

Lecture 3a: Linear Regression, Perceptron and Binary Classification

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

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:

- **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.

2 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 continuous 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 $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ 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:

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

where we have augmented the feature vector \mathbf{x}_i with a bias term, i.e., $\mathbf{x}_i^T = (x_1^{(i)}, \dots, x_n^{(i)}, 1)$, $\beta = (w_1, \dots, w_n, b)$ 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 vector 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.

2.1 Overdetermined 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 l_2 -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:

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

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 $(\mathbf{X}^T \mathbf{X} + \lambda \cdot \mathbf{I})^{-1}$ must exist to obtain the estimated parameter vector $\hat{\beta}_\lambda$.

2.2 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:

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

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

3 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., $\sigma : \mathbb{R} \rightarrow \{-1, 1\}$ in the binary classification case. Suppose there exists a data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ with m *labeled* examples, where each example $1, 2, \dots, m$ 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:

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

where $\beta = (w_1, \dots, w_n, b)$ 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 $\mathbf{x}_i^T = (x_1^{(i)}, \dots, x_n^{(i)}, 1)$ 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. Pseudo code for the perceptron algorithm is shown in Algorithm 1.

Algorithm 1 The Perceptron Algorithm

```

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

```

4 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 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.

		Model positive	Model negative
Actual positive	(+,+)	True positive (TP) actual = model	False negative (FN) actual != model
Actual negative	(-,+)	False positive (FP) actual != model	True negative (TN) model = actual
		(+,-)	(-,-)

Figure 1: Confusion matrix schematic for the binary classification problem.

4.1 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. 1). 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.

5 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 an 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.