Machine Learning concepts

Jelke Bloem

Text Mining
Amsterdam University College

March 18, 2025

Announcements

- **Individual assignment 2 deadline: 25/03, 23:59**
- **Reading assignment 3 on Transformer models: 28/03**

Questions/discussion reading assignment 2

- How does hierarchical softmax work?
- I would like to see how the CBOW would work if none of the previous nor future words are know to it, how would it be able to predict the current word then?
  - Random initialization
- In the paper of Mikolov et. al. they wrote that: "words can have multiple degrees of similarity". What are/can be those multiple degrees of similarities?
  - `https://aclanthology.org/N13-1090.pdf`
- Can the different models be combined? For instance, it seems to me generative AI works similarly to both the CBOW and Skip-gram models and neural networks.

Questions/discussion reading assignment 2

- I would want to know more about what are currently the most used ones, as I have used word embedding models from a few different libraries (Ollama, Huggingface, Chroma).
- Mikolov's paper mentions that data size one of the least researched aspects of DSMs. Why is this the case? What makes a certain aspect more or less appealing to research?
- Given that older, more interpretable model architectures can still perform well with modern techniques, could we use those for simple tasks that are being done more and more often with GPT-like models?
- Given the widespread adoption of Word2Vec, what are some of the unintended consequences or limitations in its application across various NLP tasks?

Questions/discussion reading assignment 2

- I was wondering about the CBOW accuracy in the rare words analysis. the results of one paper said it performs better than SGNS and the other said the opposite.
- When there is so much data available, in English, why use any model apart from CBOW or Skip-gram?
- To what extent can the architectures described in the paper by Mikolov et al help with other NLP tasks as sentiment analysis?
- What are the implications of this impactful research being done by a mega corporation like Google?

Questions/discussion reading assignment 2

- I would like to have more class discussions maybe reading some papers beforehand for class on Tuesdays and then create a space of debate, I would like to see how to apply this papers in real life, with real-life examples.

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.

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 $\mathcal{D}$:

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

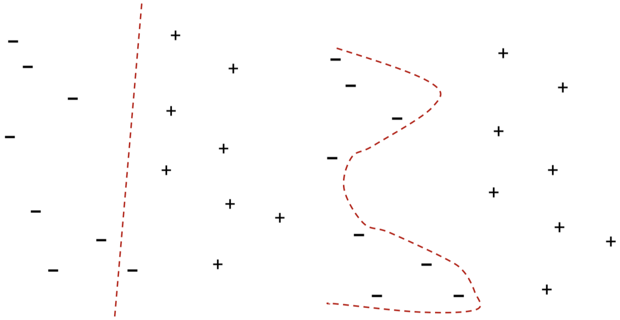
- But we do not know $\mathcal{D}$!
- Instead, we compute the **training error** $\hat{\epsilon}$ assuming it approximates ϵ:

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

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

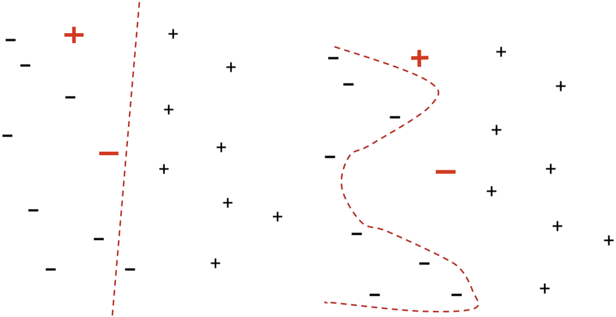
Key concepts: Underfitting and overfitting

- **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.



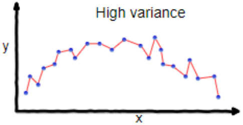
Key concepts: Underfitting and overfitting

- **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.

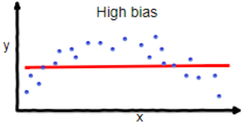


Key concepts: Underfitting and overfitting

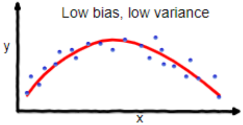
- Also referred to as **Bias-variance trade-off**.
- Note: this theory seems not to apply to deep learning! Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. “Reconciling Modern Machine-Learning Practice and the Classical Bias–Variance Trade-Off.” Proceedings of the National Academy of 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 $\boldsymbol{x} = \langle x_1, x_2, \dots, x_d \rangle, y \in \mathbb{R}$, $\forall(\boldsymbol{x}, y) \in \{X, Y\}$. 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):

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

$$\hat{y} = \boldsymbol{Xw}$$

→ Notebook 6: Linear regression (with Scikit-learn)

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**.

Regularization

- Left unconstrained, MSE can easily lead to a case of **overfitting**, e.g. by paying too much attention to **outliers** (*why?*).
- → Notebook 6: outlier example
- 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): $\lambda||\boldsymbol{w}||^2$
  - L_1-norm (Lasso): $\lambda|\boldsymbol{w}|$
- λ is a hyperparameter to control the intensity of the regularization.
  - Independent of the model and data
  - Penalty for exactly fitting the data

(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** $\bigtriangledown_w f$ is the vector consisting of the partial derivatives of this function w.r.t. each input coordinate:

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

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

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_w f$$

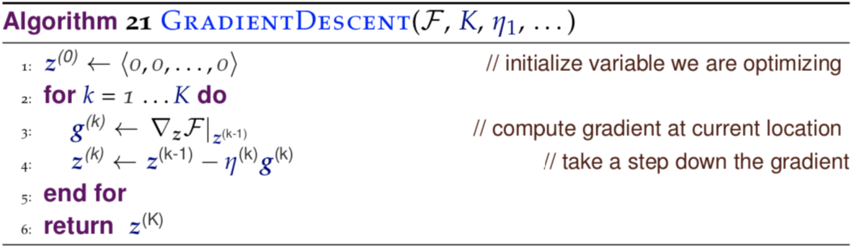
- η (eta) is called the **learning rate**. We refer to **stochastic** GD when we use one (or few) data point(s) at the time.

Stochastic Gradient Descent (SGD)

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

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_w f$$

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



```
Algorithm 21 GradientDescent($\mathcal{F}$, $K$, $\eta_1$, ...)
$z^{(0)} \leftarrow \langle 0, 0, \dots, 0 \rangle$                // initialize variable we are optimizing
for $k = 1 \dots K$ do
    $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$      // compute gradient at current location
    $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$          // take a step down the gradient
end for
return $z^{(K)}$
```

Credit: HD, ch. 7.

Stochastic Gradient Descent (SGD)

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

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_{w} f$$

For a single scalar:

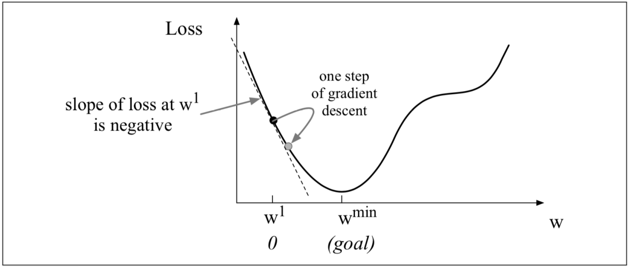


Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving *w* in the reverse direction from the slope of the function. Since the slope is negative, we need to move *w* in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means 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**:

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_{w} f$$

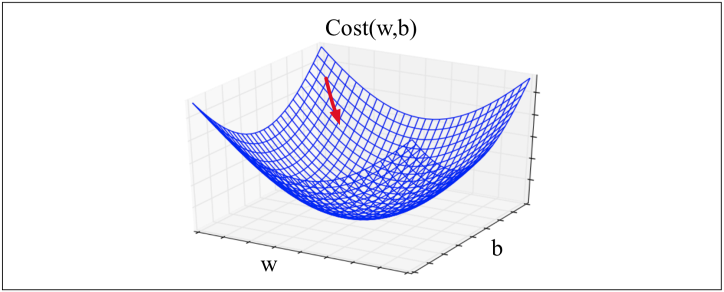


Figure 5.4 Visualization of the gradient vector in two dimensions *w* and *b*.

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**:

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_{w} f$$

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

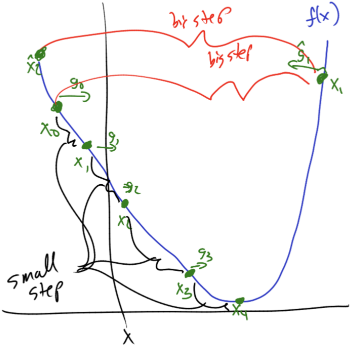
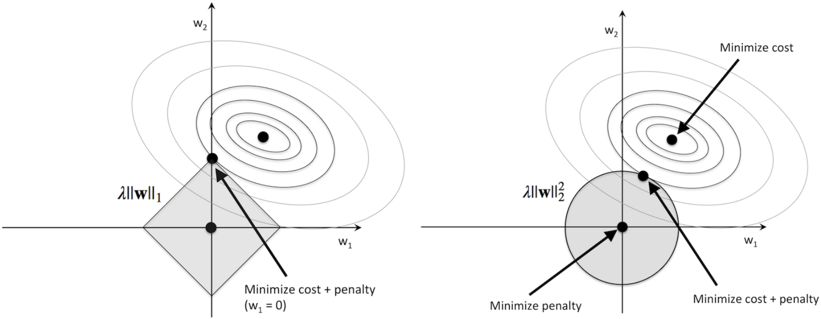


Figure 7.7: good and bad step sizes

Regularization and SGD



gray area = regularization constraint

Notation note: $||\boldsymbol{w}||_p = \left(\sum_d |\boldsymbol{w_d}|^p\right)^{\frac{1}{p}}$.
I have implied so far: $||\boldsymbol{w}||^2 = ||\boldsymbol{w}||_2^2$ and $|\boldsymbol{w}| = ||\boldsymbol{w}||_1$.

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

Putting everything together (SGD)

- SGD:

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_w f$$

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

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

- Regularized SGD for linear regression with MAE and L_2:

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \bigtriangledown_w \mathcal{L}$$

- 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

Midterm evaluation

Extras

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 $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_d \rangle, y \in \{-1, +1\}$, $\forall(\boldsymbol{x}, y) \in \{X, Y\}$. We thus have a binary classification problem.
- The **perceptron model** is defined by a linear combination of weights $\langle w_1, w_2, \ldots, w_d \rangle$ 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**.
  $l^{0/1} = \min_{\boldsymbol{w},b} \sum_n \mathbf{1}[y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) < 0]$
- 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. $a_1 = w_1^{(0)} x_{11} + w_2^{(0)} x_{12} = 1 - 1/2 = 1/2$. $sign(1/2) = +$, 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

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

$$w_1^{(2)} \leftarrow w_1^{(1)} + y_1 x_{11} = -3/2 + 1 = -1/2$$
$$w_2^{(2)} \leftarrow w_2^{(1)} + y_1 x_{12} = -3/2 + 2 = 1/2$$

2. $a_3 = w_1^{(1)} x_{31} + w_2^{(1)} x_{32} = -3/2 + 1/2 = -1.$ $sign(-1) = -$
3. $a_4 = w_1^{(1)} x_{41} + w_2^{(1)} x_{42} = -3/2 + 1 = -1/2.$ $sign(-1/2) = -$
4. $a_5 = w_1^{(1)} x_{51} + w_2^{(1)} x_{52} = -1/2 + 3/2 = 1.$ $sign(1) = +$
5. $a_6 = w_1^{(1)} x_{61} + w_2^{(1)} x_{62} = -2/2 + 3/2 = 1/2.$ $sign(1/2) = +$

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 $\boldsymbol{w}$. 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.

Loss functions: Convexity

- With the perceptron, we used the so-called **0/1 loss**. $l^{0/1} = \min_{\boldsymbol{w},b} \sum_n \mathbf{1}[y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) < 0]$
- 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.**

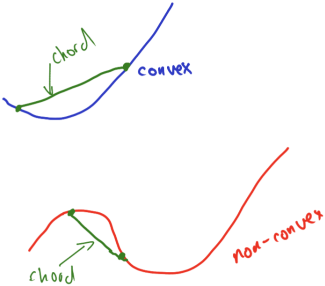


Figure 7.3: plot of convex and non-convex functions with two chords each

Credit: HD, ch. 7.

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:

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

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

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

Credit: HD, ch. 7.

Closed-form solution for MSE Linear regression

- We can express the loss as follows:

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

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

$$\begin{aligned} \bigtriangledown_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \boldsymbol{X}^T\Big(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\Big) = 0 \\ \boldsymbol{w} = \Big(\boldsymbol{X}^T\boldsymbol{X}\Big)^{-1}\boldsymbol{X}^T\boldsymbol{y} \end{aligned}$$

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

Closed-form solution with regularization

- We can express the loss as follows:

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

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

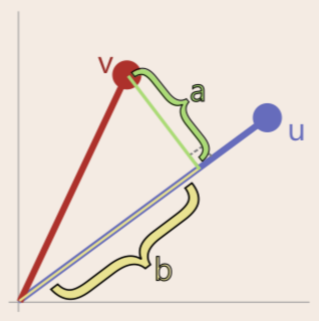
$$\begin{aligned} \bigtriangledown_{\boldsymbol{w}}|\mathcal{L}(\boldsymbol{w}) &= \boldsymbol{X}^T\Big(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\Big) + \lambda\boldsymbol{w} = 0 \\ \boldsymbol{w} &= \Big(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_D\Big)^{-1}\boldsymbol{X}^T\boldsymbol{y}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

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

Notes

Notes

Dot products



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

Dot products in the perceptron model

F

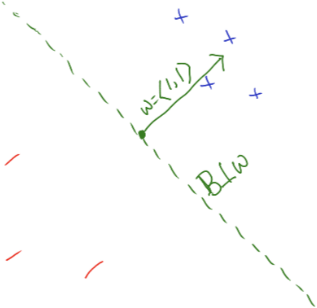


Figure 4.6: picture of data points with hyperplane and weight vector

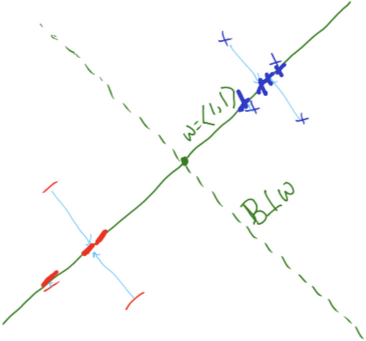


Figure 4.7: The same picture as before, but with projections onto weight vector; then, below, those points along a one-dimensional axis with zero marked.

Credit: HD, ch. 4.

Full derivation for linear regression

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

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{w}) &= \frac{1}{2}||\boldsymbol{X}\boldsymbol{w} - \hat{\boldsymbol{y}}||^2 + \frac{\lambda}{2}||\boldsymbol{w}||^2 \\
\nabla_w \mathcal{L}(\boldsymbol{w}) &= \boldsymbol{X}^T\Big(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\Big) + \lambda\boldsymbol{w} \\
\text{(put equal to zero)} \rightarrow \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} + \lambda\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{y} \\
\Big(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\Big)\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{y} \\
\boldsymbol{w} &= \Big(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\Big)^{-1}\boldsymbol{X}^T\boldsymbol{y}
\end{aligned}
$$