Lecture 3 Notes

Introduction to Data Science IS360

1 Linear Classification

The simplest case of classification is binary classification, where data is separated into 2 classes. In linear models algorithms will choose from a hypothesis set of linear classifiers, each hypothesis h is in the form:

$$h(x) = \text{sign}(\mathbf{w}^\top \cdot \mathbf{x}). \tag{1}$$

Note 1. The bias term b is omitted. Instead, we add a coordinate x_0 in the vector for $\mathbf{x}$ set to 1. If we have d features our input space is $\mathcal{X} = \{1\} \times \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^{d+1}$

1.1 Limitations of PLA

The perceptron learning algorithm (PLA) is covered in lecture 1.

PLA starts with an arbitrary weight vector w and at each time step, selects a single misclassified point i and performs the update:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \cdot y_i \cdot x_i. \tag{2}$$

In linearly separable data $\mathcal{D}$ PLA will always converge/terminate (reaching a $E_{\text{in}} = 0$). However, in data that is non-linear, even data that contains outliers or noisy data, PLA will never terminate.

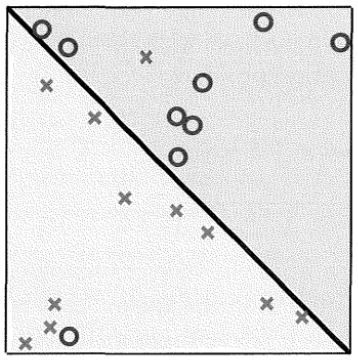


Figure 1: Linearly separable data with some noise. PLA would not terminate on this data.

We don't always want to reach $E_{\text{in}} = 0$, usually finding the solution with the lowest E_{in} is sufficient. The problem with PLA, since it only focuses on a single point at a time, is that it may discard a good solution due to a misclassified outlier.

Handwritten Digit Recognition Example

Each image is a 16×16 image, for simplicity we consider two features from the images: the symmetry, and the average intensity.

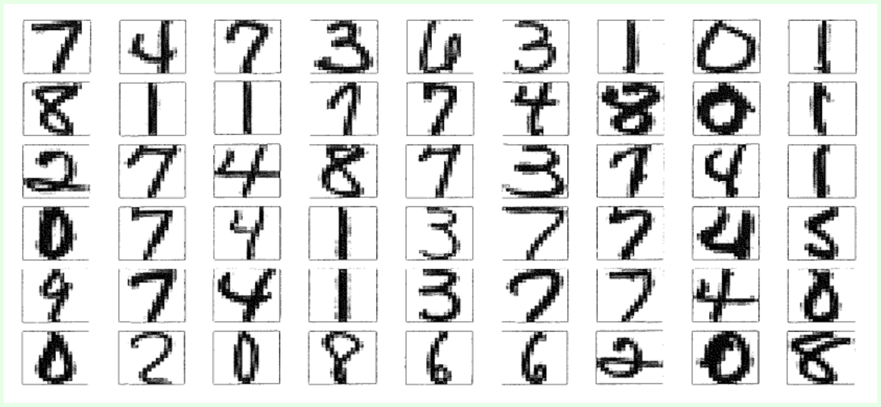


Figure 2: Examples from digit dataset.

Consider the binary classification task of identifying whether an image contains a five or a one. For this task we can visualize dataset for this task using our two features on a graph.

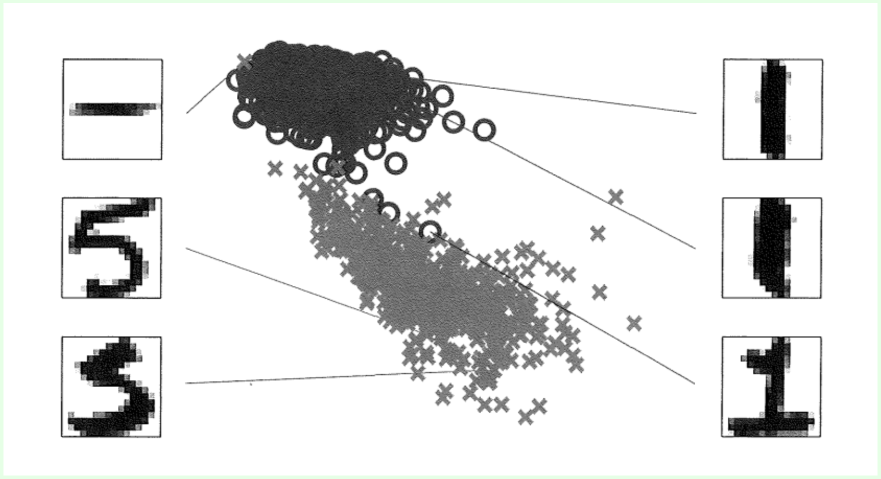
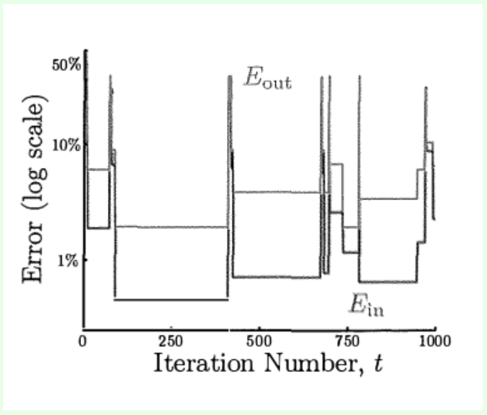
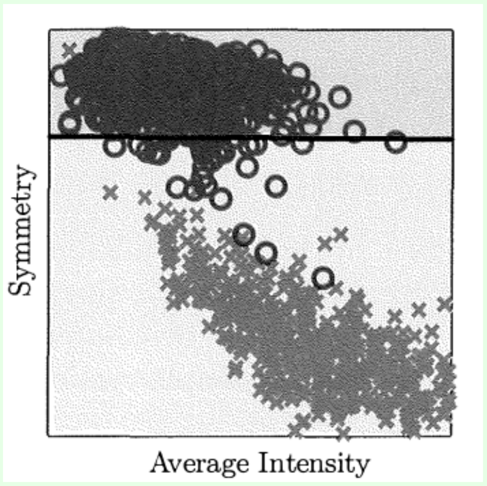


Figure 3: Visualization of the digit dataset with the x-axis (x_1) being the average intensity and the y-axis (x_2) being the symmetry. The data seems to be linearly separable with some outliers.

If we run PLA on this data, due to the noisy data points, it will not terminate. We can stop at after a set number of iterations and look at the loss curve.



(a) Loss curve of PLA algorithm. E_{out} on a test dataset is also included.



(b) Linear boundary created by PLA at termination.

Figure 4: The algorithm reaches the solution with the lowest E_{in} in the first 100 iterations, but at termination (1000 iterations) that solution is lost and we end up with a worse solution. Notice also how the loss curve fluctuates between good solutions and bad ones throughout training.

These problems are both associated with the fact that PLA only looks at a single point when learning. One possible way to prevent this from occurring would be to remove the outliers. However, outlier detection might be a (computationally) expensive process and has it's own considerations depending on the dataset.

1.2 The Pocket Algorithm

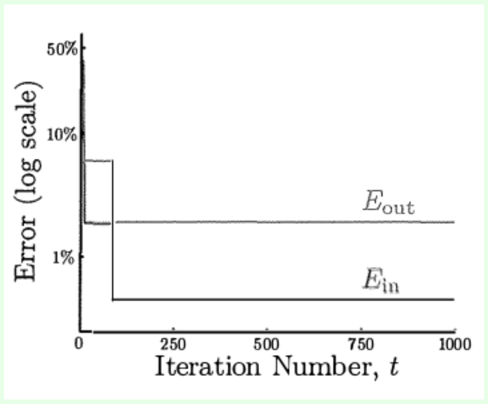
The pocket algorithm is a modified version of PLA that aims to address it's limitations. The algorithm can be described as follows

```
1. Create a variable ‘pocket‘ $\hat{\mathbf{w}}$ to store the best solution so far.
2. Initialize $\mathbf{w}$ with random values.
3. For each iteration till termination:
4.     Run PLA to obtain updated weights $\mathbf{w}$.
5.     Evaluate in-sample error with updated weights $E_{\text{in}}(\mathbf{w})$
6.     If current $\mathbf{w}$ better than $\hat{\mathbf{w}}$: $\mathbf{w} \leftarrow \hat{\mathbf{w}}$
7. Return $\hat{\mathbf{w}}$
```

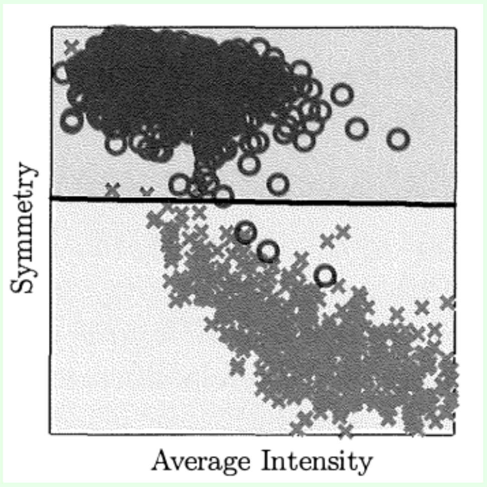
The main difference between PLA and the pocket algorithm is highlighted in steps 5 and 6. At each iteration we check the in-sample error for the entire weight vector $E_{\text{in}}(\mathbf{w})$, and we only return the best solution with the lowest error.

Note 2. Since the pocket algorithm is based on PLA it has the same convergence properties.

Handwritten Digit Recognition Example Cont.
We can run the pocket algorithm on the same binary recognition task and see how it performs differently.



(a) Loss curve of pocket algorithm. E_{out} on a test dataset is also included.



(b) Linear boundary created by pocket algorithm at termination.

Figure 5: Comparing the loss curve with the one for PLA, the one for the pocket algorithm has fewer fluctuations, and more importantly the solution with the lowest in-sample error E_{in} is the solution returned at termination.

2 Linear Regression

2.1 Regression

Regression learning tasks are associated with real-valued target functions (as opposed to the discrete target functions in classification). Instead of predicting a class for a feature vector $\mathbf{x}$, we predict a continuous value associated with those features.

It's useful to think of the target 'function' as less of a function $y = f(\mathbf{x})$ and more as a probability distribution $\Pr(y \mid \mathbf{x})$ that generates each y given the feature vector $\mathbf{x}$. The goal (similar to the goal in classification) is to find a hypothesis h that minimizes the error between $h(\mathbf{x})$ and y with respect to that distribution.

We can also think of a joint distribution $\Pr(\mathbf{x}, y)$ over both $\mathbf{x}$ and y, which generates the $(\mathbf{x}_i, y_i)$ pairs:

$$\Pr(\mathbf{x}, y) = \Pr(y \mid \mathbf{x}) \Pr(\mathbf{x}), \tag{3}$$

where $\Pr(\mathbf{x})$ is the marginal distribution of the input features.

Why a Probability Distribution Is More Useful
Error Term

Measurement in the real world is always subject to noise or error. Using a probability distribution allows us to account for this error by adding an error term ϵ with its own probability distribution.

In linear regression, we usually assume the error term ϵ follows a normal distribution:

$$\epsilon \sim \mathcal{N}(0, \sigma^2), \tag{4}$$

where σ^2 is the variance of the noise around the prediction. This distributional assumption enables us to express the target variable y for a given $\mathbf{x}$ as a random variable:

$$y = h(\mathbf{x}) + \epsilon. \tag{5}$$

$$\Pr(y \mid \mathbf{x}) = \mathcal{N}(y; h(x), \sigma^2) \tag{6}$$

Note 3. y is normally distributed around the linear prediction $h(x) = \mathbf{w}^\top \cdot \mathbf{x}$ with variance σ^2.

Predictive Uncertainty

Treating y as a random variable allows us to capture predictive uncertainty. Rather than giving a single predicted value, we can provide a range, or interval, where the true value of y is likely to fall. For a new input $\mathbf{x}_{new}$, the prediction $h(\mathbf{x}_{new}$ is associated with a confidence interval:

$$h(x_{new}) \pm z \cdot \sigma, \tag{7}$$

where z depends on the desired confidence level (e.g., $z \approx 1.96$ for a 95% confidence interval).

2.2 Linear Regression Algorithm

In linear regression, h takes the form of a linear combination of the elements of $\mathbf{x}$:

$$h(x) = \sum_{i=0}^{d} w_i \cdot x_i = \mathbf{w}^\top \cdot \mathbf{x}, \tag{8}$$

where $\mathbf{x} \in 1 \times \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}$.

Consider the squared error between $h(\mathbf{x})$ and y:

$$E_{\text{out}}(h) = \mathbb{E}\left[(h(\mathbf{x}) - y)^2\right], \tag{9}$$

where the expected value is taken with respect to $\Pr(\mathbf{x}, y)$.

However, since $\Pr(\mathbf{x}, y)$ is unknown, $E_{\text{out}}(h)$ cannot be computed. Instead, we compute E_{in}:

$$E_{\text{in}}(h) = \frac{1}{N} \sum_{n=1}^{N} (h(\mathbf{x}_n) - y_n)^2. \tag{10}$$

For a linear h, it's useful to represent $E_{\text{in}}(h)$ as a matrix:

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \|\mathbf{X} \cdot \mathbf{w} - \mathbf{y}\|^2, \tag{11}$$

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \left(\mathbf{w}^\top \cdot \mathbf{X}^\top \cdot \mathbf{X} \cdot \mathbf{w} - 2\mathbf{w}^\top \cdot \mathbf{X}^\top \cdot \mathbf{y} + \mathbf{y}^\top \cdot \mathbf{y}\right), \tag{12}$$

where

N : Number of samples,
$\mathbf{w}$: Weight vector ($\mathbf{w} \in \mathbb{R}^{d+1}$),
$\mathbf{X}$: The $N \times (d+1)$ data matrix, containing the inputs ($\mathbf{X} \in \mathbb{R}^{N \times (d+1)}$),
$\mathbf{y}$: Target vector, where each element y_i corresponds to its $\mathbf{x}_i$ pair ($\mathbf{y} \in \mathbb{R}^N$),
$\|\cdot\|^2$: Euclidean norm of a vector (*think of this as a distance metric*).

$$\underbrace{\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1^{\mathrm{T}}— \\ —\mathbf{x}_2^{\mathrm{T}}— \\ \vdots \\ —\mathbf{x}_N^{\mathrm{T}}— \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

Note 4. h and $\mathbf{w}$ are used interchangeably.

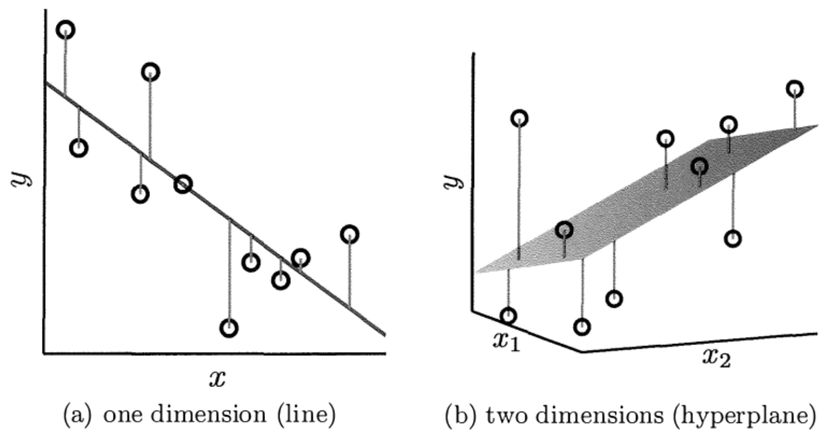


Figure 6: The solution hypothesis for a linear regression algorithm in one and two dimensions. The error between h and $\mathbf{y}$ is also shown.

The linear regression algorithm finds the weight vector $\mathbf{w}$ by minimizing $E_{\text{in}}(\mathbf{w})$ over all possible $\mathbf{w} \in \mathbb{R}^{d+1}$:

$$\mathbf{w}_{\text{lin}} = \underset{\mathbf{w} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} E_{\text{in}}(\mathbf{w}). \tag{13}$$

To minimize, first we find the gradient of the E_{in}:

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\left(\mathbf{X}^\top \cdot \mathbf{X} \cdot \mathbf{w} - \mathbf{X}^\top \cdot \mathbf{y}\right), \tag{14}$$

then if we set this gradient to zero we get:

$$\mathbf{X}^\top \cdot \mathbf{X} \cdot \mathbf{w} = \mathbf{X}^\top \cdot \mathbf{y}, \tag{15}$$

we then solve for $\mathbf{w}$:

$$\mathbf{w} = \mathbf{X}^+ y, \tag{16}$$

where $\mathbf{X}^+$ is the pseudo-inverse of $\mathbf{X}$ ($\mathbf{X}^+ = \left(\mathbf{X}^\top \cdot \mathbf{X}\right)^{-1} \mathbf{X}^\top$).

We can now formalize the linear regression algorithm as:

1. Construct the data matrix $\mathbf{X}$, and target matrix $\mathbf{y}$.
2. Compute the the pseudo-inverse $\mathbf{X}^+$ of the data matrix $\mathbf{X}$.
3. Return $w = \mathbf{X}^+ \cdot y$.

Linear Regression For Classification
Linear regression can be used for binary classification tasks (or at least give us a good starting point), since the output is real valued ($\{-1, 1\} \in \mathbb{R}$).

The approach involves finding a weight vector $\mathbf{w}$ such that the linear model approximation $\mathbf{w}^T \cdot \mathbf{x}_n \approx y_n$ holds, where $y_n = \pm 1$ represents the class labels of the data points. Once the weights $\mathbf{w}$ are determined, we can classify new data points by evaluating the sign of $\mathbf{w}^T \cdot \mathbf{x}_n$:

$$\operatorname{sign}(\mathbf{w}^T \cdot \mathbf{x}_n), \tag{17}$$

which is likely to agree with y_n for correctly classified data points.

Invertability and The Pseudo-Inverse
Invertability is a property of square matrices. For a given matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, its inverse is the matrix $\mathbf{A}^{-1}$ such that:

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}, \tag{18}$$

where $\mathbf{I}$ is the identity matrix.

In ML our matrices (especially the data matrix) are almost never square, i.e., we almost always have more examples (rows) N than features (columns) $d + 1$. For cases like these we find the pseudo-inverse. For a matrix $\mathbf{A}$ its pseudo-inverse is denoted $\mathbf{A}^{+}$ and is defined as:

$$\mathbf{A}^{+} = \left(\mathbf{A}^{\top} \cdot \mathbf{A}\right)^{-1} \mathbf{A}^{\top}. \tag{19}$$

But we are still computing an inverse here, the inverse of $\mathbf{A}^{\top} \cdot \mathbf{A}$. Is that inverse guaranteed to exist?

In (16) we assume that $\mathbf{X}^{\top} \cdot \mathbf{X}$ is invertible; it's actually one of the conditions for that equation. However, this might not always be the case. Even when $\mathbf{X}^{\top} \cdot \mathbf{X}$ is not invertible (i.e., singular) the pseudo-inverse still exists; it's just that it's calculated differently (using methods like SVD or regularization).

One reason $\mathbf{X}^{\top} \cdot \mathbf{X}$ might not be invertible is due to rank-deficiency in the data matrix $\mathbf{X}$, this happens when columns (features) of the data matrix are collinear, that is, columns of the data matrix are linearly dependent and can be represented as linear combinations of each other.

Collinearity in data creates a problem where infinitely many sets of weights produce the minimum $E_{\text{in}}(\mathbf{W})$. This means the solution is not unique, and $\mathbf{X}^{\top} \cdot \mathbf{X}$ is singular.

In order to use (16) for linear regression, we need to make sure our features are linearly independent using regularization or preprocessing.