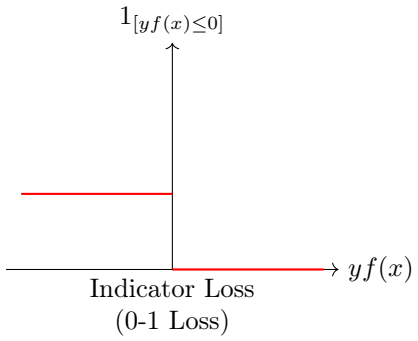
Linear Classification

Muchang Bahng

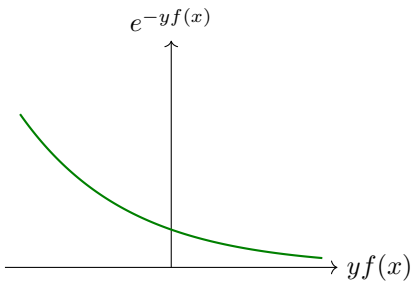
Spring 2025

Contents

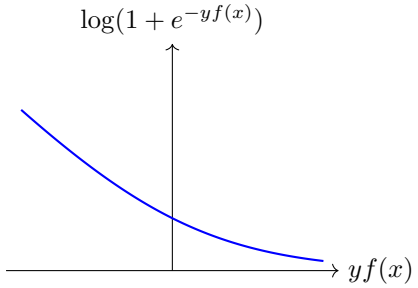
You literally just try to build a hyperplane to minimize the number of misclassifications, but this is not really differentiable and is hard. It's just a stepwise function. Therefore, you use a **surrogate loss function** to approximate the 0-1 loss function. The logistic uses some function, and the SVM uses the smallest convex function to approximate the 0-1 loss function. Here are some examples:



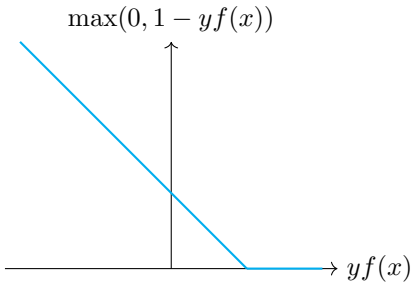
(a) Indicator/0-1 Loss can't be easily optimized.



(b) Exponential Loss for Adaboost.



(c) Log Loss for logistic regression.



(d) Hinge Loss for support vector machines.

Figure 1: Common loss functions used in classification

1 Perceptron

Definition 1.1 (Perceptron Model and Loss)

The simplest binary classification model is the **perceptron algorithm**. It is a discriminative parametric model that assigns

$$f_w(x) = \begin{cases} 1 & \text{if } w^T x + b \geq 0 \\ -1 & \text{if } w^T x + b < 0 \end{cases} \tag{1}$$

where we have chosen to label class $C_1 = 1$ and $C_2 = -1$. Note that unlike linear regression (and logistic regression, as we will see later), the perceptron is not a probabilistic model. It is a **discriminant function**, which just gives point estimates of the classes, not their respective probabilities. Like logistic regression, however, it is a linear model, meaning that the decision boundary it creates is always a linear (affine) hyperplane.

To construct the surrogate loss function, we would want a loss that penalizes not only if there is a misclassification, but how *far* that misclassified point is from the boundary. Therefore, if y and $\hat{y} = f_w(x)$ have the same sign, i.e. if $yf_w(x) > 0$, then the penalty should be 0, and if it is < 0, then the penalty should be proportional to the orthogonal distance of the misclassified point to the boundary, which is represented by $-wTxy$ (where the negative sign makes this cost term positive).

Definition 1.2 (Surrogate Loss for Perceptron)

Therefore, our cost functions would take all the points and penalize all the terms by 0 if they are correctly classified and by $-\mathbf{w}^T \boldsymbol{\phi}^{(n)} y^{(n)}$ if incorrectly classified.

$$L(y, \hat{y}) = \sum_{n=1} [-\mathbf{w}^T \boldsymbol{\phi}^{(n)} y^{(n)}]_+ \text{ where } [f(\mathbf{x})]_+ \coloneqq \begin{cases} f(\mathbf{x}) & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{else} \end{cases} \tag{2}$$

Note that this is a piecewise linear function and convex.

Code 1.1 (Perceptron in scikit-learn)

Let's implement this in scikit-learn, using two pipelines with different data standardization techniques to see the differences in the perceptron boundary.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import QuantileTransformer, StandardScaler

pipe1 = Pipeline([
    ("scale", StandardScaler()),
    ("model", Perceptron())
])

pipe2 = Pipeline([
    ("scale", QuantileTransformer(n_quantiles=100)),
    ("model", Perceptron())
])
```

Figure 2

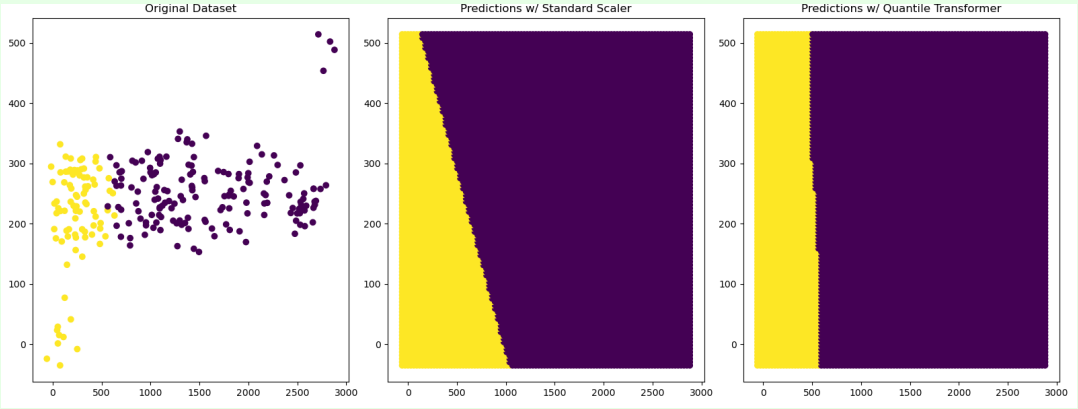


Figure 3: Perceptron Trained on Different Standardized Data

2 Logistic and Softmax Regression

We can upgrade from a discriminant function to a discriminative probabilistic model with **logistic regression**. In practice, we usually deal with **probabilistic models** where rather than giving a point estimate $\hat{y}$, we attempt to model the *distribution* $\mathbb{P}_{Y|X=\hat{x}}$. Even though in the end, we will just output the mean μ of this conditional distribution, modeling the distribution allows us to quantify uncertainty in our measurements.

Definition 2.1 (Logistic Regression)

The **logistic regression** model is a linear model of the form

$$f_w(x) = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}, \text{ where } \sigma(x) \coloneqq \frac{1}{1+e^x} \tag{3}$$

It is different from linear regression in two ways:

1. In linear regression, we assumed that the targets are linearly dependent with the covariates as $y = w^T x + b$. However, this means that the hypothesis f_w is unbounded. Since we have two classes (say with labels 0 and 1), we must have some sort of *link function* σ that takes the real numbers and compresses it into the domain $[0, 1]$. Technically, we can choose any continuous, monotinically increasing function from $\mathbb{R}$ to $(0, 1)$. However, the following property of the sigmoid makes derivation of gradients very nice.

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{4}$$

2. Once this is compressed, we assume that the residual distribution is a Bernoulli.

Definition 2.2 (Binary Cross Entropy Loss as Surrogate Loss for Logistic Regression)

The surrogate loss for logistic regression is the **binary cross entropy loss**, which is defined as

$$L(y, \hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y}) \tag{5}$$

One important observation to make is that notice that the output of our hypothesis is used as a parameter to define our residual distribution.

1. In linear regression, the f_w was used as the *mean* μ of a Gaussian.
2. In logistic regression, the f_w is used also as the mean p of a Bernoulli.

The reason we want this sigmoid is so that we make the domains of the means of the residuals match the range of the outputs of our model. It's simply a manner of convenience, and in fact we could have really chose any function that maps $\mathbb{R}$ to $(0, 1)$.

Some questions may arise, such as "why isn't the variance parameter of the Gaussian considered in the linear model?" or "what about other residual distributions that have multiple parameters?" This is all answered by generalized linear models, which uses the output of a linear model as a *natural parameter* of the canonical exponential family of residual distributions.

Unfortunately, there is no closed form solution for logistic regression like the least squares solution in linear regression. Therefore, we can only resort to maximum likelihood estimation.

Theorem 2.1 (Maximum Likelihood Estimation for Logistic)

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, our likelihood is

$$L(\theta; \mathcal{D}) = \prod_{i=1}^n p(y^{(i)} \mid x^{(i)}; \theta) = \prod_{i=1}^N \big(h_\theta(x^{(i)})\big)^{y^{(i)}} \big(1 - h_\theta(x^{(i)})\big)^{1-y^{(i)}} \tag{6}$$

We can equivalently minimize its negative log likelihood, giving us the **binary cross entropy** loss function

$$\ell(\theta) = -\log L(\theta) \tag{7}$$

$$= -\sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \tag{8}$$

Now taking the gradient for just a single sample $(x^{(i)}, y^{(i)})$ gives

$$\frac{\partial \ell}{\partial \theta} = \left(\frac{y^{(i)}}{\sigma(\theta^T x^{(i)})} - \frac{1 - y^{(i)}}{1 - \sigma(\theta^T x^{(i)})} \right) \frac{\partial}{\partial \theta} \sigma(\theta^T x^{(i)}) \tag{9}$$

$$= \frac{\sigma(\theta^T x^{(i)}) - y^{(i)}}{\sigma(\theta^T x^{(i)}) \left(1 - \sigma(\theta^T x^{(i)})\right)} \sigma(\theta^T x^{(i)}) \left(1 - \sigma(\theta^T x^{(i)})\right) x^{(i)} \tag{10}$$

$$= \left(h_\theta(x^{(i)}) - y^{(i)}\right) x \tag{11}$$

and summing it over some minibatch $M \subset \mathcal{D}$ gives

$$\nabla_\theta \ell_M = \sum_{(x,y) \in M} (y - h_\theta(x)) x \tag{12}$$

Therefore, the stochastic gradient descent algorithm is

$$\theta = \theta - \eta \nabla_\theta \ell(\theta) \tag{13}$$

$$= \theta - \eta \sum_{(x,y) \in M} (y - h_\theta(x)) x \tag{14}$$

We would like to extend this to the multiclass case.

Definition 2.3 (Softmax Function)

The softmax function is defined

$$o(\mathbf{x}) = \frac{e^{\mathbf{x}}}{||e^{\mathbf{x}}||} = \frac{1}{\sum_j e^{x_j}} \begin{pmatrix} e^{x_1} \\ \vdots \\ e^{x_D} \end{pmatrix} \tag{15}$$

What makes the softmax so popular is that the total derivative turns out to simplify functions a lot. The total derivative of the softmax can be derived as such.

Lemma 2.1 (Derivative of Softmax)

The derivative of the softmax is

$$Do(\mathbf{x}) = \text{diag}(o(\mathbf{x})) - o(\mathbf{x}) \otimes o(\mathbf{x}) \tag{16}$$

where $\otimes$ is the outer product. That is, let y_i be the output of the softmax. Then, for the 4×4 softmax function, we have

$$Do(\mathbf{x}) = \begin{pmatrix} y_1(1-y_1) & -y_1 y_2 & -y_1 y_3 & -y_1 y_4 \\ -y_2 y_1 & y_2(1-y_2) & -y_2 y_3 & -y_2 y_4 \\ -y_3 y_1 & y_3 y_3 & y_3(1-y_3) & -y_3 y_4 \\ -y_4 y_1 & -y_4 y_2 & -y_4 y_3 & y_4(1-y_4) \end{pmatrix} \tag{17}$$

Proof.

We will provide a way that allows us not to use quotient rule. Given that we are taking the partial derivative of y_i with respect to x_j, we can use the log of it to get

$$\frac{\partial}{\partial x_j}\log(y_i) = \frac{1}{y_i}\frac{\partial y_i}{\partial x_j} \implies \frac{\partial y_i}{\partial x_j} = y_i \frac{\partial}{\partial x_j}\log(y_i)$$

Now the partial of the log term is

$$\log y_i = \log\left(\frac{e^{x_i}}{\sum_l e^{x_l}} = x_i - \log\left(\sum_l e^{x_l}\right)\right. \tag{18}$$

$$\frac{\partial}{\partial x_j}\log(y_i) = \frac{\partial x_i}{\partial x_j} - \frac{\partial}{\partial x_j}\log\left(\sum_l e^{x_l}\right) \tag{19}$$

$$= 1_{i=j} - \frac{1}{\sum_l e^{x_l}} e^{x_j} \tag{20}$$

and plugging this back in gives

$$\frac{\partial y_i}{\partial x_j} = y_i(1_{i=j} - y_j) \tag{21}$$

It also turns out that the sigmoid is a specific case of the softmax. That is, given softmax for 2 classes, we have

$$o\begin{pmatrix}x_1\\x_2\end{pmatrix} = \frac{1}{e^{x_1}+e^{x_2}}\begin{pmatrix}e^{x_1}\\e^{x_2}\end{pmatrix} \tag{22}$$

So, the probability of being in class 1 is

$$\frac{e^{x_1}}{e^{x_1}+e^{x_2}} = \frac{1}{1+e^{x_2-x_1}} \tag{23}$$

and the logistic sigmoid is just a special case of the softmax function that avoids using redundant parameters. We actually end up overparameterizing the softmax because the probabilities must add up to one.

Definition 2.4 (Softmax Regression Model)

The softmax regression of K classes assumes a model of the form

$$h_\theta(x) = o(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{24}$$

where $\mathbf{W} \in \mathbb{R}^{K\times D}, \mathbf{b} \in \mathbb{R}^D$. Again, we have a linear map followed by some link function (the softmax) which allows us to nonlinearly map our unbounded linear outputs to some domain that can be easily parameterized by a probability distribution. In this case, our residual distribution is a **multinomial distribution**

$$y \sim \text{Multinomial}\big(h_{\mathbf{w}}(\mathbf{x})\big) = \text{Multinomial}\big([h_{\mathbf{w}}(\mathbf{x})]_1, \dots, [h_{\mathbf{w}}(\mathbf{x})]_K\big) \tag{25}$$

Definition 2.5 (Multiclass Cross Entropy Loss as Surrogate Loss for Softmax)

The surrogate loss for softmax regression is the **multiclass cross entropy loss**, which is defined as

$$L(\theta;\mathcal{D}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta(\mathbf{x}^{(i)})\right)_k \tag{26}$$

Theorem 2.2 (Maximum Likelihood Estimation for Softmax)

Since a closed form solution is not available for logistic regression, it is clearly not available for softmax. Therefore, we one hot encode our target variables as $\mathbf{y}^{(i)}$ and write our likelihood as

$$L(\theta; \mathcal{D}) = \prod_{i=1}^{N} \prod_{k=1}^{K} p(C_k \mid \mathbf{x}^{(i)})^{\mathbf{y}_k^{(i)}} = \prod_{i=1}^{N} \prod_{k=1}^{K} \left(\mathbf{h}_{\mathbf{W}}(\mathbf{x}^{(i)})\right)_k^{\mathbf{y}_k^{(i)}} \tag{27}$$

Taking the negative logarithm gives us the **cross entropy** loss function

$$\ell(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_k^{(i)} \log\left(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)_k = -\sum_{i=1}^{N} \mathbf{y}^{(i)} \dot{\log}\left(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right) \tag{28}$$

where $\cdot$ is the dot product. The gradient of this function may seem daunting, but it turns out to be very cute. Let us take a single sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, drop the index i, and write

$$\begin{aligned} \mathbf{x} &\mapsto \mathbf{W}\mathbf{x} + \mathbf{b} = \mathbf{z} \\ \hat{\mathbf{y}} = \mathbf{a} &= o(\mathbf{z}) \\ L &= -\mathbf{y} \cdot \log(\mathbf{a}) = -\sum_{k=1}^{K} y_k \log(a_k) \end{aligned}$$

We must compute

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}} \tag{29}$$

We can compute $\partial L / \partial \mathbf{z}$ as such, using our derivations for the softmax derivative above. We compute element wise.

$$\begin{aligned} \frac{\partial L}{\partial z_j} &= -\sum_{k=1}^{K} y_k \frac{\partial}{\partial z_j} \log(a_k) \\ &= -\sum_{k=1}^{K} y_k \frac{1}{a_k} \frac{\partial a_k}{\partial z_j} \\ &= -\sum_{k=1}^{K} \frac{y_k}{a_k} a_k (1_{\{k=j\}} - a_j) \\ &= -\sum_{k=1}^{K} y_k (1_{\{k=j\}} - a_j) \\ &= \left(\sum_{k=1}^{K} y_k a_j\right) - y_j \\ &= a_j \left(\sum_{k=1}^{K} y_k\right) - y_j \\ &= a_j - y_j \end{aligned}$$

and combining these gives

$$\frac{\partial L}{\partial \mathbf{z}} = (\mathbf{a} - \mathbf{y})^T \tag{30}$$

Now, computing $\partial \mathbf{z} / \partial \mathbf{W}$ gives us a 3-tensor, which is not ideal to work with. However, let us just

compute this with respect to the elements again. We have

$$z_k = \sum_{d=1}^{D} W_{kd} x_d + b_k$$
$$\frac{\partial z_k}{\partial W_{ij}} = \sum_{d=1}^{D} x_d \frac{\partial}{\partial W_{ij}} W_{kd}$$

where

$$\frac{\partial}{\partial W_{ij}} W_{kd} = \begin{cases} 1 & \text{if } i = k, j = d \\ 0 & \text{else} \end{cases} \tag{31}$$

Therefore, since d is iterating through all elements, the sum will only be nonzero if $k = i$. That is, $\frac{\partial z_k}{\partial W_{ij}} = x_j$ if $k = i$ and 0 if else. Therefore,

$$\frac{\partial \boldsymbol{z}}{\partial W_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{th element}$$

Now computing

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial W_{ij}} = (\mathbf{a} - \mathbf{y}) \frac{\partial \mathbf{z}}{\partial W_{ij}} = \sum_{k=1}^{K} (a_k - y_k) \frac{\partial z_k}{\partial W_{ij}} = (a_i - y_i) x_j \tag{32}$$

To get $\partial L / \partial W_{ij}$ we want a matrix whose entry (i, j) is $(a_i - y_i) x_j$. This is simply the outer product as shown below. For the bias term, $\partial \mathbf{z} / \partial \mathbf{b}$ is simply the identity matrix.

$$\frac{\partial L}{\partial \mathbf{W}} = (\mathbf{a} - \mathbf{y}) \mathbf{x}^T, \quad \frac{\partial L}{\partial \mathbf{b}} = \mathbf{a} - \mathbf{y} \tag{33}$$

Therefore, summing the gradient over some minibatch $M \subset [N]$ gives

$$\nabla_{\mathbf{W}} \ell_M = \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})(\mathbf{x}^{(i)})^T, \quad \nabla_{\mathbf{b}} \ell_M = \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \tag{34}$$

and our stochastic gradient descent algorithm is

$$\begin{aligned} \boldsymbol{\theta} = \begin{pmatrix} \mathbf{W} \\ \mathbf{b} \end{pmatrix} &= \begin{pmatrix} \mathbf{W} \\ \mathbf{b} \end{pmatrix} - \eta \begin{pmatrix} \nabla_{\mathbf{W}} \ell_M \\ \nabla_{\mathbf{b}} \ell_M \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{W} \\ \mathbf{b} \end{pmatrix} - \eta \begin{pmatrix} \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)})(\mathbf{x}^{(i)})^T \\ \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \end{pmatrix} \end{aligned}$$

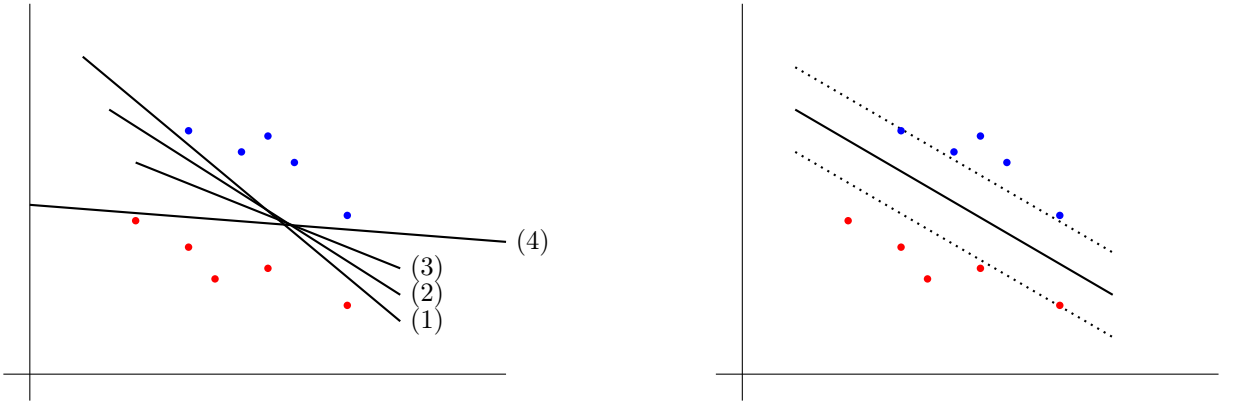
3 Support Vector Machines

Definition 3.1 (Hinge Loss)

The **hinge loss** is a convex surrogate loss function for the 0-1 loss function. It is defined as

$$L(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y}) \tag{35}$$

A support vector machine focuses only on the points that are most difficult to tell apart, whereas other classifiers pay attention all of the points. A SVM is a discriminative, non-probabilistic model. Let us first assume that our dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}$ is linearly separable with $y_i \in \{-1, +1\}$. Based on previous algorithms like the perceptron, it will find some separating hyperplane. However, there's an infinite number of separating hyperplanes as shown in Figure 4a. What support vector machines want to do is to find the best one, with the "best" defined as the hyperplane that maximizes the distance between either the closest positive or negative samples, shown in Figure 4*b*.



(a) Planes such as (1) and (4) are "too close" to the positive and negative samples.

(b) SVMs try to find the separating hyperplane with the best minimum margin.

Figure 4: Motivating problem

We want to formalize the concepts of these margins that we wish to maximize. To do this, we will define two terms.

Definition 3.2 (Geometric margin)

Given a point $\mathbf{x}_0$ and a hyperplane of equation $\mathbf{w} \cdot \mathbf{x} + b = 0$, the distance from $\mathbf{x}_0$ to the hyperplane, known as the **geometric margin**, can be computed with the formula

$$d = \frac{|\mathbf{x}_0 \cdot \mathbf{w} + b|}{||\mathbf{w}||} \tag{36}$$

Therefore, the geometric margin of the ith sample with respect to the hypothesis f is defined

$$\gamma_i = \frac{y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)}{||\mathbf{w}||} \tag{37}$$

We wish to optimize the parameters $\mathbf{w}, b$ in order to maximize the minimum of the geometric margins (the distance between the closest point and the hyperplane).

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_i \gamma_i = \underset{\mathbf{w},b}{\operatorname{argmax}} \left\{ \frac{1}{||\mathbf{w}||} \min_i \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \right] \right\} \tag{38}$$

Direct solution of this optimization problem would be very complex, and so we convert this into an equivalent problem that is much easier to solve. Note that the solution to the above term is not unique. If there was a solution $(\mathbf{w}^*, b^*)$, then

$$\frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{||\mathbf{w}||} = \frac{y_i(\lambda\mathbf{w} \cdot \mathbf{x}_i + \lambda b)}{||\lambda\mathbf{w}||} \tag{39}$$

That is, the geometric margin is not sensitive to scaling of the parameters of the hyperplane. Therefore, we can scale the numerator and the denominator by whatever we want and use this freedom to set

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$$

for the point that is closest to the surface. In that case, all data points will satisfy the constraints

$$y_n(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

In the case of data points for which the equality holds, the constraints are said to be *active*, whereas for the remainder they are *inactive*. Therefore, it will always be the case that $\min_i \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)\right] = 1$, and the constraint problem reduces to

$$\underset{\mathbf{w},b}{\text{argmax}} \frac{1}{||\mathbf{w}||} = \underset{\mathbf{w},b}{\text{argmin}} \frac{1}{2}||\mathbf{w}||^2 \text{ subject to constraints } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

This final step is the most significant step in this derivation and may be hard to wrap around the first time. So we dedicate the next subsubsection for this.

3.1 Functional and Geometric Margins

We could just work straight with this geometric margin, but for now, let's try to extend what we did with the perceptron into SVMs. We will find out that extending the concept of functional margins into SVMs leads to ill-defined problems. In the perceptron, we wanted to construct a function $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ such that

$$y_i\, f(\mathbf{x}_i) \geq 0 \text{ for all } i = 1, 2, \ldots, N$$

Definition 3.3 (Functional Margin)

The value of $y_i\, f(\mathbf{x}_i)$ gives us our confidence on our classification, and in a way it represents a kind of distance away from the separating hyperplane (if this value was 0, then we would be 50 50 split on whether to label it positive or negative). Therefore, we shall define

$$\hat{\gamma}_i = y_i f(\mathbf{x}_i)$$

as the **functional margin** of $(\mathbf{w}, b)$ with respect to the training sample $(\mathbf{x}_i, y_i)$. Therefore, the smallest of the function margins can be written

$$\hat{\gamma} = \min_i \gamma_i$$

called the **function margin**.

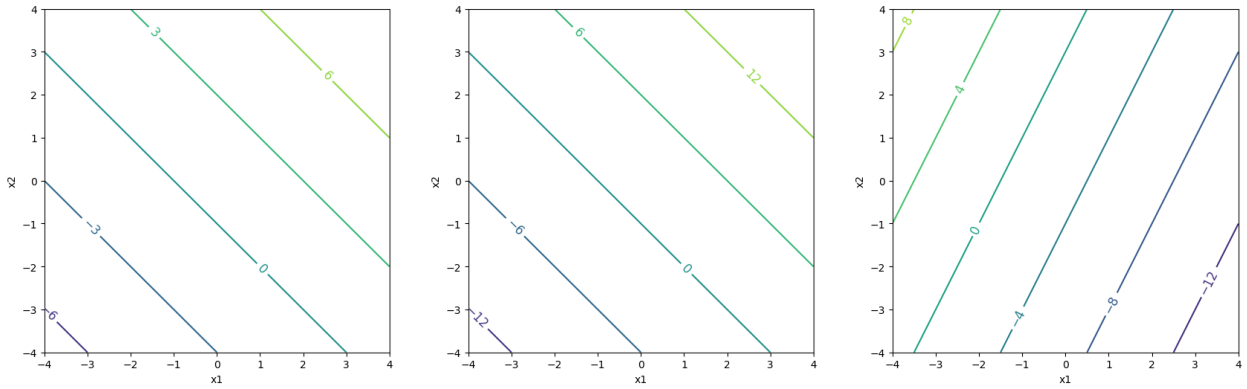
Note that the geometric margin and functional margin are related by a constant scaling factor. Given a sample $(\mathbf{x}_i, y_i)$, we have

$$\text{GeometricMargin} = \frac{y_i\left(\mathbf{w} \cdot \mathbf{x}_i + b\right)}{||\mathbf{w}||_2} = \frac{\text{FunctionalMargin}}{||\mathbf{w}||_2}$$

As we can see, the perceptron works with the functional margin, and since it does not care about how large the margin is (just whether it's positive or negative), we are left with an underdetermined system in which there exists infinite $(\mathbf{w}, b)$'s. Now what we want to do is impose a certain minimum margin $\gamma > 0$ and solve for $(\mathbf{w}, b)$ again, and keep increasing this γ until there is some unique solution. We can view this problem in two ways:

1. Take a specific minimum margin γ and find a $(\mathbf{w}, b)$, which may not exist, be unique, or exist infinitely that satisfies
$$y_i f(\mathbf{x}) = y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq \gamma \text{ for all } i = 1, \dots, N$$
2. Take a specific $(\mathbf{w}, b)$ and calculate the maximum γ that satisfies the constraint equations above.

They're both equivalent problems, but both ill-posed if we look at (2). Since the samples are linearly separable by assumption, we can say that there exists some $\epsilon > 0$ such that $y_i f(\mathbf{x}_i) \geq \epsilon$ for all i. Therefore, if we just scale $(\mathbf{w}, b) \mapsto (\lambda\mathbf{w}, \lambda b)$ for some large λ, this leads to the solution for γ being unbounded. We can see in Figure 5 that we can increased confidence at no cost. Looking at (1), we can also see that if $(\mathbf{w}, b)$ does exist, then every other $(\lambda\mathbf{w}, \lambda b)$ for $\lambda > 1$ satisfies the property.



(a) $f(x) = x_1 + x_2 + 1$ (b) $f(x) = 2x_1 + 2x_2 + 2$ (c) $f(x) = -2x_1 + x_2 - 3$

Figure 5: From (a), you can see that simply multiplying everything by two automatically increases our confidence by 2, meaning that the functional margin can be scaled arbitrarily by scaing $(\mathbf{w}, b)$. There are still too many degrees of freedom in here and so extra constraints must be imposed.

3.1.1 Lagrange Duality

To minimize the equations with the constraint equations, we can use the method of Lagrange multipliers, which leads to to Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_i \alpha_i \big[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1\big]$$

We can take the gradients with respect to $\mathbf{w}$ and b and set them to 0, which gives the two conditions

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$0 = \sum_i \alpha_i y_i \mathbf{x}_i$$

Now let's substitute our evaluated $\mathbf{w}$ back into $\mathcal{L}$, which gives the **dual representation** of the maximum

margin problem in which we maximize

$$\begin{aligned} L &= \frac{1}{2}\Big(\sum_i \alpha_i y_i \mathbf{x}_i\Big)\Big(\sum_j \alpha_j y_j \mathbf{x}_j\Big) - \sum_i \alpha_i y_i x_i \cdot \Big[\sum_j \alpha_j y_j x_j\Big] - \sum_i \alpha_i y_i b + \sum_i \alpha_i \\ &= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \, \mathbf{x}_i \cdot \mathbf{x}_j \end{aligned}$$

The summation with the b in it is 0 since we can pull the b out and the remaining sum is 0 from before. Now the optimization only depends on the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$ of all pairs of sample vectors, which is very interesting. We will see more of this when we talk about kernel methods. Now, we need to solve the dual problem

$$\max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha})$$

which can be done using some generic quadratic programming solver or some other method to get the optimum $\boldsymbol{\alpha}^*$, which gives us

$$\mathbf{w}^* = \sum_i \alpha_i^* y_i \mathbf{x}_i$$

3.2 Nonseparable Case

4 Gaussian/Linear Discriminant Analysis

4.1 Discriminative vs. Generative Models

Now we introduce our first example of a generative model, which introduces another division between models (in addition to parametric vs nonparametric, frequentist vs bayesian). Generally, there are two ways to model $\mathbb{P}_{Y|X=x}$.

Definition 4.1 (Discriminative Models)

Discriminative models attempt to do this directly by modeling only the conditional probability distribution $\mathbb{P}_{Y|X=x}$. We don't care about the underlying distribution of X, but rather we just want to try and predict Y given X. Essentially, we are trying to approximate the conditional expectation $h(X) = \mathbb{E}[Y \mid X]$, which is the best we can do. Given $X = x$, we use our model of $\mathbb{P}_{Y|X=x}$, and our hypothesis function will predict the its mean.

$$h(x) = \mathbb{E}[Y \mid X = x] \tag{40}$$

Definition 4.2 (Generative Models)

Generative models approximate this conditional probability by taking a more general approach. They attempt to model the joint probability distribution $\mathbb{P}_{X\times Y}$ (also called **inference**), which essentially gives everything we need about the data. Doing this allows us to *generate* more data (hence the name), which may be useful.
One way to approximate the joint distribution is to model the conditional distribution $\mathbb{P}_{X|Y=y}$, which gives the distribution of each labels. Now combined with the probability measure $\mathbb{P}_Y$, we can get the joint distribution. Usually in classification, the $\mathbb{P}_Y$ is easy to approximate (the MLE is simply the fequencies of the labels), so conventionally, modeling $\mathbb{P}_{X\times Y}$ and modeling $\mathbb{P}_{X|Y=y}$ are considered the same thing. Once we have these, we can calculate the joint distribution, but in high-dimensional spaces this tends to be computationally hard. Therefore, we usually resort to simply calculating $\mathbb{P}_{X|Y=y}$ and then using Bayes rule to approximate

$$\mathbb{P}_{Y|X} = \frac{\mathbb{P}_{X|Y}\mathbb{P}_Y}{\mathbb{P}_X} \tag{41}$$

where the normalizing term is computed using Monte Carlo simulations.

This is the first example of a generative model. In GDA, we basically write the likelihood as

$$\prod_{i=1}^{n} p(x_i, y_i) = \prod_i p(x_i \mid y_i)p(y_i) \tag{42}$$

where each $p(x_i \mid y_i)$ is Gaussian and $p(y_i)$ is Bernoulli. This specifies $p(x_i, y_i)$ and therefore is called a generative model. In logistic regression, we have

$$\prod_{i=1} p(x_i, y_i) = \left(\prod_i p(y_i \mid x_i)\right)\left(\prod_i p(x_i)\right) \tag{43}$$

and the first term is the logistic function and the second term is unknown. We only use the first part to classify, and this is a discriminative model. You can be agnostic about the data generating process and you can work with less data since there are less things to fit. Some people ask why should you model more unless you have to, so people tend to try to model the minimum, which is why logistic regression is more popular.

4.2 Construction

GDA assumes that $\mathbb{P}(x\,|y)$ is distributed according to a multivariate Gaussian distribution. Let us assume that the input space is d-dimensional and this is a binary classification problem. We set

$$\begin{aligned} y &\sim \text{Bernoulli}(\pi) \\ x\,|\,y=0 &\sim \mathcal{N}_d(\mu_0, \Sigma) \\ x\,|\,y=1 &\sim \mathcal{N}_d(\mu_1, \Sigma) \end{aligned}$$

This method is usually applied using only one covariance matrix Σ. The distributions are

$$\begin{aligned} p(y) &= \pi^y(1-\pi)^{1-y} \\ p(x\,|\,y=0) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) \\ p(x\,|\,y=1) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right) \end{aligned}$$

Now, what we have to do is optimize the distribution parameters $\pi \in (0,1)\mathbb{R}, \mu_0 \in \mathbb{R}^d, \mu_1 \in \mathbb{R}^d, \Sigma \in \text{Mat}(d\times d, \mathbb{R}) \simeq \mathbb{R}^{d\times d}$ so that we get the best-fit model. Assuming that each sample has been picked independently, this is equal to maximizing

$$L(\pi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{n} \mathbb{P}\big(x^{(i)}, y^{(i)}\,;\, \pi, \mu_0, \mu_1, \Sigma\big) \tag{44}$$

which is really just the probability that we get precisely all these training samples $(x^{(i)}, y^{(i)})$ given the 4 parameters. This can be done by optimizing its log-likelihood, which is given by

$$\begin{aligned} l(\pi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} \mathbb{P}(x^{(i)}, y^{(i)}; \pi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} \mathbb{P}(x^{(i)}\,|\,y^{(i)}; \mu_0, \mu_1, \Sigma)\,\mathbb{P}(y^{(i)}; \pi) \\ &= \sum_{i=1}^{n} \log\left(\mathbb{P}(x^{(i)}\,|\,y^{(i)}; \mu_0, \mu_1, \Sigma)\,\mathbb{P}(y^{(i)}; \pi)\right) \end{aligned}$$

and therefore gives the maximum likelihood estimate to be

$$\begin{aligned} \pi &= \frac{1}{N}\sum_{n=1}^{N} 1\{y^{(n)}=1\} \\ \boldsymbol{\mu}_0 &= \frac{\sum_{n=1}^{n} 1_{\{y^{(n)}=0\}}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} 1_{\{y^{(n)}=0\}}} \\ \boldsymbol{\mu}_1 &= \frac{\sum_{n=1}^{n} 1_{\{y^{(n)}=1\}}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} 1_{\{y^{(n)}=1\}}} \\ \boldsymbol{\Sigma} &= \frac{1}{N}\sum_{n=1}^{N} (\mathbf{x}^{(n)} - \mu_{y^{(n)}})(\mathbf{x}^{(n)} - \mu_{Y^{(i)}})^T \end{aligned}$$

A visual of the algorithm is below, with contours of the two Gaussian distributions, along with the straight line giving the decision boundary at which $\mathbb{P}(y=1\,|\,x) = 0.5$.

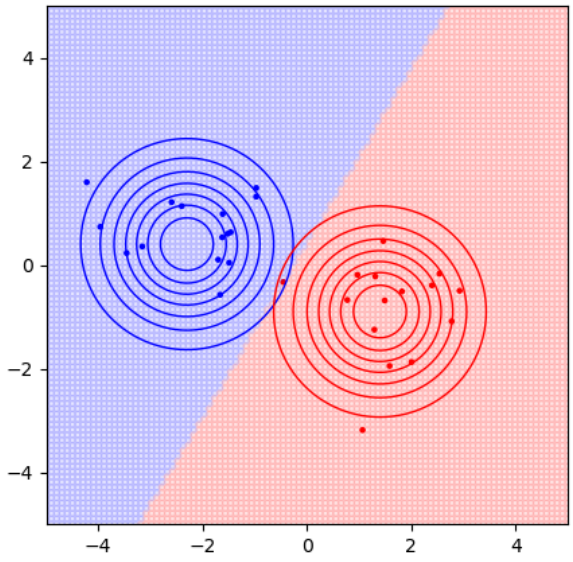


Figure 6: GDA of Data Generated from 2 Gaussisans centered at $(-2.3, 0.4)$ and $(1.4, -0.9)$ with unit covariance. The decision boundary is slightly off since MLE approximates the true means.

References