Notes of Stanford CS229

MoeKid102

Contents

1 Generalized Linear Models

Generalized Linear Model is designed for supervised learning problems. Given $\{(\boldsymbol{x}_i, y_i)|i = 1, 2, ..., N\}$ (where $\boldsymbol{x} \in \mathbb{R}^n$) as the training set, we construct our hypothesis $h : \mathbb{R}^n \to \mathbb{R}$ and try to optimize it.

For GLM specifically, just as its name "linear" suggests, we assume that $\exists\boldsymbol{\theta} \in \mathbb{R}^n$ s.t. $y = f(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x})$ models our inputs well (where $f(\cdot) : \mathbb{R} \to \mathbb{R}$ is some function).

1.1 The exponential family

• **Def Exponential Family**: a random variable X is said to be in the exponential family if for some canonical parameter η,

$$f_X(x|\eta) = b(x)\exp(\eta T(x) - a(\eta)) \tag{1}$$

where $T(x)$ is the sufficient statistic; $a(\eta)$, the log partition function, could be seen as some coefficient restricting the integration of P.D.F. to 1.

• Most of the common distributions belongs to the exponential family, e.g.

$$\begin{aligned}
\text{Norm}(\mu, \sigma^2)(x) &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) &&= \exp\left(-\frac{x^2}{2\sigma^2}\right) \times \exp(\frac{\mu x}{\sigma^2}) \times \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{\mu^2}{2\sigma^2})\\
\text{Ber}(p)(x) &= p^x(1-p)^{1-x} &&= 1 \times \exp\left(x\ln\frac{p}{1-p}\right) \times (1-p)\\
\text{Pois}(\lambda)(x) &= \frac{e^{-\lambda}\lambda^x}{x!} &&= \frac{1}{x!} \times \exp\left(x\ln\lambda\right) \times \exp\{-e^{\ln\lambda}\}
\end{aligned}$$

• **Thm** The moment generating function of $T(X)$ is $M(t) = \exp(a(t+\eta) - a(\eta))$, and therefore we have $\mathrm{E}[T(X)|\eta] = \dfrac{\partial a(\eta)}{\partial\eta}$, $\mathrm{Var}(T(X)|\eta) = \dfrac{\partial^2 a(\eta)}{\partial\eta^2}$.

Proof Since $f_X(x|\eta)$ must integrate to 1, we have

$$a(\eta) = \ln\left(\int_{-\infty}^{+\infty} b(x)e^{\eta T(x)}\mathrm{d}x\right)$$

$$\begin{aligned}
M(t) &= \mathrm{E}[e^{tT(X)}]\\
&= \int_{-\infty}^{+\infty} b(x)\cdot e^{(t+\eta)T(x)-a(\eta)}\mathrm{d}x\\
&= \int_{-\infty}^{+\infty} b(x)\cdot e^{(t+\eta)T(x)-a(t+\eta)}\mathrm{d}x \times e^{a(t+\eta)-a(\eta)}\\
&= e^{a(t+\eta)-a(\eta)}
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}[T(X)|\eta] = M'(0) &= \left(e^{a(t+\eta)-a(\eta)}\frac{\partial a(t+\eta)}{\partial t}\right)_{t=0}\\
&= \frac{\partial a(t+\eta)}{\partial(t+\eta)}_{t+\eta=\eta} = \frac{\partial a(\eta)}{\partial\eta}
\end{aligned}$$

$$\begin{aligned}\mathrm{Var}(T(X)|\eta) &= M''(0) - \left(\frac{\partial a(\eta)}{\partial \eta}\right)^2 \\ &= \left(M'(t)\frac{\partial a(t+\eta)}{\partial \eta}\right)_{t=0} + \left(M(t)\frac{\partial^2 a(t+\eta)}{\partial t^2}\right)_{t=0} - \left(\frac{\partial a(\eta)}{\partial \eta}\right)^2 \\ &= \frac{\partial^2 a(t+\eta)}{\partial t^2}_{t=0} = \frac{\partial^2 a(\eta)}{\partial \eta^2}\end{aligned}$$

Q.E.D.

1.2 Generate a GLM

• Suppose that we are the "god" who is generating the data for scientists to observe and record. We generate data with a function $h_0 : \mathbb{R}^n \to \mathbb{R}$. After getting a series of points $(\boldsymbol{x}^i, h_0(\boldsymbol{x}^i))$, the outputs presented to humans are $(\boldsymbol{x}^i, y^i)$ where y^i is randomly selected according to a certain distribution $p(y^i|\boldsymbol{x}^i)$, each y^i independent of any other $y^j (j \neq i)$. Specifically, for GLMs, the distribution is among **Exponential Family**, whose parameter $\eta := \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}^i$.

Take linear regression as an example, our function is $h_0(\boldsymbol{x}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$. The distribution of final result y^i has the distribution

$$p(y^i|\eta(\boldsymbol{x}^i)) = \mathrm{Norm}(\eta(\boldsymbol{x}^i), \sigma^2)(y^i) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\left(\frac{y^i - \eta(\boldsymbol{x}^i)}{\sigma}\right)^2\right)$$

• Now from the view of human scientists, we get a bunch of data $\{(\boldsymbol{x}^i, y^i)|i = 1, 2, ..., n\}$. We know the data is generated by a distribution $y^i : p(y^i|\eta(\boldsymbol{x}^i))$ and even the exact form of the P.D.F., but we don't know what $\boldsymbol{\theta}^{\mathrm{T}}$ is. So our goal is to select $\boldsymbol{\theta}$ such that the **likelihood** of $\boldsymbol{\theta}$ defined by $\mathcal{L}(\boldsymbol{\theta}) := p(\hat{y}|\hat{\boldsymbol{x}}; \boldsymbol{\theta})$ is maximized, i.e. we are using maximum likelihood evaluation. Since the selection of each $\boldsymbol{x}^i$ is independent, we have our target

$$\underset{\boldsymbol{\theta}}{max}\, \mathcal{L}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{max} \prod_{i=1}^{n} p(y^i|\boldsymbol{x}^i; \boldsymbol{\theta}) \tag{2}$$

If we still consider linear regression, it naturally follows that we should minimize MSELoss defined by

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

Back to GLM itself, there leaves something for us to discover. Since we want a maximum $\mathcal{L}(\boldsymbol{\theta})$, we require its deriative to be 0.

$$\begin{aligned}\underset{\boldsymbol{\theta}}{max}\, \mathcal{L}(\boldsymbol{\theta}) \Leftrightarrow \underset{\boldsymbol{\theta}}{max} \prod_{i=1}^{n} p(y^i|\boldsymbol{x}^i; \boldsymbol{\theta}) \Leftrightarrow \underset{\boldsymbol{\theta}}{max} \sum_{i=1}^{n} \eta(\boldsymbol{x}^i)T(y^i) - a(\eta(\boldsymbol{x}^i)) \\ \Leftrightarrow \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{n} \eta(\boldsymbol{x}^i)T(y^i) - a(\eta(\boldsymbol{x}^i)) = 0 \Leftrightarrow \sum_{i=1}^{n} \boldsymbol{x}^i\left(T(y^i) - \mathrm{E}[T(y^i)|\eta(\boldsymbol{x}^i)]\right) = 0\end{aligned}$$

Suppose our distribution is so well-selected as it satisfies the property that $\mathrm{E}[T(y^i)|\boldsymbol{x}^i; \boldsymbol{\theta}] = h_0(\boldsymbol{x}^i; \boldsymbol{\theta})$. (It's obvious that Bernoulli and Normal distribution both satisfy this property.) Therefore we can treat $\mathrm{E}[T(y^i)|\boldsymbol{x}^i; \boldsymbol{\theta}]$ as the output of our current model $h_{\boldsymbol{\theta}}(\boldsymbol{x}^i)$. It then follows that our update policy can be simplified (no matter what task we are faced with) to

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{\theta} + \alpha \sum_{i=1}^{n} \boldsymbol{x}^i \left(T(y^i) - h_{\boldsymbol{\theta}}(\boldsymbol{x}^i) \right) \tag{3}$$

1.3 Convexity of GLM

Furthermore, the equation above leads to not only an arbitrary local maximum, but a global maximum because the loss function $\mathcal{L}(\boldsymbol{\theta})$ is concave, which we can prove through the positive semidefiniteness of its Hessian matrix $\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$.

$$\begin{aligned}
\nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \boldsymbol{\theta}} \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \right) = \frac{\partial}{\partial \boldsymbol{\theta}} \left(\sum_{i=1}^{n} \boldsymbol{x}^i \left(T(y^i) - \frac{\partial a(\eta(\boldsymbol{x}^i; \boldsymbol{\theta}))}{\partial \eta} \right) \right) \\
&= -\sum_{i=1}^{n} \boldsymbol{x}^i \frac{\partial^2 a(\eta(\boldsymbol{x}^i; \boldsymbol{\theta}))}{\partial \eta^2} \frac{\partial \eta(\boldsymbol{x}^i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\sum_{i=1}^{n} \boldsymbol{x}^i \frac{\partial^2 a(\eta(\boldsymbol{x}^i; \boldsymbol{\theta}))}{\partial \eta^2} \boldsymbol{x}^{i^{\mathrm{T}}} \\
&= -\sum_{i=1}^{n} \boldsymbol{x}^i \boldsymbol{x}^{i^{\mathrm{T}}} \mathrm{Var}(T(y^i) | \eta(\boldsymbol{x}^i; \boldsymbol{\theta}))
\end{aligned}$$

Consider an arbitrary vector $\boldsymbol{z} \in \mathbb{R}^n$, the quadratic function

$$\boldsymbol{z}^{\mathrm{T}} \nabla^2_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \boldsymbol{z} = -\sum_{i=1}^{n} \boldsymbol{z}^{\mathrm{T}} \boldsymbol{x}^i \boldsymbol{x}^{i^{\mathrm{T}}} \boldsymbol{z} \mathrm{Var}(T(y^i) | \eta(\boldsymbol{x}^i; \boldsymbol{\theta})) = -\sum_{i=1}^{n} \left(\boldsymbol{z}^{\mathrm{T}} \boldsymbol{x}^i \right)^2 \mathrm{Var}(T(y^i) | \eta(\boldsymbol{x}^i; \boldsymbol{\theta}))$$

is obviously non-positive. Then we are sure that Equation (3) necessarily grants us with a global maximum of the likelihood of $\boldsymbol{\theta}$.

2 Generative Learning Algorithms

Generative Learning Algorithms are a different approach to similar questions that GLMs conquer. For a classification problem, inputting $(\boldsymbol{x}^i, y^i)$, we want to find a boundary between different values of y. GLM, as **discriminative learning algorithms**, models $p(\hat{y}|\hat{\boldsymbol{x}})$, while **generative learning algorithms** models $p(\hat{\boldsymbol{x}}|\hat{y})$ and then use Bayes' Rule to determine $p(\hat{y}|\hat{\boldsymbol{x}})$.

2.1 Gaussian Discriminant Analysis

In GDA, we model these probabilities: $p(y), p(\boldsymbol{x}|y = 0), p(\boldsymbol{x}|y = 1), ..., p(\boldsymbol{x}|y = k)$ where $\boldsymbol{x}$ takes on continuous values. Our assumption is $p(y) \sim \mathrm{PN}(1 : p_1, p_2, ..., p_k)$ and $p(\boldsymbol{x}|y = i) \sim \mathrm{Norm}(\boldsymbol{\mu}_i, \Sigma)$. Our parameters to optimize is $p_1, p_2, ..., p_k$; $\boldsymbol{\mu}_i$ for $i = 1, 2, ..., k$; and Σ.

$$\begin{aligned}
&\underset{p_1 \sim p_k, \boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k, \Sigma}{max} \log \mathcal{L}(p_1 \sim p_k, \boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k, \Sigma) \\
= &\underset{p_1 \sim p_k, \boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k, \Sigma}{max} \log \prod_{i=1}^{n} p(y^i; p_1 \sim p_k) p(\boldsymbol{x}^i | y^i; \boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k, \Sigma) \\
= &\underset{p_1 \sim p_k, \boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k, \Sigma}{max} \left(-\frac{nd}{2} \log(2\pi) - \frac{n}{2} \log \det\Sigma + \sum_{i=1}^{n} \left(\log p_{y^i} - \frac{1}{2} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i})^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i}) \right) \right) \\
\Leftrightarrow &\underset{p_1 \sim p_k}{max} \sum_{i=1}^{n} \log p_{y^i} \wedge \underset{\boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k, \Sigma}{max} \left(-n \log \det\Sigma - \sum_{i=1}^{n} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i})^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i}) \right)
\end{aligned}$$

Analyze these two respectively, for the former one:

$$\underset{p_1 \sim p_k}{max} \sum_{i=1}^{n} \log p_{y^i} \Leftrightarrow \mathrm{d}\sum_{i=1}^{n} \log p_{y^i} = 0 \Leftrightarrow \sum_{i=1}^{k} \frac{\mathbb{I}\{y^i = k\}}{p_k} \mathrm{d}p_k = 0$$

But we have $\sum_{i=1}^{k} p_k = 1$. Fixing any number of variables so that two variables are left, we simplify the problem into the optimization of logistic regression. Therefore by induction, we can prove that $\forall j. p_j \propto \sum_{i=1}^{n} \mathbb{I}\{y^i = j\}$, which results in the following equation:

$$\forall j \in \{1, 2, ..., k\}. p_j = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{y^i = j\}$$

For the latter maximum, we can't prove Σ's optimal value, so we only optimize $\boldsymbol{\mu}_i$.

$$\begin{aligned}
&\underset{\boldsymbol{\mu}_1 \sim \boldsymbol{\mu}_k}{max} \left(-n \log \det\Sigma - \sum_{i=1}^{n} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i})^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i}) \right) \\
\Leftrightarrow &\frac{\partial}{\partial \boldsymbol{\mu}_j} \sum_{i=1}^{n} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i})^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i}) = 0 \Leftrightarrow \frac{\partial}{\partial \boldsymbol{\mu}_j} \left(-2 {\boldsymbol{x}^i}^{\mathrm{T}} \Sigma^{-1} \boldsymbol{\mu}_{y^i} + \boldsymbol{\mu}_{y^i}^{\mathrm{T}} \Sigma^{-1} \boldsymbol{\mu}_{y^i} \right) = 0 \\
\Leftrightarrow &\sum_{i=1}^{n} \mathbb{I}\{y^i = j\} \left(\boldsymbol{x}^i - \boldsymbol{\mu}_j \right) = 0 \Leftrightarrow \boldsymbol{\mu}_j = \frac{\sum_{i=1}^{n} \mathbb{I}\{y^i = j\} \boldsymbol{x}^i}{\sum_{i=1}^{n} \mathbb{I}\{y^i = j\}}
\end{aligned}$$

Then our final conclusion is that:

$$\begin{cases}
p_j = \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} \mathbb{I}\{y^i = j\} \\
\boldsymbol{\mu}_j = \dfrac{\sum_{i=1}^{n} \mathbb{I}\{y^i = j\} \boldsymbol{x}^i}{\sum_{i=1}^{n} \mathbb{I}\{y^i = j\}} \\
\Sigma = \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} \left(\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i} \right) \left(\boldsymbol{x}^i - \boldsymbol{\mu}_{y^i} \right)^{\mathrm{T}}
\end{cases} \tag{4}$$

2.2 Comparison between GDA and Logistic Regression

Limiting the number of classes to 2, we find that GDA performs an identical task as logistic regression does, that is binary-classification. Actually, GDA generates a linear boundary like logistic regression, too.

Thm Gaussian Discriminant Analysis of 2 classes predicts the label of new inputs by $p(y = 1|\boldsymbol{x}; \phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{\theta}_0)}$, where $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_0$ are some function of $\phi, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma$.

Proof

$$p(y = 1|\boldsymbol{x}; p, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma) = \frac{p(\boldsymbol{x}|y = 1)\phi}{p(\boldsymbol{x}|y = 1)\phi + p(\boldsymbol{x}|y = 0)(1 - \phi)} = \frac{1}{1 + \dfrac{p(\boldsymbol{x}|y = 0)(1 - \phi)}{p(\boldsymbol{x}|y = 1)\phi}}$$

$$\begin{aligned}2\log\frac{p(\boldsymbol{x}|y=0)}{p(\boldsymbol{x}|y=1)} =&(\boldsymbol{x}-\boldsymbol{\mu}_0)^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_0)-(\boldsymbol{x}-\boldsymbol{\mu}_1)^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)\\ =&2\left(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0\right)^{\mathrm{T}}\Sigma^{-1}\boldsymbol{x}+\boldsymbol{\mu}_0^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_0-\boldsymbol{\mu}_1^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_1\end{aligned}$$

Then we know that $\boldsymbol{\theta}=\Sigma^{-1}(\boldsymbol{\mu}_0-\boldsymbol{\mu}_1)$, and $\boldsymbol{\theta}_0=\log\frac{1-\phi}{\phi}+\frac{1}{2}\left(\boldsymbol{\mu}_0^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_0-\boldsymbol{\mu}_1^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_1\right)$.

Q.E.D.

This theorem indicates that the assumption of logistic regression can be derived from that of GDA, so **GDA utilizes stronger assumption than logistic regression**. Such a fact further implies that if $p(\boldsymbol{x}|y)$ is Gaussian, even with small training set, GDA performs well, while logistic regression is compatible to much more kinds of data distribution. In the real world, seldom does ideal Gaussian distribution appear, so logistic regression is used more.

2.3 Naive Bayes

Rather than continuous distribution $p(\boldsymbol{x}|y)\sim\mathrm{Norm}(\boldsymbol{\mu},\Sigma)$, **Naive Bayes** is applied to situations where $\boldsymbol{x}$ has discrete values. **Naive Bayes Assumption** asserts that all features are conditionally independent given y, i.e.

$$p(\boldsymbol{x}|y)=\prod_{i=1}^{d}p(x_{[i]}|y,x_{[1]},x_{[2]},...,x_{[i-1]})=\prod_{i=1}^{d}p(x_{[i]}|y)$$

where for any i, $p(x_{[i]}|y)$ is further assumed to have a multinomial distribution. Then our parameters to optimize are

$\boldsymbol{\phi}_{i|y}=[\phi_{i|y[0]},\phi_{i|y[1]},...,\phi_{i|y[k]}]^{\mathrm{T}}=[p(x_{[i]}=1|y),p(x_{[i]}=2|y),...,p(x_{[i]}=k|y)]^{\mathrm{T}}$,

$\boldsymbol{\phi}_y=[\phi_{y[0]},\phi_{y[1]},...,\phi_{y[l]}]^{\mathrm{T}}=[p(y=0),p(y=1),...,p(y=l)]^{T}$,

where $x_{[i]}$ is assumed to have k kinds of values, while l kinds for y. Now we can again use MLE (that is exactly the same as GDA).

$$\mathop{max}_{\boldsymbol{\phi}_{i|y},\boldsymbol{\phi}_y}\mathcal{L}(\boldsymbol{\phi}_{i|y},\boldsymbol{\phi}_y)=\mathop{max}_{\boldsymbol{\phi}_{i|y},\boldsymbol{\phi}_y}\prod_{i=1}^{n}p(y^i;\boldsymbol{\phi}_y)p(\boldsymbol{x}^i|y^i;\boldsymbol{\phi}_{i|y})=\mathop{max}_{\boldsymbol{\phi}_y}\prod_{i=1}^{n}p(y^i;\boldsymbol{\phi}_y)\times\mathop{max}_{\boldsymbol{\phi}_{i|y}}\prod_{i=1}^{n}p(\boldsymbol{x}^i|y^i;\boldsymbol{\phi}_{i|y})$$

With the same analysis as GDA, $\mathop{max}_{\boldsymbol{\phi}_y}\prod_{i=1}^{n}p(y^i;\boldsymbol{\phi}_y)\Leftrightarrow\phi_{y[i]}=\frac{1}{n}\sum_{j=1}^{n}\mathbb{I}\{y^j=i\}$.

$$\begin{aligned}\mathop{max}_{\boldsymbol{\phi}_{i|y}}\prod_{i=1}^{n}p(\boldsymbol{x}^i|y^i;\boldsymbol{\phi}_{i|y})\Leftrightarrow&\mathop{max}_{\boldsymbol{\phi}_{i|y}}\sum_{j=1}^{n}\sum_{i=1}^{d}\log p(\boldsymbol{x}_i^j|y^j;\boldsymbol{\phi}_{i|y})\Leftrightarrow\mathop{max}_{\boldsymbol{\phi}_{i|y}}\sum_{j=1}^{n}\sum_{i=1}^{d}\log\phi_{i|y=y^j[x_{[i]}^j]}\\ \Leftrightarrow&\mathop{max}_{\boldsymbol{\phi}_{i|y}}\sum_{p=1}^{k}\sum_{q=1}^{l}\sum_{i=1}^{d}\left(\sum_{j=1}^{n}\mathbb{I}\{x_{[i]}^j=p\wedge y^j=q\}\right)\log\phi_{i|q[p]}\end{aligned}$$

Then because $\sum_{p=1}^{k}\phi_{i|q[p]}=1$, maximization results in: for any i,p,q,

$$\phi_{i|q[p]}=\frac{\sum_{j=1}^{n}\mathbb{I}\{x_{[i]}^j=p\wedge y^j=q\}}{\sum_{p=1}^{k}\sum_{j=1}^{n}\mathbb{I}\{x_{[i]}^j=p\wedge y^j=q\}}=\frac{\sum_{j=1}^{n}\mathbb{I}\{x_{[i]}^j=p\wedge y^j=q\}}{\sum_{j=1}^{n}\mathbb{I}\{y^j=q\}}$$

After training, when given some input $\boldsymbol{x}$, we find such a $q\in\{1,2,...,l\}$ that maximizes $p(\boldsymbol{x}|y=q)p(y=q)=\phi_{y[q]}\prod_{i=1}^{d}\phi_{i|q[x_{[i]}]}$.

2.3.1 Laplace Smoothing

Problems arise from the former equation. Suppose in class $\{y = q\}$, there isn't any training example $\boldsymbol{x}^j$ such that $x^j_{[i]} = p$, so $\phi_{i|q[p]} = 0$, then if the new input $\boldsymbol{x}$ has $x_{[i]} = p$, the algorithm would say: there is no probability that $\boldsymbol{x}$ belongs to class q, which is obviously non-sense in terms of statistics. Moreover, if all classes has such a property, our algorithm faces 0/0 difficulty.

The solution lies in Laplace Smoothing. Suppose a sampling process, $(0, k_2, k_3, 0, .., k_{n-2}, 0, k_n)$ results appear in corresponding classes, we model the result as $(1, k_2 + 1, k_3 + 1, 1, ..., k_{n-2} + 1, 1, k_n + 1)$ instead, eliminating all zeros.

2.3.2 Email Spam Classifier

Two ways of constructing a classifier.

• Let $y \in \{0, 1\}$ means whether an email is spam, $\boldsymbol{x}$ be whether each word appears in an email, e.g. if we have a 10k-word dictionary, $\boldsymbol{x} \in \{0, 1\}^{10000}$, where 0 means a word doesn't appear, 1 meaning appearance (ignorant of how many times it appears).

• Let $y \in \{0, 1\}$ be the same thing as the first method assumes, $\boldsymbol{x} \in \{0, 1, ..., |V|\}^d$ if we have a $|V|$-sized dictionary and the email has d words.

3 Support Vector Machine

Support vector machine, as a "linear" classifier algorithm, does similar work to what other GLMs or GLAs do, but from a different mathematical perspective. It turns out that GLMs and GLAs both consider the problems from a probabilistic approach, treating all imputs as random variables with a certain probability distribution, and therefore learns with maximum likelihood estimation. But SVM takes the training set as "ground truth", instead of probability-modelled data.

3.1 Optimal Margin Classifier (Naive SVM)

Given linearly-separable training set $(\boldsymbol{x}^i, y^i), i \in \{1, 2, ..., n\}$ with $y^i \in \{-1, +1\}$, we want to train a linear classifier $h(\boldsymbol{x}) = \text{sgn}\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + b\right)$. The problem is modelled as follows.

We first describe our certainty that an example belongs to some class.

Def Functional Margin (of a single training example): the functional margin of $(\boldsymbol{\omega}, b)$ with respect to training example i is $\hat{y}^i = y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + b\right)$.

Def Functional Margin (of a training set): the functional margin of $(\boldsymbol{\omega}, b)$ with respect to the training set is $\hat{y} = \underset{i=1,2,...,n}{min} \hat{y}^i$.

Next we make sure the same support hyperplane corresponds to the same "margin".

Def Geometric Margin (of a single training example): the geometric margin of $(\boldsymbol{\omega}, b)$ with respect to training example i is $\gamma^i = \dfrac{\hat{y}^i}{\|\boldsymbol{\omega}\|} = y^i\left(\left(\dfrac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\right)^{\mathrm{T}} x^i + \left(\dfrac{b}{\|\boldsymbol{\omega}\|}\right)\right)$.

Def Geometric Margin (of a training set): the geometric margin of $(\boldsymbol{\omega}, b)$ with respect to the training set is $\gamma = \underset{i=1,2,..,n}{min} \gamma^i$.

Then our optimization problem, which is maximizing the margin between two classes, can be described as $\underset{\gamma,\boldsymbol{\omega},b}{max}\, \gamma$ s.t. $\forall i \in \{1, 2, ..., n\}$, $\dfrac{y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + b\right)}{\|\boldsymbol{\omega}\|} \geq \gamma$.

$$\begin{aligned}\mathop{max}_{\gamma,\boldsymbol{\omega},b} \gamma \text{ s.t. } \frac{y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}+b\right)}{\|\boldsymbol{\omega}\|} \geq \gamma &\Leftrightarrow \mathop{max}_{\gamma,\boldsymbol{\omega},b} \gamma \text{ s.t. } y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}+b\right) \geq \gamma \text{ and } \|\boldsymbol{\omega}\| = 1 \\ &\Leftrightarrow \mathop{max}_{\gamma,\boldsymbol{\omega},b} \gamma \text{ s.t. } y^i\left(\left(\frac{\boldsymbol{\omega}}{\gamma}\right)^{\mathrm{T}}\boldsymbol{x}+\left(\frac{b}{\gamma}\right)\right) \geq 1 \text{ and } \|\boldsymbol{\omega}\| = 1 \\ &\Leftrightarrow \mathop{min}_{\boldsymbol{\omega},b} \frac{1}{2}\|\boldsymbol{\omega}\|^2 \text{ s.t. } y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}+b\right) \geq 1 \end{aligned} \tag{5}$$

which entitles us to a simple description of our optimization problem.

3.1.1 Lagrange Duality

• ***Basic Lagrange Multipliers***

The equation above is a constrained optimization problem. We can transform it into an unconstrained optimization problem.

Def Lagrangian: Given a problem $\min_{\boldsymbol{\omega}} f(\boldsymbol{\omega})$ s.t. $\boldsymbol{h}(\boldsymbol{\omega}) = \mathbf{0}$ where $\boldsymbol{h}(\cdot) = [h_1, h_2, ..., h_l]^{\mathrm{T}}$, the Lagrangian is defined by

$$\mathcal{L}(\boldsymbol{\omega},\boldsymbol{\beta}) = f(\boldsymbol{\omega}) + \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{\omega})$$

then the minimum value can be solved through $\nabla_{\boldsymbol{\omega}}\mathcal{L}(\boldsymbol{\omega},\boldsymbol{\beta}) = \mathbf{0}$ and $\nabla_{\boldsymbol{\beta}}\mathcal{L}(\boldsymbol{\omega},\boldsymbol{\beta}) = \mathbf{0}$.

Def Lagrange Multipliers: For the $\boldsymbol{\beta}$ defined above, elements of it ($\beta_i, i \in \{1, 2, ..., n\}$) are called Lagrange Multipliers.

Def Generalized Lagrangian: Given a **primal** optimization problem $\min_{\boldsymbol{\omega}} f(\boldsymbol{\omega})$ s.t. $\boldsymbol{g}(\boldsymbol{\omega}) \leq \mathbf{0}$ and $\boldsymbol{h}(\boldsymbol{\omega}) = \mathbf{0}$ where $\boldsymbol{g}(\cdot) = [g_1, g_2, ..., g_k]^{\mathrm{T}}$ and $\boldsymbol{h}(\cdot) = [h_1, h_2, ..., h_l]^{\mathrm{T}}$, the Generalized Lagrangian is defined by

$$\mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\boldsymbol{\omega}) + \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{g}(\boldsymbol{\omega}) + \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{h}(\boldsymbol{\omega})$$

but to solve the minimum there still leaves some work.

• ***Lagrange Duality***

Thm Given a generalized Lagrangian $\mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta})$, let $\theta_{\mathcal{P}}(\boldsymbol{\omega}) = \mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta})$, minimizing $f(\boldsymbol{\omega})$ is equivalent to minimizing $\theta_{\mathcal{P}}(\boldsymbol{\omega})$.

Proof Given some value of $\boldsymbol{\omega}$, if $\boldsymbol{\omega}$ satisfies the condition that $\boldsymbol{g}(\boldsymbol{\omega}) \leq \mathbf{0}$ and $\boldsymbol{h}(\boldsymbol{\omega}) = \mathbf{0}$, then $\theta_{\mathcal{P}}(\boldsymbol{\omega}) = f(\boldsymbol{\omega})$, otherwise $\theta_{\mathcal{P}}(\boldsymbol{\omega}) = +\infty$. It follows naturally that the proposition is true.
Q.E.D.

Thm $d^* = \mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathop{min}_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta}) \leq \mathop{min}_{\boldsymbol{\omega}} \mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta}) = p^*$.

Proof Let $L_1(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathop{min}_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta})$ and $L_2(\boldsymbol{\omega}) = \mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta})$.

Suppose $L_1(\boldsymbol{\alpha}_0,\boldsymbol{\beta}_0) > L_2(\boldsymbol{\omega}_0)$ where $\boldsymbol{\alpha}_0 \geq \mathbf{0}$, then

$$\mathop{min}_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha}_0,\boldsymbol{\beta}_0) \leq \mathcal{L}(\boldsymbol{\omega}_0,\boldsymbol{\alpha}_0,\boldsymbol{\beta}_0) \leq \mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathcal{L}(\boldsymbol{\omega}_0,\boldsymbol{\alpha},\boldsymbol{\beta})$$

which leads to contradiction. Therefore, $\mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathop{min}_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta}) \leq \mathop{min}_{\boldsymbol{\omega}} \mathop{max}_{\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\alpha}\geq\mathbf{0}} \mathcal{L}(\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\beta})$.
Q.E.D.

Thm If $f(\boldsymbol{\omega})$ and $g_i(\boldsymbol{\omega})$, $\forall i$ are convex, $\boldsymbol{h}(\boldsymbol{\omega})$ are affine, and $\exists\boldsymbol{\omega}$ s.t. $\boldsymbol{g}(\boldsymbol{\omega}) < \mathbf{0}$, then $\exists\boldsymbol{\omega}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ s.t. $d^* = p^* = \mathcal{L}(\boldsymbol{\omega}^*,\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)$. Moreover, $\boldsymbol{\omega}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ satisfies KKT condition. **(No Proof)**

Thm **Karush-Kuhn-Tucker's Theorem**: If $\boldsymbol{\omega}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ is a local minimum, then

$$\begin{cases} \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \mathbf{0} \\ \nabla_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\omega}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \mathbf{0} \\ \boldsymbol{\alpha}^{*\mathrm{T}} \boldsymbol{g}(\boldsymbol{\omega}^*) = \mathbf{0} \\ \boldsymbol{g}(\boldsymbol{\omega}^*) \leq \mathbf{0} \\ \boldsymbol{\alpha} \geq \mathbf{0} \end{cases}$$

(No Proof)

- ***Support Vector Machine***

Then for our problem (Equation (5)), we have our generalized Lagrangian as

$$\mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{n} \alpha_i [y^i(\boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{x}^i + b) - 1]$$

by setting the derivative to zero, we have

$$\boldsymbol{\omega} = \sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x}^i \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0$$

Substituting $\boldsymbol{\omega}$ with $\sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x}^i$, we obtain

$$\mathop{min}_{\boldsymbol{\omega}, b} \mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{\alpha}) = W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \langle \boldsymbol{x}^i, \boldsymbol{x}^j \rangle$$

Thus we have

$$\begin{aligned} & \mathop{min}_{\boldsymbol{\omega}, b} \frac{1}{2}\|\boldsymbol{\omega}\|^2 \text{ s.t. } 1 - y^i(\boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{x}^i + b) \leq 0 \\ \Leftrightarrow & \mathop{min}_{\boldsymbol{\omega}, b} \mathop{max}_{\boldsymbol{\alpha} | \boldsymbol{\alpha} \geq \mathbf{0}} \left(\frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{n} \alpha_i [y^i(\boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{x}^i + b) - 1] \right) \text{ (proved by theorem)} \\ \Leftrightarrow & \mathop{max}_{\boldsymbol{\alpha} | \boldsymbol{\alpha} \geq \mathbf{0}} \mathop{min}_{\boldsymbol{\omega}, b} \left(\frac{1}{2}\|\boldsymbol{\omega}\|^2 - \sum_{i=1}^{n} \alpha_i [y^i(\boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{x}^i + b) - 1] \right) \text{ (condition for } d^* = p^* \text{ satisfied)} \\ \Leftrightarrow & \mathop{max}_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \langle \boldsymbol{x}^i, \boldsymbol{x}^j \rangle \right) \text{ s.t. } \boldsymbol{\alpha} \geq \mathbf{0} \text{ and } \sum_{i=1}^{n} \alpha_i y^i = 0 \end{aligned}$$

The equation above can be solved through **SMO algorithm**. Suppose the optimal $\boldsymbol{\alpha}^*$ has been found, that is,$\boldsymbol{\omega}^* = \sum_{i=1}^{n} \alpha_i^* y^i \boldsymbol{x}^i$. According to the KKT Theorem, $\alpha_i^* [y^i(\boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i + b) - 1] = 0$, which means only the α_i^*s corresponding to support vectors can be non-zero.

Consider the is s.t. $\alpha_i \neq 0$. We have $y^i(\boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i + b) = 1$ and $\sum_{i|\alpha_i \neq 0} \alpha_i y^i = 0$, then $\exists y^{i_1} = 1$ s.t. $\boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^{i_1} + b = 1$ and $\exists y^{i_2} = -1$ s.t. $\boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^{i_2} + b = -1$.

Because $\forall i : y^i = 1, \boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i \geq 1$, we have $\boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^{i_1} = \mathop{min}_{i|y^i=1} \boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i$, which leads to

$$b = 1 - \mathop{min}_{i|y^i=1} \boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i = -1 - \mathop{max}_{i|y^i=-1} \boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i = -\frac{1}{2} \left(\mathop{min}_{i|y^i=1} \boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i + \mathop{max}_{i|y^i=-1} \boldsymbol{\omega}^{*\mathrm{T}} \boldsymbol{x}^i \right)$$

as the final part of solution to SVM optimization problem.

3.1.2 SMO Algorithm

The **Sequential Minimal Optimization** algorithm, iteratively does coordinate ascent, which is, given unconstrained problem $max_{\boldsymbol{\alpha}}\, W(\boldsymbol{\alpha})$ where $\boldsymbol{\alpha} \in \mathbb{R}^m$, it always fix $m-1$ features and finds the maximum value by changing the left one.

For SVM, we have the constraint $\sum_{i=1}^{n} \alpha_i y^i = 0$, therefore we should change at least two features together. But since the target function is quadratic, and the constraint restricts variable (α_i, α_j) to a line $k\alpha_i + l\alpha_j = b$ in the $\mathbb{R}^2$ space. Maximizing a quadratic function on a line within $\mathbb{R}^2$ space is pretty easy and efficient, which wraps up the support vector machine algorithm.

3.2 Kernel Methods

Support vector machines is also suitable for non-linear boundaries. Our solution is mapping the data to higher-dimensional spaces, e.g. $\mathbb{R}^2 \Rightarrow \mathbb{R}^3$ to spot an ellipsoid boundary on a plane. The function to map $\boldsymbol{x}$ into higher dimensional spaces is $\phi : \mathbb{R}^d \to \mathbb{R}^n$ (here n can be infinite) called a **feature map**. The problem is if a very very high n is required, taking $O(n^2)$ time to compute the inner product is a very bad idea. The solution is kernel methods.

Def Kernel: the kernel corresponding to a feature map ϕ is a function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ satisfying $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle \phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2) \rangle$.

[e.g.] If $\phi(\boldsymbol{x}) = [x_1x_1, x_1x_2, x_2x_1, x_2x_2]^{\mathrm{T}}$ for $\boldsymbol{x} = [x_1, x_2]^{\mathrm{T}}$, then $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle)^2$, or more generally, $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + c)^k$ corresponds to a feature map that consists of all x_ix_j, $\sqrt{2c}x_i$ and a constant c.

Thm Mercer's Theorem: Given $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, then K is a valid kernel iff. $\forall \{\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^n\}$, the kernel matrix $[K]_{ij} = K(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is positive semi-definite.

Proof

($\Rightarrow$ only) for any vector $\boldsymbol{z} \in \mathbb{R}^n$,

$$\boldsymbol{z}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{z} = \sum_{i=1}^{n}\sum_{j=1}^{n} K_{ij}z_iz_j = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k} \phi_k(\boldsymbol{x}^i)(\boldsymbol{x}^j)z_iz_j = \sum_{k}\left(\sum_{i=1}^{n} z_i\phi_k(\boldsymbol{x}^i)\right)^2 \geq 0$$

Q.E.D.

With kernels we can calculate the inner product between two feature vectors within $O(d)$ time, which is very efficient. For SVM, we have expressed the problem by inner products so far, therefore kernel methods can be efficiently implemented to help SVM.

3.3 Soft Margin

For training sets that are not linearly separable (even in high dimensional spaces) or to eliminate the influences of outliers, SVM optimization can be transformed into

$$\begin{aligned}
&\underset{\boldsymbol{\omega},b,\boldsymbol{\xi}}{min}\left(\frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n}\xi_i\right) \text{ s.t. } y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}^i + b\right) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \\
\Leftrightarrow\ &\underset{\boldsymbol{\omega},b,\boldsymbol{\xi}}{min}\ \underset{\boldsymbol{\alpha},\boldsymbol{r}|\boldsymbol{\alpha}\geq 0,\boldsymbol{r}\geq 0}{max}\left(\frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}^i + b\right) - 1 + \xi_i\right) - \boldsymbol{r}^{\mathrm{T}}\boldsymbol{\xi}\right) \\
\Leftrightarrow\ &\underset{\boldsymbol{\alpha},\boldsymbol{r}|\boldsymbol{\alpha}\geq 0,\boldsymbol{r}\geq 0}{max}\ \underset{\boldsymbol{\omega},b,\boldsymbol{\xi}}{min}\left(\frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}^i + b\right) - 1 + \xi_i\right) - \boldsymbol{r}^{\mathrm{T}}\boldsymbol{\xi}\right)
\end{aligned}$$

Taking the derivative and setting it to zero, we have

$$\boldsymbol{\omega}=\sum_{i=1}^{n}\alpha_i y^i \boldsymbol{x}^i,\ \sum_{i=1}^{n}\alpha_i y^i=0 \text{ and } C\mathbf{1}^{\mathrm{T}}-\boldsymbol{\alpha}-\boldsymbol{r}=\mathbf{0}$$

Substituting these parameters,

$$\underset{\boldsymbol{\alpha},\boldsymbol{r}|\boldsymbol{\alpha}\geq 0,\boldsymbol{r}\geq 0}{max}\ \underset{\boldsymbol{\omega},b,\boldsymbol{\xi}}{min}\left(\frac{1}{2}\|\boldsymbol{\omega}\|^2+C\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\alpha_i\left(y^i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}^i+b\right)-1+\xi_i\right)-\boldsymbol{r}^{\mathrm{T}}\boldsymbol{\xi}\right)$$

$$\Leftrightarrow \underset{\boldsymbol{\alpha}}{max}\left(\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y^i y^j\left\langle\boldsymbol{x}^i,\boldsymbol{x}^j\right\rangle\right)\ \text{s.t. } \mathbf{0}\leq\boldsymbol{\alpha}\leq C\mathbf{1} \text{ and } \sum_{i=1}^{n}\alpha_i y^i=0$$

4 Generalization

Now we try to generalize our learning theory.

Assumption: Training set $S=\{(\boldsymbol{x}^i,y^i)|i=1,2,...,m\}\subset\mathbb{R}^n\times\mathbb{Y}$ where $\mathbb{Y}$ is the codomain of the problem (e.g. for binary classification problem it is $\{0,1\}$) as random variables all drawn independently from a distribution $\mathcal{D}$, and elements in test set are also drawn independently from $\mathcal{D}$. Moreover, there exists a ground truth h_0 as the best possible hypothesis.

Training Process: We feed the m inputs into an estimator (the learning algorithm, or probabilistically, a deterministic function), which outputs a hypothesis $\hat{h}$. Specifically, the result of the algorithm $\mathrm{ALG}\in((\mathbb{R}^n\times\mathbb{Y})^m\to(\mathbb{R}^n\to\mathbb{Y}))$ is denoted by $\hat{h}_S=\mathrm{ALG}(S)\in(\mathbb{R}^n\to\mathbb{Y})$.

In learning algorithms, we limit ourselves to a certain class of hypotheses $\mathcal{H}\subset\mathbb{R}^n\to\mathbb{Y}$, e.g. the class of all logistic regression hypotheses. Then the best hypothesis we could possibly get is defined as h^* (which may differ from h_0, and the definition of h^* is ambiguous, what we often assume is that if the distribution $\mathcal{D}$ can be seen as a process of adding a noise to some specific function, then that function would be h^*). We assume that for any $\boldsymbol{x}\in\mathbb{R}^n$, $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[y-h^*(\boldsymbol{x})\right]=0$.

Def Sampling Distribution: the distribution that $\hat{h}$ follows. Since each S corresponds to one $\hat{h}$, sampling distribution is a function of $\mathcal{D}$ and m. (Since here $\hat{h}$ is a random variable, it makes sense to implement **cross validation**.)

Def Mean Squared Error: $\mathrm{MSE}(\boldsymbol{x})=\mathbb{E}_{S,(\boldsymbol{x},y)\sim\mathcal{D}}\left[(y-\hat{h}_S(\boldsymbol{x}))^2\right]$.

Therefore,

$$\mathrm{MSE}(\boldsymbol{x})=\mathbb{E}_{S,(\boldsymbol{x},y)\sim\mathcal{D}}\left[\left(y-h^*(\boldsymbol{x})+h^*(\boldsymbol{x})-\hat{h}_S(\boldsymbol{x})\right)^2\right]$$

where h^* is not affected by any random variable; $y-h^*(\boldsymbol{x})$ is only dependent on y; $\hat{h}_S$ only dependent on S. Therefore, $y-h^*(\boldsymbol{x})$ and $h^*(\boldsymbol{x})-\hat{h}_S(\boldsymbol{x})$ are independent.

$$\begin{aligned}\mathrm{MSE}(\boldsymbol{x})=&\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(y-h^*(\boldsymbol{x})^2\right]+\mathbb{E}_S\left[\left(h^*(\boldsymbol{x})-\hat{h}_S(\boldsymbol{x})\right)^2\right]\\=&\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(y-h^*(\boldsymbol{x})^2\right]+\mathbb{E}_S\left[\left(h^*(\boldsymbol{x})-\mathbb{E}_S\left[\hat{h}_S(\boldsymbol{x})\right]+\mathbb{E}_S\left[\hat{h}_S(\boldsymbol{x})\right]-\hat{h}_S(\boldsymbol{x})\right)^2\right]\\=&\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(y-h^*(\boldsymbol{x})^2\right]+\left(h^*(\boldsymbol{x})-\mathbb{E}_S\left[\hat{h}_S(\boldsymbol{x})\right]\right)^2+\mathrm{Var}_S\left(\hat{h}_S(\boldsymbol{x})\right)\end{aligned}$$

Def Bias and Variance: $\mathrm{Bias}(\boldsymbol{x})=h^*(\boldsymbol{x})-\mathbb{E}_S\left[\hat{h}_S(\boldsymbol{x})\right]$, $\mathrm{Variance}(\boldsymbol{x})=\mathrm{Var}_S\left(\hat{h}_S(\boldsymbol{x})\right)$. (The first term is uncontrollable.)

Def Statistical Efficiency: If $\forall \boldsymbol{x}$, $\mathrm{Var}_S\left(\hat{h}_S(\boldsymbol{x})\right) \to 0$ as $m \to \infty$, then we say our estimator is statiscally efficient, the rate it converges to 0 is called statistical efficiency.

Def Consistency: If $\hat{h}_S \to h^*$ as $m \to \infty$ **a.s.**, then we say our estimator is consistent.

Def Unbiased: If $\forall \boldsymbol{x}$, $\mathbb{E}_S\left[\hat{h}_S(\boldsymbol{x})\right] = h^*(\boldsymbol{x})$ whatever m is, then our estimator is unbiased.

4.1 Generalization For Classification Problems

We can further decompose the error into bias and variance terms through further definitions.

Def Risk / Generalization Error: the generalization error is defined for classification problem as $\epsilon(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\mathbb{I}\{h(\boldsymbol{x}) \neq y\}\right]$.

Def Empirical Risk: the empirical risk (or training error) is defined for each S with size m as $\hat{\epsilon}_S(h) = \frac{1}{m}\sum_{i=1}^m \mathbb{I}\{h(\boldsymbol{x}^i) \neq y^i\}$.

Def Bayes Error: Bayes error is the irreducible error or the error rate of h_0 defined as $\epsilon(h_0)$. **[e.g.]** If $\boldsymbol{x}^i = \boldsymbol{x}^j$ while $y^i \neq y^j$, then there isn't any model that can tell these two apart.

Def Approximation Error: approximation error is $\epsilon(h^*) - \epsilon(h_0)$ as the error introduced when we limit our hypotheses to $\mathcal{H}$.

Def Estimation Error: estimation error is $\epsilon(\hat{h}) - \epsilon(h^*)$ as the error introduced by the randomness of the selection of S.

Then we can write the actual error as

$$\begin{aligned}\epsilon(h^*) =& \mathrm{EstimationError} + \mathrm{ApproximationError} + \mathrm{BayesError}\\ =&(\mathrm{EstimationVar} + \mathrm{EstimationBias}) + \mathrm{ApproximationError} + \mathrm{BayesError}\end{aligned}$$

4.2 Sample Complexity of Empirical Risk Minimizers

• ***Problem Description***

Given classification problem, let the training process of ERM be selecting $\hat{h} = \arg\min_{h\in\mathcal{H}} \hat{\epsilon}(h)$, while the best hypothesis $h^* = \arg\min_{h\in\mathcal{H}} \epsilon(h)$ is invisible for us.

We want a probabilistic evaluation of the performance of our model (or the difference between the error of our hypothesis and the error of the best hypothesis), that is an evaluation of $\epsilon(\hat{h}) - \epsilon(h^*)$.

• ***Sample Complexity with finite $\mathcal{H}$***

We first claim, utilizing the Hoeffding's inequality whose proof is given later, that, for any $h \in \mathcal{H}$, $\gamma \geq 0$, we have $P(|\hat{\epsilon}(h) - \epsilon(h)| \geq \gamma) \leq e^{-2\gamma^2 n}$. Therefore, suppose $|\mathcal{H}| = k$ is a finite set,

$$P(\forall h \in \mathcal{H}, |\hat{\epsilon}(h) - \epsilon(h)| < \gamma) \geq 1 - 2ke^{-2\gamma^2 n}$$

Taking $\forall h \in \mathcal{H}, |\hat{\epsilon}(h) - \epsilon(h)| < \gamma$ as a condition, we have

$$\epsilon(\hat{h}) < \hat{\epsilon}(\hat{h}) + \gamma \leq \hat{\epsilon}(h^*) + \gamma < \epsilon(h^*) + 2\gamma$$

i.e. $P(\epsilon(\hat{h}) - \epsilon(h^*) < 2\gamma) \geq 1 - 2ke^{-2\gamma^2 n}$, which leads to our conclusion as follows:

Proposition: Given $|\mathcal{H}| = k$ as a set of hypotheses and n i.i.d. training examples drawn from $\mathcal{D}$, and another test example drawn from $\mathcal{D}$, if the training results in the hypothsis $\hat{h} = \arg\min_{h\in\mathcal{H}} \hat{\epsilon}(h)$, while the best hypothsis $h^* = \arg\min_{h\in\mathcal{H}} \epsilon(h)$, we have, for any $\delta > 0$,

$$P\left(\epsilon(\hat{h}) - \epsilon(h^*) < 2\sqrt{\frac{1}{2n}\log\frac{2k}{\delta}}\right) \geq 1 - \delta$$

• ***Sample Complexity with infinite $\mathcal{H}$***

Def Shattering: Given set $S = \{\boldsymbol{x}^i | i = 1, 2, ..., D\}$, we say $\mathcal{H}$ shatters S if for any set of labelling $L = \{y^i | i = 1, 2, ..., D\}$, there exists $h \in \mathcal{H}$ s.t. $\forall i$, $h(\boldsymbol{x}^i) = y^i$.

Def Vapnik-Chervonenkis Dimension: Given a hypothesis set $\mathcal{H}$, the Vapnik-Chervonenkis dimension VC($\mathcal{H}$) is the size of the largest set that is shattered by $\mathcal{H}$.

[e.g.] In a 2-dimensional space, given a binary-classification problem, if $\mathcal{H}$ is all linear boundaries (or lines), then VC($\mathcal{H}$) = 3 because for 3 points not on the same line, we can always find a linear boundary that separates positive and negative examples.

Thm Vapnik's Theorem: Given $\mathcal{H}$, $D = \text{VC}(\mathcal{H})$, then with probability at least $1 - \delta$, we have $\forall h \in \mathcal{H}$,

$$|\epsilon(h) - \hat{\epsilon}(h)| \leq O\left(\sqrt{\frac{D}{n}\log\frac{n}{D} + \frac{1}{n}\log\frac{1}{\delta}}\right)$$

Then it natrually follows that, with probability at least $1 - \delta$,

$$\epsilon(\hat{h}) \leq \epsilon(h^*) + O\left(\sqrt{\frac{D}{n}\log\frac{n}{D} + \frac{1}{n}\log\frac{1}{\delta}}\right)$$

• ***Hoeffding's Inequality and Proof***

Thm Hoeffding's Inequality: For independent random variables $\{X_i | i = 1, 2, ..., n\}$, suppose all X_is are bounded, i.e. $\exists a_i, b_i$ s.t. $P(X_i \in [a_i, b_i]) = 1$. Denote $\bar{X} = \frac{1}{n}\sum_{i=1}^n X_i$ as the average, then

$$P(\bar{X} - \mathbb{E}[\bar{X}] \geq t) \leq \exp\frac{-2t^2n^2}{\sum_{i=1}^n (b_i - a_i)^2}$$

Proof

Lemma Markov's Inequality: For non-negative random variable X

$$\forall \epsilon > 0, P(X \geq \epsilon) \leq \frac{1}{\epsilon}\mathbb{E}[X]$$

(Proof: $\mathbb{E}[X] = \displaystyle\int_0^{+\infty} x f_X(x)\mathrm{d}x \geq \int_\epsilon^{+\infty} \epsilon f_X(x)\mathrm{d}x = \epsilon P(X \geq \epsilon)$.)

Applying Markov's Inequality, we have

$$\begin{aligned} P(\bar{X} - \mathbb{E}[\bar{X}] \geq t) =& P(\exp\left(s\left(\bar{X} - \mathbb{E}[\bar{X}]\right)\right) \geq e^{st}) \leq e^{-st}\mathbb{E}\left[\exp\left(s\left(\bar{X} - \mathbb{E}[\bar{X}]\right)\right)\right] \\ =& e^{-st}\mathbb{E}\left[\exp\frac{s\sum_{i=1}^n (X_i - \mathbb{E}[X_i])}{n}\right] = e^{-st}\prod_{i=1}^n \mathbb{E}\left[\exp\frac{s\left(X_i - \mathbb{E}[X_i]\right)}{n}\right] \end{aligned}$$

Lemma: Given a bounded random variable X, let $P(X \in [a, b]) = 1$ and $\mathbb{E}[X] = 0$, then $\mathbb{E}\left[e^X\right] \leq \exp\dfrac{(b-a)^2}{8}$. (**No Proof**)

Therefore, $\mathbb{E}\left[\exp\dfrac{s\left(X_i - \mathbb{E}[X_i]\right)}{n}\right] \leq \exp\dfrac{s^2(b_i - a_i)^2}{8n^2}$, and moreover,

$$P(\bar{X} - \mathbb{E}[\bar{X}] \geq t) \leq \exp\left(-st + \frac{s^2}{8n^2}\sum_{i=1}^n (b_i - a_i)^2\right)$$

holds for any $s \in \mathbb{R}$. Taking the s that minimizes the equation, we have

$$P(\bar{X} - \mathbb{E}[\bar{X}] \geq t) \leq \exp \frac{-2t^2n^2}{\sum_{i=1}^{n}(b_i - a_i)^2}$$

Q.E.D.

Because, given a certain hypothesis h, $\epsilon(h) = \mathbb{E}\left[\mathbb{I}\{h(\boldsymbol{x}) \neq y\}\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\{h(\boldsymbol{x}'^i) \neq y'^i\}\right] = \mathbb{E}[\hat{\epsilon}(h)]$, applying Hoeffding's inequality leads to

$$P(|\hat{\epsilon}(h) - \epsilon(h)| \geq \gamma) \leq e^{-2\gamma^2 n}$$

which wraps up the derivation process.

4.3 Regularization

We fight high bias through setting a bigger $\mathcal{H}$, while high variance is solved through increasing m or adding regularization term to our loss function, e.g. $\lambda\|\boldsymbol{\theta}\|^2$ if the parameters to optimize is $\boldsymbol{\theta}$.

Regularization shrinks the sampling distribution and moves it towards $\mathbf{0}$, which reduces variance and introduce some bias (the actual effect is unknown because we don't know what bias is like).

• ***Implicit Regularization in Bayesian ML***

Moreover, for a generalized ML problem, if we take **frequentist statistics** as assumption (i.e. there is some $\boldsymbol{\theta}_0$ as ground truth) and **MLE** as our estimator, our solution is

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(y^i|\boldsymbol{x}^i;\boldsymbol{\theta})$$

However, if we take **Bayesian statistics** as assumption (i.e. there is some distribution of $\boldsymbol{\theta}$ as ground truth) and **MAP** (which assumes a prior distribution $\boldsymbol{\theta} \sim \text{Norm}(\mathbf{0}, \tau^2\boldsymbol{I})$) as estimator, our solution is

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(y^i|\boldsymbol{x}^i,\boldsymbol{\theta})p(\boldsymbol{\theta})$$

Obviously, the Bayesian interpretation introduces a regularization term $p(\boldsymbol{\theta})$ which assumes implicitly that $\boldsymbol{\theta}$ closer to $\mathbf{0}$ is more likely to appear. This trivial difference between MLE and MAP introduces implicit regularization without affecting any other things.

5 Basic Deep Learning

We define neural networks as extension of logistic regression. We consider logistic regression with $y = \sigma(\boldsymbol{w}\boldsymbol{x} + b)$ as a single neuron, the logarithm loss function of which is the MLELoss as $\ell(\boldsymbol{w}, b) = -\log\left(\hat{y}^y(1-\hat{y})^{1-y}\right) = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$. Here $\hat{y}(\boldsymbol{x}, b) = \sigma(\boldsymbol{w}\boldsymbol{x} + b)$.

Def Neuron: A combination of linear function and activation (non-linear part).

Def Layer: A series of neurons that are independent of each other.

Suppose the l-th layer has m neurons and accepts $\boldsymbol{x} \in \mathbb{R}^n$ as input, then the parameters of layer are $\boldsymbol{W}^{[l]} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b}^{[l]} \in \mathbb{R}^m$. The output of linear part is $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{x} + \boldsymbol{b}^{[l]}$; the output of whole layer is $\boldsymbol{a}^{[l]} = \sigma\left(\boldsymbol{z}^{[l]}\right)$. Then the architecture of the neural network with l layers is:

$$\boldsymbol{a}^{[0]} = \boldsymbol{x};\ \forall j,\ \boldsymbol{a}^{[j+1]} = \sigma\left(\boldsymbol{W}^{[j+1]}\boldsymbol{a}^{[j]} + \boldsymbol{b}^{[j+1]}\right);\ \boldsymbol{y} = \boldsymbol{a}^{[l]} \in \mathbb{R}^1$$

To parallel our code so that GPU acceleration is utilized, we input a 2D array $\boldsymbol{X} = \left[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(k)}\right] \in \mathbb{R}^{n\times k}$ (i.e. we take k inputs as one batch) instead. Here $\boldsymbol{b}^{[j]}$ is automatically broadcast to $\tilde{\boldsymbol{b}}^{[j]} = \left[\boldsymbol{b}^{[j]}, \boldsymbol{b}^{[j]}, ..., \boldsymbol{b}^{[j]}\right] \in \mathbb{R}^{m\times k}$.

For such a nerual network, we define our cost function as

$$\mathcal{J}\left(\left\{\boldsymbol{W}^{[j]}, \boldsymbol{b}^{[j]} | j = 1, 2, ..., l\right\}\right) = \frac{1}{k}\sum_{i=1}^{k}\mathcal{L}^{(i)} = -\frac{1}{k}\sum_{i=1}^{k}\left(y^{(i)}\log\hat{y}^{(i)} + \left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right)$$

Now we can use back propagation to update the network. For any layer $j = 1, 2, ..., l$, if the number of neurons is m_j, then the derivative of the neurons is

$$\frac{\partial\mathcal{L}^{(i)}}{\partial\boldsymbol{W}^{[j]}} = \frac{\partial\mathcal{L}^{(i)}}{\partial\boldsymbol{a}^{[j]}}\frac{\partial\boldsymbol{a}^{[j]}}{\partial\boldsymbol{z}^{[j]}}\frac{\partial\boldsymbol{z}^{[j]}}{\partial\boldsymbol{W}^{[j]}} = \sum_{p=1}^{m_j}\left[\frac{\partial\mathcal{L}^{(i)}}{\partial\boldsymbol{a}^{[j]}}\frac{\partial\boldsymbol{a}^{[j]}}{\partial\boldsymbol{z}^{[j]}}\right]_p\frac{\partial\boldsymbol{z}_p^{[j]}}{\partial\boldsymbol{W}^{[j]}}$$

where $\frac{\partial\boldsymbol{z}^{[j]}}{\partial\boldsymbol{W}^{[j]}} \in \mathbb{R}^{m_j\times m_j\times m_{j-1}}$. Because $\frac{\partial\boldsymbol{z}_p^{[j]}}{\partial\boldsymbol{W}^{[j]}}$ has only the p-th row that equals $\boldsymbol{a}^{[j-1]^\top}$ is non-zero,

$$\frac{\partial\mathcal{L}^{(i)}}{\partial\boldsymbol{W}^{[j]}} = \left(\frac{\partial\mathcal{L}^{(i)}}{\partial\boldsymbol{a}^{[j]}}\frac{\partial\boldsymbol{a}^{[j]}}{\partial\boldsymbol{z}^{[j]}}\right)^\top \odot \left(\boldsymbol{1}_{m_j} \times \boldsymbol{a}^{[j-1]^\top}\right)$$

Using

$$\frac{\partial\boldsymbol{a}^{[j+1]}}{\partial\boldsymbol{a}^{[j]}} = \frac{\partial\boldsymbol{a}^{[j+1]}}{\partial\boldsymbol{z}^{[j+1]}}\frac{\partial\boldsymbol{z}^{[j+1]}}{\partial\boldsymbol{a}^{[j]}} = \operatorname{diag}\left(\boldsymbol{a}^{[j+1]} \odot \left(\boldsymbol{1} - \boldsymbol{a}^{[j+1]}\right)\right)\boldsymbol{W}^{[j+1]} = \boldsymbol{a}^{[j+1]} \odot \left(\boldsymbol{1} - \boldsymbol{a}^{[j+1]}\right) \odot \boldsymbol{W}^{[j+1]}$$

we have

$$\frac{\partial\mathcal{L}^{(i)}}{\partial\boldsymbol{W}^{[j]}} = \frac{\partial\mathcal{L}^{(i)}}{\partial\hat{y}}\left(\prod_{q=l}^{j+1}\boldsymbol{a}^{[q]} \odot \left(\boldsymbol{1} - \boldsymbol{a}^{[q]}\right) \odot \boldsymbol{W}^{[q]}\right) \odot \boldsymbol{a}^{[j]} \odot \left(\boldsymbol{1} - \boldsymbol{a}^{[j]}\right) \odot \left(\boldsymbol{1}_{m_j} \times \boldsymbol{a}^{[j-1]^\top}\right)$$

5.1 Improve the Neural Network

We have to deal with gradient explosion or gradient vanishing in deep learning.

One solution lies in choosing a suitable activation. Sigmoid and Tanh both suffers from gradient vanishing when the input value is too big, so often ReLU is better.

Another solution is that we prevent such large values by implementing normalization. We affinely transform the given data so that the mean is zero and the variance is one. This not only restrict all variables to values near zero, but also makes the contour of loss function look more like sphere instead of flat ellipsoids, which helps gradient descent to converge more quickly. (Another way is to introduce mommentum, i.e. $\boldsymbol{v} = \beta\boldsymbol{v} + (1-\beta)\frac{\partial\mathcal{J}}{\partial\boldsymbol{x}}$ and $\boldsymbol{x} = \boldsymbol{x} - \alpha\boldsymbol{v}$.)

What's more, for deep networks, as the result is somewhat multiplication of a series of weight matrices $\boldsymbol{W}^{[j]}$, we want to make sure that these multiplications won't result in too large or too small results. Therefore we use initialization methods, that is we initialize $\boldsymbol{W}^{[j]}$ by random numbers near a certain value. To be exact, we want to ensure all the layers have almost the same variance. For a linear function $y = \boldsymbol{wx}$, we have $\mathrm{Var}(y) = \mathrm{Var}(\boldsymbol{wx}) = \sum_{i=1}^{n}\mathrm{Var}(w_i)\mathrm{Var}(x_i) = n\mathrm{Var}(w_i)\mathrm{Var}(x_i)$. Therefore we want $\mathrm{Var}(w_i) = 1/n$. Taking both forward propagation and backward propagation into account, it is recommended that $\mathrm{Var}(w_i) = \frac{2}{n_{\text{in}}+n_{\text{out}}}$. Given that a random variable $X \sim U(-a, a)$ has variance $\mathrm{Var}(X) = \frac{a^2}{3}$, we can let this a be $\sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}$.

6 Debugging ML Algorithms

Principle: analyze bias and variance. High bias equivalent to underfitting, while high variance equivalent to overfitting. Solution to high variance (overfitting) includes: (1) increase training examples, (2) decrease number of features (i.e. make the model simpler).

Principle: analyze which of algorithm and objective goes wrong. Suppose we have a specific target of getting an optimal $\boldsymbol{\theta}$, and there are two algorithms leading to $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$, with cost function $J_1(\boldsymbol{\theta})$ and $J_2(\boldsymbol{\theta})$. The performance of ALG1 is better than ALG2, then we compare $J_2(\hat{\boldsymbol{\theta}}_1)$ and $J_2(\hat{\boldsymbol{\theta}}_2)$.

- If $J_2(\hat{\boldsymbol{\theta}}_1) > J_2(\hat{\boldsymbol{\theta}}_2)$, then this means ALG2 (maybe) successfully minimizes the cost function, but a parameter performing bad on the cost function is better on our target, so this means we have to change our cost function.
- If $J_2(\hat{\boldsymbol{\theta}}_1) < J_2(\hat{\boldsymbol{\theta}}_2)$, then ALG2 fails to find the optimal solution of cost function, it is the algorithm that goes wrong.

Principle: in pipelines, decide which stage goes wrong. For each stage, we feed ideal results of the current stage to the next stages and measure the improved performance. We would try to modify the one component that tends to make largest improvement. (For parallel components, we can test both the improvement caused by each component independently, and the improvement of accumulating perfect outputs.)

Principle: ablative analysis to prove which part is the one that really matters. In a system with multiple parts, delete one at a time or cumulatively to test the removal of which component leads to the greatest decrease in performance.

7 Unsupervised Learning

7.1 Clustering

7.1.1 K-Means

Given a clustering problem with inputs $\mathcal{X} = \{\boldsymbol{x}^{(i)} \in \mathbb{R}^m | i = 1, 2, ..., n\}$, and k clusters, we output a partition $P = \{C_1, C_2, ..., C_k\}$ of $\mathcal{X}$. Let c_i be the cluster containing $\boldsymbol{x}^{(i)}$.

K-Means assumes that every cluster has a center $\boldsymbol{\mu}^j$ for each $j = 1, 2, ..., k$, and is desired to minimize $\mathcal{J}(P, \boldsymbol{\mu}) = \displaystyle\sum_{i=1}^{n} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{c_i}\|^2$. The process is:

- First randomly assign $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k$.
- For each item update its belonging cluster $c_i = \arg\min\limits_j \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^j\|^2$.
- For each cluster update its center $\boldsymbol{\mu}_j = \dfrac{\sum_{i=1}^n \mathbb{I}\{c_i = j\}\boldsymbol{x}^{(i)}}{\sum_{i=1}^n \mathbb{I}\{c_i = j\}}$. If not converging, then back to step 2.

It is pretty clear that step 2 minimizes $\mathcal{J}(P, \boldsymbol{\mu})$ with respect to P, while step 3 minimizes $\mathcal{J}(P, \boldsymbol{\mu})$ with respect to $\boldsymbol{\mu}$. Therefore, the algorithm must converge, but since $\mathcal{J}$ might not be convex, it is not assured that the algorithm gets global minima.

7.1.2 Expectation Maximization

Def Gaussian Mixed Model: A random variable $\boldsymbol{X} \in \mathbb{R}^m$ is said to obey GMM if there exists latent variable Z simulating a multinomial distribution with parameters $\boldsymbol{\phi} \in \mathbb{R}^k$, and $\boldsymbol{X}|Z = j \sim \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$. (i.e., $\boldsymbol{X}$ is drawn randomly from k Gaussians.)

Our target is to estimate $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \{(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)|j = 1, 2, ..., k\}\}$ with samples $\{\boldsymbol{x}^{(i)}|i = 1, 2, ..., n\}$ under GMM. The process of EM algorithm is:

- Initialize $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \{(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)|j = 1, 2, ..., k\}\}$.
- For each item, assign a value $w_j^{(i)} = \Pr(z^{(i)} = j|\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$ representing the probability of this item belonging to the j-th Gaussian.
- For each Gaussian, update the parameters by $\phi_j = \dfrac{1}{n}\displaystyle\sum_{i=1}^n w_j^{(i)}$, $\boldsymbol{\mu}_j = \dfrac{\sum_{i=1}^n w_j^{(i)} \boldsymbol{x}^{(i)}}{\sum_{i=1}^n w_j^{(i)}}$, and $\boldsymbol{\Sigma}_j = \dfrac{\sum_{j=1}^n w_j^{(i)} \left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\right)\left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\right)^\top}{\sum_{i=1}^n w_j^{(i)}}$. If not converging return to step 2.

We can generalize the EM algorithm for any problem with latent variables. The problem with latent variables is that if these variables are unknown, then our MLE would be too hard to compute (while given the values of latent variables, the calculation would be simplified a lot). For example, our MLE process is

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^n \log \Pr(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = \sum_{i=1}^n \log \left(\sum_{j=1}^k \phi_j \Pr(\boldsymbol{x}^{(i)}|z^{(i)} = j, \boldsymbol{\theta}) \right)$$

Setting the derivative to zero,

$$\frac{\partial \ell(\boldsymbol{\theta})}{\partial \phi_p} = \sum_{i=1}^n \frac{\Pr(\boldsymbol{x}^{(i)}|z^{(i)} = p, \boldsymbol{\theta})}{\sum_{j=1}^k \phi_j \Pr(\boldsymbol{x}^{(i)}|z^{(i)} = j, \boldsymbol{\theta})} = 0$$

which is extremely hard to give a explicit expression.

Therefore, EM algorithm finds a lower bound of the log likelihood function and maximizes this lower bound instead.

$$\begin{aligned}
\ell(\boldsymbol{\theta}) =& \sum_{i=1}^n \log \left(\sum_{j=1}^k \Pr(\boldsymbol{x}^{(i)}, z^{(i)} = j; \boldsymbol{\theta}) \right) = \sum_{i=1}^n \log \left(\sum_{j=1}^k Q_{z^{(i)}}(j) \frac{\Pr(\boldsymbol{x}^{(i)}, z^{(i)} = j; \boldsymbol{\theta})}{Q_{z^{(i)}}(j)} \right) \\
\geq& \sum_{i=1}^n \sum_{j=1}^k Q_{z^{(i)}}(j) \log \frac{\Pr(\boldsymbol{x}^{(i)}, z^{(i)} = j; \boldsymbol{\theta})}{Q_{z^{(i)}}(j)} := \mathrm{ELBO}(\boldsymbol{x}^{(1)} \sim \boldsymbol{x}^{(n)}; \boldsymbol{\theta}, Q_{z^{(1)}} \sim Q_{z^{(n)}})
\end{aligned}$$

where $Q_{z^{(i)}}$ is any probability distribution of $z^{(i)}$. The equality holds when $\dfrac{\Pr(\boldsymbol{x}^{(i)}, z^{(i)} = j; \boldsymbol{\theta})}{Q_{z^{(i)}}(j)}$ is a constant. Making the equality hold also means that, if we maximizes ELBO with respect to $\boldsymbol{\theta}$, we would also attain a larger value on $\ell(\boldsymbol{\theta})$. Therefore we, in each step, take

$$Q_{z^{(i)}}(j) = \Pr(z^{(i)} = j|\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Notice that, similar to the idea of K-Means, selecting such $Q_i(j)$s is also maximizing $\mathrm{ELBO}(\boldsymbol{x}; \boldsymbol{\theta}, Q)$ over Q. Our idea of EM is equivalent to: we change our function into one with higher dimension, and our target function is the envelope with respect to $\boldsymbol{\theta}$. Then we maximize the higher-dimensional function sequentially with $\boldsymbol{\theta}$ and Q. (The idea of SMO) In fact, we are using function $\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},Q)=\log p(\boldsymbol{x};\boldsymbol{\theta})-D_{\mathrm{KL}}(Q\|p_{z|\boldsymbol{x}})$. The proof of this maximization is listed below:

$$\begin{aligned}
&\max_{Q_i}\sum_{j=1}^{k}Q_i(j)\log\frac{\Pr(\boldsymbol{x}^{(i)},z^{(i)}=j;\boldsymbol{\theta})}{Q_i(j)}\text{ s.t. }\sum_{j=1}^{k}Q_i(j)=1\\
\Leftrightarrow&\log\Pr(\boldsymbol{x}^{(i)},z^{(i)}=j;\boldsymbol{\theta})-\log Q_i(j)-1+\lambda=0\\
\Leftrightarrow&Q_i(j)=\Pr(\boldsymbol{x}^{(i)},z^{(i)}=j;\boldsymbol{\theta})e^{\lambda-1}=\Pr(z^{(i)}=j|\boldsymbol{x}^{(i)};\boldsymbol{\theta})
\end{aligned}$$

Then we maximize $\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},Q)$ with this fixed Q. Specially, for GMM models,

$$\max_{\boldsymbol{\theta}}\mathrm{ELBO}(\boldsymbol{x};\boldsymbol{\theta},Q)\Leftrightarrow\max_{\boldsymbol{\theta}}\sum_{i=1}^{n}\sum_{j=1}^{k}Q_i(j)\left(\log\Pr\left(\boldsymbol{x}^{(i)}|z^{(i)}=j;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j\right)+\log\phi_j\right)$$

The second term is only relevant to $\boldsymbol{\phi}$ while the first term depends on only $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Therefore, for the $\boldsymbol{\phi}$ part, the problem is

$$\max_{\boldsymbol{\phi}}\sum_{i=1}^{n}\sum_{j=1}^{k}Q_i(j)\log\phi_j=\sum_{j=1}^{k}\left(\sum_{i=1}^{n}Q_i(j)\right)\log\phi_j\text{ s.t. }\sum_{j=1}^{k}\phi_j=1$$

Taking the Lagrangian and setting the derivative to zero, we have $\phi_j=\frac{1}{\lambda}\sum_{i=1}^{n}Q_i(j)$. Combined with the restriction $\sum_{j=1}^{k}\phi_j=1$, we know that $\lambda=n$.

For the $\boldsymbol{\mu}$ part, the problem is unconstrained, so we directly take the derivative.

$$\frac{\partial\mathrm{ELBO}}{\partial\boldsymbol{\mu}_j}=\sum_{i=1}^{n}Q_i(j)\frac{\partial}{\partial\boldsymbol{\mu}_j}\left(-\frac{1}{2}\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)^{\top}\boldsymbol{\Sigma}_j^{-1}\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)\right)=\sum_{i=1}^{n}Q_i(j)\boldsymbol{\Sigma}_j^{-1}\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)$$

Since $\boldsymbol{\Sigma}_j$ is invertible, the equation above equals zero means $\boldsymbol{\mu}_j=\dfrac{\sum_{i=1}^{n}Q_i(j)\boldsymbol{x}^{(i)}}{\sum_{i=1}^{n}Q_i(j)}$.

For the $\boldsymbol{\Sigma}$ part, we use some linear algebra techniques. The maximizing problem is equivalent to

$$\max_{\boldsymbol{\Sigma}_j}\mathrm{ELBO}\Leftrightarrow\max_{\boldsymbol{\Sigma}_j}\sum_{i=1}^{n}Q_i(j)\left(-\frac{1}{2}\log|\boldsymbol{\Sigma}_j|-\frac{1}{2}\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)^{\top}\boldsymbol{\Sigma}_j^{-1}\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)\right)$$

We assume $\boldsymbol{\Sigma}_0=\sum_{i=1}^{n}Q_i(j)\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)^{\top}$, use (1) $\det(\boldsymbol{AB})=\det\boldsymbol{A}\times\det\boldsymbol{B}$, (2) $\det\boldsymbol{A}$ is the product of all eigenvalues of $\boldsymbol{A}$, (3) $\mathrm{tr}\boldsymbol{A}$ is the sum of all eigenvalues of $\boldsymbol{A}$, (4) for vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $\mathrm{tr}(\boldsymbol{x}\boldsymbol{y}^{\top})=\boldsymbol{y}^{\top}\boldsymbol{x}$, we have

$$\max_{\boldsymbol{\Sigma}_j}\mathrm{ELBO}\Leftrightarrow\max_{\boldsymbol{\Sigma}_j}\frac{1}{2}\left(\sum_{i=1}^{n}Q_i(j)\right)\log\det\left(\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\Sigma}_0\right)-\frac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\Sigma}_0\right)$$

Assuming the eigenvalues of $\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\Sigma}_0$ are $\lambda_1,\lambda_2,...,\lambda_m$, then

$$\max_{\boldsymbol{\Sigma}_j}\mathrm{ELBO}\Leftrightarrow\max_{\boldsymbol{\Sigma}_j}\left(\sum_{i=1}^{n}Q_i(j)\right)\sum_{k=1}^{m}\log\lambda_k-\sum_{k=1}^{m}\lambda_k\Leftrightarrow\lambda_k=\sum_{i=1}^{n}Q_i(j)$$

Now, because $\boldsymbol{\Sigma}_j=\boldsymbol{\Sigma}_j^{\top}$, $\boldsymbol{\Sigma}_0=\boldsymbol{\Sigma}_0^{\top}$, we know that $\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\Sigma_0}$ is a real-valued symmetric matrix, which, we can prove by induction, can be diagonalized. Therefore, $\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\Sigma_0}=\sum_{i=1}^{n}Q_i(j)\boldsymbol{I}$, and

$$\boldsymbol{\Sigma}_j=\frac{\boldsymbol{\Sigma}_0}{\sum_{i=1}^{n}Q_i(j)}=\frac{\sum_{i=1}^{n}Q_i(j)\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)\left(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_j\right)^{\top}}{\sum_{i=1}^{n}Q_i(j)}$$

7.2 PCA and ICA

Notice that there are defects in EM family. As dimension of data increases, the performance would be worse, and moreover, when the dimension of data m is larger than the number of examples n, the $\boldsymbol{\Sigma}_j$ derived above would be singular, which makes the algorithm fails. One solution is to solve for certain models under extremely high dimensions, while the other is to map the data to a very low dimension, which turns out to be effective in multiple situations.

Principal Component Analysis

The problem is defined as: Given data $\{\boldsymbol{x}^{(i)} \in \mathbb{R}^m | i = 1, 2, ..., n\}$, we want to find such a subspace $\boldsymbol{V} = \text{span}\{\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_k\}$ (where $\|\boldsymbol{u}_j\|_2 = 1$ and $k \ll m$) that when the data are projected onto $\boldsymbol{V}$, the data retain their features most.

Mathematically, the projected data is $\hat{\boldsymbol{x}}^{(i)} = \sum_{i=1}^{k} z_k^{(i)} \boldsymbol{u}_k$ (where $z_k^{(i)} = \boldsymbol{u}_k^\top \boldsymbol{x}^{(i)}$), so in the subspace $\boldsymbol{V}$, its position is $\boldsymbol{z}^{(i)} \in \mathbb{R}^k$. We define

$$\boldsymbol{U} = \begin{bmatrix} -\boldsymbol{u}_1^\top - \\ -\boldsymbol{u}_2^\top - \\ \vdots \\ -\boldsymbol{u}_k^\top - \end{bmatrix} \in \mathbb{R}^{k \times m}$$

Then the subspace position is $\boldsymbol{z}^{(i)} = \boldsymbol{U}\boldsymbol{x}^{(i)}$.

We want the length of projected vectors to be maximal, so the problem is

$$\max_{\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_k} \sum_{i=1}^{n} \|\boldsymbol{U}\boldsymbol{x}^{(i)}\|_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{k} \|\boldsymbol{u}_j^\top \boldsymbol{x}^{(i)}\|_2^2 = \sum_{j=1}^{k} \boldsymbol{u}_j^\top \left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)^\top} \right) \boldsymbol{u}_j \text{ s.t. } \|\boldsymbol{u}_j\|_2^2 = 1$$

Let $\boldsymbol{\Sigma}_0 = \sum_{i=1}^{n} \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)^\top}$, then taking the derivative of Lagrangian,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_j} = 2\boldsymbol{\Sigma}_0 \boldsymbol{u}_j + 2\lambda_j \boldsymbol{u}_j = 0$$

Therefore, all the $\boldsymbol{u}_j$s are eigenvectors of $\boldsymbol{\Sigma}_0$. Then the problem is simply maximizing $\sum_{j=1}^{k} \lambda_j^2$, which means we only have to choose $\boldsymbol{u}_j$s to be the eigenvectors corresponding to the first k largest eigenvalues.

***** Notice that this interpretation of preserving as much features as possible only works for standardized data, so we should subtract the mean and divide the standard variation first.

Independent Component Analysis

Another problem is intended to solve the cocktail party problem: In a party, several microphones are used to record audio from different sound sources, so we want to figure out the content of each sound source.

Mathematically, we assume the sound generated is $\{\boldsymbol{s}^{(i)} \in \mathbb{R}^n | i = 1, 2, ..., M\}$ where $s_j^{(i)}$ is the signal from speaker j at time i, and what we receive is $\{\boldsymbol{x}^{(i)} \in \mathbb{R}^n | \boldsymbol{x}^{(i)} = \boldsymbol{A}\boldsymbol{s}^{(i)}\}$. Our target is to find $\boldsymbol{W} = \boldsymbol{A}^{-1}$ whose rows are $\boldsymbol{w}_1^\top, \boldsymbol{w}_2^\top, ..., \boldsymbol{w}_n^\top$. We model $\boldsymbol{s}^{(i)}$ as n dimensional continuous distribution on $\mathbb{R}^n$ and each component is independent. Therefore, $p(\boldsymbol{s}^{(i)}) = \prod_{j=1}^{n} p_S(s_j^{(i)})$, where $p_S(s)$ is the distribution of any single sound source. Usually we can take Gaussian or Laplacian or the most commonly used, Sigmoid function with cumulative distribution function $F_S(s) = \frac{1}{1+e^{-s}}$ and $p_S(s) = F_S'(s)$.

$\underline{\underline{\textbf{Lemma}}}$: Given jointly continuous $X_1, X_2, ..., X_n$ with density function $f(x_1, x_2, ..., x_n)$, jointly continuous $Y_1, Y_2, ..., Y_n$ with density function $h(y_1, y_2, ..., y_n)$, bijection $g : \mathbb{R}^n \to \mathbb{R}^n$ such that $(Y_1, Y_2, ..., Y_n) = g(X_1, X_2, ..., X_n)$, and the continuous Jacobian matrix $\boldsymbol{J}$ of function g, then

$$h(y_1 \sim y_n) = f\left(g^{-1}(x_1 \sim x_n)\right) \left|\boldsymbol{J}\left(g^{-1}(x_1 \sim x_n)\right)\right|^{-1}$$

Using the lemma above, we know that $p(\boldsymbol{x}^{(i)}) = p(\boldsymbol{W}\boldsymbol{x}^{(i)})\,|\boldsymbol{W}|$, then we can use gradient ascent to do MLE.

$$\ell(\boldsymbol{W}) = \sum_{i=1}^{M} \left(\log p(\boldsymbol{W}\boldsymbol{x}^{(i)}) + \log|\boldsymbol{W}|\right) = M\log|\boldsymbol{W}| + \sum_{i=1}^{M}\sum_{j=1}^{n} \log p_S\left(\boldsymbol{w}_j\boldsymbol{x}^{(i)}\right)$$

Then the update rule is

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \alpha\left(M\left(\boldsymbol{W}^{-1}\right)^\top + \sum_{i=1}^{M}\left(\mathbf{1} - 2\sigma\left(\boldsymbol{W}\boldsymbol{x}^{(i)}\right)\right)\boldsymbol{x}^{(i)^\top}\right)$$

8 Reinforcement Learning

8.1 RL by Discrete MDP with Full Knowledge

Reinforcement learning problem is modelled by a **Markov Decision Process** defined as a five-tuple $(S, A, \{P_{sa}\}, \gamma, R)$ where S is the set of states, A is the set of actions, $P_{s_0a_0}(s')$ is the probability of getting to state s' after making action a_0 in state s_0, γ is the discount factor, $R : S \times A \to \mathbb{R}$ or $R : S \to \mathbb{R}$ is the reward function (when modelled as the former, R takes into account the cost of taking different actions, e.g. moving costs more electricity than staying still).

Def Long-Run Reward: for a MDP taking states $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} ...$, the long run reward is $R_\infty = \sum_{i=0}^{+\infty} \gamma^i R(s_i)$.

Def Policy: a policy is a function $\pi : S \to A$. We are executing policy π if whatever state s we are in, we are taking action $a = \pi(s)$.

Def Value Function: the value function of a policy π is $V^\pi(s) = \mathbb{E}[R_\infty | s_0 = s, \pi]$.

Def Optimal Value Function: the optimal value function is $V^*(s) = \max_\pi V^\pi(s)$.

Lemma Bellman's Equations: a recursive way of describing (optimal) value function is

$$\begin{aligned} V^\pi(s) =& \mathbb{E}\left[\sum_{i=0}^{+\infty}\gamma^i R(s_i) | s_0 = s, \pi\right] = R(s) + \gamma \sum_{s'\in S} P_{s\pi(s)}(s')\mathbb{E}\left[\sum_{i=1}^{+\infty}\gamma^{i-1}R(s_i)|s_1 = s', \pi\right] \\ =& R(s) + \gamma\sum_{s'\in S} P_{s\pi(s)}(s')V^\pi(s') \end{aligned}$$

(notice that the last equality holds because of the independency of decisions and past states) and

$$\begin{aligned} V^*(s) =& \max_\pi V^\pi(s) = \max_\pi\left(R(s_0) + \gamma\sum_{s'\in S} P_{s\pi(s)}(s')V^\pi(s')\right) \\ =& R(s) + \gamma\max_\pi\sum_{s'\in S} P_{s\pi(s)}(s')V^\pi(s') \end{aligned}$$

Def Optimal Policy: the optimal policy $\pi^* : S \to A$ is defined by

$$\pi^*(s) = \arg\max_{a\in A}\sum_{s'\in S} P_{s\pi(s)}(s')V^\pi(s')$$

Therefore $V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$ for any policy π. Notice that Bellman's equations provide (through iteration) a general solution to any finite-state reinforcement learning problems where all five elements of the MDP is given, ignorant of computing costs. Two algorithms to solve the recursion problem are **value iteration** and policy iteration.

Value iteration does:

- Initialize $V(s) \leftarrow 0$ for all states s.
- Repeat until convergence that $\forall s \in S$, update $V(s) \leftarrow R(s) + \gamma \max\limits_{a\in A} \sum\limits_{s'\in S} P_{sa}(s')V(s')$.

while policy iteration does:

- Initialize π randomly.
- Repeat until convergence that $V \leftarrow V^{\pi}$ (through linear system solver) and $\forall s \in S$, update $\pi(s) \leftarrow \arg\max\limits_{a\in A} \sum\limits_{s'\in S} P_{sa}(s')V(s')$.

8.2 RL without Knowledge of P_{sa} and R

With unknown P_{sa} and R, we should estimate it from data collection. We make a series of trials, that is expressed as: for the i-th trial, the MDP is $s_0^{(i)} \xrightarrow{a_0^{(i)}} s_1^{(i)} \xrightarrow{a_1^{(i)}} s_2^{(i)} \xrightarrow{a_2^{(i)}} \ldots$

Then we can model $P_{sa}(s') = \dfrac{\text{\#times we took action } a \text{ in state } s \text{ and got to } s'}{\text{\#times we took action } a \text{ in state } s}$ (Laplace smoothing might be helpful). In algorithms we usually let trials of MDP be a part of the loop step instead of letting it be a distinct step to ensure best performance.

8.3 RL with Continuous MDP

Continuous MDP deals with real-life tasks like robot control, e.g. for an autonomous car, $S = \{(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})\} = \mathbb{R}^6$, $A = \mathbb{R}^2$ is steering and braking. If the number of features is small, then we can discretize. But usually we can't do so and we have to deal with continuous case.

For these tasks, we, obviously, don't know P_{sa}, so we have to make estimation. One solution is to use physics simulation, while another solution is to do n trials and use the data to predict $s_{t+1} = f_\phi(s_t, a_t)$ (where the ϕ is the parameters). The parameters ϕ can be learned by $\arg\min\limits_{\phi} \sum\limits_{i=1}^{n} \sum\limits_{t=0}^{T-1} \left\| s_{t+1}^{(i)} - f_\phi\left(s_t^{(i)}, a_t^{(i)}\right) \right\|_2^2$. Having learned the parameters, we use a stochastic model $s_{t+1} = f_\phi(s_t, a_t) + \epsilon_t$ where $\epsilon_t \sim \mathcal{N}(0, \mathbf{\Sigma})$ is the noise.

In fitted value iteration, we assume $V(s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s)$ where $\boldsymbol{\phi}(s)$ is appropriate feature mapping, e.g. for the autonomous car instance, maybe $\dot{x}\theta$ is helpful. The pseudo-code of the algorithm shown in **Algorithm 1**. After finishing training by this algorithm, in actual testing, we let $\epsilon_t = 0$ and only sample one s_1' to calculate $q(a)$ for all actions under current state and select our action through $\arg\min_a q(a)$.

Algorithm 1 Fitted Value Iteration

```
1: Randomly sample $n$ states $s^{(1)}, s^{(2)}, ..., s^{(n)} \in S$, $\boldsymbol{\theta} \leftarrow \mathbf{0}$
2: repeat
3:     for $s^{(i)}$ in sampled $n$ states do                                    ▷ update current $V^*$
4:         for action $a \in A$ do
5:             Sample $k$ next states using our learned model or simulator $s'_1, s'_2, ..., s'_k \sim P_{s^{(i)}a}$
6:             $q(a) \leftarrow R(s^{(i)}) + \gamma \frac{1}{k} \sum_{j=1}^{k} V(s'_j)$        ▷ as an estimate of $V^\pi(s^{(i)})$ for all possible $\pi$
           $y^{(i)} = \max_a q(a)$                       ▷ as an estimate of $V^*(s^{(i)})$ using Bellman's Equation
7:     $\boldsymbol{\theta} \leftarrow \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{n} \left(\boldsymbol{\theta}^\top \boldsymbol{\phi}\left(s^{(i)}\right) - y^{(i)}\right)^2$     ▷ generalize the updation of sampled $n$ states
   to all states in $S$
8: until convergence
```