Linear Classification

Muchang Bahng

Spring 2025

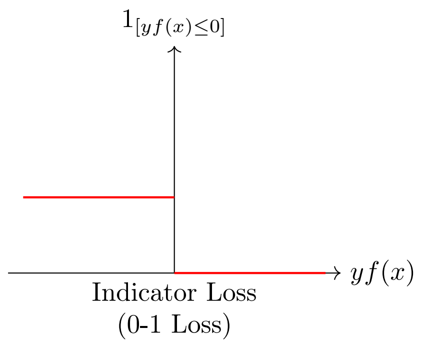
Contents

Now let's talk about linear classification. Remember that the MSE was a nice loss to choose due to the Gauss-Markov theorem, but is there an analogue for classification? Since we are now working in classification, a natural metric would be to see the proportion of samples we get correct, and this is modeled through the 0-1 loss function.

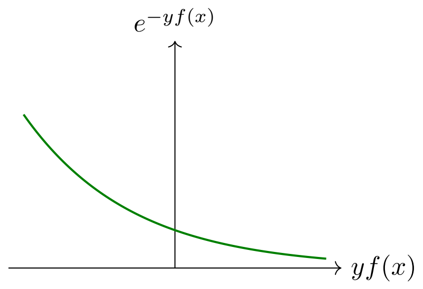
$$L(f, x, y) = \mathbb{1}(y = f(x)) \tag{1}$$

However, this isn't a smooth function to work with, with 0 gradients almost everywhere and hard to solve analytically. We can try a couple things.

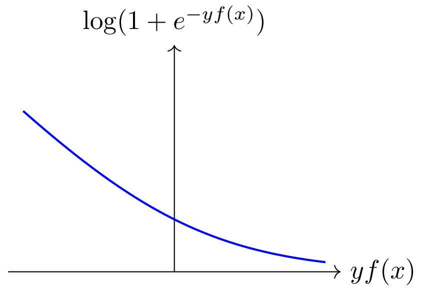
Alternatively, we can use a **surrogate loss function** to approximate the 0-1 loss function. The logistic uses some function, and the SVM uses the smallest convex function to approximate the 0-1 loss function. Here are some examples:



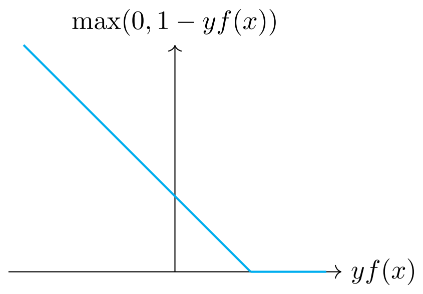
(a) Indicator/0-1 Loss can't be easily optimized.



(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

Intuitively, regression seems like a much harder problem than classification. One predicts over a finite set of values while the other predicts over a continuum. We can try and solve the problem as if we were going regression of values between 0 and 1, and then put a cutoff threshold where it classifies 0 if $f(x) \leq \frac{1}{2}$ and 1 if else. This is called the *Bayes' classifier*.

Theorem 0.1 (Minimality of Bayes Classifier)

The function f^* that minimizes the expected risk of the 0-1 loss is

$$f^*(x) = \begin{cases} 0 & \text{if } g(x) \leq \frac{1}{2} \\ 1 & \text{if } g(x) > \frac{1}{2} \end{cases} \tag{2}$$

where $g(x) = \mathbb{E}[Y \mid X = x] = \mathbb{P}(Y = 1 \mid X = x)$ denotes the regression function.

Proof.

It suffices to show that

$$\mathbb{P}(Y \neq h(X)|X = x) - \mathbb{P}(Y \neq h^*(X)|X = x) \geq 0 \quad \text{for all } x \in \mathcal{X}. \tag{3}$$

Now,

$$\begin{aligned}
\mathbb{P}(Y \neq h(X)|X = x) &= 1 - \mathbb{P}(Y = h(X)|X = x) &(4)\\
&= 1 - (\mathbb{P}(Y = 1, h(X) = 1|X = x) + \mathbb{P}(Y = 0, h(X) = 0|X = x)) &(5)\\
&= 1 - (h(x)\mathbb{P}(Y = 1|X = x) + (1 - h(x))\mathbb{P}(Y = 0|X = x)) &(6)\\
&= 1 - (h(x)m(x) + (1 - h(x))(1 - m(x))). &(7)
\end{aligned}$$

Hence,

$$\begin{aligned}
&\mathbb{P}(Y \neq h(X)|X = x) - \mathbb{P}(Y \neq h^*(X)|X = x) &(8)\\
&= (h^*(x)m(x) + (1 - h^*(x))(1 - m(x))) - (h(x)m(x) + (1 - h(x))(1 - m(x))) &(9)\\
&= (2m(x) - 1)(h^*(x) - h(x)) = 2\left(m(x) - \frac{1}{2}\right)(h^*(x) - h(x)) &(10)
\end{aligned}$$

When $m(x) \geq 1/2$ and $h^*(x) = 1$, (10) is non-negative. When $m(x) < 1/2$ and $h^*(x) = 0$, (10) is again non-negative.

In fact, by fitting a regression function over any data, we can simply compose it with the threshold function to get a binary classifier. This is called the **plug-in classifier**.

Definition 0.1 (Plug-In Classifier)

Given an regression estimator $\hat{g}$, we can define the corresponding **plug-in classifier** as

$$\hat{f}(x) = \begin{cases} 0 & \text{if } \hat{g}(x) \leq \frac{1}{2} \\ 1 & \text{if } \hat{g}(x) > \frac{1}{2} \end{cases} \tag{11}$$

Due to the optimality of the Bayes' classifier, the performance of the plug-in classifier is tied to the performance of the underlying linear regression model. We can actually formalize this with the following.

Theorem 0.2 (Classification Risk Bounded by Regression Risk)

The risk of the plug-in classifier rule in (14) satisfies

$$R(\hat{h}) - R^* \leq 2\sqrt{\int (\hat{m}(x) - m^*(x))^2 dP_X(x)}. \tag{12}$$

Proof.

In the proof of the previous theorem we showed that

$$\begin{aligned}
\mathbb{P}(Y \neq \hat{h}(X)|X = x) - \mathbb{P}(Y \neq h^*(X)|X = x) & &(13)\\
= (2\hat{m}(x) - 1)(h^*(x) - \hat{h}(x)) & &(14)\\
= |2\hat{m}(x) - 1| I(h^*(x) \neq \hat{h}(x)) & &(15)\\
= 2|\hat{m}(x) - 1/2| I(h^*(x) \neq \hat{h}(x)). & &(16)
\end{aligned}$$

Now, when $h^*(x) \neq \hat{h}(x)$, there are two possible cases: (i) $\hat{h}(x) = 1$ and $h^*(x) = 0$; (ii) $\hat{h}(x) = 0$ and $h^*(x) = 1$. In both cases, we have that $|\hat{m}(x) - m^*(x)| \geq |\hat{m}(x) - 1/2|$. Therefore,

$$\begin{aligned}
\mathbb{P}(\hat{h}(X) \neq Y) - \mathbb{P}(h^*(X) \neq Y) &= 2 \int |\hat{m}(x) - 1/2| I(h^*(x) \neq \hat{h}(x)) dP_X(x) && (17) \\
&\leq 2 \int |\hat{m}(x) - m^*(x)| I(h^*(x) \neq \hat{h}(x)) dP_X(x) && (18) \\
&\leq 2 \int |\hat{m}(x) - m^*(x)| dP_X(x) && (15) \\
&\leq 2 \sqrt{\int (\hat{m}(x) - m^*(x))^2 dP_X(x)}. && (16)
\end{aligned}$$

The last inequality follows from the fact that $\mathbb{E}|Z| \leq \sqrt{\mathbb{E}Z^2}$ for any Z.

1 Perceptron

The perceptron uses a linear regression function as a plugin-classifier, inspired by the artificial modeling of a human neuron [MP43].

Definition 1.1 (Perceptron)

The **perceptron model** is a discriminative linear classifier that assigns

$$f_w(x) = \begin{cases} 1 & \text{if } w^T x + b \geq 0 \\ -1 & \text{if } w^T x + b < 0 \end{cases} \tag{19}$$

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 $y f_w(x) > 0$, 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.

$$L(y, \hat{y}) = \sum_{n=1} [-\mathbf{w}^T \boldsymbol{\phi}^{(n)} y^{(n)}]_+ \text{ where } [f(\mathbf{x})]_+ := \begin{cases} f(\mathbf{x}) & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{else} \end{cases} \tag{20}$$

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.

```python
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

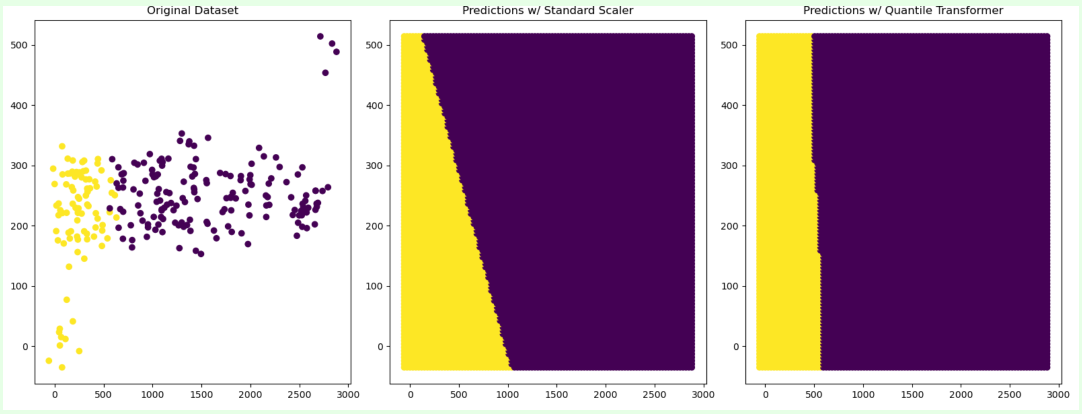


Figure 3: Perceptron Trained on Different Standardized Data

2 Logistic Regression

We can upgrade from a discriminant function to a discriminative probabilistic model with **logistic regression**. 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 function* $\sigma(x) \coloneqq \frac{1}{1+e^x}$ makes derivation of gradients very nice.

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{21}$$

Definition 2.1 (Logistic Regression)

The **logistic regression** model is probabilistic discriminative classification model

$$Y \mid X = x \sim \text{Bernoulli}(f(x)), \qquad f(x) = \sigma(\beta^T x) \tag{22}$$

where σ is the sigmoid function. It has parameters β.

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 was used as the *mean* μ of a Gaussian.
2. In logistic regression, the f 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 $(0, 1)$.[1]

Note that logistic regression has a *nonlinear regression function*, but a *linear decision boundary.* That is, by running this model through the plugin classifier, this becomes a linear classifier.

Theorem 2.1 (Decision Boundary is Linear)

The decision boundary of logistic regression is linear.

Proof.

We can solve

$$\frac{1}{1 + e^{\beta^T x}} = \frac{1}{2} \implies 2 = 1 + e^{\beta^T x} \implies 1 = e^{\beta^T x} \implies 0 = \beta^T x \tag{23}$$

which defines a linear hyperplane orthogonal to vector β.

2.1 Maximum Likelihood Estimation

Our next job is to define a loss to optimize. The pdf of a Bernoulli is not well-defined, but we can smoothly interpolate between the two points at $x = 0$ and $x = 1$ to get a smooth surrogate likelihood.

[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.

Lemma 2.1 (Likelihood for Bernoulli)

A surrogate likelihood of a Bernoulli(p) random variable is

$$\mathbb{P}(X = x) = p^x (1-p)^{1-x} \tag{24}$$

TBD. Why do we express it in this form? Just for convenience?

Proof.

Just substitute $\mathbb{P}(X = 1) = p^1(1-p)^0 = p$ and $\mathbb{P}(X = 0) = p^0(1-p)^1 = 1-p$.

Therefore, by taking the negative log-likelihood, we can get our loss function.

Definition 2.2 (Binary Cross Entropy Loss as Surrogate Loss)

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

$$L(y, \hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y}) \tag{25}$$

Now we can define our risk.

Theorem 2.2 (Risk)

The expected risk of logistic regression on the binary cross entropy loss is

$$R(f) = \mathbb{E}_{x,y}\left[-y\log(\sigma(\beta^T x)) - (1-y)\log(1-\sigma(\beta^T x))\right] \tag{26}$$

and the empirical risk on a dataset $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^n$ is

$$\hat{R}(f) = \frac{1}{n}\sum_{i=1}^n -y^{(i)}\log(\sigma(\beta^T x^{(i)})) - (1-y^{(i)})\log(1-\sigma(\beta^T x^{(i)})) \tag{27}$$

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

Theorem 2.3 (Gradients of Empirical Risk)

The gradient of the empirical risk is

$$\nabla_\beta \hat{R}(\beta) = \sum_{i=1}^n (y^{(i)} - f(x^{(i)}))x^{(i)} \tag{28}$$

Proof.

Take the negative sign out for a moment. The gradient for just a single sample $(x^{(i)}, y^{(i)})$ gives

$$\begin{aligned}
\frac{\partial \ell}{\partial \beta} &= \left(\frac{y^{(i)}}{\sigma(\beta^T x^{(i)})} - \frac{1-y^{(i)}}{1-\sigma(\beta^T x^{(i)})}\right)\frac{\partial}{\partial \beta}\sigma(\beta^T x^{(i)}) && (29)\\
&= \frac{\sigma(\beta^T x^{(i)}) - y^{(i)}}{\sigma(\beta^T x^{(i)})\left(1-\sigma(\beta^T x^{(i)})\right)}\sigma(\beta^T x^{(i)})\left(1-\sigma(\beta^T x^{(i)})\right)x^{(i)} && (30)\\
&= \left(f(x^{(i)}) - y^{(i)}\right)x && (31)
\end{aligned}$$

and now adding in the negative sign and summing it gives the result.

2.2 Significance Tests and Confidence Sets

2.3 Concentration Bounds

3 Softmax Regression

We would like to extend this to the multiclass case. In order to do this, we must start with *multivariate* linear regression and produce another link function o that maps it to the parameter space of a multinomial distribution. It should also be a generalization of the sigmoid.

Definition 3.1 (Softmax)

The **softmax function** is defined

$$o(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\|e^{\mathbf{x}}\|} = \frac{1}{\sum_j e^{x_j}} \begin{pmatrix} e^{x_1} \\ \vdots \\ e^{x_D} \end{pmatrix} \tag{32}$$

This is in fact a generalization of the sigmoid. That is, given softmax for 2 classes, we have

$$o\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \frac{1}{e^{x_1} + e^{x_2}} \begin{pmatrix} e^{x_1} \\ e^{x_2} \end{pmatrix} \tag{33}$$

So, the probability of being in class 1 is

$$\frac{e^{x_1}}{e^{x_1} + e^{x_2}} = \frac{1}{1 + e^{x_2 - x_1}} \tag{34}$$

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. Another reason to choose the softmax is that the total derivative turns out to simplify our loss, which also parallels to the sigmoid.

Lemma 3.1 (Derivative of Softmax)

The derivative of the softmax is

$$Do(\mathbf{x}) = \operatorname{diag}(o(\mathbf{x})) - o(\mathbf{x}) \otimes o(\mathbf{x}) \tag{35}$$

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

$$Do(\mathbf{x}) = \begin{pmatrix} y_1(1-y_1) & -y_1 y_2 & -y_1 y_3 & -y_1 y_4 \\ -y_2 y_1 & y_2(1-y_2) & -y_2 y_3 & -y_2 y_4 \\ -y_3 y_1 & y_3 y_3 & y_3(1-y_3) & -y_3 y_4 \\ -y_4 y_1 & -y_4 y_2 & -y_4 y_3 & y_4(1-y_4) \end{pmatrix} \tag{36}$$

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

$$\log y_i = \log\left(\frac{e^{x_i}}{\sum_l e^{x_l}} = x_i - \log\left(\sum_l e^{x_l}\right)\right) \tag{37}$$

$$\frac{\partial}{\partial x_j}\log(y_i) = \frac{\partial x_i}{\partial x_j} - \frac{\partial}{\partial x_j}\log\left(\sum_l e^{x_l}\right) \tag{38}$$

$$= 1_{i=j} - \frac{1}{\sum_l e^{x_l}} e^{x_j} \tag{39}$$

and plugging this back in gives

$$\frac{\partial y_i}{\partial x_j} = y_i(1_{i=j} - y_j) \tag{40}$$

A way to encode multiple classes is with one-hot encoding.

Definition 3.2 (One-Hot Encoding)

Given K classes $\{1, \ldots, K\}$, the **one-hot encoding** of each class is

$$k \mapsto e_k \in \mathbb{R}^K \tag{41}$$

where e_k is the kth basis vector.

We choose such an encoding since $\|e_k - e_{k'}\|$ is constant for all $k \neq k'$. Therefore, all classes are just as distinct from one another.

Definition 3.3 (Softmax Regression Model)

A **softmax regression** model of K classes is a probabilistic classification model

$$Y \mid X = x \sim \text{Multinomial}(f(x)), f(x) = o(Wx + b) \tag{42}$$

where o is the softmax function. It has the parameters $\theta = \{W \in \mathbb{R}^{k \times d}, b \in \mathbb{R}^k\}$.

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.

3.1 Maximum Likelihood Estimation

We do the same steps as that of logistic regression.

Lemma 3.2 (Likelihood for Multinomial)

The surrogate likelihood function for a multinomial is

$$\mathbb{P}(X = j) = \mathbb{P}(X = e_j) = \prod_{k=1}^{K} p_k^{(e_j)_k} \tag{43}$$

where we have one-hot encoded the kth class.

Proof.

All terms in the product will be 1 if $k \neq j$, and so the only term remaining will be $p_j^1 = p_j$.

By taking the negative log-likelihood, we can get our loss function.

Definition 3.4 (Multiclass Cross Entropy Loss as Surrogate Loss)

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

$$L(\theta, x, y) = -\sum_{k=1}^{K} y_k \log \big(f(x)\big)_k \tag{44}$$

Now we define our risk.

Theorem 3.1 (Risk)

The expected risk of softmax regression on the cross entropy loss is

$$R(f) = \mathbb{E}_{x,y}\left[-\sum_{k=1}^{K} y_k \log \big(f(x)\big)_k\right] \tag{45}$$

and the empirical risk on a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ is

$$\hat{R}(f) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k \log \big(f(x)\big)_k \tag{46}$$

Since a closed form solution is not available for logistic regression, it is clearly not available for softmax. Note that since we are taking the derivative of a vector w.r.t. a matrix, we will have to work with higher-order tensors. Fortunately, this reduces down to a cute form that we can still compute with matrices.

Theorem 3.2 (Gradient of Softmax Loss)

The gradient of the empirical risk is

$$\nabla_W \hat{R}(f) = \sum_{i=1}^{n} (f(x^{(i)}) - y^{(i)})(x^{(i)})^T \tag{47}$$

$$\nabla_b \hat{R}(f) = \sum_{i=1}^{n} (f(x^{(i)}) - y^{(i)}) \tag{48}$$

Proof.

Let's write the cross entropy loss function

$$\ell(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)} \log \big(h_\theta(x^{(i)})\big)_k = -\sum_{i=1}^{N} y^{(i)} \cdot \log \big(h_\theta(x^{(i)})\big) \tag{49}$$

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 $(x^{(i)}, y^{(i)})$, drop the index i, and write

$$z = Wx + b = z \tag{50}$$

$$\hat{y} = a = o(z) \tag{51}$$

$$L = -y \cdot \log(a) = -\sum_{k=1}^{K} y_k \log(a_k) \tag{52}$$

We must compute

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial \theta} \tag{53}$$

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

$$\begin{aligned}
\frac{\partial L}{\partial z_j} &= -\sum_{k=1}^{K} y_k \frac{\partial}{\partial z_j} \log(a_k) && (54)\\
&= -\sum_{k=1}^{K} y_k \frac{1}{a_k}\frac{\partial a_k}{\partial z_j} && (55)\\
&= -\sum_{k=1}^{K} \frac{y_k}{a_k} a_k(1_{\{k=j\}} - a_j) && (56)\\
&= -\sum_{k=1}^{K} y_k(1_{\{k=j\}} - a_j) && (57)\\
&= \left(\sum_{k=1}^{K} y_k a_j\right) - y_j && (58)\\
&= a_j\left(\sum_{k=1}^{K} y_k\right) - y_j && (59)\\
&= a_j - y_j && (60)
\end{aligned}$$

and combining these gives

$$\frac{\partial L}{\partial z} = (a-y)^T \tag{61}$$

Now, computing $\partial z/\partial W$ 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

$$z_k = \sum_{d=1}^{D} W_{kd}x_d + b_k, \qquad \frac{\partial z_k}{\partial W_{ij}} = \sum_{d=1}^{D} x_d \frac{\partial}{\partial W_{ij}} W_{kd} \tag{62}$$

where

$$\frac{\partial}{\partial W_{ij}} W_{kd} = \begin{cases} 1 & \text{if } i=k, j=d \\ 0 & \text{else} \end{cases} \tag{63}$$

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.

$$\frac{\partial z}{\partial W_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{th element} \tag{64}$$

Now computing

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial W_{ij}} = (a-y)\frac{\partial z}{\partial W_{ij}} = \sum_{k=1}^{K}(a_k - y_k)\frac{\partial z_k}{\partial W_{ij}} = (a_i - y_i)x_j \tag{65}$$

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, $\partial z / \partial b$ is simply the identity matrix.

$$\frac{\partial L}{\partial W} = (a - y) x^T, \quad \frac{\partial L}{\partial b} = a - y \tag{66}$$

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

$$\nabla_W \ell_M = \sum_{i \in M} (h_\theta(x^{(i)}) - y^{(i)})(x^{(i)})^T, \quad \nabla_b \ell_M = \sum_{i \in M} (h_\theta(x^{(i)}) - y^{(i)}) \tag{67}$$

and our stochastic gradient descent algorithm is

$$\begin{aligned} \theta = \begin{pmatrix} W \\ b \end{pmatrix} &= \begin{pmatrix} W \\ b \end{pmatrix} - \eta \begin{pmatrix} \nabla_W \ell_M \\ \nabla_b \ell_M \end{pmatrix} \\ &= \begin{pmatrix} W \\ b \end{pmatrix} - \eta \begin{pmatrix} \sum_{i \in M} (h_\theta(x^{(i)}) - y^{(i)})(x^{(i)})^T \\ \sum_{i \in M} (h_\theta(x^{(i)}) - y^{(i)}) \end{pmatrix} \end{aligned}$$

3.2 Significance Tests and Confidence Sets

3.3 Concentration Bounds

4 Regularized Softmax Regression

4.1 Ridge

In the high dimensional case, we would like to impose some regularization again to control variance.

Definition 4.1 (Loss)

The loss function of a ridge logistic regression is

$$\begin{aligned} L(\beta, x, y) &= -y \log(\sigma(\beta^T x)) - (1-y)\log(1 - \sigma(\beta^T x)) && (68) \\ &= -y \log(\hat{y}) - (1-y)\log(1-\hat{y}) && (69) \end{aligned}$$

The loss for a ridge softmax regression is

$$L(\theta, x, y) = -\sum_{k=1}^{K} y_k \log() \tag{70}$$

4.2 Lasso

5 Gaussian Discriminant Analysis

Another intuitive way to model binary data is to assume that the distribution of each class is a Gaussian. This allows us to work with the likelihood, and depending on which likelihood is greater, we can classify it at such.

Gaussian discriminant analysis in general is not actually a linear model. It turns out that because the two Gaussians can have different covariance matrices, we end up getting a quadratic boundary. However, if we restrict the two to have the *same* covariance matrix, we end up with *linear discriminant analysis*.

Definition 5.1 (Linear Discriminant Analysis)

The **LDA model** is a generative binary classification model that assumes the data is generated as

$$y \sim \text{Bernoulli}(\pi) \tag{71}$$
$$x \mid y = 0 \sim \mathcal{N}(\mu_0, \Sigma) \tag{72}$$
$$x \mid y = 1 \sim \mathcal{N}(\mu_1, \Sigma) \tag{73}$$

where the parameters are $\theta = \{\pi, \mu_0, \mu_1, \Sigma\}$. Sometimes, the shared covariance matrix Σ is assumed to be isotropic. This results in

$$f(x) = \begin{cases} 0 & \text{if } p(x \mid y = 0) \geq p(x \mid y = 1) \\ 1 & \text{if else} \end{cases} \tag{74}$$

Note that since we are creating a model of how the data is—not only distributed—but *generated*, we can also generate new data.

Theorem 5.1 (Decision Boundary is Linear)

Proof.

5.1 Comparison to Logistic Regression

Note that we can equivalently view LDA as a plug-in classifier with the Bayes rule. Let our posterior distribution be

$$p(y = 1 \mid x) = \frac{p(x \mid y = 1)\, \pi}{p(x \mid y = 0)\,(1 - \pi) + p(x \mid y = 1)\, \pi} \tag{75}$$

and we can threshold this to be $\frac{1}{2}$.

5.2 Maximum Likelihood Estimate

Lemma 5.1 (Likelihood)

The likelihood of a sample is

$$p(y) = \pi^y (1 - \pi)^{1-y} \tag{76}$$
$$p(x \,|\, y = 0) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\bigg(-\frac{1}{2} (x - \mu_0)^T \Sigma^{-1} (x - \mu_0) \bigg) \tag{77}$$
$$p(x \,|\, y = 1) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\bigg(-\frac{1}{2} (x - \mu_1)^T \Sigma^{-1} (x - \mu_1) \bigg) \tag{78}$$

Proof.

Theorem 5.2 (MLE)

The maximum likelihood estimate of LDA is

$$\pi = \frac{1}{N} \sum_{n=1}^{N} 1\{y^{(n)} = 1\} \tag{79}$$

$$\boldsymbol{\mu}_0 = \frac{\sum_{n=1}^{n} 1_{\{y^{(n)}=0\}} \mathbf{x}^{(n)}}{\sum_{n=1}^{N} 1_{\{y^{(n)}=0\}}} \tag{80}$$

$$\boldsymbol{\mu}_1 = \frac{\sum_{n=1}^{n} 1_{\{y^{(n)}=1\}} \mathbf{x}^{(n)}}{\sum_{n=1}^{N} 1_{\{y^{(n)}=1\}}} \tag{81}$$

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}^{(n)} - \mu_{y^{(n)}})(\mathbf{x}^{(n)} - \mu_{Y^{(i)}})^T \tag{82}$$

Proof.

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

$$L(\pi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{n} \mathbb{P}\big(x^{(i)}, y^{(i)}\,;\, \pi, \mu_0, \mu_1, \Sigma\big) \tag{83}$$

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

$$\begin{aligned} l(\pi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} \mathbb{P}(x^{(i)}, y^{(i)}; \pi, \mu_0, \mu_1, \Sigma) && (84) \\ &= \log \prod_{i=1}^{n} \mathbb{P}(x^{(i)} \,|\, y^{(i)}; \mu_0, \mu_1, \Sigma)\, \mathbb{P}(y^{(i)}; \pi) && (85) \\ &= \sum_{i=1}^{n} \log \Big(\mathbb{P}(x^{(i)} \,|\, y^{(i)}; \mu_0, \mu_1, \Sigma)\, \mathbb{P}(y^{(i)}; \pi) \Big) && (86) \end{aligned}$$

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 $\mathbb{P}(y = 1 \,|\, x) = 0.5$.

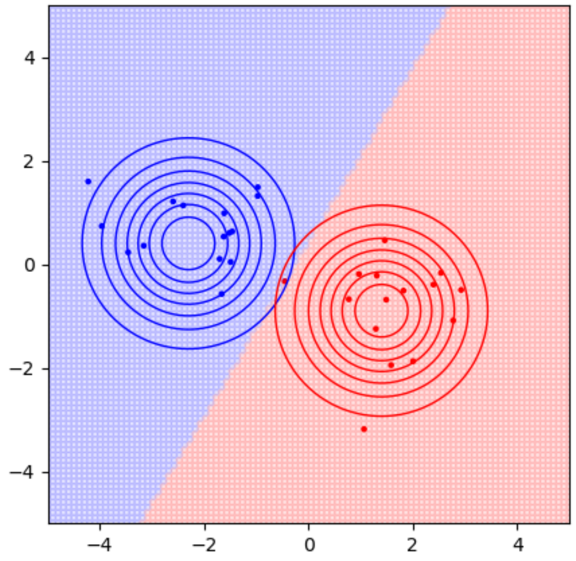


Figure 4: GDA of Data Generated from 2 Gaussisans centered at $(-2.3, 0.4)$ and $(1.4, -0.9)$ with unit covariance. The decision boundary is slightly off since MLE approximates the true means.

5.3 Quadratic Discriminant Analysis

Now let's do the general form.

Definition 5.2 (Gaussian Discriminant Analysis)

The **GDA model** is a generative binary classification model that assumes the data is generated as

$$y \sim \text{Bernoulli}(\pi) \tag{87}$$
$$x \mid y = 0 \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{88}$$
$$x \mid y = 1 \sim \mathcal{N}(\mu_1, \Sigma_1) \tag{89}$$

where the parameters are $\theta = \{\pi, \mu_0, \mu_1, \Sigma_0, \Sigma_1\}$. This results in

$$f(x) = \begin{cases} 0 & \text{if } p(x \mid y = 0) \geq p(x \mid y = 1) \\ 1 & \text{if else} \end{cases} \tag{90}$$

Theorem 5.3 (Maximum Likelihood Estimate)

We can simplify the computation to the thresholding.

Theorem 5.4 ()

If $X \mid Y = 0 \sim N(\mu_0, \Sigma_0)$ and $X \mid Y = 1 \sim N(\mu_1, \Sigma_1)$, then the Bayes rule is

$$f^*(x) = \begin{cases} 1 & \text{if } r_1^2 < r_0^2 + 2\log\left(\frac{\pi_1}{1-\pi_1}\right) + \log\left(\frac{|\Sigma_0|}{|\Sigma_1|}\right) \\ 0 & \text{otherwise} \end{cases} \tag{91}$$

where $r_i^2 = (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)$ for $i = 1, 2$ is the Mahalanobis distance.

Proof.

By definition, the Bayes rule is $h^*(x) = I(\pi_1 p_1(x) > (1 - \pi_1) p_0(x))$. Plug-in the specific forms of p_0 and p_1 and take the logarithms we get $h^*(x) = 1$ if and only if

$$(x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) - 2\log \pi_1 + \log(|\Sigma_1|) \tag{92}$$
$$< (x - \mu_0)^T \Sigma_0^{-1} (x - \mu_0) - 2\log(1 - \pi_1) + \log(|\Sigma_0|). \tag{93}$$

The theorem immediately follows from some simple algebra.

Theorem 5.5 (Decision Boundary is Quadratic)

Proof.

5.4 Multiclass GDA

Theorem 5.6 ()

Let $R(f) = \mathbb{P}(f(X) \neq Y)$ be the classification error of a classification rule $f(x)$. The Bayes rule $f^*(X)$ minimizing $R(f)$ can be written as

$$f^*(x) = \operatorname*{argmax}_k \mathbb{P}(Y = k \mid X = x) \tag{94}$$

Proof.

We have

$$R(f) = 1 - \mathbb{P}(f(X) = Y) \tag{95}$$
$$= 1 - \sum_{k=0}^{K-1} \mathbb{P}(f(X) = k, Y = k) \tag{96}$$
$$= 1 - \sum_{k=0}^{K-1} \mathbb{E}\left[I(f(X) = k)\mathbb{P}(Y = k \mid X)\right] \tag{97}$$

It's clear that $f^*(X) = \text{argmax}_k \mathbb{P}(Y = k \mid X)$ achieves the minimized classification error $1 - \mathbb{E}[\max_k \mathbb{P}(Y = k \mid X)]$.

Let $\pi_k = \mathbb{P}(Y = k)$. The next theorem extends QDA and LDA to the multiclass setting.

Theorem 5.7 ()

Suppose that $Y \in \{0, \ldots, K-1\}$ with $K \geq 2$. If $p_k(x) = p(x \mid Y = k)$ is Gaussian: $X \mid Y = k \sim N(\mu_k, \Sigma_k)$, the Bayes rule for the multiclass QDA can be written as

$$f^*(x) = \operatorname*{argmax}_k \delta_k(x) \tag{98}$$

where

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log \pi_k. \tag{99}$$

If all Gaussians have an equal variance Σ, then

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k. \tag{100}$$

Let $n_k = \sum_i \mathbb{1}(y_i = k)$ for $k = 0, \ldots, K-1$. The estimated sample quantities of π_k, μ_k, Σ_k, and Σ are:

$$\hat{\pi}_k = \frac{1}{n} \sum_{i=1}^n \mathbb{1}(y_i = k), \quad \hat{\mu}_k = \frac{1}{n_k} \sum_{i : Y_i = k} X_i, \tag{101}$$

$$\hat{\Sigma}_k = \frac{1}{n_k - 1} \sum_{i : Y_i = k} (X_i - \hat{\mu}_k)(X_i - \hat{\mu}_k)^T, \tag{102}$$

$$\hat{\Sigma} = \frac{\sum_{k=0}^{K-1} (n_k - 1) \hat{\Sigma}_k}{n - K}. \tag{103}$$

6 Support Vector Machines

Remember that it was the assumption that the conditional distribution $Y \mid X = x$ being Bernoulli/multinomial, along with the power form of the surrogate likelihood, that led to the likelihood derivation of the surrogate loss of the logistic and softmax regression. But ultimately, these are just modeling assumptions, and we may look for other surrogate losses to construct our risk.

If we want to construct such a loss function, we first want it to approximate the 0-1 step function clearly. The next thing is that we want it to be convex, so we can use convex optimization (sum of convex functions are convex). Perhaps we can think of the *smallest* convex function that stays above the binary loss, in some sense the best approximation. This is precisely the hinge function.

Definition 6.1 (Hinge Loss)

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

$$L(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y}) \tag{104}$$

Note that we have directly constructed a loss without any reference to the likelihood of the data points. Now the SVM model is really simple. We just have a linear model $g(x) = \beta^T x$, and then use it to make a plug-in classifier through a threshold function with threshold 0.

Definition 6.2 (Binary Support Vector Machine)

The **binary SVM** is a linear classifier that sets

$$f(x) = \operatorname{sign}(F(x)) = \operatorname{sign}(\beta^T x) = \begin{cases} 1 & \text{if } \beta^T x \geq 0 \\ 0 & \text{else} \end{cases} \tag{105}$$

With the model and loss set up, we can define our risk.

Theorem 6.1 (Risk)

The expected risk is

$$R(f) = \mathbb{E}_{x,y}\left[\max\{0, 1 - yF(x)\}\right] = \mathbb{E}_{x,y}\left[\max\{0, 1 - y(\beta^T x)\}\right] \tag{106}$$

and the empirical risk is

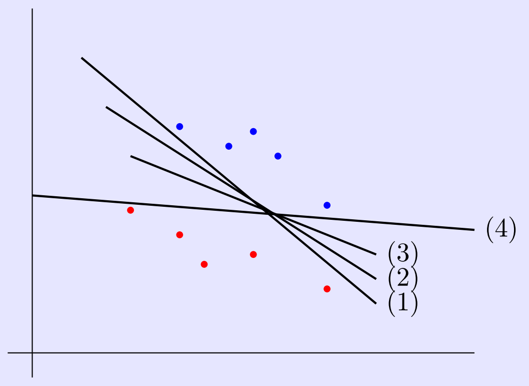
$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y(\beta^T x)\} \tag{107}$$

Note that if we classified something correctly, then $y(\beta^T x)$ would be positive, leading to the loss term being cut off to 0.[2] So the model will focus only on the points that are wrong or that are most difficult to tell apart. This is a big difference between SVMs and other classifiers.

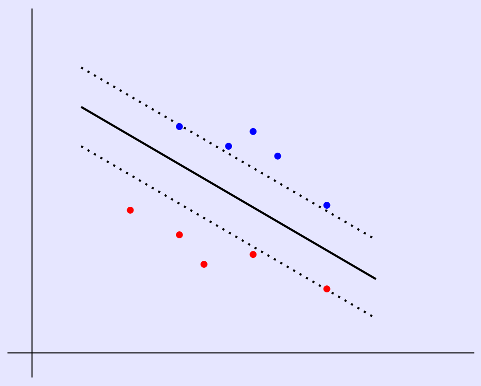
Example 6.1 (SVMs vs Other Classifiers on Linearly Separable Dataset)

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 5a. 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 5*b*.

[2]Unless $\beta^T x$ was a very small positive number, but speaking loosely here.



(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 5: Motivating problem

We want to formalize the concepts of these margins that we wish to maximize. Furthermore, this problem is not well defined since we can just set w to be 0 or an arbitrarily high norm vector, which makes this problem ill-posed. Therefore, we will need some constraints as well.

6.1 Functional and Geometric Margins

To do this, we will define two terms.

Definition 6.3 (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

$$d = \frac{|\mathbf{x}_0 \cdot \mathbf{w} + b|}{||\mathbf{w}||} \tag{108}$$

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

$$\gamma_i = \frac{y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)}{||\mathbf{w}||} \tag{109}$$

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).

$$\underset{\mathbf{w},b}{\text{argmax}} \min_i \gamma_i = \underset{\mathbf{w},b}{\text{argmax}} \left\{ \frac{1}{||\mathbf{w}||} \min_i \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)\right] \right\} \tag{110}$$

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

$$\frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{||\mathbf{w}||} = \frac{y_i(\lambda \mathbf{w} \cdot \mathbf{x}_i + \lambda b)}{||\lambda \mathbf{w}||} \tag{111}$$

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 $\min_i \big[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)\big] = 1$, and the constraint problem reduces to

$$\underset{\mathbf{w},b}{\text{argmax}} \frac{1}{||\mathbf{w}||} = \underset{\mathbf{w},b}{\text{argmin}} \frac{1}{2}||\mathbf{w}||^2 \text{ subject to constraints } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \tag{112}$$

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.

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 6.4 (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**.

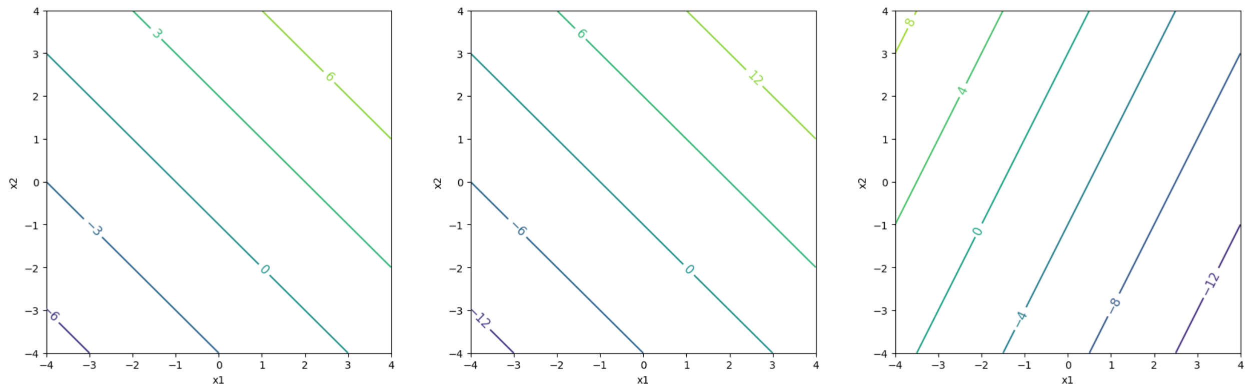
Note that the geometric margin and functional margin are related by a constant scaling factor. Given a sample $(\mathbf{x}_i, y_i)$, we have

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

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
$$y_i f(\mathbf{x}) = y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq \gamma \text{ for all } i = 1, \ldots, N$$
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 6 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 6: 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.

6.2 Analytical Solution

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

$$\begin{aligned} \mathbf{w} &= \sum_i \alpha_i y_i \mathbf{x}_i \\ 0 &= \sum_i \alpha_i y_i \mathbf{x}_i \end{aligned}$$

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

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

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

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

6.3 Significance Tests and Confidence Sets

6.4 Concentration Bounds

6.5 Nonseparable Case

References

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.