Machine Learning

DA-GA 1003

Yao Xiao

Spring 2023

Contents

1/24 Lecture

0 What is Machine Learning

Machine learning problems. Given an input x, predict an output y.

- Binary classification problem: There are two possible outputs.
- Multiclass classification problem: Choose an output out of a discrete set of possible outputs.
- Probabilistic classification or soft classification problem.
- Regression problem: The output is continuous.

Rule-based approach. Study the problem, write rules, evaluate, and analyze errors. Repeat these steps until the evaluated result can be launched. Rule-based approaches are generally interpretable and can produce reliable answers. However, rules does not generalize to unanticipated input combinations and do not naturally handle uncertainty. Moreover, it is labor-intensive to build and hard to scale.

The machine learning approach. We have the machine learn on its own from training data, which contains many examples of input-output pairs. Learning from training data of this form is called **supervised learning**. A machine learning algorithm learns from the training data, which outputs a predictions function that produces output y given input x. The success of machine learning depends on the availability of large amounts of data and **generalization** to unseen examples.

Beyond prediction.

- Unsupervised learning: Finding structures in data, *e.g.*, clustering.

- Reinforcement learning: Optimizing long-term objective, *e.g.*, Go.
- Representation learning: Learning good features of real-world objects, *e.g.*, text.

1 Statistical Learning Theory

1.1 Decision Theory

In data science problems, we generally needs to make a decision, take an action, or produce an output. Decision theory is about finding "optimal" actions, under various definitions of optimality.

Typical sequence of events. Observe input x, take action a, observe outcome y, and evaluate action in relation to the outcome. The input space is denoted $\mathscr{X}$, action space $\mathscr{A}$, and outcome space $\mathscr{Y}$.

Definition 1.1. A **prediction function** (or **decision function**) gets input $x \in \mathscr{X}$ and produces an action $a \in \mathscr{A}$, that is,

$$f : \mathscr{X} \to \mathscr{A}, \qquad x \mapsto f(x). \tag{1}$$

Definition 1.2. A **loss function** evaluates an action in the context of the outcome y, that is,

$$\ell : \mathscr{A} \times \mathscr{Y} \to \mathbb{R}, \qquad (a, y) \mapsto \ell(a, y). \tag{2}$$

Evaluating a prediction function. The goal is to find the optimal prediction function. Intuitively, if we can evaluate how good a prediction function is, we can turn this into an optimization problem. However, the loss function l evaluates only a single action. We will need to evaluate the prediction function as a whole. For this, we refer to the standard **statistical learning theory** framework.

1.2 Statistical Learning Theory

Define a space where the prediction function is applicable. Assume that there is a data generating function $P_{X \times Y}$. Also assume that all input-output pairs (x, y) are generated independently and identically distributed from $P_{X \times Y}$.

Definition 1.3. The **risk** of a prediction function $f : \mathscr{X} \to \mathscr{A}$ is

$$R(f) = \mathbb{E}_{(x,y) \sim P_{\mathscr{X} \times \mathscr{Y}}}[\ell(f(x), y)]. \tag{3}$$

In words, it is the **expected loss** of f over $P_{X \times Y}$.

Definition 1.4. A **Bayes prediction function** $f^* : \mathscr{X} \to \mathscr{A}$ is a function that achieves the minimal risk among all possible functions, that is,

$$f^* \in \arg\min_f R(f), \tag{4}$$

where the minimum is taken over all functions from $\mathscr{X}$ to $\mathscr{A}$.

The risk of a Bayes function is called the **Bayes risk**, and a Bayes prediction function is often called the "target function", since it is the best prediction function we can possibly produce. Note however, that we cannot actually compute the risk function since we do not know $P_{X \times Y}$. However, we can estimate it. Assume that we have sample data $\mathscr{D}_n = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ drawn independently and identically distributed from $P_{X \times Y}$, and we draw inspiration from the strong law of large numbers: If $z_1, \cdots, z_n$ are independently and identically distributed with expected value $\mathbb{E}z$, then

$$\mathbb{P}\left[\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} z_i = \mathbb{E}z\right] = 1. \tag{5}$$

Definition 1.5. The **empirical risk** of $f : \mathscr{X} \to \mathscr{A}$ with respect to a sample $\mathscr{D}_n$ is

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i). \tag{6}$$

By the strong law of numbers, we know that $\lim_{n\to\infty} \hat{R}_n(f) = R(f)$ almost surely.

Definition 1.6. A function $\hat{f}$ is an **empirical risk minimizer** if

$$\hat{f} \in \arg\min_f \hat{R}_n(f), \tag{7}$$

where the minimum is taken over all functions from $f : \mathscr{X} \to \mathscr{A}$.

1.3 Constrained Empirical Risk Minimization

Empirical risk minimization only memorizes the data (risk can be very different from empirical risk depending on the samples and the distribution). In order to improve generalization from the training inputs to new inputs, we need to smooth things out. One approach is **constrained empirical risk minimization**, which instead of minimizing empirical risk over all prediction functions, constrains the search to a particular subset of the space of functions, called a **hypothesis space**.

Definition 1.7. A **hypothesis space** $\mathscr{F}$ is a set of prediction functions from $\mathscr{X}$ to $\mathscr{A}$ that we consider when applying empirical risk minimization.

A hypothesis space is desired to include only those functions that have the desired "singularity", *e.g.*, smoothness, simplicity, etc., and are easy to work with.
Given a hypothesis space $\mathscr{F}$ and a set of prediction functions mapping $\mathscr{X}$ to $\mathscr{A}$, an **empirical risk minimizer** in $\mathscr{F}$ is a function $\hat{f}_n$ such that

$$\hat{f}_n \in \arg\min_{f\in\mathsf{F}} \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i). \tag{8}$$

A **risk minimizer** in $\mathscr{F}$ is a function f_F such that

$$f_\mathsf{F} \in \arg\min_{f\in\mathsf{F}} \mathbb{E}[\ell(f(x), y)]. \tag{9}$$

Definition 1.8. The **excess risk** compares the risk of f to the Bayes optimal f^*, that is,

$$\textsc{Excess Risk}(f) = R(f) - R(f^*). \tag{10}$$

The excess risk of the empirical risk minimizer $\hat{f}_n$ can thus be decomposed as

$$\textsc{Excess Risk}(\hat{f}_n) = R(\hat{f}_n) - R(f^*) = \underbrace{R(\hat{f}_n) - R(f_\mathsf{F})}_{\text{estimation error}} + \underbrace{R(f_\mathsf{F}) - R(f^*)}_{\text{approximation error}}. \tag{11}$$

We shall notice that there is a tradeoff between estimation error and approximation error.

- The approximation error $R(f_\mathsf{F}) - R(f^*)$ is a property of the class $\mathscr{F}$, acting as the penalty for restricting consideration only to $\mathscr{F}$ (rather than considering all possible functions). Therefore, bigger $\mathscr{F}$ generally means smaller approximation error.
- The estimation error $R(\hat{f}_n) - R(f_\mathsf{F})$ is the performance hit for choosing f using finite training data (equivalently, for minimizing the empirical risk rather than the true risk). Moreover, with smaller $\mathscr{F}$ we often expect smaller estimation error.

In practice, however, we do not find the empirical risk minimizer $\hat{f}_n \in \mathscr{F}$. Instead, we find $\tilde{f}_n \in \mathscr{F}$ that we hope is good enough. The excess risk of $\tilde{f}_n$ can thus be decomposed as

$$\textsc{Excess Risk}(\tilde{f}_n) = R(\tilde{f}_n) - R(f^*) = \underbrace{R(\tilde{f}_n) - R(\hat{f}_n)}_{\text{optimization error}} + \underbrace{R(\hat{f}_n) - R(f_\mathsf{F})}_{\text{estimation error}} + \underbrace{R(f_\mathsf{F}) - R(f^*)}_{\text{approximation error}}. \tag{12}$$

1/31 Lecture

2 Gradient Descent

We assume that the objective function $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable, and we want to find

$$x^* = \arg\min_{x \in \mathbb{R}^d} f(x). \tag{13}$$

Given that f is differentiable at $x_0 \in \mathbb{R}^d$, The **gradient** of f at the point x_0, denoted $\nabla_x f(x_0)$, is the direction in which $f(x)$ increases the fastest, if we start from x_0.

2.1 Gradient Descent

Given that the gradient is the direction of steepest ascent, to reach a local minimum as fast as possible, we want to go in the opposite direction from the gradient. The method of **gradient descent** is described as in Algorithm 1.

Algorithm 1 Gradient Descent

1: Initialize $x \leftarrow 0$;
2: **repeat**
3: $\quad x \leftarrow x - \eta \nabla f(x)$;
4: **until** the stopping criterion is satisfied;

Note that a fixed step size η will work eventually, as long as it is small enough. On the other hand, if η is too large, the optimization process may diverge. We will introduce the convergence theorem for fixed step size below.

Theorem 2.1. Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is convex and differentaible, and ∇f is **Lipschitz continuous** with constant $L > 0$, *i.e.*, for any $x, x' \in \mathbb{R}^d$, it holds that

$$\|\nabla f(x) - \nabla f(x')\| \leq L \|x - x'\|. \tag{14}$$

Then the gradient descent with fixed step size $\eta \leq L^{-1}$ converges. In particular,

$$f(x^{(k)}) - f(x^*) \leq \frac{\left\|x^{(0)} - x^*\right\|^2}{2\eta k}. \tag{15}$$

Stopping criterion. Recall that $\nabla f(x) = 0$ at a local minimum. Therefore, gradient descent should wait until $\|\nabla f(x)\|_2 \leq \epsilon$ for some chosen ϵ. However, there are some conditions for an early stopping. If we evaluate the loss on validation data after each iteration, we may want to stop when the loss no longer improves (or even gets worse).

2.2 Stochastic Gradient Descent

Suppose that $\mathscr{F} = \left\{f_w : \mathscr{X} \to \mathscr{A};\, w \in \mathbb{R}^d\right\}$ is the hypothesis space of functions, parametrized by $w \in \mathbb{R}^d$. Finding an empirical risk minimizer entails finding a w that minimizes

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i). \tag{16}$$

Suppose that $\ell(f_w(x_i), y_i)$ is differentiable as a function of w, then we can do gradient descent on $\hat{R}_n(w)$. At each iteration, we compute the gradient at the current w, that is,

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i). \tag{17}$$

However, note that we have to iterate over all n training points to take a single step, which is $O(n)$ for each iteration, which will *not* scale to "big data". Therefore, instead of using gradient, we can use a noisy estimate of the gradient. The intuition behind this is that gradient descent is an iterative procedure, so at every step, we will have a chance to recover from previous missteps.

Minibatch gradient. The **full gradient** for the empirical risk is

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i), \tag{18}$$

which is an average over the **full batch** of data $\mathscr{D}_n = \{(x_1, y_1), \cdots, (x_n, y_n)\}$. Now take a random subsample $\{(x_{m_1}, y_{m_1}), \cdots, (x_{m_N}, y_{m_N})\}$ of size N, which we call a **minibatch**. The **minibatch gradient** is then

$$\nabla \hat{R}_N(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(f_w(x_{m_i}), y_{m_i}). \tag{19}$$

Based on the minibatch gradient, the method of **minibatch gradient descent** is described as in Algorithm 2.

Algorithm 2 Minibatch Gradient Descent

1: Initialize $w \leftarrow 0$;
2: **repeat**
3: Randomly choose N points $\{(x_i, y_i)\}_{i=1}^{N} \subseteq \mathscr{D}_n$;
4: $w \leftarrow w - \eta \left(\frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(f_w(x_{m_i}), y_{m_i}) \right)$;
5: **until** the stopping criterion is satisfied;

The comparison between using the full batch and some minibatch is shown as in Figure 1. We can see that stochastic methods work well far from the optimum but struggle close to the optimum.

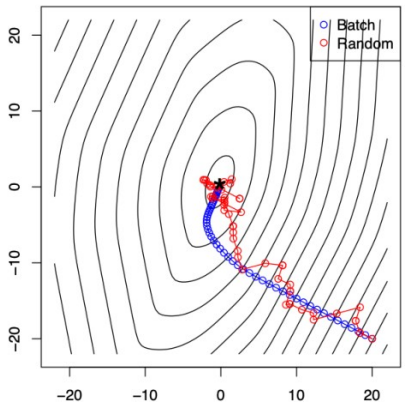

Figure 1: The comparison between gradient descents using the full batch and sampling a random minibatch.

Properties of the minibatch gradient. The minibatch gradient is an **unbiased estimator** for the (full) batch gradient. This means that

$$\mathbb{E}[\nabla \hat{R}_N(w)] = \nabla \hat{R}_n(w). \tag{20}$$

The bigger the minibatch, the better the estimation. We have that

$$\mathrm{Var}[\nabla \hat{R}_N(w)] = \frac{1}{n} \mathrm{Var}[\nabla \hat{R}_1(w)]. \tag{21}$$

There is a tradeoff on the size of the minibatch. If we choose larger sizes, the estimation of the gradient will be better, but the processing will be slower. On the other hand, if we choose smaller sizes, we will get worse estimation of the gradient, but the processing will be quite fast. Also note that due to vectorization, we can oftern get minibatches of certain sizes for free.

Convergence of stochastic gradient descent. Stochastic gradient descent is just minibatch gradient descent with $N = 1$. That is, we use a single randomly chosen point to determine the step direction. Theoretically, gradient

descent is much faster than stochastic gradient descent in convergence rate. It is much faster to add a digit of accuracy, but most of that advantage comes into play once we are already close to the minimum. However, in many machine learning we do not care about optimizing to high accuracy. For stochastic gradient descent, a fixed step size can work well in practice. Another typical approach is to keep a fixed step size but reduce by constant factor whenever validation performance stops improving. Other trick can be found in *Bottou 2012, Stochastic Gradient Descent Tricks*.

Remark 2.2. Given inconsistency in terminology these days, always clarify the batch (minibatch) size.

3 Loss Functions

3.1 Regression Loss Functions

Examples of regression problems include predicting the stock price given history prices, predicting medical cost of given age, sex, region, BMI, etc., and predicting the age of a person based on their photos. We have the input space $\mathscr{X} = \mathbb{R}^d$, the action space $\mathscr{A} = \mathbb{R}$, and the outcome space $\mathscr{Y} = \mathbb{R}$. We denote $\hat{y}$ as the predicted value (the action), and denote y as the actual observed value (the outcome).

Distance-based loss. A loss $\ell(\hat{y}, y)$ is called **distance-based** if it only depends on the residual and is zero when the residual is zero. That is, $\ell(\hat{y}, y) = \psi(y - \hat{y})$ for some $\psi : \mathbb{R} \to \mathbb{R}$ with $\psi(0) = 0$. Note that distance-based losses are translation-invariant, but sometimes we nay not want to use a translation-invariant loss. A relative error

$$\ell(\hat{y}, y) = \frac{\hat{y} - y}{y} \tag{22}$$

is a more natural loss, and is not translation-invariant.

Some losses for regression. Let $r = y - \hat{y}$ be the residual.

- **Square** or l_2 loss: $\ell(r) = r^2$.
- **Absolute** or **Laplace** or l_1 loss: $\ell(r) = |r|$.
- **Huber** loss: $\ell(r) = r^2/2$ for $|r| \le \delta$, and $\ell(r) = \delta(|r| - \delta/2)$ for $|r| > \delta$. Huber loss is quadratic for small r and linear for large r. While l_2 loss is not robust (too much affected by **outliers**) and l_1 loss is not differentiable, Huber loss is both robust and differentiable.

3.2 Classification Loss Functions

Examples of classification problems include predicting whether the image contains a cat and predicting whether the email is SPAM. We have the input space $\mathscr{X} = \mathbb{R}^d$, the outcome space $\mathscr{Y} = \{-1, 1\}$, and the action space $\mathscr{A} = \mathbb{R}$ (easier to work with than $\mathscr{A} = \{-1, 1\}$). If $f(x) > 0$, we predict 1, and otherwise we predict -1.

Definition 3.1. In this context, the real-valued prediction function $f : \mathscr{X} \to \mathscr{A} = \mathbb{R}$ is called the **score function**, and the value $f(x)$ is called the **score** for the input x.

Note that the magnitude of the score can be interpreted as our **confidence** of our prediction.

Definition 3.2. The **margin** (or **functional margin**) for a predicted score $\hat{y}$ and the true class $y \in \{-1, 1\}$ is $y\hat{y}$.

The margin is often written as $yf(x)$, where f is the score function. The margin is a measure of how correct we are. If y and $\hat{y}$ has the same sign, then the prediction is correct and the margin is positive. On the other hand, if y and $\hat{y}$ have different signs, then the prediction is incorrect and the margin is negative. Our goal is to maximize the margin, and most classification losses depend only on the margin, namely **margin-based losses**.

Some losses for classification. Let $m = yf(x)$ be the margin.

- **Zero-One** loss: $\ell(m) = \mathbb{1}_{m \leq 0}$. However, the empirical risk for the 0-1 loss is

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{y_i f(x_i) \leq 0}, \tag{23}$$

  which is non-convex, not differentiable, and not even continuous. Therefore, minimizing empirical 0-1 risk is computationally *infeasible*, and optimization is NP-hard.

- **SVM** or **hinge** loss: $\ell(m) = \max\{1 - m, 0\}$. The hinge loss is convex and upper bounds the 0-1 loss. However, it is not differentiable at $m = 1$.

- **Logistic** or **log** loss: $\ell(m) = \log(1 + \exp(-m))$. The logistic loss is differentiable, and it always rewards a larger margin (the loss is never 0).

The plots of these classification losses are shown as in Figure 2.

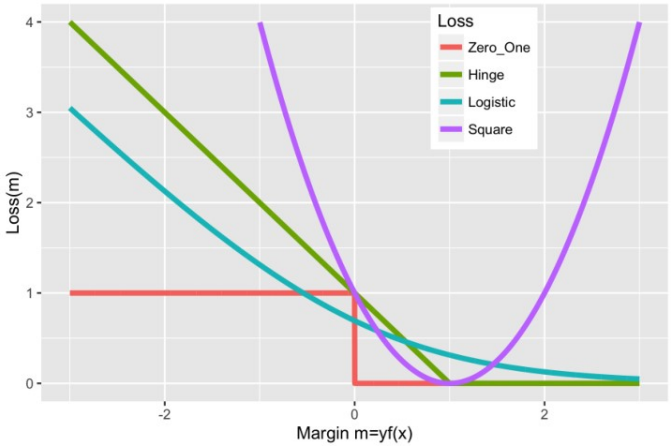

Figure 2: The comparison among different losses for classification introduced above.

2/7 Lecture

4 Feature Selection and Regularization

4.1 Feature Selection

Recall the trade-off between approximation error and estimation error. Bigger $\mathscr{F}$ can provide better approximation but may lead to overfit (needs more samples); smaller $\mathscr{F}$ is less likely to overfit but can be far from the true function. To control the size of $\mathscr{F}$, we need some measure of its **complexity**: the number of variables/features, the degree of the polynomial, etc.

General approach. Learn a sequence of models varying in complexity from the training data $\mathscr{F}_1 \subset \mathscr{F}_2 \subset \cdots \subset \mathscr{F}_n \subset \cdots \mathscr{F}$. For instance, $\mathscr{F}$ can be the family of all polynomials and $\mathscr{F}_d$ can be all polynomials with degree $\leq d$. Then, select one of these models based on a score (*e.g.*, validation error). However, this approach can be really expensive. Consider feature selection in linear regression, where $\mathscr{F}$ denotes the linear functions using all features and $\mathscr{F}_d$ denotes the linear functions using fewer than d features. Clearly we would have to iterate over all subsets which is exponential.

Greedy selection approach. The forward greedy selection algorithm is described as in Algorithm 3

Algorithm 3 Forward Greedy Selection

Initialize $S \leftarrow \varnothing$;
repeat
  for each feature $i \notin S$ **do**
    Learn a model using features $S \cup \{i\}$;
    Compute the score of the model α_i;
  end for
  $j \leftarrow \arg\max_{i \notin S} \alpha_i$;
  $S \leftarrow S \cup \{j\}$ unless α_j does not improve the current best score;
until α_j does not improve the current best score;

There is also a backward greedy selection scheme which starts with all features and continuously removes the worst. Note that forward and backward greedy selection algorithms do not guarantee to find the best solution, and do not in general result in the same subset.

4.2 l_2 and l_1 Regularization

An objective function that balances the number of features and the prediction performance is

$$\textsc{Score}(S) = \textsc{Training Loss}(S) + \lambda |S|, \tag{24}$$

where larger λ can penalize complex models more heavily. Normally, we would find λ using the validation data.

Definition 4.1 (Tikhonov regularization)**.** For complexity measure $\Omega : \mathscr{F} \to [0, \infty)$ and a fixed $\lambda > 0$, the objective under **Tikhonov regularization** can be formalized as

$$\min_{f \in F} \left(\frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \lambda \Omega(f) \right). \tag{25}$$

Linear regression with l_2 regularization. We have a linear model $\mathscr{F} = \left\{ f : \mathbb{R}^d \to \mathbb{R};\ f(x) = w^\top x,\ w \in \mathbb{R}^d \right\}$, with the square loss function $\ell(\hat{y}, y) = (y - \hat{y})^2$ and training data $\mathscr{D}_n = \{(x_1, y_1), \cdots, (x_n, y_n)\}$. The linear least squares regression is the empirical risk minimizer for square loss over $\mathscr{F}$, which is

$$\hat{w} = \arg\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(w^\top x_i - y_i \right)^2. \tag{26}$$

This often overfits, especially when d is large compared with n. For instance, in NLP one can have 1000000 features for 10000. documents. Therefore, we want to penalize large weights by

$$\hat{w} = \arg\min_{w \in \mathbb{R}^d} \left(\frac{1}{n} \sum_{i=1}^{n} \left(w^\top x_i - y_i \right)^2 + \lambda \left\| w \right\|_2^2 \right). \tag{27}$$

This is also known as **ridge regression**. Note that l_2 regularization reduces sensitivity to changes in input. To see this, we can use the Cauchy-Schwarz inequality to compute that

$$\left| \hat{f}(x + h) - \hat{f}(x) \right| = \left| w^\top (x + h) - w^\top x \right| = \left| w^\top h \right| \leq \left\| w \right\|_2 \left\| h \right\|_2. \tag{28}$$

This proves a bound on the maximum rate of change of $\hat{f}$, thus penalizing large $\|w\|_2^2$ is penalizing $\hat{f}$ with too large rate of change which often means overfitting. Now we compare linear regression and ridge regression:

Linear regression		Ridge regression
$L(w) = \frac{1}{2} \left\| Xw - y \right\|_2^2$	Objective	$L(w) = \frac{1}{2} \left\| Xw - y \right\|_2^2 + \frac{\lambda}{2} \left\| w \right\|_2^2$
$\nabla L(w) = X^\top (Xw - y)$	Gradient	$\nabla L(w) = X^\top (Xw - y) + \lambda w$
$X^\top X w = X^\top y$	Closed-form solution	$(X^\top X + \lambda I) w = X^\top y$
		note: $(X^\top X + \lambda I)$ is always invertible

(29)

Linear regression with l_1 regularization. Linear regression with l_1 regularization, also called **Lasso regression**, penalizes large weights by

$$\hat{w} = \arg\min_{w \in \mathbb{R}^d} \left(\frac{1}{n} \sum_{i=1}^{n} \left(w^\top x_i - y_i \right)^2 + \lambda \|w\|_1 \right). \tag{30}$$

Constrained versus penalized. Recall the difference between constrained empirical risk minimization (**Ivanov**) and penalized empirical risk minimization (**Tikhonov**). For complexity measure $\Omega : \mathscr{F} \to [0, \infty)$ and a fixed $r \geq 0$, Ivanov regularization can be formulated as

$$\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i), \qquad \text{such that } \Omega(f) \leq r. \tag{31}$$

On the other hand, for the same complexity measure Ω and a fixed $\lambda > 0$, Tikhonov regularization is formulated as

$$\min_{f \in F} \left(\frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \lambda \Omega(f) \right). \tag{32}$$

Here, r and λ has the same role. Let $L : \mathscr{F} \to \mathbb{R}$ be any performance measure of f (*e.g.*, the empirical risk of f). For many L and Ω, Ivanov and Tikhonov are equivalent: any solution f^* we can get from Ivanov, we can also get from Tikhonov and vice versa. The conditions for this equivalence can be derived from the Lagrangian duality theory, and in practice, both approaches are effective: we will use whichever one is more convenient for training or analysis.

4.3 Sparsity

Consider $\mathscr{F} = \{f;\ f(x) = w_1 x_1 + w_2 x_2\}$, then we can represent each function in $\mathscr{F}$ as a point $(w_1, w_2) \in \mathbb{R}^2$. We know that the Ivanov regularization constraint with l_1 regularization is the closure of the diamond $|w_1| + |w_2| = r$, and that with l_2 regularization is the closure of the circle $w_1^2 + w_2^2 = r^2$. As we can see in Figure 3, where the red lines represent the contours of the empirical risk $\hat{R}_n(w) = \sum_{i=1}^{n} \left(w^\top x_i - y_i \right)^2$ and the blue regions represent the area satisfying the complexity constraints, l_1 solutions tend to touch the corners, where $w_1 = 0$.

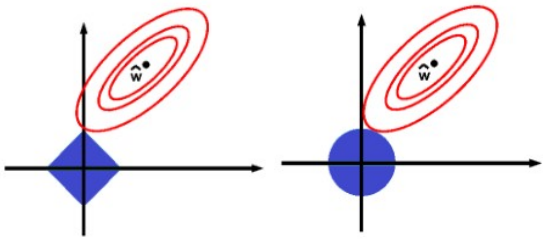

Figure 3: Estimation pictures for the lasso regression (with l_1 regularization, left) and the ridge regression (with l_2 regularization, right).

For l_2 regularization, as w_i becomes smaller, there is less and less penalty. The gradient, which determines the pace of the optimization, decreases as w_i approaches zero, and there is less incentive to make a small weight exactly equal to zero. On the other hand, for l_1 regularization, the gradient stays the same as the weights approach zero, and tends to push the weights to exactly zero even if they are already small, as is shown in Figure 3.

l_q regularization. We can generalize to l_q norm such that $\|w\|_q^q = \sum_{i=1}^{n} |w_i|^q$. Note that $\|w\|_q$ is a norm only if $q > 1$. The larger q, the closer the contour to a rectangle. The smaller q, the closer the contour to a cross. Therefore, when $q < 1$, the l_q constraint is not convex, and thus hard to optimize (Lasso is good enough in practice regarding sparsity). Also note that the l_0 norm is defined as the number of non-zero weights, *i.e.*, subset selection.

4.4 Minimizing the Lasso Objective

The ridge regression objective is differentiable, and there is a closed form solution. However, the Lasso objective

$$\min_{w\in\mathbb{R}^d} \sum_{i=1}^{n} \left(w^\top x_i - y_i\right)^2 + \lambda \|w\|_1 \tag{33}$$

is not differentiable since $\|w\|_1$ is not differentiable. Therefore, we will need some other approaches to find the minimum, including quadratic programming, projected stochastic gradient descent, and coordinate descent.

Quadratic programming. For any $a \in \mathbb{R}$, we denote its positive part as a^+ and its negative part as a^-, then clearly $a = a^+ - a^-$ and $|a| = a^+ + a^-$. Generalizing this notation to $\mathbb{R}^d$, we can reformulate the Lasso objective as

$$\min_{w^+,w^-} \sum_{i=1}^{n} \left((w^+ - w^-)^\top x_i - y_i\right)^2 + \lambda \cdot \mathbf{1}^\top (w^+ + w^-), \qquad \text{subject to } w_i^+ \geq 0,\ w_i^- \geq 0,\ \forall i. \tag{34}$$

This objective is differentiable (and in fact, convex and quadratic). Therefore, this is a convex **quadratic program** with linear constraints. Quadratic programming is a very well understood problem, and we can plug this formulation directly into a generic quadratic programming solver. However, note that this formulation does not include the constraint that w_i^+ and w_i^- are the positive and negative parts of some w, but it turns out that such a constraint is not necessary. Indeed, we can prove that the previous reformulation is equivalent to the following:

$$\min_{w} \min_{a,b} \sum_{i=1}^{n} \left((a-b)^\top x_i - y_i\right)^2 + \lambda \cdot \mathbf{1}^\top (a+b), \qquad \text{subject to } a_i \geq 0,\ b_i \geq 0,\ a_i - b_i = w_i,\ a_i + b_i = |w_i|,\ \forall i. \tag{35}$$

Projected stochastic gradient descent. Now that we have a differentiable objective, we could also use gradient descent. However, to handle the constraints, we must project w^+ and w^- into the constraint set after each step. In other words, if any component of w^+ or w^- becomes negative after some step, we set it back to 0.

Coordinate descent. The goal is to minimize $L(w) = L(w_1, \cdots, w_d)$ over $w = (w_1, \cdots, w_d) \in \mathbb{R}^d$. In gradient descent or stochastic gradient descent, each step potentially changes all entries of w. However, in **coordinate descent**, each step adjusts only a single coordinate w_i. The coordinate descent algorithm is described as in Algorithm 4.

Algorithm 4 Coordinate Descent

1: Initialize $w \leftarrow 0$;
2: **repeat**
3: Choose a coordinate $j \in \{1, \cdots, d\}$;
4: $w_j \leftarrow \arg\min_{w_j} L(w_1, \cdots, w_j, \cdots, w_d)$;
5: **until** the stopping criterion is satisfied;

Note that if the coordinate is chosen uniformly at random, then the algorithm is **stochastic coordinate descent**. If the coordinate is chosen based on a cyclic order, then the algorithm is **cyclic coordinate descent**. In general, the coordinate descent method is feasible since the Lasso objective coordinate minimization has a closed form solution. If

$$\hat{w}_j = \arg\min_{w_j\in\mathbb{R}} \sum_{i=1}^{n} \left(w^\top x_i - y_i\right)^2 + \lambda \|w\|_1, \tag{36}$$

then we have that

$$\hat{w}_j = \begin{cases} (c_j + \lambda)/a_j, & \text{if } c_j \in (-\infty, -\lambda), \\ 0, & \text{if } c_j \in [-\lambda, \lambda], \\ (c_j - \lambda)/a_j, & \text{if } c_j \in (\lambda, \infty), \end{cases} \tag{37}$$

where

$$a_j = 2\sum_{i=1}^{n} x_{i,j}^2, \quad c_j = 2\sum_{i=1}^{n} x_{i,j}\left(y_i - w_{-j}^\top x_{i,-j}\right), \tag{38}$$

$x_{i,j}$ is the jth component of x_i, w_{-j} is w without the jth component, and $x_{i,-j}$ is x_i without the jth component. In general, coordinate descent is not competitive with gradient descent, since its convergence rate is slower and the iteration cost is similar. However, it works very well for certain problems, with example applications such as the Lasso regression and support vector machines (SVM).

2/14 Lecture

5 Support Vector Machines (SVM)

5.1 The SVM Objective

There are generally two ways to derive the SVM optimization problem, that is, by maximizing the (geometric) margin or by minimizing the hinge loss with l_2 regularization.

5.1.1 Maximum Margin Classifier

Consider a linearly separable dataset $\mathscr{D}$, and we want to find a separating hyperplane such that $w^\top x_i > 0$ for all x_i where $y_i = 1$ and $w^\top x_i < 0$ for all x_i where $y_i = -1$. We introduce the **perceptron algorithm** described as in Algorithm 5, which guarantees to find a zero-error classifier (if exists) in finite steps.

Algorithm 5 Perceptron

```
Initialize w ← 0;
repeat
  for (x_i, y_i) ∈ 𝒟 do
    if y_i w^⊤ x_i < 0 (implying a wrong prediction) then
      w ← w + y_i x_i;
    end if
  end for
until the stopping criterion is satisfied (there does not exist misclassified examples);
```

However, for separable data, there are infinitely many zero-error classifiers, and the perceptron algorithm does not return a unique solution. In this case, we would prefer the classifier that is farthest from both classes of points, and the distance is measured by the **geometric margin** representing the smallest distance between the hyperplane and the points. Let us formalize the problem.

Definition 5.1. We say that $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^n$ is **linearly separable** if there exists $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, such that

$$y_i(w^\top x_i + b) > 0, \qquad \forall i \in \{1, \cdots, n\}. \tag{39}$$

The set $\left\{v \in \mathbb{R}^d;\ w^\top v + b = 0\right\}$ is called the **separating hyperplane**.

Definition 5.2. Let H be a hyperplane that separates the dataset $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^n$. The **geometric margin** of this hyperplane is

$$\min_i d(x_i, H), \tag{40}$$

the distance from the hyperplane to the closest data point.

We want to maximize the geometric margin, and given separating hyperplane $H = \left\{v;\ w^\top v + b = 0\right\}$, the objective can be written as

$$\text{maximize} \qquad \min_i d(x_i, H) = \min_i \left| \frac{w^\top x_i + b}{\|w\|_2} \right| = \min_i \frac{y_i(w^\top x_i + b)}{\|w\|_2}. \tag{41}$$

By moving the inner minimization problem, we can reformulate the problem as

$$\begin{aligned} \text{maximize} \qquad & M, \\ \text{subject to} \qquad & \frac{y_i(w^\top x_i + b)}{\|w\|_2} \geq M, \qquad \forall i. \end{aligned} \tag{42}$$

Fixing the norm $\|w\|_2$ to M^{-1}, we can obtain

$$\begin{aligned} \text{maximize} \quad & \frac{1}{\|w\|_2}, \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1, \qquad \forall i, \end{aligned} \tag{43}$$

which is equivalent to

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|_2^2, \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1, \qquad \forall i. \end{aligned} \tag{44}$$

Note that $y_i(w^\top x_i + b)$ is the (functional) margin. In other words, we want to find the minimum norm solution which has a margin of at least 1 on all examples.

Soft-margin SVM. It is possible that the dataset is not separable, *i.e.*, for any w where will be points with a negative margin. Therefore, we introduce **slack variables** to penalize small margins, such that

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i, \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i, \qquad \forall i, \\ & \xi_i \geq 0, \qquad \forall i. \end{aligned} \tag{45}$$

If $\xi_i = 0$ for all i, then the problem is reduced to hard-margin SVM. If $\xi_i = 1$, it means that x_i lies on the decision hyperplane. If $\xi_i > 1$, it means that x_i is past $\xi_i - 1$ margin width beyond the decision hyperplane. Some examples are shown as in Figure 4.

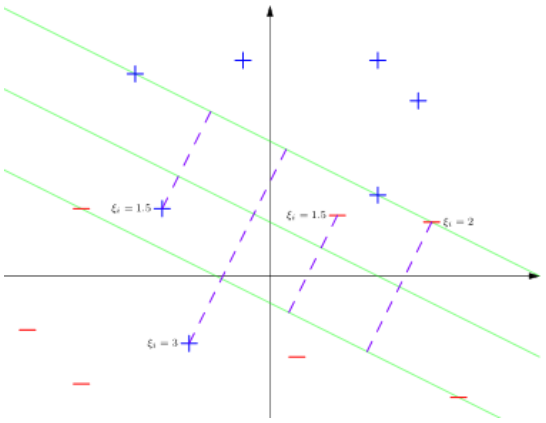

Figure 4: The implication of different values of ξ in soft margin SVM.

5.1.2 Minimizing the Hinge Loss

Recall that the SVM (or hinge) loss is defined as

$$\ell_{\text{hinge}}(m) = \max\{1 - m, 0\} = (1 - m)_+, \tag{46}$$

the "positive part" of $1 - m$ where $m = yf(x)$ is the margin. Hinge is a convex but not differentiable at $m = 1$. Moreover, it imposes a "margin error" only when $m < 1$. Using empirical risk minimization with l_2 regularization in Tikhonov style and the hinge loss, the SVM prediction function is the solution to

$$\min_{w\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\max\left\{0, 1 - y_i(w^\top x_i + b)\right\}. \tag{47}$$

The objective function is not differentiable because of the max operator, so we need to reformulate the problem as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i, \\ \text{subject to} \quad & \xi_i \geq \max\left\{0, 1 - y_i(w^\top x_i + b)\right\}, \qquad \forall i, \end{aligned} \tag{48}$$

which is equivalent to

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i, \\ \text{subject to} \quad & \xi_i \geq 1 - y_i(w^\top x_i + b), \qquad \forall i, \\ & \xi_i \geq 0, \qquad \forall i. \end{aligned} \tag{49}$$

5.2 Subgradient Descent for SVM

The SVM objective function is

$$J(w) = \frac{1}{n}\sum_{i=1}^{n}\ell_{\text{hinge}}(y_i w^\top x_i) + \lambda\|w\|_2^2, \tag{50}$$

where $\ell_{\text{hinge}}(m) = \max\{0, 1-m\}$. It is not differentiable, but think of gradient descent first. The gradient can be expressed as

$$\nabla_w J(w) = \nabla_w\left(\frac{1}{n}\sum_{i=1}^{n}\ell_{\text{hinge}}(y_i w^\top x_i) + \lambda\|w\|_2^2\right) = \frac{1}{n}\sum_{i=1}^{n}\nabla_w\ell_{\text{hinge}}(y_i w^\top x_i) + 2\lambda w. \tag{51}$$

The derivative of the hinge loss $\ell_{\text{hinge}}(m) = \max\{0, 1-m\}$ is

$$\ell'_{\text{hinge}}(m) = \begin{cases} 0, & \text{if } m > 1, \\ \text{undefined}, & \text{if } m = 1, \\ -1, & \text{if } m < 1. \end{cases} \tag{52}$$

By the chain rule, we thus have that

$$\nabla_w\ell_{\text{hinge}}(y_i w^\top x_i) = \ell'_{\text{hinge}}(y_i w^\top x_i)\cdot y_i x_i = \begin{cases} 0, & \text{if } y_i w^\top x_i > 1, \\ \text{undefined}, & \text{if } y_i w^\top x_i = 1, \\ -y_i x_i, & \text{if } y_i w^\top x_i < 1, \end{cases} \tag{53}$$

and thus if $y_i w^\top x_i \neq 1$ for all i, then

$$\nabla_w J(w) = \frac{1}{n}\sum_{y_i w^\top x_i < 1}(-y_i x_i) + 2\lambda w, \tag{54}$$

but $\nabla_w J(w)$ is undefined otherwise.

Subgradient. Recall the definition that if $f : \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, then for any $x, y \in \mathbb{R}^d$,

$$f(y) \geq f(x) = \nabla f(x)^\top (y - x). \tag{55}$$

This inspires the following definition.

Definition 5.3. A vector $g \in \mathbb{R}^d$ is a **subgradient** of a convex function $g : \mathbb{R}^d \to \mathbb{R}$ at x, if for all $z \in \mathbb{R}^d$,

$$f(z) \geq f(x) + g^\top (z - x). \tag{56}$$

An illustration of the definition of subgradient is shown as in Figure 5.

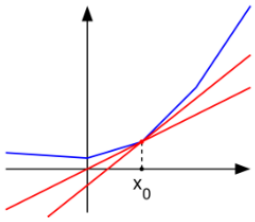

Figure 5: An illustration of subgradients. The blue curve is the graph of $f(x)$ (not differentiable at x_0), and each red line $x \mapsto f(x_0) + g^\top (x - x_0)$ is a global lower bound on $f(x)$, where g is some subgradient of f at x_0.

Definition 5.4. The set of all subgradients of f at x is called the **subdifferential**, denoted as $\partial f(x)$. We say that f is **subdifferentiable** at x if there exists at least one subgradient of f at x.

Remark 5.5. For convex functions, f is differentiable at x if and only if $\partial f(x) = \{\nabla f(x)\}$. Moreover, the subdifferential of the convex function f is always non-empty. Finally, x is a global optimum if and only if $0 \in \partial f(x)$. For non-convex functions, the subdifferential may be empty (if there exists no global underestimator).

Rules for computing subdifferential. Assume that f, f_1, and f_2 are convex functions.

- Non-negative scaling: $\partial(\alpha f)(x) = \alpha \partial f(x)$, for $\alpha > 0$.
- Summation: $\partial(f_1 + f_2)(x) = \partial f_1(x) + \partial f_2(x)$, comprehended as the Minkowski sum.
- Composing with affine functions: $\partial f(Ax + b) = A^\top \partial f(z)$, where $z = Ax + b$.
- Convex combinations of arg max gradients:

$$\partial(\max\{f_1, f_2\})(x) = \begin{cases} \{\nabla f_1(x)\}, & \text{if } f_1(x) > f_2(x), \\ \{\theta \nabla f_1(x) + (1-\theta)\nabla f_2(x);\ \theta \in [0,1]\}, & \text{if } f_1(x) = f_2(x), \\ \{\nabla f_2(x)\}, & \text{if } f_1(x) < f_2(x). \end{cases} \tag{57}$$

Therefore, we can compute the gradient of the objective function

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} \max\left\{0, 1 - y_i w^\top x_i\right\} + \lambda \|w\|_2^2 \tag{58}$$

as

$$\partial_w J(w) = \begin{cases} \{2\lambda w\}, & \text{if } y_i w^\top x_i > 1, \\ \{(1-\theta)(-y_i x_i) + 2\lambda w;\ \theta \in [0,1]\}, & \text{if } y_i w^\top x_i = 1, \\ \{-y_i x_i + 2\lambda w\}, & \text{if } y_i w^\top x_i < 1. \end{cases} \tag{59}$$

Subgradient descent. We know that the gradient always points to the direction of fastest ascent, but this is not necessarily the case for subgradients. If we move along the opposite direction of some subgradient in each step, this can increase the objective, but in the long term, we will get closer to the minimizer if f is convex and the learning rate is small enough. The method of **stochastic subgradient descent** is described as in Algorithm 6. Specifically, we take the following subgradient from the subdifferential computed as in (59):

$$\mathbb{1}_{y_i w^\top x_i < 1}(-y_i x_i) + 2\lambda w, \tag{60}$$

so the updating step can be expressed as

$$w \leftarrow w + \eta \mathbb{1}_{y_i w^\top x_i < 1}(y_i x_i) - 2\eta\lambda w = (1 - 2\eta\lambda) w + \eta \mathbb{1}_{y_i w^\top x_i < 1}(y_i x_i). \tag{61}$$

Algorithm 6 Stochastic Subgradient Descent (Pegasos)

Initialize $w \leftarrow 0$, $t \leftarrow 0$;
repeat
  Randomly shuffle the data $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^n$;
  for $j = 1, \cdots, n$ **do**
    $t \leftarrow t + 1$, $\eta \leftarrow 1/\lambda t$;
    if $y_j w^\top x_j < 1$ **then**
      $w \leftarrow (1 - 2\eta\lambda)w + \eta y_j x_j$;
    else
      $w \leftarrow (1 - 2\eta\lambda)w$;
    end if
  end for
until the stopping criterion is satisfied;

Note that subgradients do not necessarily converge to as we get closer to the minimizer w^*, and that is why we used decreasing step sizes (*e.g.*, $O(t^{-1})$ or $O(t^{-1/2})$). Also, subgradient methods are much slower than gradient descent, *i.e.*, $O(\epsilon^{-2})$ versus $O(\epsilon^{-1})$ for convex functions.

5.3 The Dual Problem

In addition to subgradient descent, we can directly solve the optimization problem using a quadratic programming solver. Recall that the SVM optimization problem is equivalent to

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i, \\ \text{subject to} \quad & \xi_i \geq 1 - y_i(w^\top x_i + b), \qquad \forall i, \\ & \xi_i \geq 0, \qquad \forall i, \end{aligned} \tag{62}$$

where the objective function is differentiable, with $n + d + 1$ unknowns and $2n$ affine constraints.

Why we care about the dual. The general (inequality-constrained) optimization problem can be formulated as

$$\begin{aligned} \text{minimize} \quad & f_0(x), \\ \text{subject to} \quad & f_i(x) \leq 0, \qquad \forall i \in \{1, \cdots, m\}. \end{aligned} \tag{63}$$

The **Lagrangian** for this optimization problem is

$$L(x, \lambda) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x), \tag{64}$$

where λ_i are called the **Lagrange multipliers** (also called the **dual variables**). The **Lagrange dual function** is then defined as

$$g(\lambda) = \inf_x L(x, \lambda), \tag{65}$$

which is concave and satisfies the **lower bound property**: if $\lambda \geq 0$, then $g(\lambda) \leq p^*$, where p^* is the optimal value of the optimization problem. Note however that $g(\lambda)$ could be $-\infty$, implying an uninformative lower bound. Therefore, for any **primal form** optimization problem as in (63), there is a recipe for constructing a corresponding **Lagrangian dual problem**, that is,

$$\begin{aligned} \text{maximize} \quad & g(\lambda), \\ \text{subject to} \quad & \lambda_i \geq 0, \qquad \forall i \in \{1, \cdots, m\}. \end{aligned} \tag{66}$$

Note that this is still a convex optimization problem. Moreover, the dual variables often have interesting and relevant interpretations, and provide certificates for optimality. Denote d^* as the optimal value for the dual problem. We always have $p^* \geq d^*$ which is called weak duality, and if $p^* = d^*$, we say that the problems have strong duality, and this is fairly typical for convex problems.

The SVM dual problem. Recall the SVM optimization problem, and we can rewrite the constraint to that it can be formulated as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i, \\ \text{subject to} \quad & 1 - y_i(w^\top x_i + b) - \xi_i \le 0, \qquad \forall i, \\ & -\xi_i \le 0, \qquad \forall i, \end{aligned} \tag{67}$$

The dual problem of the SVM optimization problem can be formulated as

$$\begin{aligned} \text{maximize} \quad & \inf_{w,b,\xi}\left(\frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i + \sum_{i=1}^n \alpha_i(1 - y_i(w^\top x_i + b) - \xi_i) + \sum_{i=1}^n \lambda_i(-\xi_i)\right), \\ \text{subject to} \quad & \alpha_i \ge 0, \qquad \forall i, \\ & \lambda_i \ge 0, \qquad \forall i, \end{aligned} \tag{68}$$

which can be rewritten as

$$\begin{aligned} \text{maximize} \quad & \inf_{w,b,\xi}\left(\frac{1}{2}w^\top w + \sum_{i=1}^n \xi_i\left(\frac{C}{n} - \alpha_i - \lambda_i\right) + \sum_{i=1}^n \alpha_i(1 - y_i(w^\top x_i + b))\right) =: \inf_{w,b,\xi} L(w,b,\xi,\alpha,\lambda), \\ \text{subject to} \quad & \alpha_i \ge 0, \qquad \forall i, \\ & \lambda_i \ge 0, \qquad \forall i. \end{aligned} \tag{69}$$

Note that

$$\partial_w L = 0 \iff w - \sum_{i=1}^n \alpha_i y_i x_i = 0 \iff w = \sum_{i=1}^n \alpha_i y_i x_i, \tag{70}$$

$$\partial_b L = 0 \iff -\sum_{i=1}^n \alpha_i y_i = 0 \iff \sum_{i=1}^n \alpha_i y_i = 0, \tag{71}$$

$$\partial_{\xi_i} L = 0 \iff \frac{C}{n} - \alpha_i - \lambda_i = 0 \iff \alpha_i + \lambda_i = \frac{C}{n}, \qquad \forall i. \tag{72}$$

Substituting these conditions, we have that

$$L(w,b,\xi,\alpha,\lambda) = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j + \sum_{i=1}^n \alpha_i - \sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j = \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j. \tag{73}$$

Therefore, we can write that

$$\inf_{w,b,\xi} L(w,b,\xi,\alpha,\lambda) = \begin{cases} \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j, & \text{if } \sum_{i=1}^n \alpha_i y_i = 0, \text{ and } \alpha_i + \lambda_i = \frac{C}{n}, \forall i, \\ -\infty, & \text{otherwise.} \end{cases} \tag{74}$$

The dual problem can thus be finally reformulated as

$$\sup_{\alpha,\lambda}\left(\sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^\top x_j\right), \tag{75}$$

such that

$$\sum_{i=1}^n \alpha_i y_i = 0, \qquad \alpha_i \in \left[0, \frac{C}{n}\right], \ \forall i. \tag{76}$$

Insights from the dual problem. Given the solution α^* to the dual problem, the primal solution would be $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$ by (70). Note that $\alpha_i^* \in [0, C/n]$, so that C controls the maximum weight on each example. Given that the SVM dual problem has strong duality, we must have that

$$\alpha_i^*(1 - y_i f^*(x_i) - \xi_i^*) = 0, \quad \lambda_i^*\xi_i^* = \left(\frac{C}{n} - \alpha_i^*\right)\xi_i^* = 0, \tag{77}$$

since those are the terms that give the complementary slackness in the dual problem, and recall that $\lambda_i^* + \alpha_i^* = C/n$. Now recall that $\xi_i^* = \max\{0, 1 - y_i f^*(x_i)\}$ is the hinge loss on (x_i, y_i), so we can observe the following:

- If $y_i f^*(x_i) > 1$, then the margin loss is $\xi_i^* = 0$, so we must have $\alpha_i^* = 0$ by the left-hand equation of (77).
- If $y_i f^*(x_i) < 1$, then the margin loss is $\xi_i^* > 0$, so we must have $\alpha_i = C/n$ by the right-hand equation of (77).
- If $\alpha_i^* = 0$, then $\xi_i^* = 0$ by the right-hand equation of (77), which implies no loss, so $y_i f^*(x_i) \geq 1$.
- If $\alpha_i^* \in (0, C/n)$, then $\xi_i = 0$ by the right-hand equation of (77), and further $1 - y_i f^*(x_i) = 0$ by the left-hand equation of (77).
- If $\alpha_i^* = C/n$, then $\xi_i^* = 1 - y_i f^*(x_i)$ by the left-hand equation of (77), which implies $y_i f^*(x_i) \leq 1$.

To summarize, the relations between the margin and the example weights α_i are

$$\alpha_i^* = 0 \implies y_i f^*(x_i) \geq 1, \tag{78}$$
$$0 < \alpha_i^* < C/n \implies y_i f^*(x_i) = 1, \tag{79}$$
$$\alpha_i^* = C/n \implies y_i f^*(x_i) \leq 1, \tag{80}$$

and

$$y_i f^*(x_i) < 1 \implies \alpha_i^* = C/n, \tag{81}$$
$$y_i f^*(x_i) = 1 \implies \alpha_i^* \in [0, C/n], \tag{82}$$
$$y_i f^*(x_i) > 1 \implies \alpha_i^* = 0. \tag{83}$$

The x_i corresponding to $\alpha_i^* > 0$ are called the **support vectors**, and few margin errors or "on the margin" examples implies the sparsity in the input examples. WHY?

2/21 Lecture

6 Kernel Methods

6.1 Feature Maps

Our general learning theory setup makes no assumptions on the input space $\mathscr{X}$, but we require $\mathscr{X} = \mathbb{R}^d$ for the specific methods we have developed, including ridge regression, Lasso regression, and SVMs. Our hypothesis space for all of these was affine functions on $\mathbb{R}^d$, *i.e.*,

$$\mathscr{F} = \left\{ x \mapsto w^\top x + b;\ w \in \mathbb{R}^d, b \in \mathbb{R} \right\}. \tag{84}$$

However, in many occasions we want to use inputs not natively in $\mathbb{R}^d$, for instance text documents, image files, sound recordings, DNA sequences, etc. However, everything in a computer is a sequence of numbers. The ith entry of each sequence should have the same "meaning".

Definition 6.1. Mapping an input from $\mathscr{X}$ to a vector in $\mathbb{R}^d$ is called **feature extraction** or **featurization**.

We use a **feature map** $\phi : \mathscr{X} \to \mathbb{R}^d$ to map into the **feature space** $\mathbb{R}^d$, and use the same hypothesis space as before on the feature space. For linear models, to grow the hypothesis space, we must add features. Sometimes we say that a larger hypothesis space is more expressive, *i.e.*, it can fit more relationships between input and action.

Handling nonlinearity with linear models. For linear predictors, it is important how features are added. The relation between a feature and the label may not be linear. Three types of nonlinearities can cause problems, which include non-monotonicity, saturation, and interactions between features.

- **Non-monotonicity:** Let the feature map $\phi(x) = (1, \texttt{temperature}(x))$. The action is to predict health score $y \in \mathbb{R}$ (positive is good), and the hypothesis $\mathscr{F}$ is the collection of all affine functions of temperature. The issue is, health is not an affine function of temperature, since both very high and very low temperatures are bad. One solution is to say $\phi(x) = (1, (\texttt{temperature}(x) - 37)^2)$, where 37 is the "normal" temperature is Celcius. However this may require manually-specified domain knowledge. An easier way to do this is to put $\phi(x) = [1, \texttt{temperature}(x), \texttt{temperature}(x)^2]$. This is in fact more **expressive** than the previous solution. **The general rule is, features should be simple building blocks that can be pieced together.**

- **Saturation:** Suppose we are finding products relevant to a user's query. The input will be product x, and the action is to score the relevance of x to the user's query. Now let the feature map be $\phi(x) = (1, N(x))$, where $N(x)$ denotes the number of people who bought x. We expect a monotonic relationship between $N(x)$ and the relevance, but we also expect **diminishing return**. To achieve this, we can use smooth nonlinear transformations such as $\phi(x) = (1, \log(1 + N(x)))$. Discretization can be another solution, for instance, $\phi(x) = (\mathbb{1}_{0 \le N(x) < 10}, \mathbb{1}_{10 \le N(x) < 100}, \cdots)$.
- **Interactions:** Say we take patient information x and score the health status $y \in \mathbb{R}$ (higher is better). Let the feature map be $\phi(x) = (\texttt{height}(x), \texttt{weight}(x))$. The issue here is, we need some interaction between height and weight, since the weight relative to the height is important. One way to do this is to use the "ideal weight" formula $\texttt{weight}(x) = 52 + 1.9(\texttt{height}(x) - 60)$, but again this requires additional information. A stupid but more flexible way is to simply include all the second order features, such that $\phi(x) = (1, \texttt{height}(x), \texttt{weight}(x), \texttt{height}(x)^2, \texttt{weight}(x)^2, \texttt{height}(x) \cdot \texttt{weight}(x))$. **The general principle is, simpler building blocks can replace a single "smart" feature.**

If we want to deal well with the three previously mentioned issues, we will need large feature spaces. Especially, interaction terms are useful building blocks to model nonlinearities in features, but dealing with high dimensions can lead to huge number of cross terms. Very large feature spaces then have two potential issues, including the overfitting problem and the high memory and computational costs. The solution to overfitting is simple: we just handle with regularization. The solution to high cost is trickier: we will use **kernel methods**, as we will discuss later.

6.2 The Kernel Trick

Let $\psi : \mathscr{X} \to \mathbb{R}^d$ be a feature map. Recall that the SVM objective (with an explicit feature map) can be written as

$$\lim_{w \in \mathbb{R}^d} \left(\frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \max\left\{0, 1 - y_i w^\top \psi(x_i)\right\} \right). \tag{85}$$

The computation is costly if d is large, for instance, with high-degree monomials. Now recall that by Lagrangian duality, this is equivalent to solving the dual problem

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \psi(x_i)^\top \psi(x_j), \tag{86}$$

such that

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \qquad \alpha_i \in \left[0, \frac{C}{n}\right], \ \forall i. \tag{87}$$

If α^* is an optimal value, then

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i \psi(x_i), \qquad \hat{f}(x) = \sum_{i=1}^{n} \alpha_i^* y_i \psi(x_i)^\top \psi(x_j). \tag{88}$$

A key observation here is that $\psi(x)$ only shows up in inner products with some other $\psi(x')$ for both training and inference. Therefore, we introduce **degree-2 monomials** using $\psi : \mathbb{R}^2 \to \mathbb{R}^3$, such that $\psi(x_1, x_2) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$. The inner product is then

$$\psi(x)^\top \psi(x') = x_1^2 x_1'^2 + (\sqrt{2} x_1 x_2)(\sqrt{2} x_1' x_2') + x_2^2 x_2'^2 = (x_1 x_1' + x_2 x_2')^2 = (x^\top x')^2. \tag{89}$$

Consequently, we can calculate the inner product $\psi(x)^\top \psi(x')$ in the original input space without even accessing the features $\psi(x)$. Now, consider monomials up to degree-2, such that $(x_1, x_2) \mapsto (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2)$. The inner product can be computed as

$$\psi(x)^\top \psi(x') = 1 + 2x_1 x_1' + 2x_2 x_2' + x_1^2 x_1'^2 + 2x_1 x_2 x_1' x_2' + x_2^2 x_2'^2 = (1 + x_1 x_1' + x_2 x_2')^2 = (1 + x^\top x')^2. \tag{90}$$

More generally, for feature maps producing monomials up to degree-p, we have that

$$\psi(x)^\top \psi(x') = (1 + x^\top x')^p. \tag{91}$$

Using this **kernel trick**, we can do implicit computation in $O(d)$ rather than doing explicit computation on features which takes $O(d^p)$. Next we will formalize these observations.

Kernel function. Let $\mathscr{X}$ be the input space and $\mathscr{H}$ be the feature space, which is a Hilbert space such as $\mathbb{R}^d$. For the feature map $\psi : \mathscr{X} \to \mathscr{H}$, the corresponding **kernel function** is

$$k(x, x') = \langle \psi(x), \psi(x') \rangle , \tag{92}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product associated with $\mathscr{H}$. The reason why we introduce this kernel function notation is that, we can sometimes evaluate $k(x, x')$ without explicitly computing $\psi(x)$ and $\psi(x')$. We will formalize the occasions where we can use this kernel function trick to improve computational complexity.

Definition 6.2. A method is **kernelized** if every feature vector $\psi(x)$ only appears inside an inner product with another feature vector $\psi(x')$. This applies to both the optimization problem and the prediction function.

Definition 6.3. The **kernel matrix** for a kernel k on $x_1, \cdots, x_n \in \mathscr{X}$ is

$$K = (k(x_i, x_j))_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \in \mathbb{R}^{n \times n}. \tag{93}$$

In machine learning, this is also called a **Gram matrix**, but traditionally (in linear algebra), Gram matrices are defined without reference to a kernel or feature map. The kernel matrix summarizes all the information we need about the training inputs $x_1, \cdots, x_n$ to solve a kernelized optimization problem. In the kernelized SVM, for instance, we can replace $\psi(x_i)^\top \psi(x_j)$ with K_{ij}, so as to obtain the alternative form of the dual problem

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij}, \tag{94}$$

such that

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \qquad \alpha_i \in \left[0, \frac{C}{n}\right], \ \forall i. \tag{95}$$

Furthermore, if α^* is an optimal value, then

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i \psi(x_i), \qquad \hat{f}(x) = \sum_{i=1}^{n} \alpha_i^* y_i K_{ij}. \tag{96}$$

The kernel trick is especially useful when $d \gg n$, but note that computing the kernel matrix may still depends on d and the essence of the trick is getting around this $O(d)$ dependence.

6.3 Example Kernels

One way to get a kernel is to explicitly construct $\psi(x) : \mathscr{X} \to \mathbb{R}^d$ (for instance, monomials) and define $k(x, x') = \psi(x)^\top \psi(x')$. Another way is to directly define the kernel function $k(x, x')$ and verify that it corresponds to $\langle \psi(x), \psi(x') \rangle$ for some ψ.Note that it is always useful to think of the kernel $k(x, x')$ as a **similarity score** for x and x'. There are many theorems to help us with the second approach.

Positive semidefinite kernels. Recall that in linear algebra, a real, symmetric matrix $M \in \mathbb{R}^{n \times n}$ is **positive semidefinite** if for any $x \in \mathbb{R}^n$, we have that $x^\top M x \geq 0$. Moreover, the following conditions are each necessary and sufficient for a symmetric matrix M ot be positive semidefinite.

- M can be factorized as $M = R^\top R$ for some matrix R.
- All eigenvalues of M are nonnegative.

Definition 6.4. A symmetric function $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ is a **positive definite** kernel on $\mathscr{X}$ if for any finite set $\{x_1, \cdots, x_n\} \in \mathscr{X}$, the kernel matrix on this set

$$K = (k(x_i, x_j))_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}, \tag{97}$$

is a positive semidefinite matrix.

Note that symmetry in this definition means that $k(x, x') = k(x', x)$ for any $x, x' \in \mathscr{X}$. An equivalent definition is that for any finite set $\{x_1, \cdots, x_n\} \in \mathscr{X}$, we have that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \geq 0, \tag{98}$$

given $\alpha_i \in \mathbb{R}$ for all $i \in \{1, \cdots, n\}$. Now let us move on to see why we are interested in positive definite kernels.

Theorem 6.5 (Mercer's Theorem)**.** A symmetric function $k(x, x')$ can be expressed as an inner product

$$k(x, x') = \langle \psi(x), \psi(x') \rangle \tag{99}$$

for some feature map ψ if and only if $k(x, x')$ is positive semidefinite.

Generating new kernels. Though given Mercer's theorem, proving that a kernel function is positive semidefinite is typically not easy. However, we can construct new kernels from valid kernels. Suppose that $k, k_1, k_2 : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ are positive definite kernels, then the following are also positive definite kernels.

- Nonnegative scaling: $\tilde{k}(x, x') = \alpha k(x, x')$ for $\alpha \geq 0$.
- Sum: $\tilde{k}(x, x') = k_1(x, x') + k_2(x, x')$.
- Product: $\tilde{k}(x, x') = k_1(x, x') k_2(x, x')$.
- Recursion: $\tilde{k}(x, x') = k(\psi(x), \psi(x'))$ for any function ψ.
- Transformation: $\tilde{k}(x, x') = f(x) f(x')$ for any 1-dimensional feature map f.

Common kernels.

- **Linear kernel:** The input space is $\mathscr{X} = \mathbb{R}^d$, and the feature space is $\mathscr{H} = \mathbb{R}^d$ with standard inner product. Consider the feature map $\psi(x) = x$, then the kernel is $k(x, x') = x^\top x'$.
- **Quadratic kernel:** The input space is $\mathscr{X} = \mathbb{R}^d$, and the feature space is $\mathscr{H} = \mathbb{R}^D$, where $D = d + \binom{d}{2}$. Consider the feature map

  $$\psi(x) = (x_1, \cdots, x_d, x_1^2, \cdots, x_d^2, \sqrt{2}x_1x_2, \cdots, \sqrt{2}x_i x_j, \cdots, \sqrt{2}x_{d-1}x_d), \tag{100}$$

  then for any $x, x' \in \mathbb{R}^d$, we have that

  $$k(x, x') = \langle \psi(x), \psi(x') \rangle = \langle x, x' \rangle + \langle x, x' \rangle^2. \tag{101}$$

  The computation cost for inner product with explicit mapping would be $O(d^2)$ since $D = d + \binom{d}{2} \approx d^2/2$, but the computation cost for implicit kernel calculation is just $O(d)$.
- **Polynomial kernel (in $\mathbb{R}^d$):** The input space is $\mathscr{X} = \mathbb{R}^d$, and consider a feature map with all monomials up to degree M. An example is (90) for the SVM objective. The kernel function would then be

  $$k(x, x') = (1 + \langle x, x' \rangle)^M. \tag{102}$$

  The computation cost for implicit kernel calculation is the same for any M, but the cost of explicit inner product computation grows rapidly in M, so using kernel tricks would be a great improvement to computational complexity especially when M is large (*i.e.*, the feature space is large).
- **Radial Basis Function (RBF) / Gaussian Kernel:** The input space is $\mathscr{X} = \mathbb{R}^d$, and the RBF (Gaussian) kernel is defined as

  $$k(x, x') = \exp\left(\frac{-\|x - x'\|^2}{2\sigma^2} \right), \tag{103}$$

  where σ^2 is known as the bandwidth parameter. This is probably the most common nonlinear kernel to use. We will go back to this later.

Some Linear Algebra: Inner Product Spaces and Hilbert Spaces

Definition 6.6. An **inner product space** over $\mathbb{R}$ is a vector space $\mathscr{V}$ with an **inner product**, defined as a mapping

$$\langle \cdot, \cdot \rangle : \mathscr{V} \times \mathscr{V} \to \mathbb{R}, \tag{104}$$

which has the following properties: for all $x, y, z \in \mathscr{V}$ and $a, b \in \mathbb{R}$, we have that

- Symmetry: $\langle x, y \rangle = \langle y, x \rangle$.
- Linearity: $\langle ax + by, z \rangle = a \langle x, z \rangle + b \langle y, z \rangle$.
- Positive-definiteness: $\langle x, x \rangle \geq 0$, and $\langle x, x \rangle = 0$ if and only if $x = 0_{\mathscr{V}}$.

Inner products are nice because that give us notions of "size", "distance", and "angle" in the vector space. For an inner product space, we can define a norm as $\|x\| = \sqrt{\langle x, x \rangle}$.

Definition 6.7. Two vectors x and x' are **orthogonal** if $\langle x, x' \rangle = 0$, which we denote by $x \perp x'$. Moreover, a vector x is **orthogonal** to a set S if $x \perp s$ for all $s \in S$, which we denote by $x \perp S$.

Theorem 6.8 (Pythagorean Theorem)**.** If $x \perp x'$, then $\|x + x'\|^2 = \|x\|^2 + \|x'\|^2$.

Proof. For any vectors $x \perp x'$, we have that $\langle x, x' \rangle = \langle x', x \rangle = 0$. Therefore, we can compute that

$$\|x + x'\|^2 = \langle x + x', x + x' \rangle = \langle x, x \rangle + \langle x, x' \rangle + \langle x', x \rangle + \langle x', x' \rangle = \langle x, x \rangle + \langle x', x' \rangle = \|x\|^2 + \|x'\|^2, \tag{105}$$

as desired. □

Definition 6.9. A **Hilbert space** is a complete inner product space.

For instance, any finite dimensional inner product space is a Hilbert space.

6.4 The Representer Theorem

Recall the SVM dual problem, where given the dual solution α^*, the primal solution is $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$, a linear combination of training inputs $x_1, \cdots, x_n$. In mathematics, we denote $w^* \in \text{span}(x_1, \cdots, x_n)$. We also recall the ridge regression solution for $\lambda > 0$, which is

$$w^* = \arg\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (w^\top x_i - y_i)^2 + \lambda \|w\|_2^2. \tag{106}$$

This has a closed form solution

$$w^* = (X^\top X + \lambda I)^{-1} X^\top y, \tag{107}$$

where X is the design matrix with $x_1, \cdots, x_n$ as rows. With some rearrangement of terms, we can write that

$$w^* = X^\top \underbrace{\left(\frac{1}{\lambda} y - \frac{1}{\lambda} X w^* \right)}_{=:\alpha^*} = X^\top \alpha^* = \sum_{i=1}^n \alpha_i^* x_i, \tag{108}$$

so again $w^* \in \text{span}(x_1, \cdots, x_n)$. Therefore, rather than minimizing over all of $w \in \mathbb{R}^d$, we can instead minimize over $\text{span}(x_1, \cdots, x_n)$, such that

$$w^* = \arg\min_{w \in \text{span}(x_1, \cdots, x_n)} \frac{1}{n} \sum_{i=1}^n (w^\top x_i - y_i)^2 + \lambda \|w\|_2^2. \tag{109}$$

This is equivalent to finding

$$\alpha^* = \arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n ((X^\top \alpha)^\top x_i - y_i)^2 + \lambda \left\| X^\top \alpha \right\|_2^2, \tag{110}$$

and to retrieve w^* from this reparametrized optimization problem, we just need to take $w^* = X^\top \alpha^*$. Also note that we have changed the dimension of our optimization variable from d to n. Consider the situation where we have very large dimensional feature space (for instance, using high-order monomial interaction terms as features). Within this reparametrization, we can then reduce the computational complexity significantly since $d \gg n$. To this end, for SVM and ridge regression, we have found that the solution is in the span of the data. In the rest of this section we will introduce the **representer theorem**, which shows that this "span of data" result occurs far more generally.

Generalized objective. We consider the generalized objective

$$\min_{w \in H} R(\|w\|) + L(\langle w, x_1 \rangle, \cdots, \langle w, x_n \rangle), \tag{111}$$

where $w, x_1, \cdots, x_n \in \mathscr{H}$ for some Hilbert space $\mathscr{H}$ (typically $\mathbb{R}^d$), $\|\cdot\|$ denotes the norm corresponding to the inner product on $\mathscr{H}$, $R : [0, \infty) \to \mathbb{R}$ is a non-decreasing **regularization term**, and $L : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary **loss term**. For instance, in the SVM objective,

$$R : x \mapsto \frac{1}{2}x^2, \qquad L : (\xi_1, \cdots, \xi_n) \mapsto \frac{C}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \xi_i\} \tag{112}$$

As another example, in ridge regression,

$$R : x \mapsto \lambda x^2, \qquad L : (\xi_1, \cdots, \xi_n) \mapsto \frac{1}{n} \sum_{i=1}^{n} (\xi_i - y_i)^2. \tag{113}$$

Now we introduce the representer theorem as follows.

Theorem 6.10 (Representer Theorem)**.** Let J be the generalized objective defined as above, then it has a minimizer of the form

$$w^* = \sum_{i=1}^{n} \alpha_i x_i. \tag{114}$$

Given the representer theorem, just as before, we can look for w^* only in the span of the data, so that

$$w^* = \min_{w \in \text{span}(x_1, \cdots, x_n)} R(\|w\|) + L(\langle w, x_1 \rangle, \cdots, \langle w, x_n \rangle). \tag{115}$$

Again, parametrizing as before, we have that

$$\alpha^* = \arg\min_{\alpha \in \mathbb{R}^n} R\left(\left\|\sum_{i=1}^{n} \alpha_i x_i\right\|\right) + L\left(\left\langle \sum_{i=1}^{n} \alpha_i x_i, x_1 \right\rangle, \cdots, \left\langle \sum_{i=1}^{n} \alpha_i x_i, x_n \right\rangle\right). \tag{116}$$

Reparametrizing the generalized objective. Define the **training score** function $s : \mathbb{R}^d \to \mathbb{R}$ by

$$s(w) = \begin{pmatrix} \langle w, x_1 \rangle \\ \vdots \\ \langle w, x_n \rangle \end{pmatrix}, \tag{117}$$

We can then rewrite the generalized objective function as

$$J(w) = R(\|w\|) + L(s(w)). \tag{118}$$

By the representer theorem, it is sufficient to minimize $J(w)$ for $w \in \text{span}(x_1, \cdots, x_n)$, so that we can simply minimize

$$J_0(\alpha) = R\left(\underbrace{\left\|\sum_{i=1}^{n} \alpha_i x_i\right\|}_{\text{norm piece}}\right) + L\left(\underbrace{s\left(\sum_{i=1}^{n} \alpha_i x_i\right)}_{\text{score piece}}\right), \qquad \alpha \in \mathbb{R}^n. \tag{119}$$

For the norm piece, we have that

$$\left\|\sum_{i=1}^{n} \alpha_i x_i\right\|^2 = \left\langle \sum_{i=1}^{n} \alpha_i x_i, \sum_{j=1}^{n} \alpha_j x_j \right\rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle = \alpha^\top K \alpha. \tag{120}$$

For the score piece, we have that

$$s\left(\sum_{i=1}^{n}\alpha_i x_i\right)=\begin{pmatrix}\langle\sum_{i=1}^{n}\alpha_i x_i,x_1\rangle\\ \vdots\\ \langle\sum_{i=1}^{n}\alpha_i x_i,x_n\rangle\end{pmatrix}=\begin{pmatrix}\sum_{i=1}^{n}\alpha_i\langle x_i,x_1\rangle\\ \vdots\\ \sum_{i=1}^{n}\alpha_i\langle x_i,x_n\rangle\end{pmatrix}=\begin{pmatrix}\langle x_1,x_1\rangle & \cdots & \langle x_1,x_n\rangle\\ \vdots & \ddots & \vdots\\ \langle x_n,x_1\rangle & \cdots & \langle x_n,x_n\rangle\end{pmatrix}\begin{pmatrix}\alpha_1\\ \vdots\\ \alpha_n\end{pmatrix}=K\alpha. \tag{121}$$

Putting the two pieces together, we can rewrite the reparametrized generalized objective as

$$J_0(\alpha)=R\left(\sqrt{\alpha^\top K\alpha}\right)+L(K\alpha),\qquad \alpha\in\mathbb{R}^n. \tag{122}$$

Again, note that all information needed about the training input $x_1,\cdots,x_n$ is summarized in the Gram matrix K, and we are now minimizing over $\mathbb{R}^n$ rather than $\mathbb{R}^d$. If $d\gg n$, then this can reduce the computational complexity a lot. Finally to make a prediction, we just need to compute

$$\hat{f}(x)=\langle w^*,x\rangle=\left\langle\sum_{i=1}^{n}\alpha_i^* x_i,x\right\rangle=\sum_{i=1}^{n}\alpha_i^*\langle x_i,x\rangle. \tag{123}$$

If x is among the training input $x_1,\cdots,x_n$, then again we do not need any other information aside from K. If we need to make a new prediction, however, we may need to touch all the training inputs $x_1,\cdots,x_n$. To do this, define for any $x\in\mathscr{H}$ that

$$k_x=\begin{pmatrix}\langle x_1,x\rangle\\ \vdots\\ \langle x_n,x\rangle\end{pmatrix}, \tag{124}$$

so we can write $\hat{f}(x)=k_x^\top\alpha^*$.

Brief Summary

Recall that our original plan is to find

$$w^*\in\arg\min_{w\in H}\ R(\|w\|)+L(\langle w,x_1\rangle,\cdots,\langle w,x_n\rangle), \tag{125}$$

and our prediction for x is done by calculating $\hat{f}(x)=\langle w^*,x\rangle$. Now we have proved that the following is equivalent: we first find

$$\alpha^*\in\arg\min_{\alpha\in\mathbb{R}^n}R\left(\sqrt{\alpha^\top K\alpha}\right)+L(K\alpha), \tag{126}$$

and predict via $\hat{f}(x)=k_x^\top\alpha^*$, where

$$K=\begin{pmatrix}\langle x_1,x_1\rangle & \cdots & \langle x_1,x_n\rangle\\ \vdots & \ddots & \vdots\\ \langle x_n,x_1\rangle & \cdots & \langle x_n,x_n\rangle\end{pmatrix},\qquad k_x=\begin{pmatrix}\langle x_1,x\rangle\\ \vdots\\ \langle x_n,x\rangle\end{pmatrix}. \tag{127}$$

Also, remember our new recipe for optimizing the model and making the prediction.

- Recognize the kernelized problem: $\psi(x)$ occurs only in the inner products $\psi(x)^\top\psi(x')$.
- Choose a kernel function (recall that it is often comprehended as the "similarity score").
- Compute the kernel matrix.
- Optimize the model and make predictions by accessing the kernel matrix.

2/28 Lecture

Midterm. Also see announcement on Brightspace.

- Date and time: Mar 7, 2023, 4:55 pm – 6:35 pm (EST).
- Coverage: up to kernel methods (not including this week).
- Review: this week's lab.
- Difficulty: easier than last year.

7 Probabilistic Modeling

Probabilistic modeling is a unified framework that covers many models, including but not limited to linear regression, logistic regression, etc. In probabilistic models, we are learning as **statistical inference**, providing principled ways to incorporate belief on the data generating distribution (inductive biases). There are mainly two ways to model how the data is generated, that is, **conditional models** $\mathbb{P}[y|x]$ and **generative models** $\mathbb{P}[x, y]$. We will use maximum likelihood estimation to estimate the parameters of the models, and also compare and contrast between these models.

7.1 Conditional Models

7.1.1 Linear Regression

Linear regression is one of the most important methods in machine learning and statistics. The goal is to predict a real-valued target y (also called the **response**) from a vector of features x (also called **covariates**). DATA: Let the training examples be denoted by $\mathscr{D} = \left\{(x^{(n)}, y^{(n)}\right\}_{n=1}^{N}$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Note that we are using superscripts for indices of examples and subscripts for indices of dimensions. MODEL: We use a linear function h (parametrized by θ) to predict y from x, such that

$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^\top x, \tag{128}$$

where $\theta \in \mathbb{R}^d$ are the parameters (also called the **weights**). Note that we have incorporated the **bias term** (also called the **intercept term**) into x, *i.e.*, $x_0 = 1$. LOSS FUNCTION: We then estimate θ by minimizing the square loss (the least squares method), such that

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \left(y^{(n)} - \theta^\top x^{(n)}\right)^2. \tag{129}$$

Note that this loss function is the empirical loss. MATRIX FORM: Now let $X \in \mathbb{R}^{N \times d}$ be the design matrix whose rows are the input features, and let $y \in \mathbb{R}^N$ be the vector of all targets. We want to solve

$$\hat{\theta} = \arg\min_\theta (X\theta - y)^\top (X\theta - y). \tag{130}$$

SOLUTION: Note that there is a closed-form solution to this objective, that is, $\hat{\theta} = (X^\top X)^{-1} X^\top y$. The above is a brief summary of linear regression that we have deduced before. Now we are going to derive linear regression from a probabilistic modeling perspective. Suppose that x and y are related through a linear function

$$y = \theta^\top x + \epsilon, \tag{131}$$

where ϵ is the **residual error** capturing all unmodeled effects (*e.g.*, noise). The errors are distributed identically and independently, such that

$$\epsilon \in \mathscr{N}(0, \sigma^2). \tag{132}$$

The distribution of Y given $X = x$ is then

$$\mathbb{P}[y|x; \theta] = \mathscr{N}(\theta^\top x, \sigma^2). \tag{133}$$

We can imagine this as putting a Gaussian bump around the output of the linear predictor. Given a probabilistic model and a dataset $\mathscr{D}$ in which all examples are independent and identically distributed, the **maximum likelihood principle** says that we should maximize the (conditional) likelihood of the data, which is defined by

$$L(\theta) = \mathbb{P}[\mathscr{D}; \theta] = \prod_{n=1}^{N} \mathbb{P}[y^{(n)}|x^{(n)}; \theta]. \tag{134}$$

In practice, we maximize the **log likelihood** $l(\theta)$ (since logarithm can convert products into sums), or equivalently, minimize the negative log likelihood. Let us find the maximum likelihood solution for our model. Recall that the

distribution of Y given $X = x$ is given by a normal distribution with mean $\theta^\top x$ and variance σ^2, then

$$\begin{aligned} l(\theta) = \log L(\theta) &= \log \prod_{n=1}^{N} \mathbb{P}[y^{(n)}|x^{(n)};\theta] = \sum_{n=1}^{N} \log \mathbb{P}[y^{(n)}|x^{(n)};\theta] \\ &= \sum_{n=1}^{N} \log \left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y^{(n)} - \theta^\top x^{(n)})^2}{2\sigma^2} \right) \right) = N \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(y^{(n)} - \theta^\top x^{(n)} \right)^2. \end{aligned} \tag{135}$$

In order to find the maximum likelihood solution, we compute the derivatives of the likelihood with respect to each parameter, such that

$$\frac{\partial l}{\partial \theta_i}(\theta) = \frac{1}{\sigma^2} \sum_{n=1}^{N} \left(y^{(n)} - \theta^\top x^{(n)} \right) x_i^{(n)}. \tag{136}$$

7.1.2 Logistic Regression

Consider binary classification where $Y \in \{0, 1\}$. We model $\mathbb{P}[y|x]$ as a **Bernoulli** distribution, such that

$$\mathbb{P}[y|x] = h(x)^y (1 - h(x))^{1-y}. \tag{137}$$

We will need to parametrize $h(x) \in (0, 1)$. To this end, we need a function f to map the linear predictor $\theta^\top x \in \mathbb{R}$ to $(0, 1)$, where we will use the logistic function such that

$$f(\eta) = \frac{1}{1 + e^{-\eta}}. \tag{138}$$

Note that now we have

$$\mathbb{P}[y|x;\theta] = \left(f(\theta^\top x) \right)^y \left(1 - f(\theta^\top x) \right)^{1-y} = \left(\frac{1}{1 + e^{-\theta^\top x}} \right)^y \left(1 - \frac{1}{1 + e^{-\theta^\top x}} \right)^{1-y}, \tag{139}$$

so we can compute the **log odds** as

$$\log \frac{\mathbb{P}[1|x;\theta]}{\mathbb{P}[0|x;\theta]} = \log \frac{\frac{1}{1+e^{-\theta^\top x}}}{1 - \frac{1}{1+e^{-\theta^\top x}}} = \log \frac{1}{e^{-\theta^\top x}} = \log e^{\theta^\top x} = \theta^\top x, \tag{140}$$

which implies a linear decision boundary. As for how to extend to multiclass classification, we will discuss more on that later. Now, similar to linear regression, we estimate θ by maximizing the conditional log likelihood, such that

$$l(\theta) = \sum_{n=1}^{N} \log \mathbb{P}[y^{(n)}|x^{(n)};\theta] = \sum_{n=1}^{N} \left(y^{(n)} \log \left(\frac{1}{1 + e^{-\theta^\top x^{(n)}}} \right) + (1 - y^{(n)}) \log \left(1 - \frac{1}{1 + e^{-\theta^\top x^{(n)}}} \right) \right). \tag{141}$$

Closed-form solutions are not available to this problem, but we can estimate via gradient descent since this conditional log likelihood is concave, and maximizing a concave function is equivalent to minimizing a convex function. We can compute the derivatives of the likelihood with respect to each parameter, such that

$$\begin{aligned} \frac{\partial l}{\partial \theta_i}(\theta) &= \sum_{n=1}^{N} y^{(n)} \frac{\partial}{\partial \theta_i} \left(\log \left(\frac{1}{1 + e^{-\theta^\top x^{(n)}}} \right) \right) + \sum_{n=1}^{N} (1 - y^{(n)}) \frac{\partial}{\partial \theta_i} \left(\log \left(1 - \frac{1}{1 + e^{-\theta^\top x^{(n)}}} \right) \right) \\ &= \sum_{n=1}^{N} y^{(n)} \frac{-e^{-\theta^\top x^{(n)}}}{1 + e^{-\theta^\top x^{(n)}}} (-x_i^{(n)}) + \sum_{n=1}^{N} (1 - y^{(n)}) \frac{1}{1 + e^{-\theta^\top x^{(n)}}} (-x_i^{(n)}) \\ &= \sum_{n=1}^{N} \frac{e^{-\theta^\top x^{(n)}} x_i^{(n)} y^{(n)}}{1 + e^{-\theta^\top x^{(n)}}} + \sum_{n=1}^{N} \frac{y^{(n)} x_i^{(n)} - x_i^{(n)}}{1 + e^{-\theta^\top x^{(n)}}} \\ &= \sum_{n=1}^{N} \frac{(1 + e^{-\theta^\top x^{(n)}}) y^{(n)} x_i^{(n)} - x_i^{(n)}}{1 + e^{-\theta^\top x^{(n)}}} = \sum_{n=1}^{N} \left(y^{(n)} - \frac{1}{1 + e^{-\theta^\top x^{(n)}}} \right) x_i^{(n)}. \end{aligned} \tag{142}$$

7.1.3 Generalized Regression

If we compare the linear regression and the logistic regression as we have just discussed, their gradients actually looked very similar, in that the only difference is the coefficient and the logistic regression has an additional logistic function to map the linear predictor $\theta^\top x$ to $(0,1)$. Moreover, they have the following similarities and differences. COMBINE INPUTS: Both use the linear predictor $\theta^\top x$ for combining inputs. OUTPUT: Linear regression outputs a real number while logistic regression outputs a binary (categorical). CONDITIONAL DISTRIBUTION: Linear regression assumes a Gaussian distribution, while logistic regression assumes a Bernoulli distribution. TRANSFER MAP: Linear regression uses the identity map (*i.e.*, it does not transfer $\theta^\top x$) while logistic regression uses the logistic map to transfer $\theta^\top x$ into $(0,1)$. MEAN: The expectation $\mathbb{E}[y|x;\theta]$ of both regression models is equal to $f(\theta^\top x)$, where f is the transfer map. Given the characteristics above, next we construct a generalized form of regression models.

Generalized regression model. Given input x, we want to predict $\mathbb{P}[y|x]$. For the model, we choose a parametric family of distributions $\mathbb{P}[y;\lambda]$ with parameters $\lambda \in \Lambda$, and choose a transfer map that maps a linear predictor in $\mathbb{R}$ to Λ. In brief, we have that

$$\underbrace{x}_{\in\mathbb{R}^d} \overset{\text{parameters}}{\longrightarrow} \underbrace{\theta^\top x}_{\in\mathbb{R}} \overset{\text{transfer map}}{\longrightarrow} \underbrace{\lambda = f(\theta^\top x)}_{\in\Lambda}. \tag{143}$$

As for the learning step, we use maximum likelihood estimation, such that

$$\hat{\theta} \in \arg\max_\theta \log \mathbb{P}[\mathscr{D};\lambda] = \arg\max_\theta \log \mathbb{P}[\mathscr{D}; f(\theta^\top x)]. \tag{144}$$

For prediction, we will use $x \mapsto f(\theta^\top x)$ to compute the predicted result.

Poisson regression. Say we want to predict the number of people entering a restaurant in New York during lunch time. The output would thus be the set of all nonnegative integers. Therefore, we can model $\mathbb{P}[y|x]$ as a **Poisson** distribution[1], so that

$$\mathbb{P}[y|x] = \frac{\lambda^y e^{-\lambda}}{y!} = \frac{f(\theta^\top x)^y \cdot e^{-f(\theta^\top x)}}{y!}. \tag{145}$$

The conditional log likelihood can thus be computed as

$$l(\theta) = \sum_{n=1}^N \log \mathbb{P}[y^{(n)}|x^{(n)};\theta] = \sum_{n=1}^N \left(y^{(n)} \log f(\theta^\top x^{(n)}) - f(\theta^\top x^{(n)}) - \log(y^{(n)}!) \right). \tag{146}$$

The standard approach is to take the transfer map as $f(\eta) = \exp(\eta)$, so that the conditional log likelihood becomes

$$l(\theta) = \sum_{n=1}^N \left(y^{(n)} \theta^\top x^{(n)} - \exp(\theta^\top x^{(n)}) - \log(y^{(n)}!) \right). \tag{147}$$

Multinomial logistic regression. Say we want to extend the logistic regression to output a classification among multiple classes $\{1, \cdots, k\}$ rather than just binary (two classes). Therefore, instead of Bernoulli distribution, we can use the **categorical** distribution, such that

$$\mathbb{P}[y|x] = \prod_{i=1}^k \lambda_i^{[y=i]}, \tag{148}$$

where $[\texttt{property}]$ denotes the Iverson bracket, which takes 1 if `property` evaluates to be true and 0 otherwise. For this, we will need a predictor $\lambda \in \mathbb{R}^k$, such that its entries add up to 1 (this is our $\Lambda \subseteq \mathbb{R}^k$). In order to do this, we use the **softmax function** as the transfer map, such that

$$f(s_1, \cdots, s_k) = \left(\frac{e^{s_1}}{\sum_{i=1}^k e^{s_i}}, \cdots, \frac{e^{s_k}}{\sum_{i=1}^k e^{s_i}} \right). \tag{149}$$

[1] Given a random variable $Y \in \{0,1,2,\cdots\}$ such that $Y \sim \text{Poisson}(\lambda)$, we have that

$$\mathbb{P}[Y=k] = \frac{\lambda^k e^{-\lambda}}{k!},$$

where $\lambda > 0$ and $\mathbb{E}[Y] = \lambda$.

But then instead of using just $\theta^\top x$ as the linear predictor, we now need $(\theta_1^\top x, \cdots, \theta_k^\top x)$ instead, so that

$$\underbrace{x}_{\in\mathbb{R}^d} \xrightarrow{\text{parameters}} \underbrace{(\theta_1^\top x, \cdots, \theta_k^\top x)}_{\in\mathbb{R}^k} \xrightarrow{\text{transfer map}} \underbrace{\lambda = f(\theta_1^\top x, \cdots, \theta_k^\top x) = \left(\frac{e^{\theta_1^\top x}}{\sum_{i=1}^k e^{\theta_i^\top x}}, \cdots, \frac{e^{\theta_k^\top x}}{\sum_{i=1}^k e^{\theta_i^\top x}}\right)}_{\in\Lambda}. \tag{150}$$

Now we can write that

$$\mathbb{P}[y|x;\theta_1,\cdots,\theta_k] = \prod_{i=1}^k \left(\frac{e^{\theta_i^\top x}}{\sum_{j=1}^k e^{\theta_j^\top x}}\right)^{[y=i]}. \tag{151}$$

7.2 Generative Models

We have learned how to directly map x to y, for instance, using perceptron. We have also just discussed how to model the conditional distribution $\mathbb{P}[y|x]$ using generalized regression models (generalized linear models). Next, we will model the joint distribution $\mathbb{P}[x,y]$, and predict the label for x as $\arg\max_{y\in\mathcal{Y}} \mathbb{P}[x,y]$.

Naive Bayes (NB) models. Let us consider binary text classification (for instance, fake versus genuine review) as a motivating example. We use the **bag-of-words** representation of a document, so that binary $x_i \in \{0,1\}$ represents whether the ith word in our vocabulary exists in the input document. Therefore, the probability of a document x can be expressed as

$$\mathbb{P}[x|y] = \mathbb{P}[x_1,\cdots,x_d|y] = \mathbb{P}[x_1|y]\cdot\mathbb{P}[x_2|y,x_1]\cdot\mathbb{P}[x_3|y,x_1,x_2]\cdots\mathbb{P}[x_d|y,x_{d-1},\cdots,x_1] = \prod_{i=1}^d \mathbb{P}[x_i|y,x_{<i}]. \tag{152}$$

The challenge is, $\mathbb{P}[x_i|y,x_{<i}]$ is hard to model and estimate, especially for large values of i. The solution is to make the **naive Bayes assumption** that features are **conditionally independent** given the label, so that

$$\mathbb{P}[x|y] = \prod_{i=1}^d \mathbb{P}[x_i|y]. \tag{153}$$

For binary x_i, we assume that $\mathbb{P}[x_i|y]$ follows a Bernoulli distribution, that is,

$$\mathbb{P}[x_i=1|y=1] = \theta_{i,1}, \qquad \mathbb{P}[x_i=0|y=1] = 1-\theta_{i,1}, \tag{154}$$

$$\mathbb{P}[x_i=1|y=0] = \theta_{i,0}, \qquad \mathbb{P}[x_i=0|y=0] = 1-\theta_{i,0}. \tag{155}$$

Therefore, we can write that

$$\mathbb{P}[x,y] = \mathbb{P}[x|y]\mathbb{P}[y] = \mathbb{P}[y]\prod_{i=1}^d \mathbb{P}[x_i|y] = \mathbb{P}[y]\prod_{i=1}^d \left(\theta_{i,y}\mathbb{1}_{x_i=1} + (1-\theta_{i,y})\mathbb{1}_{x_i=0}\right). \tag{156}$$

We consider the likelihood of the data $L(\theta) = \prod_{n=1}^N \mathbb{P}[x^{(n)},y^{(n)};\theta]$, as opposed to the conditional likelihood that we have seen in the previous section. Instead of maximizing the likelihood itself, we again maximize the log likelihood, for which the derivative with respect to each parameter $\theta_{j,1}$ can be computed as

$$\begin{aligned}
\frac{\partial l}{\partial\theta_{j,1}}(\theta) &= \frac{\partial}{\partial\theta_{j,1}}\sum_{n=1}^N \left(\log\mathbb{P}[y^{(n)};\theta] + \sum_{i=1}^d \log\left(\theta_{i,y^{(n)}}\mathbb{1}_{x_i^{(n)}=1} + (1-\theta_{i,y^{(n)}})\mathbb{1}_{x_i^{(n)}=0}\right)\right) \\
&= \frac{\partial}{\partial\theta_{j,1}}\sum_{n=1}^N \log\left(\theta_{j,y^{(n)}}\mathbb{1}_{x_j^{(n)}=1} + (1-\theta_{j,y^{(n)}})\mathbb{1}_{x_j^{(n)}=0}\right) \\
&= \frac{\partial}{\partial\theta_{j,1}}\sum_{n=1}^N \log\left(\theta_{j,1}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} + (1-\theta_{j,1})\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)}\right) \\
&= \sum_{n=1}^N \frac{\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} - \mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)}}{\theta_{j,1}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} + (1-\theta_{j,1})\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)}} \\
&= \sum_{n=1}^N \left(\frac{1}{\theta_{j,1}}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} - \frac{1}{1-\theta_{j,1}}\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)}\right).
\end{aligned} \tag{157}$$

By setting the derivative $\partial l(\theta)/\partial\theta_{j,1}$ equal to zero, we can deduce that

$$\begin{aligned}
&\frac{1}{\theta_{j,1}}\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} = \frac{1}{1-\theta_{j,1}}\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)} \\
\implies &(1-\theta_{j,1})\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} = \theta_{j,1}\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)} \\
\implies &\theta_{j,1}\left(\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)} + \sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)}\right) = \sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)} \\
\implies &\theta_{j,1} = \frac{\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)}}{\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=0)\wedge(y^{(n)}=1)} + \sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)}} = \frac{\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=1)}}{\sum_{n=1}^{N}\mathbb{1}_{y^{(n)}=1}}.
\end{aligned} \tag{158}$$

We can similarly show that

$$\theta_{j,0} = \frac{\sum_{n=1}^{N}\mathbb{1}_{(x_j^{(n)}=1)\wedge(y^{(n)}=0)}}{\sum_{n=1}^{N}\mathbb{1}_{y^{(n)}=0}}, \tag{159}$$

but the deduction would be analogous to above and thus details will be ignored here. Moreover, we may also assume that $\mathbb{P}[y]$ follows a Bernoulli distribution, such that

$$\mathbb{P}[y=1]=\theta_0, \qquad \mathbb{P}[y=0]=\theta_1. \tag{160}$$

Therefore, we can also show that

$$\theta_0 = \frac{\sum_{n=1}^{N}\mathbb{1}_{y^{(n)}=1}}{\sum_{n=1}^{N}\mathbb{1}_{y^{(n)}=0}+\sum_{n=1}^{N}\mathbb{1}_{y^{(n)}=1}} = \frac{\sum_{n=1}^{N}\mathbb{1}_{y^{(n)}=1}}{N}, \tag{161}$$

and the deduction is again analogous to above, by simply writing

$$\mathbb{P}[y^{(n)};\theta] = \theta_0\mathbb{1}_{y^{(n)}=1} + (1-\theta_0)\mathbb{1}_{y^{(n)}=0}. \tag{162}$$

Concluding the naive Bayes models as we have discussed above, we have made an assumption that the features are conditionally independent given the label. Our recipe for learning a naive Bayes model is then as follows.

- Choose $\mathbb{P}[x_i|y]$, for instance, a Bernoulli distribution for each binary x_i. If the chosen distribution involves k parameters, and furthermore there are l possible values of y, then we need to estimate kl parameters for this.
- Choose $\mathbb{P}[y]$, often a categorical distribution (but in the discussion above, just a Bernoulli distribution). If the chosen distribution involves m parameters, then we need to estimate these m parameters for this.
- Estimate the parameters via maximum likelihood estimation, same as the strategy for conditional models.

Naive Bayes with continuous inputs. Now we consider a multiclass classification with continuous inputs. Say

$$\mathbb{P}[x_i|y] \sim \mathcal{N}(\mu_{i,y},\sigma_{i,y}^2), \qquad \mathbb{P}[y=k]=\theta_k. \tag{163}$$

Denote $\texttt{params} = \{\{\mu_{i,y}\},\{\sigma_{i,y}\},\{\theta_k\}\}$. Then, we can compute the likelihood of the data as

$$\begin{aligned}
\mathbb{P}[\mathscr{D};\texttt{params}] &= \prod_{n=1}^{N}\mathbb{P}[x^{(n)},y^{(n)};\texttt{params}] = \prod_{n=1}^{N}\mathbb{P}[x^{(n)}|y^{(n)};\texttt{params}]\cdot\mathbb{P}[y^{(n)};\texttt{params}] \\
&= \prod_{n=1}^{N}\mathbb{P}[y^{(n)};\texttt{params}]\prod_{i=1}^{d}\mathbb{P}[x_i^{(n)}|y^{(n)};\texttt{params}] \\
&= \prod_{n=1}^{N}\left(\sum_k \theta_k\mathbb{1}_{y^{(n)}=k}\right)\prod_{i=1}^{d}\frac{1}{\sigma_{i,y^{(n)}}\sqrt{2\pi}}\exp\left(-\frac{(x_i^{(n)}-\mu_{i,y^{(n)}})^2}{2\sigma_{i,y^{(n)}}^2}\right).
\end{aligned} \tag{164}$$

The log likelihood would then be

$$l(\texttt{params}) = \sum_{n=1}^{N} \left(\log \left(\sum_k \theta_k \mathbb{1}_{y^{(n)}=k} \right) + \sum_{i=1}^{d} \left(\log \frac{1}{\sigma_{i,y^{(n)}}\sqrt{2\pi}} - \frac{(x_i^{(n)} - \mu_{i,y^{(n)}})^2}{2\sigma_{i,y^{(n)}}^2} \right) \right). \tag{165}$$

If we compute its partial derivatives with respect to each parameter, we have that

$$\begin{aligned}
\frac{\partial l}{\partial \mu_{j,k}}(\texttt{params}) &= \frac{\partial}{\partial \mu_{j,k}} \sum_{n:y^{(n)}=k} \left(-\sum_{i=1}^{d} \frac{(x_i^{(n)} - \mu_{i,y^{(n)}})^2}{2\sigma_{i,y^{(n)}}^2} \right) \\
&= \sum_{n:y^{(n)}=k} \frac{\partial}{\partial \mu_{j,k}} \left(-\frac{(x_j^{(n)} - \mu_{j,y^{(n)}})^2}{2\sigma_{j,y^{(n)}}^2} \right) = \sum_{n:y^{(n)}=k} \frac{x_j^{(n)} - \mu_{j,k}}{\sigma_{j,k}^2}, \qquad (166) \\
\frac{\partial l}{\partial \sigma_{j,k}}(\texttt{params}) &= \frac{\partial}{\partial \sigma_{j,k}} \sum_{n:y^{(n)}=k} \sum_{i=1}^{d} \left(\log \frac{1}{\sigma_{i,y^{(n)}}\sqrt{2\pi}} - \frac{(x_i^{(n)} - \mu_{i,y^{(n)}})^2}{2\sigma_{i,y^{(n)}}^2} \right) \\
&= \sum_{n:y^{(n)}=k} \frac{\partial}{\partial \sigma_{j,k}} \left(\log \frac{1}{\sigma_{j,y^{(n)}}\sqrt{2\pi}} - \frac{(x_j^{(n)} - \mu_{j,y^{(n)}})^2}{2\sigma_{j,y^{(n)}}^2} \right) \\
&= \sum_{n:y^{(n)}=k} \left(-\frac{1}{\sigma_{j,k}} + \frac{(x_j^{(n)} - \mu_{j,y^{(n)}})^2}{\sigma_{j,y^{(n)}}^3} \right) = \sum_{n:y^{(n)}=k} \frac{(x_j^{(n)} - \mu_{j,k})^2 - \sigma_{j,k}^2}{\sigma_{j,k}^3}, \qquad (167) \\
\frac{\partial l}{\partial \theta_m}(\texttt{params}) &= \frac{\partial}{\partial \theta_m} \sum_{n=1}^{N} \log \left(\sum_k \theta_k \mathbb{1}_{y^{(n)}=k} \right) = \frac{\partial}{\partial \theta_m} \log \left(\theta_m \mathbb{1}_{y^{(n)}=m} + (1-\theta_m)\mathbb{1}_{y^{(n)}\neq m} \right) \\
&= \sum_{n=1}^{N} \left(\frac{1}{\theta_m} \mathbb{1}_{y^{(n)}=m} - \frac{1}{1-\theta_m} \mathbb{1}_{y^{(n)}\neq m} \right). \qquad (168)
\end{aligned}$$

Setting each partial derivative to zero, we can obtain the parameters such that

$$\sum_{n:y^{(n)}=k} x_j^{(n)} = |\{n;\, y^{(n)} = k\}| \cdot \mu_{j,k} \implies \mu_{j,k} = \frac{\sum_{n:y^{(n)}=k} x_j^{(n)}}{|\{n;\, y^{(n)} = k\}|}, \tag{169}$$

$$\sum_{n:y^{(n)}=k} (x_j^{(n)} - \mu_{j,k})^2 = |\{n;\, y^{(n)} = k\}| \cdot \sigma_{j,k}^2 \implies \sigma_{j,k}^2 = \frac{\sum_{n:y^{(n)}=k} (x_j^{(n)} - \mu_{j,k})^2}{|\{n;\, y^{(n)} = k\}|}, \tag{170}$$

$$(1-\theta) \sum_{n=1}^{N} \mathbb{1}_{y^{(n)}=m} = \theta \sum_{n=1}^{N} \mathbb{1}_{y^{(n)}\neq m} \implies \theta_m = \frac{\sum_{n=1}^{N} \mathbb{1}_{y^{(n)}=m}}{\sum_{n=1}^{N} \mathbb{1}_{y^{(n)}=m} + \sum_{n=1}^{N} \mathbb{1}_{y^{(n)}\neq m}} = \frac{|\{n;\, y^{(n)} = m\}|}{N}. \tag{171}$$

3/21 Lecture

8 Bayesian Methods

A **parametric family of densities** is a set $\{\mathbb{P}[y;\theta];\ \theta \in \Theta\}$, where $\mathbb{P}[y;\theta]$ is a density on a sample space $\mathscr{Y}$, and θ is a parameter in a finite dimensional parameter space Θ. This is the common starting point for a treatment of classical or Bayesian statistics. In this lecture, the terms "density" and "integral" are equivalent to saying "mass function" and "sum".

8.1 Classical Statistics

This is also known as **frequentist statistics**. We assume that $\mathbb{P}[y;\theta]$ governs the world we are observing for some $\theta \in \Theta$. If we knew the right θ, then there would be no need for statistics. But instead of θ, we have the data $\mathscr{D} = \{y_1, \cdots, y_n\}$ sampled independently from $\mathbb{P}[y;\theta]$. Statistics is about how to get by with $\mathscr{D}$ in place of θ. One type of statistical problem is **point estimation**. A statistic $s = s(\mathscr{D})$ is any function on the data, and a statistic

$\hat{\theta} = \hat{\theta}(\mathscr{D})$ taking values in Θ is a **point estimator** of θ. A good point estimator needs to be consistent, *i.e.*, $\hat{\theta}_n \to \theta$ as data size $n \to \infty$. It also needs to be efficient, which roughly speaking, means that $\hat{\theta}_n$ is as accurate as we can get from a sample of size n. Maximum likelihood estimators are consistent and efficient under reasonable conditions. As an example, consider the coin flipping problem, where the parametric family of mass functions is

$$\mathbb{P}[\mathtt{H}; \theta] = \theta, \tag{172}$$

for $\theta \in \Theta = (0, 1)$. Assume that we have independently sampled data $\mathscr{D}$, where there are n_h heads and n_t tails. Thus, the likelihood function for data $\mathscr{D}$ would just be

$$L_{\mathscr{D}}(\theta) = \mathbb{P}[\mathscr{D}; \theta] = \prod_{i=1}^{n_h+n_t} \theta^{[\mathtt{H}]}(1-\theta)^{[\mathtt{T}]} = \theta^{n_h}(1-\theta)^{n_t}. \tag{173}$$

As usual, it is easier to maximize the log-likelihood function, such that

$$\hat{\theta}_{\mathrm{MLE}} = \arg\max_{\theta\in\Theta} \log L_{\mathscr{D}}(\theta) = \arg\max_{\theta\in\Theta}(n_h \log\theta + n_t \log(1-\theta)). \tag{174}$$

The first order condition, *i.e.*, equating the derivative to zero, can be computed as

$$\frac{n_h}{\theta} - \frac{n_t}{1-\theta} = 0 \implies \theta = \frac{n_h}{n_h + n_t}, \tag{175}$$

which implies that the point estimator $\hat{\theta}_{\mathrm{MLE}}$ found via maximum likelihood estimation is the empirical fraction of heads in the sampled dataset.

8.2 Bayesian Statistics

Compared with classical statistics, Bayesian statistics introduces a critical new ingredient, that is, the **prior distribution**. A prior distribution $\mathbb{P}[\theta]$ is a distribution on the parameter space Θ. It reflects our belief about θ, prior to seeing any data. A parametric Bayesian model consists of two pieces: a parametric family of densities $\{\mathbb{P}[\mathscr{D}|\theta];\ \theta \in \Theta\}$ and a prior distribution $\mathbb{P}[\theta]$ on a parametric space Θ. Putting the pieces together, we get a joint density on θ and $\mathscr{D}$, such that

$$\mathbb{P}[\mathscr{D}; \theta] = \mathbb{P}[\mathscr{D}|\theta]\mathbb{P}[\theta]. \tag{176}$$

The **posterior distribution** for θ is $\mathbb{P}[\theta|\mathscr{D}]$, which represents the rationally updated belief about θ after having seen $\mathscr{D}$. Recall that by Bayes' rule (in probability and statistics), we have that

$$\mathbb{P}[\theta|\mathscr{D}] = \frac{\mathbb{P}[\mathscr{D}|\theta]\mathbb{P}[\theta]}{\mathbb{P}[\mathscr{D}]} \implies \underbrace{\mathbb{P}[\theta|\mathscr{D}]}_{\text{posterior}} \propto \underbrace{\mathbb{P}[\mathscr{D}|\theta]}_{\text{likelihood}} \underbrace{\mathbb{P}[\theta]}_{\text{prior}}. \tag{177}$$

Again, take the coin flopping problem as an example, suppose that we have a parametric family of mass functions

$$\mathbb{P}[\mathtt{H}|\theta] = \theta, \tag{178}$$

for $\theta \in \Theta = (0, 1)$. We need a prior distribution $\mathbb{P}[\theta]$ on $\Theta = (0, 1)$, and a convenient choice would be a distribution from the beta family of distributions, as is shown in Figure 6.

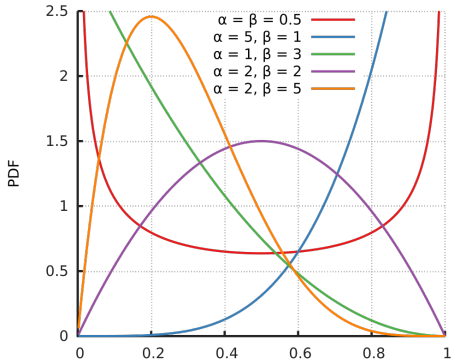

Figure 6: The probability density functions of the beta family of distributions.

Suppose we choose $\theta \sim \text{Beta}(\alpha, \beta)$ with $\alpha, \beta > 1$, then

$$\mathbb{P}[\theta] \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}. \tag{179}$$

Note that the mean and the mode of a beta distribution would be respectively

$$\mathbb{E}[\theta] = \frac{\alpha}{\alpha+\beta}, \qquad \arg\max_\theta \mathbb{P}[\theta] = \frac{\alpha-1}{\alpha+\beta-2}. \tag{180}$$

Back to our example, the likelihood function would be

$$L_{\mathcal{D}}(\theta) = \mathbb{P}[\mathscr{D}|\theta] = \theta^{n_h}(1-\theta)^{n_t}. \tag{181}$$

The posterior density can thus be considered as

$$\mathbb{P}[\theta|\mathscr{D}] \propto \mathbb{P}[\mathscr{D}|\theta]\mathbb{P}[\theta] \propto \theta^{n_h}(1-\theta)^{n_t} \cdot \theta^{\alpha-1}(1-\theta)^{\beta-1} = \theta^{n_h+\alpha-1}(1-\theta)^{n_t+\beta-1}. \tag{182}$$

That is, for a fixed dataset $\mathscr{D}$, the posterior $\theta|\mathscr{D} \sim \text{Beta}(n_h+\alpha, n_t+\beta)$, again following a beta family distribution. The interpretation for this is that, the prior initializes our counts with α heads and β tails, while the posterior increments the counts by the observed n_h and n_t respectively.

Definition 8.1. A family of distributions π is **conjugate to** the parametric model P if for any prior distribution in π, the posterior distribution is always in π.

As we have seen above, the beta family of distributions is conjugate to the coin flipping (*i.e.*, Bernoulli) model.

Now we are ready to establish the **Bayesian decision theory**. We need a parameter space Θ, a prior distribution $\mathbb{P}[\theta]$ on Θ, an action space $\mathscr{A}$, and a loss function $\ell : \mathscr{A} \times \Theta \to \mathbb{R}$. The posterior risk of an action $a \in \mathscr{A}$ is

$$r(a) = \mathbb{E}[\ell(\theta, a)|\mathscr{D}] = \int \ell(\theta, a)\mathbb{P}[\theta|\mathscr{D}]d\theta, \tag{183}$$

which is the expected loss under the posterior distribution. A **Bayes action** a^* is an action that minimizes the posterior risk, such that

$$r(a^*) = \min_{a \in A} r(a). \tag{184}$$

Now assume we have data $\mathscr{D}$ generated independently by $\mathbb{P}[y|\theta]$, but $\theta \in \Theta$ is unknown. We again want to produce a point estimate for θ. To do this, we find the action $\hat{\theta} \in \Theta$ that minimizes the posterior risk

$$r(\hat{\theta}) = \mathbb{E}[\ell(\hat{\theta}, \theta)|\mathscr{D}] = \int \ell(\hat{\theta}, \theta)\mathbb{P}[\theta|\mathscr{D}]d\theta. \tag{185}$$

There are a few important loss functions.

- Squared loss: $\ell(\hat{\theta}, \theta) = (\theta - \hat{\theta})^2$, giving the posterior mean.
- Zero-one loss: $\ell(\hat{\theta}, \theta) = \mathbb{1}_{\theta \neq \hat{\theta}}$, giving the posterior mode.
- Absolute loss: $\ell(\hat{\theta}, \theta) = |\theta - \hat{\theta}|$, giving the posterior median.

Take the squared loss as an example. We find the action $\hat{\theta} \in \Theta$ that minimize the squared posterior risk

$$r(\hat{\theta}) = \int (\theta - \hat{\theta})^2 \mathbb{P}[\theta|\mathscr{D}]d\theta. \tag{186}$$

Differentiating with respect to $\hat{\theta}$, we have that

$$\frac{dr(\hat{\theta})}{d\hat{\theta}} = -\int 2(\theta - \hat{\theta})\mathbb{P}[\theta|\mathscr{D}]d\theta = -2\underbrace{\int \theta\mathbb{P}[\theta|\mathscr{D}]d\theta}_{=\mathbb{E}[\theta|\mathcal{D}]} + 2\hat{\theta}\underbrace{\int \mathbb{P}[\theta|\mathscr{D}]d\theta}_{=1} = -2\mathbb{E}[\theta|\mathscr{D}] + 2\hat{\theta}. \tag{187}$$

The first order condition thus gives us $\hat{\theta} = \mathbb{E}[\theta|\mathscr{D}]$, proving that the Bayes action for squared posterior loss is indeed the posterior mean.

8.3 Bayesian Conditional Probability Models

Recall the classical conditional probability modeling. Suppose we have data $\mathscr{D} = \{y_1, \cdots, y_n\}$, then the probability density would be

$$\mathbb{P}[\mathscr{D}|x_1, \cdots, x_n; \theta] = \prod_{i=1}^{n} \mathbb{P}[y_i|x_i; \theta]. \tag{188}$$

For a fixed $\mathscr{D}$, the likelihood function (which is a function of θ) is just the probability density above. The maximum likelihood estimator for θ in the family $\{\mathbb{P}[y|x;\theta];\ \theta \in \Theta\}$ is defined as

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta\in\Theta} L_{\mathsf{D}}(\theta). \tag{189}$$

MLE corresponds to ERM if we set the loss to be the negative log-likelihood. By the way, the corresponding MLE prediction function is

$$\hat{f}(x) = \mathbb{P}[y|x; \hat{\theta}_{\text{MLE}}]. \tag{190}$$

Bayesian conditional models. Recall that the Bayesian condition model has two basic components: a parametric family of conditional densities $\{\mathbb{P}[y|x;\theta];\ \theta \in \Theta\}$ and a prior distribution $\mathbb{P}[\theta]$ on $\theta \in \Theta$. Suppose we have not yet observed any data. In the Bayesian setting, we can still produce a prediction function. The prior predictive distribution is given by

$$x \mapsto \mathbb{P}[y|x] = \int \mathbb{P}[y|x;\theta]\mathbb{P}[\theta]d\theta, \tag{191}$$

that is, an average of all conditional densities in our family, weighted by the prior. Now suppose we have already seen data $\mathscr{D}$. The posterior predictive distribution is given by

$$x \mapsto \mathbb{P}[y|x;\mathscr{D}] = \int \mathbb{P}[y|x;\theta]\mathbb{P}[\theta|\mathscr{D}]d\theta, \tag{192}$$

that is, an average of all conditional densities in our family, weighted by the posterior. Compared to the frequentist approach where we directly choose the optimal θ for prediction, in the Bayesian approach we use the weighted average by the posterior. Now, once we have a predictive function $\mathbb{P}[y|x;\mathscr{D}]$, we can easily generate single point predictions.

- $x \mapsto \mathbb{E}[y|x;\mathscr{D}]$, which is for minimizing the expected squared error.
- $x \mapsto \arg\max_{y\in\mathscr{Y}} \mathbb{P}[y|x;\mathscr{D}]$, which is for minimizing the expected zero-one loss.
- $x \mapsto \texttt{median}[y|x;\mathscr{D}]$, which is for minimizing the expected absolute error.

Gaussian regression example in 1-dimensional space. Let the input space be $\mathscr{X} = [-1, 1]$ and output space be $\mathscr{Y} = \mathbb{R}$. Suppose that for a given x, the real world generates y such that

$$y = w_0 + w_1 x + \epsilon, \tag{193}$$

where $\epsilon \sim \mathscr{N}(0, 0, 2^2)$. In other words, the conditional probability model setup is

$$y|x; w_0, w_1 \sim \mathscr{N}(w_0 + w_1 x, 0.2^2). \tag{194}$$

We will take the parameter space as $\mathbb{R}^2$ (*i.e.*, no restrictions on w_0 and w_1 except that they are real). The prior distribution will be taken as

$$w = (w_0, w_1) \sim \mathscr{N}\left(0, \frac{1}{2}I\right). \tag{195}$$

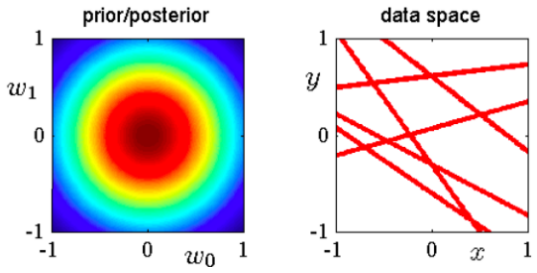

Figure 7: The image on the left is the prior distribution. Since this is the first iteration, we have nothing other than the prior distribution to use. The image on the right shows $x \mapsto \mathbb{E}[y|x; w] = w_0 + w_1 x$ for randomly chosen w following the prior distribution.

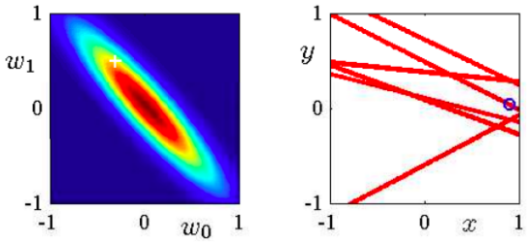

Figure 8: The image on the left is the posterior distribution after observing one datapoint. The white cross indicates the true parameters. The image on the right shows $x \mapsto \mathbb{E}[y|x; w] = w_0 + w_1 x$ for randomly chosen w following the posterior distribution $\mathbb{P}[w|\mathscr{D}]$.

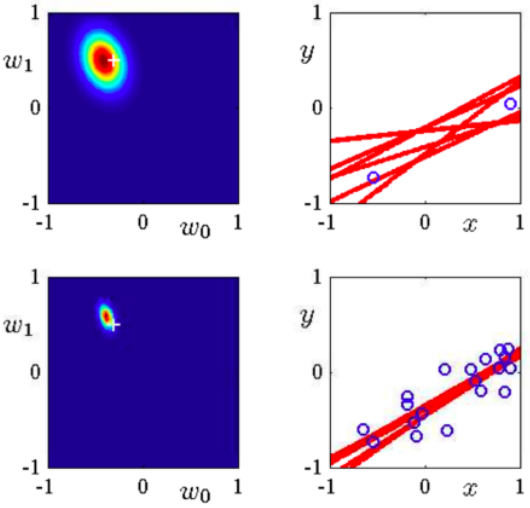

Figure 9: The images on the left are the posterior distributions after observing two datapoints and twenty datapoints. The white cross indicates the true parameters. The images on the right show $x \mapsto \mathbb{E}[y|x; w] = w_0 + w_1 x$ for randomly chosen w following the posterior distributions $\mathbb{P}[w|\mathscr{D}]$ after observing two datapoints and twenty datapoints respectively. In brief, these are just iterating the process in Figure 8.

Closed form for posterior in Gaussian regression. Suppose we have the model

$$w \sim \mathcal{N}(0, \Sigma_0), \qquad y_i|x; w \sim \mathcal{N}(w^\top x_i, \sigma^2). \tag{196}$$

Moreover, we denote by X the design matrix and by y the response column vector as usual. The posterior distribution is a Gaussian distribution

$$w|\mathscr{D} \sim \mathcal{N}(\mu_P, \Sigma_P), \tag{197}$$

where

$$\mu_P = (X^\top X + \sigma^2 \Sigma_0^{-1})^{-1} X^\top y, \qquad \Sigma_P = (\sigma^{-2} X^\top X + \Sigma_0^{-1})^{-1}. \tag{198}$$

The posterior variance σ_P gives us a natural uncertainty measure. Now if we want point estimates of w, the posterior mean is given by

$$\hat{w} = \mu_P = (X^\top X + \sigma^2 \Sigma_0^{-1})^{-1} X^\top y. \tag{199}$$

The **maximum a posteriori (MAP) estimate** (*i.e.*, the posterior mode) is the also given as above in this case. If we have prior variance $\Sigma_0 = \frac{\sigma^2}{\lambda} I$, we then obtain that

$$\hat{w} = (X^\top X + \lambda I)^{-1} X^\top y, \tag{200}$$

which is exactly the closed-form solution of the ridge regression.

3/28 Lecture

9 Multiclass Classification

So far, most algorithm we have learned are designed for binary classification. However, many real-world problems have a lot more than 2 classes, so the problem is, how can we reduce multiclass classification to binary classification, or how we can generalize binary classification to multiclass classification.

9.1 Reduction to Binary Classification

One-vs-all / One-vs-rest. Let $\mathscr{X}$ be the input space and $\mathscr{Y} = \{1, \cdots, k\}$ be the output space. We train k binary classifiers, one for each class, such that $h_1, \cdots, h_k : \mathscr{X} \to \mathbb{R}$. Each classifier h_i would be used to distinguish class i from the rest. In order to make a prediction, we take the majority vote, such that

$$h(x) = \arg\max_{i=1,\cdots,k} h_i(x), \tag{201}$$

where ties can be broken arbitrarily.

All-vs-all / One-vs-one / All-pairs. Again, let $\mathscr{X}$ be the input space and $\mathscr{Y} = \{1, \cdots, k\}$ be the output space. We train $\binom{k}{2}$ binary classifiers, one for each pair, such that $h_{ij} : \mathscr{X} \to \mathbb{R}$ for $1 \le i < j \le k$. Each classifier h_{ij} would be used to distinguish class i from class j. In order to make a prediction, we again take the majority vote, such that

$$h(x) = \arg\max_{i=1,\cdots,k} \sum_{j \neq i} (\mathbb{1}_{i<j} h_{ij}(x) - \mathbb{1}_{j<i} h_{ji}(x)), \tag{202}$$

where ties can also be broken arbitrarily. This is like a tournament.

Error correcting output codes. If we encode labels as binary codes and predict the code bits directly, then OvA encoding would become something like $10\cdots0$, $01\cdots0$, etc., until $00\cdots1$. It uses k bits to encode each label. We can also use fewer. For instance, we can use 6 bits to encode 8 classes. Then we need 6 binary classifiers, each distinguishing whether the ith bit of the code is 1. The prediction is then made via finding the closest label in terms of Hamming distance. The hamming distance between to binary strings is computed via counting the number of 1's in their XOR result. This is computationally more efficient that OvA (which is in fact a special case of ECOC), and can be better for large k. Then why not use the minimal number of bits, *i.e.*, $\log_2 k$ bits? Clearly there is a tradeoff between code distance and binary classification performance. If we use fewer bits, then the minimal code distance will be small, leading to less robustness. If we use more bits, ECOC would be robust to errors, but it can significantly increase runtime.

9.2 Multiclass Loss

In binary logistic regression, given an input x, we would like to output a classification between $\{0, 1\}$. We can use

$$f(x) = \text{sigmoid}(z) = \frac{1}{1+\exp(-z)} = \frac{1}{1+\exp(-w^\top x - b)}, \tag{203}$$

$$1 - f(x) = \frac{\exp(-w^\top x - b)}{1+\exp(-w^\top x - b)} = \frac{1}{\exp(w^\top x + b)} = \frac{1}{1+\exp(z)} = \text{sigmoid}(-z). \tag{204}$$

Then, the loss function would be given by $L = -\sum_i \left(y^{(i)} \log f(x^{(i)}) + (1-y^{(i)}) \log(1 - f(x^{(i)}))\right)$. Now in multiclass logistic regression, if we have one w_c for each class c, we can use

$$f_c(x) = \frac{\exp(w_c^\top x + b_c)}{\sum_c \exp(w_c^\top x + b_c)}. \tag{205}$$

This is also called **softmax** in neural networks. The loss function would be $L = -\sum_i (y_c^{(i)} \log f_c(x^{(i)}))$ for each class.

Multiclass perceptron. The base linear estimators are $h_i(x) = w_i^\top x$, where $w_i \in \mathbb{R}^d$. Given a multiclass dataset $\mathscr{D} = \{(x, y)\}$, the multiclass perceptron algorithm can be described as in Algorithm 7.

Algorithm 7 Multiclass perceptron (hard to scale)

Initialize $w_i \leftarrow 0$ for each w_i;
repeat
  for $(x, y) \in \mathscr{D}$ **do**
    $\hat{y} \leftarrow \arg\max_{y' \in Y} w_{y'}^\top x$;
    if $\hat{y} \neq y$ (implying a wrong prediction) **then**
      $w_y \leftarrow w_y + x$; `// Move the target class scorer towards x`
      $w_{\hat{y}} \leftarrow w_{\hat{y}} - x$; `// Move the wrong class scorer away from x`
    end if
  end for
until the stopping criterion is satisfied (hopefully there does not exist misclassified examples, or the maximum number of iterations is reached);

However, remember that we want to scale to very large number of classes and reuse algorithms and analysis for binary classification. Therefore, a single weight vector is desired instead of a separate weight vector for each class. The way to do this is to write

$$w_i^\top x = w^\top \psi(x, i), \qquad h_i(x) = h(x, i) \text{ (the score function)}. \tag{206}$$

By doing this, we encode labels into the feature space, and the score of each label would become the score for the "compatibility" of a label and an input. Then, how do we construct the feature map ψ? An easy way to do this is to stack the w_i's together. For instance, suppose we have

$$w = \left(\underbrace{-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}}_{w_1}, \underbrace{0, 1}_{w_2}, \underbrace{\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}}_{w_3} \right), \tag{207}$$

then we can take $\psi : \mathbb{R}^2 \times \{1, 2, 3\} \to \mathbb{R}^6$ such that

$$\psi(x, 1) = (x_1, x_2, 0, 0, 0, 0), \qquad \psi(x, 2) = (0, 0, x_1, x_2, 0, 0), \qquad \psi(x, 3) = (0, 0, 0, 0, x_1, x_2). \tag{208}$$

As a result, we can see that $w_i^\top x = w^\top \psi(x, i)$. Therefore, we would like to modify the multiclass perceptron algorithm using the multivector construction, as described in Algorithm 8. The setting is the same as above.

Algorithm 8 Multiclass perceptron (modified)

Initialize $w \leftarrow 0$;
repeat
  for $(x, y) \in \mathscr{D}$ **do**
    $\hat{y} \leftarrow \arg\max_{y' \in \mathcal{Y}} w^\top \psi(x, y')$;
    if $\hat{y} \neq y$ (implying a wrong prediction) **then**
      $w \leftarrow w + \psi(x, y)$; `// Move the target class scorer towards x`
      $w \leftarrow w - \psi(x, \hat{y})$; `// Move the wrong class scorer away from x`
    end if
  end for
until the stopping criterion is satisfied (hopefully there does not exist misclassified examples, or the maximum number of iterations is reached);

9.3 Linear Multiclass SVM

Recall that the binary margin for $(x^{(n)}, y^{(n)})$ is $y^{(n)} w^\top x^{(n)}$. We want the margin to be large and positive, so that $w^\top x^{(n)}$ has the same sign as $y^{(n)}$ and we are confident of the prediction. Now in the multiclass setting, the class-specific margin for $(x^{(n)}, y^{(n)})$ would be

$$h(x^{(n)}, y^{(n)}) - h(x^{(n)}, y) = w^\top \psi(x^{(n)}, y^{(n)}) - w^\top \psi(x^{(n)}, y), \tag{209}$$

the difference between scores of the correct class and each other class. We thus want the margin to be large and positive for all $y \neq y^{(n)}$. Now we consider the separable case. The binary classification problem would be

$$\min_w \frac{1}{2} \|w\|^2, \qquad \text{s.t. } \underbrace{y^{(n)} w^\top x^{(n)}}_{\text{margin}} \geq 1, \qquad \forall (x^{(n)}, y^{(n)}) \in \mathscr{D}. \tag{210}$$

Analogously, the multiclass classification problem would be

$$\min_w \frac{1}{2} \|w\|^2, \qquad \text{s.t. } \underbrace{w^\top \psi(x^{(n)}, y^{(n)}) - w^\top \psi(x^{(n)}, y)}_{\text{margin}} \geq 1, \qquad \forall (x^{(n)}, y^{(n)}) \in \mathscr{D}. \tag{211}$$

Note that in the nonseparable case, we use the penalty instead of hard constraint. Now we recall the hinge loss for binary classification, which is a convex upper bound on the 0-1 loss, such that

$$\ell_{\text{hinge}}(y, x, w) = \max(0, 1 - y w^\top x). \tag{212}$$

Similarly, we generalize the hinge loss for the multiclass case such that

$$\ell_{\text{hinge}}(y, x, w) = \max_{y' \in \mathcal{Y}} \left(\Delta(y, y') - \left(w^\top \psi(x, y) - w^\top \psi(x, y') \right) \right), \tag{213}$$

where $\Delta(y, y')$ is 1 if $y = y'$ and 0 otherwise. Similar to the hinge loss for binary classification, we can prove show that the hinge loss for multiclass classification is an upper bound on $\Delta(y, y')$. Note that

$$\begin{aligned} \hat{y} := \arg\max_{y' \in \mathcal{Y}} w^\top \psi(x, y') &\implies w^\top \psi(x, y) \leq w^\top \psi(x, \hat{y}) \\ &\implies \Delta(y, \hat{y}) \leq \Delta(y, \hat{y}) - w^\top \psi(x, y) + w^\top \psi(x, \hat{y}). \end{aligned} \tag{214}$$

Finally, recall the hinge loss formulation for binary SVM (without the bias term), which can be written as

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \max \left(0, 1 - y^{(n)} w^\top x^{(n)} \right). \tag{215}$$

Analogously, the multiclass hinge loss objective can be formulated as

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \max_{y' \in \mathcal{Y}} \left(\Delta(y^{(n)}, y') - \left(w^\top \psi(x^{(n)}, y^{(n)}) - w^\top \psi(x^{(n)}, y') \right) \right). \tag{216}$$

Here $\Delta(y, y')$ is the target margin for each class, and if the margin $w^\top \psi(x^{(n)}, y^{(n)}) - w^\top \psi(x^{(n)}, y')$ exceeds its target $\Delta(y, y')$, there would be no loss on example n.

Multiclass vs OvA. In OvA, we train k models $h_1, \cdots, h_k : \mathscr{X} \to \mathbb{R}$ and predict with $\arg\max_{y \in \mathsf{Y}} h_y(x)$. With multiclass loss, we train only one model $h : \mathscr{X} \times \mathscr{Y} \to \mathbb{R}$ and predict via solving $\arg\max_{y \in \mathsf{Y}} h(x, y)$. In practice, both performed roughly the same. However, since we want to generalize to situations where k is very large, the OvA approach would be intractable.

9.4 Introduction to Structured Prediction

Let us start with an example. Given a sentence, we want to give a part of speech tag for each word. In this case,

$$\begin{aligned}
\mathscr{V} &= \{\text{all English words}\} \cup \{\texttt{[START]}, \text{“.”}\}, \\
\mathscr{X} &= \mathscr{V}^n, \qquad n = 1, 2, 3, \cdots, && \text{(word sequences of any length)} \\
\mathscr{P} &= \{\texttt{[START]}, \text{pronoun}, \text{verb}, \text{noun}, \text{adjective}\}, \\
\mathscr{Y}(x) &= \mathscr{P}^n, \qquad n = 1, 2, 3, \cdots. && \text{(part of speech sequences of any length)}
\end{aligned}$$

We have the discrete output space $\mathscr{Y}(x)$. It is very large but has certain structures, such as linear chains (sequence labeling) and trees (parsing). Also, its size depends on the input x. The base hypothesis space would be $\mathscr{H} = \{h : \mathscr{X} \times \mathscr{Y}(x) \to \mathbb{R}\}$, where each $h(x, y)$ gives a compatibility score between input x and output y. Now, we use the multiclass hypothesis space, such that

$$\mathscr{F} = \left\{ x \mapsto \arg\max_{y \in \mathsf{Y}(x)} h(x, y);\ h \in \mathscr{H} \right\}. \tag{217}$$

The final prediction function would be some $f \in \mathscr{F}$. Also, remember that for each $f \in \mathscr{F}$, there is an underlying compatibility score function $h \in \mathscr{H}$. Now we need to design the feature map ψ so that

$$h(x, y) = w^\top \psi(x, y). \tag{218}$$

- **Unary features.** A unary feature depends solely on the label at a single position y_i and the input x. As an example, we can have

$$\begin{aligned}
\phi_1(x, y_i) &= \mathbb{1}_{x_i = \text{runs}} \cdot \mathbb{1}_{y_i = \text{verb}}, && (219) \\
\phi_2(x, y_i) &= \mathbb{1}_{x_i = \text{runs}} \cdot \mathbb{1}_{y_i = \text{noun}}, && (220) \\
\phi_3(x, y_i) &= \mathbb{1}_{x_{i-1} = \text{He}} \cdot \mathbb{1}_{x_i = \text{runs}} \cdot \mathbb{1}_{y_i = \text{verb}}, && (221) \\
&\cdots
\end{aligned}$$

- **Markov features.** A Markov feature depends on two adjacent labels y_{i-1} and y_i, and the input x. For instance, we can have

$$\begin{aligned}
\theta_1(x, y_{i-1}, y_i) &= \mathbb{1}_{y_{i-1} = \text{pronoun}} \cdot \mathbb{1}_{y_i = \text{verb}}, && (222) \\
\theta_2(x, y_{i-1}, y_i) &= \mathbb{1}_{y_{i-1} = \text{pronoun}} \cdot \mathbb{1}_{y_i = \text{noun}}, && (223) \\
&\cdots
\end{aligned}$$

  These are reminiscent of Markov models in the output space, and are possible to have higher-order features.

Now at each position i in the sequence, we define the **local feature vector** (unary and Markov), such that

$$\psi_i(x, y_{i-1}, y_i) = (\phi_1(x, y_i), \phi_2(x, y_i), \cdots, \theta_1(x, y_{i-1}, y_i), \theta_2(x, y_{i-1}, y_i), \cdots), \tag{224}$$

and thus a **local compatibility score** at position i would be $w^\top \psi_i(x, y_{i-1}, y_i)$. The compatibility score for (x, y) can then be obtained by summing up local compatibility scores, such that

$$\sum_i w^\top \psi_i(x, y_{i-1}, y_i) = w^\top \left(\sum_i \psi_i(x, y_{i-1}, y_i) \right) := w^\top \psi(x, y), \tag{225}$$

where we define the **sequence feature vector** by

$$\psi(x, y) = \sum_i \psi_i(x, y_{i-1}, y_i). \tag{226}$$

Now, we can use the structured perceptron algorithm, which would be identical to the multiclass perceptron algorithm except that we now take $\hat{y}$ as the argmax across the structured output space $\mathscr{Y}(x)$ instead of $\mathscr{Y}$. Again, similar to multiclass classification problem, we have the structured hinge loss and the structured SVM objective, respectively identical to the multiclass hinge loss and the multiclass SVM objective, except that we substitute $\mathscr{Y}$ with the structure output space $\mathscr{Y}(x)$.

The argmax problem for sequences. To make predictions, recall that we need to find $\arg\max_{y\in\mathcal{Y}(x)} w^\top\psi(x,y)$. However, $|\mathcal{Y}(x)|$ would be exponentially large, so it is hard to make direct computation. Instead, we note that $\psi(x,y)$ can be decomposed into the sum of local compatibility scores. Then, we can use dynamic programming (for instance, by Viterbi algorithm) to solve the problem.

9.5 Conditional Random Field

Recall that we can write logistic regression in a general form

$$\mathbb{P}[y|x] = \frac{1}{Z(x)}\exp(w^\top\psi(x,y)), \tag{227}$$

where $Z(x)$ is a normalization coefficient, such that

$$Z(x) = \sum_{y\in\mathcal{Y}}\exp(w^\top\psi(x,y)). \tag{228}$$

Now with a linear chain $\{y_t\}$, we can also incorporate unary and Markov features, such that

$$\mathbb{P}[y|x] = \frac{1}{Z(x)}\exp\left(\sum_t w^\top\psi(x,y_{t-1},y_t)\right). \tag{229}$$

Compared to SVM, CRF has a probabilistic interpretation, and we can learn via maximum log likelihood (with regularization term). The loss function would thus be

$$\begin{aligned} l(w) =& -\frac{1}{N}\sum_{i=1}^N \log\mathbb{P}[y^{(i)}|x^{(i)}] + \frac{\lambda}{2}\|w\|^2 = -\frac{1}{N}\sum_i\sum_t w^\top\psi(x,y_{t-1},y_t) + \frac{1}{N}\sum_i \log Z(x^{(i)}) + \frac{\lambda}{2}\|w\|^2 \\ =& -\frac{1}{N}\sum_i\sum_t\sum_k w_k\psi_k(x^{(i)},t_{t-1}^{(i)},y_t^{(i)}) + \frac{1}{N}\sum_i\log Z(x^{(i)}) + \frac{\lambda}{2}\sum_k w_k^2. \end{aligned} \tag{230}$$

The gradient can then be computed as

$$\frac{\partial l}{\partial w_k}(w) = -\frac{1}{N}\sum_i\sum_t\psi_k(x^{(i)},t_{t-1}^{(i)},y_t^{(i)}) + \frac{1}{N}\sum_i\frac{\partial}{\partial w_k}\log Z(x^{(i)}) + \lambda w_k, \tag{231}$$

where the middle term need to be expanded as

$$\begin{aligned} \frac{1}{N}\sum_i\frac{\partial}{\partial w_k}\log Z(x^{(i)}) &= \frac{1}{N}\sum_i\frac{\partial}{\partial w_k}\log\sum_{y'\in\mathcal{Y}(x^{(i)})}\exp\left(\sum_t w^\top\psi(x^{(i)},y'_{t-1},y'_t)\right) \\ &= \frac{1}{N}\sum_i\frac{\partial}{\partial w_k}\log\sum_{y'\in\mathcal{Y}(x^{(i)})}\exp\left(\sum_t\sum_{k'} w_{k'}\psi_{k'}(x^{(i)},y'_{t-1},y'_t)\right) \\ &= \frac{1}{N}\sum_i\left(\sum_{y'\in\mathcal{Y}(x^{(i)})}\exp\left(\sum_t\sum_{k'} w_{k'}\psi_{k'}(x^{(i)},y'_{t-1},y'_t)\right)\right)^{-1} \\ &\qquad\left(\sum_{y'\in\mathcal{Y}(x^{(i)})}\exp\left(\sum_t\sum_{k'} w_{k'}\psi_{k'}(x^{(i)},y'_{t-1},y'_t)\right)\sum_t\psi_k(x^{(i)},y'_{t-1},y'_t)\right) \\ &= \frac{1}{N}\sum_i\frac{1}{Z(x^{(i)})}\sum_{y'\in\mathcal{Y}(x^{(i)})}\exp\left(\sum_t w^\top\psi(x^{(i)},y'_{t-1},y'_t)\right)\sum_t\psi_k(x^{(i)},y'_{t-1},y'_t) \\ &= \frac{1}{N}\sum_i\sum_{y'\in\mathcal{Y}(x^{(i)})}\mathbb{P}[y'|x^{(i)}]\left(\sum_t\psi_k(x^{(i)},y'_{t-1},y'_t)\right). \end{aligned} \tag{232}$$

Moving the summation over t outside, we can see that this is just the expectation of $\psi_k(x^{(i)},y'_{t-1},y'_t)$ under the model distribution $\mathbb{P}[y'|x^{(i)}]$. Therefore, in order to compute the gradient, we need to infer the expectation under the

model distribution. In the linear chain structure, we can use the forward-backward algorithm for inference, similar to Viterbi algorithm. We initiate $\alpha_j(1) = \exp(w^\top \psi(x_1, y_1 = j))$, then apply the recursive relation

$$\alpha_j(t) = \sum_i \alpha_i(t-1) \exp\left(w^\top \psi(x_t, y_{t-1} = i, y_t = j)\right). \tag{233}$$

This is the forward direction, and the backward direction is similar. The inference algorithm can be generalized to belief propagation (BP) in a tree structure (exact inference), but in general graphs we rely on approximate inference. Comparing with SVM in which we need to compute the argmax, both are NP-hard for general graphs.

4/4 Lecture

10 Decision Trees

In this section, we will introduce our first inherently nonlinear model: decision trees.

10.1 Decision Trees

We focus on binary trees, as opposed to multiway trees where each node can have more than two children. In our binary tree, each node contains a subset of data points. The data splits created by each node involve only a single feature. For continuous variables, the splits are always in the form $x_i \le t$, while for discrete variables, we partition values into two sets (which we will not discuss at present). The predictions are made in the terminal (leaf) nodes.

Constructing the tree. Goal: We want to find boxes $R_1, \cdots, R_J$ that minimize $\sum_{j=1}^J \sum_{i \in R_J} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints. Problem: However, finding the optimal binary tree would be computationally intractable. Solution: Therefore, we want to apply a greedy algorithm, such that starting from the root and repeating until a stopping criterion is reached (*e.g.*, maximum depth), we find the non-terminal node that results in the **best split**. Prediction: Our prediction would then be the mean value of a terminal node $\mathbb{E}[y_i | x_i \in R_m]$, *i.e.*, the average of all training instances in R_m. However, note that a greedy algorithm is the one that makes the best local decisions, without looking ahead to evaluate their downstream consequences. Therefore, this procedure is not very likely to result in a globally optimal tree.

Finding the best split point. We enumerate all features and all possible split points for each feature. There are infinitely many split points, but suppose now we are considering splitting the jth feature x_j, and let $x_j^{(1)}, \cdots, x_j^{(n)}$ be the sorted values of the jth feature. We only need to consider split points between to adjacent values, and any split point in some interval $(x_j^{(r)}, x_j^{(r+1)})$ will result in the same loss. It is therefore common to split half way between two adjacent values, that is,

$$s_j \in \left\{ \frac{1}{2}\left(x_j^{(r)} + x_j^{(r+1)}\right);\ r = 1, \cdots, n-1 \right\}. \tag{234}$$

Note that in decision tree classification, our plan is to predict the majority label in each region. Therefore, we want to produce more pure nodes, *i.e.*, nodes where most instances have the same class. This is the standard of "good split" for classification problems.

Overfitting of decision trees. If we keep splitting the data into more and more regions, this will end up overfitting, *i.e.*, every data point will be in its own region. Therefore, we need to have certain stopping criterion to control the complexity of the hypothesis space. For instance, we can limit the total number of nodes, the number of terminal nodes, the tree depth, or require a minimum number of data points in a terminal node. Aside from these simple approaches, we can also use **backward pruning**, in which we first build a really big tree (*e.g.*, until all regions have no more than 5 data points), the prune the tree backward greedily (potentially all the way to the root), until the validation performance (*e.g.*, **cross validation**) starts decreasing.

Node impurity measures. Consider the multiclass classification case $\mathscr{Y} = \{1, 2, \cdots, K\}$. Let node m represent the region R_m, with N_m observations. We denote the proportion of observations in R_m with class k by

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{y_i = k}. \tag{235}$$

We predict the majority class in node m such that $k(m) = \arg\max_k \hat{p}_{m.k}$. There are three measures of **node impurity** for leaf node m, *i.e.*, the misclassification error, the Gini index, and the entropy (information gain).

- Misclassification error: $1 - \hat{p}_{m,k(m)}$.
- Gini index: $\sum_{k=1}^{K} \hat{p}_{m,k}(1 - \hat{p}_{m,k})$, which encourages $\hat{p}_{m,k}$ to be close to either 0 or 1.
- Entropy: $-\sum_{k=1}^{K} \hat{p}_{m,k} \log \hat{p}_{m,k}$, which measures the information gain.

The Gini index and the entropy are numerically similar to each other, and both work better in practice than the misclassfication error. Now, note that we also need to score a potential split that produces the left and right nodes R_L and R_R. Suppose that we have N_L data points in R_L and N_R data points in R_R. Let $Q(R_L)$ and $Q(R_R)$ be their node impurity measures respectively. We aim to find a split that minimizes the weighted average of node impurities

$$\frac{N_L Q(R_L) + N_R Q(R_R)}{N_L + N_R}. \tag{236}$$

Interpretability of decision trees. Trees are easy to visualize and explain than other classifiers (even linear regression). Small trees are especially interpretable though larger trees may not be so much. Compared with linear models, trees may have to work harder to capture the linear decision boundaries, but can easily capture certain nonlinear ones. As is shown in Figure 10, we can see that decision tree models may need to zigzag to approximate a simple linear boundary, but can easily separate those angular boundaries.

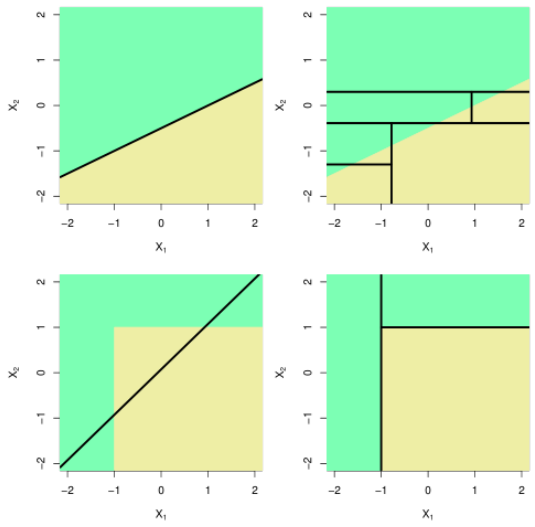

Figure 10: The comparison between trees and linear models. The plots on the left correspond to a linear model and the plots on the right correspond to a decision tree.

Overall, compared with linear models, decision trees are nonlinear so that they can capture certain nonliear boundaries easily. They are also non-metric, *i.e.*, they do not rely on the geometry of the space (inner products or distances). Moreover, they are non-parametric, so that they need no assumptions on the distribution of the data. However, decision trees may struggle to capture linear decision boundaries, and they have high variance and tend to overfit, since they are extremely sensitive to small changes in the training data. As we will discuss later, we need certain ensemble technique to mitigate the overfitting issues.

10.2 Bagging and Random Forests

We recall statistics and point estimators. Suppose we observe data $\mathscr{D} = \{x_1, \cdots, x_n\}$ sampled independently from a parametric distribution $\mathbb{P}[\cdot;\ \theta]$. A statistic $s = s(\mathscr{D})$ is any function of the data, *e.g.*, sample mean, sample variance, histogram, empirical data distribution, etc. A statistic $\hat{\theta} = \hat{\theta}(\mathscr{D})$ is a point estimator of θ if $\hat{\theta} \approx \theta$. Statistics are random, so they have probability distributions, which are called sampling distributions. The standard deviation of a sampling distribution is called the **standard error**. Some parameters of the sampling distribution we are interested in include the bias $\text{Bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta}] - \theta$ and the variance $\text{Var}[\hat{\theta}] = \mathbb{E}[\hat{\theta}^2] - \mathbb{E}^2[\hat{\theta}]$.

Now let $\hat{\theta}(\mathscr{D})$ be an unbiased estimator with variance σ^2, *i.e.*, $\mathbb{E}[\hat{\theta}] = \theta$ and $\text{Var}[\hat{\theta}] = \sigma^2$. Its standard error would be the square root of the variance, *i.e.*, σ. So far we have used only a single statistic to estimate θ, but now consider a new estimator that takes the average of independent and identically distributed statistics $\hat{\theta}_1, \cdots, \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(\mathscr{D}^{(i)})$. This new estimator would have the same expected value, but with a smaller standard error, such that

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \theta, \qquad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}[\hat{\theta}_i] = \frac{\sigma^2}{n}. \tag{237}$$

This similar concept can be applied if we have B independent training sets, all drawn from the same distribution $\mathbb{P}[\cdot;\ \theta]$. Our learning algorithm gives us B prediction functions $\hat{f}_1(x), \cdots, \hat{f}_B(x)$. We then define the average prediction function $\hat{f}_{\text{avg}}$ by taking the arithmetic mean of these prediction functions. The average prediction function would then have the same expectation as each of these prediction function, but with a smaller variance. However, the problem here is that, we do not have B independent training sets in practice. This leads to the idea of **bootstrapping** in order to simulate multiple samples when we have only one.

The bootstrap method. A **bootstrap sample** from the dataset $\mathscr{D}_n = \{x_1, \cdots, x_n\}$ is a sample of size n drawn with replacement from $\mathscr{D}_n$. That is, some elements of $\mathscr{D}_n$ can show up multiple times, while some elements may not show up at all. In this scenario, each x_i has a probability $(1 - 1/n)^n$ of not being included in a given bootstrap sample. For large n, this probability tends to $1/e \approx 0.368$. Therefore, we expect approximately 63.2% of elements of $\mathscr{D}_n$ to show up at least once in a given bootstrap sample. Now, the **bootstrap method** simulates B independent samples from $\mathbb{P}[\cdot;\ \theta]$ by taking B bootstrap samples from $\mathscr{D}_n$, say $D_n^{(1)}, \cdots, D_n^{(B)}$. This often ends up being close to what we would get with B independent samples from $\mathbb{P}[\cdot;\ \theta]$.

Now that we have applied the bootstrap method and we will the resulting bootstrap samples as independent training sets. Then, we can use certain **ensemble methods** to combine multiple weak models into a single and more powerful model. As we have previously shown, averaging over the bootstrap samples will reduce the standard error (or variance) without changing the bias (or expected value). The ensemble strategies include **parallel ensemble** in which models are built independently, and **sequential ensemble** in which models are built sequentially. An example of the former is bagging, and an example for the latter is boosting, *i.e.*, trying to find learners that do well where previous learners fall short.

Bagging. Suppose that we draw B bootstrap samples $D^{(1)}, \cdots, D^{(B)}$ from the original dataset $\mathscr{D}$. Let $\hat{f}_1, \cdots, \hat{f}_B$ be the prediction functions resulting from training on $D^{(1)}, \cdots, D^{(B)}$, respectively. The **bagged prediction function** is simply a combination of these prediction functions. For instance, we can take the average for regression problems and take the majority vote for classification problems. Bagging is a general method for variance reduction, but it is particularly useful for decision trees. The reason is, small perturbations between each bootstrap sample can lead to high degree of model variability since decision tree models are sensitive. Then bagging can help a lot since these base learners are relatively unbiased but with high variance. Now compared with simple decision tree models, bagging tends not to lead to overfitting. However, the downside is that, if we have many trees, then the bagged predictor would be much less interpretable.

Out-of-bag error estimation. Recall that each bagged predictor is trained on approximately 63.2% of the original dataset. The remaining 36.8% would then be called **out-of-bag (OOB)** observations. For the ith training point x_i in $\mathscr{D}$, let $S_i = \left\{b;\ D^{(b)} \text{ does not contain } x_i\right\}$. Then, the **OOB prediction** on x_i would then be

$$\hat{f}_{\text{OOB}}(x_i) = \frac{1}{|S_i|}\sum_{b \in S_i}\hat{f}_b(x_i). \tag{238}$$

The OOB error is a good estimate of the testing error, because none of the prediction functions that we take average over has ever been trained with x_i. This is similar to cross validation.

Random Forests. We recall the motivating principle of bagging, that is, if we have n independent and identically distributed estimators, we can use bagging to reduce the variance. However, note that though bootstrap samples can be treated as independent from the training set, they are not independent from $\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$. Then, if we take the average and compute the variance, the covariance terms would dominate, limiting the benefits of averaging. Therefore, we want to reduce the dependence between the estimators.

The key idea here is, we still use bagged decision trees, but we modify the tree-growing procedure. First of all, we build a collection of trees independently as before. Then, when constructing each tree node, we restrict the choice of splitting variable to a randomly chosen subset of feature of size m. This would prevent situations in which all trees are dominated by the same small number of strong features and therefore too similar to each other. Typically, we choose $m \approx \sqrt{p}$ where p is the total number of features, or we can choose m using cross validation. Note that when $m = p$, random forest is essentially just bagging.

10.3 Boosting

Bagging reduces variance of a low-bias and high-variance estimator by ensembling many estimators trained in parallel. On the other hand, **boosting** would reduce the error rate of a high-bias estimator by ensembling many estimators trained in sequence (without bootstrapping). Similar to bagging, boosting is a general method that is particularly popular with decision trees. The main intuition here is that, instead of fitting the data very closely using a large decision tree, we train gradually using a sequence of simpler trees. Here, we will focus on a specific implementation, **AdaBoost** (Freund & Schapire, 1997).

Consider binary classification $\mathscr{Y} = \{-1, 1\}$. Typical base hypothesis spaces include decision stumps (*i.e.*, trees with a single split), trees with few terminal nodes, and linear decision functions. Each base learner would be trained on weighted data, say training set $\mathscr{D} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ with weights $w_1, \cdots, w_n$ associated with each example respectively. The **weighted empirical risk** is then given by

$$\hat{R}_n^w(f) = \frac{1}{W}\sum_{i=1}^{n} w_i \ell(f(x_i), y_i), \tag{239}$$

where $W = \sum_{i=1}^n w_i$ is the total weight. In this way, examples with larger weights would affect the loss more. In the AdaBoost algorithm, we start with equal weights for all training points $w_1 = \cdots = w_n = 1$. Then, we train a base classifier $G_1(x)$ on the training data (note that this classifier may not fit the data well). But then we increase the weights of the points misclassified by $G_1(x)$, so that these points would affect the loss more, hopefully making further learners do better on these points where $G_1(x)$ falls short. After that, we again train a base classifier $G_2(x)$ on the training data with updated weights, and modify the weights again. We repeat this process for M times where M is the number of classifiers we plan to train, so that we have $G_1(x), \cdots, G_M(x)$. Our final prediction would then by made via

$$G(x) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right), \tag{240}$$

such that each α_m is nonnegative, and we would have larger α_m for G_m that fits its weighted training data well. Note that the **weighted** 0-1 **error** of each $G_m(x)$ is given by

$$\textsc{Error}_m = \frac{1}{W}\sum_{i=1}^{n} w_i \mathbb{1}_{y_i \neq G_m(x_i)}, \tag{241}$$

where W is again the total weight as defined before. Note that these weights are not the final weights, but the weights that G_m is trained with. Now, we determine a formula for the classifier weights α_m. That is,

$$\alpha_m = \ln\left(\frac{1 - \textsc{Error}_m}{\textsc{Error}_m}\right). \tag{242}$$

This way, higher weighted error leads to lower weight, and it is always non-negative, meeting our requirements. Now, how do we update the weights? Recall that we want to increase the weights of the points misclassified by the current

base estimator. To do this, suppose that $w_1, \cdots, w_n$ are the weights before training. If the current base estimator G_m classifies x_i correctly, then we keep w_i as it is. Otherwise, we increase x_i such that

$$w_i \leftarrow w_i e^{\alpha_m} = w_i \left(\frac{1 - \textsc{Error}_m}{\textsc{Error}_m} \right). \tag{243}$$

In this way, if G_m is a strong classifier overall, it will have low $\textsc{Error}_m$ and thus large α_m. Then if any example is misclassified even with such a strong classifier, w_i will increase to a greater extent. To sum up, we describe the AdaBoost algorithm as shown in Algorithm 9.

Algorithm 9 AdaBoost

1: Initialize observation weights $w_i = 1$, $i = 1, \cdots, n$;
2: **for** $m = 1$ to M **do**
3: Use a base learner to fit the weighted training data and obtain $G_m(x)$;
4: Compute the weighted empirical 0-1 risk $\textsc{Error}_m = \frac{1}{W} \sum_{i=1}^n w_i \mathbb{1}_{y_i \neq G_m(x_i)}$, where $M = \sum_{i=1}^n w_i$;
5: Compute the classifier weight $\alpha_m = \ln \left(\frac{1 - \textsc{Error}_m}{\textsc{Error}_m} \right)$;
6: Update the example weights such that $w_i \leftarrow w_i \exp \left(\alpha_m \mathbb{1}_{y_i \neq G_m(x_i)} \right)$ for $i = 1, \cdots, n$;
7: **end for**
8: **return** the voted classifier $G(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m G_m(x) \right)$;

4/11 Lecture

11 Gradient Boosting

Recall the AdaBoost algorithm as above. Instead of voting the final classifier $G(x)$, we may try to learn it directly. Consider the following scenario where we fit a linear combination of transformations of the input, such that

$$f(x) = \sum_{m=1}^M v_m h_m(x), \tag{244}$$

where h_m's are called **basis functions** (or **feature functions** in machine learning) such that $h_1, \cdots, h_M : \mathscr{X} \to \mathbb{R}$, and are fixed and known (*i.e.*, chosen ahead of time). For instance, if we take $h_m(x) = x^m$, then it would become polynomial regression. Now, what if we want to learn the basis functions? The base hypothesis space $\mathscr{H}$ would consist of all functions $h : \mathscr{X} \to \mathbb{R}$, and an **adaptive basis function expansion** over $\mathscr{H}$ is an ensemble model, such that

$$f(x) = \sum_{m=1}^M v_m h_m(x), \tag{245}$$

where $v_m \in \mathbb{R}$ and $h_m \in \mathscr{H}$. The combined hypothesis would then be

$$\mathscr{F}_M = \left\{ \sum_{m=1}^M v_m h_m(x);\ v_m \in \mathbb{R}, h_m \in \mathscr{H}, m = 1, \cdots, M \right\}. \tag{246}$$

The learning objective is that

$$\hat{f} = \arg\min_{f \in \mathsf{F}_m} \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i)), \tag{247}$$

for some loss functions ℓ. The ERM objective can thus be written as

$$J(v_1, \cdots, v_M, h_1, \cdots, h_M) = \frac{1}{n} \sum_{i=1}^n \ell \left(y_i, \sum_{m=1}^M v_m h_m(x) \right). \tag{248}$$

Now in order to learn this objective, suppose that our base hypothesis space $\mathscr{H}$ is parametrized by $\Theta = \mathbb{R}^b$, so that

$$J(v_1, \cdots, v_M, \theta_1, \cdots, \theta_M) = \frac{1}{n} \sum_{i=1}^{n} \ell \left(y_i, \sum_{m=1}^{M} v_m h(x; \theta_m) \right). \tag{249}$$

For some hypothesis spaces and typical loss functions, we can optimize this objective via stochastic gradient descent. However, if our base hypothesis space $\mathscr{H}$ consists of decision trees, we cannot even parametrize trees (and even if we can, the predictions would not be continuous and thus nondifferentiable). Therefore, we would like to try a greedy algorithm similar to AdaBoost, which can be applied to even non-parametric and non-differentiable basis functions, that is, **gradient boosting**, or in other words, gradient descent in the functional space. It has only two restrictions, *i.e.*, the loss function must be subdifferentiable with respect to training predictions $f(x_i)$, and we need to be able to perform regression with the base hypothesis space $\mathscr{H}$.

11.1 Forward Stagewise Additive Modeling

The goal is to fit a model

$$f(x) = \sum_{m=1}^{M} v_m h_m(x) \tag{250}$$

given some loss function. Our approach is to greedily fit one function at a time without adjusting previous functions, hence "forward stagewise". After $m-1$ stages, we would have that

$$f_{m-1}(x) = \sum_{i=1}^{m-1} v_i h_i(x), \tag{251}$$

then in the mth round, we want to find $h_m \in \mathscr{H}$ (*i.e.*, a basis function) and $v_m > 0$, such that

$$f_m(x) = \underbrace{f_{m-1}(x)}_{\text{fixed}} + v_m h_m, \tag{252}$$

improves objective function value by as much as possible. For instance for ERM, we plug in our objective function and thus we can obtain the algorithm shown as in Algorithm 10.

Algorithm 10 Forward Stagewise Additive Modeling for ERM

1: Initialize $f_0(x) = 0$;
2: **for** $m = 1$ to M **do**
3: Compute

$$(v_m, h_m) = \arg\min_{v \in \mathbb{R}, h \in H} \frac{1}{n} \sum_{i=1}^{n} \ell \left(y_i, f_{m-1}(x_i) + v h(x_i) \right); \tag{253}$$

4: $f_m \leftarrow f_{m-1} + v_m h_m$;
5: **end for**
6: **return** f_M;

Now alternatively, we consider doing binary classification. Let $\mathscr{Y} = \{-1, 1\}$ be the outcome space, $\mathscr{A} = \mathbb{R}$ be the action space, and $f : \mathscr{X} \to \mathscr{A}$ be the score function. Recall that the margin for example (x, y) is given by $m = yf(x)$, where $m > 0$ means the classification is correct, and that larger m means better confidence. As for loss function, we introduce the **exponential loss**, such that

$$\ell(y, f(x)) = \exp(-yf(x)), \tag{254}$$

i.e., $\ell(m) = \exp(-m)$, which is also an upper bound of the 0-1 loss. Now the objective function in the mth round is

$$J(v, h) = \frac{1}{n} \sum_{i=1}^{n} \exp\left(-y_i \left(f_{m-1}(x) + v h(x_i) \right) \right). \tag{255}$$

By defining $w_i^{(m)} = \exp(-y_i f_{m-1}(x_i))$, we can rewrite that

$$J(v,h) = \frac{1}{n}\sum_{i=1}^{n} w_i^{(m)} \exp\left(-y_i v h(x_i)\right) = \sum_{i=1}^{n} w_i^m \underbrace{\left(\mathbb{1}_{y_i = h(x_i)} e^{-v} + \mathbb{1}_{y_i \neq h(x_i)} e^{v}\right)}_{\text{recall } H = \{h : X \to \{-1,1\}\}} = \sum_{i=1}^{n} w_i^m \left(\left(e^{-v} - e^{v}\right)\mathbb{1}_{y_i \neq h(x_i)} + e^{-v}\right). \tag{256}$$

If $v > 0$, then

$$h_m = \arg\min_{h \in H} \; J(v,h) = \arg\min_{h \in H} \; \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}_{y_i \neq h(x_i)} = \arg\min_{h \in H} \; \frac{1}{\sum_{i=1}^{n} w_i^{(m)}} \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}_{y_i \neq h(x_i)}, \tag{257}$$

i.e., h_m is the minimizer of the weighted 0-1 loss. If we denote this weighted 0-1 error by $\textsc{Error}_m$, then we can show that the optimal v is

$$v_m = \frac{1}{2}\log\left(\frac{1 - \textsc{Error}_m}{\textsc{Error}_m}\right). \tag{258}$$

This is the same as the classifier weights in AdaBoost (differing by a constant). If $\textsc{Error}_m < 1/2$ (better than chance), then $v_m > 0$. Now we compute the weights in the next round, such that

$$\begin{aligned} w_i^{(m+1)} &= \exp\left(-y_i f_m(x_i)\right) = \exp\left(-y_i\left(f_{m-1}(x_i) + v_m h_m(x_i)\right)\right) = w_i^{(m)} \exp\left(-y_i v_m h_m(x_i)\right) \\ &= w_i^{(m)} \exp\left(-v_m \mathbb{1}_{y_i = h_m(x_i)} + v_m \mathbb{1}_{y_i \neq h_m(x_i)}\right) = w_i^{(m)} \exp\left(2 v_m \mathbb{1}_{y_i \neq h_m(x_i)}\right) \exp\left(-v_m\right). \end{aligned} \tag{259}$$

The constant scalar $\exp(-v_m)$ will get cancelled out during normalization, and compared with AdaBoost, $v_m = 2\alpha_m$. To compare the exponential loss with other loss functions for classification, the exponential loss puts a very high penalty on misclassified examples, thus not robust to outliers and noise. Also, the logistic loss performs better in settings with high Bayes error rate (intrinsic randomness in the labels). However, exponential loss has some computational advantages over logistic loss.

11.2 Gradient Boosting: AnyBoost

With squared loss, the FSAM objective function at the mth round would be

$$J(v,h) = \frac{1}{n}\sum_{i=1}^{n} \left(y_i - \left(f_{m-1}(x) + v h(x_i)\right)\right)^2. \tag{260}$$

If $\mathscr{H}$ is closed under rescaling, *i.e.*, if $h \in \mathscr{H}$ then $vh \in \mathscr{H}$ for any $h \in \mathbb{R}$, then we do not need v. Without loss of generality we take $v = 1$ and minimize

$$J(h) = \frac{1}{n}\sum_{i=1}^{n} \left(\underbrace{\left(y_i - f_{m-1}(x)\right)}_{\text{residual}} - h(x_i)\right)^2. \tag{261}$$

This is just fitting the residuals with least-squares regression. An example base hypothesis space $\mathscr{H}$ is just the regression stumps (decision stumps). For this, we consider the **functional gradient descent**, in which we want to minimize

$$J(f) = \sum_{i=1}^{n} \ell(y_i, f(x_i)). \tag{262}$$

In some sense, we want to take the gradient with respect to f, and since $J(f)$ depends only on f at the n training points, we can define the "paramters" such that $f = (f(x_1), \cdots, f(x_n))^\top$, so we can rewrite the functional gradient descent objective as

$$J(f) = \sum_{i=1}^{n} \ell(y_i, f_i). \tag{263}$$

Then, the negative gradient step direction at f would be

$$-g = -\nabla_f J(f) = -(\partial_{f_1}\ell(y_1, f_1), \cdots, \partial_{f_n}\ell(y_n, f_n)), \tag{264}$$

which we can easily calculate. $-g \in \mathbb{R}^n$ is the direction we want to change each of our n predictions on the training data. With gradient descent, our final predictor would be an additive model

$$f_0 + \sum_{m=1}^{M} v_t(-g_t). \tag{265}$$

Now the problem is, we only know how to change each of our n predictions in each step, but we also need to take a gradient step in $\mathscr{H}$. The solution is simple: we approximate by the closest base hypothesis $h \in \mathscr{H}$ (in the l^2 sense), such that

$$\min_{h \in H} \sum_{i=1}^{n} (-g_i - h(x_i))^2. \tag{266}$$

In other words, we take $h \in \mathscr{H}$ that best approximates $-g$ as our step direction. As a temporary summary, we recap what we have discussed up till now.

$$\text{Objective function} \qquad J(f) = \sum_{i=1}^{n} \ell(y_i, f(x_i)), \tag{267}$$

$$\text{Unconstrained gradient} \qquad g = \nabla_f J(f) = (\partial_{f_1}\ell(y_1, f_1), \cdots, \partial_{f_n}\ell(y_n, f_n)), \tag{268}$$

$$\text{Projected negative gradient} \qquad h = \arg\min_{h \in H} \sum_{i=1}^{n} (-g_i - h(x_i))^2, \tag{269}$$

$$\text{Gradient descent} \qquad f \leftarrow f + vh. \tag{270}$$

As for the step size (learning rate) v, we can choose by **line search**, such that

$$v_m = \arg\min_v \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i) + v h_m(x_i)\right), \tag{271}$$

or to make things simple, we may simply choose a fixed hyperparameter v. We can also perform regularization through shrinkage, such that

$$f_m \leftarrow f_{m-1} + \lambda v_m h_m, \tag{272}$$

with $\lambda \in [0, 1]$. A typical choice would be $\lambda = 0.1$. Now it suffices to choose M (the terminating round), tuned on the validation set. This version of gradient boosting algorithm can thus be summarized as in Algorithm 11.

Algorithm 11 Gradient Boosting

1: Initialize f to a constant, such that $f_0(x) \equiv \arg\min_\gamma \sum_{i=1}^{n} \ell(y_i, \gamma)$.
2: **for** $m = 1$ to M **do**
3: Compute the **pseudo-residuals** (negative gradient), such that

$$r_{i,m} = -\left(\frac{\partial}{\partial f(x_i)}\ell(y_i, f(x_i))\right)_{f(x_i)=f_{m-1}(x_i)}. \tag{273}$$

4: Fit a base learner h_m with squared loss using the dataset $\{(x_i, r_{i,m})\}_{i=1}^{n}$;
5: (Optional) Find the best step size $v_m = \arg\min_v \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i) + v h_m(x_i)\right)$;
6: $f_m \leftarrow f_{m-1} + \lambda v_m h_m$;
7: **end for**
8: **return** f_M;

We also recap the ingredients for the gradient boosting machine, that is, (1) any loss function subdifferentiable with respect to the prediction $f(x_i)$, (2) a base hypothesis space for regression, (3) `maxiter` (or a stopping criterion), and

(4) the step size methodology (fixed hyperparameter or line search). Let us apply gradient boosting with logistic loss as an example. Recall the logistic loss for binary classification with $\mathscr{Y} = \{-1, 1\}$ is given by

$$\ell(y, f(x)) = \log\left(1 + \exp(-yf(x))\right). \tag{274}$$

The pseudo-residual for the ith example is the negative derivative of loss with respect to the prediction, such that

$$r_i = -\frac{\partial}{\partial f(x_i)}\ell(y_i, f(x_i)) = -\frac{\partial}{\partial f(x_i)}\left(\log\left(1 + \exp(-y_i f(x_i))\right)\right) = \frac{y_i \exp(-y_i f(x_i))}{1 + \exp(-y_i f(x_i))} = \frac{y_i}{1 + \exp(y_i f(x_i))}. \tag{275}$$

Then we proceed to fit a base learner, and do the update step, which will not be explicitly shown here.

Last Modified: April 15, 2023.