Discriminative Models of Text Classification and Regression

4

From Random Experiments to Natural Language Processing – by Wilker Aziz. This book has not been published yet.

To complete this class you need to know the following:

1. From *probability theory*:
   - discrete and continuous random variables, probability mass function (pmf), probability density function (pdf);
2. From *statistics*:
   - pmfs/pdfs of single-parameter (*e.g.*, Bernoulli, Poisson), two-parameter (*e.g.*, Beta, Gamma, Normal), and multi-parameter (*e.g.*, Categorical, Dirichlet) distributions;
3. From *linear algebra*:
   - vector and matrix multiplication;
4. From *calculus*:
   - derivatives, partial derivatives, and gradient.

We often need to analyse documents in terms of diverse properties. Where these properties are discrete and finite (*e.g.*, negative/neutral/positive, or spam/no-spam, or politics/sport/science) we talk about text analysis as **text classification** (or text categorisation), where these properties are numerical we talk about text analysis as **text regression**.

Examples of numerical quantities we may be interested in:

- Number of people attending an event, based on tweets posted about that event in a given interval of time. Example: 100,000 people are predicted to attend the pride in Amsterdam.
- Demand for a product (expressed in number of items), based on reviews on an online platform. Example: we need to stock some 10,000 iPads.
- Market value of a product (expressed in euros), based on description. This guitar tuner must be worth EUR 17.99.
- Polling data (expressed as percentages), based on how voters talk about candidates on social media, or based on a survey voters filled in (where the survey has some open questions). Example: 53% of the votes are going to party A, 21% of the votes to party B, 11% to party C, 15% of the voters are unsure about their voting intentions.

Text analysis often concerns an extrinsic property, somewhat elicited by the text, but not really part of the text without it being embedded in a wider human context. For example, a piece of text approximately conveys the sentiment of whoever wrote it, and, upon reading it, we might be able to approximately retrieve what that sentiment was. A collection of reviews of a product may contain information that helps predict the demand for that product in the next trimester. These predictions convey information about a specific dataset (a population) and are subject to various degrees of approximation (from data collection, to task definition, to modelling assumptions).

4.1 Data and task definition

Our approach is data-driven, that is, we observe a collection $\mathcal{D}$ of documents annotated along one or more numerical dimensions of interest, and learn a model that reproduces (aspects of) it.

An observation is a pair (x, y), where $x \in \mathcal{X}$ is a document, a sequence $x = \langle w_1, \ldots, w_l \rangle$ of $l = |x|$ words, each from a finite vocabulary $\mathcal{W} = \{1, \ldots, V\}$ of known words, and $y \in \mathcal{Y}$ is a response (or target) variable. In text classification, $\mathcal{Y}$ is a countably finite set of K disjoint categories (*e.g.*, `negative/neutral/positve`, `spam/not-spam`). In text regression, $\mathcal{Y}$ can be a countably infinite set of ordinal numbers (e.g., $\mathbb{N}$, $\mathbb{Z}$) or a subset of it (*e.g.*, $[-3, \ldots, 3] \subset \mathbb{Z}$), it can be an uncountable set such as the real line $\mathbb{R}$ or the real coordinate space $\mathbb{R}^K$, or a subset of it (*e.g.*, $(0, 1) \subset \mathbb{R}$, $\mathbb{R}_{>0} \subset \mathbb{R}$, or $\Delta_{K-1} \subset \mathbb{R}^K$).

As from now own we will be manipulating both discrete and continuous random variables, we will often state results in terms of probability mass or density functions rather than probability distributions.[1]

1: A **probability mass function** $f(y; \phi)$ prescribes the probability with which the random variable Y takes on a specific value $y \in \mathcal{Y}$ via the relation $P_Y(Y = y) = f(y; \phi)$. A pmf allows us to specify a probability distribution by relating an outcome y, a set of parameters ϕ and a probability mass $f(y; \phi)$ through a mathematical law. For example, the Bernoulli pmf is defined for $y \in \{0, 1\}$, it has a single parameter $0 \leq \phi \leq 1$, and it is defined as $f(y; \phi) = \phi^y (1 - \phi)^{1-y}$. A **probability density function** $f(y; \phi)$ prescribes the probability with which the random variable Y takes on any value in an interval $(a, b) \subseteq \mathcal{Y}$ via the relation $P_Y(y \in (a, b)) = \int_a^b f(y; \phi)\, \mathrm{d}y$. A pdf allows us to specify a probability distribution by relating an outcome y, a set of parameters ϕ and a probability density $f(y; \phi)$ through a mathematical law. For example, the Exponential pdf is defined for $y \in \mathbb{R}_{>0}$, it has a single parameter $\phi > 0$, and it is defined as $f(y; \phi) = \phi e^{-\phi y}$.

Statistical task

In statistical terms, we want a mechanism that can be used to map any document $x \in \mathcal{X}$ to the probability that the response variable will take on a certain value (for discrete responses, where $\mathcal{Y}$ is countable) or fall within a certain range of values (for continuous responses, where $\mathcal{Y}$ is uncountable). For example, if we read tweets about an event, we may want to predict the probability that 1,000,0000 people will attend it. If we read comments about a politician running for office, we may be interested in predicting whether she will receive about more than 50% of a town's votes in the next election. A bit more formally, we want to map $x \in \mathcal{X}$ to a conditional probability distribution $P_{Y|X=x}$ over $\mathcal{Y}$. We do so by mapping x to the parameter(s) of the pmf (for discrete responses) or of the pdf (for continuous responses) of a certain parametric family of distributions.

NLP task

In NLP applications, the practitioner generally wants to map text x to a *single* value of the responsible variable (*e.g.*, to decide whether something is or isn't a spam, to decide whether to stock 100,000 iPads, whether someone is leading a poll). The NLP task therefore ignores any uncertainty inherent to the mapping from x to y. In order to do so, the practitioner needs to choose a **decision rule**, that is, an algorithm to map from the distribution $P_{Y|X=x}$—the output of the statistical task—to a single decision. In text categorisation, the *most probable class* is a common rule. This rule predicts the mode of the conditional distribution. When dealing with continuous responses, it may happen that the outcome with highest density is not qualitatively interesting (see Figure 4.1). To deal with the decision problem the practitioner might introduce a utility function $u(y, o)$ which quantifies the benefit in choosing outcome o when

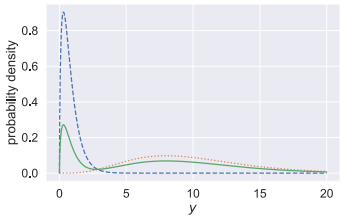

Figure 4.1: The figure shows three pdfs over the positive real line. The first two are unimodal (dashed and dotted), they each have a single hilltop, whereas the third (solid) is bimodal (it is in fact obtained by combination of the other two). The outcome with highest density in the combined pdf is somewhere close to $y = 0.3$, it coincides with the hilltop of the first pdf. That hilltop however does not accumulate as much probability mass as the second (lower) hilltop around $y = 7.8$. In some cases, it can be argued that a prediction close to the 'centre of the lower hilltop' is "safest" to make. Decision theory can formalise this notion as we shall see.

the true response is y. Then, given $X = x$, we choose the outcome o that maximises *expected utility*:

$$y^\star = \arg\max_{o \in \mathcal{Y}} \mathbb{E}[u(Y, o)|X = x] \,, \tag{4.1}$$

where we use the probability distribution $P_{Y|X=x}$ to quantify our uncertainty about the true response. Some decision makers prefer to think in terms of a function $\ell(y, o)$ that quantifies the loss incurred when we choose o rather than the true response y. In that case, the decision rule is to minimise expected loss: $\arg\min_{o \in \mathcal{Y}} \mathbb{E}[\ell(Y, o)|X = x]$. In most cases, the two views are equivalent.

When $\mathcal{Y}$ has K elements, the most probable class rule is a special case of this general decision rule, where we choose the 'exact match' utility $u(y, o) = [y = o]$. Proof using $f(y)$ as the pmf/pdf associated with $P_{Y|X=x}$: $\mathbb{E}[u(Y, o)|X = x] = \sum_{y=1}^{K} f(y)[y = o] = f(o)$, and thus $\arg\max_{o \in \mathcal{Y}} \mathbb{E}[u(Y, o)|X = x] = \arg\max_{o \in \mathcal{Y}} f(o)$.

4.2 Familiar modelling ideas

Why not tabular CPDs?

Having already learnt about tabular conditional probability distributions (cpds) one might ask 'why not use tabular cpds here?'.

There are a few reasons. First and foremost, our cpds now might not look like tables. As the response variable may come from an infinite set, we would need infinitely many values to completely specify the cpd $P_{Y|X=x}$, for any given x. Luckily, pmfs and pdfs solve that problem for us. Rather than storing the probabilities themselves, we need only store a parameter vector that can be used to compute probability values on demand using a mathematical law. For example, rather than storing the infinitely many probability values of a Poisson distribution, we need only store the Poisson rate (a single number). Another example, rather than storing the infinitely many probability values of a Normal distribution, we need only store the Normal location (a single real number) and scale (a single strictly positive real number). A table is a good device to store a Categorical distribution because the Categorical parameter is a vector of finite dimensionality, but, in general, what we really store is the parameter of a pmf (or of a pdf).

So, for any given $x \in \mathcal{X}$ we simply store the parameter $\phi^{(x)}$ that specifies the pmf (or pdf) that prescribes the cpd $P_{Y|X=x}$. Great, right? Not really, just like before, there are infinitely many documents we may be interested in. Hence, storing one parameter vector for each and every x is an impossible task, computationally speaking. Statistically speaking, we have another problem—the same we encountered before. We estimate the numerical values of each $\phi^{(x)}$ from data, with finitely many data points that we can observe it is really hard to learn meaningful estimates for infinitely many cpds.

Why not naive Bayes?

Having alreayd learnt about naive Bayes models, one might ask 'how about we relate X and Y through a joint distribution P_{YX}, make a conditional independence assumption, and infer $P_{Y|X=x}$ from simpler factors $P_{W|Y=y}$?'.

Excellent question, but, as always, it is all about the details. The NB model depends crucially on the set of possible target values being *finite*. If we have infinitely many possibilities for y, then we would have infinitely many cpds of the kind $P_{W|Y=y}$.

Here is the punchline. For a general text analysis model, not only the documents come from an infinite set $\mathcal{X}$, but also the responses come from an infinite set $\mathcal{Y}$.

4.3 Feature Functions

Our key modelling idea is to realise the mapping from documents to $P_{Y|X=x}$ in a mathematical way, by relating x and some parameters that we can adjust in order to fit $P_{Y|X=x}$ to data. To do that, we will need to introduce a tool that allows us to treat a document as if it were a point in the *real coordinate space* $\mathbb{R}^D$. This tool is what we call a feature function.

A **vector-valued feature function**

$$\mathbf{h} : \mathcal{X} \to \mathbb{R}^D \tag{4.2}$$

maps a document $x \in \mathcal{X}$ to a D-dimensional **feature vector**. The feature vector is something that "describes" a document along D numerical dimensions, each of which quantifies an aspect of the problem that's believed to play some significant role in the kind of analysis we are making.

Take the example of spam detection, where our goal is to map an email x to the probability with which it is or isn't a spam. Useful features might capture whether

- the email subject line is all capital letters;
- the email contains phrases such as "urgent reply", "reply imediately";
- the email contains phrases such as "will be removed", "will be deleted";
- the email contains links to certain block-listed sites;
- the relative frequency of the word "click".

Our feature function could then be

$$\begin{aligned}
h_1(x) &= \text{IsTitleAllCaps}(x) \\
h_2(x) &= [\text{'urgent reply'} \in x \text{ or 'reply immediately'} \in x] \\
h_3(x) &= [\text{'will be removed'} \in x \text{ or 'will be deleted'} \in x] \\
h_4(x) &= \frac{1}{l} \sum_{i=1}^{l} [\text{click} = w_i]
\end{aligned}$$

This feature function in particular is the 4-dimensional vector $\mathbf{h}(x) = (h_1(x), h_2(x), h_3(x), h_4(x))^\top$ whose coordinates are defined as above.

See Figure 4.2 for another example.

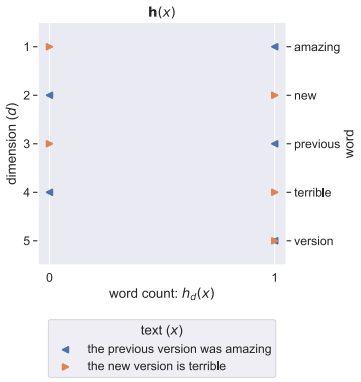

Figure 4.2: This feature function maps sentences to a feature space whose coordinates capture the number of times a certain word occurred. In this toy example, our feature function only knows 5 words (shown in the vertical axis). Therefore, our feature vectors are 5-dimensional (*i.e.*, $\mathbf{h}(x) \in \mathbb{R}^5$, no matter the x).

4.4 Generalised Linear Models

Our new general tool for text analysis (both classification and regression) is a *generalised linear model* (GLM). A GLM is a conditional model of $Y|X = x$ where we compute the parameter $\phi^{(x)}$ of our conditional statistical model with the help of a parametric linear transformation of $\mathbf{h}(x)$. The linear transformation has its own parameters (the coefficients we multiply and the biases we add), which we denote generically by $\boldsymbol{\theta}$.

The key idea is that rather than storing the statistical parameter $\phi^{(x)}$, for every x, we will predict it whenever needed using a function

$$\phi^{(x)} = g(x; \boldsymbol{\theta}) \,. \tag{4.3}$$

Then, by plugging the predicted parameter value in our choice of pmf/pdf $f(y; \phi^{(x)})$, we will be able to assign probability mass/density for any value $y \in \mathcal{Y}$ of the response variable.

Next, we introduce GLMs by discussing example models.

Bernoulli GLM

In spam classification we want to map a document $x \in \mathcal{X}$ to the probability of it being spam or not, let's denote this probability by $g(x; \boldsymbol{\theta})$ and recall that $0 \leq g(x; \theta) \leq 1$. The statistical model of interest is

$$Y|X = x \sim \text{Bernoulli}(g(x; \boldsymbol{\theta})) \tag{4.4}$$

where $g(x; \boldsymbol{\theta})$ is the *probability of labelling a document as spam*.

Let's define this function. As documents are not objects in a numerical space, our GLMs start by mapping x to a feature vector via a feature function $\mathbf{h}(x)$. Once that is done, a bit of linear algebra can help us map the feature vector to a single number, and then to a probability value.

A possible GLM for this binary classification problem is the following:

$$g(x; \boldsymbol{\theta}) = \text{sigmoid}(\mathbf{w}^\top \mathbf{h}(x) + b) \tag{4.5}$$

where $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the parameters of the model $\boldsymbol{\theta} = \{\mathbf{w}, b\}$. Step by step:

1. we map $x \in \mathcal{X}$ to a feature vector $\mathbf{h}(x) \in \mathbb{R}^D$; this is a deterministic step and it does not require interacting with the parameters $\boldsymbol{\theta}$ of the GLM—see Figure 4.2;
2. we take the dot-product between the feature vector and the parameter $\mathbf{w}$, which gives us a scalar (a real value);
3. we add a bias value b to that scalar, obtaining a scalar (a real value).

The dot product between $\mathbf{a} \in \mathbb{R}^D$ and $\mathbf{b} \in \mathbb{R}^D$ denoted $\mathbf{a}^\top \mathbf{b}$ is defined as $\sum_{d=1}^{D} a_d b_d$. Note that $\mathbf{a}^\top \mathbf{b} \in \mathbb{R}$.

The sigmoid is a function $\text{sigmoid} : \mathbb{R} \to (0, 1)$ defined as $\text{sigmoid}(a) = \frac{1}{1+e^{-a}}$.

The linear function $\mathbf{w}^\top \mathbf{h}(x) + b$ maps the document to a real value that has the right dimensionality (i.e., the dimensionality of the Bernoulli parameter is 1). The result of this operation is called the **linear predictor**. While this value has the right dimensionality, it is not correctly constrained to be a valid parameter. That is, the linear predictor can be negative, positive, or any number really (of whatever magnitude). A valid Bernoulli parameter is a number constrained between 0 and 1, so, to obtain such

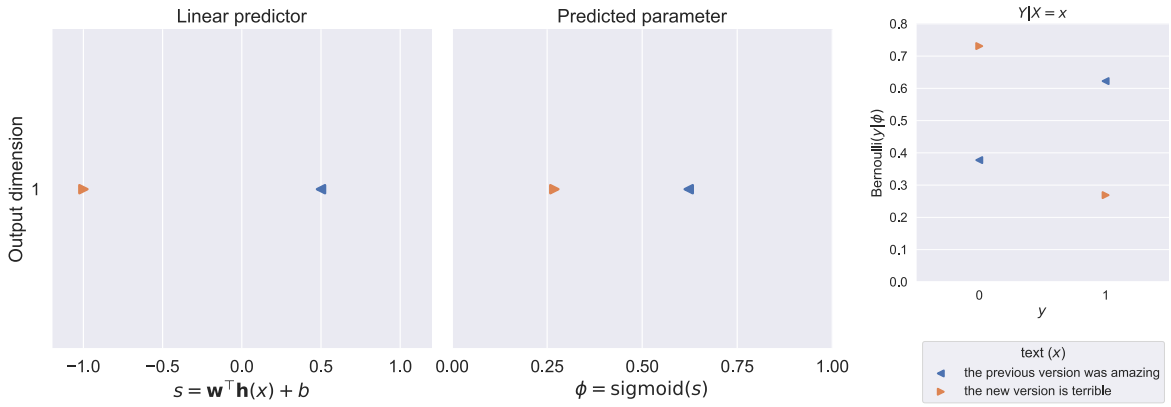

Figure 4.3: Here we show the linear predictor (left) and the Bernoulli parameter (centre) that the GLM predicts for two pieces of text, each represented using the feature function from Figure 4.2 and the parameters $\mathbf{w} = (1, 1/2, 0, -1, 0)^\top$, each associated with one dimension of the feature space, and $b = -1/2$, which shifts the result of the dot product. Using the GLM output (centre) we can prescribe the distribution of Y given $X = x^{(1)}$ or given $X = x^{(2)}$ via the Bernoulli pmf (right).

a valid parameter this GLM uses the sigmoid function, whose output is *always* constrained to being a valid probability value (between 0 and 1), no matter the input. Figure 4.3 illustrates GLM parameterisation of conditional distributions for two pieces of text.

The GLM has *linear* in its name because the predictors $\mathbf{h}(x)$ and the parameters $\boldsymbol{\theta}$ interact linearly, it has *generalised* in its name because the final mapping from the linear predictor to the statistical parameter of the distribution need not be linear. In statistics this last step of the mapping (which was realised by the sigmoid function in this example) is called *the inverse link function*. A name that is more popular in machine learning, and, in particular, in deep learning is *activation function*. In deep learning, the input to the activation function (that is, the linear predictor) is often called *score* or *logit* in the context of a binary classification model.

Categorical GLM

This idea is so powerful that we can build pretty much every text classifier and every text regressor in the book. For example, for a K-way classifier

$$Y|X = x \sim \text{Cat}(\mathbf{g}(x; \boldsymbol{\theta})) \tag{4.6}$$

here $\mathbf{g}(x; \boldsymbol{\theta})$ must be a K-dimensional vector of positive values that sum to 1. This is a valid GLM for this model:

$$\mathbf{s} = \mathbf{W}^\top \mathbf{h}(x) + \mathbf{b} \tag{4.7a}$$

$$\mathbf{g}(x; \boldsymbol{\theta}) = \text{softmax}(\mathbf{s}) \tag{4.7b}$$

where $\boldsymbol{=} \{\mathbf{W}, \mathbf{b}\}$ are the parameters of the model, $\mathbf{W} \in \mathbb{R}^{D \times K}$ is a matrix that maps D inputs (the feature values) to K outputs (real values), $\mathbf{b} \in \mathbb{R}^K$ is a vector of biases (one real value bias per class), and softmax is a K-dimensional vector-valued function whose jth coordinate is

$$[\text{softmax}(\mathbf{s})]_j = \frac{\exp(s_j)}{\sum_{k=1}^{K} \exp(s_k)} . \tag{4.8}$$

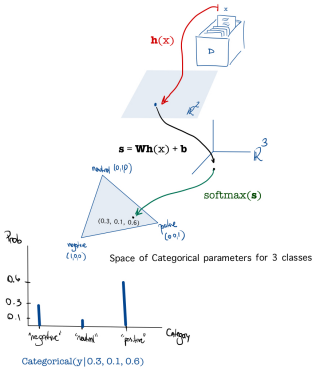

Figure 4.4: A document x is first mapped to a vector-valued feature representation $\mathbf{h}(x) \in \mathbb{R}^D$ using a predefine feature function (in this illustration $D = 2$). A linear transformation then maps the feature vector to K real values (in this illustration $K = 3$), these are known as linear predictors. The linear predictors have the right dimensionality to parameterise our Categorical pmf, but they are not valid parameter values (the Categorical pmf requires positive values that sum to 1). Last, but not least, we let the softmax function map the K linear predictors to a K-dimensional probability vector (*i. e.*, a point in the simplex $\Delta_{K-1} \subset \mathbb{R}^K$). This is one way to map from text to a Categorical pmf through a parametric function: $\mathbf{W} \in \mathbb{R}^{K \times D}$ and $\mathbf{b} \in \mathbb{R}^K$ are the free parameters of this model (*i. e.*, no matter what values we choose for those, the mapping will remain valid; later we discuss a good way to choose them).

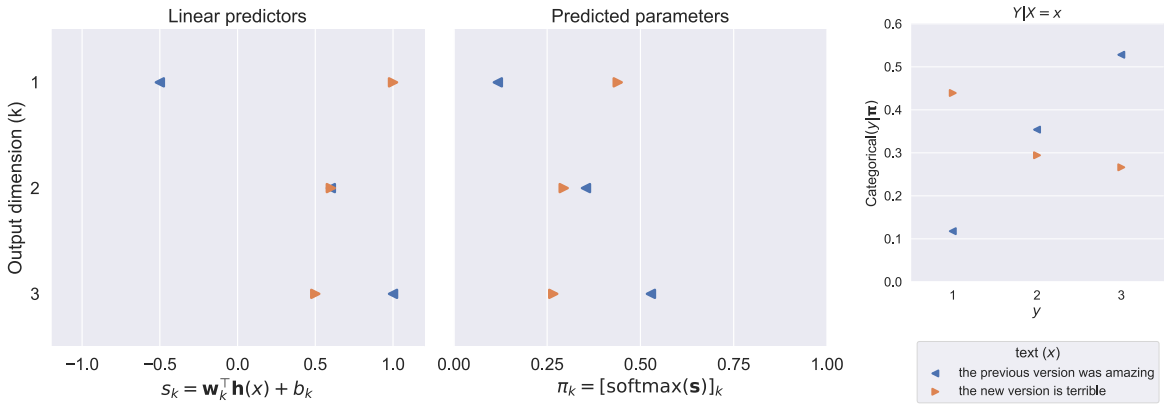

Figure 4.5: Here we show the K linear predictors (left) and the K Categorical parameters (centre) that the GLM predicts for two pieces of text, each represented using the feature function from Figure 4.2. In this example, the GLM parameters are $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)^\top$ with $\mathbf{w}_1 = (0, 0, -1/2, 1, 0)^\top$, $\mathbf{w}_2 = (0, 1/10, 1/10, 0, 1/2)^\top$, $\mathbf{w}_3 = (1, 1/2, 0, 0, 0)^\top$ and $\mathbf{b} = (0, 0, 0)^\top$. Using the GLM output (centre) we can prescribe the distribution of Y given $X = x^{(1)}$ or given $X = x^{(2)}$ via the Categorical pmf (right).

In deep learning literature the vector $\mathbf{s}$ that we use as input to the softmax function is often called the vector of *scores* or *logits*, each one of its coordinates quantifies the importance of the corresponding class in a logarithmic scale. This model is also known as a *log-linear* model, that is because the logarithm of the softmax is a linear function of $\mathbf{h}(x)$ and $\boldsymbol{\theta}$.

Poisson GLM

Let's attempt to parameterise a conditional distribution over the number of likes Y that a tweet x will receive, based on its content as captured by a feature function $\mathbf{h}(x) \in \mathbb{R}^D$. Suppose that for this statistical model, we decide to use a Poisson distribution, then

$$Y|X = x \sim \text{Poisson}(g(x; \boldsymbol{\theta})) \tag{4.9}$$

because this is a Poisson, we know that $g(x; \boldsymbol{\theta})$ must be strictly positive. So, here's a valid GLM:

$$g(x; \boldsymbol{\theta}) = \text{softplus}(\mathbf{w}^\top \mathbf{h}(x) + b) \tag{4.10}$$

with model parameters are $\boldsymbol{\theta} = \{\mathbf{w}, b\}$ with $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$.

The parameter of the Poisson distribution Poisson(λ) is called *rate*, and it holds that $\lambda \in \mathbb{R}_{>0}$. The Poisson rate is also its mean and variance: $\mathbb{E}[\text{Poisson}(\lambda)] = \text{Var}(\text{Poisson}(\lambda)) = \lambda$. The Poisson pmf is given by $\text{Poisson}(y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!}$.

For $s \in \mathbb{R}$, the softplus output $\text{softplus}(s) = \log(1 + \exp(s))$ is *strictly* positive (that is, larger than zero).

Normal/Gaussian GLM

Let's attempt to parameterise a conditional distribution over the market value Y of a product based on product description and reviews x. Again, we count on a feature function $\mathbf{h}(x) \in \mathbb{R}^D$ that captures aspects of the description and reviews that are essential for the task. Suppose that for this statistical model, we decide to use a Normal/Gaussian distribution with fixed variance, then

$$Y|X = x \sim \mathcal{N}(g(x; \boldsymbol{\theta}), 1^2) \tag{4.11}$$

The parameters of the Normal distribution $\mathcal{N}(\mu, \sigma^2)$ are called *location* and *scale*. For the location, it holds that $\mu \in \mathbb{R}$. For the scale, it holds that $\sigma \in \mathbb{R}_{>0}$. The Normal location is also its mean, that is, $\mathbb{E}[\mathcal{N}(\mu, \sigma^2)] = \mu$. The Normal scale is also its standard deviation (the squared root of its variance): $\text{Var}(\mathcal{N}(\mu, \sigma^2)) = \sigma^2$. The Normal pdf is given by $\mathcal{N}(y|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma}\right)$.

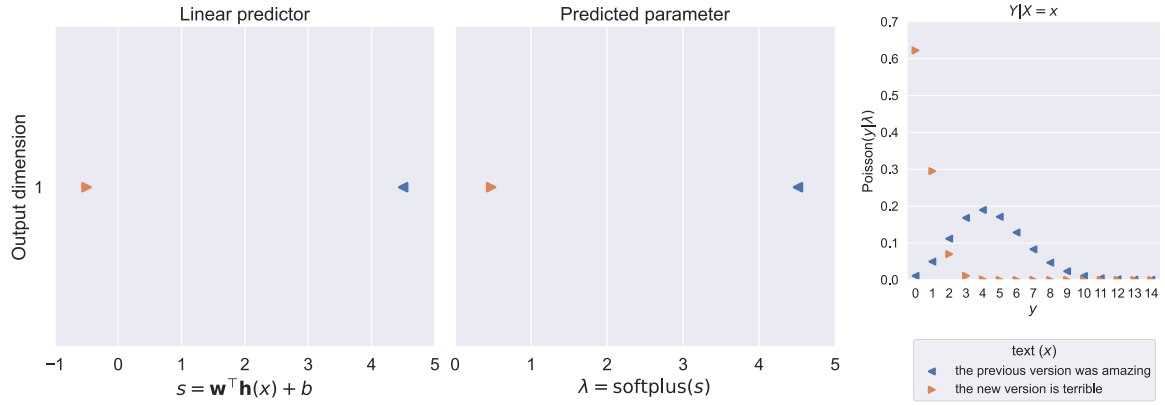

Figure 4.6: Here we show the linear predictor (left) and the Poisson rate (centre) that the GLM predicts for two pieces of text, each represented using the feature function from Figure 4.2. In this example, the GLM parameters are $\mathbf{w} = (2, -1, 0, -2, -1)^\top$ and $b = 3/2$. Using the GLM output (centre) we can prescribe the distribution of Y given $X = x^{(1)}$ or given $X = x^{(2)}$ via the Poisson pmf (right).

because this is a Normal, we now that $g(x; \boldsymbol{\theta})$ must be real valued. So, here's a valid GLM:

$$g(x; \boldsymbol{\theta}) = \mathbf{w}^\top \mathbf{h}(x) + b \tag{4.12}$$

with model parameters are $\boldsymbol{\theta} = \{\mathbf{w}, b\}$ with $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$. We might even attempt to predict the scale, rather than have it fixed, for example:

$$Y|X = x \sim \mathcal{N}(\mu, \sigma^2) \tag{4.13a}$$
$$(\mu, \sigma)^\top = \mathbf{g}(x; \boldsymbol{\theta}) \tag{4.13b}$$
$$g_1(x; \boldsymbol{\theta}) = \mathbf{w}^\top \mathbf{h}(x) + b \tag{4.13c}$$
$$g_2(x; \boldsymbol{\theta}) = \exp\left(\mathbf{m}^\top \mathbf{h}(x) + c\right). \tag{4.13d}$$

Where we constrain the second output to being strictly positive with the exponential activation, because the Normal/Gaussian scale must be a strictly positive number. This time our GLM has more parameters $\boldsymbol{\theta} = \{\mathbf{w}, \mathbf{m}, b, c\}$ with $\mathbf{w}, \mathbf{m} \in \mathbb{R}^D$ and $b, c \in \mathbb{R}$.

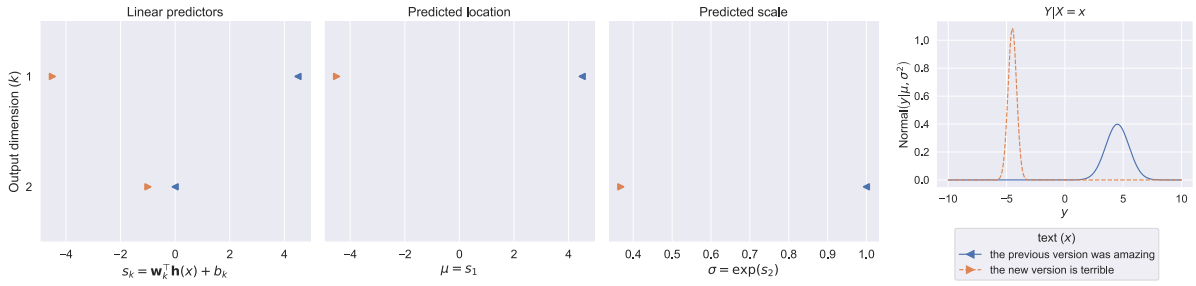

Figure 4.7: Here we show the linear predictors (left) and the Normal parameters (location and scale) that the GLM predicts for two pieces of text, each represented using the feature function from Figure 4.2. In this example, the GLM parameters are $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2)^\top$ with $\mathbf{w}_1 = (2, -5, 0, -2, 1)^\top$, $\mathbf{w}_2 = (-1, -1, 0, -1, 1)^\top$ and $\mathbf{b} = (3/2, 0)^\top$. Using the GLM output (centre) we can prescribe the distribution of Y given $X = x^{(1)}$ or given $X = x^{(2)}$ via the Normal pdf (right).

4.5 General principles for prescribing GLMs

1. Start with choosing the family you will model with. This depends on the type of data you have (e.g., binary means Bernoulli, K-way classification usually means Categorical, ordinal regression usually means Poisson, but it could be some other distribution over natural numbers or a subset thereof, like the Binomial, continuous regression might require a Normal distribution, regressing to vectors of proportions might require a Dirichlet distribution, etc.). You don't need to know all these distributions by heart, when needed, we will give you information about them that will help you judge their relevance in context.
2. The input is text, but GLMs operate with inputs in the real coordinate space, so you need a vector-valued feature function $\mathbf{h}(x)$. For now, you have to design that yourself, mostly by hand with the aid of some handcrafted rules (later we will give you machine learning devices to learn feature functions from data).
3. In a GLM, the input $\mathbf{h}(x)$ and the parameters $\boldsymbol{\theta}$ interact linearly. This constrains us to either operations like dot product, matrix multiplication and scalar or vector addition. Whether we have "dot product plus scalar" or "matrix multiplication plus vector" depends exclusively on the dimensionality we need for the linear predictor. If we need a single scalar, we will use the former. If we need a vector, we will use the latter.
4. Example 1: the Poisson parameter is a single scalar, thus we know that we need to map from $\mathbf{h}(x)$ to a single scalar, as we achieve with "dot product plus scalar". Example 2: the Categorical parameter is a vector, thus we know that we need to map from $\mathbf{h}(x)$ to a K-dimensional vector, as we achieve with "matrix multiplication plus vector".
5. Finally, the statistical parameter is generally constrained to a subset of the real numbers, so we need an activation function that constrains the linear predictor accordingly. Example 1: the Poisson parameter is *strictly positive* by definition, so we need to wrap the linear function around something whose output is never negative and never 0, no matter which real-valued input we give it. The exponential function does that for us. There are other activation functions that achieve the same result, but the exponential is convenient for certain reasons (e.g., it's logarithm is linear). In our implementation below we will see other options. Example 2: the Categorical parameter must be a probability vector, the softmax function can realise that constraint for us.

4.6 Parameter Estimation

Given a training set $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$ of N observed input-target pairs, we would ideally assess the log-likelihood of the model:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log f(y^{(n)}; \phi_n) \qquad \text{with } \phi_n = g(x^{(n)}; \boldsymbol{\theta}) \,, \tag{4.14}$$

which depends on the parameter $\boldsymbol{\theta}$ through our choice of pmf/pdf $f(y;\phi)$, and then choose the parameter $\boldsymbol{\theta}$ that maximises it:

$$\boldsymbol{\theta}^{\star} = \arg\max_{\boldsymbol{\theta}} \mathscr{L}_{\mathscr{D}}(\boldsymbol{\theta}) \,. \tag{4.15}$$

The *gradient* of a function $b(\mathbf{v})$ with respect to $\mathbf{v} \in \mathbb{R}^D$, denoted $\boldsymbol{\nabla}_{\mathbf{v}} b(\mathbf{v})$, is the vector of partial derivatives $(\frac{\partial}{\partial v_1} b(\mathbf{v}), \ldots, \frac{\partial}{\partial v_D} b(\mathbf{v}))^{\top}$. In machine learning, where $\boldsymbol{\theta}$ is a collection $\{\mathbf{w}, b\}$ of parameters, when we write $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\theta})$, we mean both $\boldsymbol{\nabla}_{\mathbf{w}} \mathscr{L}(\mathbf{w}, b)$ and $\frac{\partial}{\partial b} \mathscr{L}(\mathbf{w}, b)$.

Unlike tabular categorical cpds, there is no simple expression for the MLE of a GLM. But, we can employ a gradient-based search. This search uses the gradient $\nabla_{\boldsymbol{\theta}} \mathscr{L}_{\mathscr{D}}(\boldsymbol{\theta})$ to iteratively update an existing $\boldsymbol{\theta}$, starting from an initial guess $\boldsymbol{\theta}^{(0)}$, which is typically a random initialisation of the parameters.

At iteration t, the update rule is

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \gamma_t \nabla_{\boldsymbol{\theta}^{(t)}} \mathscr{L}_{\mathscr{D}}(\boldsymbol{\theta}^{(t)}) \tag{4.16}$$

If you have seen the update formula before with a *minus* rather than a *plus* for the gradient, don't worry, it is the same notion. You *sum* the gradient if you are maximising the log-likelihood, and you *subtract* the gradient if you are minimising the negative log-likelihood. The two procedures yield the exact same optimum.

where the log-likelihood is assessed using the current parameters, we then obtain the gradient for it, and combine it with the current parameters to get the next iterate. The quantity $\gamma_t > 0$ is called a *learning rate*.

In certain instances the learning rate γ_t is the same fixed value no matter the t. But, for certain algorithms (*e.g.*, stochastic gradient ascent/descent) it must decay over time (for the algorithm's mathematical correctness). Examples of schedule include $\gamma_t = \frac{\gamma_0}{1+t}$ and $\gamma_t = \frac{\gamma_0}{\log_{10}(10+t)}$, where $\gamma_0 > 0$ is a fixed initial value.

Stochastic gradient-based optimisation

Assessing the log-likelihood of a certain value of the parameter vector $\boldsymbol{\theta}$ requires assessing the probability mass (or density, for continuous variables) of each one of our observations under the current value of the parameter. Each one such assessment on its own is not at all challenging, but assessing all the N terms can be challenging for large N.

Fortunately, we can use a stochastic gradient procedure, which still has the same guarantees as the deterministic procedure. At each iteration t, we compute an approximation to $\nabla_{\boldsymbol{\theta}^{(t)}} \mathscr{L}_{\mathscr{D}}(\boldsymbol{\theta}^{(t)})$. This approximation is a Monte Carlo (MC) estimate obtained using $S < N$ data points uniformly sampled from $\mathscr{D}$:

$$\nabla_{\boldsymbol{\theta}^{(t)}} \mathscr{L}_{\mathscr{D}}(\boldsymbol{\theta}^{(t)}) \overset{\text{MC}}{\approx} \frac{1}{S} \sum_{n=1}^{S} \nabla_{\boldsymbol{\theta}^{(t)}} \log f(y^{(s)};\phi_s) \quad \text{with } \phi_s = g(x^{(s)};\boldsymbol{\theta}^{(t)}) \tag{4.17}$$

We can obtain this gradient by essentially pretending, at each iteration t, that the log-likelihood function depends only on a small *batch* of S observations $\{(x^{(s)}, y^{(s)})\}_{s=1}^{S} \subset \mathscr{D}$ drawn from the training set:

$$\mathscr{L}_{\mathscr{B}}(\boldsymbol{\theta}^{(t)}) = \frac{1}{S} \sum_{s=1}^{S} \log f(y^{(s)};\phi_s) \quad \text{with } \phi_s = g(x^{(s)};\boldsymbol{\theta}^{(t)}) \,. \tag{4.18}$$

Computing the gradient of the log-likelihood (its exact value or MC estimate) is a tedious task, but, luckily, it can be exactly and efficiently automated for all continuously differentiable functions. *Automatic differentiation* is a powerful tool in machine learning and statistics, one of its efficient algorithmic implementations is the well known gradient **backpropagation** algorithm. In this course, we will rely on software packages that implement it for us. Our only practical concern is to guarantee that the operations that map from $\mathbf{h}(x)$ and $\boldsymbol{\theta}$ to a probability mass (or density) value $f(y; g(x;\boldsymbol{\theta}))$ are all continuously differentiable functions

of $\boldsymbol{\theta}$. Luckily that is a very mild requirement, which the linear function (which gives us linear predictors) satisfies, and the same is true for the vast majority of activation functions (*e.g.*, exp, sigmoid, softmax, softplus, *etc.*).

Regularisation

Oftentimes, we have many features, and thus *many parameters*. This gives models the capacity to discover correlations that have no real predictive power (e.g., that the token 'capivaras' implies a negative sentiment, simply because the one time that word was seen in the data was in a document labelled with the negative class, for example the document might have been "I loved the capivaras but overall the horrible weather ruined the trip"). These are called *spurious correlations* and we would rather not be misled by them.

In an attempt to get rid of them, we employ a so called **regulariser**. This is a penalty on the objective based on the norm of the parameter vector, and we usually employ the L2 norm.

The L2 norm of a vector $\mathbf{v} \in \mathbb{R}^D$ is $\mathscr{R}(\mathbf{v}) = \sqrt{\sum_{d=1}^{D} v_d^2}$. For a collection $\boldsymbol{\theta}$ of parameter vectors, we sum the L2 norms of each vector in the collection.

Thus, the final objective for *regularised maximum likelihood estimation* is

$$\boldsymbol{\theta}^\star = \arg\max_{\boldsymbol{\theta}} \; \mathscr{L}_{\mathcal{D}}(\boldsymbol{\theta}) - \lambda \mathscr{R}(\boldsymbol{\theta}) \tag{4.19a}$$

or, equivalently,

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \; -\mathscr{L}_{\mathcal{D}}(\boldsymbol{\theta}) + \lambda \mathscr{R}(\boldsymbol{\theta}) \;, \tag{4.19b}$$

where $\lambda \geq 0$ is a hyperparameter used to control the importance of the regulariser. Unfortunately, there is no simple way to choose the value of λ directly from the training data, we have to try a handful of values and test the model's performance on heldout data.