e-mail corresponds to the number 1600, then $x_1 = 1600$. This is the **multinomial event model**.

In this model, we have (just as in Naive Bayes):

$$p(x,y) = p(x \mid y)p(y) = \prod_{j=1}^{n} p(x_j \mid y)p(y)$$

and now the parameters are - $\varphi_y = p(y=1)$; - The choice of the $j$-th word being $k$ if $y=0$, i.e.,

$$\varphi_{k|y=0} = p(x_j = k \mid y = 0)$$

- The choice of the $j$-th word being $k$ if $y=1$.

The MLE is

$$\varphi_{k|y=0} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} \sum_{j=1}^{n_i} \mathbf{1}_{\{x_j^{(i)}=k\}}}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} n_i}$$

The denominator is the total number of words in all the non-spam emails, and the numerator counts, for non-spam emails, how many words in that email are the word $k$.

With Laplace smoothing,

$$\varphi_{k|y=0} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} \sum_{j=1}^{n_i} \mathbf{1}_{\{x_j^{(i)}=k\}} + 1}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} n_i + 10000}$$

## 1.5 Support Vector Machines

Support vector machines generalize decision boundaries for classification.

Our labels will be $y = \pm 1$, $h$ will output a value $\pm 1$ and

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Dropping the $x_0 = 1$ convention, our hypothesis will be

$$h_{w,b}(x) = g(w^{\mathrm{T}}x + b)$$

### 1.5.1 Optimal Margin Classifier (Separable Case)

Consider the logistic classifier $h_\theta(x) = g(\theta^{\mathrm{T}}x)$, which predicts one if $\theta^{\mathrm{T}}x > 0$ and zero otherwise. So, if $y^{(i)} = 1$, we want that $\theta^{\mathrm{T}}x^{(i)} \gg 0$, and if $y^{(i)} = 0$, we want that $\theta^{\mathrm{T}}x \ll 0$.

We define the **functional margin** of the hyperplane defined by $(w,b)$ with respect to $(x^{(i)},y^{(i)})$ as

$$\hat{\gamma}^{(i)} = y^{(i)}(w^{\mathrm{T}}x^{(i)} + b)$$

We want $\hat{\gamma}^{(i)} \gg 0$. Notice that $\hat{\gamma}^{(i)} > 0$ means that $h(x^{(i)}) = y^{(i)}$.

We define the **functional margin** with respect to the entire training set as

$$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}$$

The functional margin measures how confident/accurate our classification is.

We can suppose, by rescaling, that $\|w\| = 1$.

The **geometric margin** of the hyperplane defined by $(w, b)$ with respect to $(x^{(i)}, y^{(i)})$ is

$$\gamma^{(i)} = \frac{y^{(i)}(w^{\mathrm{T}}x^{(i)} + b)}{\|w\|}$$

Intuitively, the geometric margin is the Euclidean distance between the example $(x^{(i)}, y^{(i)})$ and the decision boundary.

Remark that the functional and geometric margin are related by

$$\gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{\|w\|}$$

The **geometric margin** with respect to the entire training set is defined as

$$\gamma = \min_i \gamma^{(i)}$$

The **optimal margin classifier** chooses the parameters $w$ and $b$ to maximize the geometric margin $\gamma$, that is, to

$$\begin{aligned} \max_{w,b} \quad & \gamma \\ \text{s.t.} \quad & \frac{y^{(i)}(w^{\mathrm{T}}x^{(i)} + b)}{\|w\|} \geq \gamma, \quad i = 1, \ldots, m \end{aligned}$$

We can reformulate this problem into

$$\begin{aligned} \min_{w,b} \quad & \|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^{\mathrm{T}}x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

Now, let us suppose that the parameter $w$ can be written as a linear combination of the training examples, i.e.,

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

The **representer theorem** states that we can make this assumption without losing performance.

This is a reasonable assumption because using logistic regression with gradient descent (stochastic or batch), for example, no matter how many iterations, the parameter $\theta$ (or $w$) is always a linear combination of the training examples.

Another intuition for this is that $w$ lies in the span of the training examples.

Our goal is to

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^{\mathrm{T}}x^{(i)}+b) \geq 1, \quad i=1,\dots,m \end{aligned}$$

Replacing our expression for $w$, we have

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\left(\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\right)^{\mathrm{T}}\left(\sum_{j=1}^{m}\alpha_j y^{(j)}x^{(j)}\right) \\ &= \min_{w,b}\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y^{(i)}y^{(j)}x^{(i)\mathrm{T}}x^{(j)} \\ &= \min_{w,b}\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y^{(i)}y^{(j)}\langle x^{(i)},x^{(j)}\rangle \end{aligned}$$

and

$$y^{(i)}\left(\left(\sum_{j=1}^{m}\alpha_j y^{(j)}x^{(j)}\right)^{\mathrm{T}}x^{(i)}+b\right) = y^{(i)}\left(\sum_{j=1}^{m}\alpha_j y^{(j)}\langle x^{(j)},x^{(i)}\rangle + b\right) \geq 1$$

The procedure here is 1. Solve for $\alpha_i$ and $b$; 2. Predict

$$h_{w,b}(x) = g(w^{\mathrm{T}}x+b) = g\left(\sum_{i=1}^{m}\alpha_i y^{(i)}\langle x^{(i)},x\rangle + b\right)$$

Using convex optimization, the problem above can be further simplified. This is called the **dual optimization problem**.

### 1.5.2 Kernels

The **kernel trick** is the following:

1. Write the algorithm in terms of inner products $\langle x^{(i)},x^{(j)}\rangle = \langle x,z\rangle$.
2. Define a mapping from the set of features to higher-dimensional features, i.e., $x \mapsto \varphi(x)$.
3. Compute the **kernel function** $\mathrm{K}(x,z) = \varphi(x)^{\mathrm{T}}\varphi(z)$.
4. Replace $\langle x,z\rangle$ in the algorithm with $\mathrm{K}(x,z)$.

The **support vector machine** is taking the optimal margin classifier and applying the kernel trick to it.

How to make kernels? If $x$ and $z$ are similar, then $\mathrm{K}(x,z)$ should be large. Conversely, if $x$ and $z$ are dissimilar, then $\mathrm{K}(x,z)$ is small. For example, we may define the **Gaussian kernel**

$$\mathrm{K}(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

This is a valid kernel if there exists a function $\varphi$ such that $\mathrm{K}(x,z) = \varphi(x)^{\mathrm{T}}\varphi(z)$. Thus, it is necessary that $\mathrm{K}(x,z)$ be a positive semi-definite function.

To prove that this is the case, let $\{x^{(1)}, \ldots, x^{(d)}\}$ be $d$ points, and $K \in \mathbf{R}^{d \times d}$ be the **kernel matrix** defined by $K_{ij} = K(x^{(i)}, x^{(j)})$.

Given any vector $z$,

$$\begin{aligned} z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\ &= \sum_i \sum_j z_i \varphi(x^{(i)})^T \varphi(x^{(j)}) z_j \\ &= \sum_i \sum_j z_i \sum_k \varphi_k(x^{(i)}) \varphi_k(x^{(j)}) z_j \\ &= \sum_k \sum_i \sum_j z_i \varphi_k(x^{(i)}) \varphi_k(x^{(j)}) z_j \\ &= \sum_k \left( \sum_i z_i \varphi_k(x^{(i)}) \right)^2 \geq 0 \end{aligned}$$

This proves that the kernel matrix K is positive semi-definite. It turns out that this is also a sufficient condition.

**Theorem (Mercer).** K is a valid kernel function (i.e., there exists a function $\varphi$ such that $K(x, z) = \varphi(x)^T \varphi(z)$) iff. for any $d$ points, the corresponding kernel matrix K is positive semi-definite.

Another example of kernel is the **linear kernel** $K(x, z) = x^T z$, with $\varphi(x) = x$.

### 1.5.3 Inseparable Case

If the data is not linearly separable, we reformulate our optimization as follows:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 + c \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \end{aligned}$$

where $\xi_i \geq 0$. This is called the $L_1$**-norm soft margin SVM**.

The problem can also be simplified using the same calculation as the representer theorem.

Some examples of applications of SVMs are handwritten digit classification and protein sequence classifier.

# List of Listings

# Bibliography

[HTF17] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference, and prediction, 2017.