

Introduction to Machine Learning

Jelke Bloem & Giovanni Colavizza

Text Mining
Amsterdam University College

March 15, 2022

Announcements

- **Individual assignment 2 deadline: 18/03, 23:59**

Overview

1 Machine Learning

2 Linear Regression

3 Extras

Machine Learning

What is Machine Learning?

- What does it mean to learn?
 - ▶ Given historical data, we are interested in **predicting** the unseen future.
 - ▶ Given unstructured data, we are interested in uncovering **structure** (patterns).
 - ▶ Given an environment and a goal, we are interested in **acting** to reach the goal.
- Memorization is not learning, **generalization** is what matters.
- We usually achieve this using an **inductive approach**: by seeing known examples (**train dataset**), we attempt to distill the signal and filter out the noise, in order to predict future examples. We use a left-out slice of the data (**test dataset**) to simulate the future.
- You don't look at your test data! It has to be **unseen**.

Types of ML

- **Supervised learning:** we are given a labelled dataset $\{X, Y\}$, with labels for every data point. Our goal is to learn to **predict** labels for future data points. Examples: regression, classification.
- **Unsupervised learning:** we are given an unlabelled dataset $\{X\}$. Our goal is to learn to **structure** data points in some meaningful way. Examples: clustering, distribution fitting.
- **Reinforcement learning:** we are given an environment and some **agents** acting in it, which have access to a notion of **reward**. Agents seek to take actions in the environment, maximizing the reward. Usually, actions (or sequences thereof) are linked to rewards.
- More: Semi-supervised, Multi-task, Transfer learning, etc.

The components of a probabilistic ML classifier

- A **dataset** and its **feature representation**.
- A **classification function**, or **model**. This model specifies a relationship between inputs and outputs, using parameters (to be learned) and hyperparameters (given by us).
- A **loss function** (also called objective or cost function): something we want to minimize as a proxy for “learning”. This function encapsulates what it means to learn for us.
- An algorithm for **optimization**: a way to find good model parameters which minimize the loss function.

Example: The perceptron

- The very beginnings (Rosenblatt 1958), still at the core for the neural model of learning.
- Inspired by a neuron with incoming connections from other neurons that represent features
- We have a **dataset** $\{X, Y\}$, so that $\mathbf{x} = \langle x_1, x_2, \dots, x_d \rangle, y \in \{-1, +1\}, \forall (\mathbf{x}, y) \in \{X, Y\}$. We thus have a binary classification problem.
- The **perceptron model** is defined by a linear combination of weights $\langle w_1, w_2, \dots, w_d \rangle$ and features, plus an optional bias term:

$$a = \left[\sum_{d=1}^D w_d x_d \right] + b$$

- *sign* is our **classification function** (positive or negative): does the neuron fire? The bias shifts the decision threshold.

Perceptron loss

- Training a perceptron is online: exemplar by exemplar
- When do we have to adjust the weights?
- We can use the so-called **0/1 loss**.

$$l^{0/1} = \min_{\mathbf{w}, b} \sum_n \mathbf{1}[y_n(\mathbf{w} \cdot \mathbf{x}_n + b) < 0]$$

- Equivalently, to simplify: $l^{0/1} = \min \sum_n \mathbf{1}[y_n \hat{y} < 0]$
- For each mistaken prediction, penalty of 1
- When the data are linearly separable, the minimum can be zero.
Why?

Perceptron optimization

- How does the perceptron learn?

$$a = \left[\sum_{d=1}^D w_d x_d \right] + b$$

- **Optimization:** it is an online (1 data point at the time) and error-driven algorithm (we want to make no errors). Given a datapoint in the train dataset, *if the perceptron's prediction is correct, do nothing, else:*

$$w_d^t \leftarrow w_d^{t-1} + y x_d$$

$$b^t \leftarrow b^{t-1} + y$$

(y is the label, positive or negative)

Exercise

Exercise: Training a perceptron

Your dataset is the following:

$\{(2, 1; -1), (1, 2; +1), (3, 1; -1), (3, 2; -1), (1, 3; +1), (2, 3; +1)\}$. Assume we use a perceptron without bias term. Also assume we start with random weights: $w_1^{(0)} = 1/2, w_2^{(0)} = -1/2$.

- The first iteration goes as follows:

- ① $a_1 = w_1^{(0)} x_{11} + w_2^{(0)} x_{12} = 1 - 1/2 = 1/2$. $\text{sign}(1/2) = +$, thus we have an error and we need to update weights.
- ② Weight update at iteration 1:

$$w_1^{(1)} \leftarrow w_1^{(0)} + y_1 x_{11} = 1/2 - 2 = -3/2$$

$$w_2^{(1)} \leftarrow w_2^{(0)} + y_1 x_{12} = -1/2 - 1 = -3/2$$

- ③ Proceed to do the same for w_2 and the following data points. Does your perceptron converge to a boundary after one pass on the data? If so, can you draw the boundary?

Exercise

Exercise: Training a perceptron

$$\textcircled{1} \quad a_2 = w_1^{(1)} x_{21} + w_2^{(1)} x_{22} = -3/2 - 3 = -4.5. \quad \text{sign}(-4.5) = -$$

$$w_1^{(2)} \leftarrow w_1^{(1)} + y_1 x_{11} = -3/2 + 1 = -1/2$$

$$w_2^{(2)} \leftarrow w_2^{(1)} + y_1 x_{12} = -3/2 + 2 = 1/2$$

$$\textcircled{2} \quad a_3 = w_1^{(1)} x_{31} + w_2^{(1)} x_{32} = -3/2 + 1/2 = -1. \quad \text{sign}(-1) = -$$

$$\textcircled{3} \quad a_4 = w_1^{(1)} x_{41} + w_2^{(1)} x_{42} = -3/2 + 1 = -1/2. \quad \text{sign}(-1/2) = -$$

$$\textcircled{4} \quad a_5 = w_1^{(1)} x_{51} + w_2^{(1)} x_{52} = -1/2 + 3/2 = 1. \quad \text{sign}(1) = +$$

$$\textcircled{5} \quad a_6 = w_1^{(1)} x_{61} + w_2^{(1)} x_{62} = -2/2 + 3/2 = 1/2. \quad \text{sign}(1/2) = +$$

In summary

- Some properties of the perceptron include:
 - ① **It always converges if the data points are linearly separable.**
 - ② It is unable to distinguish among decision boundaries.
 - ③ The linear model it embeds computes **a projection of every feature** x_d onto the vector \mathbf{w} . This means that we basically order the projected features on a line, sum them up and check if they are above or below a threshold!
 - ④ It is unable to go beyond linearly separable data (infamous XOR problem). Extensions include: 'stacking up' perceptrons (**neural networks**) and doing feature maps (**kernel methods**).
 - ⑤ See HD, ch. 4 for more.

Key concepts: Generalization

- We have a loss function l and a dataset $\{X, Y\}$. We take a probabilistic view and state that we *assume* the existence of a data generating distribution \mathcal{D} over data pairs (x, y) , giving probabilities to pairs of data points. We then learn a function f that minimizes the loss for our data points, in view of generalizing to new data points under \mathcal{D} .

- We would like to learn to minimize the **expected loss** ϵ over \mathcal{D} :

$$\epsilon := \mathbb{E}_{(x,y) \sim \mathcal{D}} [l(y, f(x))] = \sum_{(x,y)} \mathcal{D}(x,y) l(y, f(x))$$

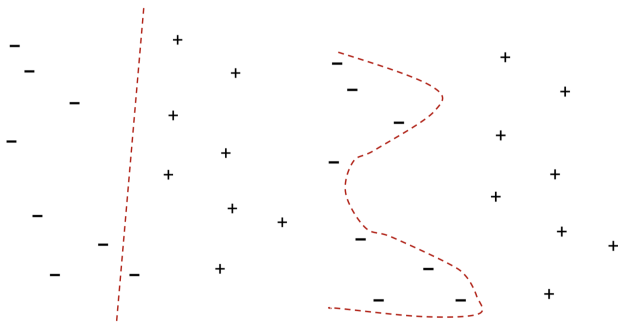
- But we do not know \mathcal{D} !
- Instead, we compute the **training error** $\hat{\epsilon}$ assuming it approximates ϵ :

$$\hat{\epsilon} := \frac{1}{N} \sum_{n=1}^N l(y_n, f(x_n))$$

- Which means we assume \mathcal{D} to be uniform over our training examples and zero anywhere else. That is: **independent, uniformly and identically distributed**.

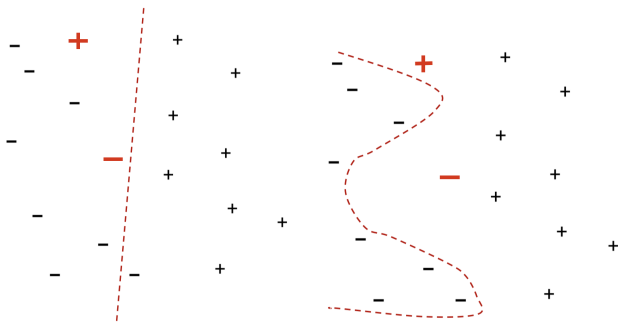
Key concepts: Underfitting and overfitting

- **Underfitting:** “you had an opportunity to learn something but did not”. **Overfitting:** “you pay too much attention to the idiosyncracies of the data, and are not able to generalize well.” HD, ch. 2.



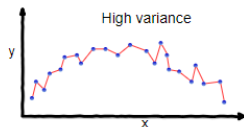
Key concepts: Underfitting and overfitting

- **Underfitting:** “you had an opportunity to learn something but did not”. **Overfitting:** “you pay too much attention to the idiosyncracies of the data, and are not able to generalize well.” HD, ch. 2.

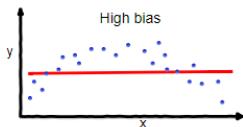


Key concepts: Underfitting and overfitting

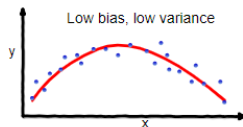
- Also referred to as **Bias-variance trade-off**.
- Note: this theory seems not to apply to deep learning! Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. "Reconciling Modern Machine-Learning Practice and the Classical Bias-Variance Trade-Off." Proceedings of the National Academy of Sciences 116 (32): 15849–54.
<https://doi.org/10.1073/pnas.1903070116>.



overfitting



underfitting



Good balance

Key concepts: Optimization

- Every **parametric model** expresses a set of parameters, which we need to tune during learning. E.g., word2vec.
- **Non-parametric models** instead, use the whole dataset as parameters. E.g., k-NN (nearest neighbours), as we will see later on.
- **Regularization**: adding constraints to parameters to avoid overfitting.
- **Hyperparameters**: not learned with the model/optimization. We can still use the data to find good values. E.g., **cross-validation**: train different models over a range of hyperparameter combinations, and pick the best. We use a third slice of the dataset for this: the **validation (or development) set**.

Linear Regression

Linear models for regression

- We have a **dataset** $\{X, Y\}$, so that $\mathbf{x} = \langle x_1, x_2, \dots, x_d \rangle, y \in \mathbb{R}$, $\forall (\mathbf{x}, y) \in \{X, Y\}$. We thus have a regression problem.
- Examples: predict house prices, predict height of persons.
- The **model** is a **linear, weighted combination of the inputs**. In general:

$$\hat{y} = b + \sum_{d=1}^D w_d x_d$$

- y is the true value, \hat{y} is the predicted value from the model.
- I will put the intercept b in the summation as w_0 by adding an $x_0 = 1$, and use matrix notation (in bold):

$$\hat{y} = \sum_{d=0}^D w_d x_d$$

$$\hat{y} = \mathbf{X}\mathbf{w}$$

Loss functions: Convexity

- With the perceptron, we used the so-called **0/1 loss**.

$$l^{0/1} = \min_{\mathbf{w}, b} \sum_n \mathbf{1}[y_n(\mathbf{w} \cdot \mathbf{x}_n + b) < 0]$$

- Equivalently, to simplify: $l^{0/1} = \min \sum_n \mathbf{1}[y_n \hat{y} < 0]$
- Unfortunately, the perceptron's learning algorithm is feasible only if the data points are linearly separable, i.e. if the minimum of $l^{0/1}$ is zero. This is rarely the case in practice. *Question: what happens if we use the perceptron on a dataset which is not linearly separable?*
- A popular alternative is to choose less exact but easier to work with loss functions. In particular, we pick from **convex functions**, so that we can use techniques from calculus.

Convexity

- A function is convex if, equivalently: its second derivative is always positive or any chord of the function lies above it.

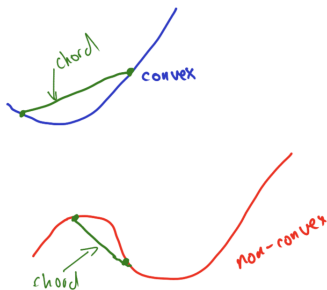


Figure 7.3: plot of convex and non-convex functions with two chords each

Loss functions for linear regression

- There is a variety of loss functions which are convex or semi-convex. There are several options for linear regression. For example:
 - ① **Mean Squared Error (MSE):** $l^{MSE} = \min \sum_n (y_n - \hat{y})^2$
 - ② **Mean Absolute Error (MAE):** $l^{MAE} = \min \sum_n |y_n - \hat{y}|$
 - ③ **Hinge:** $l^{hin} = \min \sum_n \max\{0, 1 - y_n \hat{y}\}$
- They vary on how they deal with erroneous predictions (e.g., MSE is very sensitive to them) and with confident correct predictions (e.g., ignore them with Hinge).
- They are **differentiable or semi-differentiable**.

Closed-form solution for MSE Linear regression

- Let us pick MSE. In this particular case, we can derive a closed-form solution via calculus.
- What we have, in matrix notation:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

$$\mathcal{L} = \frac{1}{2} \|\hat{\mathbf{y}} - \mathbf{y}\|^2$$

$$\underbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,D} \\ x_{2,1} & x_{2,2} & \dots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,D} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} \sum_d x_{1,d} w_d \\ \sum_d x_{2,d} w_d \\ \vdots \\ \sum_d x_{N,d} w_d \end{bmatrix}}_{\hat{\mathbf{y}}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\mathbf{y}} \quad (7.29)$$

Closed-form solution for MSE Linear regression

- We can express the loss as follows:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- We use calculus to minimize the loss by setting its derivative to zero:

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- This is an exact, but costly solution. Complexity: $\mathcal{O}(D^3 + D^2N)$, with D number of features and N number of data points.

Regularization

- Left unconstrained, MSE can easily lead to a case of **overfitting**, e.g. by paying too much attention to **outliers** (*why?*).
- Regularization is a way to compensate for this, by constraining weights to be small. It puts a premium on learning simple functions, by moving the model towards being more biased (*why?*).
- Examples of regularizers:
 - ▶ L_2 -norm (Ridge): $\lambda ||\mathbf{w}||^2$
 - ▶ L_1 -norm (Lasso): $\lambda |\mathbf{w}|$
- λ is a hyperparameter to control the intensity of the regularization.
 - ▶ Independent of the model and data
 - ▶ Penalty for exactly fitting the data

Closed-form solution with regularization

- We can express the loss as follows:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \hat{\mathbf{y}}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- We use calculus to minimize the loss by setting its derivative to zero:

$$\begin{aligned} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) &= \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w} = 0 \\ \mathbf{w} &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

- It still works as long as the regularization term is also convex

(Stochastic) Gradient Descent (SGD)

- There is not always a closed-form solution
- General-purpose method to find a minimum of differentiable functions. The bread and butter of deep learning.
- The **gradient of a function** $\nabla_w f$ is the vector consisting of the partial derivatives of this function w.r.t. each input coordinate:

$$\nabla_w f = \left\langle \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_1}, \dots, \frac{\partial f}{\partial w_D} \right\rangle$$

- SGD defined an iterative approach to reach a minimum of a function by **gradual update steps**:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_w f$$

- η (eta) is called the **learning rate**. We refer to **stochastic** GD when we use one (or few) data point(s) at the time.

Stochastic Gradient Descent (SGD)

- SGD defined an iterative approach to reach a minimum of a function by **gradual update steps**:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_{\mathbf{w}} f$$

$$w_1^{(t)} \leftarrow w_1^{(t-1)} - \eta \frac{\partial f}{\partial w_1}$$

Algorithm 21 GRADIENTDESCENT($\mathcal{F}, K, \eta_1, \dots$)

```
1:  $\mathbf{z}^{(0)} \leftarrow \langle o, o, \dots, o \rangle$  // initialize variable we are optimizing
2: for  $k = 1 \dots K$  do
3:    $\mathbf{g}^{(k)} \leftarrow \nabla_{\mathbf{z}} \mathcal{F}|_{\mathbf{z}^{(k-1)}}$  // compute gradient at current location
4:    $\mathbf{z}^{(k)} \leftarrow \mathbf{z}^{(k-1)} - \eta^{(k)} \mathbf{g}^{(k)}$  // take a step down the gradient
5: end for
6: return  $\mathbf{z}^{(K)}$ 
```

Stochastic Gradient Descent (SGD)

- SGD defined an iterative approach to reach a minimum of a function by **gradual update steps**:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_{\mathbf{w}} f$$

For a single scalar:

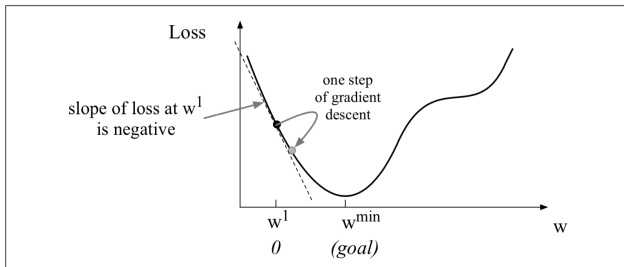


Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

Credit: J&M, ch. 5.

Stochastic Gradient Descent (SGD)

- SGD defined an iterative approach to reach a minimum of a function by **gradual update steps**:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_{\mathbf{w}} f$$

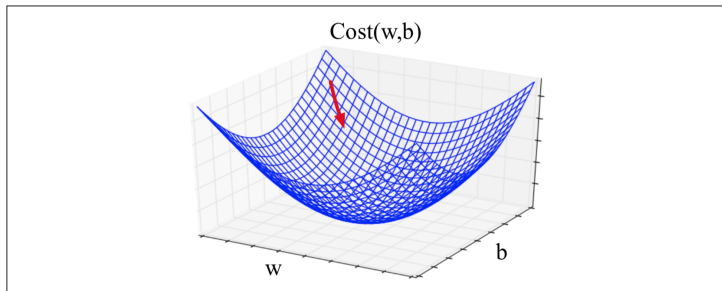


Figure 5.4 Visualization of the gradient vector in two dimensions w and b .

Credit: J&M, ch. 5.

Stochastic Gradient Descent (SGD)

- SGD defined an iterative approach to reach a minimum of a function by **gradual update steps**:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_{\mathbf{w}} f$$

- η (eta) is called the **learning rate**: this is crucial for convergence.

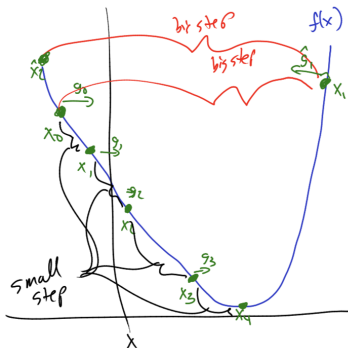
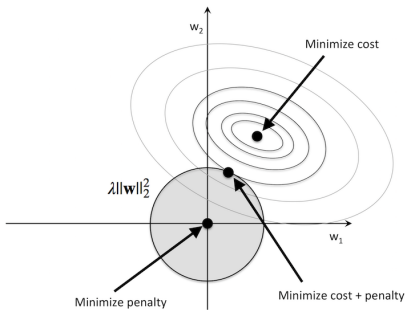
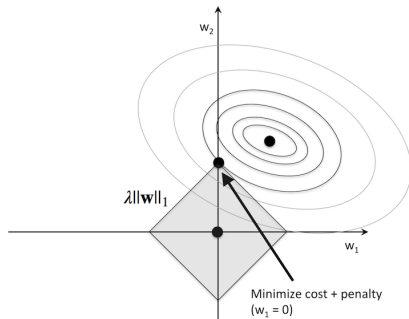


Figure 7.7: good and bad step sizes

Regularization and SGD



gray area = regularization constraint

Notation note: $\|\mathbf{w}\|_p = \left(\sum_d |\mathbf{w}_d|^p \right)^{\frac{1}{p}}$.

I have implied so far: $\|\mathbf{w}\|^2 = \|\mathbf{w}\|_2^2$ and $|\mathbf{w}| = \|\mathbf{w}\|_1$.

http://rasbt.github.io/mlxtend/user_guide/general_concepts/regularization-linear.

Putting everything together (SGD)

- SGD:

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_{\mathbf{w}} f$$

- Loss for linear regression with Mean Absolute Error and L_2 :

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \hat{\mathbf{y}}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- Regularized SGD for linear regression with MAE and L_2 :

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla_{\mathbf{w}} \mathcal{L}$$

- Same, for datapoint x_z and for weight w_1 :

$$w_1^{(t)} \leftarrow w_1^{(t-1)} - \eta \frac{x_{z1}y_z}{x_{z1}^2 + \lambda}$$

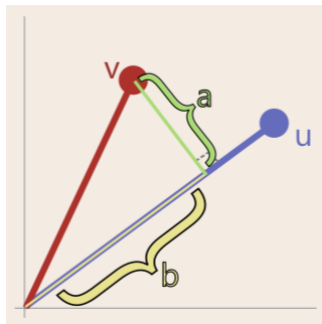
- Often, we feed data to SGD in **batches** (e.g., a few tens or hundreds data points at the time). Data size issue

Notes

Notes

Extras

Dot products



- Suppose $\|\mathbf{u}\| = 1$, i.e. we have a unit vector (of length one, this makes the point easier to see).
- We can think of \mathbf{v} as the sum of two components, one parallel (b) and another perpendicular (a) to \mathbf{u} .
- The dot product $\mathbf{u} \cdot \mathbf{v}$ gives you b , the projection of \mathbf{v} onto \mathbf{u} over all their dimensions.

Dot products in the perceptron model

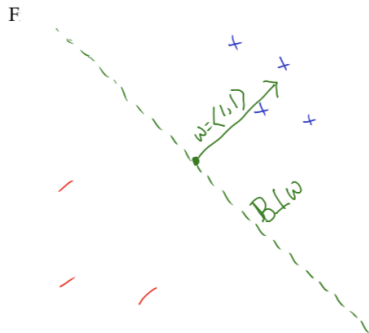


Figure 4.6: picture of data points with hyperplane and weight vector

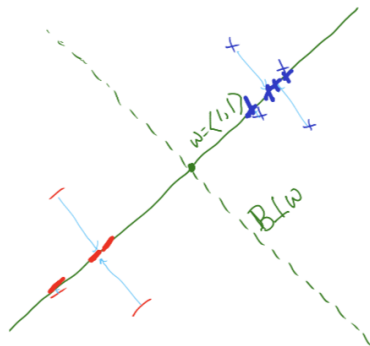


Figure 4.7: The same picture as before, but with projections onto weight vector; then, below, those points along a one-dimensional axis with zero marked.

Full derivation for linear regression

- Closed-form, with MSE loss and L_2 regularization:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \hat{\mathbf{y}}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}$$

$$(\text{put equal to zero}) \rightarrow \mathbf{X}^T \mathbf{X} \mathbf{w} + \lambda \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$