CHEME 5820: Machine Learning for Engineers **Spring 2025**

Lecture 3c: Logistic Regression for Binary Classification Problems

Lecturer: Jeffrey Varner

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

The key concepts covered in this lecture include:

- **Logistic regression** is a statistical method used for binary classification that models the relationship between a dependent categorical variable (label) and one or more independent variables (features) by estimating probabilities through the logistic function.
- **Maximum likelihood estimation (MLE)** is a statistical technique to estimate the parameters of a probability distribution by maximizing the likelihood function, thereby determining the parameter values that make the observed data most probable.
- **Gradient descent** is an optimization algorithm used to minimize a function by iteratively adjusting parameters in the opposite direction of the gradient. Iteration continues until a local minimum of the function is found.
- **Alternatives to gradient descent** include heuristic optimization algorithms such as the Nelder-Mead Simplex Algorithm, Simulated Annealing, Genetic Algorithms, and Particle Swarm Optimization, which can estimate model parameters without relying on the gradient.

1 Introduction

In this lecture, we will introduce logistic regression for binary classification problems. Logistic regression is a statistical method used for binary classification that models the relationship between a dependent categorical variable (label) and one or more independent variables (features) by estimating probabilities through the logistic function. We will start by discussing the logistic model and then we'll explore the maximum likelihood estimation of the model parameters. Unfortunately, the likelihood function of the logistic regression model has no closed-form analytical solution, so we must estimate the model parameters using numerical optimization algorithms. Toward this challenge, we will introduce the gradient descent algorithm, which is a numerical optimization algorithm that minimizes a function by iteratively adjusting the parameters in the opposite direction of the gradient. In addition to gradient descent, we will also discuss alternative heuristic optimization algorithms that can be used to estimate the model parameters without relying on the gradient. You might consider these alternative methods when the objective function is non-convex, non-differentiable, or has many local minima. Let's start by discussing the logistic regression model.

2 Logistic Regression

Logistic regression is a statistical method used for binary classification problems, where the dependent variable (label) is a binary categorical variable (e.g., ± 1, etc), and the independent variables (features) are continuous or categorical variables. Unlike the Perceptron model, which outputs the class label directly, logistic regression models the probability that a given input belongs to a particular class based on the input features. The logistic regression model estimates the probability that a given feature vector belongs to a particular class based on the input features. In particular, logistic regression is a discriminative model, which means it directly models the conditional probability of the label given the features, i.e., $P(y|\mathbf{x})$. This is in contrast to generative models, e.g., Naive Bayes, which we'll explore later, which model the joint probability of the features and the label, i.e., $P(y, \mathbf{x}) = P(\mathbf{x}|y) \cdot P(y)$. The logistic regression model uses the logistic function to model the probability of the binary label $y \in \{-1, +1\}$ given the (augmented) feature vector $\hat{\mathbf{x}} = (x_1, x_2, \dots, x_m, 1)$:

$$P_\theta(y|\hat{\mathbf{x}}) = \frac{1}{1 + e^{-y \cdot (\hat{\mathbf{x}}^\top \theta)}} \tag{1}$$

where $\theta \in \mathbb{R}^p$ $(p = m + 1)$ is an (unknown) parameter vector (that we need to estimate somehow), and e is the base of the natural logarithm. The logistic function is a sigmoid function that maps the input, i.e., $-y \cdot \left(\hat{\mathbf{x}}^\top \theta\right)$ to the range $[0, 1]$, which is suitable for modeling probabilities. The logistic model defines ("parameterizes") a probability distribution $P_\theta(y|\hat{\mathbf{x}}) : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ which is given by the logistic function:

$$\begin{aligned} P_\theta(y = 1|\mathbf{x}) &= \sigma(y \left(\hat{\mathbf{x}}^\top \theta\right)) \\ P_\theta(y = -1|\mathbf{x}) &= 1 - \sigma(y \left(\hat{\mathbf{x}}^\top \theta\right)). \end{aligned}$$

where $\sigma(z) = 1/(1 + e^{-z})$. Clearly this sums to one over y for all $\mathbf{x}$. The logistic regression model predicts the label $y \in \{-1, +1\}$ for a given feature vector $\hat{\mathbf{x}}$ by comparing the probability $P_\theta(y|\mathbf{x})$ to a threshold, e.g., 0.5. The model predicts the positive class if the probability exceeds the threshold $(y = 1)$. Otherwise, it predicts the negative class $(y = -1)$. The logistic regression model is trained by estimating the parameters θ that maximize the likelihood of the observed labels given the features. The next section will discuss the maximum likelihood estimation of the logistic regression model parameters.

2.1 Maximum Likelihood Estimation (MLE)

Maximum likelihood estimation (MLE) is a technique to estimate the parameters of a probability distribution by maximizing the likelihood function. In logistic regression, MLE estimates the parameters of the logistic regression model that make the observed label conditioned on the features the most probable. Given a set of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^m$ is an m-dimensional feature vector and $y_i \in \mathbb{R}$ is the binary (scalar) label, the likelihood function is defined as:

$$L(\theta) = \prod_{i=1}^{n} P_\theta(y_i|\hat{\mathbf{x}}_i) \tag{2}$$

where $P_\theta(y_i|\hat{\mathbf{x}}_i)$ is the probability of observing the label y_i given the feature vector $\hat{\mathbf{x}}_i$ and the model parameters θ. It's hard to maximize the likelihood function directly (because of the product), so we take the logarithm of the likelihood function to simplify the optimization:

$$\begin{aligned} \log L(\theta) &= \sum_{i=1}^{n} \log P_\theta(y_i|\mathbf{x}_i) \quad (\text{substituting}\, P_\theta(y_i|\mathbf{x}_i)) \\ &= -\sum_{i=1}^{n} \log \left(1 + e^{-y_i \cdot \left(\mathbf{x}_i^\top \theta\right)}\right) \end{aligned}$$

The maximum likelihood estimation (MLE) of the logistic regression model parameters $\theta^{\star}$ is obtained by maximizing the log-likelihood function $\log L(\theta)$:

$$\theta^{\star} = \arg\max_{\theta} \left[-\sum_{i=1}^{n} \log\left(1 + e^{-y_i \cdot \left(\hat{\mathbf{x}}_i^{\top}\theta\right)}\right)\right] \tag{3}$$

or alternatively by minimizing the negative log-likelihood function:

$$\mathcal{L}(\theta) = -\log L(\theta) = \sum_{i=1}^{n} \log\left(1 + e^{-y_i \cdot \left(\hat{\mathbf{x}}_i^{\top}\theta\right)}\right) \tag{4}$$

Unfortunately, whichever perspective we take, the optimization problem has no closed-form analytical solution. Thus, we must estimate the model parameters using some numerical algorithm, such as gradient descent algorithm (or one of many other approaches). These approaches iteratively update the parameters θ to maximize the log-likelihood function until a stopping criterion is met, e.g., the change in the parameters is below a threshold, or the maximum number of iterations is reached.

2.2 Gradient Descent

Gradient descent is a numerical seach algorithm that minimizes a function by iteratively adjusting the parameters in the opposite direction of the gradient. Suppose there exists an objective function $\mathcal{L}(\theta)$ that we want to minimize with respect to the parameter θ, i.e., the negative log-likelihood function. In general, an objective function measures the difference between the predicted values and the observed values in some way, e.g., the mean squared error (MSE), the cross-entropy loss, or the negative log-likelihood. In logistic regression, the objective function is the negative log-likelihood function, which measures the difference between the predicted probabilities and the observed labels. However, whatever form the objective function takes, we assume that it is differentiable and that we can compute the gradient, i.e., $\nabla l(\theta)$ for the negative log-likelihood function, which points in the direction of the steepest increase of the function. This gives us a way to update the parameters to minimize the objective function using the update rule:

$$\theta_{k+1} = \theta_k - \alpha(k) \cdot \nabla_{\theta}\mathcal{L}(\theta_k) \quad \text{where } k = 0, 1, 2, \dots$$

The (hyper) parameter $\alpha(k) > 0$ is the learning rate (which can be a function of the iteration count k), and $\nabla_{\theta}\mathcal{L}(\theta)$ is the gradient of the negative log-likelihood function with respect to the parameters. We iterate until a stopping criterion is met, i.e., $||\theta_{k+1} - \theta_k|| \leq \epsilon$, the maximum number of iterations is reached, or some other stopping criterion is met. Pusedo-code for a naive gradient descent algorithm (for a fixed learning rate) is shown in Algorithm 1.

Algorithm 1 Naive Gradient Descent for Negative Log-Likelihood $\mathcal{L}(\theta)$

```
1: Input: Initial parameters $\theta_0$, learning rate $\alpha$, stopping criterion $\epsilon$, maximum iterations $N$
2: Output: Optimal parameter estimates $\theta$
3: Initialize $\theta \leftarrow \theta_0$                        ▷ Initialize parameters to the initial guess
4: $k \leftarrow 0$                                                ▷ Initialize iteration counter
5: while $k \leq N$ or $\|\theta_{k+1} - \theta_k\| \leq \epsilon$ do
6:     $\mathbf{d} \leftarrow \nabla\mathcal{L}(\theta_k)$        ▷ Compute gradient using analytical or numerical method, evaluate at $\theta_k$
7:     $\theta_{k+1} \leftarrow \theta_k - \alpha \cdot \mathbf{d}$   ▷ Update parameters using the gradient direction $\mathbf{d}$
8:     $k \leftarrow k + 1$
9: end while
10: return $\theta$
```

Choose the learning Rate $\alpha(k)$

The choice of the learning rate α is crucial, as a too-large value can cause the algorithm to diverge, while a too-small value can slow down convergence. Choosing an appropriate learning rate is often challenging and may require tuning through hyperparameter optimization techniques. In practice, we may use adaptive learning rate methods, such as the Adam optimizer, which adjusts the learning rate based on the gradient's magnitude and the second moment of the gradient (1). Alternatively, Adagrad and RMSprop (unpublished) are other adaptive learning rate methods that can be used to improve the convergence of the gradient descent algorithm (2).

2.3 Alternatives to Gradient Descent

The central issue with gradient descent is that it can be slow to converge, especially when the objective function is non-convex or has many local minima. The choice of the learning rate α is also crucial, as a too-large value can cause the algorithm to diverge, while a too-small value can slow down convergence. Further, the objective function may not be differentiable, or the gradient may be challenging to compute. In these cases, alternative heuristic optimization algorithms can be used to estimate the model parameters. The central theme of these approaches to is to directly evaluate the objective function at different points in the parameter space to find the optimal solution. New search points are generated based randomly or with some heuristic, and the objective function is evaluated at these points to determine the next search direction. Let's walk through some of the alternatives to gradient descent.

Nelder-Mead Simplex Algorithm

The Nelder-Mead algorithm (3), also known as the simplex algorithm, is a direct search optimization algorithm that does not require the gradient of the objective function. It works by maintaining a simplex (a geometric shape with $n + 1$ vertices in an n-dimensional space) that evolves through reflection, expansion, contraction, and shrinkage operations. Thus, the Nelder-Mead algorithm can be used for optimizing non-differentiable or noisy objective functions, making it suitable for a wide range of optimization problems. However, the Nelder-Mead algorithm may struggle with high-dimensional problems or functions with many local minima, as it does not leverage gradient information to guide the search. Pseudo-code for the Nelder-Mead algorithm is shown in Algorithm 2.

Algorithm 2 Nelder-Mead Simplex Algorithm

```
1: Input: Initial simplex vertices $x_0, x_1, \ldots, x_n$, reflection coefficient $\rho$, expansion coefficient $\chi$, contraction
   coefficient $\gamma$, shrinkage coefficient $\sigma$
2: Output: Optimal solution $x^*$
3: Sort vertices such that $f(x_0) \leq f(x_1) \leq \ldots \leq f(x_n)$
4: while stopping criterion not met do
5:     Compute the centroid $x_c$ of the $n$ best vertices
6:     Reflect: $x_r = x_c + \rho(x_c - x_n)$
7:     if $f(x_0) \leq f(x_r) < f(x_{n-1})$ then
8:         Replace $x_n$ with $x_r$
9:     else if $f(x_r) < f(x_0)$ then
10:        Expand: $x_e = x_c + \chi(x_r - x_c)$
11:        if $f(x_e) < f(x_r)$ then
12:            Replace $x_n$ with $x_e$
13:        else
14:            Replace $x_n$ with $x_r$
15:        end if
16:    else
17:        Contract: $x_c = x_c + \gamma(x_n - x_c)$
18:        if $f(x_c) < f(x_n)$ then
19:            Replace $x_n$ with $x_c$
20:        else
21:            Shrink: $x_i = x_0 + \sigma(x_i - x_0)$ for $i = 1, \ldots, n$
22:        end if
23:    end if
24: end while
25: return $x^*$
```

Simulated Annealing

Simulated annealing, originally developed by Kirkpatrick et al (4), is a probabilistic optimization algorithm inspired by the physical process of heating and then slowly cooling (annealing) materials to minimize defects. Simulated annealing works by iteratively exploring the solution space. First a random (candidate) solution is generated, and the objective function is evaluated at this point. The difference in the objective function between the current and candidate solutions is computed, and the candidate solution is accepted if it improves the objective function. Alternatively, the candidate solution may be accepted with a certain probability even if it worsens the objective function. Thus, simulated annealing can escape local minima and explore the solution space more thoroughly than gradient descent, at least when the system is hot. As the number of iterations increases, the algorithm gradually decreases the temperature, i.e., the probability of accepting worse solutions, allowing it to converge to the global optimum. This method effectively solves complex optimization problems with large search spaces, where traditional techniques may struggle to find the global optimum. However, simulated annealing can be computationally expensive and may require tuning of hyperparameters, particularly the annealing schedule, i.e., the decrease is temperature at each iteration which impacts how willing the algorithm is to accept worse solutions to perform well. Pseudo-code for the simulated annealing algorithm is shown in Algorithm 3.

Algorithm 3 Simulated Annealing

```
1: Initialize current solution $x \leftarrow x_0$
2: Initialize temperature $T \leftarrow T_0$
3: Define cooling schedule $T \leftarrow \alpha \cdot T$, where $0 < \alpha < 1$
4: Define maximum iterations $N$
5: Initialize best solution $x_{\text{best}} \leftarrow x$
6: for $i = 1$ to $N$ do
7:     Generate a new candidate solution $x_{\text{new}}$ in the neighborhood of $x$
8:     Compute $\Delta E \leftarrow f(x_{\text{new}}) - f(x)$
9:     if $\Delta E < 0$ then
10:        Accept $x_{\text{new}}$: $x \leftarrow x_{\text{new}}$
11:    else
12:        Accept $x_{\text{new}}$ with probability $P \leftarrow e^{-\Delta E/T}$
13:        Draw a random number $r \in [0, 1]$
14:        if $r < P$ then
15:            $x \leftarrow x_{\text{new}}$
16:        end if
17:    end if
18:    if $f(x) < f(x_{\text{best}})$ then
19:        Update $x_{\text{best}} \leftarrow x$
20:    end if
21:    Update temperature: $T \leftarrow \alpha \cdot T$
22:    if $T$ is below a threshold $T_{\text{min}}$ then
23:        break
24:    end if
25: end for
26: return $x_{\text{best}}$
```

Genetic Algorithms

Genetic algorithms (GAs), popularized by John Holland in the 1970s (5), are adaptive heuristic search techniques inspired by natural selection and genetics principles. They are designed to solve optimization and search problems by iteratively evolving a population of candidate solutions through selection, crossover, and mutation. By mimicking the evolutionary process, GAs aim to improve solution quality over generations, making them particularly effective for complex problems that may be discontinuous, non-differentiable, or highly nonlinear. However, GAs have several hyperparameters that need to be tuned, such as the population size, crossover rate, mutation rate, and the number of generations. Further, GAs can be computationally expensive, especially for large search spaces, and may require significant computational resources to converge to the optimal solution. This will be especially true for complex objective functions that are expensive to evaluate, as the algorithm will need to evaluate the objective function for each candidate solution in the population. Pseudo-code for the genetic algorithm is shown in Algorithm 4.

Algorithm 4 Genetic Algorithm

```
1: Input: Population size $P$, crossover rate $p_c$, mutation rate $p_m$, maximum generations $G$, fitness function $f$
2: Output: Best solution found after $G$ generations (individual with highest fitness)
3: Initialize population $P_0$ randomly
4: Evaluate fitness of each individual in $P_0$
5: $t \leftarrow 0$
6: while $t < G$ do
7:     Select parents from $P_t$ based on fitness
8:     Apply crossover with probability $p_c$ to produce offspring
9:     Apply mutation with probability $p_m$ to offspring
10:    Evaluate fitness of offspring
11:    Form new population $P_{t+1}$ by selecting the best individuals from parents and offspring
12:    $t \leftarrow t + 1$
13: end while
14: Identify the best individual in $P_t$
15: return Best individual
```

Particle Swarm Optimization

Particle Swarm Optimization (PSO), developed by Kennedy and Eberhart in the mid-1990s (6) is an example of a meta-heuristic optimization algorithm inspired by the social behavior of birds and fish, which utilizes a population of candidate solutions, referred to as particles, that move through the search space to find optimal solutions. Each particle adjusts its position based on its own experience and the collective knowledge of the swarm, allowing for efficient exploration and exploitation of the solution space to address complex optimization problems across various fields.

Algorithm 5 Particle Swarm Optimization (PSO)

```
1: Input: Number of particles $N$, maximum iterations $T$, inertia weight $\omega$, cognitive parameter $c_1$, social parameter $c_2$, objective function $f$
2: Output: Best solution found $x_{\text{best}}$
3: Initialize particles' positions $x_i$ and velocities $v_i$ randomly for $i = 1, \dots, N$
4: Evaluate fitness of each particle and initialize personal best positions $p_i \leftarrow x_i$
5: Set global best position $g \leftarrow \arg\min_{p_i} f(p_i)$
6: for $t = 1$ to $T$ do
7:     for each particle $i = 1$ to $N$ do
8:         Update velocity:
               $v_i \leftarrow \omega v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (g - x_i)$
   where $r_1, r_2 \sim U(0, 1)$
9:         Update position:
               $x_i \leftarrow x_i + v_i$
10:        Evaluate fitness $f(x_i)$
11:        if $f(x_i) < f(p_i)$ then
12:            Update personal best: $p_i \leftarrow x_i$
13:        end if
14:        if $f(p_i) < f(g)$ then
15:            Update global best: $g \leftarrow p_i$
16:        end if
17:    end for
18: end for
19: return $g$
```

3 Summary and Conclusions

In this lecture, we introduced logistic regression for binary classification problems. Logistic regression is a powerful tool for binary classification and is widely applied in numerous fields, including healthcare, finance, and marketing. Like the Perceptron model, logistic regression models the relationship between the features and the label, but it estimates the probability that a given input belongs to a particular class using the logistic function. To estimate the parameters that appear in the logistic regression model, we discussed the maximum likelihood estimation (MLE) technique, which maximizes the likelihood of the observed labels given the features. However, unlike our previous discussion of linear regression, the logistic regression model's likelihood function has no closed-form analytical solution. Thus, we must estimate the model parameters using numerical optimization algorithms, such as the gradient descent algorithm. Gradient descent is an optimization algorithm employed to minimize a function by iteratively adjusting parameters in the opposite direction of the gradient. However, we also discussed various other (heuritisc) optimization algorithms that be used to estimate model parameters without relying on the gradient. For instance, simulated annealing, genetic algorithms, and particle swarm optimization are all methods that can be utilized to estimate the model parameters. These methods can be particularly useful when the objective function is non-convex, non-differentiable, or has many local minima.

References

1. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization; 2017. Available from: `https://arxiv.org/abs/1412.6980`.

2. Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. J Mach Learn Res. 2011;12(null):2121–2159.

3. Nelder JA, Mead R. A Simplex Method for Function Minimization. The Computer Journal. 1965;7(4):308–313. doi:10.1093/comjnl/7.4.308.

4. Kirkpatrick S, Gelatt CD Jr, Vecchi MP. Optimization by simulated annealing. Science. 1983;220(4598):671–80. doi:10.1126/science.220.4598.671.

5. Holland JH. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Ann Arbor: University of Michigan Press; 1975.

6. Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks. vol. 4; 1995. p. 1942–1948 vol.4.