Generalized Linear Models

The only predictive model we have considered so far is linear regression. Let us recall: at the input, we have features with continuous values x_i, which we weight with coefficients θ_i and add a bias term (intercept) θ_0, thereby obtaining a prediction of the target value $\hat{y}$. The model is obtained by estimating parameter values that minimize the sum of squared errors between the actual and predicted values of the target variable in the training set. Linear regression is one of the simplest regression models; an even simpler approach would be to predict using the mean value of the dependent variable in the training set, but such an approach does not take into account the values of the input features, and therefore can hardly be considered a true predictive model.

In this chapter, the vector and matrix notation of the above model will be useful, where we represent the data in a feature matrix X, combine the parameters into a vector $\boldsymbol{\theta}$, and express the predictions as $\hat{\mathbf{y}} = X\boldsymbol{\theta}$, which allows for a more compact representation and more efficient implementation of learning algorithms. We assume that the first column in the data matrix is a vector of ones, so that we avoid special treatment of the intercept, and that the indices for θ run from 0 to d, where d is the number of independent variables.

The question arises whether there exist other similarly simple models, perhaps for somewhat different predictive tasks, that is, models in which features are again combined into a weighted sum, and then the resulting value is transformed by an appropriate (nonlinear) link function, so that we can also model discrete or otherwise constrained target variables. In what follows, we begin with an example of such a model, which we will use for classification, and then ask whether such extensions are sufficiently general for a broader class of models, what assumptions they actually make, where their objective functions come from, and whether this represents an interesting class of models with shared properties.

Example

Let us begin with a (fictional) example. Table 4 contains swimmers at Bled whom we asked how many hours per week they exercise and how many hours they slept the previous night. We also recorded whether, on the day of the interview, they managed to swim to the island. The island is more than half a kilometer away from the nearest swimming area in one direction, so swimming to the island and back is quite a challenge and would not be suitable for weaker swimmers. The goal is to develop an application that would advise swimmers, based on their physical fitness and level of rest, whether they should attempt such an endeavor. The application, of course, requires a predictive model, which we can build from our data.

Since the data are two-dimensional, it is best to plot them in a scatter plot. We also mark the class. At first glance, the plot suggests that it might be possible to separate good swimmers from those who mostly just bathe with a line, i.e., a decision boundary. This boundary is linear, so we can write it as $X\theta = 0$. The plot also includes three new visitors: Sara, Martin, and Leon. For which of them will our application, or model, recommend that they can swim to the island and back?

Sara is on the swimmers' side. She is far from the decision boundary that separates the two classes. She can certainly swim to the island and back. Martin is far on the other side; he should definitely not move away from the shore. Leon is, with respect to the decision boundary, on the swimmers' side, but only barely. Advising him to try swimming to the island would be quite wrong. Our problem is indeed a classification one—we want to predict one of two possible classes—but it would be better to do this cautiously, using probabilities. Since Sara is very far from the decision boundary, she is certainly a good candidate for going to the island; Martin definitely is not, while Leon is somewhere in between, his probability of belonging to the "swimmer" class is around 50%.

This suggests that the distance from the decision boundary should be mapped to probabilities. The linear combination $z = X\theta$ is in fact proportional to the distance from the line determined by the parameters θ. For the transformation, we can use a function whose range is between 0 and 1. An example of such a function is the sigmoid $\sigma(z)$, and our probability is then

$$P(y = 1 \mid X) = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

The decision boundary in Figure has parameters $\theta_0 = -15.6$, $\theta_1 = 0.8$, and $\theta_2 = 1.6$, so we can write the decision equation for the

Figure 4: An example of classification data with a meta attribute, independent variables, and the class (Island).

Name	Exercise	Sleep	Island
Alenka	7	8	1
Ana	2	8	0
Andrej	5	5	0
Blaž	5	9	1
Boštjan	7	5	0
Goran	12	6	1
Gregor	10	9	1
Helena	4	9	1
Irena	9	3	0
Janez	5	6	0
Jure	8	4	0
Katarina	4	3	0
Klara	3	9	1
Luka	5	4	0
Maja	9	6	0
Marko	4	7	0
Matej	4	8	1
Miha	6	4	0
Mojca	11	5	1
Nika	2	5	0
Nina	8	6	1
Petra	3	6	0
Polona	9	8	1
Rok	10	6	1
Sara	6	7	1
Sašo	10	7	1
Sebastjan	12	5	1
Tatjana	12	9	1
Tjaša	11	3	0
Tomaz	12	5	1

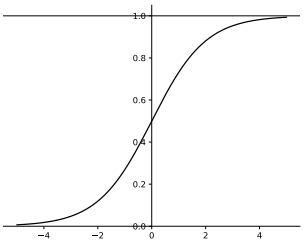

Figure 5: Sigmoid function.

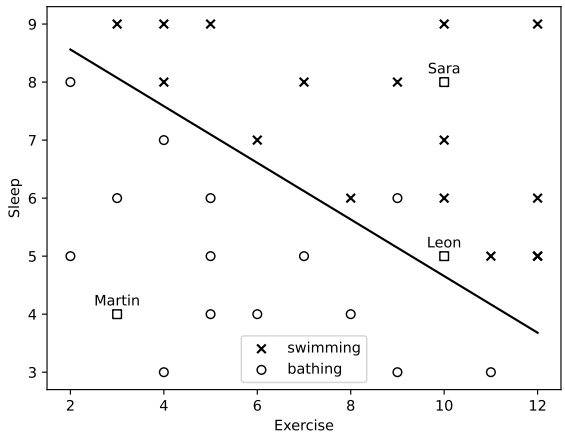

Figure 6: Training data, a possible decision boundary between the classes, and new (named) examples that we still need to classify.

distance from this boundary as

$$z = -15.6 + 0.8 \cdot \text{exercise} + 1.6 \cdot \text{sleep}.$$

For the new examples, we obtain: Sara has $z = 5.5$ and $P(\text{island} = 1) \approx 1.0$, so she is a very suitable candidate for swimming to the island; Martin has $z = -6.7$ and $P(\text{island} = 1) \approx 0.0$, so we advise against it; Leon has $z = 0.6$ and $P(\text{island} = 1) \approx 0.6$, which means he is close to the decision boundary and the decision is quite uncertain.

The model that we introduced rather quickly, and perhaps somewhat superficially, in our example is called logistic regression. It is important to note that, like linear regression, this model also uses a linear combination of independent variables, but this time the result is transformed using a sigmoid function, precisely so that we can output probabilities. However, several questions arise: how do we actually learn the "correct" parameters of our model, i.e., the vector θ? What objective function do we optimize for this purpose? From what assumptions is it derived? Besides linear and logistic regression, are there other models of this type? And finally, can we also use automatic differentiation and gradient descent to learn such a model?

It is time for a bit of theory.

Exponential Family of Distributions

Where do models such as linear and logistic regression come from? What assumptions do we actually make about the data? We take a similar approach as we already know from linear regression: instead of inventing an objective function (e.g., the sum of squared errors on the training set), we start from a probabilistic model, that is, a model

that generates the data in the training set, and from this assumption we derive the objective function.

A class of distributions that turns out to be particularly useful for our purposes is the *exponential family*. A distribution belongs to this family if its probability function (or density) for the variable y can be written in the form

$$p(y \mid \theta) = h(y) \exp\left(\eta(\theta)\, T(y) - A(\theta)\right),$$

where θ denotes the parameter of the distribution. Since in machine learning we are interested in the log-likelihood, it makes sense to take the logarithm of this expression:

$$\log p(y \mid \theta) = \eta(\theta) T(y) - A(\theta) + \log h(y).$$

The function $T(y)$ is called the *sufficient statistic* and in the simplest cases is equal to y. The function $\eta(\theta)$ is the so-called *natural parameter*, which represents a transformation of the original parameter θ. The function $A(\theta)$ ensures normalization of the distribution (this factor ensures that probabilities sum or integrate to 1), while $h(y)$ is the part independent of the parameter. We observe that the log-likelihood is linear in $T(y)$, which enables simple optimization and a connection with linear models.

In the exponential family of distributions, we find most of the distributions of interest in machine learning that are related to regression (predicting a continuous target), classification (predicting a discrete variable), and modeling counts. These distributions include, for example, the normal, Bernoulli, and Poisson distributions. Further details, of course, follow.

Natural parameter and link function

In supervised learning, we aim to model the conditional expected value of the target variable, that is, its mean value given input features x (in the case of classification, this corresponds to the probability of belonging to a particular class):

$$\mu(x) = \mathbb{E}[y \mid x].$$

In generalized linear models, we make the key assumption that the natural parameter is related to the features through a linear predictor

$$\eta = X\theta.$$

This is not a consequence of the exponential family, but rather a modeling assumption that extends the idea of linear regression to a broader class of distributions. For distributions from the exponential

family, however, it holds that that the expected value is determined by the natural parameter, $\mu = A'(\eta)$. In typical cases, we can invert this relationship and write

$$\eta = g(\mu).$$

This gives us the model

$$g(\mu(x)) = X\theta,$$

where the function g is called the link function.

If we choose the function g such that it coincides with the natural relationship between μ and η, we speak of the *canonical link function*. This choice leads to particularly simple expressions for the likelihood and its derivatives, and often to convex optimization problems.

Since the density must be normalized, we have

$$\int h(y) \exp\left(\eta T(y) - A(\eta)\right) dy = 1.$$

Differentiating with respect to η gives

$$\mathbb{E}[T(y)] - A'(\eta) = 0,$$

therefore

$$\mathbb{E}[T(y)] = A'(\eta).$$

Model learning

Here we only recall that, in order to determine the model parameters, we will need a criterion function, and for that we need the likelihood or its logarithm. In fact, we already have everything prepared for this. Assume that the training examples are independent of each other, and once we choose the target distribution of the classes, we can write the likelihood for the training data as

$$p(\mathbf{y} \mid X, \theta) = \prod_{i=1}^{n} p(y_i \mid x_i, \theta).$$

To learn the parameters, we maximize the log-likelihood

$$\ell(\theta) = \sum_{i=1}^{n} \log p(y_i \mid x_i, \theta),$$

which is equivalent to minimizing the negative log-likelihood, which can be interpreted as a criterion function.

This connects directly to standard machine learning practice: different choices of distributions lead to different criterion functions, while the optimization proceeds in the same way, e.g. using gradient descent.

Linear regression

Let us start with the simplest example, which we have already encountered, but now view it from a new perspective. Suppose that the target variable is continuous and follows a normal distribution

$$y \mid x \sim \mathcal{N}(\mu(x), \sigma^2).$$

The density can be written as

$$p(y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right).$$

If we rearrange the expression, we obtain

$$p(y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{\mu}{\sigma^2}y - \frac{\mu^2}{2\sigma^2} - \frac{y^2}{2\sigma^2}\right),$$

which is of the exponential family form

$$p(y \mid \theta) = h(y)\, \exp\big(\eta(\theta)\, T(y) - A(\theta)\big),$$

where we identify:

$$T(y) = y, \qquad \eta(\theta) = \frac{\mu}{\sigma^2}, \qquad A(\theta) = \frac{\mu^2}{2\sigma^2}, \qquad \log h(y) = -\frac{y^2}{2\sigma^2} - \frac{1}{2}\log(2\pi\sigma^2).$$

If we assume that the variance σ^2 is constant, then the natural parameter η is proportional to μ, so (with a slight abuse of notation) we can treat it simply as μ, which is equal to a linear combination of the features. In this case, the canonical link function is the identity,

$$g(\mu) = \mu = X\theta.$$

The log-likelihood for a single example is

$$\log p(y \mid x, \theta) = -\frac{1}{2\sigma^2}(y - X\theta)^2 + C,$$

where C does not depend on θ. Maximizing the likelihood is therefore equivalent to minimizing the sum of squared errors:

$$L(\theta) = \sum_{i=1}^{n} (y_i - x_i^T\theta)^2.$$

As we already know, the criterion function of linear regression directly follows from the assumption of normally distributed noise.

Logistic regression

In the case of binary classification, we assume that the target variable follows a Bernoulli distribution:

$$y \mid x \sim \text{Bernoulli}(p(x)),$$

where x is the vector of observed features for a given example, and $y \in \{0, 1\}$ is the class. The function $p(x)$ represents the probability that the class is equal to 1 given the features x, that is,

$$P(y = 1 \mid x) = p(x), \qquad P(y = 0 \mid x) = 1 - p(x).$$

The expected value of the target variable y given features x is therefore

$$\mu(x) = \mathbb{E}[y \mid x] = p(x).$$

The probability function of the target variable y given features x can be written in a single equation as

$$p(y \mid x) = p^y (1-p)^{1-y},$$

and its logarithm as

$$\log p(y \mid x) = y \log p + (1-y) \log(1-p).$$

This probability will be used to express the log-likelihood or the negative log-likelihood.

We rearrange this expression:

$$\log p(y \mid x) = y \log \frac{p}{1-p} + \log(1-p).$$

Under the assumption of independent training examples, this form is already suitable for constructing the likelihood, and then the criterion function. But that follows shortly. First, we must address how to relate p to $X\theta$.

Now we can compare the expression with the general form of the exponential family

$$\log p(y \mid \theta) = \eta(\theta)\, T(y) - A(\theta) + \log h(y),$$

and identify the components:

$$T(y) = y, \qquad \eta = \log \frac{p}{1-p}, \qquad A(\eta) = -\log(1-p), \qquad h(y) = 1.$$

The variable η, or the function $\eta(p)$ (the natural parameter), transforms the probability p into the logarithm of the odds ratio, i.e. $\log \frac{p}{1-p}$ (eng. *log-odds*). Since $\mu = p$, we obtain the link function, called the logit:

$$g(\mu) = \log \frac{p(x)}{1-p(x)} = X\theta,$$

which leads to the well-known sigmoid form with which we, somewhat intuitively, began this chapter:

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

Poisson regression

To model counts (e.g. the number of events in a given time interval), we assume a Poisson distribution:

$$y \mid x \sim \text{Poisson}(\lambda(x)).$$

This means that $y \in \{0, 1, 2, \dots\}$ and

$$p(y \mid x) = \frac{\lambda^y e^{-\lambda}}{y!}.$$

The expected value of this distribution is

$$\mu(x) = \mathbb{E}[y \mid x] = \lambda(x).$$

To see the connection with the exponential family, we take the logarithm:

$$\log p(y \mid x) = y \log \lambda - \lambda - \log(y!).$$

This expression is already almost in the form

$$\log p(y \mid x) = \eta\, y - A(\eta) + \log h(y),$$

from which we identify

$$\eta = \log \lambda.$$

Since $\mu = \lambda$, we obtain the link function

$$g(\mu) = \log \mu,$$

which is called the log link. The linear model can thus be written as

$$\log \lambda(x) = X\theta,$$

from which it follows that

$$\lambda(x) = e^{X\theta}.$$

The log-likelihood for a single example is

$$\log p(y \mid x, \theta) = yX\theta - e^{X\theta} - \log(y!),$$

and the negative log-likelihood leads to the criterion function

$$L(\theta) = \sum_{i=1}^{n} \left(e^{x_i^T \theta} - y_i x_i^T \theta \right),$$

where the term $\log(y!)$ can be omitted since it does not depend on the parameters.

Some coding

The above was a lot of theory and few examples. For a short break, let us check whether logistic regression really gives a solution similar to the one in our initial example. As with linear regression from the previous chapter, we begin with a class that implements the criterion function over the input data.

```
class LogReg:
    def __init__(self, n_inputs, reg=None, reg_strength=0.0):
        self.weights =
```

```
        [Value(random.uniform(-1, 1), label=f"w{i}")
         for i in range(n_inputs)]
        self.b = Value(0.0, label="b")
        self.reg = reg
        self.reg_strength = reg_strength

    def linear(self, x):
        return sum(w * xi for w, xi in zip(self.weights, x)) + self.b

    def __call__(self, x):
        return self.linear(x).sigmoid()

    def parameters(self):
        return self.weights + [self.b]

    def loss(self, xs, ys):
        eps = 1e-8
        losses = []
        for x, y in zip(xs, ys):
            yhat = self(x)
            y_val = Value(float(y))
            term = -(y_val * (yhat + eps).log() + \
              (1 - y_val) * (1 - yhat + eps).log())
            losses.append(term)
        data_loss = sum(losses) / Value(len(xs))

        if self.reg == "l2" and self.reg_strength > 0:
            l2_penalty = self.reg_strength * sum(w * w for w in self.weights)
            return data_loss + l2_penalty
        return data_loss

    def __repr__(self):
        weights_str = ", ".join(f"w{i}={w.data:.3f}" \
          for i, w in enumerate(self.weights))
        return f"LogReg({weights_str}, b={self.b.data:.3f})"
```

Upon initialization, `LogReg` creates a vector of (randomly initialized) weights and a bias term, optionally supporting L2 regularization controlled by `reg` and `reg_strength`. The method `linear` computes the affine combination of inputs and parameters, while the `__call__` method applies the sigmoid function to obtain a probabilistic prediction. The `parameters` method exposes all learnable quantities for optimization, which are used in the `train` function of autograd library (we defined this in our previous writings, and will not change it here). The key methods is the `loss` function, which constructs the computational graphs for the average binary cross-entropy over a traing set. A small numerical constant is added for numerical stability. Optionally, we add L2 penalty to the loss.

We invoke training with a call similar to one we have used in our

previous chapters for linear regression.

```
df_train = pd.read_excel("bled.xlsx")
feature_names = ["Exercise", "Sleep"]
xs = df_train[feature_names].values.tolist()
ys = df_train["Island"].astype(int).tolist()

model = LogReg(n_inputs=len(feature_names),
               reg="l2", reg_strength=0.01)
model = train(model, xs, ys, learning_rate=0.1,
              n_epochs=10000, batch_size=None)
```

The optimization converges relatively fast, and the weights and the end result is actually very similar to the one from Fig. . We could speed-up the process through data standardization, and, of course, by using faster optimization routines.

The changes to implement Poisson regression would be rather minimal. Instead of mapping the linear predictor through a sigmoid, we would use the exponential function to ensure that the predicted rate parameter remains positive, and the loss would correspond to the negative log-likelihood of the Poisson distribution. Concretely, the following parts of the implementation would change:

```
class PoissonReg(LogReg):
    def __call__(self, x):
        return self.linear(x).exp()

    def loss(self, xs, ys):
        losses = []
        for x, y in zip(xs, ys):
            yhat = self(x)
            y_val = Value(float(y))
            term = yhat - y_val * yhat.log()
            losses.append(term)
        return sum(losses) / Value(len(xs))
```

Multinomial logistic regression

The logistic regression we have considered so far is intended for binary classification. However, we often encounter tasks where there are more than two possible classes. For example, we may want to predict which mode of transport an individual will choose, or which category a document belongs to. Such problems can be addressed by an extension of logistic regression called *multinomial logistic regression*. This model is a special case of generalized linear models, where the target variable follows a categorical distribution.

Let the target variable be $y \in \{1, 2, \ldots, m\}$, where m is the number of classes. For each class j, we define a linear predictor

$$z_j = x^T \theta_j.$$

Since we want to obtain probabilities that are non-negative and sum to 1, we use the function

$$P(y = j \mid x) = \frac{e^{z_j}}{\sum_{l=1}^{m} e^{z_l}},$$

which is called the *softmax*. This function maps a vector of real values $(z_1, \ldots, z_m)$ to a probability vector.

The model is not uniquely determined, since we can add the same constant to all z_j without changing the probabilities. Therefore, we usually choose one class as the reference class and set its linear predictor to zero:

$$z_m = 0.$$

The model thus has $(m - 1)$ parameter vectors. Multinomial logistic regression is a generalized linear model where the target variable follows a categorical distribution, the linear predictor is $z_j = x^T \theta_j$, and the link function is the generalized logit, whose inverse is the softmax function.

The model can also be written as

$$y \mid x \sim \text{Categorical}(p_1(x), \ldots, p_m(x)),$$

where

$$p_j(x) = \frac{e^{x^T \theta_j}}{\sum_{l=1}^{m} e^{x^T \theta_l}}.$$

If we have only two classes ($m = 2$), the softmax simplifies to the sigmoid function, and we recover standard logistic regression.

The model parameters are estimated by maximizing the log-likelihood

$$\ell(\theta) = \sum_{i=1}^{n} \log P(y_i \mid x_i),$$

which leads to the criterion function known as multiclass cross-entropy. As in logistic regression, there is no closed-form solution, so the parameters are estimated numerically, e.g. using gradient descent.

Ordinal logistic regression

In some problems, the target variable is not only discrete, but its values also have a natural ordering. Such variables are called *ordinal*. Examples include survey responses (e.g. “disagree”, “neutral”,

"agree") or ratings (e.g. from 1 to 5). Such data could be handled using multinomial logistic regression, but that approach does not take into account the ordering between classes. Ordinal logistic regression explicitly models this ordering, and is therefore often more efficient and interpretable.

The basic idea of the model is that there exists a latent (unobserved) continuous variable z, which depends linearly on the features:

$$z = x^T\theta.$$

The observed ordinal variable y is determined by which interval the latent variable z falls into. These intervals are defined by thresholds (cutpoints) $t_1, t_2, \ldots, t_{m-1}$:

$$y = j \quad \text{if} \quad t_{j-1} < z \leq t_j,$$

where by convention we take $t_0 = -\infty$ and $t_m = \infty$.

To obtain probabilities, we assume that the latent variable is subject to logistic noise. This leads to a model where the probability that y is less than or equal to a given class is

$$P(y \leq j \mid x) = \frac{1}{1 + e^{-(t_j - x^T\theta)}}.$$

This expression represents a cumulative probability, which is why the model is often called the *cumulative logit model*. The probabilities of individual classes are obtained as differences between consecutive cumulative probabilities:

$$P(y = j \mid x) = P(y \leq j \mid x) - P(y \leq j - 1 \mid x).$$

Ordinal logistic regression is a generalized linear model where the target variable follows a categorical distribution with ordered classes, the linear predictor is $x^T\theta$, and the link function is the cumulative logit.

The model uses a single parameter vector θ, while the thresholds t_j determine the boundaries between classes. As a result, the model has fewer parameters than multinomial logistic regression.

The model parameters are estimated by maximizing the log-likelihood

$$\ell(\theta) = \sum_{i=1}^{n} \log P(y_i \mid x_i),$$

which again leads to an optimization problem without a closed-form solution, so the parameters are estimated numerically.