Machine Learning

Daniel Mao

Contents

Chapter 1

Formal Setup

1.1 Definitions

Definition. *Let $\mathcal{X}$ be the domain. Let Y be the label set. We define a* ***classifier****, denoted by h, to be a function from X to Y.*

Definition (Error)**.** *Let $\mathcal{X}$ denote the domain. Let $\mathcal{D}$ be a probability distribution on X. Let f be the true labeling function on X. Let h be a classifier. We define the* ***error*** *of h, with respect to $\mathcal{D}$ and f, denoted by $\mathcal{L}_{\mathcal{D},f}$, to be a probability given by*

$$\mathcal{L}_{\mathcal{D},f}(h) := \mathcal{D}\{x \in \mathcal{X} : h(x) \neq f(x)\}.$$

- *We are never sure about the true error. We only approximate the true error.*
- *Note that there is no label set in this definition. What we have is the true labeling function f.*

1.2 Empirical Risk Minimization (ERM)

Definition (Empirical Error)**.** *Let $\mathcal{S}$ be a training data. Let h be a classifier. We define the* ***empirical error****, with respect to $\mathcal{S}$, denoted by $L_{\mathcal{S}}$, to be a proportion given by*

$$L_{\mathcal{S}}(h) := \frac{|\{i : h(x_i) \neq y_i\}|}{|S|}.$$

Definition (The Realizability Assumption)**.** *Assume that $\exists h^* \in \mathcal{H}$ such that $\mathcal{L}_{\mathcal{D},f}(h^*) = 0$.*

Theorem 1. *Assume that the data are independent and identically distributed. Assume realizability. Then if $\mathcal{H}$ is a finite class of classifiers, the $ERM(H)$ algorithm is guaranteed to succeed.*

Intuitively, the training set S is a window through which the learner gets partial information about the distribution D over the world and the labeling function, f. The larger the sample gets, the more likely it is to reflect more accurately the distribution and labeling used to generate it.

Proof Idea. Given some error rate ε and a sample of size m, what is the probability that $ERM(H)$ will have error ε.

$$\Pr\left(\mathcal{L}_{D,f}(A[S]) > \varepsilon\right).$$

Let's call a sample S "BAD" if $\exists h \in H$ such that $L_S(h) = 0$ but $L_{\mathcal{D},f}(h) > \varepsilon$. Let's show that the probability of picking a BAD sample is small. For every given h, such that $L_{D,f}(h) > \varepsilon$, the probability of picking x on which h is correct is $\leq 1-\varepsilon$. So by independence,

$$D^m\{S : S \text{ is BAD w.r.t. this } h\} \leq (1-\varepsilon)^m.$$

Given a BAD h, the probability of S failing to show that this h is BAD $\leq (1-\varepsilon)^m$. The probability that S is BAD, S fails any $h \in H$, by the Union Bound, is $\leq \sum_{BAD h \in H} \Pr[S fails to alert on h]$. $\leq |H| \cdot (1-\varepsilon)^m$.

<u>Conclusion</u>

The prob of picking an S that will mislead $ERM(H)$ to pick some h with true error $> \varepsilon$ is at most $|H| \cdot (1-\varepsilon)^m$. So

$$\Pr\left(\mathcal{L}_{\mathcal{D},f}(ERM(H)(S)) > \varepsilon\right) < |H| \cdot (1-\varepsilon)^m \overset{m\to\infty}{\longrightarrow} 0.$$

■

Formal Proof. Let $\mathcal{X}$ denote the domain. Let $\mathcal{Y}$ denote the labeling set. Let $\mathcal{D}$ be some probability distribution on $\mathcal{X}$. Let f be some true labeling function from $\mathcal{X}$ to $\mathcal{Y}$. Let $\mathcal{H}$ be a finite class of classifiers. Define for each sample S a classifier h_S as $h_S := \operatorname{argmin}_{h\in\mathcal{H}} \mathcal{L}_{S,f}(h)$. The realizability assumption implies that $\mathcal{L}_{S,f}(h_S) = 0$. Define a set $\mathcal{H}_B$ as

$$\mathcal{H}_B := \{h \in \mathcal{H} : \mathcal{L}_{\mathcal{D},f}(h) > \varepsilon\}.$$

i.e., $\mathcal{H}_B$ is the set of bad hypotheses. Define a set M as

$$M := \{S \in \mathcal{X}^m : \exists h \in \mathcal{H}, \mathcal{L}_{\mathcal{D},f}(h) > \varepsilon \text{ and } \mathcal{L}_{S,f}(h) = 0\}.$$

i.e., M is the set of misleading samples. Note that

$$\{S \in \mathcal{X}^m : \mathcal{L}_{\mathcal{D},f}(h_S) > \varepsilon\} \subseteq M.$$

So

$$\begin{aligned}
\mathcal{D}^m\{S \in \mathcal{X}^m : \mathcal{L}_{\mathcal{D},f}(h_S) > \varepsilon\} &\leq \mathcal{D}^m(M) \\
&= \mathcal{D}^m \bigcup_{h \in \mathcal{H}_B} \{S \in \mathcal{X}^m : \mathcal{L}_{S,f}(h) = 0\} \\
&\leq \sum_{h \in \mathcal{H}_B} \mathcal{D}^m\{S \in \mathcal{X}^m : \mathcal{L}_{S,f}(h) = 0\} \\
&= \sum_{h \in \mathcal{H}_B} \mathcal{D}^m\{S \in \mathcal{X}^m : \forall x \in S, h(x) = f(x)\} \\
&= \sum_{h \in \mathcal{H}_B} \mathcal{D}^m \prod_{i=1}^{m} \{x \in \mathcal{X} : h(x) = f(x)\} \\
&= \sum_{h \in \mathcal{H}_B} \prod_{i=1}^{m} \mathcal{D}\{x \in \mathcal{X} : h(x) = f(x)\} \\
&= \sum_{h \in \mathcal{H}_B} \prod_{i=1}^{m} (1 - \mathcal{L}_{\mathcal{D},f}(h)) \\
&\leq \sum_{h \in \mathcal{H}_B} \prod_{i=1}^{m} (1 - \varepsilon) \\
&= \sum_{h \in \mathcal{H}_B} (1 - \varepsilon)^m \\
&= |\mathcal{H}|(1 - \varepsilon)^m.
\end{aligned}$$

■

Chapter 2

Perceptron

2.1 The Perceptron Algorithm

Algorithm 1: The Perceptron Algorithm (Rosenblatt 1958)

Input: Dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1..n\}$, initialization $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$.

Output: Approximate solutions $\boldsymbol{w}$ and b so that $\forall i \in \{1..n\}$, $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 0$.

```
1 while true do
2     receive training example index $i \in \{1..n\}$;
3     if $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \leq \delta$ then
4         $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$;
5         $b \leftarrow b + y_i$;
```

Remark. *We only perform updates when we make a mistake, and this is a necessary rule.*

Theorem 2. *Assume there exists some $\boldsymbol{z}$ such that $A^\top \boldsymbol{z} > 0$, then the perceptron algorithm converges to some $\boldsymbol{z}^*$. If each column of A is selected indefinitely often, then $A^\top \boldsymbol{z}^* > \delta$.*

Corollary. *Let $\delta = 0$ and $\boldsymbol{z}_0 = \boldsymbol{0}$. Then the perceptron algorithm converges after at most $(R/\gamma)^2$ steps, where R and γ are dataset properties given by*

$$R := \|A\|_{2,\infty} = \max_i \|\boldsymbol{a}_i\|_2 \quad \text{and} \quad \gamma := \max_{\|z\|_2 \leq 1} \min_i \langle \boldsymbol{z}, \boldsymbol{a}_i \rangle.$$

Note that the parameters R and γ are independent of the dataset size n.

Theorem 3. *The iterate $\boldsymbol{z} = (\boldsymbol{w}; b)$ of the perceptron algorithm is always bounded. In particular, if there is no separating hyperplane, then perceptron cycles.*

Chapter 3

Logistic Regression

3.1 Logit Transform

Definition (Logit Transform)**.**

$$\log \frac{p(\boldsymbol{x}; \boldsymbol{w})}{1 - p(\boldsymbol{x}; \boldsymbol{w})} = \boldsymbol{w}^\top \boldsymbol{x}.$$

Or equivalently,

$$p(\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})}.$$

This $p(\boldsymbol{x}; \boldsymbol{w})$ is the confidence in the prediction.

3.2 Hypothesis Representation

Logistic Regression Model

Want $0 \le h_\theta(x) \le 1$.

$$h_\theta(x) = g(\theta^\top x)$$

where

$$g(z) = \frac{1}{1 + e^{-z}}.$$

So

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}.$$

Sigmoid function (logistic function)

$$g(z) = \frac{1}{1 + e^{-z}}.$$

Properties:

- $g(0) = 0.5$.
- $\lim_{z\to+\infty} g(z) = 1$.
- $\lim_{z\to-\infty} g(z) = 0$.

Interpretation of the Hypothesis Output

$h_\theta(x)$ is the probability that $y = 1$ on input x. i.e. $h_\theta(x) = P(y = 1 \mid x; \theta)$, the probability that $y = 1$, given x, parameterizad by θ.

Since y is either 0 or 1,

$$P(y = 0 \mid x; \theta) + P(y = 1 \mid x; \theta) = 1, \text{ or}$$
$$P(y = 0 \mid x; \theta) = 1 - P(y = 1 \mid x; \theta).$$

3.3 Decision Boundary

Proposition 3.3.1. *The decision boundary $\mathcal{D}$ of logistic regression with parameter $\boldsymbol{w} \in \mathbb{R}^n$ is*

$$\mathcal{D} = \{x \in \mathbb{R}^n : \boldsymbol{w}^\top \boldsymbol{x} = 0\}.$$

Proof. Predict $y = 1$ if $h_{\boldsymbol{w}}(x) \geq 0.5$ and predict $y = 0$ if $h_{\boldsymbol{w}}(x) < 0.5$. Notice

$$h_{\boldsymbol{w}}(x) = 0.5 \iff \boldsymbol{w}^\top x = 0.$$

So the model predicts $\hat{y} = \mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x})$ but with confidence $p(\boldsymbol{x}; \boldsymbol{w})$. ■

Example 3.3.1. *Consider* $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$. *Predict "$y = 1$" if $\theta^\top x = -3 + x_1 + x_2 \geq 0$. That is, $x_2 \geq 3 - x_1$. This gives a "half space" in $\mathbb{R}^2$. The line $x_1 + x_2 = 3$ separates $\mathbb{R}^2$ into a region where the model predicts "$y = 0$" and a region where the model predicts "$y = 1$".*

3.4 Maximum Likelihood Estimation

Proposition 3.4.1 (Maximum Likelihood Estimation of the Parameter $\boldsymbol{w}$)**.** *The maximum likelihood estimation of $\boldsymbol{w}$ is given by the following update rule:*

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - (\boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{p})$$

where

$$\boldsymbol{p} = \left[\frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}\right]_{i=1}^{n} \quad \text{and} \quad \boldsymbol{W} = \mathrm{diag}(p_i(1 - p_i))_{i=1}^{n}.$$

Proof. The maximum likelihood function $\mathcal{L}$ is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}_1, ..., \boldsymbol{x}_n; \boldsymbol{w}) &= \prod_{i=1}^{n} f_i(\boldsymbol{x}_i; \boldsymbol{w}) \\ &= \prod_{i=1}^{n} \begin{cases} \Pr(y = 1 \mid \boldsymbol{X} = \boldsymbol{x}_i), & \text{if } y_i = 1 \\ \Pr(y = 0 \mid \boldsymbol{X} = \boldsymbol{x}_i), & \text{if } y_i = 0 \end{cases} \\ &= \prod_{i=1}^{n} \big[\Pr(y = 1 \mid \boldsymbol{X} = \boldsymbol{x}_i)^{y_i} \Pr(y = 0 \mid \boldsymbol{X} = \boldsymbol{x}_i)^{1-y_i} \big].\end{aligned}$$

So the log maximum likelihood function ℓ is:

$$\begin{aligned}\ell(\boldsymbol{x}_1, ..., \boldsymbol{x}_n; \boldsymbol{w}) &= \log \mathcal{L}(\boldsymbol{x}_1, ..., \boldsymbol{x}_n; \boldsymbol{w}) \\ &= \log \prod_{i=1}^{n} \big[\Pr(y = 1 \mid \boldsymbol{X} = \boldsymbol{x}_i)^{y_i} \Pr(y = 0 \mid \boldsymbol{X} = \boldsymbol{x}_i)^{1-y_i} \big] \\ &= \sum_{i=1}^{n} \log \big[\Pr(y = 1 \mid \boldsymbol{X} = \boldsymbol{x}_i)^{y_i} \Pr(y = 0 \mid \boldsymbol{X} = \boldsymbol{x}_i)^{1-y_i} \big] \\ &= \sum_{i=1}^{n} \big[y_i \log \Pr(y = 1 \mid \boldsymbol{X} = \boldsymbol{x}_i) + (1 - y_i) \log \Pr(y = 0 \mid \boldsymbol{X} = \boldsymbol{x}_i) \big] \\ &= \sum_{i=1}^{n} \left[y_i \log \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} + (1 - y_i) \log \frac{1}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \right] \\ &= \sum_{i=1}^{n} \left[y_i \big[\boldsymbol{w}^\top \boldsymbol{x}_i - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)) \big] - (1 - y_i) \big[\log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)) \big] \right] \\ &= \sum_{i=1}^{n} \left[y_i \boldsymbol{w}^\top \boldsymbol{x}_i - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)) \right].\end{aligned}$$

So the derivative of ℓ, with respect to $\boldsymbol{w}$, is:

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{w}} \ell(\boldsymbol{x}_1, ..., \boldsymbol{x}_n; \boldsymbol{w}) &= \frac{\partial}{\partial \boldsymbol{w}} \sum_{i=1}^{n} \left[y_i \boldsymbol{w}^\top \boldsymbol{x}_i - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)) \right] \\ &= \sum_{i=1}^{n} \left[\frac{\partial}{\partial \boldsymbol{w}} y_i \boldsymbol{w}^\top \boldsymbol{x}_i - \frac{\partial}{\partial \boldsymbol{w}} \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)) \right] \\ &= \sum_{i=1}^{n} \left[y_i \boldsymbol{x}_i^\top - \frac{1}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \exp(\boldsymbol{w}^\top \boldsymbol{x}_i) \boldsymbol{x}_i^\top \right] \\ &= \sum_{i=1}^{n} \left[y_i - \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \right] \boldsymbol{x}_i^\top \\ &= \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{p}).\end{aligned}$$

So the second derivative of ℓ, with respect to $\boldsymbol{w}$, is:

$$\frac{\partial^2}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top} \ell(\boldsymbol{x}_1, ..., \boldsymbol{x}_n; \boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}^\top} \frac{\partial}{\partial \boldsymbol{w}} \ell(\boldsymbol{x}_1, ..., \boldsymbol{x}_n; \boldsymbol{w})$$

$$
\begin{aligned}
&= \frac{\partial}{\partial \boldsymbol{w}^\top} \sum_{i=1}^{n} \left[y_i - \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}{1+\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \right] \boldsymbol{x}_i^\top \\
&= -\sum_{i=1}^{n} \frac{\partial}{\partial \boldsymbol{w}^\top} \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}{1+\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \boldsymbol{x}_i^\top \\
&= -\sum_{i=1}^{n} \frac{\exp(-\boldsymbol{w}^\top \boldsymbol{x}_i)\boldsymbol{x}_i}{(1+\exp(-\boldsymbol{w}^\top \boldsymbol{x}_i))^2} \boldsymbol{x}_i^\top \\
&= -\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^\top \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)}{1+\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \frac{1}{1+\exp(\boldsymbol{w}^\top \boldsymbol{x}_i)} \\
&= \boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top.
\end{aligned}
$$

So the update rule is:

$$
\begin{aligned}
\boldsymbol{w}_{k+1} &= \boldsymbol{w}_k - \left(\frac{\partial^2 \ell}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top} \right)^{-1} \left(\frac{\partial \ell}{\partial \boldsymbol{w}} \right) \\
&= \boldsymbol{w}_k - (\boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top)^{-1} \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{p}).
\end{aligned}
$$

■

3.5 Cost Function

Training set: $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$. $x_0 = 1$. $x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$. $y \in \{0, 1\}$.

If

$$
cost(h_\theta(x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2
$$

as in linear regression, then the cost function is non-convex.

Logistic Regression Cost Function

$$
cost(h_\theta(x), y) := \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0. \end{cases}
$$

Property:

- If $y = 1$ and $h_\theta(x) \uparrow 1$, then $cost(h_\theta(x), y) \approx 0$.
- If $y = 1$ and $h_\theta(x) \downarrow 0$, then $cost(h_\theta(x), y) \to +\infty$.
- Similar for $y = 0$.

Captures intuition that if $h_\theta(x) = 0$ (the model predicts $P(y = 1 \mid x; \theta) = 0$), but $y = 1$, we'll penalize the learning algorithm by a very large cost.

Cost Function for the Training Set

$$J(\theta) := \frac{1}{m}\sum_{i=1}^{m} cost(h_\theta(x^{(i)}, y^{(i)}))$$

where *cost* is the function given above.

Rewrite the cost function:

$$cost(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x)).$$

So

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)\right].$$

To fit parameter θ: $\min_\theta J(\theta)$.

3.6 Optimization

Gradient Descent

$$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}.$$

This looks identical to linear regression.

Other Optimization Algorithms

- Conjugate Gradient
- BFGS
- L-BFGS

Advantages:

- No need to pick learning rate α.
- Often faster than gradient descent.

3.7 Multiclass Classification

One-vs-rest

$$h_\theta^{(i)}(x) = P(y = i \mid x; \theta) \quad i = 1, 2, 3.$$

Train a logistic regression classifier for each i to predict the probability that $y = i$. On a new input x, pick class $i = \underset{i}{\text{argmax}} h_\theta^{(i)}(x)$.

Softmax

$$\mathbb{P}(Y = k \mid \boldsymbol{x}; \boldsymbol{W}) = \frac{\exp(\boldsymbol{w}_k^\top \boldsymbol{x})}{\sum_{l=1}^C \exp(\boldsymbol{w}_l^\top \boldsymbol{x})}.$$

3.8 Introducing Non-linearity

Consider

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

and $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$. Then the model predicts "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$. That is, if $x_1^2 + x_2^2 \geq 1$.

3.9 Beyond Logistic Regression

The logit transform is just one choice among the others. What we need is a function Q from $[0, 1]$ onto $\mathbb{R}$. The inverse of any cumulative distribution function can be used as the Q function. e.g. the probit regression which uses the inverse of the CDF of a standard normal distribution.

Definition (Logistic Distribution)**.** *We define the **logistic distribution** to be a probability distribution with CDF given by*

$$F(x; \mu; s) := \frac{1}{1 + \exp(-\frac{x-\mu}{s})}.$$

Proposition 3.9.1 (Mean of Logistic Distribution)**.** *Let $X \sim F(x; \mu; s)$. Then*

$$\mathbb{E}[X] = \mu.$$

Proposition 3.9.2. *Let $X \sim F(x; \mu; s)$. Then*

$$\text{var}[X] = \frac{s^2 \pi^2}{3}.$$

Chapter 4

Dimensionality Reduction

4.1 Motivations

- Data compression.
- Data visualization.

4.2 Principal Component Analysis

PCA is trying to find a lower dimensional surface onto which to project the data so as to minimize squared projection error.

Principal component analysis vs. linear regression:

- Linear regression tries to minimize residuals. PCA tries to minimize projection error.
- In linear regression, we try to predict one feature with other features. In PCA, all features are equivalent.

Algorithm: (from n dimensions to k dimensions)

- Preprocessing: mean normalization and feature scaling (standardization).
- Compute covariance matrix
$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T.$$
- Compute the eigenvalues of Σ.
$$\Sigma = USV.$$
Let U_{reduce} denote the first k columns of U. Then $z = U_{reduce}^T x$.

Choosing k:

- Average squared projection error:
$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{approx}\|^2.$$
- Total variation in the data:
$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2.$$
- Typically, we choose k to be the smallest value so that
$$\frac{\text{average squared projection error}}{\text{total variation}} < 0.01,$$
i.e. 99% of variance is retained.
- The above ratio can be computed in an easier way. Say
$$S = \begin{bmatrix} s_{11} & & O \\ & \ddots & \\ O & & s_{nn} \end{bmatrix}.$$
Then
$$\frac{\text{average squared projection error}}{\text{total variation}} = 1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}}.$$
So we can start with $k = 1$ and increase k and check whether
$$\frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} > 0.99.$$

Chapter 5

Support Vector Machines

5.1 Hard-Margin SVM

Recall that the perceptron algorithm is

$$\min 0$$

$$\text{subject to } \forall i, y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 0.$$

This is a feasibility problem.

$$\begin{aligned} \max_{\boldsymbol{w}, b} \quad & \frac{1}{\|\boldsymbol{w}\|_2} \\ \text{subject to} \quad & \forall i, y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1. \end{aligned}$$

The objective function is the margin between the hyperplanes $H_{+1} : \boldsymbol{w}^\top \boldsymbol{x}_i + b = 1$ and $H_{-1} : \boldsymbol{w}^\top \boldsymbol{x}_i + b = -1$.

Note that this problem is equivalent to

$$\begin{aligned} \max_{\boldsymbol{w}, b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{subject to} \quad & \forall i, y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1. \end{aligned}$$

This is a quadratic programming whereas the perceptron algorithm is a linear programming.

Definition (Support Vectors)**.** *We define the* ***support vectors*** *to be the points that are on the hyperplanes H_{+1} and H_{-1}.*

Proposition 5.1.1 (Existence)**.** *Assume that the data points are linearly separable. Then the minimizers $\boldsymbol{w}$ and b exist.*

Proposition 5.1.2. *Assume that the data points are linearly separable. Then the minimizers $\boldsymbol{w}$ and b exist and are unique.*

Proposition 5.1.3. *The Lagrangian dual problem is*

$$\begin{aligned} \max_{\alpha \geq 0} \quad & \sum_i \alpha_i - \frac{1}{2} \| \sum_i \alpha_i y_i \boldsymbol{x}_i \|_2^2 \\ \textit{subject to} \quad & \sum_i \alpha_i y_i = 0, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \min_{\alpha \geq 0} \quad & \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j - \sum_k \alpha_k \\ \textit{subject to} \quad & \sum_i \alpha_i y_i = 0. \end{aligned}$$

Proposition 5.1.4. *Define for any $d \in \mathbb{N}$ a set $\Delta \subseteq \mathbb{R}^d$ as*

$$\Delta := \{\boldsymbol{x} \in \mathbb{R}^d : \sum_i \boldsymbol{x}_i = 1\}.$$

$$\begin{aligned} \min_{\bar{\alpha} \in 2\Delta} \quad & \frac{1}{2} \| \sum_i \bar{\alpha}_i y_i \boldsymbol{x}_i \|_2^2 \\ \textit{subject to} \quad & \sum_i \bar{\alpha}_i y_i = 0. \end{aligned}$$

Define sets P and N as

$$P := \{i : y_i = 1\} \textit{ and } N := \{i : y_i = -1\}.$$

Define vectors μ and ν as

$$\mu := [\alpha_i]_{i \in P} \textit{ and } \nu := [\alpha_i]_{i \in N}.$$

Then the problem is equivalent to

$$\min_{\mu \in \Delta, \nu \in \Delta} \quad \frac{1}{2} \left\| \sum_{i \in P} \mu_i \boldsymbol{x}_i - \sum_{j \in N} \nu_j \boldsymbol{x}_j \right\|_2 .$$

Note that $\sum_{i \in P} \mu_i \boldsymbol{x}_i \in \operatorname{conv}\{\boldsymbol{x}_i : i \in P\}$ and $\sum_{j \in N} \nu_j \boldsymbol{x}_j \in \operatorname{conv}\{\boldsymbol{x}_i : i \in P\}$. So the objective function is the distance between the two convex hulls. So $\boldsymbol{w}$ is in the direction of the line segment between the pair of points in the positive/negative convex hull with the minimum distance, and the hyperplane is the bisector of this line segment.

Chapter 6

Overfitting

6.1 Introduction

Large neural nets trained on relatively small datasets can overfit the training data. This has the effect of the model learning the statistical noise in the training data, which results in poor performance when the model is evaluated on new data, e.g. a test dataset. Generalization error increases due to overfitting.

6.2 Dropout

What

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "dropped out". This has the effect of making the layer look like and be treated like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "view" of the configured layer. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. This conceptualization suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust.

When to Use

Dropout regularization is a generic approach. It can be used with most, perhaps all, types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers such as the long short-term memory network layer. In the case of LSTMs, it may be desirable to use different dropout rates for the input and recurrent connections. In fact, a large network (more nodes per layer) may be required as dropout will probabilistically reduce the capacity of the network. A good rule of thumb is to divide the number of nodes in the layer before dropout by the proposed dropout rate and use that as the number of nodes in the new network that uses dropout. For example, a network with 100 nodes and a proposed dropout rate of 0.5 will require 200 nodes (100 / 0.5) when using dropout.

Where to Add

Dropout is implemented per-layer in a neural network. Dropout may be implemented on any or all hidden layers in the network as well as the input (visible) layer. It is not used on the output layer. The term "dropout" refers to dropping out units (hidden and visible) in a neural network.

Dropout Rate

The default interpretation of the dropout hyperparameter is the probability of training a given node in a layer, where 1.0 means no dropout, and 0.0 means no outputs from the layer. A good value for dropout in a hidden layer is between 0.5 and 0.8. A common value is a probability of 0.5 for retaining the output of each node in a hidden layer and a value close to 1.0, such as 0.8, for retaining inputs from the visible layer. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5. Dropout is not used after training when making a prediction with the fit network.

Finalizing the Network

The weights of the network will be larger than normal because of dropout. Therefore, before finalizing the network, the weights are first scaled by the chosen dropout rate. The network can then be used as per normal to make predictions. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time. The rescaling of the weights can be performed at training time instead, after each weight update at the end of the mini-batch. This is sometimes called "inverse dropout" and does not require any modification of weights during training. Both the Keras and PyTorch deep learning libraries implement dropout in this way.

Weight Constraint

Network weights will increase in size in response to the probabilistic removal of layer activations. Large weight size can be a sign of an unstable network. To counter this effect, a weight constraint can be imposed to force the norm (magnitude) of all weights in a layer to be below a specified value. For example, the maximum norm constraint is recommended with a value between 3-4.

In Practice

Dropout works well in practice, perhaps replacing the need for weight regularization (e.g. weight decay) and activity regularization (e.g. representation sparsity). Dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization. Dropout may also be combined with other forms of regularization to yield a further improvement.

Like other regularization methods, dropout is more effective on those problems where there is a limited amount of training data and the model is likely to overfit the training data. Problems where there is a large amount of training data may see less benefit from using dropout. For very large datasets, regularization confers little reduction in generalization error. In these cases, the computational cost of using dropout and larger models may outweigh the benefit of regularization.

6.3 Ensembling

Chapter 7

Handling Imbalanced Datasets

7.1 The Problem Described

The imbalanced dataset is extremely common when handling real-world scenarios. A machine learning model is not robust if it uses an imbalanced dataset for training purposes. Therefore, a balanced dataset is preferred for training machine learning models.

7.2 Change the Metric

Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be very misleading. Metrics that can provide better insight include:

- Confusion Matrix: a table showing correct predictions and types of incorrect predictions.
- Precision: the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- Recall: the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- F1: Score: the weighted average of precision and recall.

7.3 Change the Algorithm

Decision trees frequently perform well on imbalanced data. They work by learning a hierarchy of if/else questions and this can force both classes to be addressed.

7.4 Change the Dataset

7.4.1 Oversampling

Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don't have a ton of data to work with.

Always split into test and train sets BEFORE trying oversampling techniques! Oversampling before splitting the data can allow the exact same observations to be present in both the test and train sets. This can allow our model to simply memorize specific data points and cause overfitting and poor generalization to the test data.

7.4.2 Undersampling

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data - think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

7.4.3 Generate Synthetic Samples: SMOTE

A technique similar to upsampling is to create synthetic samples. Here we will use imblearn's SMOTE or Synthetic Minority Oversampling Technique. SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.

Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data.