Lecture 2

Machine Learning Recap

Part 1: Linear models

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Outline:

What are “features”?

So far, we assumed $x^{(i)}$ is numerically encoded.

What are “features”?

So far, we assumed $x^{(i)}$ is numerically encoded.

But usually there are multiple possible ways to represent the same object.
We need to handle this.

$x =$ 🐧 $\rightarrow$ [culmen length (mm), culmen depth (mm)]

 🐧 $\rightarrow$ [culmen depth (km), weight (kg)]

What are “features”?

So far, we assumed $x^{(i)}$ is numerically encoded.

But usually there are multiple possible ways to represent the same object.
We need to handle this.

$x =$ 🐧 $\rightarrow$ [culmen length (mm), culmen depth (mm)]

 🐧 $\rightarrow$ [culmen depth (km), weight (kg)]

We call this a **feature representation**.

$$\boldsymbol{h}(x) = [\underbrace{\text{culmen length of x in mm}}_{=h_1(x)}, \underbrace{\text{culmen depth of x in mm}}_{=h_2(x)}] \in \mathbb{R}^2$$

Linear regression

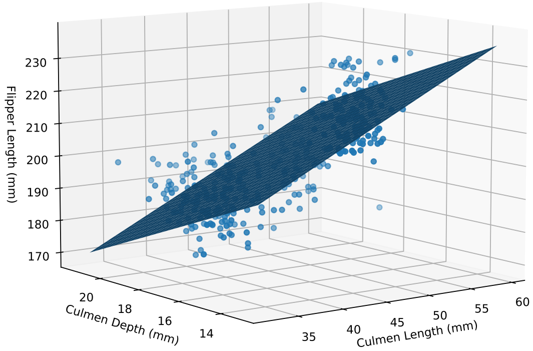
Numeric targets: $\mathcal{Y} = \mathbb{R}$

Penguins example:

- $\boldsymbol{h}(x) \in \mathbb{R}^2$, h_0 =cu. length, h_1 =cu. depth.
- y = flipper length.

model: $$f_\theta(x) = \underbrace{\boldsymbol{w} \cdot \boldsymbol{h}(x)}_{:=\sum_j w_j h_j(x)} + b$$

(parameters: $\theta = \{\boldsymbol{w}, b\}$)



Linear regression

Numeric targets: $\mathcal{Y} = \mathbb{R}$

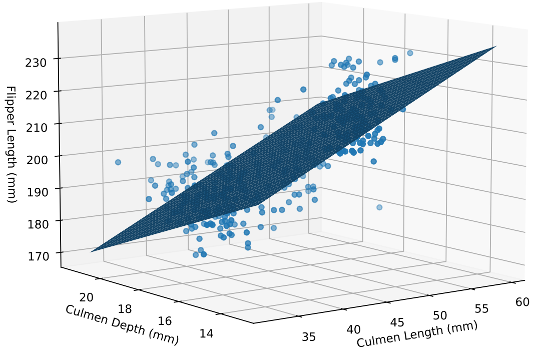
Penguins example:

- $\boldsymbol{h}(x) \in \mathbb{R}^2$, h_0 =cu. length, h_1 =cu. depth.
- y = flipper length.

$$\textbf{model:}\quad f_\theta(x) = \underbrace{\boldsymbol{w} \cdot \boldsymbol{h}(x)}_{:=\sum_j w_j h_j(x)} + b$$

(parameters: $\theta = \{\boldsymbol{w}, b\}$)

$$\textbf{prediction rule:}\quad \hat{y} = f_\theta(x)$$



Linear regression

Numeric targets: $\mathcal{Y} = \mathbb{R}$

Penguins example:

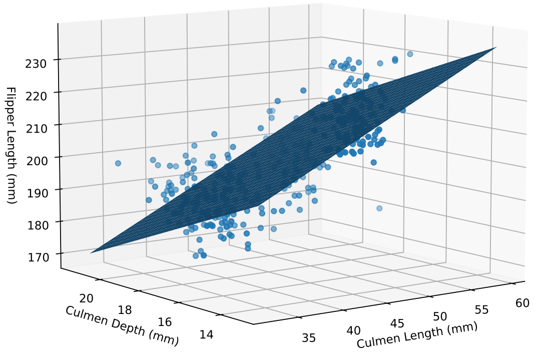
- $\boldsymbol{h}(x) \in \mathbb{R}^2$, h_0 =cu. length, h_1 =cu. depth.
- y = flipper length.

model: $f_\theta(x) = \underbrace{\boldsymbol{w} \cdot \boldsymbol{h}(x)}_{:=\sum_j w_j h_j(x)} + b$

(parameters: $\theta = \{\boldsymbol{w}, b\}$)

prediction rule: $\hat{y} = f_\theta(x)$

evaluation: **1.** mean squared error
MSE$= \frac{1}{N} \sum_i (\hat{y}^{(i)} - y^{(i)})^2$



Linear regression

Numeric targets: $\mathcal{Y} = \mathbb{R}$

Penguins example:

- $\boldsymbol{h}(x) \in \mathbb{R}^2$, h_0 =cu. length, h_1 =cu. depth.
- y = flipper length.

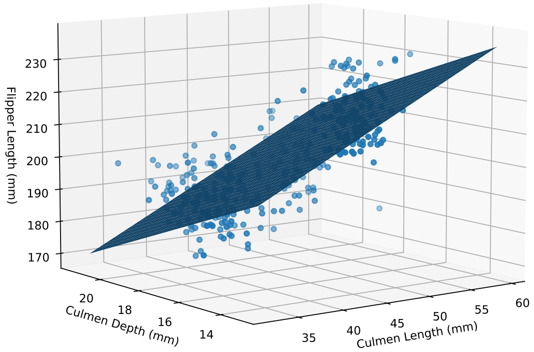
model: $f_\theta(x) = \underbrace{\boldsymbol{w} \cdot \boldsymbol{h}(x)}_{:=\sum_j w_j h_j(x)} + b$

(parameters: $\theta = \{\boldsymbol{w}, b\}$)

prediction rule: $\hat{y} = f_\theta(x)$

evaluation:

1. mean squared error
   MSE= $\frac{1}{N} \sum_i (\hat{y}^{(i)} - y^{(i)})^2$
2. root mse, RMSE=$\sqrt{\text{MSE}}$



Linear regression

Numeric targets: $\mathcal{Y} = \mathbb{R}$

Penguins example:

- $\boldsymbol{h}(x) \in \mathbb{R}^2$, h_0 =cu. length, h_1 =cu. depth.
- y = flipper length.

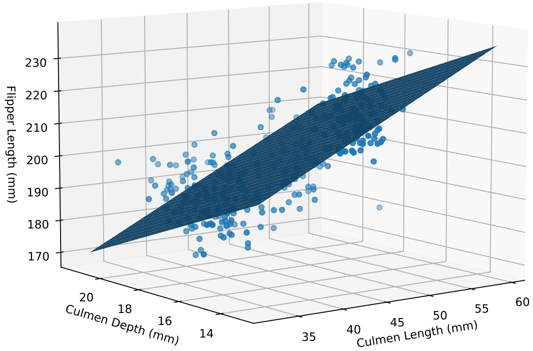
model: $f_\theta(x) = \underbrace{\boldsymbol{w} \cdot \boldsymbol{h}(x)}_{:=\sum_j w_j h_j(x)} + b$

(parameters: $\theta = \{\boldsymbol{w}, b\}$)

prediction rule: $\hat{y} = f_\theta(x)$

evaluation:

1. mean squared error MSE= $\frac{1}{N} \sum_i (\hat{y}^{(i)} - y^{(i)})^2$
2. root mse, RMSE=$\sqrt{\text{MSE}}$
3. mean absolute error MAE=$\frac{1}{N} \sum_i |\hat{y}^{(i)} - y^{(i)}|$



Linear regression

Numeric targets: $\mathcal{Y} = \mathbb{R}$

Penguins example:

- $\boldsymbol{h}(x) \in \mathbb{R}^2$, h_0 =cu. length, h_1 =cu. depth.
- y = flipper length.

model: $f_\theta(x) = \underbrace{\boldsymbol{w} \cdot \boldsymbol{h}(x)}_{:=\sum_j w_j h_j(x)} + b$

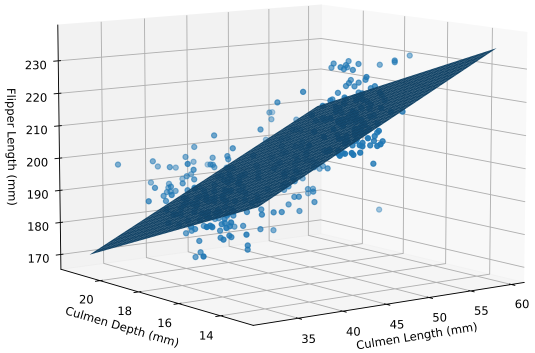
(parameters: $\theta = \{\boldsymbol{w}, b\}$)

prediction rule: $\hat{y} = f_\theta(x)$

evaluation:

1. mean squared error
   MSE= $\frac{1}{N}\sum_i (\hat{y}^{(i)} - y^{(i)})^2$
2. root mse, RMSE=$\sqrt{\text{MSE}}$
3. mean absolute error
   MAE=$\frac{1}{N}\sum_i |\hat{y}^{(i)} - y^{(i)}|$

loss: $L_{\text{SE}}(\hat{y}, y) = (\hat{y} - y)^2$



Fitting linear regression

To keep things shorter, let's fix $b = 0$ and write $\boldsymbol{x} := \boldsymbol{h}(x)$

$$\text{minimize}_{\boldsymbol{w}} \left\{ \mathcal{L}(\boldsymbol{w}) := \frac{1}{N} \sum_i \underbrace{(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} - y^{(i)})^2}_{L_{\text{SE}}(\hat{y}^{(i)}, y^{(i)})} \quad + \quad \alpha \quad \underbrace{\|\boldsymbol{w}\|^2}_{\text{regularizer}} \right\}$$

Fitting linear regression

To keep things shorter, let's fix $b = 0$ and write $\boldsymbol{x} := \boldsymbol{h}(x)$

$$\text{minimize}_{\boldsymbol{w}} \left\{ \mathcal{L}(\boldsymbol{w}) := \frac{1}{N} \sum_i \underbrace{(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} - y^{(i)})^2}_{L_{\text{SE}}(\hat{y}^{(i)}, y^{(i)})} \quad + \quad \alpha \quad \underbrace{\|\boldsymbol{w}\|^2}_{\text{regularizer}} \right\}$$

Method 1: work it out by hand

$$\begin{aligned} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) &= \frac{2}{N} \sum_i (\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} - y^{(i)}) \boldsymbol{x}^{(i)} + 2\alpha \boldsymbol{w} \\ &= \frac{2}{N} \boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + 2\alpha \boldsymbol{w}. \end{aligned}$$

At optimum, the gradient must be zero:

$$\begin{aligned} (\boldsymbol{X}^\top \boldsymbol{X} + \alpha N \boldsymbol{I}) \boldsymbol{w} &= \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{w} &= (\boldsymbol{X}^\top \boldsymbol{X} + \alpha N \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \end{aligned}$$

Fitting linear regression

To keep things shorter, let's fix $b = 0$ and write $\boldsymbol{x} := \boldsymbol{h}(x)$

$$\text{minimize}_{\boldsymbol{w}} \left\{ \mathcal{L}(\boldsymbol{w}) := \frac{1}{N} \sum_i \underbrace{(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} - y^{(i)})^2}_{L_{\text{SE}}(\hat{y}^{(i)}, y^{(i)})} + \alpha \underbrace{\|\boldsymbol{w}\|^2}_{\text{regularizer}} \right\}$$

Method 1: work it out by hand

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \frac{2}{N} \sum_i (\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} - y^{(i)}) \boldsymbol{x}^{(i)} + 2\alpha \boldsymbol{w}$$
$$= \frac{2}{N} \boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + 2\alpha \boldsymbol{w}.$$

At optimum, the gradient must be zero:

$$(\boldsymbol{X}^\top \boldsymbol{X} + \alpha N \boldsymbol{I}) \boldsymbol{w} = \boldsymbol{X}^\top \boldsymbol{y}$$
$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha N \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}.$$

Method 2: gradient-based optimization

pick $\boldsymbol{w}^{(0)}$, step size sequence $\eta^{(t)}$

repeat until converged or tired:

$$\boldsymbol{w}^{(t+1)} \leftarrow \eta^{(t)} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^{(t)})$$

stochastic/mini-batch version: replace $\nabla_w \mathcal{L}$ with an approx computed on one or a few data points.

Classification

Binary case: $\mathcal{Y} = \{0, 1\}$

Same model $f(x) = \boldsymbol{w} \cdot \boldsymbol{h}(x) + b$;

Classification

Binary case: $\mathcal{Y} = \{0, 1\}$

Same model $f(x) = \boldsymbol{w} \cdot \boldsymbol{h}(x) + b$; Prediction rule: $\hat{y} = \begin{cases} 1, & f(x) \geq 0 \\ 0, & f(x) < 0 \end{cases}$

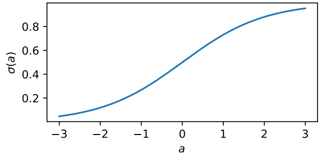
Classification

Binary case: $\mathcal{Y} = \{0, 1\}$

Same model $f(x) = \boldsymbol{w} \cdot \boldsymbol{h}(x) + b$; Prediction rule: $\hat{y} = \begin{cases} 1, & f(x) \geq 0 \\ 0, & f(x) < 0 \end{cases}$

A probabilistic approach: writing $a = f(x)$,

$$\begin{cases} \Pr(Y = 1|x) = \sigma(a) \\ \Pr(Y = 0|x) = 1 - \sigma(a) \end{cases} \quad \text{where } \sigma(a) := \frac{1}{1 + e^{-a}}$$



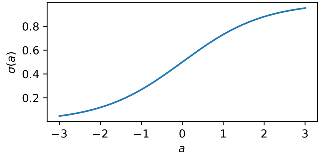
Classification

Binary case: $\mathcal{Y} = \{0, 1\}$

Same model $f(x) = \boldsymbol{w} \cdot \boldsymbol{h}(x) + b$; Prediction rule: $\hat{y} = \begin{cases} 1, & f(x) \geq 0 \\ 0, & f(x) < 0 \end{cases}$

A probabilistic approach: writing $a = f(x)$,

$$\begin{cases} \Pr(Y = 1|x) = \sigma(a) \\ \Pr(Y = 0|x) = 1 - \sigma(a) \end{cases} \qquad \text{where } \sigma(a) := \frac{1}{1 + e^{-a}}$$



Maximizing the probability over an entire dataset:

$$\Pr(Y^{(1)} = y^{(1)}, \ldots, Y^{(N)} = y^{(N)}|x^{(1)}, \ldots, x^{(N)}) = \prod_{i=1}^{N} \Pr(Y^{(i)} = y^{(i)}|x^{(i)})$$ (because iid)

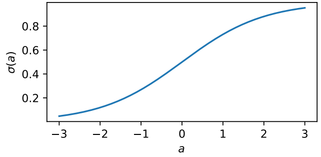
Classification

Binary case: $\mathcal{Y} = \{0, 1\}$

Same model $f(x) = \boldsymbol{w} \cdot \boldsymbol{h}(x) + b$; Prediction rule: $\hat{y} = \begin{cases} 1, & f(x) \geq 0 \\ 0, & f(x) < 0 \end{cases}$

A probabilistic approach: writing $a = f(x)$,

$$\begin{cases} \Pr(Y = 1|x) = \sigma(a) \\ \Pr(Y = 0|x) = 1 - \sigma(a) \end{cases} \quad \text{where } \sigma(a) := \frac{1}{1 + e^{-a}}$$



Maximizing the probability over an entire dataset:

$$\Pr(Y^{(1)} = y^{(1)}, \ldots, Y^{(N)} = y^{(N)} | x^{(1)}, \ldots, x^{(N)}) = \prod_{i=1}^{N} \Pr(Y^{(i)} = y^{(i)} | x^{(i)})$$ (because iid)

We like sums more than products, and minimize rather than maximize, so:

$$-\log \Pr(Y^{(1)}, \ldots | x^{(1)}, \ldots) = \sum_{i=1}^{N} -\log \Pr(Y^{(i)} | x^{(i)})$$

Classification

Binary logistic regression

This negative log-probability is called the logistic loss or binary cross-entropy:

$$L_{\mathrm{LG}}(a, y) = \begin{cases} -\log \sigma(a), & y = 1 \\ -\log 1 - \sigma(a), & y = 0 \end{cases} = \begin{cases} \log(1 + \exp(-a)), & y = 1 \\ \log(1 + \exp(a)), & y = 0. \end{cases}$$

Classification

Binary logistic regression

This negative log-probability is called the logistic loss or binary cross-entropy:

$$L_{\mathrm{LG}}(a, y) = \begin{cases} -\log \sigma(a), & y = 1 \\ -\log 1 - \sigma(a), & y = 0 \end{cases} = \begin{cases} \log(1 + \exp(-a)), & y = 1 \\ \log(1 + \exp(a)), & y = 0. \end{cases}$$

Logistic regression:

$$\underset{\boldsymbol{w}}{\text{minimize}} \sum_i L_{\mathrm{LG}}(\boldsymbol{w} \cdot \boldsymbol{h}(x^{(i)}) + b, y^{(i)}) + \alpha \|\boldsymbol{w}\|^2$$

No closed-form solution available.

Must do some form of gradient-based optimization.

Classification

Multi-class case: $\mathcal{Y} = \{1, \dots, K\}$

Why do we use $\sigma(a)$ in the binary case?

- squish to $(0, 1)$
- symmetry: $\sigma(a) + \sigma(-a) = 1$.

Classification

Multi-class case: $\mathcal{Y} = \{1, \ldots, K\}$

Why do we use $\sigma(a)$ in the binary case?

- squish to $(0, 1)$
- symmetry: $\sigma(a) + \sigma(-a) = 1$.

Proof:

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

$$\sigma(-a) = \frac{1}{1+e^{a}} \cdot \frac{e^{-a}}{e^{-a}} = \frac{e^{-a}}{e^{-a}+e^{0}} = \frac{e^{-a}}{1+e^{-a}}.$$

Classification

Multi-class case: $\mathcal{Y} = \{1, \ldots, K\}$

Why do we use $\sigma(a)$ in the binary case?

- squish to $(0, 1)$
- symmetry: $\sigma(a) + \sigma(-a) = 1$.

Extension to K classes:

$$a = [a_1, \ldots, a_K]$$

$$\operatorname{softmax}(a) = \left[\frac{e^{a_1}}{Z}, \ldots, \frac{e^{a_K}}{Z}\right], \qquad Z = \sum_i e^{a_i}.$$

Classification

Multi-class case: $\mathcal{Y} = \{1, \ldots, K\}$

Why do we use $\sigma(a)$ in the binary case?

- squish to $(0, 1)$
- symmetry: $\sigma(a) + \sigma(-a) = 1.$

Extension to K classes:

$$a = [a_1, \ldots, a_K]$$

$$\text{softmax}(a) = \left[\frac{e^{a_1}}{Z}, \ldots, \frac{e^{a_K}}{Z}\right], \qquad Z = \sum_i e^{a_i}.$$

Multi-class logistic regression:

model: $\boldsymbol{f}(x) = \boldsymbol{W}\boldsymbol{h}(x) + \boldsymbol{b} = [\boldsymbol{w}_1^\top \boldsymbol{h}(x) + b_1, \ldots, \boldsymbol{w}_K^\top \boldsymbol{h}(x) + b_K], \qquad (\theta = \{\boldsymbol{W}, \boldsymbol{b}\})$

Classification

Multi-class case: $\mathcal{Y} = \{1, \ldots, K\}$

Why do we use $\sigma(a)$ in the binary case?

- squish to $(0, 1)$
- symmetry: $\sigma(a) + \sigma(-a) = 1.$

Extension to K classes:

$$a = [a_1, \ldots, a_K]$$

$$\text{softmax}(a) = \left[\frac{e^{a_1}}{Z}, \ldots, \frac{e^{a_K}}{Z}\right], \qquad Z = \sum_i e^{a_i}.$$

Multi-class logistic regression:

model: $\boldsymbol{f}(x) = \boldsymbol{W}\boldsymbol{h}(x) + \boldsymbol{b} = [\boldsymbol{w}_1^\top \boldsymbol{h}(x) + b_1, \ldots, \boldsymbol{w}_K^\top \boldsymbol{h}(x) + b_K], \qquad (\theta = \{\boldsymbol{W}, \boldsymbol{b}\})$

loss: $L_{\text{LR}}(\boldsymbol{a}, y) = -\log \Pr(Y = y|x) = -a_y + \log \sum_{k=1}^{K} \exp a_k.$

Classification

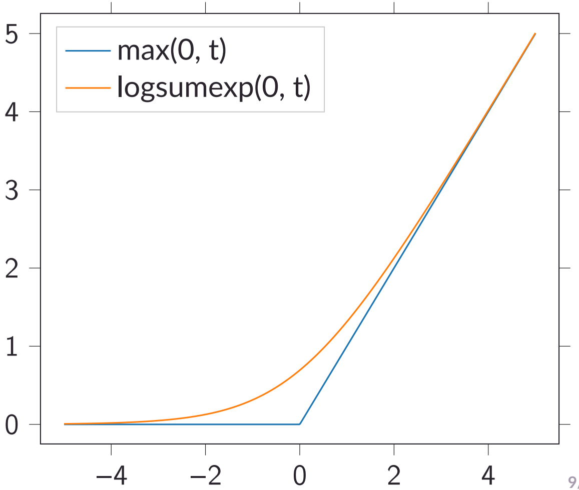
The Perceptron

An even simpler classifier.

$$\boldsymbol{a} = f(x)$$

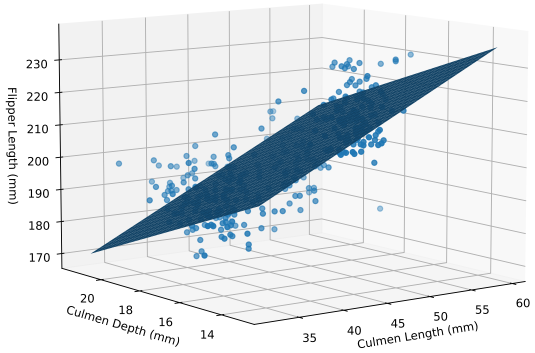
$$L_{\text{Perc}}(\boldsymbol{a}, y) = -a_y + \max_{k \in 1,\ldots,K} a_k$$

compare $L_{\text{LR}}(\boldsymbol{a}, y) = -a_y + \log \sum_k \exp a_j$



Linear versus NN regression

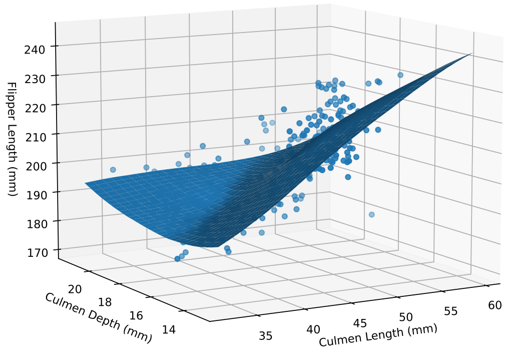
Penguins: $x \in \mathbb{R}^2$, h_1 =bill length, h_2 =bill depth. y = flipper length.



Linear model

$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{h}(x) + b$$

MAE= 6.83



One-hidden-layer NN

$$\hat{y} = \boldsymbol{w} \cdot \left(\mathrm{ReLU}\left(\boldsymbol{W}_1 \boldsymbol{h}(x) + \boldsymbol{b}_1\right)\right) + b$$

MAE= 5.56

Linear models summary

- Predict based on a linear function of the features.
- Efficient (fast) learning for regression and classification.
- Probabilistic interpretation.
- Limited expressivity means features must be well designed.

Outline:

Deep models

Instead of $f(x) = \boldsymbol{W} h(x) + b$ with fixed features, can we learn more/better features?

$$\theta = \{\boldsymbol{W}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{W}_m, \boldsymbol{b}_m, \boldsymbol{W}, \boldsymbol{b}\}$$

$$\begin{aligned}
\text{input} \quad & \boldsymbol{z}_0 = h_0(x) \\
\text{1st hidden layer} \quad & \boldsymbol{z}_1 = \phi\left(\boldsymbol{W}_1 \boldsymbol{z}_0 + \boldsymbol{b}_1\right) \\
\text{2nd hidden layer} \quad & \boldsymbol{z}_2 = \phi\left(\boldsymbol{W}_2 \boldsymbol{z}_1 + \boldsymbol{b}_2\right) \\
& \vdots \\
& \boldsymbol{z}_m = \phi\left(\boldsymbol{W}_m \boldsymbol{z}_{m-1} + \boldsymbol{b}_m\right) \\
\text{output} \quad & f(x) = \boldsymbol{a} = \boldsymbol{W} \boldsymbol{z}_m + \boldsymbol{b}
\end{aligned}$$

ϕ is a nonlinearity, e.g., ReLU

Deep models

Instead of $f(x) = \boldsymbol{W}h(x) + b$ with fixed features, can we learn more/better features?

$$
\begin{array}{rrcl}
\text{input} & \boldsymbol{z}_0 & = & h_0(x) \\
\text{1st hidden layer} & \boldsymbol{z}_1 & = & \phi\left(\boldsymbol{W}_1\boldsymbol{z}_0 + \boldsymbol{b}_1\right) \\
\text{2nd hidden layer} & \boldsymbol{z}_2 & = & \phi\left(\boldsymbol{W}_2\boldsymbol{z}_1 + \boldsymbol{b}_2\right) \\
 & & \vdots & \\
 & \boldsymbol{z}_m & = & \phi\left(\boldsymbol{W}_m\boldsymbol{z}_{m-1} + \boldsymbol{b}_m\right) \\
\text{output} & f(x) = \boldsymbol{a} & = & \boldsymbol{W}\boldsymbol{z}_m + \boldsymbol{b}
\end{array}
$$

$\theta = \{\boldsymbol{W}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{W}_m, \boldsymbol{b}_m, \boldsymbol{W}, \boldsymbol{b}\}$

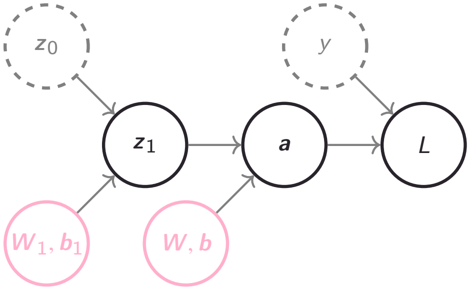
ϕ is a nonlinearity, e.g., ReLU

On top of this model, we could use any loss function we know.

- NN regression: $L_{\text{SE}}(a, y) = (a - y)^2$
- NN probabilistic classification: $L_{\text{LR}}(\boldsymbol{a}, y) = -a_y + \log \sum_{k=1}^{K} \exp a_k$

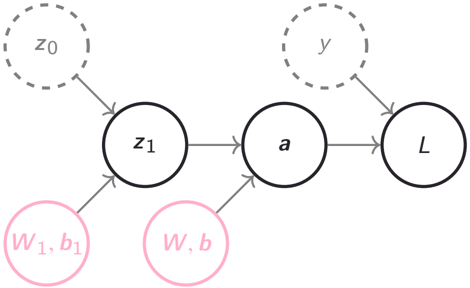
Computation graphs

Deep models require chaining many operations together. This forms a graph:



Computation graphs

Deep models require chaining many operations together. This forms a graph:



z_0
y
z_1
$\boldsymbol{a}$
L
$\boldsymbol{W}_1, \boldsymbol{b}_1$
$\boldsymbol{W}, \boldsymbol{b}$

This graph helps us compute gradients wrt parameters.

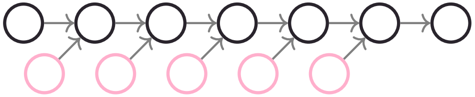
$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}$$
$$\frac{\partial L}{\partial \boldsymbol{W}_1} = \frac{\partial L}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{z}_1} \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{W}_1}$$

PyTorch & co do this automatically!

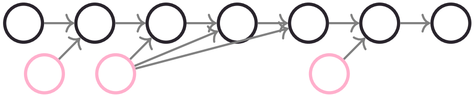
More complicated networks

If you took DSA, you'll remember graphs are quite flexible:

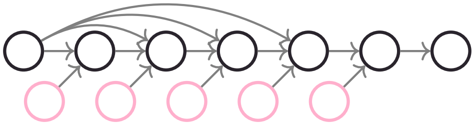
standard feed-forward



weight sharing



residual connections



Deep learning summary

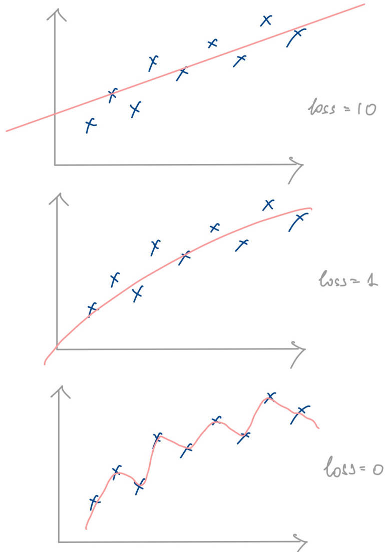
- Flexible paradigm for expressing complicated functions of the input.
- “Automated feature learning” instead of hand-crafting.
- …but now we must hand-craft a good neural network architecture.

Machine Learning for Structured Data

Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

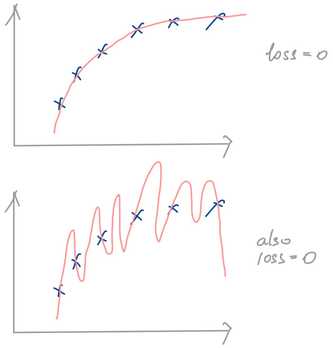
Outline:

Overfitting



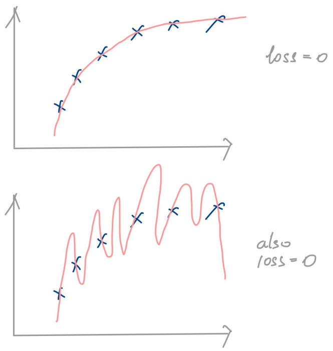
- The loss of a model on training data should not be taken as a good indicator of how good the model actually is.

Overfitting



- The loss of a model on training data should not be taken as a good indicator of how good the model actually is.
- Zero loss not necessarily bad. But many models have zero loss: some good, some bad.

Overfitting



- The loss of a model on training data should not be taken as a good indicator of how good the model actually is.
- Zero loss not necessarily bad. But many models have zero loss: some good, some bad.
- In ML, what we care about is generalization to unseen data. **Always evaluate performance on an held-out test set.**

Tuning

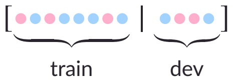
- In designing a ML model we have many choices to make: hyperparameters.
    - What model to use?
    - Which loss to use?
    - Regularization strength α
    - Number of hidden layers?
- Need a sound scientific strategy to evaluate which choices work well.
- Simplest correct way: have two held-out datasets:
    - a test set, for the final evaluation and reporting.
    - a development set, for comparing design choices while working.

Data splitting

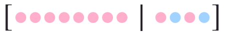
Even in the binary classification case, we have some complications.

Shuffle split

- Shuffle the data, leave out a subset.

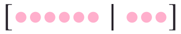


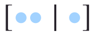
- What can happen if $y = 1$ is rare?



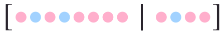
Stratified shuffle split

- Group data by label, split each separately in the same proportion.





- and combine:



Machine Learning for Structured Data

Vlad Niculae · LTL, UvA · `https://vene.ro/mlsd`

Outline:

Baselines

- Say we have a binary classification task and a classifier $f(x)$. We train and evaluate it and we get 80% validation accuracy. **Is this good?**

Baselines

- Say we have a binary classification task and a classifier $f(x)$. We train and evaluate it and we get 80% validation accuracy. **Is this good?**
- More details about the task: it is a lung xray pathology detection task, and 95% of the images are healthy. What can you say now?

Baselines

- Say we have a binary classification task and a classifier $f(x)$. We train and evaluate it and we get 80% validation accuracy. **Is this good?**
- More details about the task: it is a lung xray pathology detection task, and 95% of the images are healthy. What can you say now?
- A much simpler classifier $f_0(x) = \texttt{healthy}$ gets 95% accuracy with no training and two lines of Python code.

Baselines

- Say we have a binary classification task and a classifier $f(x)$. We train and evaluate it and we get 80% validation accuracy. **Is this good?**
- More details about the task: it is a lung xray pathology detection task, and 95% of the images are healthy. What can you say now?
- A much simpler classifier $f_0(x) = \texttt{healthy}$ gets 95% accuracy with no training and two lines of Python code.
- Ok, another scenario. You have a huge deep network that gets 64% accuracy. Is this good?

Baselines

- Say we have a binary classification task and a classifier $f(x)$. We train and evaluate it and we get 80% validation accuracy. **Is this good?**
- More details about the task: it is a lung xray pathology detection task, and 95% of the images are healthy. What can you say now?
- A much simpler classifier $f_0(x) = \texttt{healthy}$ gets 95% accuracy with no training and two lines of Python code.
- Ok, another scenario. You have a huge deep network that gets 64% accuracy. Is this good?
- What if a linear model gets 69%?

Baselines

- Say we have a binary classification task and a classifier $f(x)$. We train and evaluate it and we get 80% validation accuracy. **Is this good?**
- More details about the task: it is a lung xray pathology detection task, and 95% of the images are healthy. What can you say now?
- A much simpler classifier $f_0(x) = \texttt{healthy}$ gets 95% accuracy with no training and two lines of Python code.
- Ok, another scenario. You have a huge deep network that gets 64% accuracy. Is this good?
- What if a linear model gets 69%?
- **Always run simple baselines first.**

Baselines

- A good classification baseline: majority-class prediction.
  Does this baseline require training?

Baselines

- A good classification baseline: majority-class prediction.
  Does this baseline require training?
- The majority-class baseline is indeed trained: we select the majority class by looking only at training data, otherwise it's not fair.

Baselines

- A good classification baseline: majority-class prediction.
  Does this baseline require training?
- The majority-class baseline is indeed trained: we select the majority class by looking only at training data, otherwise it's not fair.
- What would be an equivalent of "majority prediction" for regression problems where the outputs are continuous?
  We want a constant predictor $f_0(x) = b$, but what do we set b to?

Baselines

- A good classification baseline: majority-class prediction.
  Does this baseline require training?
- The majority-class baseline is indeed trained: we select the majority class by looking only at training data, otherwise it's not fair.
- What would be an equivalent of "majority prediction" for regression problems where the outputs are continuous?
  We want a constant predictor $f_0(x) = b$, but what do we set b to?
- Let's *train* b as a parameter, to minimize training MSE!

$$\begin{aligned} b_* &= \arg\min_{b \in \mathbb{R}} L(b), \qquad \text{where} \quad L(b) := \sum_i .5(y^{(i)} - b)^2 \\ \nabla L(b_*) &= \sum_i (y^{(i)} - b_*) = 0 \\ \text{this means} \quad \sum_i y^{(i)} &= N b_*, \qquad \text{so} \quad b_* = \frac{\sum_i y^{(i)}}{N}. \end{aligned} \tag{1}$$

Baselines

- The constant prediction baseline for regression (trained to minimize MSE) should always predict the mean of the **training labels**.
- Common mistake: predict training mean on training, validation mean on validation, test mean on test: this is not a fair baseline, because model parameters (including b) should be trained on training data only.
- You may find this constant prediction baseline in scikit-learn under the name *DummyRegressor*: `https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor.html`

Negative results

- A more powerful / more expressive / better motivated model will not necessarily work better on the test set.
- This happens (and will happen to you many times in your career.) Not every good idea will perform well on every problem and metric.
- A more powerful model should always fit the training data better (i.e., higher train acc), but, unless tuned and regularized very carefully, might fail to generalize well by overfitting to noisy phenomena in the training data.
- But this doesn't mean a model that didn't work is a bad idea or never works. That is a much stronger hypothesis that needs extensive evidence.

From scratch vs. existing modules

- Should you implement ML code from scratch or reuse libraries?
- Both need skill! Understanding APIs and documentation takes time.
- IMO: For real work, prefer to use existing code, under the following conditions:
    - **The code is high quality and actively maintained.**
      (Avoid random blog posts and github repos with zero issues on them!)
    - **You understand what the code does and how it works.**
      (You should be able to explain, e.g., why you chose a CNN over a RNN). Many people understand ML models better if they implement them from scratch once. But treat this as a learning exercise.

Summary

1. **Linear models**

   Features

   Regression

   Classification

2. **Deep models**

3. **Tuning**

4. **Baselines and negative results**