Introduction to Machine Learning
Spring 2020
Lecturer: Reinhard Heckel
Lecture notes week 6

1 Logistic regression

In this lecture, we start discussing classification problems. In a classification problem, our goal is, given a feature vector $\mathbf{x}$, to predict its label y, which can only take on a small set of discrete values. A special case is binary classification, where the label y only takes on two values, 0 and 1 for example. Most applications of machine learning are classification problems. A concrete classical example of a classification problem is spam filtering, were the vector $\mathbf{x}$ contains features of an email, and $y = 1$ stands for spam and $y = 0$ stands for not spam.

Formally, the classification problem is as follows: Given a training set $\{(\mathbf{x}_1, y_1), \ldots, \mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1, \ldots, K-1\}$, our goal is to find a classifier h such that given an arbitrary feature vector $\mathbf{x}$, the classifier h predicts the corresponding class $\{0, 1, \ldots, K-1\}$.

Models for classification problems can be distinguished as being generative or discriminative. A discriminative model describes the distribution $\mathrm{P}[y|\mathbf{x}]$ directly and thus directly discriminates between the target value y for any given feature vector $\mathbf{x}$. In contrast, a generative model learns $\mathrm{P}[\mathbf{x}|y]$ and $\mathrm{P}[y]$ directly, we will discuss an example of a generative model/classifier in the next lecture.

1.1 Logistic regression model

For simplicity, we start with a binary classification problem with $y \in \{0, 1\}$. We could approach the classification problem as a continuous-valued regression problem by ignoring the fact that y is discretely valued, but that usually performs very poorly. Instead, we can change the parametric form of our classifier, to ensure that it only outputs values in the interval $[0, 1]$. The output of the classifier can then be interpreted as the probability that the feature belongs to one class or the other. In the logistic regression model, the classifier takes the following form:

$$h_\theta(\mathbf{x}) = g(\langle \theta, \mathbf{x} \rangle),$$

where

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

is the logistic function, also called sigmoid function. Figure 1 contains a plot. The logistic function smoothly varies between 0 and 1. We can also choose other functions instead of the logistic function, but we will see later that the logistic function has some advantages. Let us interpret $h_\theta(\mathbf{x})$ as a probability:

$$\mathrm{P}[y = 1|\mathbf{x}] = h_\theta(\mathbf{x}) \quad \text{and} \quad \mathrm{P}[y = 0|\mathbf{x}] = 1 - h_\theta(\mathbf{x}).$$

We then classify an example as belonging to class 1 if $\mathrm{P}[y = 1|\mathbf{x}] > 1/2$ which is equivalent to $\langle \theta, \mathbf{x} \rangle > 0$. Logistic regression is called a discriminative classifier, since it directly estimates the parameters of the distribution $\mathrm{P}[y|\mathbf{x}]$.

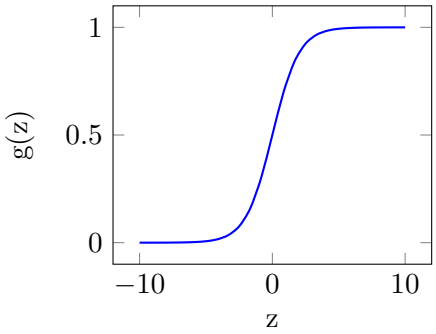

Figure 1: The sigmoid function $g(z) = (1 + e^{-z})^{-1}$.

1.2 Fitting a model

We next discuss methods to learn the feature vector θ based on training data. In the lecture on linear regression, we saw that we can learn parameters based on modeling assumptions on how the training data was generated. Let us assume that we are given a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, \mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. We furthermore assume that the feature vectors $\mathbf{x}_i$ are deterministic, but the labels are generated at random and independently across i and follow the distribution

$$\mathrm{P}\left[y_i = 1|\mathbf{x}_i\right] = h_\theta(\mathbf{x}_i) \quad \text{and} \quad \mathrm{P}\left[y_i = 0|\mathbf{x}_i\right] = 1 - h_\theta(\mathbf{x}_i).$$

As before, we estimate the parameter θ as the maximum likelihood estimate, i.e., the estimate that maximizes the probability that the data $\mathcal{D}$ is from the model h_θ. This amounts to finding the parameter θ that maximizes the likelihood function, defined as

$$\begin{aligned}
\mathcal{L}(\theta, \mathcal{D}) &= \mathrm{P}\left[y_1, \ldots, y_n|\theta\right] \\
&= \prod_{i=1}^{n} \mathrm{P}\left[y_i|\mathbf{x}_i\right] \\
&= \prod_{i=1}^{n} (h_\theta(\mathbf{x}_i))^{y_i} (1 - h_\theta(\mathbf{x}_i))^{1-y_i}.
\end{aligned}$$

The second inequality follows by our assumption that each label y_i is generated independently, and the third inequality follows from our assumption that the data is generated by the model h_θ.

As before, we maximize the log-likelihood instead (which we can do since log is strictly monotonic)

$$\log \mathcal{L}(\theta, \mathcal{D}) = \sum_{i=1}^{n} y_i \log(h(\mathbf{x}_i)) + (1 - y_i) \log(1 - h_\theta(\mathbf{x}_i)).$$

In conclusion, the logistic regression estimate of the parameter θ is given by:

$$\hat{\theta} = \arg\min_\theta \sum_{i=1}^{n} y_i \log(1/h_\theta(\mathbf{x}_i)) + (1 - y_i) \log 1/(1 - h_\theta(\mathbf{x}_i)).$$

Before discussing how to solve this optimization problem, we note that in regression, we started our discussion by fitting a model by minimizing the loss between the data predicted by the model

and the training data, and we choose the quadratic loss function $\text{loss}(y, z) = (y - z)^2$ to measure the loss. The theoretical justification was that learning with the quadratic loss corresponds to maximum likelihood estimation under Gaussian noise.

We could approach logistic regression in the same manner. Specifically, we can fit a model by solving

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \text{loss}(h_\theta(\mathbf{x}_i), y_i),$$

where loss is some loss function. Logistic regression does exactly that, and chooses as the loss function the log-loss or the negative cross entropy defined as

$$\text{loss}(y, z) = -y \log(z) - (1 - y) \log(1 - z) = y \log(1/z) + (1 - y) \log(1/(1 - z))$$

1.3 Computing the logistic regression estimate with gradient descent

The negative log-likelihood

$$-\log \mathcal{L}(\theta, \mathcal{D}) = \sum_{i=1}^{n} \text{loss}(g(\langle \theta, \mathbf{x}_i \rangle), y_i),$$

with

$$\text{loss}(g(\langle \theta, \mathbf{x}_i \rangle), y_i) = -y_i \log(g(\langle \theta, \mathbf{x}_i \rangle)) - (1 - y_i) \log(1 - g(\langle \theta, \mathbf{x}_i \rangle))$$

is convex since $\text{loss}(g(\langle \theta, \mathbf{x}_i \rangle), y_i)$ is convex, as we show later. Thus, we can approximate the logistic regression estimate $\hat{\theta}$ with gradient descent:

$$\begin{aligned} \theta^{k+1} &= \theta^k - \alpha \nabla(-\log \mathcal{L}(\theta, \mathcal{D})) \\ &= \theta^k - \alpha \sum_{i=1}^{n} \nabla \text{loss}(g(\langle \theta, \mathbf{x}_i \rangle), y_i). \end{aligned}$$

In practice we would actually often use stochastic gradient descent to minimize the function above, because of computational reasons. The stochastic gradient method is similar to gradient descent, and will be discussed in detail later in class.

We next compute the gradient of the loss above. Towards this goal, we first note that a convenient property of the logistic function is that its derivative can be written as:

$$g'(z) = \frac{-1}{(1 + e^{-z})^2} e^{-z}(-1) = \frac{1}{(1 + e^{-z})} \left(1 - \frac{1}{(1 + e^{-z})}\right) = g(z)(1 - g(z)),$$

where the first inequality is by the chain rule. With this relation, the gradients can be computed as

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \text{loss}(y, h_\theta(\mathbf{x})) &= -\left(y \frac{1}{g(\langle \theta, \mathbf{x} \rangle)} - (1 - y) y \frac{1}{1 - g(\langle \theta, \mathbf{x} \rangle)}\right) \frac{\partial}{\partial \theta_j} g(\langle \theta, \mathbf{x} \rangle) \\ &= -\left(\frac{y - g(\langle \theta, \mathbf{x} \rangle)}{g(\langle \theta, \mathbf{x} \rangle)(1 - g(\langle \theta, \mathbf{x} \rangle))}\right) g(\langle \theta, \mathbf{x} \rangle)(1 - g(\langle \theta, \mathbf{x} \rangle)) \frac{\partial}{\partial \theta_j} \langle \theta, \mathbf{x} \rangle \\ &= (g(\langle \theta, \mathbf{x} \rangle) - y) x_j, \end{aligned}$$

where the second inequality follows from the convenient relation $g'(z) = g(z)(1 - g(z))$, shown above.

Next, we show that the loss above is indeed convex in θ, as claimed. We can prove this as follows. Suppose that $d = 1$. Then the likelihood is a function $f\colon \mathbb{R} \to \mathbb{R}$. A differentiable function f is convex if and only if its second derivative obeys $f''(\mathbf{x}) \geq 0$ for all $\mathbf{x}$. For example x^2 is convex and it's second derivative is 1. This generalizes to multivariate functions as follows. Define the Hessian of a multivariate function $f\colon \mathbb{R}^d \to \mathbb{R}$ as the $d \times d$ matrix $\nabla^2 f(\mathbf{x})$ with entries

$$[\nabla^2 f(\mathbf{x})]_{ij} = \frac{\partial}{\partial \theta_i}\frac{\partial}{\partial \theta_j} f(\mathbf{x}).$$

A multivariate function is convex if and only if $\nabla^2 f(\mathbf{x})$ is positive semidefinite (recall that a matrix $\mathbf{A}$ is positive semidefinite if $\mathbf{z}^T\mathbf{A}\mathbf{z} \geq 0$ for all $\mathbf{z}$).

Thus, all we need to do is to show that the Hessian of the loss if positive semidefinite. Towards this goal, we compute

$$\begin{aligned}
\frac{\partial}{\partial \theta_i}\frac{\partial}{\partial \theta_j}\text{loss}(y, h_\theta(\mathbf{x})) &= \frac{\partial}{\partial \theta_i}(g(\langle \theta, \mathbf{x}\rangle) - y)x_j \\
&= \frac{\partial}{\partial \theta_i}(g(\langle \theta, \mathbf{x}\rangle))x_j \\
&= g(\langle \theta, \mathbf{x}\rangle)(1 - g(\langle \theta, \mathbf{x}\rangle))x_j x_i.
\end{aligned}$$

Thus, the Hessian can be written as

$$\nabla^2 \text{loss}(y, h_\theta(\mathbf{x})) = g(\langle \theta, \mathbf{x}\rangle)(1 - g(\langle \theta, \mathbf{x}\rangle))\mathbf{x}\mathbf{x}^T.$$

Since $g(\langle \theta, \mathbf{x}\rangle)(1 - g(\langle \theta, \mathbf{x}\rangle))$ is non-negative, and $\mathbf{x}\mathbf{x}^T$ is positive semidefinite, the Hessian above is positive semidefinite, and therefore the loss is convex in θ. Since the negative log-likelihood $-\log \mathcal{L}(\theta, \mathcal{D})$ is s a sum of convex functions, it is also convex. As a consequence, gradient descent will converge to a minimizer.

1.4 Multiclass logistic regression

Let us next generalize logistic regression to the case with $K > 2$ classes instead of only two classes. This is called multiclass logistic regression or softmax regression.

A key idea is to use what is called a one-hot-encoding $\mathbf{y} \in \{0, 1\}^K$ for representing the labels. The one-hot encoding $\mathbf{y}$ associated with a label $y = k$ is equal to 1 only at the position k:

$$[\mathbf{y}]_k = 1, \text{ if } y = k, \quad \text{and} \quad [\mathbf{y}]_j = 0 \text{ for all } j \neq k.$$

We associate a parameter vector $\theta_k \in \mathbb{R}^d$ with each class. For a given feature vector, we can associate a score with each class $z_k = \langle \theta_k, \mathbf{x}\rangle$. Computing this score for each k yields the vector $\mathbf{z} = [z_1, \ldots, z_K]$. The generalization of the logistic function is the softmax function $\sigma\colon \mathbb{R}^d \to [0, 1]^d$, and is defined as:

$$[\sigma(\mathbf{z})]_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, \quad \text{for } k = 1, \ldots, K.$$

Note that the components of the sofmax function $\sigma(\mathbf{z})$ applied to any vector $\mathbf{z}$ sum to one and lie in the range $[0, 1]$. Thus, they can be interpreted as a probability distribution. Given a feature vector $\mathbf{x}$, the model then predict its label as the index of the largest component of $\sigma(\mathbf{z})$.

We can estimate the parameters θ based on training data in a similar manner as before. Specifically, by summarizing the parameter vectors with the matrix $\Theta = [\theta_1, \ldots, \theta_K]$, we estimate the parameters by maximizing the likelihood or equivalently, minimizing the log-likeliehood:

$$\begin{aligned}
\hat{\Theta} &= \arg\max_{\Theta} \mathcal{L}(\Theta, \mathcal{D}) \\
&= \arg\max_{\Theta} \prod_{i=1}^{n} \mathrm{P}\left[y_i|\mathbf{x}_i\right] \\
&= \arg\max_{\Theta} \prod_{i=1}^{n} \prod_{k=1}^{k} \mathrm{P}\left[y_i = k|\mathbf{x}_i\right]^{\mathbb{1}\{y_i=k\}} \\
&= \arg\min_{\Theta} -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{y_i = k\} \log \mathrm{P}\left[y_i = k|\mathbf{x}_i\right] \\
&= \arg\min_{\Theta} -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{y_i = k\} \log \left(\frac{e^{\langle \theta_k, \mathbf{x}_i \rangle}}{\sum_{j=1}^{K} e^{\langle \theta_j, \mathbf{x}_i \rangle}} \right).
\end{aligned}$$

As for the binary regression case, this is a convex optimization problem and can be solved with gradient descent.

2 Generative models and the naive Bayes algorithm

Generative models assume that the examples $(\mathbf{x}, y)$ are drawn from a joint distribution, modeled by a joint probability distribution. Thus, both $\mathbf{x}$ and y are treated as random variables. For simplicity let's again consider binary classification, and assume that the the feature vector $\mathbf{x}$ is binary as well ($\mathbf{x} \in \{0, 1\}^d$). We only make the latter simplifying assumption so we can work with probabilities directly.

Suppose the condition probability $\mathrm{P}\left[y|\mathbf{x}\right]$ were known. Then we could classify an arbitrary feature vector $\mathbf{x}$ with the label that maximizes the joint probability:

$$\hat{y} = \arg\max_{y \in \{0,1\}} \mathrm{P}\left[y|\mathbf{x}\right].$$

We do not know the distribution $\mathrm{P}\left[y|\mathbf{x}\right]$ but we can estimate it based on training data. By Bayes rule:

$$\mathrm{P}\left[y|\mathbf{x}\right] = \frac{\mathrm{P}\left[\mathbf{x}|y\right] \mathrm{P}\left[y\right]}{\mathrm{P}\left[\mathbf{x}\right]} = \frac{\mathrm{P}\left[\mathbf{x}|y\right] \mathrm{P}\left[y\right]}{\mathrm{P}\left[\mathbf{x}|0\right] \mathrm{P}\left[0\right] + \mathrm{P}\left[\mathbf{x}|1\right] \mathrm{P}\left[1\right]}.$$

Thus one possibility to learn $\mathrm{P}\left[y|\mathbf{x}\right]$ is to use the training data for estimating $\mathrm{P}\left[\mathbf{x}|y\right]$ and $\mathrm{P}\left[y\right]$. From those estimates, we can then compute $\mathrm{P}\left[y|\mathbf{x}\right]$ for a new given feature vector $\mathbf{x}$.

It is easy to estimate $\mathrm{P}\left[y\right]$ accurately from few samples. To see this, note that y is a binary random variable with mean $\theta = \mathrm{P}\left[1\right]$. Given the example $y_1, \ldots, y_n$, an estimate of the mean is

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^{n} y_i.$$

Hoeffding's inequality states that, for all $\beta > 0$, we have that (using $\mathbb{E}[y_i] = \theta$):

$$\mathrm{P}\left[|\hat{\theta} - \theta| \geq \beta\right] \leq 2e^{-2\beta^2 n}.$$

With a change of variables (i.e., choosing δ such that $2e^{-2\beta^2 n} = \delta$, this is equal to

$$\mathrm{P}\left[|\hat{\theta} - \theta| \geq \sqrt{\frac{\log(2/\delta)}{2n}}\right] \leq \delta.$$

Setting $\delta = 0.05$ and $n = 100$, we see that with probability at least 95%, $|\hat{\theta} - \theta| \leq 0.13$; thus with on the order of 100 examples, we can obtain an fairly accurate estimate of the distribution $\mathrm{P}[y]$.

Now consider estimating the distribution $\mathrm{P}[y|\mathbf{x}]$, which has about 2^d many parameters. Using the same line of arguments as above, accurately estimating $\mathrm{P}[y|\mathbf{x}]$ requires prohibitory many examples (i.e., a number on the order of $O(2^d)$) without further assumptions on the distribution, unless d is very small.

2.1 Naive Bayes algorithm

We have seen that without making further assumptions on the distribution, estimating the distribution $\mathrm{P}[y|\mathbf{x}]$ is infeasible. The naive Bayes classifier makes a conditional independence assumption that reduces the number of parameters from $O(2^d)$ to $O(d)$.

To state this assumption, recall the definition of conditional independence: Given three random variables X, Y, and Z, we say that X is conditionally independent of Y given Z, if and only if, given the value of Z, the probability distribution of X is the same for all values of Y. For discrete random variables, this is equivalent to

$$\mathrm{P}[X = x|Y = y, Z = z] = \mathrm{P}[X = x|Z = z] \text{ for all triplets } (x, y, z).$$

We now assume that the entries of $\mathbf{x} = [x_1, \ldots, x_d]$ are conditionally independent, i.e., x_i is conditionally independent of all other entries $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d$. This assumptions dramatically simplifies the conditional probability $\mathrm{P}[\mathbf{x}|y]$:

$$\mathrm{P}[\mathbf{x}|y] = \mathrm{P}[x_1, \ldots, x_d|y] = \prod_{i=1}^{d} \mathrm{P}[x_i|y].$$

If both x_i and y are binary, then this distribution is fully characterized by $2d$ parameters, significantly less than $O(2^d)$ parameters required if we do not make any assumption on the distribution.

With this assumption we have by Bayes rule that

$$\mathrm{P}[y|x_1, \ldots, x_d] = \frac{\mathrm{P}[y] \prod_i \mathrm{P}[x_i|y]}{\mathrm{P}[x_1, \ldots, x_d]}.$$

This equation is central to the Naive Bayes classifier, as it enables computation of the conditional probability of a class label given a feature vector. With that, the Bayes classification rule becomes

$$\hat{y} = \arg \max_{y \in \{0, \ldots, K-1\}} \mathrm{P}[y] \prod_i \mathrm{P}[x_i|y],$$

where we used that the denominator $\mathrm{P}[x_1, \ldots, x_d]$ does not depend on y.

The distributions above are unknown. The Naive Bayes estimators computes the Bayes classification rule based on estimated probabilities.

Naive Bayes for discrete-valued features: For discrete valued features $x_i \in \{0, \ldots, J-1\}$, the naive Bayes learning algorithm estimates the parameters

$$\theta_{ijk} = \mathrm{P}\left[x_i = j | y = k\right], \quad \eta_k = \mathrm{P}\left[y = k\right],$$

for all possible triplets. These parameters can be estimated based on maximum likelihood estimation, or using Bayesian MAP estimators, by assuming a prior distribution over the parameters θ and η. Maximum likelihood estimation simply estimates the parameters above as

$$\theta_{ijk} = \frac{\text{number of examples with } x_i = j \text{ and } y = k}{\text{number of examples with } y = k}.$$

Naive Bayes for continuous-valued features: For continuous features, it is common to make a parametric assumptions on the distribution $\mathrm{P}\left[x_i | y\right]$, for example that it is Gaussian, and then estimate the parameters of the distribution. This is called Gaussian naive Bayes.

2.2 Logistic regression vs. Gaussian naive Bayes

None of the methods is uniformly better. However, as shown in a paper by Ng and Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes", as the number of training examples goes to infinity, logistic regression performs better than the generative model naive Bayes. However, naive Bayes converges faster than logistic regression to its asymptotic error, thus one can expect that with lots of examples, logistic regression performs better. This is also experimentally observed by Ng and Jordan.

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.