Advanced Machine Learning

Cristian Perez Jensen

January 20, 2025

Note that these are not the official lecture notes of the course, but only notes written by a student of the course. As such, there might be mistakes. The source code can be found at `github.com/cristianpjensen/eth-cs-notes`. If you find a mistake, please create an issue or open a pull request.

Contents

List of symbols

$\doteq$	Equality by definition
$\approx$	Approximate equality
$\propto$	Proportional to
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
$i : j$	Set of natural numbers between i and j. *I.e.*, $\{i, i+1, \ldots, j\}$
$f : A \to B$	Function f that maps elements of set A to elements of set B
$\mathbb{1}\{\text{predicate}\}$	Indicator function (1 if predicate is true, otherwise 0)
$\boldsymbol{v} \in \mathbb{R}^n$	n-dimensional vector
$\boldsymbol{M} \in \mathbb{R}^{m \times n}$	$m \times n$ matrix
$\boldsymbol{M}^\top$	Transpose of matrix $\boldsymbol{M}$
$\boldsymbol{M}^{-1}$	Inverse of matrix $\boldsymbol{M}$
$\det(\boldsymbol{M})$	Determinant of $\boldsymbol{M}$
$\frac{\mathrm{d}}{\mathrm{d}x} f(x)$	Ordinary derivative of $f(x)$ w.r.t. x at point $x \in \mathbb{R}$
$\frac{\partial}{\partial x} f(\boldsymbol{x})$	Partial derivative of $f(\boldsymbol{x})$ w.r.t. x at point $\boldsymbol{x} \in \mathbb{R}^n$
$\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R}^n$	Gradient of $f : \mathbb{R}^n \to \mathbb{R}$ at point $\boldsymbol{x} \in \mathbb{R}^n$
$\boldsymbol{\nabla}^2_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$	Hessian of $f : \mathbb{R}^n \to \mathbb{R}$ at point $\boldsymbol{x} \in \mathbb{R}^n$

1 *Paradigms of data science*

Let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be i.i.d. samples, generated by an unknown distribution P. Assume that this distribution is in a distribution family,

$$\mathcal{H} = \{p(\cdot \mid \boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}.$$

The goal is to learn the parameters $\boldsymbol{\theta}$ that fit the data $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ best.

Frequentism. In frequentism, the maximum likelihood estimator (MLE) parameters maximize the following,

$$\begin{aligned}\boldsymbol{\theta}^\star &\in \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \mid \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \log p(\boldsymbol{x}_i \mid \boldsymbol{\theta}).\end{aligned}$$

Bayesianism. Bayesianism assumes that there is a prior over distributions. The maximum a posteriori (MAP) parameters maximize the following,

$$\begin{aligned}\boldsymbol{\theta}^\star &\in \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\boldsymbol{\theta} \mid \boldsymbol{X}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \mid \boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}) && \text{Bayes' rule.} \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log p(\boldsymbol{x}_i \mid \boldsymbol{\theta}).\end{aligned}$$

In practice, the prior acts as a regularization term.

Statistical learning. Now, assume that we have labeled samples $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$, where y is the target variable. Let $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. For a predictor function $f : \mathcal{X} \to \mathcal{Y}$, we define its risk as the expected loss,

$$\mathcal{R}(f) \doteq \mathbb{E}_{X,Y}[\ell(y, f(\boldsymbol{x}))].$$

In statistical learning, we want to find a function that minimizes the risk. However, since the distribution over X, Y is unknown, we cannot compute $\mathcal{R}(f)$ directly. Instead, we use the empirical risk as a surrogate,

$$\hat{\mathcal{R}}(f) \doteq \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i)).$$

The goal is to obtain the empirical risk minimizer,

$$f^\star \in \operatorname*{argmin}_{f\in\mathcal{F}} \hat{\mathcal{R}}(f),$$

where $\mathcal{F}$ is a family of functions that we assume f belongs to.

2 *Anomaly detection*

In anomaly detection, we are given a sample of objects $\mathcal{X} \subseteq \mathbb{R}^d$ with a normal class $\mathcal{N} \in \mathcal{X}$—the data points that we wish to keep. We wish to construct a function $\phi : \mathcal{X} \to \{0, 1\}$, such that

$$\phi(\boldsymbol{x}) = 1 \iff \boldsymbol{x} \notin \mathcal{N}.$$

Formally, an anomaly is an unlikely event. Hence, the strategy is to fit a model of a parametric family of distributions to the data $\mathcal{X}$,

$$\mathcal{H} = \{p(\cdot \mid \boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}.$$

Then, we define the anomaly score of $\boldsymbol{x}$ as a low probability $p(\boldsymbol{x} \mid \boldsymbol{\theta}^\star)$ according to the optimal model in this hypothesis class.

Anomaly detection in a high-dimensional space is hard, because the normal class can be very complex. The idea is to project $\mathcal{X}$ down to a lower dimensionality and perform anomaly detection there—hopefully the projected version of the normal class $\Pi(\mathcal{N})$ is less complex. In order to find the optimal linear projection, we will use principal component analysis (PCA).

Furthermore, it has been observed that linear projections of high-dimensional distributions onto low-dimensional spaces resemble Gaussian distributions. Hence, after performing PCA, we will fit a Gaussian mixture model (GMM) to the projected data.

Principal component analysis. The goal of PCA is to linearly project $\mathbb{R}^d$ to $\mathbb{R}^{d^-}$ such that the maximum amount of variance of the data is preserved.[1] Consider the base case $d^- = 1$. Let $\boldsymbol{u} \in \mathbb{R}^d$ with $\|\boldsymbol{u}\| = 1$, we project onto $\boldsymbol{u}$ by inner product,

$$\boldsymbol{x} \mapsto \boldsymbol{u}^\top \boldsymbol{x}.$$

[1] Components with larger variance are more informative.

The sample mean of the reduced dataset is computed by

$$\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{u}^\top \boldsymbol{x}_i = \boldsymbol{u}^\top \bar{\boldsymbol{x}},$$

where $\boldsymbol{x}$ is the sample mean of the original dataset. Further, the sample variance of the reduced dataset is

$$\begin{aligned}
\frac{1}{n} \sum_{i=1}^{n} \left(\boldsymbol{u}^\top \boldsymbol{x}_i - \boldsymbol{u}^\top \bar{\boldsymbol{x}} \right)^2 &= \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{u}^\top (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top \boldsymbol{u} \\
&= \boldsymbol{u}^\top \left(\frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x} - \bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}})^\top \right) \boldsymbol{u} \\
&= \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u},
\end{aligned}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the dataset. Since we want the projection that preserves the maximum variance, we have the following objective,

$$\boldsymbol{u}^\star \in \operatorname*{argmax}_{\|\boldsymbol{u}\|=1} \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}.$$

The Lagrangian of this problem is

$$\mathcal{L}(\boldsymbol{u};\lambda) = \boldsymbol{u}^\top \boldsymbol{\Sigma}\boldsymbol{u} + \lambda\left(1 - \|\boldsymbol{u}\|^2\right)$$

with gradient

$$\frac{\partial \mathcal{L}(\boldsymbol{u};\lambda)}{\partial \boldsymbol{u}} = 2\boldsymbol{\Sigma}\boldsymbol{u} - 2\lambda\boldsymbol{u} \stackrel{!}{=} 0.$$

So, $\boldsymbol{u}$ must satisfy $\boldsymbol{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$—$\boldsymbol{u}$ is an eigenvector of $\boldsymbol{\Sigma}$. It is easy to see that this must be the principal eigenvector by rewriting the objective,

$$\begin{aligned}
\boldsymbol{u}^\star &\in \operatorname*{argmax}_{\|\boldsymbol{u}\|=1} \boldsymbol{u}^\top \boldsymbol{\Sigma}\boldsymbol{u} \\
&= \operatorname*{argmax}_{\substack{\|\boldsymbol{u}\|=1 \\ (\boldsymbol{u},\lambda)\in\operatorname{eig}(\boldsymbol{\Sigma})}} \lambda\|\boldsymbol{u}\|^2 \\
&= \operatorname*{argmax}_{\substack{\boldsymbol{u}\in\mathbb{R}^d \\ (\boldsymbol{u},\lambda)\in\operatorname{eig}(\boldsymbol{\Sigma})}} \lambda \\
&= \boldsymbol{u}_1.
\end{aligned}$$

For $d^- > 1$, the remaining principal components can be computed with a similar idea. Iteratively, we factor out the previous principal components and do as above on the transformed dataset. For example, to get the second principal component, we first factor out the first principal component,

$$\mathcal{X}_1 \doteq \{\boldsymbol{x} - \operatorname{proj}_{\boldsymbol{u}_1}(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}\} = \{\boldsymbol{x} - \boldsymbol{u}_1^\top \boldsymbol{x} \cdot \boldsymbol{u}_1 \mid \boldsymbol{x} \in \mathcal{X}\}.$$

Then, we do the same as above.

Gaussian mixture model. The probability density function (PDF) of a Gaussian mixture model with k components is formalized as a convex combination of Gaussians,

$$p(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j).$$

The parameters of this model are

$$\boldsymbol{\theta} = \{\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j \mid j \in [k]\},$$

where $\sum_{j=1}^{k} \pi_j = 1$ and $\{\boldsymbol{\Sigma}_j \mid j \in [k]\}$ are positive definite. We fit the parameters of this model by maximizing the log-likelihood,

$$\begin{aligned}
\boldsymbol{\theta}^\star &\in \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \log p(\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\};\boldsymbol{\theta}) \\
&= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \log p(\boldsymbol{x}_i;\boldsymbol{\theta}) \\
&= \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \log \sum_{j=1}^{k} \pi_j \mathcal{N}(\boldsymbol{x}_i;\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j).
\end{aligned}$$

1: Initialize $\boldsymbol{\theta}_0$
2: **for** $t \in [T]$ **do**
3: $\quad q^\star \in \operatorname{argmin}_q E(q, \boldsymbol{\theta}_{t-1})$
4: $\quad \boldsymbol{\theta}_t \in \operatorname{argmax}_{\boldsymbol{\theta}} M(q^\star, \boldsymbol{\theta})$
5: **end for**
6: **return** $\boldsymbol{\theta}_T$

Algorithm 1. The expectation-maximization algorithm, where

$$M(q, \boldsymbol{\theta}) \doteq \mathbb{E}_{z\sim q}\left[\log \frac{p(\boldsymbol{X}, z; \boldsymbol{\theta})}{q(z)}\right]$$
$$E(q, \boldsymbol{\theta}) \doteq \mathbb{E}_{z\sim q}\left[\log \frac{q(z)}{p(z \mid \boldsymbol{X}; \boldsymbol{\theta})}\right].$$

Note that the above is intractable, so we would like to decompose it into tractable terms that can be computed. Let's assume that we know from which latent variable each data point was generated, then we can compute the MLE of the extended dataset $\{(\boldsymbol{x}_i, z_i) \mid i \in [n]\}$ as

$$\begin{aligned}\log p(\boldsymbol{X}, \boldsymbol{z}; \boldsymbol{\theta}) &= \sum_{i=1}^n \log p(\boldsymbol{x}_i, z_i) \\ &= \sum_{i=1}^n \log(\pi_{z_i} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i})) \\ &= \sum_{i=1}^n \log \pi_{z_i} + \sum_{i=1}^n \log \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i}),\end{aligned}$$

which is tractable to maximize. Let q be a distribution over $[k]$, then we can rewrite the log-likelihood into tractable terms,

$$\begin{aligned}\log p(\boldsymbol{X}; \boldsymbol{\theta}) &= \mathbb{E}_{z\sim q}[\log p(\boldsymbol{X}; \boldsymbol{\theta})] \\ &= \mathbb{E}_{z\sim q}\left[\log\left(\frac{p(\boldsymbol{X}, z; \boldsymbol{\theta})}{p(z \mid \boldsymbol{X}; \boldsymbol{\theta})}\right)\right] \\ &= \mathbb{E}_{z\sim q}\left[\log\left(\frac{p(\boldsymbol{X}, z; \boldsymbol{\theta})}{p(z \mid \boldsymbol{X}; \boldsymbol{\theta})}\frac{q(z)}{q(z)}\right)\right] \\ &= \underbrace{\mathbb{E}_{z\sim q}\left[\log \frac{p(\boldsymbol{X}, z; \boldsymbol{\theta})}{q(z)}\right]}_{\doteq M(q, \boldsymbol{\theta})} + \underbrace{\mathbb{E}_{z\sim q}\left[\log \frac{q(z)}{p(z \mid \boldsymbol{X}; \boldsymbol{\theta})}\right]}_{\doteq E(q, \boldsymbol{\theta})}.\end{aligned}$$

These terms have the following two properties,

$$\begin{aligned}&\log p(\boldsymbol{X}; \boldsymbol{\theta}) \geq M(q, \boldsymbol{\theta}), \quad \forall q, \boldsymbol{\theta} \\ &\log p(\boldsymbol{X}; \boldsymbol{\theta}) = M(q^\star, \boldsymbol{\theta}), \quad q^\star = p(\cdot \mid \boldsymbol{X}; \boldsymbol{\theta}), \quad \forall \boldsymbol{\theta}.\end{aligned}$$

Hence, we can use $M(q^\star, \boldsymbol{\theta})$ as an approximation of $\log p(\boldsymbol{X}; \boldsymbol{\theta})$ around $\boldsymbol{\theta}$.

Theorem 2.1 (EM algorithm convergence)**.** Using the expectation-maximization algorithm, $\{\log p(\boldsymbol{x}; \boldsymbol{\theta}_t)\}_{t=0}^T$ is non-decreasing.

Proof. Given a data point $\boldsymbol{x}$ and current parameters $\boldsymbol{\theta}$, we have the following update,

$$\boldsymbol{\theta}' \in \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} M(q^\star, \boldsymbol{\theta}).$$

Hence, we have

$$\log p(\boldsymbol{x}) = M(q^\star, \boldsymbol{\theta}) \leq M(q^\star, \boldsymbol{\theta}') \leq \log p(\boldsymbol{x}; \boldsymbol{\theta}').$$

Thus, $\{\log p(\boldsymbol{x}; \boldsymbol{\theta}_t)\}_{t=0}^{T}$ is non-decreasing. ■

Summary. In conclusion, given a set of data points $\mathcal{X}$ with normal points $\mathcal{N} \subseteq \mathcal{X}$, we train an anomaly detector as follows,

1. Fit a projector $\pi : \mathbb{R}^d \to \mathbb{R}^{d^-}$ using PCA;
2. Fit a probability density function $p(\cdot \mid \boldsymbol{\theta})$ with k components to $\{\pi(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{X}\}$ using the EM algorithm;
3. For a point $\boldsymbol{x} \in \mathcal{X}$, its "anomaly score" is computed by $-\log p(\pi(\boldsymbol{x}); \boldsymbol{\theta})$.

3 *Density estimation*

In this section, we will consider parametric models,

$$\{p(\boldsymbol{x};\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}.$$

The problem we concern ourselves with is finding the best parameter $\boldsymbol{\theta}$. The most common method of finding the best parameters is maximum likelihood estimation (MLE),

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\text{MLE}} &\in \operatorname*{argmax}_{\boldsymbol{\theta}\in\Theta} \prod_{i=1}^{n} p(\boldsymbol{x}_i;\boldsymbol{\theta}) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} -\sum_{i=1}^{n} \log p(\boldsymbol{x}_i;\boldsymbol{\theta}).\end{aligned}$$

The following properties makes the MLE estimator attractive,

1. Consistency—in the limit of n, $\hat{\boldsymbol{\theta}}_{\text{MLE}}$ converges to the true parameter $\boldsymbol{\theta}^\star$;
2. Equivariance—if $\hat{\boldsymbol{\theta}}$ is the MLE of $\boldsymbol{\theta}$, then $g(\hat{\boldsymbol{\theta}})$ is the MLE of $g(\boldsymbol{\theta})$ for any function g;
3. Asymptotically normal—in the limit of n, $\hat{\theta}_{\text{MLE}} - \theta / \sqrt{n}$ converges to a random variable with distribution $\mathcal{N}(0, \mathcal{I}(\boldsymbol{\theta})^{-1})$, where $\mathcal{I}$ is the Fisher information matrix;
4. Asymptotically efficient—in the limit of n, the MLE estimator has the smallest variance of all unbiased estimators.

We can understand the asymptotic efficiency (property 4) of estimators better by the Rao–Cramér bound, which provides a bound on the variance of the estimator. We will only consider the general scalar case, but it generalizes to the multivariate case.

Theorem 3.1 (Rao–Cramér bound (scalar case))**.** For any (scalar) unbiased estimator $\hat{\theta} : \mathcal{Y}^n \to \mathbb{R}$, given n data points, of $\theta \in \mathbb{R}$, we have the following bound on its variance,

$$\operatorname{Var}\left[\hat{\theta}(\boldsymbol{y})\right] \geq \frac{\left(\frac{\partial}{\partial\theta} b_{\hat{\theta}} + 1\right)^2}{\mathcal{I}_n(\theta)} + b_{\hat{\theta}}^2,$$

where $\mathcal{I}_n(\theta)$ is the Fisher information,

$$\mathcal{I}_n(\theta) \doteq \mathbb{E}_{\boldsymbol{y}|\theta}\left[\left(\frac{\partial}{\partial\theta} \log p(\boldsymbol{y} \mid \theta)\right)^2\right] \overset{\text{iid}}{=} n \cdot \mathcal{I}(\theta),$$

and $b_{\hat{\theta}}$ is the bias of $\hat{\theta}$,

$$b_{\hat{\theta}} \doteq \mathbb{E}_{\boldsymbol{y}|\theta}\left[\hat{\theta}(\boldsymbol{y})\right] - \theta.$$

Proof. Let the "score" be defined as follows,

$$\Lambda(\boldsymbol{y}, \theta) \doteq \frac{\partial}{\partial \theta} \log p(\boldsymbol{y} \mid \theta) = \frac{1}{p(\boldsymbol{y} \mid \theta)} \frac{\partial}{\partial \theta} p(\boldsymbol{y} \mid \theta).$$

Then we have $\mathcal{I}_n(\theta) = \mathbb{E}_{\boldsymbol{y}|\theta}\big[\Lambda(\boldsymbol{y}, \theta)^2\big]$.

The expected score is equal to zero,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{y}|\theta}[\Lambda(\boldsymbol{y}, \theta)] &= \int p(\boldsymbol{y} \mid \theta) \Lambda(\boldsymbol{y}, \theta) \mathrm{d}\boldsymbol{y} \\
&= \int \frac{\partial}{\partial \theta} p(\boldsymbol{y} \mid \theta) \mathrm{d}\boldsymbol{y} \\
&= \frac{\partial}{\partial \theta} \int p(\boldsymbol{y} \mid \theta) \mathrm{d}\boldsymbol{y} \\
&= \frac{\partial}{\partial \theta} 1 \\
&= 0.
\end{aligned}$$

(Hence, the Fisher information is equivalent to the variance of the score.) Furthermore, the cross-correlation between $\Lambda(\boldsymbol{y}, \theta)$ and $\hat{\theta}(\boldsymbol{y})$ can be computed as

$$\begin{aligned}
\mathrm{Cov}_{\boldsymbol{y}|\theta}\big(\Lambda(\boldsymbol{y}, \theta), \hat{\theta}(\boldsymbol{y})\big) &= \mathbb{E}_{\boldsymbol{y}|\theta}\big[(\Lambda(\boldsymbol{y}, \theta) - \mathbb{E}[\Lambda(\boldsymbol{y}, \theta)])(\hat{\theta}(\boldsymbol{y}) - \mathbb{E}\big[\hat{\theta}(\boldsymbol{y})\big])\big] \\
&= \mathbb{E}_{\boldsymbol{y}|\theta}\big[\Lambda(\boldsymbol{y}, \theta)\hat{\theta}(\boldsymbol{y})\big] - \mathbb{E}_{\boldsymbol{y}|\theta}[\Lambda(\boldsymbol{y}, \theta)]\mathbb{E}_{\boldsymbol{y}|\theta}\big[\hat{\theta}(\boldsymbol{y})\big] \\
&= \mathbb{E}_{\boldsymbol{y}|\theta}\big[\Lambda(\boldsymbol{y}, \theta)\hat{\theta}(\boldsymbol{y})\big] \\
&= \int p(\boldsymbol{y} \mid \theta) \Lambda(\boldsymbol{y}, \theta) \hat{\theta}(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \\
&= \int \frac{\partial}{\partial \theta} p(\boldsymbol{y} \mid \theta) \hat{\theta}(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \\
&= \frac{\partial}{\partial \theta} \int p(\boldsymbol{y} \mid \theta) \hat{\theta}(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \\
&= \frac{\partial}{\partial \theta} \Big(\mathbb{E}_{\boldsymbol{y}|\theta}[\hat{\theta}(\boldsymbol{y})] - \theta \Big) + 1 \\
&= \frac{\partial}{\partial \theta} b_{\hat{\theta}} + 1.
\end{aligned}$$

Using the Cauchy-Schwarz inequality, we have the following bound,

$$\begin{aligned}
\mathrm{Cov}_{\boldsymbol{y}|\theta}\big(\Lambda(\boldsymbol{y}, \theta), \hat{\theta}(\boldsymbol{y})\big)^2 &= \Big(\mathbb{E}_{\boldsymbol{y}|\theta}\big[\Lambda(\boldsymbol{y}, \theta)\big(\hat{\theta}(\boldsymbol{y}) - \mathbb{E}\big[\hat{\theta}(\boldsymbol{y})\big]\big)\big] \Big)^2 \\
\left(\frac{\partial}{\partial \theta} b_{\hat{\theta}} + 1 \right)^2 &\leq \mathbb{E}_{\boldsymbol{y}|\theta}\Big[\Lambda(\boldsymbol{y}, \theta)^2\Big] \cdot \mathbb{E}_{\boldsymbol{y}|\theta}\Big[\big(\hat{\theta}(\boldsymbol{y}) - \mathbb{E}\big[\hat{\theta}(\boldsymbol{y})\big]\big)^2\Big] \\
&= \mathcal{I}_n(\theta) \cdot \mathbb{E}_{\boldsymbol{y}|\theta}\Big[\big(\hat{\theta}(\boldsymbol{y}) - \theta - \mathbb{E}\big[\hat{\theta}(\boldsymbol{y}) - \theta\big]\big)^2\Big] \\
&= \mathcal{I}_n(\theta) \cdot \Big(\mathbb{E}_{\boldsymbol{y}|\theta}\Big[\big(\hat{\theta}(\boldsymbol{y}) - \theta\big)^2\Big] + \Big(\mathbb{E}_{\boldsymbol{y}|\theta}\big[\hat{\theta}\big] - \theta\Big)^2 \\
&\qquad - 2\mathbb{E}_{\boldsymbol{y}|\theta}\big[\hat{\theta}(\boldsymbol{y}) - \theta\big]^2 \Big) \\
&\leq \mathcal{I}_n(\theta) \cdot \Big(\mathbb{E}_{\boldsymbol{y}|\theta}\Big[\big(\hat{\theta}(\boldsymbol{y}) - \theta\big)^2\Big] - b_{\hat{\theta}}^2 \Big).
\end{aligned}$$

Re-arranging yields

$$\mathbb{E}_{\boldsymbol{y}|\theta}\Big[\big(\hat{\theta}(\boldsymbol{y}) - \theta\big)^2\Big] \geq \frac{\left(\frac{\partial}{\partial \theta} b_{\hat{\theta}} + 1 \right)^2}{\mathcal{I}_n(\theta)} + b_{\hat{\theta}}^2.$$

■

Note the trade-off between variance and bias. If $\frac{\partial}{\partial\theta} b_{\hat{\theta}} < 0$, then the variance might be smaller than the variance of the best unbiased estimator.

Corollary. Let $\hat{\theta}$ be an unbiased estimator—*i.e.*, $b_{\hat{\theta}} = 0$—then

$$\operatorname{Var}\big[\hat{\theta}(\boldsymbol{y})\big] \geq \frac{1}{\mathcal{I}_n(\theta)}.$$

Lemma 3.2. The maximum likelihood estimator $\hat{\theta}_{\mathrm{ML}}$ is asymptotically efficient,

$$\lim_{n\to\infty} \operatorname{Var}\big[\hat{\theta}_{\mathrm{ML}}\big] = \frac{1}{\mathcal{I}_n(\theta)}.$$

However, the MLE estimator is not necessarily efficient for finite samples.

4 Regression

In regression, we try to estimate a function $f : \mathbb{R}^d \to \mathbb{R}$ that best fits a given dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \mathbb{R}$. *I.e.*, we try to minimize the following loss function,

$$\ell(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \|f(\boldsymbol{x}_i) - y_i\|^2.$$

4.1 Linear regression

In linear regression, we assume the underlying data to be linear with a fixed noise,

$$Y \mid \boldsymbol{X} = \boldsymbol{x} \sim \mathcal{N}\left(\boldsymbol{\beta}_\star^\top \boldsymbol{x}, \sigma^2\right)$$

for some ground truth $\boldsymbol{\beta}_\star$. We parametrize f as a linear function,

$$f(\boldsymbol{x}; \boldsymbol{\beta}) = \boldsymbol{\beta}^\top \boldsymbol{x}.$$

Under these assumptions, the minimizer of the loss function is analytically tractable—the ordinary least squares estimator (OLSE),

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

where $\boldsymbol{X} \in \mathbb{R}^{d \times n}$ is the design matrix with respective outputs $\boldsymbol{y} \in \mathbb{R}^n$.

In practice, it is important to remove outliers, because linear models can be heavily influenced by them. Also, the data should be standardized, such that all features are on the same scale, because differences in scale can make matrix inversion unstable.

TODO: Curse of dimensionality [Sur and Candès, 2019], Figures 4.1 and 4.2.

Further, if features are collinear, then the model might only learn the correlation with the target variable of one of them and discard the other. In addition, some singular values will equal zero. This makes matrix inversion unstable. This can easily be shown by considering the singular value decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$, then

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y} \\
&= \left(\boldsymbol{V}\boldsymbol{D}^\top \boldsymbol{U}^\top \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top\right)^{-1} \boldsymbol{V}\boldsymbol{D}^\top \boldsymbol{U}^\top \boldsymbol{y} \\
&= \left(\boldsymbol{V}\boldsymbol{D}^\top \boldsymbol{D}\boldsymbol{V}^\top\right)^{-1} \boldsymbol{V}\boldsymbol{D}^\top \boldsymbol{U}^\top \boldsymbol{y} \\
&= \boldsymbol{V}\left(\boldsymbol{D}^{-1}\right)^\top \boldsymbol{D}^{-1} \boldsymbol{V}^\top \boldsymbol{V}\boldsymbol{D}^\top \boldsymbol{U}^\top \boldsymbol{y} \\
&= \boldsymbol{V}\left(\boldsymbol{D}^{-1}\right)^\top \boldsymbol{D}^{-1} \boldsymbol{D}^\top \boldsymbol{U}^\top \boldsymbol{y} \\
&= \boldsymbol{V}\boldsymbol{D}^{-1} \boldsymbol{U}^\top \boldsymbol{y}.
\end{aligned}$$

The inversion of $\boldsymbol{D}$ is unstable if the singular values are small. The solution to this is to remove collinear features and/or to add regularization.

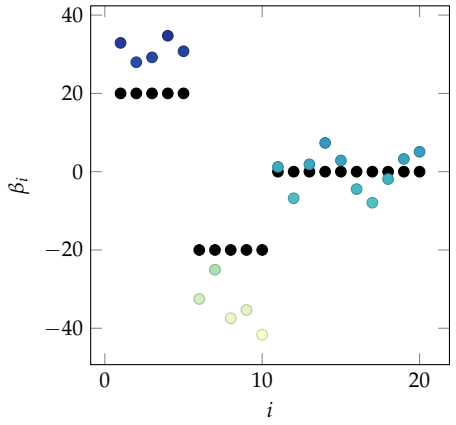

Figure 4.1. $\beta^\star$ is shown as black marks and $\hat{\boldsymbol{\beta}}$ is indicated by the marks. As can be seen, $\hat{\boldsymbol{\beta}}$ is overestimated for indices where $\beta_i^\star \neq 0$.

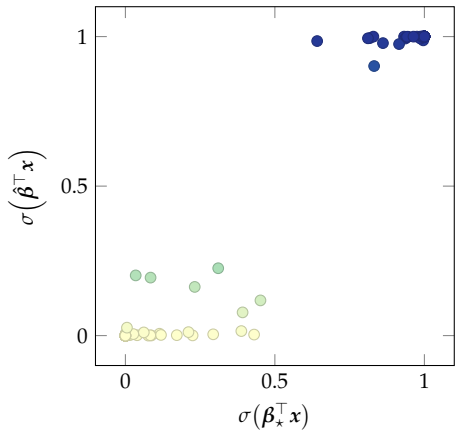

Figure 4.2. True *vs.* predicted probability—the estimator is overconfident in its predictions.

Consider the mean-squared error loss. The risk (expected loss) can be decomposed as follows into bias, variance, and noise terms,

$$\mathbb{E}\Big[(f(\boldsymbol{x})-y)^2\Big] = \underbrace{(\mathbb{E}[f(\boldsymbol{x})]-\mathbb{E}[y])^2}_{\text{squared bias}} + \underbrace{\mathrm{Var}[f(\boldsymbol{x})]}_{\text{variance}} + \underbrace{\mathbb{E}\Big[(\mathbb{E}[y]-y)^2\Big]}_{\text{noise}}.$$

The OLSE is the minimum variance unbiased estimator.[2] However, this does not mean it is the best, because introducing some bias may considerably decrease the variance. Bayesianism adds bias by introducing a prior—priors often induce a regularizing term.[3]

[2] This is proven by the Gauss-Markov theorem.

[3] *E.g.*, a Gaussian prior induces an ℓ_2-norm regularizing term.

4.2 *Polynomial regression*

In polynomial regression, we construct a feature function of all possible polynomials, *e.g.*,

$$\phi([x_1,x_2]) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, \ldots].$$

We then perform linear regression in this space,

$$\psi(\boldsymbol{x};\boldsymbol{\beta}) = \langle \boldsymbol{\beta}, \phi(\boldsymbol{x})\rangle.$$

The problem is that this space is infinitely dimensional, and the inner product is ill-defined in an infinitely dimensional space.[4] We can solve this by transforming the vector such that inner products cannot diverge by introducing a data-dependent scalar,

$$\phi(\boldsymbol{x}) = \alpha(\boldsymbol{x})\cdot\tilde{\phi}(\boldsymbol{x}), \quad \alpha(\boldsymbol{x}) > 0.$$

[4] This is due to $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle = \infty$ for some $\boldsymbol{x}, \boldsymbol{x}'$, *e.g.*, $\boldsymbol{x} = \boldsymbol{x}' = [1,1]$.

$\tilde{\phi}$ contains all polynomials in this equation.

An inner product w.r.t. this feature function is a valid inner product. Such transformations can have a closed form for the inner product, called kernelization. Commonly, the radial basis function (RBF) kernel is used,

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right), \quad \sigma \in \mathbb{R}.$$

Let $\boldsymbol{\Phi} \in \mathbb{R}^{n\times\infty}$ contain all feature vectors, then the OLSE for polynomial regression can be computed by

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y}, \quad \boldsymbol{\Phi} \in \mathbb{R}^{n\times\infty}.$$

However, $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ cannot be computed, because it is $\infty\times\infty$-dimensional. We can fix this by observing that $\boldsymbol{\Phi}\boldsymbol{\Phi}^\top \in \mathbb{R}^{n\times n}$ and rewriting the OLSE as

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} \\
&= \left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\right)\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\right)^{-1}\boldsymbol{y} \\
&= \left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\right)^{-1}\boldsymbol{y} \\
&= \boldsymbol{\Phi}^\top\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\right)^{-1}\boldsymbol{y}.
\end{aligned}$$

Let $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$, then it only contains kernel evaluations—$k_{ij} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$. The next problem is that $\boldsymbol{\Phi}^\top$ is still infinitely dimensional. However, this is not a problem, since when we want to make a prediction, we do the following,

$$\begin{aligned} \psi(\hat{\boldsymbol{x}}) &= \langle \phi(\hat{\boldsymbol{x}}), \hat{\boldsymbol{\beta}} \rangle \\ &= \phi(\hat{\boldsymbol{x}})^\top \boldsymbol{\Phi}^\top \boldsymbol{K}^{-1} \boldsymbol{y}. \end{aligned}$$

Let $\boldsymbol{k}(\hat{\boldsymbol{x}}) = \phi(\hat{\boldsymbol{x}})^\top \boldsymbol{\Phi}^\top$, then it only contains kernel evaluations with the new point—$k_i(\hat{\boldsymbol{x}}) = \langle \phi(\hat{\boldsymbol{x}}), \phi(\boldsymbol{x}_i) \rangle$. In conclusion, we can make predictions by

$$\psi(\hat{\boldsymbol{x}}) = \boldsymbol{k}(\hat{\boldsymbol{x}}) \boldsymbol{K}^{-1} \boldsymbol{y}, \quad \boldsymbol{k}(\hat{\boldsymbol{x}}) \in \mathbb{R}^{1 \times n}, \boldsymbol{K} \in \mathbb{R}^{n \times n}, \boldsymbol{y} \in \mathbb{R}^n.$$

However, the problem with this approach is that it takes $O(n^3)$ to make a prediction—it scales in the amount of data points.

4.3 Regularization

TODO: Ridge regression—ℓ_2-norm.

TODO: LASSO—ℓ_1-norm induces sparsity. Better interpretability properties.

TODO: Figure showing coefficient weights under different regularization factors.

5 Causality

In general, models do not distinguish between causal and non-causal factors in the feature space. Therefore, they might identify non-causal factors as highly correlating with the output variable. *E.g.*, when classifying images as either depicting a cow or a camel, the model might identify the background as an important feature, because cows are usually in grass and camels are usually in the desert. However, if this classifier were to see a cow with a different background, it would fail.

The following are examples of causal fallacies, where one might conclude that X causes Y,

- Reverse causality, where Y actually predicts X and not the other way around;
- Third-cause fallacy, where there is a confounding factor Z that influences both X and Y;
- Bidirectional causation, where X influence each other;

A domain shift happens when the test samples are drawn from a different distribution than the training samples. *E.g.*, COVID-19 detection models trained on a western population might not perform well on an eastern population.

Shortcut learning happens when there is a spurious correlation between causal and non-causal features in the training dataset. The resulting estimator abuses the non-causal features to generalize in the testing dataset. The solution to this is to encode the features such that they do not depend on the environment.

5.1 Counterfactual invariance

Let $\boldsymbol{X}$ be the feature vector representing the object and let Y be a target variable of interest, and let f be the function that estimates Y from $\boldsymbol{X}$. Further, let $\boldsymbol{W}$ describe features that influence $\boldsymbol{X}$, but should not influence the estimator f. Let a counterfactual be denoted as $\boldsymbol{X}(\boldsymbol{w})$, which is the feature vector we would have obtained if we would have had $\boldsymbol{W} = \boldsymbol{w}$, after the fact. Then, the estimator f is counterfactually invariant if

$$f(\boldsymbol{X}(\boldsymbol{w})) = f(\boldsymbol{X}(\boldsymbol{w}')), \quad \forall \boldsymbol{w}, \boldsymbol{w}' \in \text{range}(\boldsymbol{W}).$$

In words, f is not influenced by the value of $\boldsymbol{W}$. One way of obtaining a counterfactually invariant estimator is by extending the training dataset to contain enough counterfactuals. This can, for example, be achieved by data augmentation. However, this is not always possible.

Let $\boldsymbol{X}_{\boldsymbol{A}}$ be the parts of $\boldsymbol{X}$ causally influenced by $\boldsymbol{A}$, and let $\boldsymbol{A}^{\perp}$ be the set of variables independent of $\boldsymbol{A}$. Then, f is counterfactually invariant if and only if f only depends on $\boldsymbol{X}_{\boldsymbol{W}^{\perp}}$.

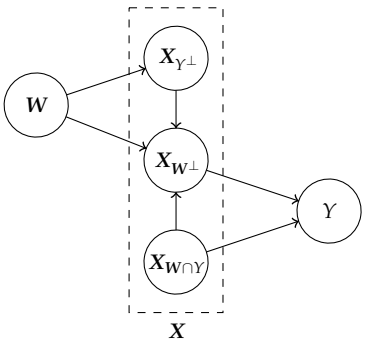

Figure 5.1. Causal graph.

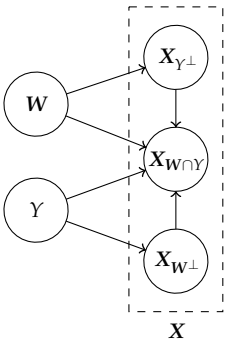

Figure 5.2. Anti-causal graph.

Theorem 5.1 (Necessary conditions for counterfactual invariance [Veitch et al., 2021])**.** If f is a counterfactually invariant predictor, then

- In the anti-causal scenario, $f(\boldsymbol{X}) \perp \boldsymbol{W} \mid Y$;
- In the causal scenario without selection (but possibly confounded), $f(\boldsymbol{X}) \perp \boldsymbol{W}$;
- In the causal scenario without confoundedness (but possibly with selection), $f(\boldsymbol{X}) \perp \boldsymbol{W} \mid Y$, as long as $\boldsymbol{X} \perp Y \mid \boldsymbol{X}_{\boldsymbol{W}^{\perp}}, \boldsymbol{W}$.

Proof. This can be proven by using d-separation. ■

6 Gaussian processes

Let the inputs be $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, the outputs be $\boldsymbol{y} \in \mathbb{R}^n$, and the weight matrix be $\boldsymbol{\beta} \in \mathbb{R}^d$, then linear regression models the generative process as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\Big(\mathbf{0}, \sigma^2 \boldsymbol{I}\Big).$$

This is equivalent to defining a Gaussian over the predictions,

$$\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\beta} \sim \mathcal{N}\Big(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 \boldsymbol{I}\Big).$$

BLR (***B**ayesian **L**inear **R**egression*) extends linear regression by defining a prior over the regression coefficients,

$$\boldsymbol{\beta} \sim \mathcal{N}\Big(\mathbf{0}, \boldsymbol{\Lambda}^{-1}\Big), \quad \boldsymbol{\Lambda} \in \mathbb{R}^{d \times d}.$$

The posterior can be analytically computed as

$$\boldsymbol{\beta} \mid \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where

$$\boldsymbol{\mu} = \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{X}^\top \boldsymbol{y}, \quad \boldsymbol{\Sigma} = \sigma^2\Big(\boldsymbol{X}^\top \boldsymbol{X} + \sigma^2 \boldsymbol{\Lambda}\Big)^{-1}.$$

Proof.

$$\begin{aligned}
p(\boldsymbol{\beta} \mid \boldsymbol{X}, \boldsymbol{y}) &\propto p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\beta}) \cdot p(\boldsymbol{\beta}) && \text{Bayes' rule.} \\
&= \mathcal{N}\Big(\boldsymbol{y}; \boldsymbol{X}\boldsymbol{\beta}, \sigma^2 \boldsymbol{I}\Big) \cdot \mathcal{N}\Big(\boldsymbol{\beta}; \mathbf{0}, \boldsymbol{\Lambda}^{-1}\Big) \\
&\propto \exp\bigg(-\frac{1}{2}\Big(\frac{1}{\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \boldsymbol{\beta}^\top \boldsymbol{\Lambda}\boldsymbol{\beta}\Big)\bigg) \\
&= \exp\bigg(-\frac{1}{2}\Big(\frac{1}{\sigma^2}\Big(\|\boldsymbol{y}\|^2 + \|\boldsymbol{X}\boldsymbol{\beta}\|^2 - 2\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{\beta}\Big) + \boldsymbol{\beta}^\top \boldsymbol{\Lambda}\boldsymbol{\beta}\Big)\bigg) && \text{Cosine theorem.} \\
&= \exp\bigg(-\frac{1}{2}\Big(\frac{1}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{y} + \frac{1}{\sigma^2}\boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} - \frac{2}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\beta}^\top \boldsymbol{\Lambda}\boldsymbol{\beta}\Big)\bigg) \\
&= \exp\bigg(-\frac{1}{2}\Big(\boldsymbol{\beta}^\top\Big(\frac{1}{\sigma^2}\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{\Lambda}\Big)\boldsymbol{\beta} - \frac{2}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{\beta} + \frac{1}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{y}\Big)\bigg) \\
&= \exp\bigg(-\frac{1}{2}\Big(\boldsymbol{\beta}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\beta} - \frac{1}{\sigma^2}\boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{y} - \frac{1}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{\beta} + \frac{1}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{y}\Big)\bigg) \\
&= \exp\bigg(-\frac{1}{2}\Big(\boldsymbol{\beta}^\top \boldsymbol{\Sigma}^{-1}\Big(\boldsymbol{\beta} - \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{X}^\top \boldsymbol{y}\Big) - \frac{1}{\sigma^2}\boldsymbol{y}^\top \boldsymbol{X}\Big(\boldsymbol{\beta} - \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{X}^\top \boldsymbol{y}\Big)\Big)\bigg) \\
&= \exp\bigg(-\frac{1}{2}\Big(\Big(\boldsymbol{\beta} - \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{X}^\top \boldsymbol{y}\Big)\boldsymbol{\Sigma}^{-1}\Big(\boldsymbol{\beta} - \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{X}^\top \boldsymbol{y}\Big)\Big)\bigg) \\
&\propto \mathcal{N}\Big(\boldsymbol{\beta}; \frac{1}{\sigma^2}\boldsymbol{\Sigma}\boldsymbol{X}^\top \boldsymbol{y}, \boldsymbol{\Sigma}\Big).
\end{aligned}$$

■

In general, we do not know the true weights that generated the data points. But, we can now define a joint distribution over output variables with unknown weights,

$$\boldsymbol{y} \mid \boldsymbol{X} \sim \mathcal{N}\Big(\mathbf{0}, \boldsymbol{X}\boldsymbol{\Lambda}^{-1}\boldsymbol{X}^\top + \sigma^2 \boldsymbol{I}\Big).$$

Proof. As we saw earlier, we compute outputs as follows,

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\Big(\mathbf{0}, \sigma^2 \boldsymbol{I}\Big).$$

However, now the weights are unknown, so we make use of its prior to compute the distribution over $\boldsymbol{y}$,

$$\begin{aligned}
\mathbb{E}[\boldsymbol{y}] &= \mathbb{E}[\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}] \\
&= \boldsymbol{X}\mathbb{E}[\boldsymbol{\beta}] + \mathbb{E}[\boldsymbol{\epsilon}] \\
&= \mathbf{0}. \\
\mathrm{Cov}[\boldsymbol{y}] &= \mathbb{E}\Big[(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon})^\top\Big] \\
&= \mathbb{E}\Big[\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^\top\boldsymbol{X}^\top\Big] + \mathbb{E}\Big[\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\epsilon}^\top\Big] + \mathbb{E}\Big[\boldsymbol{\epsilon}\boldsymbol{\beta}^\top\boldsymbol{X}^\top\Big] + \mathbb{E}\Big[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\Big] \\
&= \boldsymbol{X}\mathbb{E}\Big[\boldsymbol{\beta}\boldsymbol{\beta}^\top\Big]\boldsymbol{X}^\top + \boldsymbol{X}\mathbb{E}[\boldsymbol{\beta}]\mathbb{E}\Big[\boldsymbol{\epsilon}^\top\Big] + \mathbb{E}[\boldsymbol{\epsilon}]\mathbb{E}\Big[\boldsymbol{\beta}^\top\Big]\boldsymbol{X}^\top + \sigma^2\boldsymbol{I} && \boldsymbol{\beta} \text{ and } \boldsymbol{\epsilon} \text{ are independent.} \\
&= \boldsymbol{X}\boldsymbol{\Lambda}^{-1}\boldsymbol{X}^\top + \sigma^2\boldsymbol{I}.
\end{aligned}$$

■

GPs (***G**aussian **P**rocesses*) generalize BLR by observing that we can kernelize the covariance matrix,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{\Lambda}^{-1} \boldsymbol{x}_j.$$

We could instead use any other kernel function to model other functions.

6.1 Kernels

Kernel functions specify the similarity between any two data points. They encode assumptions about the function that is to be learned.

Definition 6.1 (Kernel function)**.** A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ over $\mathcal{X} \subset \mathbb{R}$ must satisfy the following properties,

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x}) \qquad \text{Symmetry.}$$

$$\int k(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}) f(\boldsymbol{x}') \mathrm{d}\boldsymbol{x} \mathrm{d}\boldsymbol{x}' \geq 0, \quad \forall f \in L_2. \qquad \text{Positive semi-definiteness.}$$

Definition 6.2 (Stationary and isotropic)**.** A kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ is stationary if it only depends on $\boldsymbol{x} - \boldsymbol{x}'$. Further, it is isotropic if it only depends on $\|\boldsymbol{x} - \boldsymbol{x}'\|_2$.

The following are common kernels,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}' \qquad \text{Linear kernel.}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \Big(\boldsymbol{x}^\top \boldsymbol{x}' + 1\Big)^p, \qquad p \in \mathbb{N} \qquad \text{Polynomial kernel.}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{\ell^2}\right), \qquad \ell \in \mathbb{R} \qquad \text{RBF (***R**adial **B**asis **F**unction*) kernel.}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh\Big(\kappa \boldsymbol{x}^\top \boldsymbol{x}'\Big) - b, \qquad \kappa, b \in \mathbb{R} \qquad \text{Sigmoid kernel.}$$

Different kernels have different invariance properties, such as invariance to rotation or translation. In order to learn invariances from data, many samples are needed. So, it might be better to encode them if we know them a priori.

Given two kernel functions k_1, k_2 defined on the same data space and positive scalar $c > 0$, the following are also valid kernels,

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{x}') &= k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}') \\ k(\boldsymbol{x}, \boldsymbol{x}') &= k_1(\boldsymbol{x}, \boldsymbol{x}') \cdot k_2(\boldsymbol{x}, \boldsymbol{x}') \\ k(\boldsymbol{x}, \boldsymbol{x}') &= c \cdot k_1(\boldsymbol{x}, \boldsymbol{x}') \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \exp(k_1(\boldsymbol{x}, \boldsymbol{x}')). \end{aligned}$$

In practice, the kernels are often composed together and hyperparameters are determined by performance on a held out validation dataset.

6.2 *Prediction*

As we saw earlier in the case of BLR, the marginal is jointly Gaussian,

$$\begin{bmatrix} \boldsymbol{y} \\ y^\star \end{bmatrix} \mid \boldsymbol{X}, \boldsymbol{x}^\star \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K} + \sigma^2 \boldsymbol{I} & \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}^\star) \\ \boldsymbol{k}(\boldsymbol{x}^\star, \boldsymbol{X}) & k(\boldsymbol{x}^\star, \boldsymbol{x}^\star) \end{bmatrix}\right).$$

Theorem 6.3 (Conditional Gaussian distribution)**.** Given

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}\right).$$

Then, the conditional Gaussian is given by

$$\boldsymbol{x}_2 \mid \boldsymbol{x}_1 = \boldsymbol{z} \sim \mathcal{N}\Big(\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{z} - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\Big).$$

Using the above theorem, we can easily compute the predictive distribution,

$$y^\star \mid \boldsymbol{x}^\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}\Big(\boldsymbol{k}^\top \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}, c - \boldsymbol{k}^\top \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}\Big),$$

where

$$\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}), \quad \boldsymbol{k} = \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}^\star), \quad c = k(\boldsymbol{x}^\star, \boldsymbol{x}^\star).$$

Now, we can compute a prediction along with its uncertainty. However, the problem with this approach is that it has $\mathcal{O}(n^3)$ runtime.

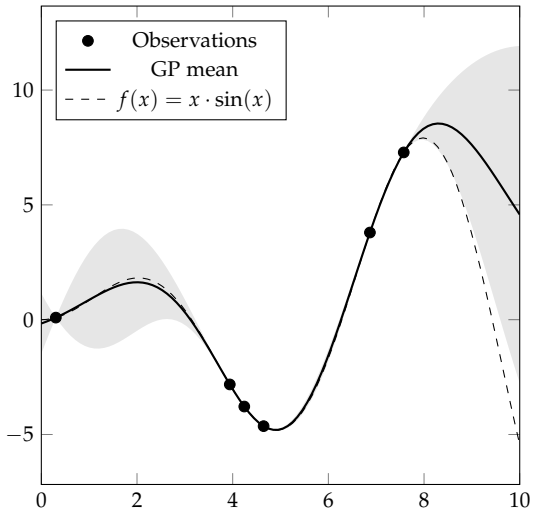

Figure 6.1. Fitted Gaussian process (RBF kernel, $\ell = 1.86$) with its 95% confidence interval.

7 Uncertainty quantification

7.1 Statistical model validation

In statistical modeling, we typically want to predict a target random variable $Y \in \mathcal{Y}$ given a random vector $\boldsymbol{X} \in \mathcal{X}$. Formally, we want to find a function $f : \mathcal{X} \to \mathcal{Y}$ that minimizes the expected risk,

$$\mathcal{R}(f) \doteq \mathbb{E}_{\boldsymbol{X},Y}[\mathbb{1}\{f(\boldsymbol{X}) \neq Y\}].$$

However, this function is intractable, because (1) we do not have access to the joint distribution over $\boldsymbol{X}, Y$, (2) it is intractable to find f without any assumptions on its structure, and (3) it is unclear how to minimize the expectation of the indicator function. To resolve (1), we obtain a dataset of samples from the joint distribution. Problem (2) can be resolved by restricting the functions to a parameterized hypothesis space $\mathcal{H}$. Lastly, the solution to (3) is to approximate the indicator function by a differentiable loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. As a result, given a dataset $\mathcal{Z} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, we can approximate the expected risk by the empirical risk,

$$\hat{\mathcal{R}}(f) \doteq \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i)).$$

We denote the minimizer of this function by $\hat{f}$.

Since $\hat{f}$ is obtained by training on the dataset $\mathcal{Z}$, we need to find a way to measure the performance of $\hat{f}$ on unseen data points—we want to estimate $\mathcal{R}(\hat{f})$. Furthermore, let $\mathcal{A} : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \to \mathcal{H}$ be the algorithm that maps datasets to functions, then we want to estimate its expected risk,

$$\mathcal{R}(\mathcal{A}) \doteq \mathbb{E}_{\mathcal{Z}}[\mathcal{R}(\mathcal{A}(\mathcal{Z}))].$$

Cross-validation. Cross-validation works as follows,

1. Partition the data $\mathcal{Z}$ in K equally sized disjoint subsets, such that

$$\mathcal{Z} = \bigcup_{k=1}^{K} \mathcal{Z}_k;$$

2. Use $\mathcal{A}$ to produce K estimators $\hat{f}^{-k}$ from $\mathcal{Z} - \mathcal{Z}_k$ for all $k \in [K]$;

3. Estimate the expected risk by

$$\mathcal{R}^{\mathrm{CV}}(\mathcal{A}) \doteq \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, \hat{f}^{-(k(i))}(\boldsymbol{x}_i)\Big),$$

   where $k : [n] \to [K]$ maps i to the partition such that $\boldsymbol{x}_i \in \mathcal{Z}_{k(i)}$.

In conclusion, the loss terms in the expected risk depend on approximators that have not seen the data it is evaluating.

Bootstrap. The bootstrap method is used for measuring the distribution over statistical parameters. It does so by creating B bootstrap samples by sampling with replacement from the dataset. For each bootstrap sample, the parameters are computed. Using these parameters, we can estimate the variability and uncertainty of the parameter. Note that this works for any type of parameter that can be estimated from the dataset—not only model parameters. Let $S : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}^d$ be a function that processes datasets and outputs parameters, then we can compute the statistics of this parameter as follows,

1. Draw B samples $\mathcal{Z}^{*b}$ of size n from $\mathcal{Z}$ with replacement;
2. Compute an estimate $S(\mathcal{Z}^{*b})$ for each bootstrap sample.

Now, we can use this set of parameter estimates to compute its statistics. For example, we can compute its mean and covariance,

$$\boldsymbol{\mu}(S) = \frac{1}{B}\sum_{b=1}^{B} S\Big(\mathcal{Z}^{*b}\Big)$$
$$\boldsymbol{\Sigma}(S) = \frac{1}{B-1}\sum_{b=1}^{B}\Big(S\Big(\mathcal{Z}^{*b}\Big) - \mu(S)\Big)\Big(S\Big(\mathcal{Z}^{*b}\Big) - \boldsymbol{\mu}(S)\Big)^{\top}.$$

The algorithm $\mathcal{A}$ is a function that processes a dataset and outputs an estimator, which is used to compute a loss term. We can use this to estimate the empirical risk across n data points,

$$\hat{\mathcal{R}}^{\text{BS}}(\mathcal{A}) \doteq \frac{1}{n \cdot B}\sum_{b=1}^{B}\sum_{i=1}^{n} \ell\Big(y_i, f^{*b}(\boldsymbol{x}_i)\Big).$$

However, computing this validation metric includes data that the bootstrap estimates were trained on—the estimate is overly optimistic. Thus, we can use the bootstrap method to estimate the expected risk $\mathcal{R}(\mathcal{A})$ as follows,

$$\mathcal{R}^{\text{BS}}(\mathcal{A}) \doteq \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathcal{C}^{-i}|}\sum_{b\in\mathcal{C}^{-i}} \ell\Big(y_i, \hat{f}^{*b}(\boldsymbol{x}_i)\Big),$$

where $\mathcal{C}^{-i}$ contains the indices of the bootstraps that do not contain observation $(\boldsymbol{x}_i, y_i)$.

In order to correct for the optimism of $\hat{\mathcal{R}}(\mathcal{A})^{\text{BS}}$, we combine it with $\mathcal{R}^{\text{BS}}$,

$$\mathcal{R}^{(0.632)} \doteq 0.368\hat{\mathcal{R}}^{\text{BS}} + 0.632\mathcal{R}^{\text{BS}}.$$

Here, a weight of 0.632 is used because that is the probability that a sample $(\boldsymbol{x}_i, y_i)$ appears at least once in a bootstrap sample of size n,

$$1 - \left(1 - \frac{1}{n}\right)^{n} \to 1 - \frac{1}{e} = 0.632.$$

TODO: Figure out why we are weighting it, because the optimism is already removed in $\mathcal{R}^{\text{BS}}$.

7.2 *Uncertainty in linear models*

Suppose we have n observations in our dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{y} \in \mathbb{R}^n$ be the design matrix and output vector. We further assume

$$\boldsymbol{y} \mid \boldsymbol{X} \sim \mathcal{N}\Big(\boldsymbol{X}\boldsymbol{\beta}^\star, \sigma^2 \boldsymbol{I}\Big).$$

As we have seen before, the ordinary least squares estimator is computed as follows,

$$\hat{\boldsymbol{\beta}} = \Big(\boldsymbol{X}^\top \boldsymbol{X}\Big)^{-1} \boldsymbol{X}^\top \boldsymbol{y}.$$

Thus, we have the following distribution over estimators,

$$\hat{\boldsymbol{\beta}} \sim \mathcal{N}\left(\boldsymbol{\beta}^\star, \sigma^2 \Big(\boldsymbol{X}^\top \boldsymbol{X}\Big)^{-1}\right).$$

An unbiased estimator of σ^2 is

$$\hat{\sigma}^2 = \frac{1}{n-d} \sum_{i=1}^n \Big(\hat{\boldsymbol{\beta}}^\top \boldsymbol{x}_i - y_i\Big)^2.$$

Thus, we can approximate an $1 - \alpha$ confidence interval for $\beta_j^\star$ by

$$\hat{\beta}_j \pm z_{\alpha/2} \hat{\epsilon}\big(\hat{\beta}_j\big),$$

where $z_{\alpha/2} = \Phi^{-1}(\alpha/2)$, Φ is the CDF of the standard Gaussian, and $\epsilon(\hat{\beta}_j)$ is the j-th diagonal element of $\sigma^2 \big(\boldsymbol{X}^\top \boldsymbol{X}\big)^{-1}$

TODO: Find out how the $1 - \alpha$ confidence interval of $\beta_j^\star$ is computed.

7.3 *Statistical testing*

Assume we have a hypothesis set $\mathcal{H} = \{p(\cdot \mid \boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}$ and let $\boldsymbol{\theta}^\star \in \Theta$ be the (unknown) true parameter. Furthermore, we are given a null hypothesis $H_0 : \boldsymbol{\theta}^\star \in \Theta_0$ and an alternative hypothesis $H_1 : \boldsymbol{\theta}^\star \in \Theta_1$ for predefined $\Theta_0, \Theta_1 \subseteq \Theta$. Lastly, we are given n samples $x_1, \ldots, x_n \sim p(\cdot \mid \boldsymbol{\theta}^\star)$ and a test statistic $t : \mathcal{X}^n \to \mathbb{R}$. The goal is to find a critical value $c \in \mathbb{R}$ for the test t such that

$$\mathbb{P}(t(X_1, \ldots, X_n) \geq c \mid \boldsymbol{\theta})$$

is "low" when $\boldsymbol{\theta} \in \Theta_0$ and "high" when $\boldsymbol{\theta} \in \Theta_1$. So, we accept the null hypothesis when the probability of the test statistic being greater than c is low and we reject it if it is high.

We want to minimize the probability of incorrectly choosing H_1 as this is worse than incorrectly choosing H_0. We can quantify this notion of risk as

$$\alpha_c \doteq \sup_{\boldsymbol{\theta} \in \Theta_0} \mathbb{P}(t(x_1, \ldots, x_n) \geq c \mid \boldsymbol{\theta}).$$

Intuitively, this is the maximum probability of incorrectly choosing H_1, because the test is passed, but the null hypothesis holds. Note that $\alpha_c \to 0$

as $c \to \infty$. So, $c^\star \to \infty$ minimizes the risk. But, this comes with the problem that we would never accept H_1.

The solution to this problem is to forget about choosing the optimal critical value a priori and running an experiment to obtain a realization $x_1, \dots, x_n$. We then run the test with the realization $t(x_1, \dots, x_n)$ and compute the risk of the least risky critical value that would incorrectly reject H_0,

$$p = \inf_{c \in \mathbb{R}} \{\alpha_c \mid t(x_1, \dots, x_n) \geq c\}.$$

One can show that this is the common p-value,

$$p \doteq \sup_{\boldsymbol{\theta} \in \Theta_0} \mathbb{P}(t(X_1, \dots, X_n) \geq t(x_1, \dots, x_n) \mid \boldsymbol{\theta}).$$

Intuitively, this is the probability that the test is higher than our current observation if we run the experiment again.

Wald test. The Wald test is an example of a statistical test. Let $\hat{\theta}$ be an estimator of θ with standard deviation $\hat{\sigma}$ that is asymptotically normal. Let the null hypothesis H_0 be $\theta = \theta_0$ and alternative hypothesis H_1 be $\theta \neq \theta_0$, then the Wald test statistic is

$$W = \frac{(\hat{\theta} - \theta_0)^2}{\hat{\sigma}^2}.$$

TODO: Figure out what the point of this is in the context of machine learning. Also, figure out how this can be done practically, because I have no idea how one would compute α_c or p.

7.4 Bayesian neural networks

As we have seen, BLR incorporates Bayesian uncertainty into linear regression. In a similar fashion, BNNs (***B**ayesian **N**eural **N**etworks*) incorporate Bayesian inference into neural networks.

Neural networks. We define a neural network as having L alternating linear layers and element-wise activation functions,

$$\begin{aligned} \boldsymbol{z}_0 &= \boldsymbol{x} \\ \boldsymbol{z}_\ell &= \phi(\boldsymbol{a}_\ell) \\ \boldsymbol{a}_\ell &= \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1} + \boldsymbol{b}_\ell, \end{aligned}$$

where ϕ is a differentiable activation function and $\boldsymbol{a}$ is its pre-activation. The output of the neural network is then

$$f(\boldsymbol{x} \mid \boldsymbol{\theta}) = \boldsymbol{z}_L, \quad \boldsymbol{\theta} = \{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=1}^L.$$

In practice, the weights of the neural network are chosen to minimize the empirical risk,

$$\hat{\boldsymbol{\theta}} \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \hat{\mathcal{R}}(f(\cdot \mid \boldsymbol{\theta})) \doteq \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(\boldsymbol{x}_i \mid \boldsymbol{\theta})).$$

In general, the weights are optimized by gradient descent,

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{\nabla}_{\boldsymbol{\theta}} \hat{\mathcal{R}}(f(\cdot \mid \boldsymbol{\theta}_{t-1})).$$

TODO: Justify stochastic gradient descent with Robbins-Monro algorithm.

For large datasets, it is too expensive to compute the gradient, so we instead use SGD (***S**tochastic **G**radient **D**escent*), which replaces the gradient by an unbiased estimate,

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{\nabla}_{\boldsymbol{\theta}} \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell(y_i, f(\boldsymbol{x}_i \mid \boldsymbol{\theta}_{t-1})) \right), \quad \mathcal{B} \subseteq [n],$$

where $\mathcal{B}$ contains uniformly sampled sample indices.

Bayesian. The disadvantage of (stochastic) gradient descent is that it yields only a single point estimate of the weights. There is no quantification of uncertainty in this estimate, which leads to overconfidence problems that can result in poor generalization in the presence of domain shifts. We will use a Bayesian approach to alleviate this problem.

The Bayesian approach always has three steps—(1) define a prior, (2) define a likelihood, and (3) use Bayes rule to compute the posterior.

We must first define a prior $p(\boldsymbol{\theta})$ over the weights $\boldsymbol{\theta} \in \Theta$,

$$\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}).$$

Then, we can define the likelihood function $p(\mathcal{D} \mid \boldsymbol{\theta})$ that computes the likelihood of the dataset $\mathcal{Z}$ for a given $\boldsymbol{\theta}$,

$$p(\mathcal{Z} \mid \boldsymbol{\theta}) = \prod_{(\boldsymbol{x},y) \in \mathcal{Z}} p(y \mid \boldsymbol{x}, \boldsymbol{\theta}),$$

where $p(y \mid \boldsymbol{x}, \boldsymbol{\theta})$ is a probability computed by the neural network. Lastly, we use Bayes rule to compute the posterior over weights,

$$p(\boldsymbol{\theta} \mid \mathcal{Z}) = \frac{p(\boldsymbol{\theta}) p(\mathcal{Z} \mid \boldsymbol{\theta})}{p(\mathcal{Z})}.$$

The problem is that $p(\mathcal{Z}) = \int p(\boldsymbol{\theta}) p(\mathcal{Z} \mid \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$ is intractable. The solution is VI (***V**ariational **I**nference*)—search a distribution family $\mathcal{Q}$ for the closest distribution to the posterior.[5] We do this by minimizing the KL divergence between the two,

$$q^\star \in \operatorname*{argmin}_{q \in \mathcal{Q}} D_{\mathrm{KL}}(q \| p).$$

[5] $\mathcal{Q}$ is often called the variational family.

In our case, we will search the space of isotropic Gaussians. *I.e.*, we search for some mean vector $\boldsymbol{\mu} \in \mathbb{R}^p$ and standard deviation $\sigma > 0$ to form the following distribution,

$$q = \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I}).$$

Thus, we have the following optimization problem,

$$\boldsymbol{\mu}^\star, \sigma^\star \in \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathbb{R}^p, \sigma > 0} D_{\mathrm{KL}}\Big(\mathcal{N}\Big(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I}\Big) \| p(\boldsymbol{\theta} \mid \mathcal{Z}) \Big).$$

Note that this problem does not have an analytical solution, so we must optimize it with (stochastic) gradient descent. The KL divergence can be rewritten to

$$\begin{aligned} D_{\mathrm{KL}}(q\|p) &\doteq \mathbb{E}_{\boldsymbol{\theta}\sim q}\left[\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \mathcal{Z})}\right] \\ &\propto \mathbb{E}_{\boldsymbol{\theta}\sim q}[\log q(\boldsymbol{\theta}) - \log p(\mathcal{Z} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{\theta})]. \end{aligned}$$

Let

$$F(\boldsymbol{\mu}, \sigma, \boldsymbol{\theta}) \doteq \log q(\boldsymbol{\theta}; \boldsymbol{\mu}, \sigma) - \log p(\mathcal{Z} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}),$$

then we can derive the gradients to be

$$\begin{aligned} \nabla_{\boldsymbol{\mu}} D_{\mathrm{KL}}(q(\boldsymbol{\mu}, \sigma)\|p(\boldsymbol{\theta} \mid \mathcal{Z})) &= \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\mu}, \sigma, \boldsymbol{\theta}) + \nabla_{\boldsymbol{\mu}} F(\boldsymbol{\mu}, \sigma, \boldsymbol{\theta})\right] \\ \nabla_{\sigma} D_{\mathrm{KL}}(q(\boldsymbol{\mu}, \sigma)\|p(\boldsymbol{\theta} \mid \mathcal{Z})) &= \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\mu}, \sigma, \boldsymbol{\theta}) + \nabla_{\sigma} F(\boldsymbol{\mu}, \sigma, \boldsymbol{\theta})\right]. \end{aligned}$$

We can then approximate the gradients by making a single sample Monte Carlo estimate and apply gradient descent to optimize the weights.

We use the reparameterization trick to obtain $\boldsymbol{\theta}$,

$$\boldsymbol{\theta} = \boldsymbol{\mu} + \sigma\boldsymbol{\epsilon}.$$

7.5 *Information-based transductive learning*

We are given a domain $\mathcal{X}$ that contains a safe area $\mathcal{S} \subseteq \mathcal{X}$ and area of interest $\mathcal{A} \subseteq \mathcal{X}$. Furthermore, we have an unknown function $f^\star : \mathcal{X} \to \mathbb{R}$ that we wish to explore within $\mathcal{A}$. For any point in the safe area $\boldsymbol{x} \in \mathcal{S}$, we can query noisy observations,

$$y_{\boldsymbol{x}} = f^\star(\boldsymbol{x}) + \epsilon_{\boldsymbol{x}}, \quad \mathbb{E}[\epsilon_x] = 0.$$

However, this is prohibitively expensive in some way, so we want to minimize the amount of queries. So, for every query we make, we want to maximize the amount of information that we receive. Formally, given $n-1$ previous samples $\mathcal{D}_{n-1} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n-1} \subseteq \mathcal{S} \times \mathbb{R}$, we want to find the next point $\boldsymbol{x}_n$ that will give the most information about f in the area of interest $\mathcal{A}$.

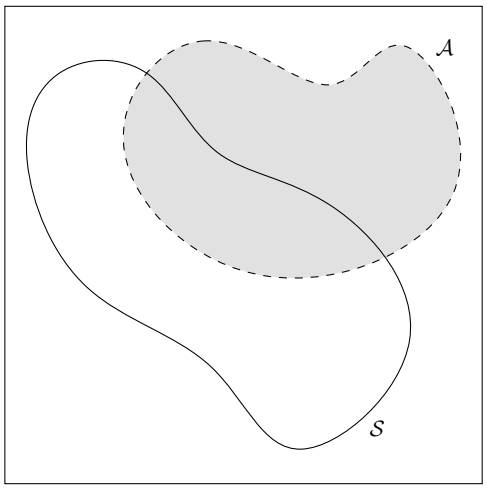

Figure 7.1. Space of active learning.

In ITL (***I**nformation-based **T**ransductive **L**earning*) [Hübotter et al., 2024], the next point is selected as follows,

$$\boldsymbol{x}_n \in \operatorname*{argmax}_{\boldsymbol{x}\in\mathcal{S}} \mathrm{I}(\boldsymbol{f}_{\mathcal{A}}; y_{\boldsymbol{x}} \mid \mathcal{D}_{n-1}).$$

Intuitively, we are looking for a point $\boldsymbol{x} \in \mathcal{S}$ that maximizes the conditional mutual information between $y_{\boldsymbol{x}}$ and f restricted to $\mathcal{A}$. When $f \sim \mathcal{GP}(\mu, k)$ with known mean function μ and kernel k, one can show

$$\mathrm{I}(\boldsymbol{f}_{\mathcal{A}}; y_{\boldsymbol{x}} \mid \mathcal{D}_{n-1}) = \frac{1}{2}\log\left(\frac{\mathrm{Var}[y_{\boldsymbol{x}} \mid \mathcal{D}_{n-1}]}{\mathrm{Var}[y_{\boldsymbol{x}} \mid \boldsymbol{f}_{\mathcal{A}}, \mathcal{D}_{n-1}]}\right).$$

Proof. TODO: Look at exercises. ■

Next, we will look at special cases of ITL.

Safe Bayesian optimization. In safe Bayesian optimization, we want to find the maximum of an unknown function $f^\star$, which is done by iteratively choosing points $\boldsymbol{x}$ and observing their realizations $y = f^\star(\boldsymbol{x}) + \epsilon_{\boldsymbol{x}}$. Furthermore, there is an unknown function $g^\star$ that defines the safe area,

$$\mathcal{S}^\star = \{\boldsymbol{x} \in \mathcal{X} \mid g^\star(\boldsymbol{x}) \geq 0\}.$$

We also observe the realizations of this function $z = g^\star(\boldsymbol{x})$. Now, we want to iteratively select points such that we find the maximum of $f^\star$ while staying within $\mathcal{S}^\star$.

Assume we are given $n-1$ points again $\{(\boldsymbol{x}_i, y_i, z_i)\}_{i=1}^{n-1}$. We then fit a GP f on $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n-1}$, which induces a lower bound function ℓ_n^f and an upper bound function u_n^f, such that $\left[\ell_n^f(\boldsymbol{x}), u_n^f(\boldsymbol{x})\right]$ is the 95% confidence interval of $\mathbb{E}[f(\boldsymbol{x})]$ for any $\boldsymbol{x} \in \mathcal{X}$. Similarly, we fit a GP g on $\{(\boldsymbol{x}_i, z_i)\}_{i=1}^{n-1}$, which analogously produces two functions ℓ_n^g and u_n^g. These induce the pessimistic and optimistic estimates of the safe set,

$$\mathcal{S}_n \doteq \left\{\boldsymbol{x} \;\middle|\; \ell_n^g(\boldsymbol{x}) \geq 0\right\}, \quad \hat{\mathcal{S}}_n \doteq \left\{\boldsymbol{x} \;\middle|\; u_n^g(\boldsymbol{x}) \geq 0\right\}.$$

Further, we define the area of interest,

$$\mathcal{A}_n \doteq \left\{\boldsymbol{x} \in \hat{\mathcal{S}}_n \;\middle|\; u_n^f(\boldsymbol{x}) \geq \max_{\boldsymbol{x}' \in \mathcal{S}_n} \ell_n^f(\boldsymbol{x}')\right\}.$$

Lastly, we can apply ITL using safe set $\mathcal{S}_n$ and area of interest $\mathcal{A}_n$.

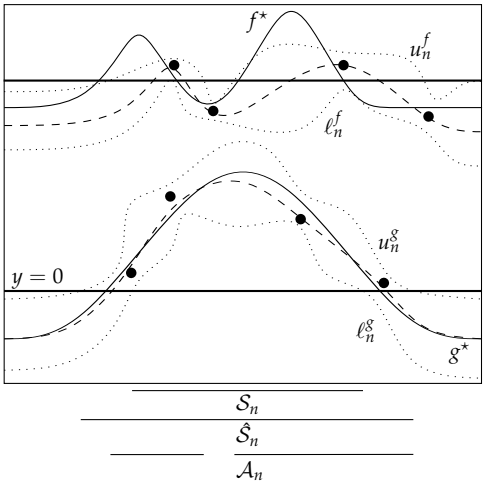

Figure 7.2. Safe Bayesian optimization.

Batch active learning. In batch active learning, we do not select a single point at a time, but we select a batch of points. Formally, we are given an unknown function $f : \mathcal{X} \to \mathcal{Y}$ that we wish to explore, a population set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\} \subseteq \mathcal{X}$, and a budget $b \leq m$. The goal is to find a subset $L \subseteq X$ such that $|L| = b$ for which we query the oracle $\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in L\}$.

We define $B_\delta(\boldsymbol{x})$ as the set of the δ-radius ball of points around $\boldsymbol{x}$,

$$B_\delta(\boldsymbol{x}) \doteq \{\boldsymbol{x}' \in \mathcal{X} \mid \|\boldsymbol{x} - \boldsymbol{x}'\| \leq \delta\}.$$

We call $B_\delta(\boldsymbol{x})$ "pure" if f is constant all over $B_\delta(\boldsymbol{x})$, *i.e.*, every point in a ball around $\boldsymbol{x}$ is classified as the same class. Further, $C(L, \delta)$ is the union of all balls of the chosen subset of points $L \subseteq X$,

$$C(L, \delta) \doteq \bigcup_{\boldsymbol{x} \in L} B_\delta(\boldsymbol{x}).$$

Further, we define

$$C_r(L, \delta) \doteq \{\boldsymbol{x} \in C(L, \delta) \mid \hat{f}(\boldsymbol{x}) = f(\boldsymbol{x})\} \qquad r \text{ stands for "right".}$$
$$C_w(L, \delta) \doteq \{\boldsymbol{x} \in C(L, \delta) \mid \hat{f}(\boldsymbol{x}) \neq f(\boldsymbol{x})\}, \qquad w \text{ stands for "wrong".}$$

where $\hat{f}$ is a classifier of the space $\mathcal{X}$. Note $C(L, \delta) = C_r(L, \delta) \cup C_w(L, \delta)$. Lastly, we define $\pi(\delta)$ to be the total probability density of points that have an impure δ-radius ball,

$$\pi(\delta) \doteq \mathbb{P}(\{\boldsymbol{x} \in \mathcal{X} \mid B_\delta(\boldsymbol{x}) \text{ is not pure}\}).$$

Note that π increases with δ. Suppose we know $\mathcal{Z} = \{(\boldsymbol{x}, f(\boldsymbol{x})) \mid \boldsymbol{x} \in L\}$, then denote the fitted 1-nearest neighbor classifier on this dataset as $\hat{f}$.

We have $C_w(L, \delta) \subseteq \{\boldsymbol{x} \in \mathcal{X} \mid B_\delta(\boldsymbol{x}) \text{ is not pure}\}$. (This can easily be verified using Figures 7.3 and 7.4.) So, we have

$$\mathbb{P}(C_w(L, \delta)) \leq \pi(\delta).$$

Furthermore, looking at Figure 7.3, we can easily see the following

$$\{\boldsymbol{x} \in \mathcal{X} \mid \hat{f}(\boldsymbol{x}) \neq f(\boldsymbol{x})\} \subseteq C_r^c \cup C_w \subseteq C^c \cup \{\boldsymbol{x} \mid B_\delta(\boldsymbol{x}) \text{ is not pure}\},$$

where A^c denotes the complement of a set A. As a result,

$$\begin{aligned}\mathcal{R}(\hat{f}) &\doteq \mathbb{E}[\mathbb{1}\{\hat{f}(\boldsymbol{x}) \neq f(\boldsymbol{x})\}] \\ &= \mathbb{P}(\hat{f}(\boldsymbol{x}) \neq f(\boldsymbol{x})) \\ &\leq 1 - \mathbb{P}(C(L, \delta)) + \pi(\delta).\end{aligned}$$

We want to choose L and δ that minimize $\mathcal{R}(\hat{f})$. This is intractable, so we minimize the upper bound instead. This is done by picking δ first and then choosing L that maximizes $\mathbb{P}(C(L, \delta))$. *I.e.*, we want to solve

$$L^\star \in \operatorname*{argmax}_{L \subseteq X, |L| = b} \mathbb{P}\left(\bigcup_{\boldsymbol{x} \in L} B_\delta(\boldsymbol{x})\right).$$

This has two problems—the distribution of $\boldsymbol{x}$ is unknown and this problem is NP hard. We address the first problem by approximating the distribution with the empirical distribution induced by X. As a result, we get the following optimization problem,

$$L^\star \in \operatorname*{argmax}_{L \subseteq X, |L| = b} \frac{1}{|X|} \left|\{\boldsymbol{x}' \in X \mid \|\boldsymbol{x}' - \boldsymbol{x}\| \leq \delta, \exists \boldsymbol{x} \in L\}\right|.$$

We circumvent the second problem by greedily picking points with the most other points in its ball that are not within the ball of previously picked points.

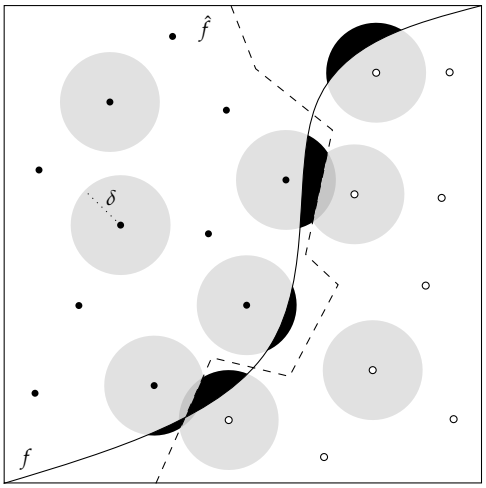

Figure 7.3. Batch active learning. The dashed line is the fitted 1-nearest neighbor classifier on this dataset. Here, $C_w(L, \delta)$ denotes the black area and $C_r(L, \delta)$ denotes the gray area.

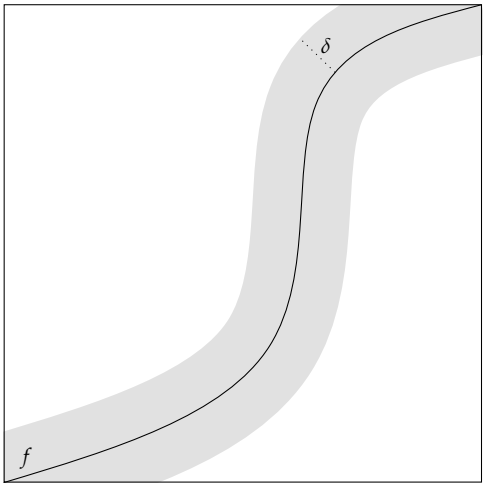

Figure 7.4. $\pi(\delta)$ is the probability density of the marked area, which is the space of impure data points under a given δ.

8 Convex optimization

In standard form, an optimization problem looks as follows,

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) = 0, \quad \forall i \in [n] \\ & h_j(\boldsymbol{x}) \leq 0, \quad \forall j \in [m], \end{aligned}$$

This is called a convex program if f and all h_j are convex and all g_i are affine. We will focus on convex programming. The constraints define a convex set $\mathcal{C} \subseteq \mathcal{X}$, where $\mathcal{X}$ is the intersection of domains of f and all constraints. For our purposes, let $\mathcal{X} = \mathbb{R}^d$. The aim is to compute

$$\operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x}).$$

Definition 8.1 (Lagrangian dual function)**.** For an optimization problem, the Lagrangian $\mathcal{L} : \mathcal{X} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \doteq f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{m} \alpha_j h_j(\boldsymbol{x}).$$

The scalars λ_i and α_j are called Lagrange multipliers. The Lagrange dual function $\delta : \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R} \cup \{-\infty\}$ is defined as

$$\theta(\boldsymbol{\lambda}, \boldsymbol{\alpha}) \doteq \inf_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}).$$

Lemma 8.2 (Weak duality)**.** Let $\boldsymbol{x} \in \mathcal{C}$ (so it satisfies all constraints), $\boldsymbol{\lambda} \in \mathbb{R}^n$, $\boldsymbol{\alpha} \in \mathbb{R}^m$ with $\boldsymbol{\alpha} \geq \mathbf{0}$, and θ be the Lagrange dual function, then

$$\theta(\boldsymbol{\lambda}, \boldsymbol{\alpha}) \leq f(\boldsymbol{x}).$$

The dual Lagrange function lower bounds the primal solution.

Proof. Define $\boldsymbol{x}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\alpha}$ as in Lemma 8.2, then we have

$$\begin{aligned} \theta(\boldsymbol{\lambda}, \boldsymbol{\alpha}) &\doteq \inf_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \\ &\leq \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \\ &= f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{m} \alpha_j h_j(\boldsymbol{x}) \\ &\leq f(\boldsymbol{x}). \end{aligned}$$

We have $g_i(\boldsymbol{x}) = 0$ and $\alpha_j h_j(\boldsymbol{x}) \leq 0$.

This concludes the proof. ∎

As shown in Lemma 8.2, the Lagrange dual function θ lower bounds the primal f for all $\boldsymbol{x} \in \mathcal{C}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\alpha} \geq \mathbf{0}$. We are interested in the maximum

lower bound, which results in the dual program,

$$\begin{aligned} &\text{maximize} && \theta(\boldsymbol{\lambda}, \boldsymbol{\alpha}) \\ &\text{subject to} && \alpha_i \geq 0, \quad \forall i \in [n]. \end{aligned}$$

This is a convex program, even if the primal is not convex.

Corollary.

$$\max_{\boldsymbol{\lambda}, \boldsymbol{\alpha} \geq \mathbf{0}} \theta(\boldsymbol{\lambda}, \boldsymbol{\alpha}) \leq \min_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x}).$$

Lemma 8.3 (Sufficient condition for strong duality)**.** If Slater's condition holds (there exists an $\boldsymbol{x} \in \mathcal{C}$ such that $h_j(\boldsymbol{x}) < 0$ for all $j \in [m]$), then we have strong duality,

$$\max_{\boldsymbol{\lambda}, \boldsymbol{\alpha} \geq \mathbf{0}} \theta(\boldsymbol{\lambda}, \boldsymbol{\alpha}) = \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}).$$

For most problems, a Slater point exists. So, most of the time, strong duality holds. If all g_i and h_j are differentiable, then the Karush-Kuhn-Tucker (KKT) conditions provide necessary conditions for strong duality.

Lemma 8.4 (KKT necessary conditions for strong duality)**.** Let $\boldsymbol{x}^\star$ and $(\boldsymbol{\lambda}^\star, \boldsymbol{\alpha}^\star)$ be feasible solutions to the primal and dual problems. Further, assume that strong duality holds. Then,

$$\alpha_j^\star h_j(\boldsymbol{x}^\star) = 0, \quad \forall j \in [m] \qquad \text{Complementary slackness.}$$

$$\boldsymbol{\nabla}_x \mathcal{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}^\star) = \boldsymbol{\nabla} f(\boldsymbol{x}^\star) + \sum_{i=1}^n \lambda_i^\star \boldsymbol{\nabla} g_i(\boldsymbol{x}^\star) + \sum_{j=1}^m \alpha_j^\star \boldsymbol{\nabla} h_j(\boldsymbol{x}^\star) = \mathbf{0}. \qquad \text{Vanishing Lagrangian gradient.}$$

The second condition can also be written as the following if $\mathcal{L}$ is finite,

$$\boldsymbol{x}^\star \in \operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}^\star).$$

Proof. Define $\boldsymbol{x}^\star$, $\boldsymbol{\lambda}^\star$, and $\boldsymbol{\alpha}^\star$ as in Lemma 8.4, then we get the Master equation,

$$\begin{aligned} f(\boldsymbol{x}^\star) &= \theta(\boldsymbol{\lambda}^\star, \boldsymbol{\alpha}^\star) && \text{Strong duality.} \\ &\doteq \inf_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}^\star) \\ &\leq \mathcal{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}^\star) \\ &\doteq f(\boldsymbol{x}^\star) + \sum_{i=1}^n \lambda_i^\star g_i(\boldsymbol{x}^\star) + \sum_{j=1}^m \alpha_j^\star h_j(\boldsymbol{x}^\star) \\ &\leq f(\boldsymbol{x}^\star). && \text{We have } g_i(\boldsymbol{x}) = 0 \text{ and } \alpha_j h_j(\boldsymbol{x}) \leq 0. \end{aligned}$$

Thus, all inequalities become equalities. The first condition comes from the second-to-last and last line, which results in $\alpha_j^\star h_j(\boldsymbol{x}^\star) = 0$ for all

$j \in [m]$. The second condition comes from the second and third line, where there must be vanishing gradient, because $x^\star$ is the minimizer of $\mathcal{L}(x, \lambda^\star, \alpha^\star)$. ■

Lemma 8.5 (KKT sufficient conditions for strong duality)**.** Let the primal be a convex program. Further, let $x^\star$ and $(\lambda^\star, \alpha^\star)$ be feasible solutions to the primal and dual, respectively. Lastly, let the KKT conditions hold,

$$\begin{aligned} \alpha_j^\star h_j(x^\star) &= 0, \quad \forall j \in [m] \\ \nabla_x \mathcal{L}(x^\star, \lambda^\star, \alpha^\star) &= \mathbf{0}. \end{aligned}$$

Then, strong duality holds.

Proof.

$$\begin{aligned} f(x^\star) &= f(x^\star) + \sum_{i=1}^{n} \lambda_i g_i(x^\star) + \sum_{j=1}^{m} \alpha_j h_j(x^\star) \\ &= \mathcal{L}(x^\star, \lambda^\star, \alpha^\star) \\ &= \inf_{x \in \mathcal{X}} \mathcal{L}(x, \lambda^\star, \alpha^\star) \\ &\doteq \theta(\lambda^\star, \alpha^\star). \end{aligned}$$

$x^\star$ is feasible and we have complementary slackness.

$\mathcal{L}$ is convex, because convexity is preserved under summation and positive scaling. Also, affine functions remain convex under negative scaling. Furthermore, we have vanishing gradient, so $x^\star$ is a global minimizer of $\mathcal{L}(\cdot, \lambda^\star, \alpha^\star)$.

■

8.1 Support vector machine

In SVMs (***S**upport **V**ector **M**achines*), we are given a dataset with binary labels and we want to find the linear separator with the maximum margin. Formally, we are given data $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathbb{R}^d \times \{-1, +1\}$ and we model the linear separator as

$$f[w, b](x) \doteq \begin{cases} +1 & w^\top x + b > 0 \\ -1 & w^\top x + b < 0. \end{cases}$$

We want to model this as a convex program, so we need to get rid of the strict inequality and the cases. We are given the dataset, so we can use information of y to reverse the inequality if necessary. Furthermore, we introduce a small scalar $\epsilon > 0$ to get a non-strict inequality,

$$f[w, b](x, y) = y\Big(w^\top x + b\Big) \geq \epsilon > 0.$$

Now, we have the following optimization problem,

$$\begin{aligned} \text{maximize} \quad & m(w, b) \\ \text{subject to} \quad & y_i\Big(w^\top x_i + b\Big) \geq \epsilon, \quad \forall i \in [n], \end{aligned}$$

where $m : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ is the margin function. The margin function is the minimum distance of any point to the linear separator. Let x^+ and

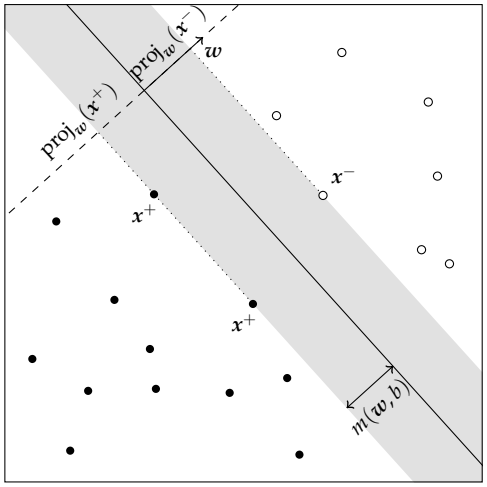

Figure 8.1. Support vector machines.

x^- be the positive and negative support vectors, where a support vector is a point that touches the margin—see Figure 8.1. Then, we have the following definition of the margin,

$$\begin{aligned} 2 \cdot m(\boldsymbol{w}, b) &\doteq \|\text{proj}_{\boldsymbol{w}}(\boldsymbol{x}^+) - \text{proj}_{\boldsymbol{w}}(\boldsymbol{x}^-)\| \\ &= \left\| \frac{\boldsymbol{w}^\top \boldsymbol{x}^+}{\|\boldsymbol{w}\|^2} \boldsymbol{w} - \frac{\boldsymbol{w}^\top \boldsymbol{x}^-}{\|\boldsymbol{w}\|^2} \boldsymbol{w} \right\| \\ &= \frac{\left| \boldsymbol{w}^\top \boldsymbol{x}^+ - \boldsymbol{w}^\top \boldsymbol{x}^- \right|}{\|\boldsymbol{w}\|^2} \|\boldsymbol{w}\| \\ &= \frac{\left| \boldsymbol{w}^\top \boldsymbol{x}^+ - \boldsymbol{w}^\top \boldsymbol{x}^- \right|}{\|\boldsymbol{w}\|} \\ &= \left| \left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right)^\top (\boldsymbol{x}^+ - \boldsymbol{x}^-) \right|, \end{aligned}$$

where $\bar{\boldsymbol{w}}$ is a normalized $\boldsymbol{w}$. Note that the margin does not depend on the norm of $\boldsymbol{w}$ or the intercept b—it only depends on the direction of $\boldsymbol{w}$. As a result, there are an infinite number of solutions, which results in an ill-posed problem. However, only one solution $(\boldsymbol{w}, b)$ will satisfy

$$\boldsymbol{w}^\top \boldsymbol{x}^+ + b = 1, \quad \boldsymbol{w}^\top \boldsymbol{x}^- + b = -1.$$

Under this constraint, the margin can be computed as follows,

$$2 \cdot m(\boldsymbol{w}, b) = \frac{2}{\|\boldsymbol{w}\|},$$

because $|\boldsymbol{w}^\top \boldsymbol{x}^+ - \boldsymbol{w}^\top \boldsymbol{x}^-| = |1 - (-1)| = 2$. Thus, we get the following convex program,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{subject to} \quad & 1 - y_i\Big(\boldsymbol{w}^\top \boldsymbol{x}_i + b\Big) \leq 0, \quad \forall i \in [n]. \end{aligned}$$

Minimizing $1/f(x)$ is equivalent to maximizing $f(\boldsymbol{x})$.

It is easy to see that this convex program has a Slater point. Let $(\boldsymbol{w}, b)$ be a solution to the convex program, so

$$y_i\Big(\boldsymbol{w}^\top \boldsymbol{x}_i + b\Big) \geq 1, \quad \forall i \in [n].$$

Then, $(\gamma \boldsymbol{w}, \gamma b)$ for $\gamma > 1$ is a Slater point, because for all $i \in [n]$, we have

$$y_i\Big(\gamma \boldsymbol{w}^\top \boldsymbol{x}_i + \gamma b\Big) = \gamma y_i\Big(\boldsymbol{w}^\top \boldsymbol{x}_i + b\Big) > y_i\Big(\boldsymbol{w}^\top \boldsymbol{x}_i + b\Big) \geq 1.$$

Hence, strong duality holds and we can use the KKT conditions to find the minimizer $\boldsymbol{x}^\star$.

This convex program has the following Lagrangian,

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^n \alpha_i \Big(1 - y_i\Big(\boldsymbol{w}^\top \boldsymbol{x}_i + b\Big)\Big), \quad \alpha_i \geq 0 \\ &= \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^n \alpha_i - \boldsymbol{w}^\top \sum_{i=1}^n \alpha_i y_i \boldsymbol{x}_i - b \sum_{i=1}^n \alpha_i y_i. \end{aligned}$$

This function has the following gradients,

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$$
$$\frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i.$$

From the vanishing Lagrangian gradient KKT condition, we know

$$\frac{\partial \mathcal{L}(\boldsymbol{w}^\star, b^\star, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \mathbf{0} \implies \boldsymbol{w}^\star = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i.$$

Furthermore, we get the following constraint on α_i,

$$\frac{\partial \mathcal{L}(\boldsymbol{w}^\star, b^\star, \boldsymbol{\alpha})}{\partial b} = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Thus, we have the following dual program,

$$\begin{aligned} \text{maximize} \quad & \theta(\boldsymbol{\alpha}) \doteq \mathcal{L}(\boldsymbol{w}^\star, b^\star, \boldsymbol{\alpha}) \\ \text{subject to} \quad & \alpha_i \geq 0, \quad \forall i \in [n] \\ & \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}$$

We can simplify the Lagrangian by making use of the found $\boldsymbol{w}^\star$,

$$\begin{aligned} \theta(\boldsymbol{\alpha}) &\doteq \mathcal{L}(\boldsymbol{w}^\star, b^\star, \boldsymbol{\alpha}) \\ &= \frac{1}{2}\|\boldsymbol{w}^\star\|^2 + \sum_{i=1}^{n} \alpha_i - \|\boldsymbol{w}^\star\|^2 \\ &= -\frac{1}{2}\|\boldsymbol{w}^\star\|^2 + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j. \end{aligned}$$

Thus, the dual program is a quadratic optimization problem on a simplex, which is easy to optimize. Once we have solved the dual program for $\boldsymbol{\alpha}^\star$, we can use this to compute $\boldsymbol{w}^\star$,

$$\boldsymbol{w}^\star = \sum_{i=1}^{n} \alpha_i^\star y_i \boldsymbol{x}_i.$$

Due to complementary slackness, the following holds,

$$\alpha_i^\star \left(1 - y_i \left(\boldsymbol{w}_\star^\top \boldsymbol{x}_i + b^\star\right)\right) = 0.$$

So, $\alpha_i^\star = 0$ often, which makes $\boldsymbol{\alpha}^\star$ sparse. In fact, $1 - y_i\left(\boldsymbol{w}_\star^\top \boldsymbol{x}_i + b^\star\right) = 0$ only if $\boldsymbol{x}_i$ is a support vector. Thus, the solution $\boldsymbol{w}^\star$ is a sparse linear combination of support vectors.

We can compute intercept by finding support vectors,

$$\boldsymbol{x}^+ \in \operatorname*{argmin}_{\boldsymbol{x}_i : y_i = +1} \boldsymbol{w}^\top \boldsymbol{x}_i, \quad \boldsymbol{x}^- \in \operatorname*{argmax}_{\boldsymbol{x}_i : y_i = -1} \boldsymbol{w}^\top \boldsymbol{x}_i.$$

Further, remember the constraints $\boldsymbol{w}_\star^\top \boldsymbol{x}^+ + b = 1$ and $\boldsymbol{w}_\star^\top \boldsymbol{x}^- + b = -1$, from which we can derive the optimal intercept,

$$b^\star = -\frac{1}{2}\Big(\boldsymbol{w}_\star^\top \boldsymbol{x}^+ + \boldsymbol{w}_\star^\top \boldsymbol{x}^-\Big).$$

8.2 Soft-margin SVM

The SVM only works if the data is linearly separable, but this does not hold in general. We can work around this by introducing a slackness parameter to each data point, which gives the following convex program,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^n \xi_i \\ \text{subject to} \quad & y_i\Big(\boldsymbol{w}^\top \boldsymbol{x}_i + b\Big) \geq 1 - \xi_i, \quad i \in [n] \\ & \xi_i \geq 0, \quad i \in [n]. \end{aligned}$$

Here, we introduced slackness parameters that we wish to be as small as possible with some weight C. This allows for incorrect classifications if it is justified. For example, the slack of an outlier might be very high. The hyperparameter C has an important role in balancing the size of the margin with the number of neglected samples. As C increases, the margin will get more narrow, but fewer points will be neglected.

The optimal slackness parameters can be computed as follows,

$$\xi_i^\star = \max\Big\{0, 1 - y_i\Big(\boldsymbol{w}_\star^\top \boldsymbol{x}_i + b^\star\Big)\Big\}.$$

And, α_i is further constrained to be less than C.

8.3 Kernelization

The data points might be separated by a non-linear separator. In this case, we can use a different feature space by employing a feature function ϕ. This results in the following solution $\boldsymbol{w}^\star$,

$$\boldsymbol{w}^\star = \sum_{i=1}^n \alpha_i^\star y_i \phi(\boldsymbol{x}_i).$$

And, we get the following Lagrangian,

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}^\star, \boldsymbol{\alpha}) &= \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j y_i y_j \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j) \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j). \end{aligned}$$

So, we do not even need the feature function to compute the Lagrangian—we only need a kernel. Furthermore, we can use the kernel for classification (without loss of generalization, assume $b^\star = 0$),

$$\boldsymbol{w}_\star^\top \phi(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i^\star y_i k(\boldsymbol{x}_i, \boldsymbol{x}).$$

8.4 Multi-class SVM

We can generalize the notion of a margin to multi-class problems by introducing weights $\boldsymbol{w}_z$ for each class $z \in [k]$. Then, we define the margin as the maximum $m \in \mathbb{R}$ such that

$$m \leq \left(\boldsymbol{w}_{z_i}^\top \boldsymbol{y}_i + b_{z_i}\right) - \max_{z \neq z_i} \left\{\boldsymbol{w}_z^\top \boldsymbol{y}_i + b_z\right\}, \quad \forall i \in [n].$$

Intuitively, it is the closest distance of any point to a decision boundary. We then define the following optimization problem, which we can solve

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\sum_{z=1}^{k} \|\boldsymbol{w}_z\|^2 \\ \text{subject to} \quad & \left(\boldsymbol{w}_{z_i}^\top \boldsymbol{y}_i + b_{z_i}\right) - \max_{z \neq z_i} \left\{\boldsymbol{w}_z^\top \boldsymbol{y}_i + b_z\right\} \geq 1, \quad \forall i \in [n]. \end{aligned}$$

For data that is not linearly separable, we can again introduce slack,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & \left(\boldsymbol{w}_{z_i}^\top \boldsymbol{y}_i + b_{z_i}\right) - \max_{z \neq z_i} \left\{\boldsymbol{w}_z^\top \boldsymbol{y}_i + b_z\right\} \geq 1 - \xi_i, \quad \forall i \in [n] \\ & \xi_i \geq 0, \quad \forall i \in [n]. \end{aligned}$$

8.5 Structural SVM

SVMs can also be used to predict complex objects by making use of its structure. A naive way of achieving this is to to give each unique structure its own class and doing multi-class SVM. However, this would result in exponentially (or infinitely) many classes, which is intractable. Thus, the structural SVM needs the following things,

1. Compact representation of the output space;
2. Algorithm that allows for efficient prediction—we cannot enumerate every possible structure;
3. Notion of a prediction error—some structures are more similar than others;
4. Efficient training algorithm that has a runtime complexity that is sub-linear in the number of structures.

The first can be solved by defining a joint feature map $\psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ that combines properties of inputs and outputs.[6] Then, we can define the following scoring function,

$$f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{w}^\top \psi(\boldsymbol{x}, \boldsymbol{y}).$$

We use this function to perform classification,

$$c(\boldsymbol{x}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}} f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}).$$

[6] Notice that the dimensionality of this feature map is independent of the number of structures $|\mathcal{Y}|$.

We would need to construct an algorithm to efficiently compute this argmax. In general, we cannot enumerate over all structures, so we would need to make use of the structure of the data to somehow compute this. For example, the output space might be decomposable into independent parts, such that $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_m$.

Moreover, we define the margin as the maximum m that satisfies

$$\boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \max_{\boldsymbol{y} \neq \boldsymbol{y}_i} \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}) \geq m, \quad \forall i \in [n].$$

Thus, we have the following optimization problem,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{subject to} \quad & \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \max_{\boldsymbol{y} \neq \boldsymbol{y}_i} \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}) \geq 1, \quad \forall i \in [n]. \end{aligned}$$

Here, again the maximum would need a specialized algorithm.

To quantify errors, we introduce a loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, where $\Delta(\boldsymbol{y}, \boldsymbol{y}')$ is the loss of predicting $\boldsymbol{y}'$ when the correct output is $\boldsymbol{y}$. Reformulating the optimization problem with slack results in the following,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \max_{\boldsymbol{y} \neq \boldsymbol{y}_i} \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}) \geq \Delta(\boldsymbol{y}, \boldsymbol{y}_i) - \xi_i, \quad \forall i \in [n], \boldsymbol{y} \neq \boldsymbol{y}_i. \end{aligned}$$

Or, equivalently with (significantly) less constraints,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \max_{\boldsymbol{y} \neq \boldsymbol{y}_i} \Big\{ \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}) + \Delta(\boldsymbol{y}, \boldsymbol{y}_i) \Big\} \geq -\xi_i, \quad \forall i \in [n]. \end{aligned}$$

In conclusion, we need to design the following four elements,

1. Joint feature map $\psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$;
2. Loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$;
3. Algorithm during training to compute

$$\tilde{\boldsymbol{y}} \in \operatorname*{argmax}_{\boldsymbol{y} \neq \boldsymbol{y}_i} \Big\{ \boldsymbol{w}^\top \psi(\boldsymbol{x}_i, \boldsymbol{y}) + \Delta(\boldsymbol{y}, \boldsymbol{y}_i) \Big\};$$

4. Inference algorithm to compute

$$\hat{\boldsymbol{y}} \in \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}^\top \psi(\boldsymbol{x}, \boldsymbol{y}).$$

9 Ensembles

The idea of ensembling is simple: create a highly accurate estimator by combining many relatively weak and inaccurate estimators. The advantage over large models is that weak models are easy to train—they only need to be better than random. In general, we have a set of B estimators,

$$\hat{f}_1, \hat{f}_2, \dots, \hat{f}_B.$$

Model averaging is very common in machine learning and data science. For example, the Bayesian approach for inference weights parameters according to their posterior,

$$\int p(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}) p(\boldsymbol{x}^\star \mid \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}.$$

We could take the final estimator to be a convex combination of the estimators,

$$\hat{f}(\boldsymbol{x}) = \sum_{b=1}^{B} p_b \hat{f}_b(\boldsymbol{x}), \quad \sum_{b=1}^{B} p_b = 1, p_b \geq 0.$$

The bias of this estimator is the same convex combination of the biases of the set of estimators,

$$\begin{aligned}
\operatorname{bias}(\hat{f}) &\doteq \mathbb{E}[\hat{f}(X)] - \mathbb{E}[Y \mid X] \\
&= \sum_{b=1}^{B} p_b \mathbb{E}[\hat{f}_b(X)] - \mathbb{E}[Y \mid X] \\
&= \sum_{b=1}^{B} p_b \Big(\mathbb{E}[\hat{f}_b(X)] - \mathbb{E}[Y \mid X] \Big) \\
&= \sum_{b=1}^{B} p_b \operatorname{bias}(\hat{f}_b).
\end{aligned}$$

Thus, unbiased estimators remain unbiased after averaging. We can also look at the variance of this estimator,

$$\begin{aligned}
\operatorname{Var}[\hat{f}] &\doteq \mathbb{E}\left[\Big(\hat{f}(X) - \mathbb{E}[\hat{f}(X)] \Big)^2 \right] \\
&= \mathbb{E}\left[\left(\sum_{b=1}^{B} p_b \Big(\hat{f}_i(X) - \mathbb{E}[\hat{f}_i(X)] \Big) \right)^2 \right] \\
&= \frac{1}{B^2} \sum_{b=1}^{B} \operatorname{Var}[\hat{f}_b] + \frac{1}{B^2} \sum_{b=1}^{B} \sum_{b' \neq b}^{B} \operatorname{Cov}[\hat{f}_b, \hat{f}_{b'}].
\end{aligned}$$

Assume $p_b = 1/B$.

Often, the covariances are small ($\operatorname{Cov}[\hat{f}_b, \hat{f}_{b'}] \approx 0$) and the variances similar ($\operatorname{Var}[\hat{f}_b] \approx \sigma^2$), so we gain from averaging because the variance is reduced,

$$\operatorname{Var}[\hat{f}] \approx \frac{\sigma^2}{B}.$$

We will consider techniques for ensembling classifiers, $c_1, c_2, \dots, c_B$ with weights $\alpha_1, \alpha_2 \dots, \alpha_B$. These estimators are composited as follows,

$$\hat{c}(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{b=1}^{B} \alpha_b c_b(\boldsymbol{x}) \right).$$

It is important to note that this procedure can only succeed if the classifiers are diverse. Otherwise, their covariances are large and we do not gain anything.

1: **for** $b \in [B]$ **do**
2: $\quad \mathcal{Z}^{*b} = \text{BootstrapSample}(\mathcal{Z})$
3: $\quad \hat{c}_b = \text{TrainClassifier}(\mathcal{Z}^{*b})$
4: **end for**
5: **return** $\hat{c}(\boldsymbol{x}) = \text{sgn}\left(\sum_{b=1}^{B} \hat{c}_b(\boldsymbol{x})\right)$

Algorithm 2. Bagging with bootstrapped subsets.

1: **for** $b \in [B]$ **do**
2: $\quad \mathcal{Z}^{*b} = \text{BootstrapSample}(\mathcal{Z})$
3: $\quad$ **repeat**
4: $\quad\quad$ Select $m \approx \sqrt{p}$ features at random
5: $\quad\quad$ Pick the best feature to split
6: $\quad\quad$ Split the node into two daughter nodes
7: $\quad$ **until** node size n_{min} is reached
8: **end for**
9: **return** Set of decision trees $\{\hat{c}_b\}_{b=1}^{B}$

Algorithm 3. Random forest algorithm over a dataset $\mathcal{Z}$, where the feature space is p-dimensional.

9.1 *Bagging*

Bagging generates diversity in the classifier pool by training on different bootstrapped subsets—see Algorithm 2. Bootstrapped subsets are generated by sampling i.i.d. samples from the dataset with replacement.

This works well, because the covariances are small due to using different subsets for training. Furthermore, the variances are similar, because each subsample behaves similarly on average. Finally, the bias is not affected, since the final bias is an average of its estimators' biases.

9.2 *Random forests*

A random forest is a collection of decision trees, which are used for classification. The idea of the random forest algorithm is to grow a sufficiently deep decision tree to reduce bias, which gives a noisy classifier with high variance—see Algorithm 3. The decision trees are grown by repeatedly splitting nodes based on feature values. An example splitting criterium is splitting to reduce entropy. We can then average to enhance the robustness, since the variance is reduced. Classification is finally done by a majority vote of all decision trees.

9.3 *AdaBoost*

AdaBoost [Freund and Schapire, 1995] does not use bootstrapped samples, but re-weights the data at each iteration according to the error of previous weak learners. As a result, the weak estimators should be diverse and hence reduce variance. The diversity is due to later classifiers being trained to classify points that previous classifiers found hard with

more weight. The weighting of the data is defined recursively,

$$w_i^{(b+1)} = w_i^{(b)} \exp(\alpha_b \mathbb{1}\{c_b(\boldsymbol{x}_i) \neq y_i\}), \quad w_i^{(1)} = \frac{1}{n}$$
$$\alpha_b \doteq \log\left(\frac{1-\epsilon_b}{\epsilon_b}\right)$$
$$\epsilon_b \doteq \sum_{i=1}^{n} \frac{w_i^{(b)}}{\sum_{j=1}^{n} w_j^{(b)}} \mathbb{1}\{c_b(\boldsymbol{x}_i) \neq y_i\}.$$

Log-odds of no error *vs.* error.

Probability of an error, where the normalized weights serve as a prior.

We then iteratively fit B classifiers on the full training data using the recursive weights placed on the data points. At inference time, the classifiers are weighted according to their log-odds of error,

$$\hat{c}(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{b=1}^{B} \alpha_b c_b(\boldsymbol{x})\right).$$

Note that AdaBoost is a deterministic algorithm.

Friedman et al. [2000] showed that the minimizer of the exponential loss function[7] w.r.t. the true distribution is the log-odds of the class probabilities. As such, AdaBoost effectively fits an additive model in base learners, optimizing the exponential function.

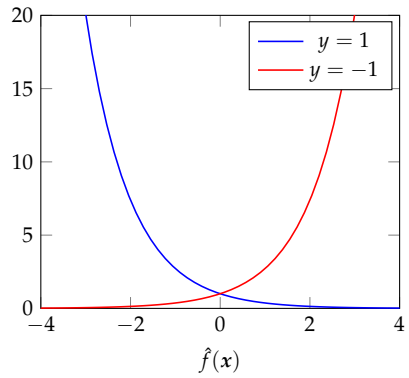

Figure 9.1. Exponential loss function.

[7] The exponential loss function is $\exp(-y\hat{f}(\boldsymbol{x}))$.

Lemma 9.1. The minimizer of $\mathbb{E}[\exp(-yf(\boldsymbol{x}))]$ is

$$f^\star(\boldsymbol{x}) = \frac{1}{2} \log \frac{\mathbb{P}(y=1 \mid \boldsymbol{x})}{\mathbb{P}(y=-1 \mid \boldsymbol{x})}.$$

Thus, we have the following posteriors,

$$\mathbb{P}(y=1 \mid \boldsymbol{x}) = \frac{\exp(f^\star(\boldsymbol{x}))}{\exp(-f^\star(\boldsymbol{x})) + \exp(f^\star(\boldsymbol{x}))}$$
$$\mathbb{P}(y=-1 \mid \boldsymbol{x}) = \frac{\exp(-f^\star(\boldsymbol{x}))}{\exp(-f^\star(\boldsymbol{x})) + \exp(f^\star(\boldsymbol{x}))}$$

Proof. The expected exponential loss function is the following,

$$\mathbb{E}[\exp(-yf(\boldsymbol{x})) \mid \boldsymbol{x}] = \mathbb{P}(y=1 \mid \boldsymbol{x}) \exp(-f(\boldsymbol{x})) + \mathbb{P}(y=-1 \mid \boldsymbol{x}) \exp(f(\boldsymbol{x}))$$

The exponential loss is convex, so the minimizer has vanishing gradient,

$$\frac{\partial \mathbb{E}[\exp(-yf(\boldsymbol{x})) \mid \boldsymbol{x}]}{\partial f(\boldsymbol{x})} = -\mathbb{P}(y=1 \mid \boldsymbol{x}) \exp(-f(\boldsymbol{x})) + \mathbb{P}(y=-1 \mid \boldsymbol{x}) \exp(f(\boldsymbol{x}))$$

Hence, the minimizer $f^\star$ can be derived to be the following (by setting the gradient to zero),

$$\mathbb{P}(y=1 \mid \boldsymbol{x}) \exp(-f^\star(\boldsymbol{x})) = \mathbb{P}(y=-1 \mid \boldsymbol{x}) \exp(f^\star(\boldsymbol{x}))$$
$$\iff \quad \exp(2 \cdot f^\star(\boldsymbol{x})) = \frac{\mathbb{P}(y=1 \mid \boldsymbol{x})}{\mathbb{P}(y=-1 \mid \boldsymbol{x})}$$
$$\iff \quad f^\star(\boldsymbol{x}) = \frac{1}{2} \log \frac{\mathbb{P}(y=1 \mid \boldsymbol{x})}{\mathbb{P}(y=-1 \mid \boldsymbol{x})}.$$

This concludes the proof. ■

Theorem 9.2. The discrete AdaBoost algorithm builds an additive logistic regression model via Newton-like updates for minimizing the exponential loss function, $\mathbb{E}[\exp(-yf(\boldsymbol{x}))]$.

Proof. Let $J(f) = \mathbb{E}[\exp(-yf(\boldsymbol{x}))]$. Suppose that $f(\boldsymbol{x})$ is the current estimate and $f(\boldsymbol{x}) + \alpha c(\boldsymbol{x})$ is an improvement with $c : \mathcal{X} \to \{-1, +1\}$ deciding the direction for a small step α. Then, for a fixed α and $\boldsymbol{x}$, the second-order Taylor expansion around $c(\boldsymbol{x}) = 0$ yields

$$\begin{aligned} J(f + \alpha c) &= \mathbb{E}[\exp(-y(f(\boldsymbol{x}) + \alpha c(\boldsymbol{x})))] \\ &\approx \mathbb{E}\left[\exp(-yf(\boldsymbol{x}))\left(1 - y\alpha c(\boldsymbol{x}) + \frac{1}{2}\alpha^2 y^2 c(\boldsymbol{x})^2\right)\right] && \text{Second-order Taylor approximation around } c(\boldsymbol{x}) = 0. \\ &= \mathbb{E}\left[\exp(-yf(\boldsymbol{x}))\left(1 - y\alpha c(\boldsymbol{x}) + \frac{\alpha^2}{2}\right)\right]. && y^2 = 1 \text{ and } c(\boldsymbol{x})^2 = 1. \end{aligned}$$

Minimizing pointwise w.r.t. $c(\boldsymbol{x}) \in \{-1, 1\}$, we write

$$\begin{aligned} c(\boldsymbol{x}) &= \operatorname*{argmin}_c \mathbb{E}_w\left[1 - y\alpha c(\boldsymbol{x}) + \frac{\alpha^2}{2} \;\middle|\; \boldsymbol{x}\right] \\ &= \operatorname*{argmax}_c \mathbb{E}_w[yc(\boldsymbol{x})]. \end{aligned}$$

where $\mathbb{E}_w[\cdot \mid \boldsymbol{x}]$ is a weighted conditional expectation, where $w(\boldsymbol{x}, y) = \exp(-yf(\boldsymbol{x}))$, and

$$\mathbb{E}_w[g(\boldsymbol{x}, y) \mid \boldsymbol{x}] \doteq \frac{\mathbb{E}[w(\boldsymbol{x}, y)g(\boldsymbol{x}, y) \mid \boldsymbol{x}]}{\mathbb{E}[w(\boldsymbol{x}, y)]}.$$

The solution is

$$c^\star(\boldsymbol{x}) = \begin{cases} 1 & \mathbb{E}_w[y \mid \boldsymbol{x}] = \mathbb{P}_w(y = 1 \mid \boldsymbol{x}) - \mathbb{P}_w(y = -1 \mid \boldsymbol{x}) > 0 \\ -1 & \text{otherwise.} \end{cases}$$

We can directly minimize $J(f + \alpha c)$ without making a Taylor approximation to determine the optimal $\alpha^\star$,

$$\begin{aligned} \alpha^\star &= \operatorname*{argmin}_\alpha \mathbb{E}_w[-y\alpha c(\boldsymbol{x})] \\ &= \frac{1}{2}\log\left(\frac{\mathbb{P}_w(y = 1 \mid \boldsymbol{x})}{\mathbb{P}_w(y = -1 \mid \boldsymbol{x})}\right) && \text{Application of Lemma 9.1.} \\ &= \frac{1}{2}\log\left(\frac{1 - \epsilon_w}{\epsilon_w}\right), \end{aligned}$$

where $\epsilon_w = \mathbb{E}_w[\mathbb{1}\{y \neq f(\boldsymbol{x})\}]$.

Combining these steps yields the AdaBoost update,

$$\begin{aligned} f(\boldsymbol{x}) &\leftarrow f(\boldsymbol{x}) + \alpha^\star c(\boldsymbol{x}) \\ w(\boldsymbol{x}, y) &\leftarrow w(\boldsymbol{x}, y)\exp(-y\alpha^\star c(\boldsymbol{x})) = w(\boldsymbol{x}, y)\exp(\alpha^\star \mathbb{1}\{c(\boldsymbol{x}) \neq y\}). \end{aligned}$$

This concludes the proof that the AdaBoost algorithm minimizes the exponential loss function by iteratively adding weighted models via second-order optimization. ∎

10 Stable diffusion

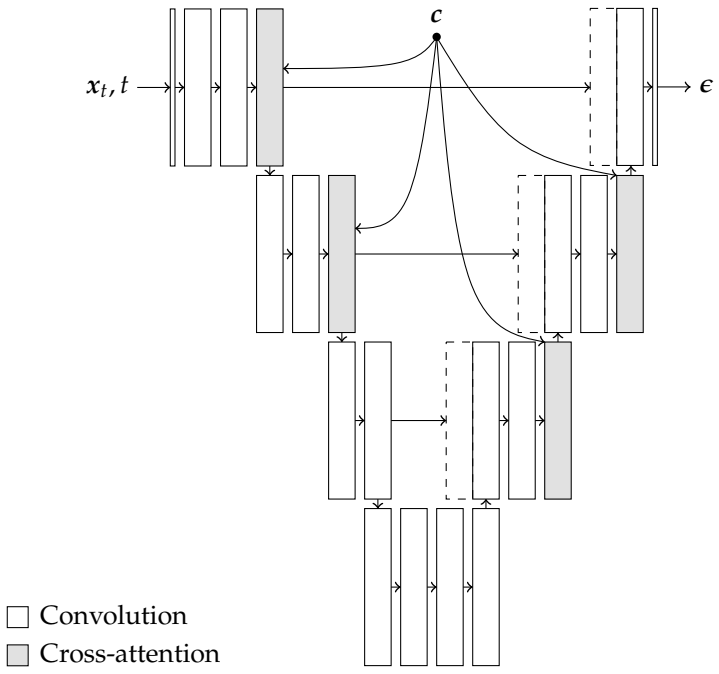

Figure 10.1. Architecture of stable diffusion.

10.1 Diffusion models

An SDE (***S**tochastic **D**ifferential **E**quation*) is a differential equation in which one or more terms is a stochastic process, resulting in a solution that is also stochastic. Typically, the SDE of a diffusion process is of the following form,

$$\mathrm{d}\boldsymbol{X}_t = \boldsymbol{\mu}(\boldsymbol{X}_t, t)\mathrm{d}t + \sigma(\boldsymbol{X}_t, t)\mathrm{d}\boldsymbol{W}_t,$$

where $\boldsymbol{W}_t$ is a Wiener process (or Brownian motion). This equations tells us that the change in $\boldsymbol{X}_t$ is driven by a deterministic factor $\boldsymbol{\mu}(\boldsymbol{X}_t, t)$ and a stochastic factor $\sigma(\boldsymbol{X}_t, t)d\boldsymbol{W}_t$. Note that $\boldsymbol{\mu}(\cdot, \cdot)$ and $\sigma(\cdot, \cdot)$ induce a probability distribution over time, p_t of $\boldsymbol{X}_t$.

Anderson [1982] showed that the reverse SDE of the diffusion process can be computed as follows,

$$\mathrm{d}\boldsymbol{X}_t = \Big[\boldsymbol{\mu}(\boldsymbol{X}_t, t) - \sigma^2(\boldsymbol{X}_t, t)\boldsymbol{\nabla}_{\boldsymbol{X}} \log p_t(\boldsymbol{X}_t)\Big]\mathrm{d}t + \sigma(\boldsymbol{X}_t, t)\mathrm{d}\bar{\boldsymbol{W}}_t,$$

where $\bar{\boldsymbol{W}}_t$ is a standard Wiener process when time flows backwards from T to 0, and $\mathrm{d}t$ is an infinitesimal negative timestep.

Using the DDPM scheduler, diffusion models have the following forward process,

$$\boldsymbol{x}_{t+1} = \sqrt{1-\beta_t}\boldsymbol{x}_t + \sqrt{\beta_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}).$$

Conditioned on $\boldsymbol{x}_t$, we can reconstruct $\boldsymbol{x}_{t-1}$ as follows with a predicted noise $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$,

$$\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\right) + \sqrt{1-\alpha_t}\boldsymbol{z}, \quad \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \quad (1)$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{\tau=1}^{t} \alpha_t$. Thus, we need to learn this function.

It can be shown that a diffusion model with the DDPM scheduler is an approximation of a discretization of the following SDE,

$$\mathrm{d}\boldsymbol{x}_t = -\frac{1}{2}\beta_t \boldsymbol{x}_t \mathrm{d}t + \sqrt{\beta_t}\mathrm{d}\boldsymbol{w}_t.$$

Drift term $\boldsymbol{\mu}(\boldsymbol{x}_t, t) \doteq -\frac{1}{2}\beta_t \boldsymbol{x}_t$; diffusion term $\sigma(t) \doteq \sqrt{\beta_t}$. $\mathrm{d}\boldsymbol{w}_t$ is a Gaussian with variance dt. We can show that this approximates the diffusion model by discretizing,

$$\begin{aligned}\boldsymbol{x}_{t+1} - \boldsymbol{x}_t &= -\frac{1}{2}\beta_t \boldsymbol{x}_t + \sqrt{\beta_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \\ \boldsymbol{x}_{t+1} &= \left(1 - \frac{1}{2}\beta_t\right)\boldsymbol{x}_t + \sqrt{\beta_t}\boldsymbol{\epsilon} \\ &\approx \sqrt{1-\beta_t}\boldsymbol{x}_t + \sqrt{\beta_t}\boldsymbol{\epsilon}.\end{aligned}$$

The reverse process is thus given by the following reverse SDE,

$$\mathrm{d}\boldsymbol{x}_t = \left[-\frac{1}{2}\beta_t \boldsymbol{x}_t - \beta_t \boldsymbol{\nabla}_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)\right]\mathrm{d}t + \sqrt{\beta_t}\mathrm{d}\tilde{\boldsymbol{w}}_t.$$

In practice, we train a diffusion model by randomly sampling $\boldsymbol{x}_0 \sim p_0, t \sim \mathrm{Unif}([T]), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and performing a gradient step on the following loss function,

$$\ell = \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\sqrt{1-\beta_t}\boldsymbol{x}_0 + \sqrt{\beta_t}\boldsymbol{\epsilon}\right)\right\|^2.$$

We can sample by iteratively denoising using Equation (1), starting from $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

10.2 U-net

U-nets [Ronneberger et al., 2015] are models used for image-to-image translation tasks. In the case of diffusion models, we have such a task, where we get $\boldsymbol{x}_t$ as input and want to predict $\boldsymbol{\epsilon}$. This framework is used to model $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$. U-nets work by downsampling the input in stages and then upsampling back to the original space in the same stages. At every step of upsampling, the output of the corresponding downsampling step is concatenated to its input. In this way, we get low-level and high-level information.

10.3 Latent diffusion models

Stable diffusion [Rombach et al., 2022] performs diffusion modeling in the latent space of a pretrained VAE [Kingma, 2013]. During training, we thus first map the input image $\boldsymbol{x}_0 \in \mathbb{R}^d$ to its latent encoding,

$$\boldsymbol{z}_0 = \boldsymbol{\mathcal{E}}(\boldsymbol{x}_0), \quad \boldsymbol{z}_0 \in \mathbb{R}^{d'}, \quad d' \ll d.$$

Then, we use the same loss function as above, where we sample a random timestep and noise,

$$\ell = \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\sqrt{1-\beta_t}\boldsymbol{z}_0 + \sqrt{\beta_t}\boldsymbol{\epsilon}\right)\right\|^2, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d'}).$$

Then, during inference, we sample a noise vector in the latent space $z_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d'})$ and denoise using Equation (1) to get $\tilde{\boldsymbol{z}}_0$. Lastly, we decode the latent vector back into pixel space,

$$\tilde{\boldsymbol{x}}_0 = \boldsymbol{\mathcal{D}}(\tilde{\boldsymbol{z}}_0).$$

This process requires a well-behaving latent space, so the regularization term that the VAE framework places on the latent space is very important.

10.4 Text embeddings

In order to perform text-to-image generation, we will need a continuous high-dimensional representation of the input text. For this, we use CLIP [Radford et al., 2021]. CLIP trains image and text transformer models to align text-image pairs. Because of this, they contain more semantic embeddings than other methods of training models to obtain text embeddings. Given a text input sequence of size T_c, the CLIP text model returns a sequence of embeddings,

$$\boldsymbol{C} \in \mathbb{R}^{T_c \times d_c}.$$

We denote this matrix by $\boldsymbol{C}$, because we use it for conditioning.

This sequence is contextualized, because CLIP makes use of self-attention.

10.5 Cross-attention

Now the question becomes how to condition a U-net on the input text sequence $\boldsymbol{C} \in \mathbb{R}^{T_c \times d_c}$. Stable diffusion [Rombach et al., 2022] does this by making use of cross-attention blocks, which it places at the end of every downsampling stage of the U-net. It first rearranges the output of the downsampling block into timesteps,

$$\boldsymbol{X} \in \mathbb{R}^{T \times d}.$$

Then, it computes queries from $\boldsymbol{X}$, and keys and values from $\boldsymbol{C}$,

$$\begin{aligned}
\boldsymbol{Q} &= \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{W}_Q \in \mathbb{R}^{d_k \times d} \\
\boldsymbol{K} &= \boldsymbol{C}\boldsymbol{W}_K, \quad \boldsymbol{W}_K \in \mathbb{R}^{d_k \times d_c} \\
\boldsymbol{V} &= \boldsymbol{C}\boldsymbol{W}_V, \quad \boldsymbol{W}_V \in \mathbb{R}^{d_v \times d_c}.
\end{aligned}$$

It uses these to perform the attention mechanism with a residual connection,

$$\boldsymbol{\Xi} = \boldsymbol{X} + \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V},$$

where $\boldsymbol{\Xi}$ denotes the conditioned representation of $\boldsymbol{X}$. Note that this architecture is agnostic to the type of the condition. As long as we can embed the conditioning variable, we can condition on it in this way—*e.g.*, we can additionally condition on images [Ye et al., 2023].

References

Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

Jonas Hübotter, Bhavya Sukhija, Lenart Treven, Yarden As, and Andreas Krause. Information-based transductive active learning. *arXiv preprint arXiv:2402.15898*, 2024.

Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

Pragya Sur and Emmanuel J Candès. A modern maximum-likelihood theory for high-dimensional logistic regression. *Proceedings of the National Academy of Sciences*, 116(29):14516–14525, 2019.

Victor Veitch, Alexander D'Amour, Steve Yadlowsky, and Jacob Eisenstein. Counterfactual invariance to spurious correlations in text classification. *Advances in neural information processing systems*, 34:16196–16208, 2021.

Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint arXiv:2308.06721*, 2023.