Malaviya National Institute of Technology
Jaipur
Department of Computer Science and Engineering

Coding Competition

360 Degree Visualization of Non-Linear Classification Using Multinomial Logistic Regression

1 Introduction

Logistic regression is a probabilistic classifier that makes use of supervised machine learning. Machine learning classifiers require a training corpus of n input/output pairs (x_n, y_n). A machine learning system for classification then has four components: Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. It is a kind of statistical algorithm that analyzes the relationship between a set of independent variables and the dependent binary variables.

1. A feature representation of the input. For each input observation x_i, this will be a vector of features $[x_1, x_2, \cdots, x_n]$. We will generally refer to feature j for input x_i as x_{ij}, sometimes simplified as x_i, but we will also see the notation f_i, $f_i(x)$, or, for multiclass classification, $f_i(c, x)$.

2. A classification function that computes $\hat{y}$, the estimated class, via $p(y|x)$. In the next section, we will introduce the sigmoid and softmax tools for classification.

3. An objective function for learning, usually involving minimizing errors on training examples. We will introduce the cross-entropy loss function.

4. An algorithm for optimizing the objective function, for example, the stochastic gradient descent algorithm.

Logistic regression has two phases:

training: We train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.
test: Given a test example x we compute $p(y|x)$ and return the higher probability label $y = 1$ or $y = 0$.

A brief description of training, validation, and testing is given in Appendix 4.

2 The Sigmoid function

The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of a new input observation. Here,, we introduce the sigmoid classifier to help us make this decision. Consider a single input observation $x \in R^d$ (d dimensional), which we represent by a vector of d features $[x_1, x_2, \cdots, x_d]$. The classifier output y can be 1 (meaning the observation is a class member) or 0 (the observation is not a class member).

For example, to know the probability $P(y = 1|x)$ that this observation is a class member. The decision is 'positive' versus 'negative.' The features represent counts of words in a document, $P(y = 1|x)$ is the probability that the document has positive sentiment, and $P(y = 0|x)$ is the probability that the document has negative sentiment. Logistic regression solves this task by learning, from a training set, a vector of weights and a bias term. Each weight w_j is a real number associated with one of the input features x_j. The weight w_j represents how important that input feature x_j is to the classification decision and can be positive (providing evidence that the instance being classified belongs in the positive class) or negative (providing evidence that the instance being classified belongs in the negative class). Thus, we might expect the word awesome to have a high positive weight in a sentiment task and bias term abysmal to have a very negative weight. The bias term, the intercept, is another real number added to the weighted inputs.

To decide on a test instance—after we've learned the weights in training, the classifier first multiplies each x_{ij} by its weight w_j, sums up the weighted features and adds the bias term b. The resulting single number s (signal) expresses the weighted sum of the evidence for the class.

$$s = \sum_j w_j x_j + b = w.x + b = w^T x + b$$

Note that nothing forces s in the above equation to be a legal probability that lies between 0 and 1. Since weights are real-valued, the output might even be negative; s ranges from 1∞ to ∞. To create a probability, we'll pass s through the sigmoid function, $\sigma(s)$. The sigmoid function (named because it looks like an S) is also called the logistic function and gives logistic regression its name. The sigmoid has the following equation

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

, logistic function shown graphically in Fig 1 (a). The figure shows the behavior of a logistic function, its $3D$ visualization, and the $3D$ visualization of non-linear decision boundaries.

1. The sigmoid function is a mathematical function that maps the predicted values to probabilities.
2. It maps any real value into another value within 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the S form.

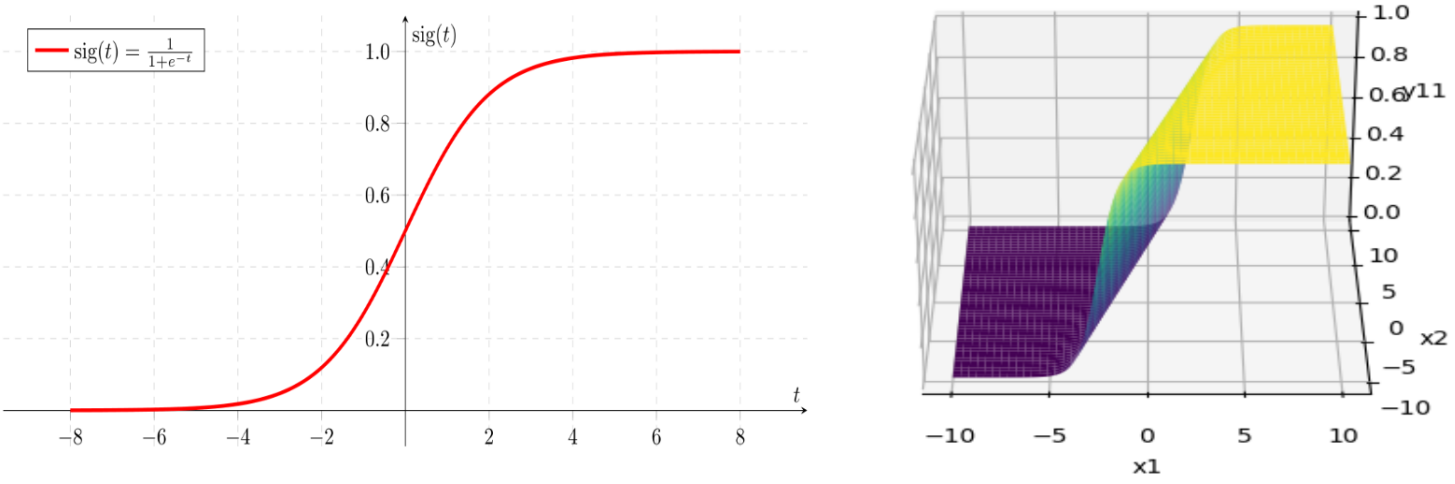

Figure 1: (a) The logistic function. b) The 3D visualization of a logistic function.

3. The S-form curve is called the Sigmoid function or the logistic function.

4. In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Values above the threshold value tend to 1, and those below the threshold value are 0.

3 Logistic Regression

The core of the linear model is the signal $s = w^T x$ that combines the input variables linearly. In linear regression, the signal itself is taken as the output, which is appropriate if you are trying to predict a real response that could be unbounded. In linear classification, the signal is thresholded at zero to produce a $pm1$ output appropriate for binary decisions. A third possibility, which has wide application in practice, is to output a probability between 0 and 1. This model is called logistic regression. It has similarities to both previous models, as the output is real (like regression) but bounded (like classification).

3.1 Predicting a Probability

Linear classification uses a hard threshold on the signal $s = w^T x$,

$$h(x) = sign(w^T x)$$

while linear regression uses no such threshold at all

$$h(x) = w^T x$$

For logistic regression, we need something in between these two cases that smoothly restricts the output to the probability range $[0, 1]$. One such choice that accomplishes this goal is the

logistic regression model,

$$h(x) = \sigma(w^T x)$$

where σ is the so-called logistic function $\sigma(s) = \frac{e^s}{1+e^s}$ whose output is between 0 and 1. The output can be interpreted as a probability for a binary event (heart attack or no heart attack, spam or no spam, digit '1' versus digit '4', etc.). Linear classification also deals with a binary event. Still, the difference is that the classification in logistic regression is allowed to be uncertain, with intermediate values between 0 and 1 reflecting this uncertainty. The logistic function *sigma* is referred to as a soft threshold, in contrast to the hard threshold in classification. It is also called a sigmoid because its shape resembles a flattened-out 's' as shown in Figure 1.

The specific formula $\sigma(s)$ will allow us to define an error measure for learning with analytical and computational advantages. Let us first look at the target that logistic regression is trying to learn. The target is the probability of a patient being at risk for heart attack, which depends on the input x (the patient's characteristics). Formally, we are trying to learn the target function

$$f(x) = P[y = +1|x]$$

The data does not give us the value of f explicitly. Rather, it gives us samples generated by this probability, e.g., patients who had heart attacks and patients who didn't. Therefore, the data is infact generated by a noisy target $P(y|x)$,

$$P(y|x) = \begin{cases} f(x) & \text{for y} = +1; \\ 1 - f(x) & \text{for y} = -1. \end{cases}$$

To learn from such data, we need to define a proper error measure that gauges how close a given hypothesis h is to f in terms fo these noisy ± 1 examples.

3.2 Error Measure

The standard error measure $e(h(x), y)$ used in logistic regression is based on the notion of *likelihood*; how 'likely' is it that we would get this output y from the input x if the target distribution $P(y|x)$ was indeed captured by our hypothesis $h(x)$? Based on Equation 3.1, that likelihood would be

$$P(y|x) = \begin{cases} h(x) & \text{for y} = +1; \\ 1 - h(x) & \text{for y} = -1. \end{cases}$$

We substitute for $h(x)$ by its value $\sigma(w^T x)$, and use the fact that $1 - \sigma(s) = \sigma(-s)$ (easy to verify) to get

$$P(y|x) = \sigma(yw^T x)$$

. One of our reasons for choosing the mathematical form $\sigma(s) = \frac{e^s}{1+e^s}$ is that it leads to this simple expression for $P(y|x)$.

Since the data points $(x_1, y_1), \cdots, (x_N, y_N)$ are independently generated, the probability of getting all the y_n's in the data set from the corresponding x_n's would be the product

$$\prod_{n=1}^{N} P(y_n|x_n)$$

The method of *maximum likelihood* selects the hypothesis h, which maximizes this probability. We can equivalently minimize a more convenient quantity,

$$-\frac{1}{N}\ln(\prod_{n=1}^{N} P(y_n|x_n)) = \frac{1}{N}\sum_{n=1}^{N}\ln(\frac{1}{P(y_n|x_n)})$$

Since '$-\frac{1}{N}\ln(.)$' is a monotonically decreasing function. Substituting with Equation 3.2, we would be minimizing

$$\frac{1}{N}\sum_{n=1}^{N}\ln(\frac{1}{\sigma(y_n w^T x_n)})$$

with respect to the weight vector w. The fact that we are *minimizing* this quantity allows us to treat it as an 'error measure'. Substituting the functional form for $\sigma(y_n w^T x_n)$ produces the in-sample error (training error) for logistic regression,

$$E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}\ln(1 + e^{(-y_n w^T x_n)})$$

The implied pointwise error measure is $e(h(x_n), y_n) = \ln(1 + e^{(-y_n w^T x_n)})$. Notice that this error measure is small when $y_n w^T x_n$ is large and positive, implying that $sign(w^T x_n) = y_n$z. Therefore, as our intuition would expect, the error measure encourages w to 'classify' each x_n correctly.

Cross-entropy error measure

More generally if we are learning from ± 1 data to predict a noisy target $P(y|x)$ with candidate hypothesis h, the maximum likelihood measure reduces to the task of finding h that minimizes

$$E_{in}(w) = \sum_{n=1}^{N}[[y_n = +1]]\ln\frac{1}{h(x_n)} + [[y_n = -1]]\ln\frac{1}{1-h(x_n)}$$

. For the case $h(x) = \sigma(w^T x)$, minimizing the in-sample error is equivalent to minimizing the one in Equation 3.2. For two probability distributions $\{p, 1-p\}$ and $\{q, 1-q\}$ with binary cross outcomes, the cross entropy (from information theory) is

$$p\log\frac{1}{q} + (1-p)\log\frac{1}{1-q}$$

The insample error corresponds to a cross-entropy error measure on the data point (x_n, y_n), with $p = [[y_n = +1]]$ and $q = h(x_n)$.

To train logistic regression, we will take an approach similar to linear regression in that we will try to set the gradient $\nabla E_{in}(w) = 0$. For logistic regression we can easily prove that

$$\nabla E_{in}(w) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n x_n}{1 + e^{y_n w^T x_n}} = \frac{1}{N}\sum_{n=1}^{N} -y_n x_n \sigma(-y_n w^T x_n)$$

It can be seen that a 'missclassified' example contributes more to that gradient that a correctly classified one. Unfortunately, unlike the case of linear regression, the mathematical form of the gradient of E_{in} for logistic regression is not easy to manipulate, so an analytical solution is not feasible. Instead of analytically setting the gradient to zero, we will *iteratively* set it to zero. To do so, we will introduce *gradient descent*. Gradient descent is a very general algorithm that can be used to train many other learning models with smooth error measures. For logistic regression, gradient descent has particularly nice properties.

3.3 Gradient descent

Gradient descent is a general technique for minimizing a twice-differentiable function, such as $E_{in}(w)$ in logistic regression. A useful physical analogy of gradient descent is a ball rolling down a hilly surface. If the ball is placed on a hill, it will roll down, resting at the bottom of the valley. The same basic idea underlies gradient descent. $E_{in}(w)$ is a 'surface' in a high-dimensional space. At step 0, we start somewhere on this surface, at $w(0)$, and try to roll down this surface, thereby decreasing E_{in}. You immediately notice from the physical analogy that the ball will not necessarily come to rest in the lowest valley of the entire surface. Depending on where you start the ball rolling, you will end up at the bottom of one of the valleys - a *local minimum*. In general, the same applies to gradient descent. Depending on your starting weights, the descent path will take you to a local minimum in the error surface.

A particular advantage of logistic regression with the cross-entropy error is that the picture looks much nicer. There is only one valley! So, it does not matter where you start your ball rolling, and it will always roll down to the same (unique) *global minimum*. This is a consequence of the fact that $E_{in}(w)$ is a *convex* function of w, a mathematical property that implies a single 'valley' as shown in Figure 2.

This means that gradient descent will not be trapped in local minima when minimizing such convex error measures.

Let's now determine how to 'roll' down the E_{in}-surface. We want to step toward the steepest descent to gain the biggest bang for our buck. Suppose we take a small step of size η (learning

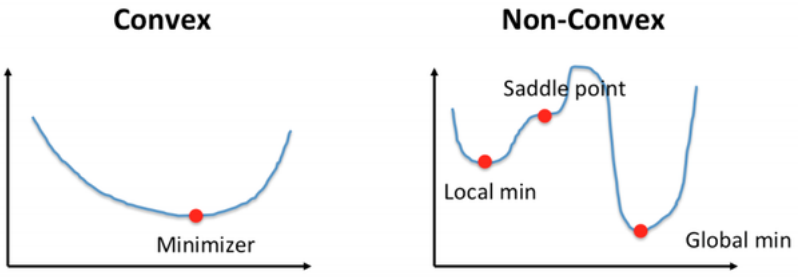

Figure 2: Convex and non-convex functions.

rate) in the direction of a unit vector $\hat{v}$. The new weights are $w(0)+\eta\hat{v}$. Since η is small, using the Taylor expansion to first order, we compute the change in E_{in} as

$$\begin{aligned} \triangle E_{in} &= E_{in}(w(0)+\eta\hat{v}) - E_{in}(w(0)) && (1)\\ &= \eta\nabla E_{in}(w(0))^T\hat{v} + O(\eta^2) && (2)\\ &\geq -\eta||\nabla E_{in}(w(0))|| && (3) \end{aligned}$$

where we have ignored the small term $O(\eta^2)$. Since $\hat{v}$ is a unit vector, equality holds if and only if

$$\hat{v} = -\frac{\nabla E_{in}(w(0))}{||\nabla E_{in}(w(0))||}$$

This direction specified by $\hat{v}$ leads to the largest decrease in E_{in} for a given step size η. A fixed step size (if it is too small) is inefficient when you are far from the local minimum. On the other hand, too large a step size when you are close to the minimum leads to bouncing around, possibly even increasing E_{in}. Ideally, we would like to have large steps when far from the minimum to get in the right ballpark quickly and then small (more careful) steps when close to the minimum. A simple heuristic can accomplish this: far from the minimum, the gradient norm is typically large, and close to the minimum, it is small. Thus we could set $\eta_t = \eta||\nabla E_{in}||$ to obtain the desired behavior for the variable step size; choosing the step size proportional to the norm of the gradient will also conveniently cancel the normalizing the unit vector $\hat{v}$, leading to the fixed learning rate gradient descent algorithm for minimizing E_{-n} (with redefined η). The algorithm is shown below in Algorithm 1.

3.4 Initialization and termination

We have two more loose ends to tie: the first is choosing $w(0)$, the initial weights, and the second is setting the criteria for 'not converged' or 'until it is time to stop.' In some cases, such as logistic regression, initializing the weights $w(0)$ as zeros works well. However, in general, is is safer to initialize the weights randomly to avoid getting stuck on a perfectly symmetric hilltop. Choosing each weight independently from a Normal distribution with zero mean and small variance usually works well in practice.

How do we decide when to stop? Termination is a non-trivial topic in optimization. One simple approach is to set an upper bound on the number of iterations, where the upper bound

Algorithm 1 Logistic regression algorithm

Require: $n \geq 0$
Ensure: $y = f(x^n)$
  $X \leftarrow x$
  $N \leftarrow n$
  $Y \leftarrow y$
  while (*not converged*) **do**
    Compute the gradient
    $g_t = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n x_n}{1+e^{y_n w^T(t) x_n}}$
    Set the direction to move, $v_t = -g_t$
    Update the weights: $w(t+1) = w(t) + \eta v_t$
    Iterate to the next step until it is time to stop.
  end while
  Return the final weights w

is typically in the thousands, depending on our training time. The problem with this is that there is no guarantee on the quality of the final weights.
Another plausible approach is based on the gradient being zero at any minimum. A natural termination criterion would be to stop once $||g_t||$ drops below a certain threshold. Eventually, this must happen, but we do not know when. For logistic regression, a combination of the two conditions (setting a large upper bound for the number of iterations and a small lower bound for the gradient size) usually works well in practice.

4 Logistics

The Department of Computer Science and Engineering will organize the competition at the Departmental Labs as a two-stage event. The competition is open to all. The participants will have to register their team and its members at the time of registration.

1. In the first stage (last week of October 2023), the participants will register and get acquainted with the kind of training and testing dataset they might get. Build their model for multinomial/multivariable logistic regression at different complexity levels. They will be evaluated on their logistic regression model performance based on training/ test error and final hypothesis.

2. The shortlisted candidates will be eligible to participate in the final competition (last week of November 2023) - stage two, where they will be evaluated on their active $3D$ visualization of non-linear decision boundaries on a 360 view.

The evaluation committee is as shown in Table 1. Figure 3 shows a simple 3D visualization of a non-linear decision boundary.

S.No	Name
1.	Dr. Namita Mittal
2.	Dr. Neeta Nain
3.	Dr. Mahipal Jadeja
4.	Dr. Satyendra Singh Chouhan
5.	Dr. Deepak Ranjan Nayak
6.	Dr. Smita Naval
7.	Dr. Ashish Kumar Tripathi

Table 1: Organising team

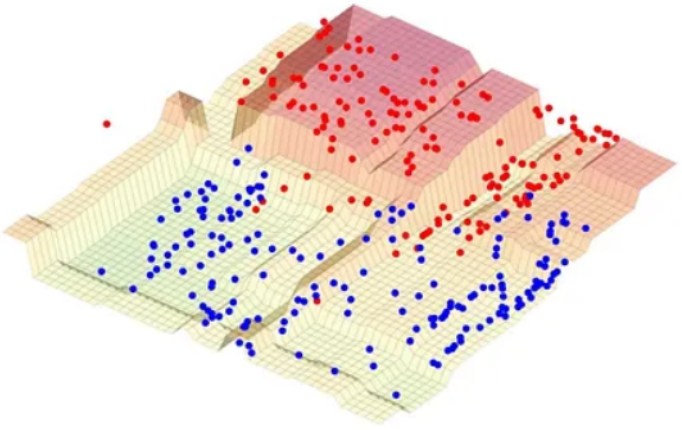

Figure 3: The 3D visualization of non-linear decision boundary.

Evaluation method

The evaluation method for each stage will be:

1. **Initial Screening.** Review the code submissions to ensure they meet the competition's requirements and adhere to the guidelines.
2. **Code Review.** Each submission will undergo a detailed code review by the judges. Judges will assess code quality, functionality, modularity, efficiency, and scalability. Code comments and documentation will also be evaluated.
3. **Model Evaluation.** The models will be tested on a predefined dataset or test cases. Performance metrics like accuracy, completeness, and robustness will be assessed.
4. **Innovation Assessment.** Judges will evaluate the novelty and creativity of each submission based on their understanding of the problem statement and solution.
5. **Scalability and Presentation.** Scalability and presentation aspects will be reviewed and scored accordingly.

6. **Final Scoring and Ranking.** Scores from all evaluation criteria[1] will be combined to calculate a final score for each submission. Submissions will be ranked based on their final scores, and winners will be determined accordingly.

7. **Prizes and Feedback.** Prizes or recognition will be awarded to the top-performing submissions. Constructive feedback will be provided to participants to help them improve their skills.

8. **Public Voting (Optional).** In some cases, an optional public vote or peer evaluation process may be included as an additional factor in determining winners. By following these evaluation criteria and methods, your institute's Deep Learning coding competition can fairly assess participants' coding skills, creativity, and problem-solving abilities while promoting good coding practices and innovation.

The evaluation parameters

1. To evaluate the classification task on linear and non-linearly separable data. To visualize in $3D$ the behavior of the sigmoid function as the training proceeds and the convergence to the final hypothesis.

2. To observe the non-linear relationship between the data and the class, its run time complexity, model complexity, training error, and test error through exhaustive experiments.

3. To visualize underfitting and overfitting. To visualize the Bias and Variance tradeoff. And to see the impact of regularized logistic regression through cross-validation.

4. We also intend to examine the real-time graphical output of the non-linear class boundaries as the data and the classes change dynamically.

Prizes

There are three prizes for the competition (to be announced later). The winner will be provided certificates from Dept. CSE MNIT Jaipur.

Implementation details

The participants can make a team (maximum of three members). The participants must register the team name and members at registration.

The programming language to be used is Python. You are allowed to use the following opensource libraries:

1. pandas

2. numpy

3. matplotlib

4. seaborn

[1]It will be decided and informed to participants during the competition

References

1. Learning from data, Yaser S. Abu-Mostafa, Malik Magdon Ismail, Hsuan-Tien Lin, AMLbook.com

2. A First course in machine learning, Rogers, Girolami, CRC Press

3. Machine learning, Stephen Marsland, CRC Press

4. An introduction to machine learning, Miroslav Kubat, Springer

---

A Testing machine learning algorithms

The purpose of learning is to get better at predicting the outputs, be they class labels or continuous regression values. The only way to know how successfully the algorithm has learnt is to compare the predictions with known target labels, which is how the training is done in supervised learning. This suggests that one thing you can do is to look at the error that the algorithm makes on the training set.

However, we want the algorithms to generalize to examples not seen in the training set, and we obviously can't test this by using the training set. So we need some different data, as a test set, to test it on as well. We use this test set of (input, target) pairs by feeding them into the network and comparing the predicted output with the target. Still, we don't modify the weights or other parameters for them; we use them to decide how well the algorithm has learnt. The only problem with this is that it reduces the amount of data that we have available for training, but this I something that we will have to live with.

A.1 Overfitting

Unfortunately, things are a little bit more complicated than that since we might also want to know how well the algorithm generalizes. There is at least as much danger in overfitting as in underfitting. The number of degrees of variability in most machine learning algorithms is huge - there are many weights, and each varies. This is undoubtedly more variation than there is in the function we are learning, so we need to be careful. If we train too long, then we will overfit the data, which means that we have learnt about the noise and inaccuracies in the data as well as the actual function. Therefore, the model that we learn will be much too complicated and won't be able to generalize.

Figure 4 shows this by plotting the predictions of some algorithms (as the curve). On the left figure, the model is underfitted, as the model complexity is low, so there is a high bias. In the middle figure, the curve fits the overall trend of the data well (it has generalized to the underlying general function). However, the training error would still not be close to zero since

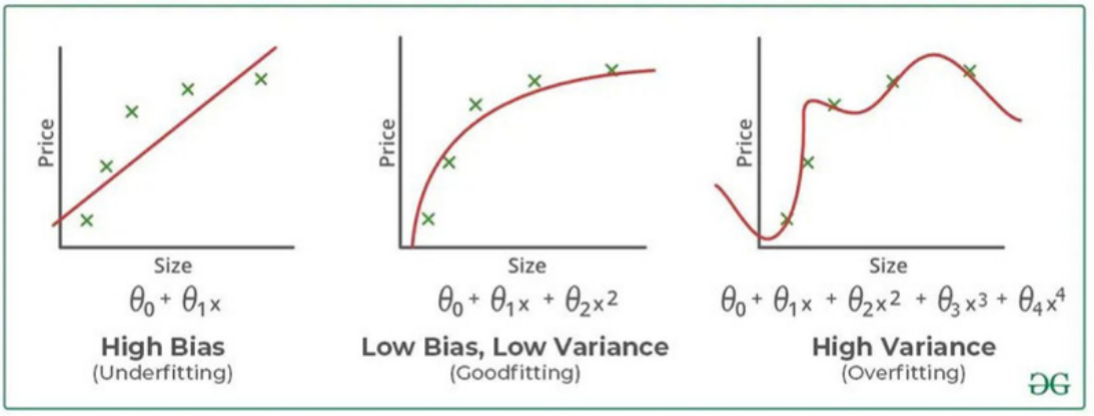

Figure 4: The Bias-variance trade-off. Overfitting and underfitting.

it passes near but not through the training data. As the network continues to learn, it will eventually produce a much more complex model that has a lower training error (close to zero), meaning that it has memorized the training examples, including any noise component of them, so that it has overfitted the training data (rightmost figure).

We want to stop the learning process before the algorithm overfits, so we need to know how well it generalizes at each timestep. We can't use the training data for this because we wouldn't detect overfitting, but we can't use the testing data either because we are saving that for the final tests. So we need a third set of data for this purpose, which is called the validation set because we are using it to validate the learning so far. This is known as cross-validation in statistics. It is part of model selection: choosing the right parameters for the model to generalize as well as possible.

A.2 Traing, testing and validation sets

We now need three sets of data: the training set to train the algorithm, the validation set to track how well it is doing as it learns, and the test set to produce the final results. This is becoming expensive in data, especially since in supervised learning, all data has to have target values attached (and even for supervised learning, the validation and test sets need targets so that you have something to compare to). It is not always easy to get accurate labels (which may be why you want to learn about the data).

Each algorithm will need some reasonable amount of data to learn from (precise needs vary, but the more data the algorithm sees, the more likely it is to have seen examples of each possible input type, although more data also increases the computational time to learn). However, the same argument can be used to argue that the validation and test should also be reasonably large. Generally, the exact proportion of training to testing to validation data is up to you, but it is typically to do something like $50 : 25 : 25$ if you have plenty of data, and $60 : 20 : 20$ if you don't. How you do the splitting can also matter. Many datasets are presented with the first set of data points being in class 1, the next in class 2, and so on. If you pick the first few

points to be the training set, the next the test see, etc., the results will be pretty bad, since the training did not see all the classes. This can be dealt with by randomly reordering the data first or assigning each point randomly to one of the sets.

If you are short of training data if you have a separate validation set, there is a worry that the algorithm won't be sufficiently trained. Performing *leave-some-out*, multi-fold *cross-validation* is possible. The dataset is randomly partitioned into k subsets, and one subset is used as a validation set while the algorithm is trained on all of the others. A different subset is then left out and a new model is trained on that subset, repeating the same process for all subsets. Finally, the model that produced the lowest validation error is tested and used. We have traded off data for computation time, since we have to train k different models instead of just one. In the most extreme case, there is *leave-one-out* cross-validation, where the algorithm is validated on just one piece of data, training on the rest.

B Multiclass classification

Multiclass-Classification Multinomial logistic regression is an extension of logistic regression for multi-class classification. The simplest implementation of Multiclass classification follows the same ideas as the binary classification. As you know, we solve a yes or no problem in binary classification. In multi-class classification, we have more than two classes. We will treat each class as a binary classification problem the way we solved the binary classification. We will train the classifier as one versus the rest and compute the likelihood of each class. The class with the maximum likelihood is the output class.

Logistic regression is designed for two-class problems, modeling the target using a binomial probability distribution function. The class labels are mapped to 1 for the positive class or outcome and 0 for the negative class or outcome. The fit model predicts the probability that an example belongs to class 1.

By default, logistic regression cannot be used for classification tasks with more than two class labels, so-called multi-class classification. Instead, it requires modification to support multi-class classification problems. One common approach for adapting logistic regression to multi-class classification problems is splitting the problem into multiple binary classification problems and fitting a standard logistic regression model on each subproblem. Techniques of this type include one-vs-rest and one-vs-one wrapper models.

An alternate approach involves changing the logistic regression model to directly support the prediction of multiple class labels. Specifically, to predict the probability that an input example belongs to each known class label. The probability distribution that defines multi-class probabilities is called a multinomial probability distribution. Multinomial Logistic Regression is a logistic regression model adapted to learn and predict a multinomial probability distribution. Similarly, we might refer to default or standard logistic regression as Binomial Logistic Regression.

Binomial Logistic Regression: Standard logistic regression that predicts a binomial probability (i.e., for two classes) for each input example. Multinomial Logistic Regression: Modified version of logistic regression that predicts a multinomial probability (i.e., more than two classes) for each input example. The multinomial logistic regression model will be fit using cross-entropy

loss and predict the integer value for each integer encoded class label. *Softmax* extends this idea into a multi-class world. The softmax function is defined as

$$P(y = c|x) = \frac{e^{(x^T w_c)}}{\sum_{k=1}^{K} e^{(x^T w_k)}}$$

Softmax assigns decimal probabilities to each class c in a multi-class (K classes) problem. Those decimal probabilities must add up to. This additional constraint helps training converge more quickly than it otherwise would.