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

Lecture 3a: Linear Regression, Perceptron and Binary Classification

Lecturer: Jeffrey Varner

- **Linear regression models**: A class of models used in machine learning for regression tasks, i.e., predicting a continuous output variable from one or more continuous or discrete input features. Linear regression models assume that the output variable is a linear combination of the input features.
- **Binary classification**: The problem of classifying data points into one of two classes. Binary classification is a type of supervised machine learning task that involves categorizing data points into one of two distinct classes based on their features. These features can be continuous or discrete, and the classes can be represented as binary labels, e.g., $\{-1, 1\}$ or $\{0, 1\}$.
- **Perceptron**: The perceptron algorithm for binary classification problems is a linear classifier that separates two classes of data points. The perceptron algorithm is an iterative algorithm that incrementally updates the weights of the linear classifier to minimize the classification error. The perceptron algorithm is guaranteed to converge to a solution with no mistakes in a finite number of iterations if the data set is linearly separable. However, if the data set is not linearly separable, the perceptron algorithm will not converge to a perfect solution, but rather to a solution with some classification errors.

1 Introduction

In this lecture, we will introduce the perceptron algorithm for binary classification problems. The perceptron is a simple linear classifier that can be used to separate two classes of data points. The perceptron algorithm is a simple iterative algorithm that incrementally updates the weights of the linear classifier to minimize the classification error. We will first introduce the perceptron algorithm and then discuss its convergence properties, i.e., when the algorithm converges to a solution and when it does not. The key concepts covered in this lecture include:

2 Optimizer

The material in this section was heavily inspired by the Applied Machine Learning (Cornell CS5785, Fall 2024) course notes. Before we present our first supervised learning algorithm, namely linear regression, let's do a quick calculus review as it relates to the supervised learning problem.

2.1 Calcular Review

A key part of a supervised learning algorithm is the `optimizer`, which takes an objective J (also called a loss function) and a model $\mathcal{M}$. In general, an objective function J 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. The optimizer outputs a model $f \in \mathcal{M}$ with the smallest value of the

objective J, i.e., $\min_{f \in \mathcal{M}} J(f)$. Intuitively, this is the function that best describes the (training) dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, ..., n\}$. But what is this magical `optimizer` creature?

The key tool from calculus that we will use to develop the `optimizer` is the derivative and its extensions. Suppose we have a a univariate function $f_\theta : \mathbb{R} \to \mathbb{R}$ then the derivative $df(\theta_0)/d\theta$ is the instantaneous rate of change of the function $f(\theta)$ with respect to its parameter θ at the point θ_0. The partial derivative $\partial f(\theta)/\partial \theta_j$ of a multivariate function $f : \mathbb{R}^d \to \mathbb{R}$ is the derivative of f with respect to θ_j while all the other dimensions θ_k for $k \neq j$ are held constant. Finally, the gradient ∇f is the vector of all the partial derivatives:

$$\nabla f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_d} \end{bmatrix} \tag{1}$$

The j-th entry of the gradient is the partial derivative $\partial f(\theta)/\partial \theta_j$ of f with respect to the j-th component of θ. We use the gradient to help us find fruitful directions to minimize our objective function J.

2.2 Gradient descent

Suppose there exists an objective function $J(\theta)$ that we want to minimize with respect to the parameters θ. Gradient descent is a numerical seach algorithm that minimizes an objective function by iteratively adjusting the parameters in the opposite direction of the gradient:

$$\theta_{k+1} = \theta_k - \alpha(k) \cdot \nabla J(\theta_k) \quad \text{where } k = 0, 1, 2, \ldots$$

where k denotes the iteration index, and $\alpha(k) > 0$ is a hyperparamter called the learning rate, which can be a function of the iteration count k. 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 objective $J(\theta)$

Input: Initial parameters θ_0, learning rate α, stopping criterion ϵ, maximum iterations N
Output: Optimal parameter estimates θ
Initialize $\theta \leftarrow \theta_0$ ▷ Initialize parameters to the initial guess
$k \leftarrow 0$ ▷ Initialize iteration counter
while $k \leq N$ **or** $\|\theta_{k+1} - \theta_k\| \leq \epsilon$ **do**
    $\mathbf{d} \leftarrow \nabla J(\theta_k)$ ▷ Compute gradient using analytical or numerical method, evaluate at θ_k
    $\theta_{k+1} \leftarrow \theta_k - \alpha \cdot \mathbf{d}$ ▷ Update parameters using the gradient direction $\mathbf{d}$
    $k \leftarrow k + 1$
end while
return θ

3 Linear Regression Models

Linear regression models are a class of models used in machine learning for regression tasks, i.e., predicting a continuous output variable from one or more continous or discrete input features. Linear regression models assume that the output variable is a linear combination of the input features, i.e., the output variable is a linear function of the input features. Linear in this context is a misnomer in the sense that the features are

not necessarily linear, but the model is linear in the parameters and the features. Suppose we have a data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ with m examples, where each example where $\mathbf{x}_i \in \mathbb{R}^n$ is a feature vector and $y_i \in \mathbb{R}$ is the output variable. The linear regression model predicts the output variable y_i for feature vector $\mathbf{x}_i$ using the linear function:

$$y_i = \hat{\mathbf{x}}_i^T \cdot \beta + \epsilon_i$$

where we have augmented the feature vector $\mathbf{x}_i$ with a bias term, i.e., $\mathbf{x}_i^T = \left(x_1^{(i)}, \ldots, x_n^{(i)}, 1\right)$, $\beta = (w_1, \ldots, w_n, b)$ is a column vector of (unknown) parameters $w_j \in \mathbb{R}$ corresponding to the importance (weight) of feature j and a bias parameter $b \in \mathbb{R}$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is a noise term, typically assumed to be a Normal distribution with mean zero and variance σ^2. Depending upon the shape of the data and various other problem constraints, there are analytical solutions to for linear regression parameter vecrtoe β, e.g., the normal equations, or iterative solutions, e.g., gradient descent can be used to estimate the parameters β for other linear regression problems.

3.1 Overdetemined Linear Regression models

If the number of examples m is greater than the number of features n, the linear regression model is said to be overdetermined. In other words, there are more examples than features. Regularized linear regression models incorporate penalty terms to constrain the size of the coefficient estimates, thereby reducing overfitting and enhancing the model's generalizability to new data. Consider an overdetermined data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, i.e., the case where $m > n$ (more examples than unknown parameters). A regularized least squares estimate of the unknown parameters β for an overdetermined system will minimize a loss (objective) function of the form:

$$\hat{\beta}_\lambda = \arg\min_\beta || \ \mathbf{y} - \mathbf{X} \cdot \beta \ ||_2^2 + \lambda \cdot || \ \beta \ ||_2^2$$

where $||\star||_2^2$ is the square of the $l2$-vector norm, $\lambda \geq 0$ denotes a regularization parameter, and $\hat{\beta}_\lambda$ denotes the estimated parameter vector. The parameters $\hat{\beta}_\lambda$ that minimize the $||\star||_2^2$ loss plus penalty for overdetermined data matrix $\mathbf{X}$ are given by:

$$\hat{\beta}_\lambda = \left(\mathbf{X}^T\mathbf{X} + \lambda \cdot \mathbf{I}\right)^{-1} \mathbf{X}^T\mathbf{y} - \left(\mathbf{X}^T\mathbf{X} + \lambda \cdot \mathbf{I}\right)^{-1} \mathbf{X}^T\epsilon$$

The matrix $\mathbf{X}^T\mathbf{X} + \lambda \cdot \mathbf{I}$ is the `regularized normal matrix`, while $\mathbf{X}^T\mathbf{y}$ is the `moment vector`. The inverse $\left(\mathbf{X}^T\mathbf{X} + \lambda \cdot \mathbf{I}\right)^{-1}$ must exist to obtain the estimated parameter vector $\hat{\beta}_\lambda$.

3.2 Underdetermined Linear Regression models

Assume the data matrix $\mathbf{X}$ is `underdetermined`, i.e., $m < n$ (more columns than rows), and the error vector $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \cdot \mathbf{I})$. Then, an ordinary least squares estimate of the unknown parameters is the *smallest* parameter vector β that satisfies the original equations:

$$\begin{aligned} &\text{minimize} && || \, \beta \, || \\ &\text{subject to} && \mathbf{X} \cdot \beta = \mathbf{y} \end{aligned}$$

The least-norm problem has an analytical estimate for the unknown parameter vector $\hat{\beta}$ given by:

$$\hat{\beta} = \mathbf{X}^T \left(\mathbf{X}\mathbf{X}^T\right)^{-1} \cdot \mathbf{y} - \mathbf{X}^T \left(\mathbf{X}\mathbf{X}^T\right)^{-1} \cdot \epsilon$$

where inverse $\left(\mathbf{X}\mathbf{X}^T\right)^{-1}$ must exist to obtain the estimated model parameter vectors $\hat{\beta}$.

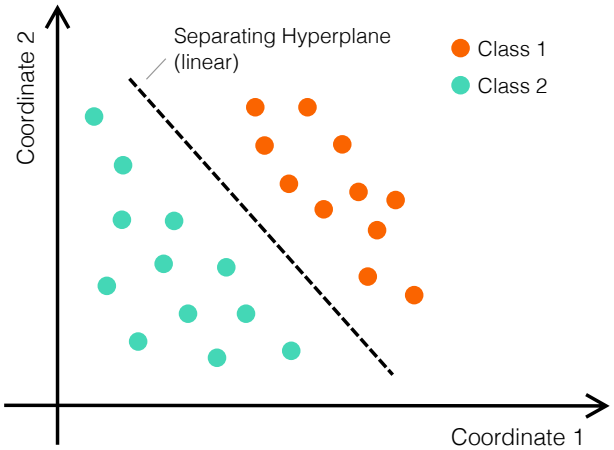

Figure 1: Schematic of a binary classification tasks with linearly separable data. Above the hyperplane are positive examples, while below the hyperplane are negative examples.

4 The Perceptron and Binary Classification

The Perceptron (1) is a simple yet powerful algorithm used in machine learning for binary classification tasks. The Perceptron (Rosenblatt, 1957) takes the (scalar) output of a linear regression model $y_i \in \mathbb{R}$ and transforms it, using a transform function $\sigma(\star) = \text{sign}(\star)$, into a discrete value representing a category, e.g., $\sigma : \mathbb{R} \to \{-1, 1\}$ in the binary classification case. Suppose there exists a data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ with m *labeled* examples, where each example $1, 2, \ldots, m$ has been labeled by an expert, i.e., a human to be in a category $\hat{y}_i \in \{-1, 1\}$, given the feature vector $\mathbf{x}_i \in \mathbb{R}^n$. The Perceptron *incrementally* learns a linear decision boundary between two classes of possible objects (binary classification) in $\mathcal{D}$ by repeatedly processing the data. During each pass, a regression parameter vector β is updated until it makes no more than a specified number of mistakes.

The Perceptron computes the label $\hat{y}_i$ for feature vector $\mathbf{x}_i$ using the $\sigma(\star) = \text{sign}(\star)$ function:

$$\hat{y}_i = \text{sign}\left(\mathbf{x}_i^T \cdot \beta\right)$$

where $\beta = (w_1, \ldots, w_n, b)$ is a column vector of (unknown) weight parameters $w_j \in \mathbb{R}$ corresponding to the importance of feature j and a bias parameter $b \in \mathbb{R}$, the features $\mathbf{x}_i^T = \left(x_1^{(i)}, \ldots, x_n^{(i)}, 1\right)$ is the $n+1$-dimensional feature (row) vector (features augmented with bias term), and $\text{sign}(z)$ is the `sign` function:

$$\text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

If data set $\mathcal{D}$ is linearly separable, the Perceptron will find a separating hyperplane in a finite number of passes through $\mathcal{D}$. However, if the data set $\mathcal{D}$ is not linearly separable, the Perceptron will not converge. Pusedo code for the perceptron algorithm is shown in Algorithm 2.

Algorithm 2 The Perceptron Algorithm

Input: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, tolerance $\epsilon \geq 0$, maximum iterations `maxiter`
Features: $\mathbf{x}_i = (x_{i1}, \ldots, x_{in}, 1)$ are augmented with a bias term, labels $y_i \in \{-1, 1\}$.
Output: Classifier parameters $\beta = (w_1, \ldots, w_n, b)$
$\beta \leftarrow$ `rand` ▷ Initialize parameter vector β to a random vector
$i \leftarrow 0$ ▷ Initialize the loop counter to zero
while true **do** ▷ Repeat until stopping criterion is met
    error $\leftarrow 0$ ▷ Initialize the error count to zero for this pass through $\mathcal{D}$
    for $(\mathbf{x}, y) \in \mathcal{D}$ **do** ▷ Iterate over each pair $(\mathbf{x}, y)$ in data set $\mathcal{D}$
        if $y \cdot \left(\mathbf{x}^T \cdot \beta\right) \leq 0$ **then** ▷ Ooops! The data pair $(\mathbf{x}, y)$ is misclassified
            $\beta \leftarrow \beta + y \cdot \mathbf{x}$ ▷ Update the weight vector β
            error $\leftarrow$ error $+ 1$ ▷ Increment the error count
        end if
    end for
    if error $\leq \epsilon$ **or** $i \geq$ `maxiter` **then** ▷ Stopping criterion: tolerance or max iterations?
        break ▷ Exit the training loop
    end if
    $i \leftarrow i + 1$ ▷ Increment the loop counter and repeat
end while

What the algorithm does is to update the weight vector β for each example $(\mathbf{x}_i, y_i)$ in the data set $\mathcal{D}$. If the example is misclassified, i.e., $y_i \left(\mathbf{x}_i^T \cdot \beta\right) \leq 0$, the weight vector β is updated by adding the feature vector $\mathbf{x}_i$ multiplied by the label y_i. The algorithm continues to iterate through the data set $\mathcal{D}$ until all examples are correctly classified or until a stopping criterion is met, e.g., a maximum number of iterations or a tolerance on the number of classification errors. In the end, the alogrithm estimates a linear hyperplane $\mathcal{H} = \left\{\mathbf{x} \mid \mathbf{x}^T \cdot \beta = 0\right\}$ that separates the two classes of data points in $\mathcal{D}$ (Fig. 1).

4.1 Convergence of the Perceptron Algorithm

The perceptron algorithm is guaranteed to converge to a solution if the data set is linearly separable in a finite number of passes through the data set. Let's sketch out why this is the case. Suppose the data set $\mathcal{D}$ is linearly separable, i.e., there exists a weight vector $\beta^\star$ such that the true labels y_i are correctly classified, i.e., $y_i \left(\mathbf{x}^T \cdot \beta\right) > 0$ for all examples $(\mathbf{x}_i, y_i) \in \mathcal{D}$. Next, suppose we rescale the weight vector β and the data set $\mathcal{D}$ such that $||\beta|| = 1$ and $||\mathbf{x}_i|| = 1$ for all examples $(\mathbf{x}_i, y_i) \in \mathcal{D}$. Finally, we define the margin of the data set $\mathcal{D}$ as the minimum distance between the data points and the decision boundary defined by the weight vector β:

$$\gamma = \min_i | \left(\mathbf{x}_i^T \cdot \beta\right) |$$

Theorem 1. *If all the condtions above are satisfied, the perceptron algorithm will make at most* $1/\gamma^2$ *mistakes. For a proof of this theorem, see the CS4780 lecture notes from 2018.*

5 Evaluation of the Binary Classifier

Once the Perceptron (or any binary classifier) has converged, we can evaluate the performance of the binary classifier using a variety of metrics. See Sidey-Gibbon et al (2) for a detailed discussion of these metrics in the context of medical classification problems. The central idea is to compare the predicted labels $\hat{y}_i$ to the

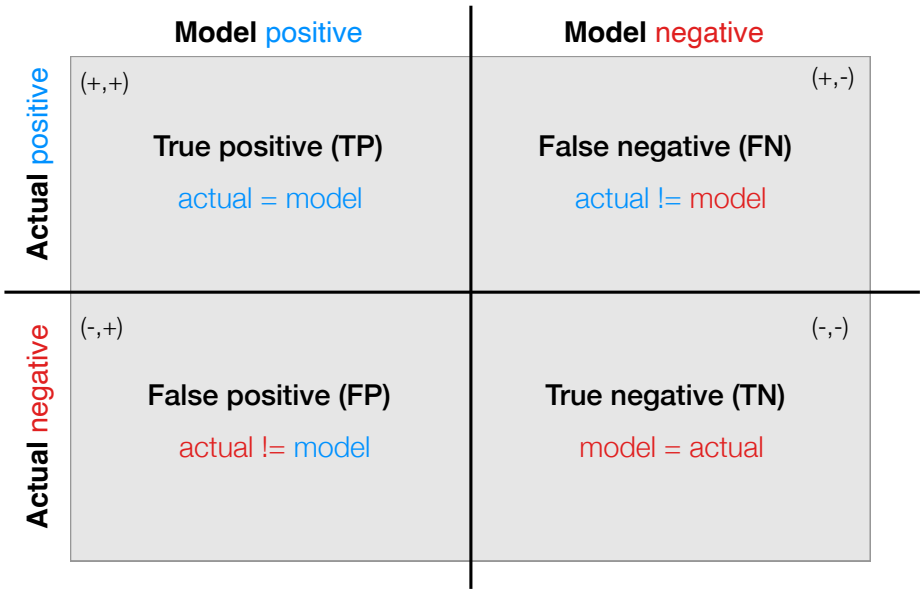

Figure 2: Confusion matrix schematic for the binary classification problem.

true labels y_i in the data set $\mathcal{D}$. There are a variety of metrics that can be used to evaluate the performance of a binary classifier, but they all start with computing the confusion matrix.

5.1 Confusion Matrix

The confusion matrix is a table that is often used to describe the performance of a classification model on a set of data for which the true values are known (Fig. 2). The confusion matrix is a 2×2 matrix that contains four entries: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). The true positive (TP) entry in the confusion matrix is the number of positive examples that were correctly classified as positive. The false negative (FN) entry is the number of positive examples that were incorrectly classified as negative by the model. The false positive (FP) entry is the number of negative examples that were incorrectly classified as positive by the model. The true negative (TN) entry is the number of negative examples that were correctly classified as negative by the model. The confusion matrix is used to calculate a variety of performance metrics, including accuracy, precision, recall, and the F1 score.

Accuracy. The accuracy of a binary classifier is the proportion of correctly classified *total* examples in the data set. The accuracy is calculated as the sum of the true positive and true negative entries in the confusion matrix divided by the total number of examples in the data set:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Thus, accuracy is a measure of the overall performance of the classifier and indicates how well the classifier is able to correctly classify examples (both positive and negative).

Precision. The precision of a binary classifier is the proportion of correctly classified *positive* examples out of all examples that were classified as positive by the model. The precision is calculated as the true

positive entry in the confusion matrix divided by the sum of the true positive and false positive entries:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision is a measure of the classifier's ability to correctly classify positive examples and is useful when the cost of false positives is high.

Recall (Sensitivity). The recall of a binary classifier is the proportion of correctly classified *positive* examples out of all examples that are truly positive in the data set. The recall is calculated as the true positive entry in the confusion matrix divided by the sum of the true positive and false negative entries:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall is a measure of the classifier's ability to correctly classify positive examples and is useful when the cost of false negatives is high.

F1 Score. The F1 score of a binary classifier is the harmonic mean of precision and recall. The F1 score is calculated as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score is a measure of the classifier's overall performance and balances the trade-off between precision and recall.

6 Summary

In this lecture, we introduced the perceptron algorithm for binary classification problems. The perceptron is a simple linear classifier that can be used to separate two classes of data points. The perceptron algorithm is a simple iterative algorithm that incrementally updates the weights of the linear classifier to minimize the classification error. Thus, it is one of the first examples of on online learning algorithm, i.e., an algorithm that learns from data in an incremental fashion. The perceptron algorithm is guaranteed to converge to a solution if the data set is linearly separable. However, if the data set is not linearly separable, the perceptron algorithm will not converge to a perfect solution. If we are willing to accept some classification errors, we can use the perceptron algorithm to find a separating hyperplane in a finite number of passes through the data set, even if the data set is not linearly separable.

References

1. Rosenblatt F. Perceptron Simulation Experiments. Proceedings of the IRE. 1960;48(3):301–309. doi:10.1109/JRPROC.1960.287598.

2. Sidey-Gibbons JAM, Sidey-Gibbons CJ. Machine learning in medicine: a practical introduction. BMC Med Res Methodol. 2019;19(1):64. doi:10.1186/s12874-019-0681-4.