3 Logistic Regression

Goal

Understand logistic regression. Confidence. Comparison with linear regression. Newton's algorithm.

Alert 3.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Unlike the slides, in this note we encode the label $\mathsf{y} \in \{\pm 1\}$ and we arrange $\mathbf{x}_i$ in the columns of X.

We use $\mathbf{x}$ and $\mathbf{w}$ for the original vectors and $\mathbf{x}$ and $\mathbf{w}$ for the padded versions (with constant 1 and bias b respectively). Similar, we use X and W for the original matrices and $\mathbf{X}$ and $\mathbf{W}$ for the padded versions.

Remark 3.2: Confidence of prediction

In perceptron we make predictions directly through a linear threshold function:

$$\hat{\mathsf{y}} = \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b).$$

Often, we would also like to know how confident we are about this prediction $\hat{\mathsf{y}}$. For example, we can use the magnitude $|\mathbf{w}^\top \mathbf{x} + b|$ as the indication of our "confidence." This choice, however, can be difficult to interpret at times, after all the magnitude could be any positive real number.

In the literature there are many attempts to turn the real output of a classifier into probability estimates, see for instance Vovk and Petej (2014) and Vovk et al. (2015).

Vovk, V. and I. Petej (2014). "Venn-Abers Predictors". In: *Conference on Uncertainty in Artificial Intelligence UAI.*

Vovk, V., I. Petej, and V. Fedorova (2015). "Large-scale probabilistic predictors with and without guarantees of validity". In: *Advances in Neural Information Processing Systems 28.*

Remark 3.3: Reduce classification to regression?

Recall that the optimal Bayes classifier is

$$h^\star(\mathbf{x}) = \operatorname{sign}(2\eta(\mathbf{x}) - 1), \quad \text{where} \quad \eta(\mathbf{x}) = \Pr(\mathsf{Y} = 1 | \mathsf{X} = \mathbf{x}).$$

The posterior probability $\eta(\mathbf{x})$ is a perfect measure of our confidence in predicting $\hat{\mathsf{y}} = h^\star(\mathbf{x})$. Therefore, one may attempt to estimate the posterior probability

$$\eta(\mathbf{x}) = \Pr(\mathsf{Y} = 1 | \mathsf{X} = \mathbf{x}) = \mathbb{E}(\mathbf{1}_{\mathsf{Y}=1} | \mathsf{X} = \mathbf{x}).$$

If we define $\tilde{\mathsf{Y}} = \mathbf{1}_{\mathsf{Y}=1} \in \{0, 1\}$, then $\eta(\mathsf{X})$ is exactly the regression function of $(\mathsf{X}, \tilde{\mathsf{Y}})$, see Definition 2.8. So, in principle, we could try to estimate the regression function based on some i.i.d. samples $\{(\mathsf{X}_i, \tilde{\mathsf{Y}}_i) : i = 1, \dots, n\}$.

The issue with the above approach is that we are in fact reducing an easier problem (classification) to a more general hence harder problem (regression). Note that the posterior probability $\eta(\mathbf{x})$ always lies in $[0, 1]$, and we would like to exploit this *a priori* knowledge. However, a generic approach to estimate the regression function would not be able to take this structure into account. In fact, an estimate of the regression function (e.g. through linear regression) may not always take values in $[0, 1]$ at all.

As a practical rule of thumb: Never try to solve a more general problem than necessary. (Theoreticians violate this rule all the time but nothing is meant to be practical in theory anyways.)

Definition 3.4: Bernoulli model

Let us consider the binary classification problem with labels $\mathsf{y} \in \{\pm 1\}$. With the parameterization:

$$\Pr(\mathsf{Y} = 1 | \mathsf{X} = \mathbf{x}) =: p(\mathbf{x}; \mathbf{w}), \tag{3.1}$$

where p is a function that maps $\mathbf{x}$ and $\mathbf{w}$ into $[0, 1]$, we then have the Bernoulli model for generating the label $\mathsf{y} \in \{\pm 1\}$:

$$\Pr(\mathsf{Y} = \mathsf{y} | \mathsf{X} = \mathbf{x}) = p(\mathbf{x}; \mathbf{w})^{(1+\mathsf{y})/2} [1 - p(\mathbf{x}; \mathbf{w})]^{(1-\mathsf{y})/2}.$$

Let $\mathcal{D} = \{(\mathbf{x}_i, \mathsf{y}_i) : i = 1, \ldots, n\}$ be an i.i.d. sample from the same distribution as (X, Y). The conditional likelihood factorizes under the i.i.d. assumption:

$$\begin{aligned} \Pr(\mathsf{Y}_1 = \mathsf{y}_1, \ldots, \mathsf{Y}_n = \mathsf{y}_n | \mathsf{X}_1 = \mathbf{x}_1, \ldots, \mathsf{X}_n = \mathbf{x}_n) &= \prod_{i=1}^n \Pr(\mathsf{Y}_i = \mathsf{y}_i | \mathsf{X}_i = \mathbf{x}_i) \\ &= \prod_{i=1}^n p(\mathbf{x}_i; \mathbf{w})^{(1+\mathsf{y}_i)/2} [1 - p(\mathbf{x}_i; \mathbf{w})]^{(1-\mathsf{y}_i)/2}. \end{aligned} \tag{3.2}$$

A standard algorithm in statistics and machine learning for parameter estimation is to maximize the (conditional) likelihood. In this case, we can maximize (3.2) w.r.t. $\mathbf{w}$. Once we figure out $\mathbf{w}$, we can then make probability estimates on any new test sample $\mathbf{x}$, by simply plugging $\mathbf{w}$ and $\mathbf{x}$ into (3.1).

Example 3.5: What is that function $p(\mathbf{x}; \mathbf{w})$?

Let us consider two extreme cases:

- $p(\mathbf{x}; \mathbf{w}) = p(\mathbf{x})$, i.e., the function p can take any value on any $\mathbf{x}$. This is the extreme case where anything we learn from one data point $\mathbf{x}_i$ may have nothing to do with what $p(\mathbf{x}_j)$ can take. Denote $p_i = p(\mathbf{x}_i)$, take logarithm on (3.2), and negate:

  $$\min_{p_1, \ldots, p_n} \; -\frac{1}{2} \sum_{i=1}^n (1 + \mathsf{y}_i) \log p_i + (1 - \mathsf{y}_i) \log(1 - p_i).$$

  Since the p_i's are not related, we can solve them separately. Recall the definition of the KL divergence, we know

  $$-\tfrac{1+\mathsf{y}_i}{2} \log p_i - \tfrac{1-\mathsf{y}_i}{2} \log(1 - p_i) = \mathsf{KL}\left(\begin{pmatrix} \frac{1+\mathsf{y}_i}{2} \\ \frac{1-\mathsf{y}_i}{2} \end{pmatrix} \middle\| \begin{pmatrix} p_i \\ 1 - p_i \end{pmatrix} \right) - \tfrac{1+\mathsf{y}_i}{2} \log \tfrac{1+\mathsf{y}_i}{2} - \tfrac{1-\mathsf{y}_i}{2} \log \tfrac{1-\mathsf{y}_i}{2}.$$

  Since the KL divergence is nonnegative, to maximize the conditional likelihood we should set

  $$p_i = \tfrac{1+\mathsf{y}_i}{2}.$$

  This result does make sense, since for $\mathsf{y}_i = 1$ we set $p_i = 1$ while for $\mathsf{y}_i = -1$ we set $p_i = 0$ (so that $1 - p_i = 1$, recall that p_i is the probability of y_i being 1).

- $p(\mathbf{x}; \mathbf{w}) = p(\mathbf{w}) = p$, i.e., the function p is independent of $\mathbf{x}$ hence is a constant. The is the extreme case where anything we learn from one data point immediately applies in the same way to any other data point. Similar as above, we find p by solving

  $$\min_{p} \; -\frac{1}{2} \sum_{i=1}^n (1 + \mathsf{y}_i) \log p + (1 - \mathsf{y}_i) \log(1 - p).$$

  Let $\bar{p} = \frac{1}{n} \sum_{i=1}^n \frac{1+\mathsf{y}_i}{2}$, which is exactly the fraction of positive examples in our training set $\mathcal{D}$. Obviously then $1 - \bar{p} = \frac{1}{n} \sum_{i=1}^n \frac{1-\mathsf{y}_i}{2}$. Prove by yourself that $\bar{p}$ is indeed the optimal choice. This again makes sense: if we have to pick one and only one probability estimate (for every data point), intuitively we should just use the fraction of positive examples in our training set.

The above two extremes are not satisfactory: it is either too flexible by allowing each data point to have its own probability estimate (which may have nothing to do with each other hence learning is impossible) or it is too inflexible by restricting every data point to use the same probability estimate. Logistic regression, which we define next, is an interpolation between the two extremes.

Definition 3.6: Logistic Regression (Cox 1958)

Motivated by the two extreme cases in Example 3.5, we want to parameterize $p(\mathbf{x};\mathbf{w})$ in a not-too-flexible and not-too-inflexible way. One natural choice is to set p as an affine function (how surprising): $p(\mathbf{x};\mathbf{w}) = \langle \mathbf{x}, \mathbf{w}\rangle + b$. However, this choice has the disadvantage in the sense that the left-hand side takes value in $[0,1]$ while the right-hand side takes value in $\mathbb{R}$. To avoid this issue, we first take a logit transformation of p and then equate it to an affine function:

$$\log \frac{p(\mathbf{x};\mathbf{w})}{1-p(\mathbf{x};\mathbf{w})} = \langle \mathbf{x}, \mathbf{w}\rangle + b.$$

The ratio on the left-hand side is known as odds ratio (probability of 1 divide by probability of -1). Or equivalently,

$$p(\mathbf{x};\mathbf{w}) = \frac{1}{1+\exp(-\langle \mathbf{x}, \mathbf{w}\rangle - b)} = \mathtt{sgm}(\langle \mathbf{x}, \mathbf{w}\rangle + b), \quad \text{where} \quad \mathtt{sgm}(t) = \frac{1}{1+\exp(-t)} = \frac{\exp(t)}{1+\exp(t)} \tag{3.3}$$

is the so-called sigmoid function. Note that our definition of p involves $\mathbf{x}$ but not the label $\mathtt{y}$. This is crucial as later on we will use $p(\mathbf{x};\mathbf{w})$ to predict the label $\mathtt{y}$.

Plugging (3.3) into the conditional likelihood (3.2) and maximizing $\mathbf{w} = (\mathbf{w}, b)$ we get the formulation of logistic regression, or in equivalent form:

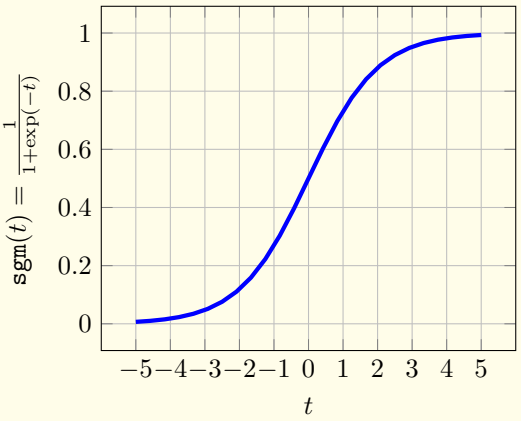
$$\min_{\mathbf{w}} \quad \frac{1}{2}\sum_{i=1}^{n}(1+\mathtt{y}_i)\log(1+\exp(-\mathbf{x}_i^\top \mathbf{w})) + (1-\mathtt{y}_i)\log(1+\exp(-\mathbf{x}_i^\top \mathbf{w})) + (1-\mathtt{y}_i)\mathbf{x}_i^\top \mathbf{w},$$

which is usually written in the more succinct form:

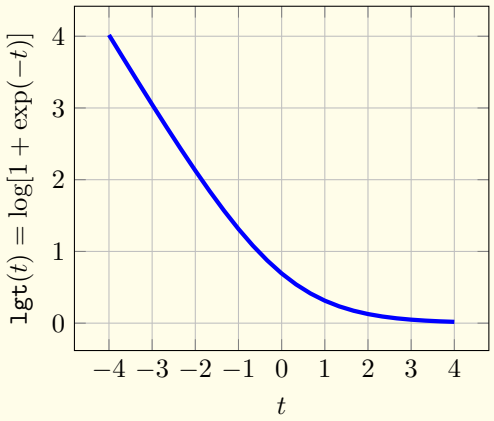
$$\min_{\mathbf{w}} \quad \sum_{i=1}^{n} \mathtt{lgt}(\mathtt{y}_i \hat{\mathtt{y}}_i), \quad \text{where} \quad \hat{\mathtt{y}}_i = \langle \mathbf{x}_i, \mathbf{w}\rangle, \quad \text{and } \mathtt{lgt}(t) := \log(1+\exp(-t)) \tag{3.4}$$

is the so-called logistic loss in the literature. (Clearly, the base of log is immaterial and we use the natural log.) In the above we have applied padding, see Remark 1.17, to ease notation.

sigmoid function



logistic loss



Cox, D. R. (1958). "The Regression Analysis of Binary Sequences". *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 215–242.

Alert 3.7: Binary labels: $\{\pm 1\}$ or $\{0, 1\}$?

In this note we choose to encode binary labels as $\{\pm 1\}$. In the literature the alternative choice $\{0, 1\}$ is also common. In essence there is really no difference if we choose one convention or the other: the eventual conclusions would be the same. However, some formulas do look a bit different on the surface! For example, the neat formula (3.4) becomes

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log[\exp\left((1 - \mathsf{y}_i) \langle \mathbf{x}_i, \mathbf{w} \rangle\right) + \exp(-\mathsf{y}_i \langle \mathbf{x}_i, \mathbf{w} \rangle)],$$

had we chosen the convention $\mathsf{y}_i \in \{0, 1\}$. Always check the convention before you subscribe to any formula!

Remark 3.8: Prediction with confidence

Once we solve $\mathbf{w}$ as in (3.4) and given a new test sample $\mathbf{x}$, we can compute $p(\mathbf{x}; \mathbf{w}) = \frac{1}{1+\exp(-\langle \mathbf{x}, \mathbf{w} \rangle)}$ and predict

$$\hat{\mathsf{y}}(\mathbf{x}) = \begin{cases} 1, & \text{if } p(\mathbf{x}; \mathbf{w}) \geq 1/2 \iff \langle \mathbf{x}, \mathbf{w} \rangle \geq 0 \\ -1, & \text{otherwise} \end{cases}.$$

In other words, for predicting the label we are back to the familiar rule $\hat{\mathsf{y}} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle)$. However, now we are also equipped with the probability confidence $p(\mathbf{x}; \mathbf{w})$.

It is clear that logistic regression is a linear classification algorithm, whose decision boundary is given by the hyperplane

$$H = \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle = 0\}.$$

Alert 3.9: Something is better than nothing?

It is tempting to prefer logistic regression over other classification algorithms since the former spits out not only label predictions but also probability confidences. However, one should keep in mind that in logistic regression, we make the assumption (see Equation (3.3))

$$\Pr(\mathsf{Y} = 1 | \mathsf{X} = \mathbf{x}) = \texttt{sgm}(\langle \mathbf{x}, \mathbf{w} \rangle) = \frac{1}{1 + \exp(-\langle \mathbf{x}, \mathbf{w} \rangle)},$$

which may or may not hold on your particular dataset. So the probability estimates we get from logistic regression can be totally off. Is it really better to have a probability estimate that is potentially very wrong than not to have anything at all? To exaggerate in another extreme, for any classification algorithm we can "make up" a 100% confidence on each of its predictions. Does this "completely fake" probability confidence bring any comfort? But, how is this any different from the numbers you get from logistic regression?

Alert 3.10: Do not do extra work

Logistic regression does more than classification, since it also tries to estimate the posterior probabilities. However, if prediction (of the label) is our sole goal, then we do not have to, and perhaps should not, estimate the posterior probabilities. Put it more precisely, all we need to know is whether or not $\eta(\mathbf{x}) = \Pr(\mathsf{Y} = 1 | \mathsf{X} = \mathbf{x})$ is larger than $1/2$. The precise value of $\eta(\mathbf{x})$ is not important; only its comparison with $1/2$ is. As we shall see, support vector machines, in contrast, only tries to estimate the decision boundary (i.e. the relative comparison between $\eta(\mathbf{x})$ and $1/2$), hence can be more efficient.

Remark 3.11: More than logistic regression

The main idea behind logistic regression is to equate the posterior probability $p(\mathbf{x};\mathbf{w})$ with some transformation F of the affine function $\langle \mathbf{x},\mathbf{w}\rangle$. Here the transformation F turns a real number into some value in $[0,1]$ (where the posterior probability belongs to). Obviously, we can choose F to be any cumulative distribution function (cdf) on the real line. Indeed, plug the formula

$$p(\mathbf{x};\mathbf{w}) = F(\langle \mathbf{x},\mathbf{w}\rangle),$$

into the conditional likelihood (3.2) gives us many variants of logistic regression. If we choose F to be the cdf of the logistic distribution (hence the name)

$$F(x;\mu,s) = \frac{1}{1+\exp\left(-\frac{x-\mu}{s}\right)},$$

where μ is the mean and s is some shape parameter (with variance $s^2\pi^2/3$), then we recover logistic regression (provided that $\mu = 0$ and $s = 1$).

If we choose F to be the cdf of the standard normal distribution, then we get the so-called probit regression.

Algorithm 3.12: Gradient descent for logistic regression

Unlike linear regression, logistic regression no longer admits a closed-form solution. Instead, we can apply gradient descent to iteratively converge to a solution. All we need is to apply the chain rule to compute the gradient of each summand of the objective function in (3.4):

$$\begin{aligned}
\nabla \mathtt{lgt}(\mathsf{y}_i \langle \mathbf{x}_i,\mathbf{w}\rangle) &= -\frac{\exp(-t)}{1+\exp(-t)}\Big|_{t=\mathsf{y}_i\langle \mathbf{x}_i,\mathbf{w}\rangle} \cdot \mathsf{y}_i\mathbf{x}_i = -\mathtt{sgm}(-\mathsf{y}_i\langle \mathbf{x}_i,\mathbf{w}\rangle)\cdot \mathsf{y}_i\mathbf{x}_i \\
&= -\mathsf{y}_i\mathbf{x}_i + \mathtt{sgm}(\mathsf{y}_i\langle \mathbf{x}_i,\mathbf{w}\rangle)\mathsf{y}_i\mathbf{x}_i \\
&= \begin{cases} (p(\mathbf{x}_i;\mathbf{w})-1)\mathbf{x}_i, & \text{if } \mathsf{y}_i = 1 \\ (p(\mathbf{x}_i;\mathbf{w})-0)\mathbf{x}_i, & \text{if } \mathsf{y}_i = -1 \end{cases} \\
&= \left(p(\mathbf{x}_i;\mathbf{w}) - \tfrac{\mathsf{y}_i+1}{2}\right)\mathbf{x}_i.
\end{aligned}$$

In the following algorithm, we need to choose a step size η. A safe choice is

$$\eta = \frac{4}{\|\mathbf{X}\|_{\mathrm{sp}}^2},$$

namely, the inverse of the largest singular value of the Hessian (see below). An alternative is to start with some small η and decrease it whenever we are not making progress (known as step size annealing).

Algorithm: Gradient descent for binary logistic regression.

Input: $X \in \mathbb{R}^{d\times n}$, $\mathbf{y} \in \{-1,1\}^n$ (training set), initialization $\mathbf{w} \in \mathbb{R}^{\mathsf{p}}$
Output: $\mathbf{w} \in \mathbb{R}^{\mathsf{p}}$

for $t = 1,2,\ldots,$ `maxiter` **do**
  sample a minibatch $I = \{i_1,\ldots,i_{\mathsf{m}}\} \subseteq \{1,\ldots,n\}$
  $\mathbf{g} \leftarrow \mathbf{0}$
  for $i \in I$ **do** `// use for-loop only in parallel implementation`
    $p_i \leftarrow \frac{1}{1+\exp(-\langle \mathbf{x}_i,\mathbf{w}\rangle)}$ `// in serial, replace with` $\mathbf{p} \leftarrow \frac{1}{1+\exp(-\mathbf{X}_{:,I}^\top \mathbf{w})}$
    $\mathbf{g} \leftarrow \mathbf{g} + (p_i - \frac{1+\mathsf{y}_i}{2})\mathbf{x}_i$ `// in serial, replace with` $\mathbf{g} \leftarrow \mathbf{X}_{:,I}(\mathbf{p} - \frac{1+\mathbf{y}_I}{2})$
  choose step size $\eta > 0$
  $\mathbf{w} \leftarrow \mathbf{w} - \eta\mathbf{g}$
  check stopping criterion `// e.g.` $\|\eta\mathbf{g}\| \le \mathtt{tol}$

For small problems ($n \le 10^4$ say), we can set $I = \{1,\ldots,n\}$, i.e., use the entire dataset in every iteration.

Algorithm 3.13: Newton iteration for logistic regression

We can also apply Newton's algorithm for solving logistic regression. In addition to computing the gradient, we now also need to compute the Hessian:

$$H_i = \nabla^2_{\mathbf{w}} \mathtt{lgt}(\mathtt{y}_i \langle \mathbf{x}_i, \mathbf{w} \rangle) = \mathbf{x}_i [\nabla_{\mathbf{w}} p(\mathbf{x}_i; \mathbf{w})]^\top = p(\mathbf{x}_i; \mathbf{w})[1 - p(\mathbf{x}_i; \mathbf{w})] \mathbf{x}_i \mathbf{x}_i^\top.$$

Algorithm: Newton iteration for binary logistic regression.

Input: $X \in \mathbb{R}^{d \times n}$, $\mathbf{y} \in \{-1, 1\}^n$ (training set), initialization $\mathbf{w} \in \mathbb{R}^{\mathsf{p}}$
Output: $\mathbf{w} \in \mathbb{R}^{\mathsf{p}}$

1 **for** $t = 1, 2, \ldots, \mathtt{maxiter}$ **do**
2 sample a minibatch $I = \{i_1, \ldots, i_{\mathsf{m}}\} \subseteq \{1, \ldots, n\}$
3 $\mathbf{g} \leftarrow \mathbf{0}$, $H \leftarrow \mathbf{0}$
4 **for** $i \in I$ **do** // use for-loop only in parallel implementation
5 $p_i \leftarrow \frac{1}{1+\exp(-\langle \mathbf{x}_i, \mathbf{w} \rangle)}$ // in serial, replace with $\mathbf{p} \leftarrow \frac{1}{1+\exp(-\mathbf{X}_{:,I}^\top \mathbf{w})}$
6 $\mathbf{g} \leftarrow \mathbf{g} + (p_i - \frac{1+\mathtt{y}_i}{2}) \mathbf{x}_i$ // in serial, replace with $\mathbf{g} \leftarrow \mathbf{X}_{:,I}(\mathbf{p} - \frac{1+\mathbf{y}_I}{2})$
7 $H \leftarrow H + p_i(1 - p_i) \mathbf{x}_i \mathbf{x}_i^\top$ // in serial, replace with $H \leftarrow \mathbf{X}_{:,I} \operatorname{diag}(\mathbf{p} \odot (1 - \mathbf{p})) \mathbf{X}_{:,I}^\top$
8 choose step size $\eta > 0$
9 $\mathbf{w} \leftarrow \mathbf{w} - \eta H^{-1} \mathbf{g}$ // solve $H^{-1}\mathbf{g}$ as linear system
10 check stopping criterion // e.g. $\|\mathbf{g}\| \le \mathtt{tol}$

Typically, we need to tune η at the initial phase but quickly we can just set $\eta \equiv 1$. Newton's algorithm is generally much faster than gradient descent. The downside, however, is that computing and storing the Hessian can be expensive. For example, Line 9 has per-step time complexity $O(\mathsf{m}d)$ and space complexity $O(d)$ (or $O(nd)$ if X is stored explicitly in memory) while Line 10 has per-step time complexity $O(\mathsf{m}d^2 + d^3)$ and space complexity $O(d^2)$ (or $O(nd + d^2)$ if X is stored explicitly in memory).

Alert 3.14: Overflow and underflow (Goldberg 1991)

Numerically computing $\exp(\mathbf{a})$ can be tricky when the vector $\mathbf{a}$ has very large or small entries. The usual trick is to shift the origin as follows. Let $t = \max_i a_i - \min_i a_i$ be the range of the elements in $\mathbf{a}$. Then, after shifting 0 to $t/2$:

$$\exp(\mathbf{a}) = \exp(\mathbf{a} - t/2) \exp(t/2).$$

Computing $\exp(\mathbf{a} - t/2)$ may be numerically better than computing $\exp(\mathbf{a})$ directly. The scaling factor $\exp(t/2)$ usually will cancel out in later computations so we do not need to compute it. (Even when we have to, it may be better to return $t/2$ than $\exp(t/2)$.)

Goldberg, D. (1991). "What every computer scientist should know about floating-point arithmetic". *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48.

Remark 3.15: Logistic regression as iterative re-weighted linear regression

Let us define the diagonal matrix $\hat{S} = \operatorname{diag}\big(\hat{\mathbf{p}} \odot (1 - \hat{\mathbf{p}})\big)$. If we set $\eta \equiv 1$, then we can interpret Newton's iteration in Line 10 as iterative re-weighted linear regression (IRLS):

$$\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - (\mathbf{X}\hat{S}\mathbf{X}^\top)^{-1}\mathbf{X}(\hat{\mathbf{p}} - \tfrac{1+\mathbf{y}}{2}) \\
&= (\mathbf{X}\hat{S}\mathbf{X}^\top)^{-1}[(\mathbf{X}\hat{S}\mathbf{X}^\top)\mathbf{w} - \mathbf{X}(\hat{\mathbf{p}} - \tfrac{1+\mathbf{y}}{2})] \\
&= (\mathbf{X}\hat{S}\mathbf{X}^\top)^{-1}\mathbf{X}\hat{S}\mathfrak{y}, \quad \mathfrak{y} := \mathbf{X}^\top\mathbf{w} - \hat{S}^{-1}(\hat{\mathbf{p}} - \tfrac{1+\mathbf{y}}{2}) \\
&= \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^n \hat{s}_i(\mathbf{w}^\top\mathbf{x}_i - \mathfrak{y}_i)^2, \quad \hat{s}_i := \hat{p}_i(1 - \hat{p}_i),
\end{aligned} \tag{3.5}$$

where the last equality can be seen by setting the derivative w.r.t. $\mathbf{w}$ to $\mathbf{0}$.

So, Newton's algorithm basically consists of two steps:

- given the current $\mathbf{w}$, compute the weights $\hat{s}_i$ and update the targets $\mathfrak{y}_i$. Importantly, if the current $\mathbf{w}$ yields very confident prediction $\hat{p}_i$ for the i-th training example (i.e., when $\hat{p}_i$ is close to 0 or 1), then the corresponding weight $\hat{s}_i$ is close to 0, i.e., we are down-weighting this training example whose label we are already fairly certain about. On the other hand, if $\hat{p}_i$ is close to $1/2$, meaning we are very unsure about the i-th training example, then the corresponding weight $\hat{s}_i$ will be close to the maximum value $1/4$, i.e. we pay more attention to it in the next iteration.
- solve the re-weighted least squares problem (3.5).

We have to iterate the above two steps because $\hat{\mathbf{p}}$ hence $\hat{\mathbf{s}}$ are both functions of $\mathbf{w}$ themselves. It would be too difficult to solve $\mathbf{w}$ in one step. This iterative way of solving complicated problems is very typical in machine learning.

Remark 3.16: Linear regression vs. logistic regression

In the following comparison, $\hat{S} = \operatorname{diag}\big(\hat{\mathbf{p}} \odot (1 - \hat{\mathbf{p}})\big)$. We note that as $\hat{y}_i$ deviates from y_i, the least squares loss varies from 0 to ∞. Similarly, as $\hat{p}_i$ deviates from y_i, the cross-entropy loss varies from 0 to ∞ as well.

• least-squares: $\sum_{i=1}^n (y_i - \hat{y}_i)^2$	• cross-entropy: $\sum_{i=1}^n -\frac{1+\mathsf{y}_i}{2} \log \hat{p}_i - \frac{1-\mathsf{y}_i}{2} \log(1-\hat{p}_i)$
• prediction: $\hat{y}_i = \langle \mathbf{x}_i, \mathbf{w} \rangle$	• prediction: $\hat{\mathsf{y}}_i = \operatorname{sign}(\langle \mathbf{x}_i, \mathbf{w} \rangle)$, $\hat{p}_i = \mathtt{sgm}(\langle \mathbf{x}_i, \mathbf{w} \rangle)$
• objective: $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$	• objective: $\mathsf{KL}(\frac{1+\mathbf{y}}{2} \| \hat{\mathbf{p}})$
• grad: $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{X}(\hat{\mathbf{y}} - \mathbf{y})$	• grad: $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{X}(\hat{\mathbf{p}} - \frac{1+\mathbf{y}}{2})$
• Newton: $\mathbf{w} \leftarrow \mathbf{w} - \eta (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{X}(\hat{\mathbf{y}} - \mathbf{y})$	• Newton: $\mathbf{w} \leftarrow \mathbf{w} - \eta (\mathbf{X}\hat{S}\mathbf{X}^\top)^{-1} \mathbf{X}(\hat{\mathbf{p}} - \frac{1+\mathbf{y}}{2})$

Exercise 3.17: Linearly separable

If the training data $\mathcal{D} = \{(\mathbf{x}_i, \mathsf{y}_i) : i = 1, \dots, n\}$ is linearly separable (see Definition 1.24), does logistic regression have a solution $\mathbf{w}$? What happens if we run gradient descent (Line 9) or Newton's iteration (Line 10)?

Exercise 3.18: Regularization

Derive the formulation and an algorithm (gradient or Newton) for ℓ_2-regularized logistic regression, where we add $\lambda \|\mathbf{w}\|_2^2$.

Remark 3.19: More than 2 classes

We can easily extend logistic regression to $\mathsf{c} > 2$ classes. As before, we make the assumption

$$\Pr(\mathsf{Y} = k | \mathsf{X} = \mathbf{x}) = f_k(\mathbf{W}\mathbf{x}), \quad k = 1, \dots, \mathsf{c},$$

where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_\mathsf{c}]^\top \in \mathbb{R}^{\mathsf{c} \times \mathsf{p}}$ and the vector-valued function $\mathbf{f} = [f_1, \dots, f_\mathsf{c}] : \mathbb{R}^\mathsf{c} \to \Delta_{\mathsf{c}-1}$ maps a vector of size $\mathsf{c} \times 1$ to a probability vector in the simplex $\Delta_{\mathsf{c}-1}$. Given an i.i.d. training dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, n\}$, where each $\mathbf{y}_i \in \{0, 1\}^\mathsf{c}$ is a one-hot vector, i.e. $\mathbf{1}^\top \mathbf{y}_i = 1$, then the (negated) conditional

log-likelihood is:

$$-\log \Pr(\mathsf{Y}_1 = \mathbf{y}_1, \dots, \mathsf{Y}_n = \mathbf{y}_n | \mathsf{X}_1 = \mathbf{x}_1, \dots, \mathsf{X}_n = \mathbf{x}_n) = -\log \prod_{i=1}^{n} \prod_{k=1}^{\mathsf{c}} [f_k(\mathbf{W}\mathbf{x}_i)]^{y_{ki}}$$
$$= \sum_{i=1}^{n} \sum_{k=1}^{\mathsf{c}} -y_{ki} \log f_k(\mathbf{W}\mathbf{x}_i).$$

To minimize the negated log-likelihood, we can apply gradient descent or Newton's iteration as before:

$$\nabla \ell_i(\mathbf{W}) = \sum_{k=1}^{\mathsf{c}} -y_{ki} \frac{1}{f_k(\mathbf{W}\mathbf{x}_i)} \left[\nabla f_k|_{\mathbf{W}\mathbf{x}_i}\right] \cdot \mathbf{x}_i^\top, \tag{3.6}$$

$$\forall \mathbf{G} \in \mathbb{R}^{\mathsf{c}\times\mathsf{p}}, \quad [\nabla^2 \ell_i(\mathbf{W})](\mathbf{G}) = \sum_{k=1}^{\mathsf{c}} -y_{ki} \frac{1}{f_k^2(\mathbf{W}\mathbf{x}_i)} \left[\nabla f_k \nabla f_k^\top - f_k \nabla^2 f_k\right]|_{\mathbf{W}\mathbf{x}_i} \mathbf{G}\mathbf{x}_i\mathbf{x}_i^\top, \tag{3.7}$$

where recall that $\nabla \ell_i(\mathbf{W}) \in \mathbb{R}^{\mathsf{c}\times\mathsf{p}}$ and $\nabla^2 \ell_i(\mathbf{W}) : \mathbb{R}^{\mathsf{c}\times\mathsf{p}} \to \mathbb{R}^{\mathsf{c}\times\mathsf{p}}$. Note that due to our one-hot encoding, the above summation has actually one term.

Definition 3.20: Multiclass logistic regression, a.k.a. Multinomial logit or softmax regression

The multinomial logit model corresponds to choosing the softmax function:

$$\mathbf{f}(\mathbf{W}\mathbf{x}) = \mathtt{softmax}(\mathbf{W}\mathbf{x}), \quad \text{where} \quad \mathtt{softmax} : \mathbb{R}^{\mathsf{c}} \to \Delta_{\mathsf{c}-1}, \ \mathfrak{y} \mapsto \frac{\exp(\mathfrak{y})}{\mathbf{1}^\top \exp(\mathfrak{y})}.$$

Let $\hat{\mathbf{p}}_i = \mathbf{f}(\mathbf{W}\mathbf{x}_i)$ and specialize (3.6) and (3.7) to the softmax function we obtain its gradient and Hessian:

$$\nabla \ell_i(\mathbf{W}) = (\hat{\mathbf{p}}_i - \mathbf{y}_i)\mathbf{x}_i^\top,$$
$$\forall \mathbf{G} \in \mathbb{R}^{\mathsf{c}\times\mathsf{p}}, \quad [\nabla^2 \ell_i(\mathbf{W})](\mathbf{G}) = \left(\operatorname{diag}(\hat{\mathbf{p}}_i) - \hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^\top\right)\mathbf{G}\mathbf{x}_i\mathbf{x}_i^\top.$$

In the multiclass setting, solving the Newton step could quickly become infeasible ($O(\mathsf{d}^3\mathsf{c}^3)$). As Böhning (1992) pointed out, we can instead use the upper bound:

$$0 \preceq \operatorname{diag}(\hat{\mathbf{p}}_i) - \hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^\top \preceq \frac{1}{2}(I_k - \tfrac{1}{k+1}\mathbf{1}\mathbf{1}^\top),$$

which would reduce the computation to inverting only the data matrix $\mathbf{X}\mathbf{X}^\top = \sum_i \mathbf{x}_i\mathbf{x}_i^\top$.

Böhning, D. (1992). "Multinomial logistic regression algorithm". *Annals of the Institute of Statistical Mathematics*, vol. 44, no. 1, pp. 197–200.

Remark 3.21: Mean and Covariance

We point out the following "miracle:" Let Y be a random vector taking values on standard bases $\{\mathbf{e}_k \in \{0,1\}^{\mathsf{c}} : k = 1, \dots, \mathsf{c}, \mathbf{1}^\top \mathbf{e}_k = 1\}$ and following the multinomial distribution:

$$\Pr(\mathsf{Y} = \mathbf{e}_k) = p_k, \quad k = 1, \dots, \mathsf{c}.$$

Then, straightforward calculation verifies:

$$\mathbb{E}(\mathsf{Y}) = \mathbf{p},$$
$$\operatorname{Cov}(\mathsf{Y}) = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top.$$

Remark 3.22: Removing translation invariance in `softmax`

In the above multiclass logistic regression formulation, we used a matrix $\mathbf{W}$ with c rows to represent c classes. Note however that the `softmax`function is translation invariant:

$$\forall \mathbf{w},\quad \texttt{softmax}((\mathbf{W}+\mathbf{1}\mathbf{w}^\top)\mathbf{x}) = \texttt{softmax}(\mathbf{W}\mathbf{x}).$$

Therefore, for identifiability purposes, we may assume w.l.o.g. $\mathbf{w}_\mathsf{c} = \mathbf{0}$ and we need only optimize the first $\mathsf{c}-1$ rows. If we denote $L(\mathbf{w}_1,\dots,\mathbf{w}_{\mathsf{c}-1},\mathbf{w}_\mathsf{c})$ as the original negated log-likelihood in Definition 3.20, then fixing $\mathbf{w}_\mathsf{c} = \mathbf{0}$ changes our objective to $L(\mathbf{w}_1,\dots,\mathbf{w}_{c-1},\mathbf{0})$. Clearly, the gradient and Hessian formula in Definition 3.20 still works after deleting the entries corresponding to $\mathbf{w}_\mathsf{c}$.

Setting $\mathsf{c}=2$ we recover binary logistic regression, with the alternative encoding $y\in\{0,1\}$ though.

Exercise 3.23: Alternative constraint to remove translation invariance

An alternative fix to the translation-invariance issue of `softmax` is to add the following constraint:

$$\mathbf{1}^\top\mathbf{W} = \mathbf{0}. \tag{3.8}$$

In this case our objective changes to $L(\mathbf{w}_1,\dots,\mathbf{w}_{c-1},-\sum_{k=1}^{\mathsf{c}-1}\mathbf{w}_k)$. How should we modify the gradient and Hessian?

Interestingly, after we add ℓ_2 regularization to the unconstrained multiclass logistic regression:

$$\min_{\mathbf{W}}\ L(\mathbf{w}_1,\dots,\mathbf{w}_\mathsf{c}) + \lambda\|\mathbf{W}\|_\mathsf{F}^2,$$

the solution automatically satisfies the constraint (3.8). Why? What if we added ℓ_1 regularization?

Definition 3.24: Generalized linear models (GLMs)

The similarity between linear regression and logistic regression is not coincidental: they both belong to generalized linear models (i.e. exponential family noise distributions), see Nelder and Wedderburn (1972).

Nelder, J. A. and R. W. M. Wedderburn (1972). "Generalized Linear Models". *Journal of the Royal Statistical Society. Series A (General)*, vol. 135, no. 3, pp. 370–384.