Information Retrieval & Data Mining [COMP0084]

Introduction to machine learning & data mining – Part 2

Vasileios Lampos

Computer Science, UCL

@lampos

Preliminaries

- In this lecture:
  - Introduction to machine learning — Part 2
  - Supervised learning: regression, classification
  - Unsupervised learning: clustering
  - With examples!

- Useful additional reads
  - Chapters 2, 4 of "*Web Data Mining*" by Bing Liu (2006)
  - Chapters 3, 4, 14 of "*The Elements of Statistical Learning*" by Hastie, Tibshirani, and Friedman (2008)
  - Chapter 5 of "*Speech and language processing*" by Jurafsky and Martin (2021)
  - *Advanced* — Flu rate prediction from web search activity, nature.com/articles/srep12760.pdf

- Some slides adapted from Bing Liu's course — cs.uic.edu/~liub/teach/cs583-fall-21/cs583.html

- Many slides were adapted from Prof. Emine Yilmaz's lectures in previous years

Machine learning

- Arthur Samuel (IBM, 1959): "*Machine learning is the field of study that gives the computer the ability to learn* (**a task**) *without being explicitly programmed.*"
  — credited for coining the term
  — although we are still explicitly programming them to learn!

- Tom Mitchell (CMU, 1998): "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*."
  — more formal definition
  — learning from experience (observations, data)

Notational conventions for this lecture

$x \in \mathbb{R}$ denotes a real-valued scalar

$\mathbf{x} \in \mathbb{R}^n$ denotes a real-value vector with n elements

$\mathbf{X} \in \mathbb{R}^{n \times m}$ denotes a real-valued matrix with n rows and m columns

$\mathbf{y} \in \mathbb{R}^m$ denotes m instances of a real valued response (or target) variable

$\hat{\mathbf{y}} \in \mathbb{R}^m$ denotes m inferences of a real valued response variable

$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$ denotes the L_p-norm of $\mathbf{x} \in \mathbb{R}^n, p \in \mathbb{N}_{>0}$

Learning from experience

- Experience is something tangible, i.e. an observation and eventually a data point, something that can take a numeric form
- $\mathbf{x}_i$ denotes a numeric interpretation of an input
  y_i denotes a numeric interpretation of an output

$< \mathbf{x}_i, y_i >$ is an observation / sample

model function

$$\begin{tikzcd}
\begin{array}{l} < \mathbf{x}_1, y_1 > \\ < \mathbf{x}_2, y_2 > \\ \ldots \\ < \mathbf{x}_n, y_n > \end{array} \arrow[r] & \boxed{\text{training}} \arrow[r] & f(\cdot)
\end{tikzcd}$$

loss / error function

$$\begin{tikzcd}
\begin{array}{l} < \mathbf{x}_{n+1}, \hat{y}_{n+1} = f\left(\mathbf{x}_{n+1}\right) > \\ \ldots \\ < \mathbf{x}_{n+m}, \hat{y}_{n+m} = f\left(\mathbf{x}_{n+m}\right) > \end{array} \arrow[r] & \boxed{\text{testing}} \arrow[r] & \mathscr{L}\left(\hat{\mathbf{y}}, \mathbf{y}\right)
\end{tikzcd}$$

Common machine learning categorisation

- **Supervised learning**
  Learn a mapping f from inputs $\mathbf{X}$ to outputs $\mathbf{y}$ – also can be expressed by $f : \mathbf{X} \rightarrow \mathbf{y}$
  – $\mathbf{X}$ are also called features, observations, covariates, predictors
  – $\mathbf{y}$ are also called labels, targets, responses, ground truth
  – $< \mathbf{X}, \mathbf{y} >$ can also be referred to as observations or samples

- **Unsupervised learning**
  No outputs associated with the input $\mathbf{X}$ – the task becomes to discover an underlying structure or patterns in $\mathbf{X}$

- **Reinforcement learning**
  The system or agent has to learn how to interact with its environment
  Policy: which action to take in response to an input $\mathbf{X}$
  Different from supervised learning because no definitive responses are given
  Only rewards – *learning with a critic as opposed to learning with a teacher*

Supervised learning

- **Regression**
  estimate / predict a continuous output / target variable

  i.e. learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \mathbb{R}^n$

  <u>Examples</u>: predict a time series trend (e.g. in finance), estimate the prevalence of a condition in epidemiology

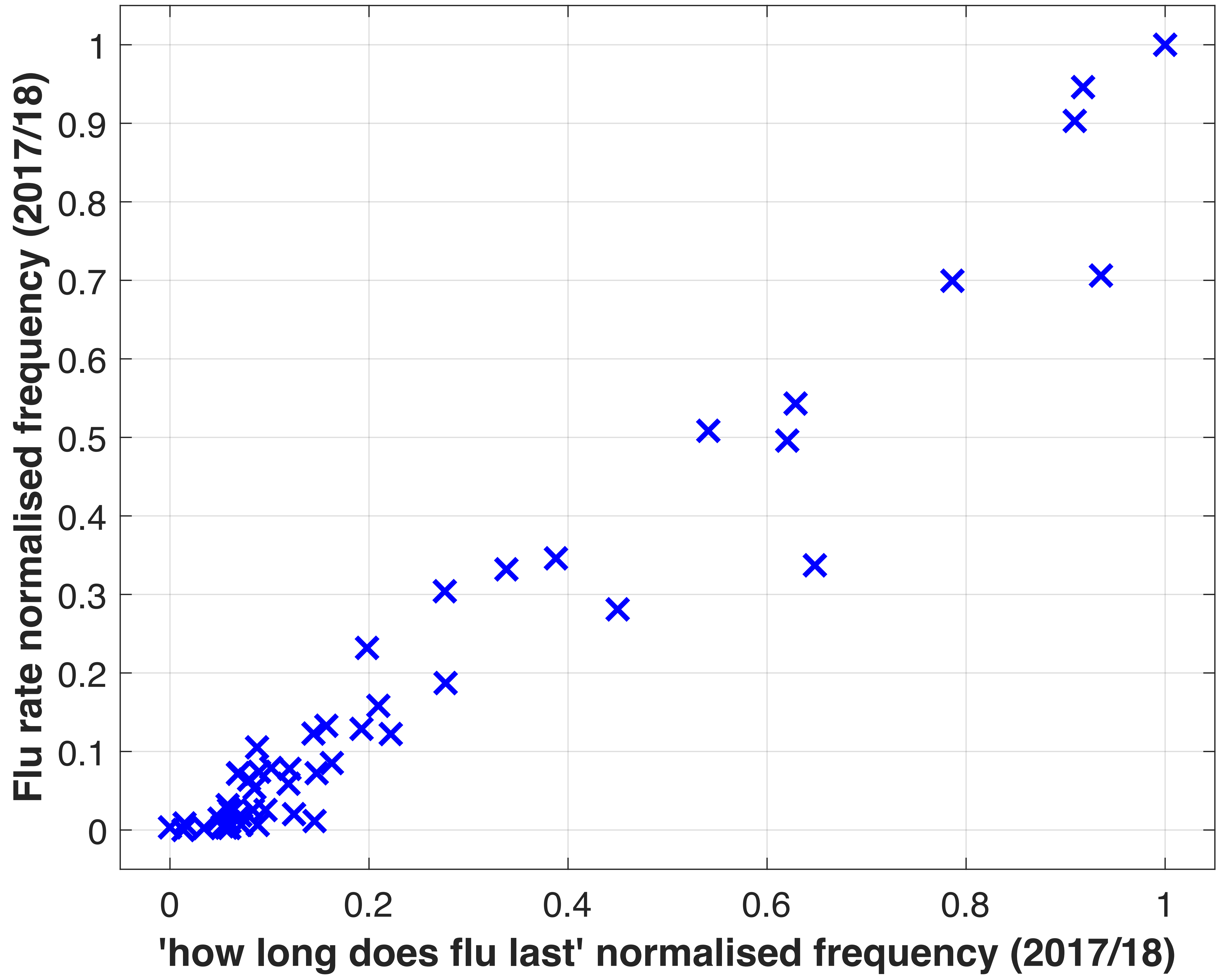
- **Classification**
  estimate a set of C unordered (and mutually exclusive) labels / classes

  i.e. learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \{1, 2, \ldots, C\}$

  <u>Examples</u>: detect spam email, medical imaging, text/document classification

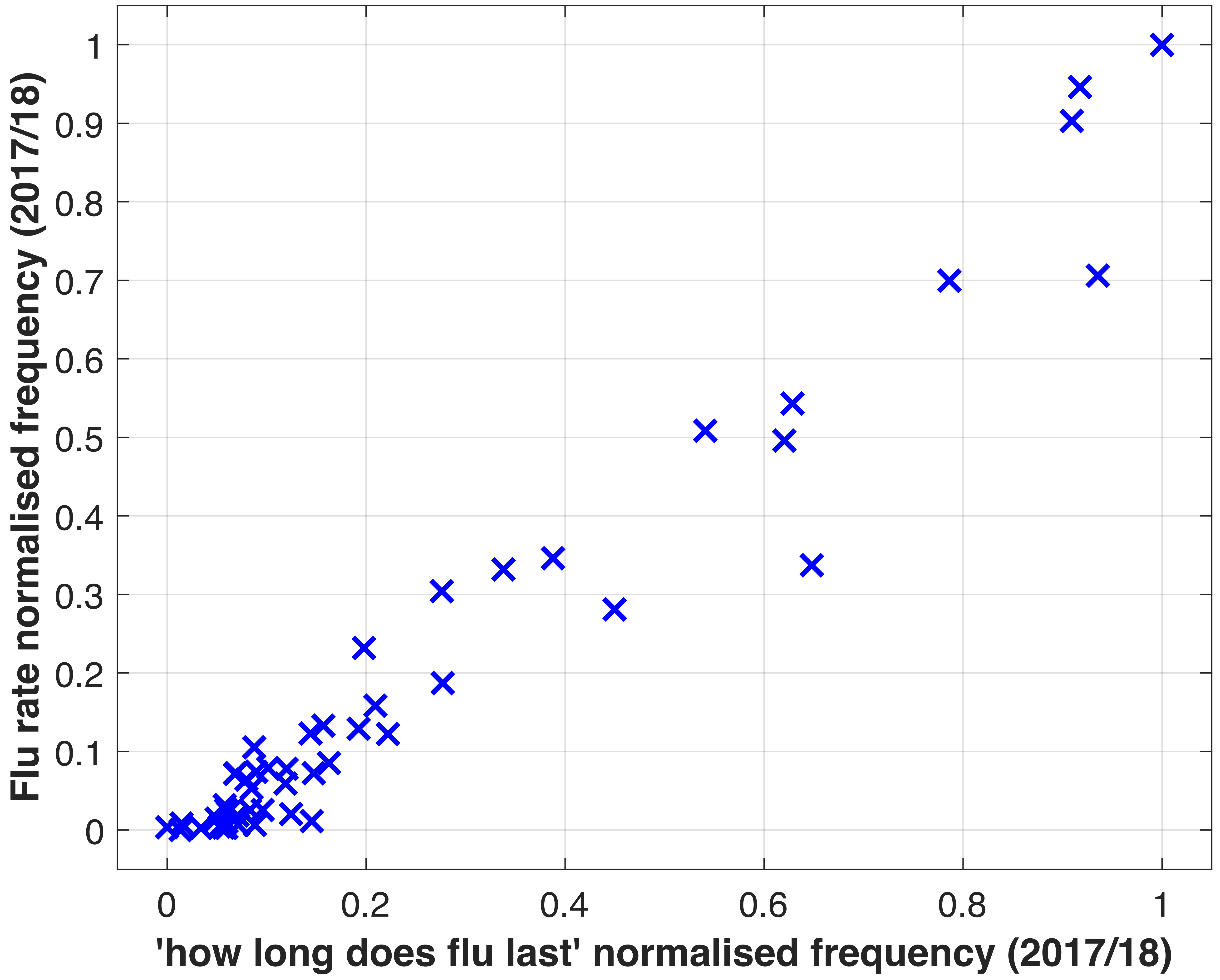
Supervised learning — Regression

- Estimate the **prevalence of influenza-like illness** in England based on the **frequency of the search query** “*how long does flu last*”



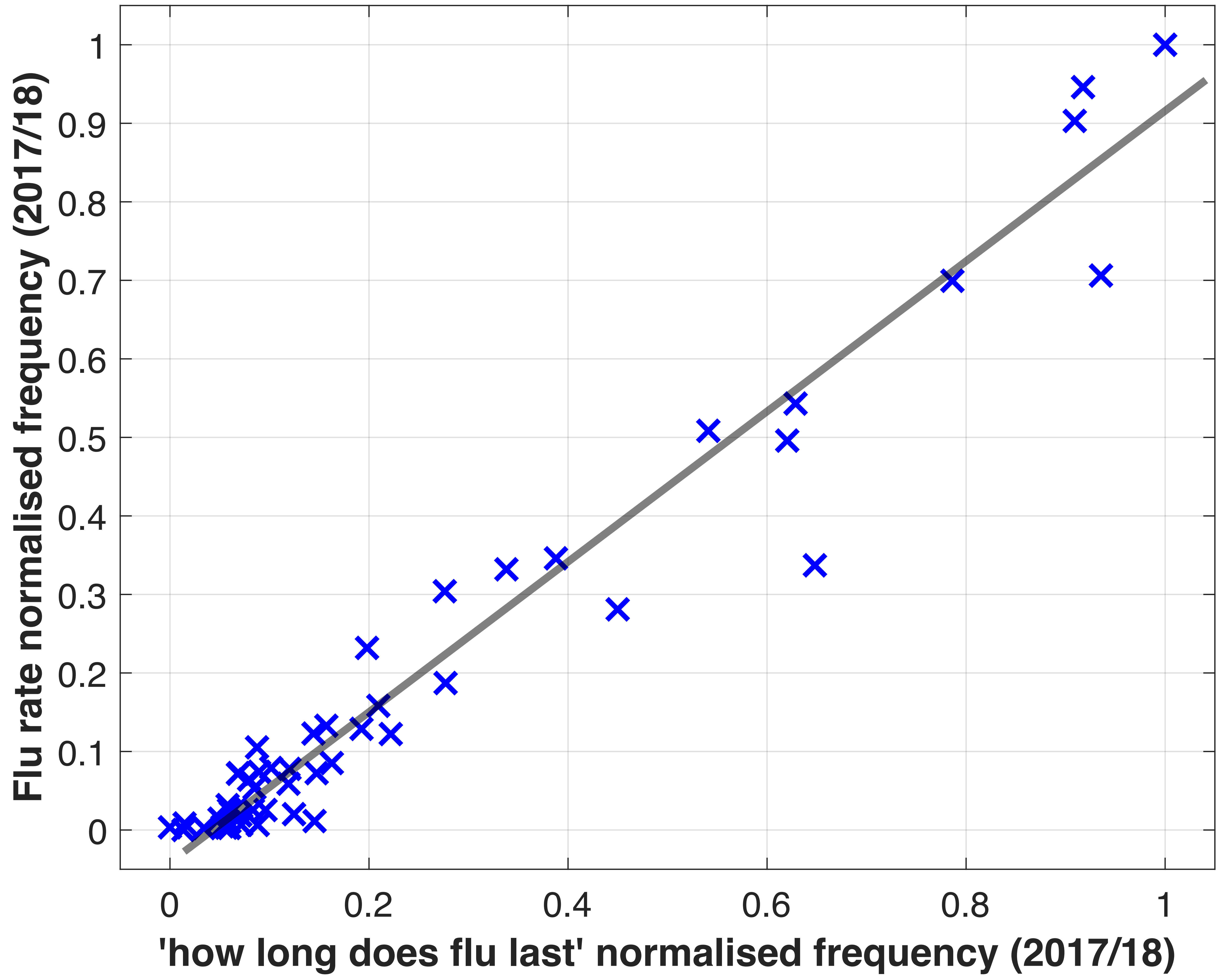
Supervised learning — Regression

- Estimate the **prevalence of influenza-like illness** in England based on the **frequency of the search query** "*how long does flu last*"
- Linearly related, bivariate correlation of 0.975
- Can we capture this relationship with a straight line?
- **Data:** dropbox.com/s/rgyg190whw26qrj/data-COMP0084-intro-to-ml.zip?dl=0



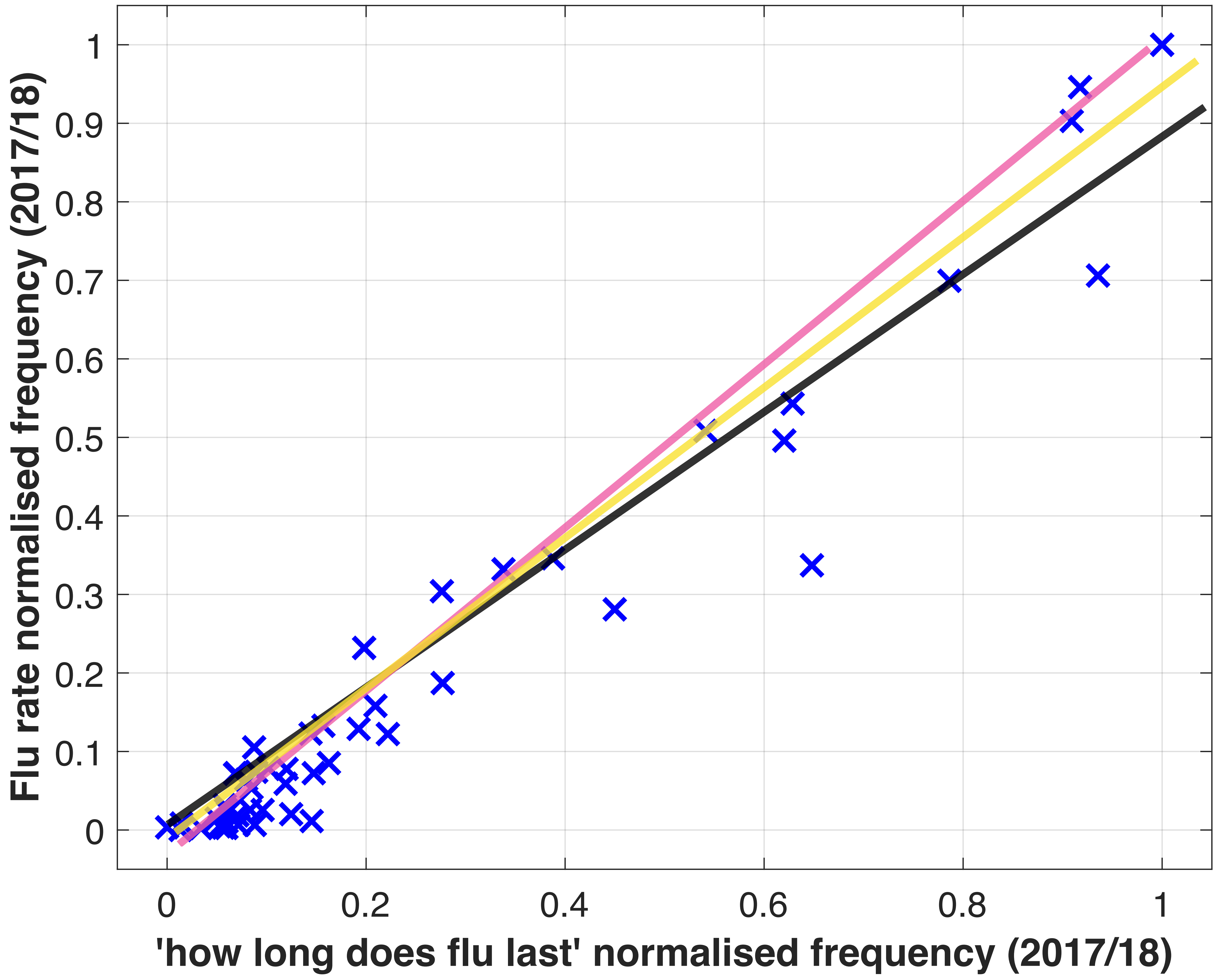
Supervised learning — Regression

- Estimate the **prevalence of influenza-like illness** in England based on the **frequency of the search query** “*how long does flu last*”
- Linearly related, bivariate correlation of 0.975
- Can we capture this relationship with a straight line?



Supervised learning — Regression

- Estimate the **prevalence of influenza-like illness** in England based on the **frequency of the search query** “*how long does flu last*”
- Linearly related, bivariate correlation of 0.975
- Can we capture this relationship with a straight line?
- Which line is the “best” though?

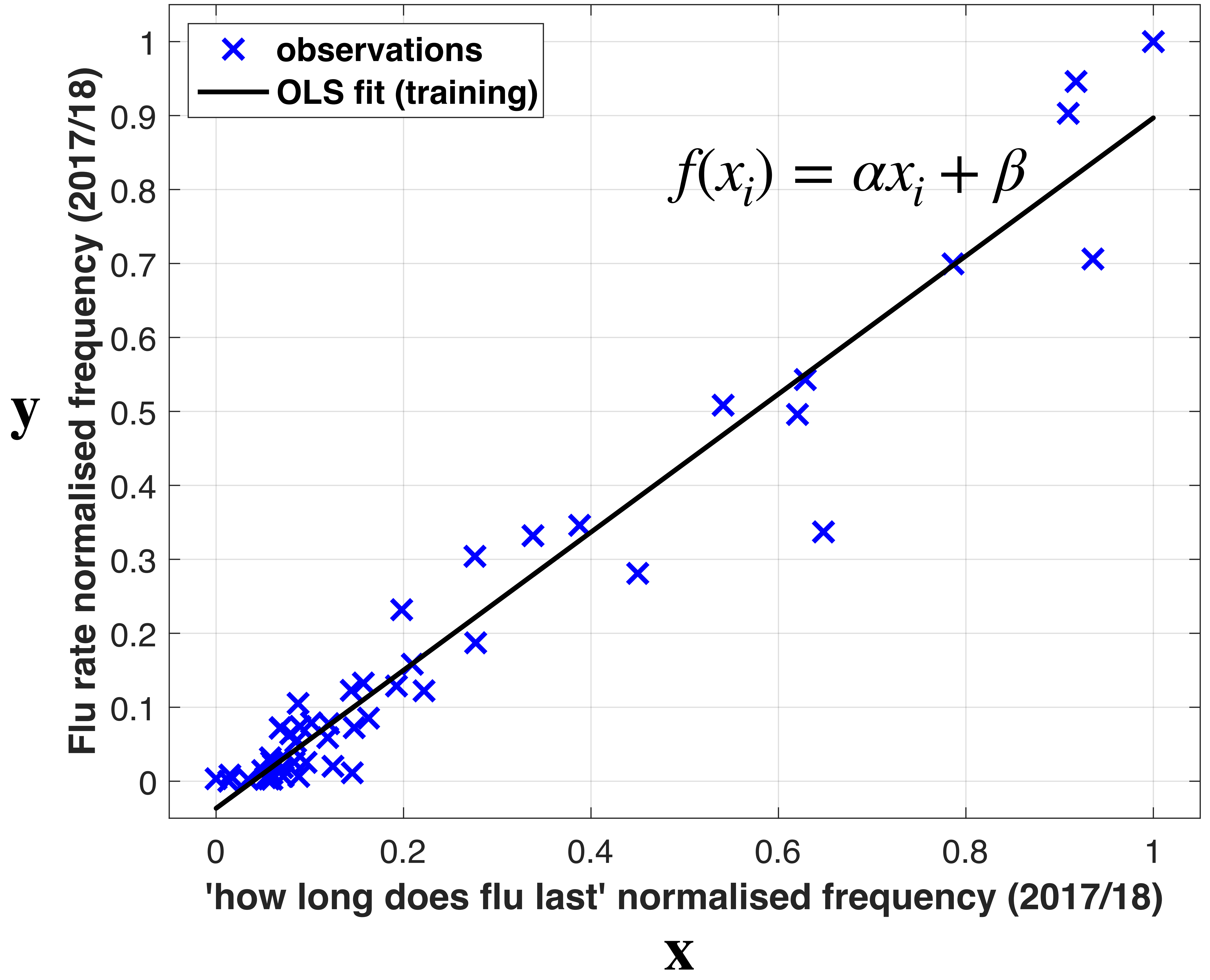


Supervised learning — Ordinary least squares (*linear*) regression

- $\mathbf{y}$ denotes the weekly influenza-like illness prevalence in England from September 2017 until the end of August 2018
- $\mathbf{x}$ denotes the corresponding weekly frequency of the search query “how long does flu last” (Google) for the same time period
- We want to learn a linear mapping f from the input $\mathbf{x}$ to the output $\mathbf{y}$ based on our current observations, i.e. for a weekly query frequency x_i, $f(x_i) = \hat{y}_i = \alpha x_i + \beta \approx y_i$
- This linear mapping has two unknown hyper-parameters: $\{\alpha, \beta\}$
- Find a line that best fits to our observations

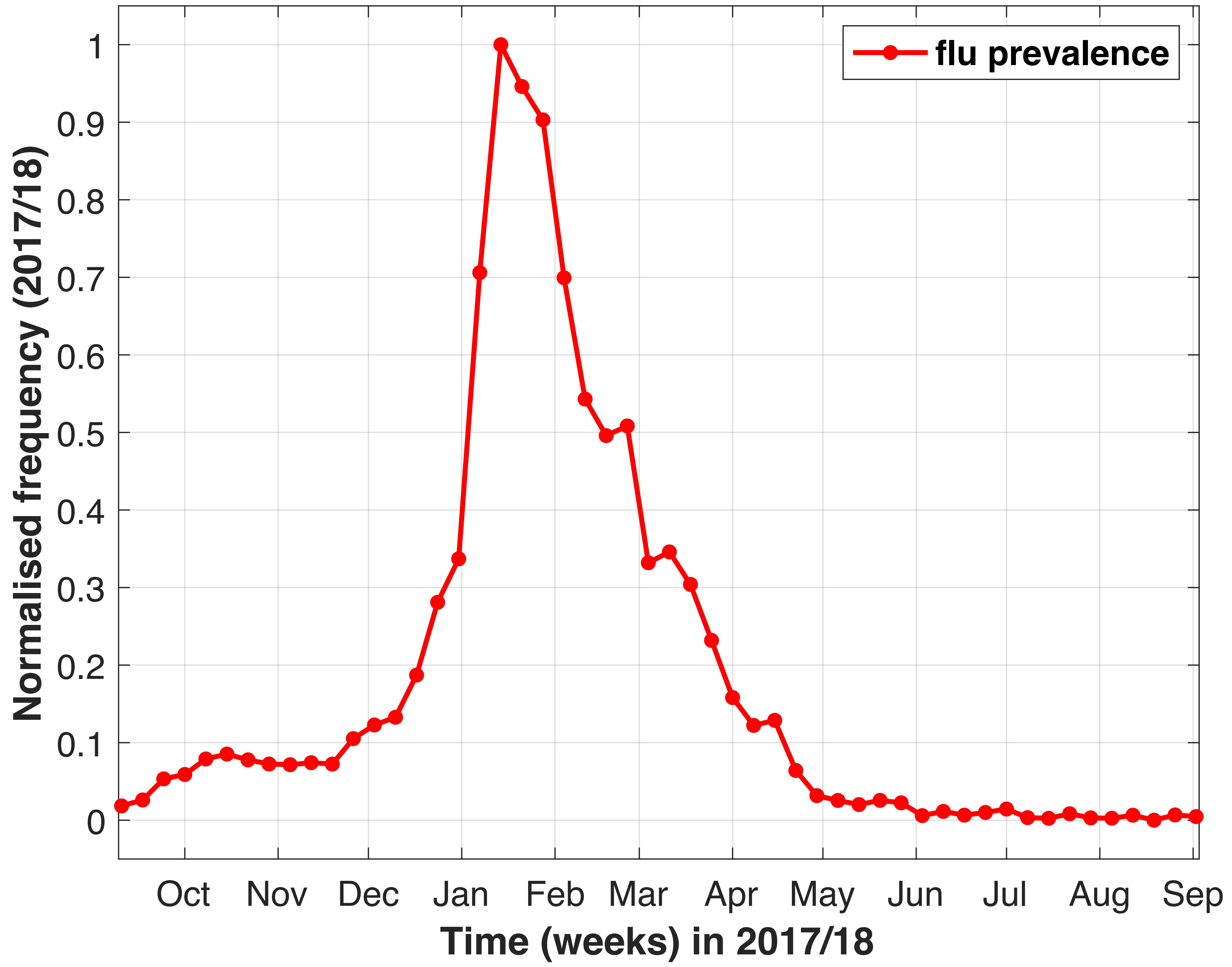
Supervised learning — Ordinary least squares (*linear*) regression

- $\mathbf{y}$ ~ weekly flu prevalence
- $\mathbf{x}$ ~ weekly search frequency of "*how long does flu last*"
- $f : \mathbf{x} \rightarrow \mathbf{y}$ such that $f(x_i) = \hat{y}_i = \alpha x_i + \beta \approx y_i$
- Find a line that best fits to our observations using **ordinary least squares** (OLS) regression



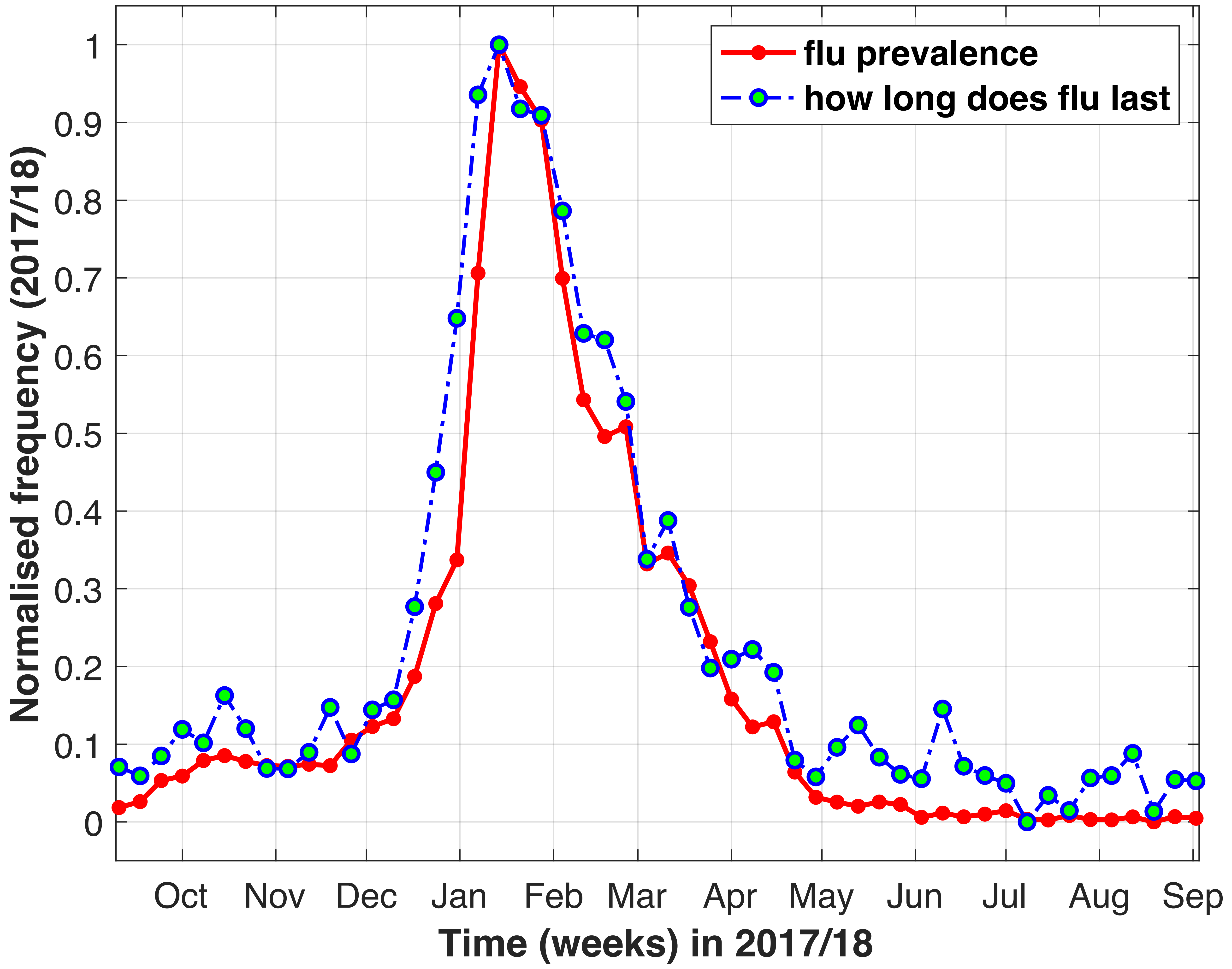
Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ weekly flu prevalence
- $\mathbf{x}$ ~ weekly search frequency of "*how long does flu last*"
- $f : \mathbf{x} \rightarrow \mathbf{y}$ such that $f(x_i) = \hat{y}_i = \alpha x_i + \beta \approx y_i$
- Find a line that best fits to our observations using **ordinary least squares** (OLS) regression



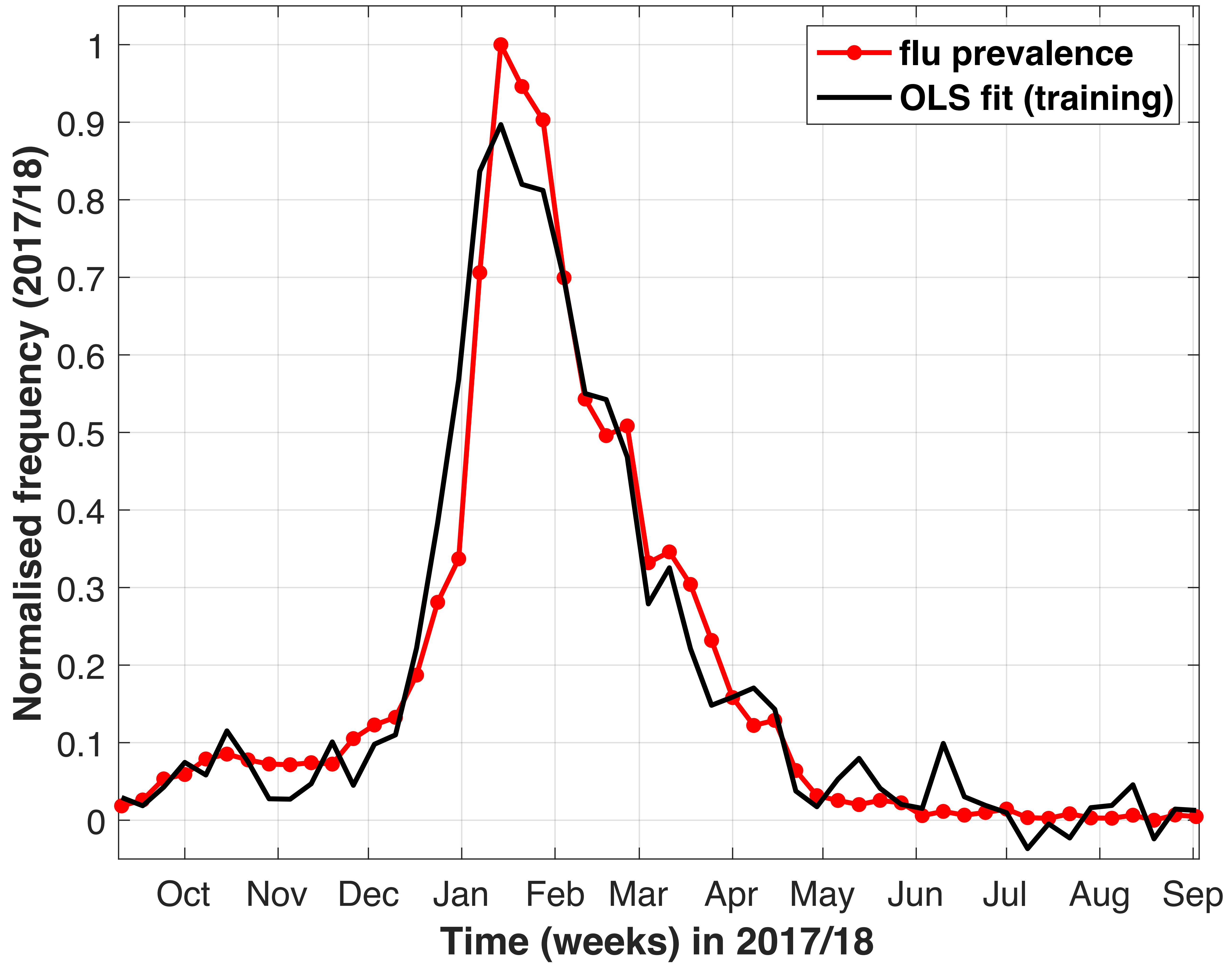
Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ weekly flu prevalence
- $\mathbf{x}$ ~ weekly search frequency of "*how long does flu last*"
- $f : \mathbf{x} \rightarrow \mathbf{y}$ such that
  $f(x_i) = \hat{y}_i = \alpha x_i + \beta \approx y_i$
- Find a line that best fits to our observations using **ordinary least squares** (OLS) regression



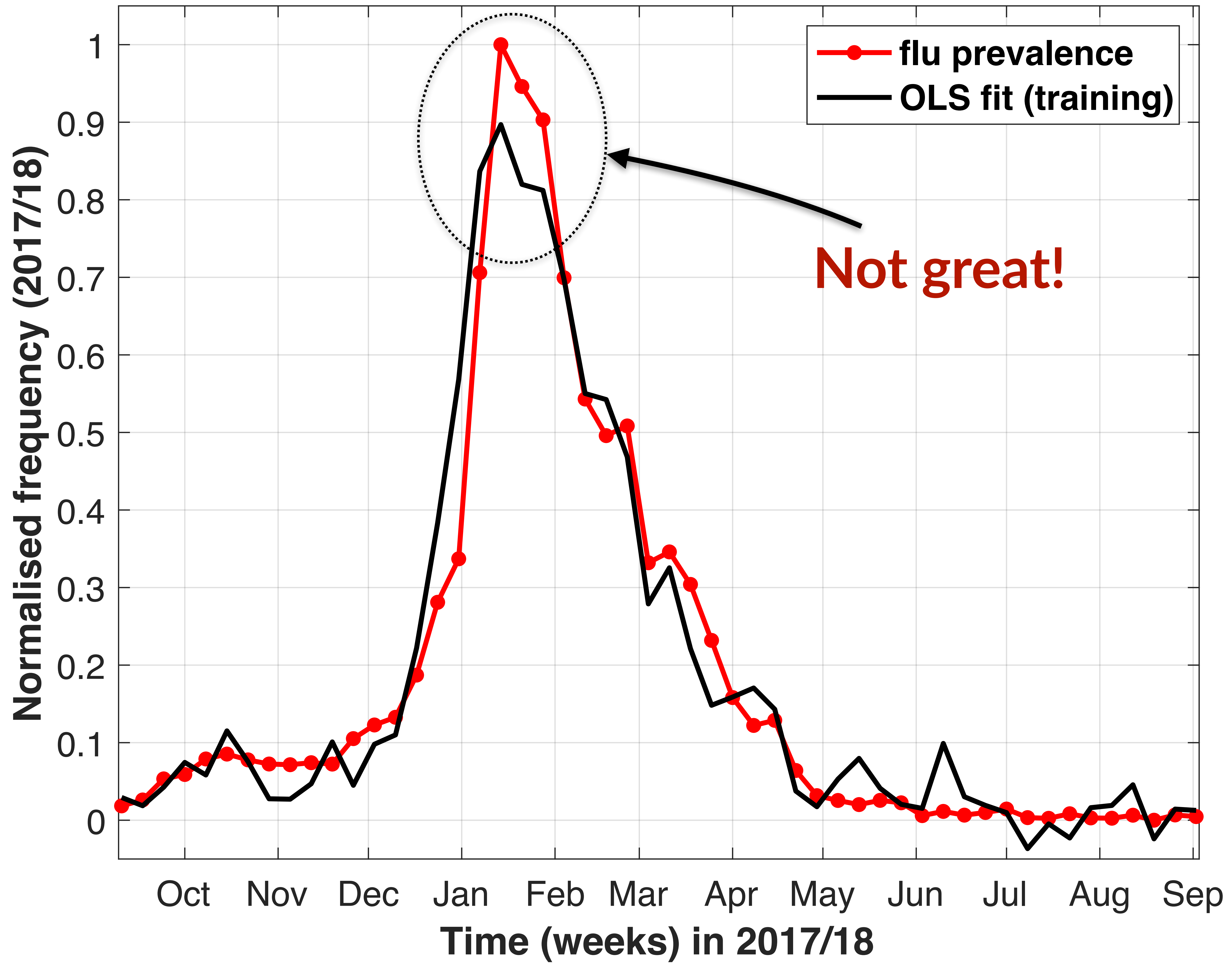
Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ weekly flu prevalence
- $\mathbf{x}$ ~ weekly search frequency of "*how long does flu last*"
- $f : \mathbf{x} \rightarrow \mathbf{y}$ such that $f(x_i) = \hat{y}_i = \alpha x_i + \beta \approx y_i$
- Find a line that best fits to our observations using **ordinary least squares** (OLS) regression



Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ weekly flu prevalence
- $\mathbf{x}$ ~ weekly search frequency of "*how long does flu last*"
- $f : \mathbf{x} \rightarrow \mathbf{y}$ such that $f(x_i) = \hat{y}_i = \alpha x_i + \beta \approx y_i$
- Find a line that best fits to our observations using **ordinary least squares** (OLS) regression



Supervised learning — OLS regression calculus solution

- The aim is to learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \mathbb{R}^{n}$
- f is a linear function, a set of weights and an intercept term; denoted by $\mathbf{w} \in \mathbb{R}^{m}$
- In our regression task (*see previous slides*), there is 1 weight (α) and the intercept (β)
  $\mathbf{X}$ has one column with the values of $\mathbf{x}$ and the other column is 1s
- Minimise a loss function known as residual sum or squares (*equivalent to mean squared error that we will see next*): $\mathscr{L}\left(\mathbf{w}\right) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$
- This can also be written as: $\mathscr{L}\left(\mathbf{w}\right) = \mathscr{L}\left(\alpha, \beta\right) = \sum_{i=1}^{n}\left(\alpha x_i + \beta - y_i\right)^2$

Supervised learning — OLS regression calculus solution

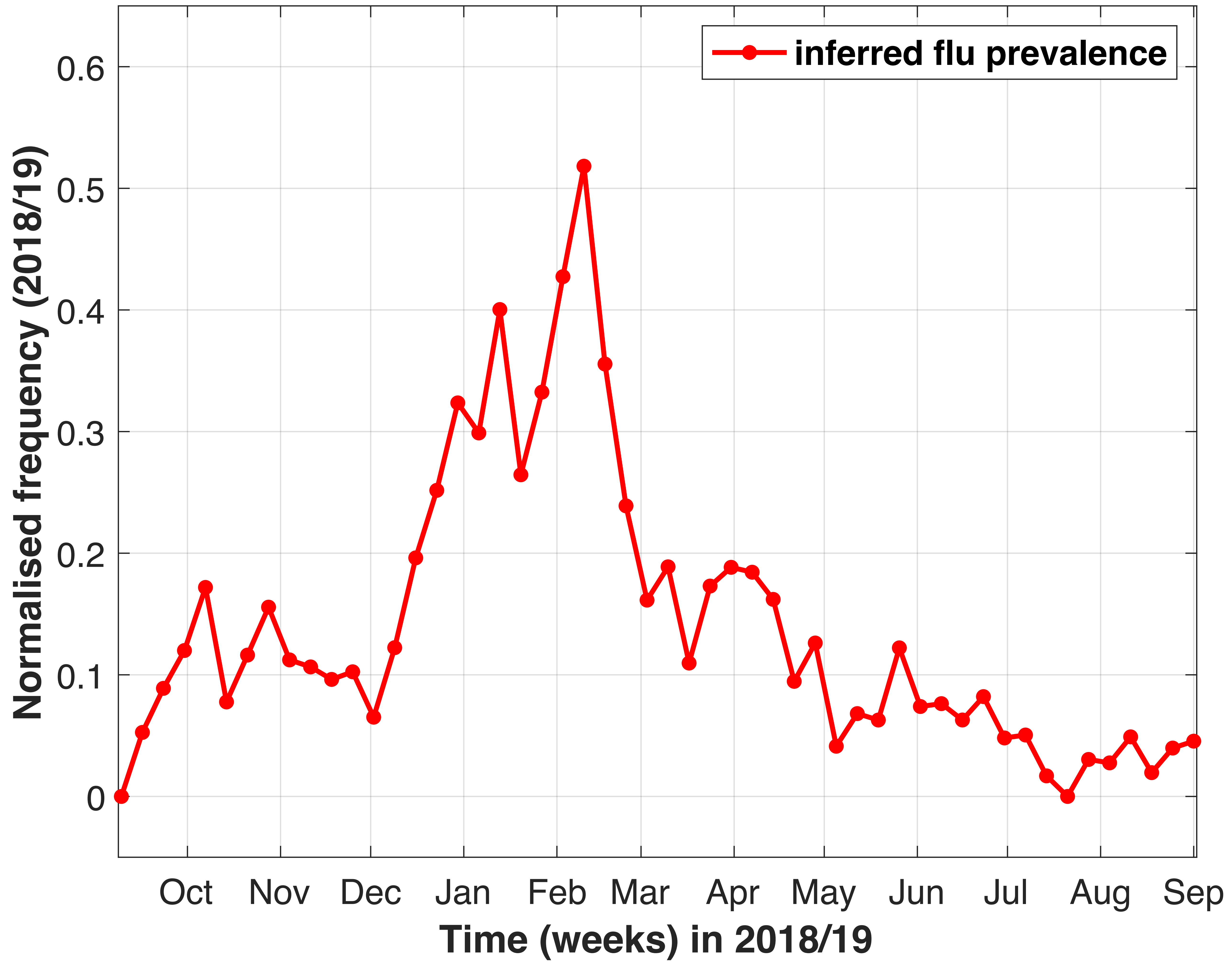
- The aim is to learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \mathbb{R}^{n}$
- f is a linear function, a set of weights and an intercept term; denoted by $\mathbf{w} \in \mathbb{R}^{m}$
- In our regression task (*see previous slides*), there is 1 weight (α) and the intercept (β) $\mathbf{X}$ has one column with the values of $\mathbf{x}$ and the other column is 1s
- Minimise a loss function known as residual sum or squares (*equivalent to mean squared error that we will see next*): $\mathscr{L}(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$
- Derivative with respect to $\mathbf{w}$: $\dfrac{\partial \mathscr{L}}{\partial \mathbf{w}} = -2\mathbf{X}^{\top}\mathbf{y} + 2\mathbf{X}^{\top}\mathbf{X}\mathbf{w}$
- Set this to 0 and hence $\mathbf{w} = \left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}\mathbf{y}$ as long as $\mathbf{X}^{\top}\mathbf{X}$ is full rank which means that the observations (rows) in $\mathbf{X}$ are more than the features ($n > m$) and that the features have no linear dependence

Supervised learning — OLS model training & testing

- Going back to our example, $\mathbf{w} = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}$ would give $\mathbf{w} = \left[0.93351 \; -0.036631\right]$, i.e. $\alpha = 0.93351$ and $\beta = -0.036631$
- The question now becomes, **how well will this model do in the next flu season**?
- Let's use the above values of α and β to estimate weekly flu prevalence in England for the season 2018/19 based on the corresponding frequency of the search query "*how long does flu last*"
- And then compare it with the actual flu prevalence in England for 2018/19

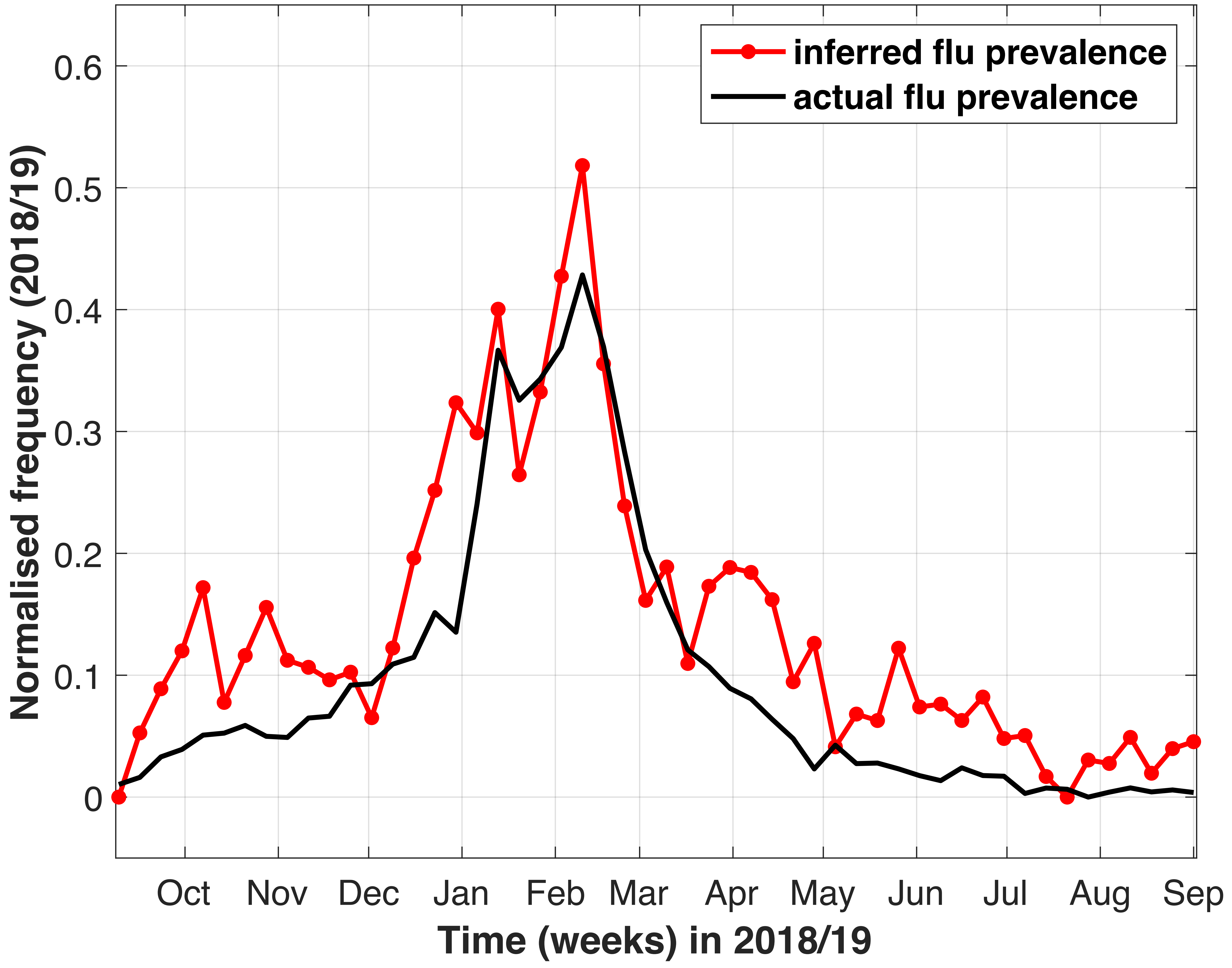
Supervised learning — OLS model training & testing

- These (*red line, dot • marker*) are the estimated (*inferred*) flu rates in 2018/19 (*to be exact from September 2018 to August 2019*) based on the OLS model and the frequency of the search query "*how long does flu last*"
- Recall, we trained our model using non-overlapping data from 2017/18 (*September 2017 to August 2018*)



Supervised learning — OLS model training & testing

- The **black solid** line represents the corresponding flu rates as reported by a health agency in the UK
- *Do you think this simple OLS model based on a single web search query did well?*
- *r* = 0.919 (*bivariate correlation*)
  RMSE = 0.0632 (*root mean squared error*)
  MAE = 0.0519 (*mean absolute error*)
- considering the simplicity of the model, its accuracy is quite surprising



Supervised learning — Gradient descent

- **Gradient descent**: optimisation algorithm that minimises a loss function $\mathscr{J}$ with respect to a set of hyperparameters
- Loss function for ordinary least squares (OLS) regression? If $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ denotes our estimates for $\mathbf{y}$, then the loss function for OLS is their mean squared difference (error):

$$\mathscr{J}(\mathbf{w}) = \frac{1}{2n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2 \text{, where } \hat{y}_i \in \hat{\mathbf{y}}, y_i \in \mathbf{y}$$

- Basic **steps** of gradient descent
  — define a loss function, $\mathscr{J}$
  — compute the partial derivatives of $\mathscr{J}$ w.r.t. each hyperparameter
  — update hyperparameters using their partial derivatives and learning rate ℓ often $\in (0,1)$
  — repeat until convergence

Supervised learning — Gradient descent

- **Learning rate:** how far away are we going to go in the opposite direction of the partial derivative — *we are going to see an example of this*
- **Why does it work?** We are taking steps in the opposite direction of the partial gradient to identify a local minimum.
- **When does it not work?** Not directly applicable to non-differentiable loss functions (but there exist workarounds)

Supervised learning — OLS with gradient descent

In our example, we are modelling a flu rate y_i using the frequency of a search query x_i

- **Hypothesis:** $\hat{y}_i = \alpha x_i + \beta$
  — *a flu estimate is a linear function of the frequency of the search query*
- **Hyperparameters:** $\{\alpha, \beta\}$
  these are unknown and should be estimated using gradient descent
- **Loss function:** $\mathcal{J}\left(\alpha, \beta\right) = \dfrac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2$
- **Goal:** $\min\limits_{\alpha,\beta} \mathcal{J}\left(\alpha, \beta\right)$

Supervised learning — OLS with gradient descent

In our example, we are modelling a flu rate y_i using the frequency of a search query x_i

- Start with some initial values for α and β denoted by α_0 and β_0, respectively
- In iteration $t+1$ of the gradient descent algorithm, update α and β with:

$$\alpha_{t+1} = \alpha_t - \ell \frac{\partial \mathcal{J}\left(\alpha, \beta\right)_t}{\partial \alpha} \text{ and } \beta_{t+1} = \beta_t - \ell \frac{\partial \mathcal{J}\left(\alpha, \beta\right)_t}{\partial \beta}$$

  where ℓ often $\in (0,1)$ denotes the learning rate we want to impose

- NB: both derivatives update in iteration $t+1$ based on values from iteration t
- Repeat until convergence

Supervised learning — OLS with gradient descent, the derivatives

Loss function: $\mathcal{J}(\alpha, \beta) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$ n samples, $2n$ is a convention, $\mathcal{J} = \text{MSE}/2$

$$= \frac{1}{2n} \sum_{i=1}^{n} (\alpha x_i + \beta - y_i)^2$$

$$\frac{\partial \mathcal{J}(\alpha, \beta)}{\partial \alpha} = \frac{1}{2n} \sum_{i=1}^{n} \left(2(\alpha x_i + \beta - y_i)\, x_i\right) = \frac{1}{n} \sum_{i=1}^{n} \left((\alpha x_i + \beta - y_i)\, x_i\right)$$

$$\frac{\partial \mathcal{J}(\alpha, \beta)}{\partial \beta} = \frac{1}{n} \sum_{i=1}^{n} (\alpha x_i + \beta - y_i)$$

Supervised learning — OLS with gradient descent, the derivatives

$$\mathcal{J}\left(\mathbf{w}, \beta\right) = \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2$$

What if we had m predictors?

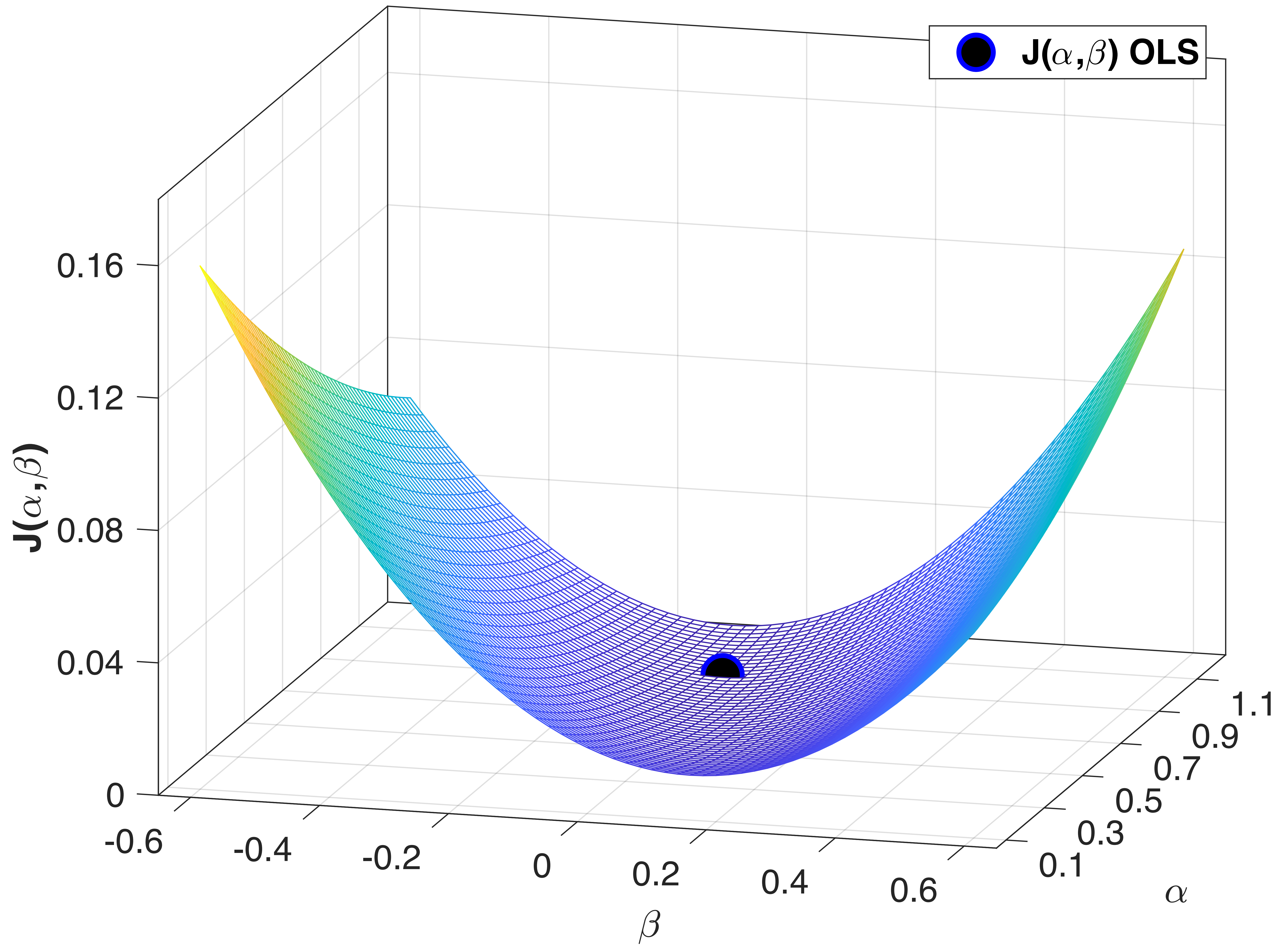
$$= \frac{1}{2n} \sum_{i=1}^{n} \left(w_1 x_{i,1} + \ldots + w_m x_{i,m} + \beta - y_i\right)^2$$

$$\frac{\partial \mathcal{J}\left(\mathbf{w}, \beta\right)}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} \left(\left(w_1 x_{i,1} + \ldots + w_m x_{i,m} + \beta - y_i\right) x_{i,j}\right)$$

$$\frac{\partial \mathcal{J}\left(\mathbf{w}, \beta\right)}{\partial \beta} = ?$$

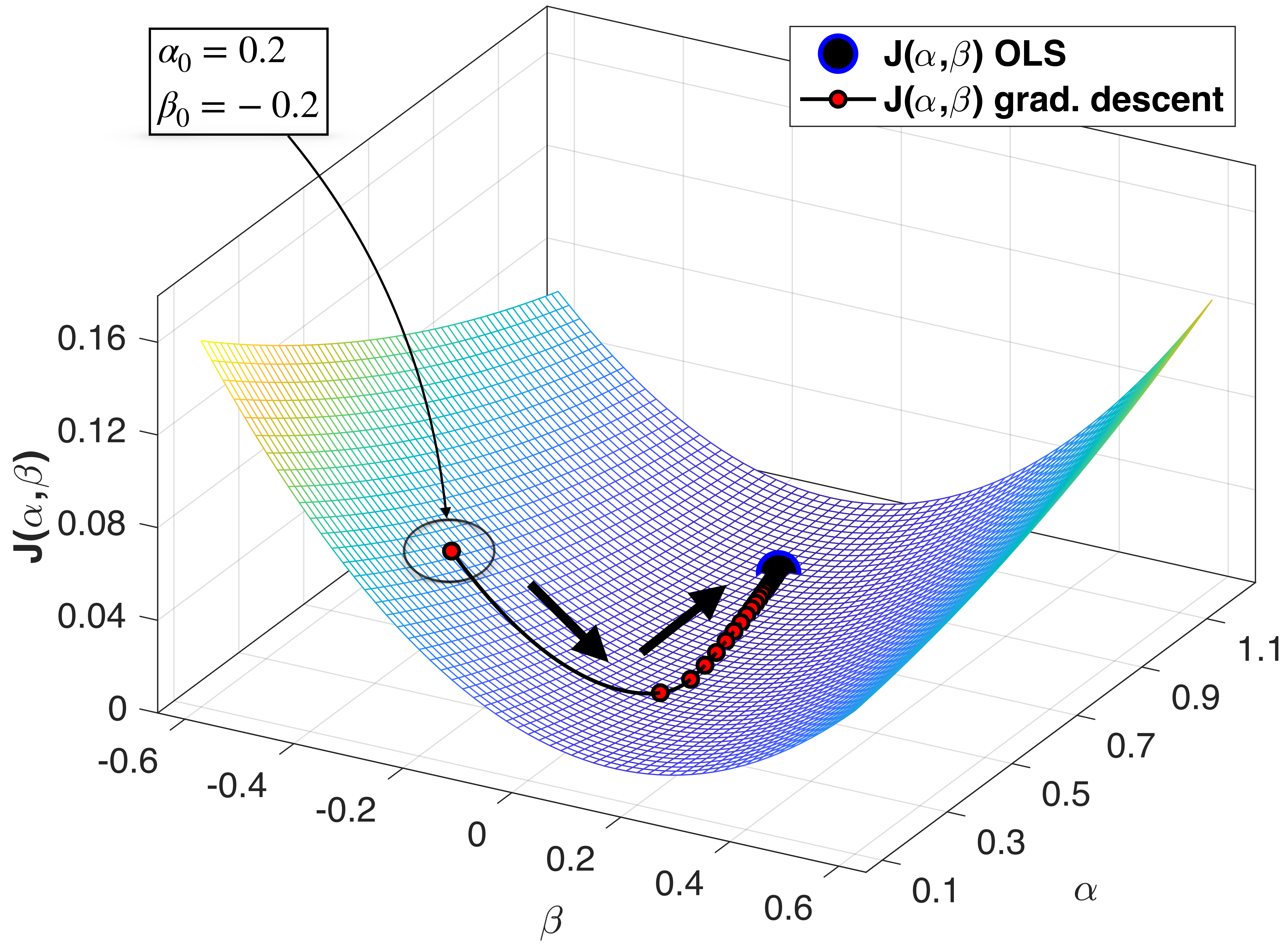
Supervised learning — OLS with gradient descent

- OLS example: inferring flu prevalence based on the frequency of 1 search query
- Let's explore the space of hyperparameter values for OLS $\{\alpha, \beta\}$ and the corresponding values of the loss function $\mathcal{J}\left(\alpha, \beta\right)$ – 3-dimensional plot (surface or mesh plot)
- Convex loss (**easier task?**)
- Big (half) dot/ball denotes the exact OLS solution (*no gradient descent used*)



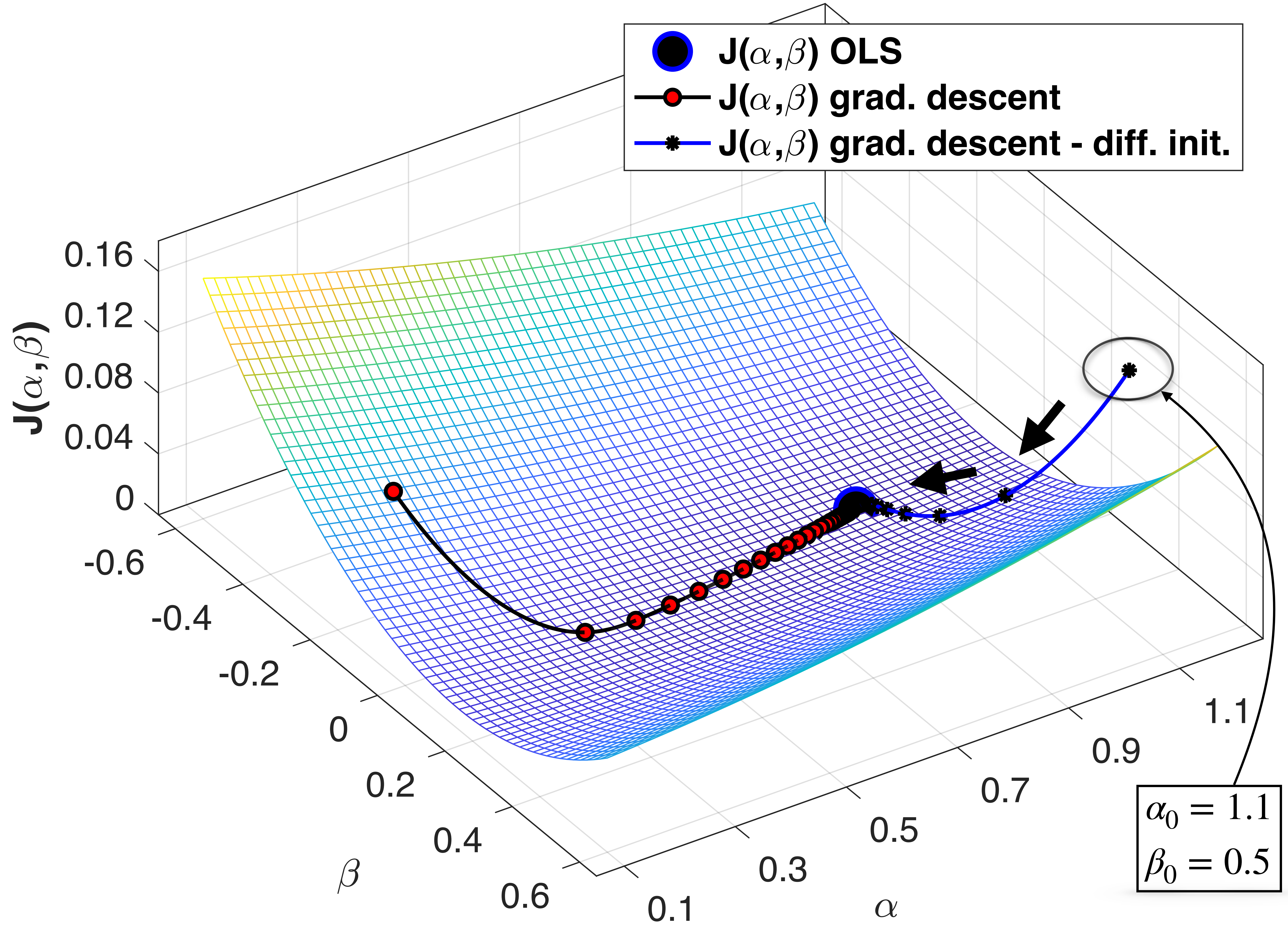
Supervised learning — OLS with gradient descent

- Let's start from a point in the grid, set some initial values for the hyperparameters and attempt to solve this with coordinate descent
- $\alpha_0 = 0.2, \beta_0 = -0.2$
- $\ell = 0.02$ *(learning rate)*
- Convergence criterion: How much has $\mathcal{J}\left(\alpha, \beta\right)$ changed in the past k iterations?
- Gradient descent's solution almost identical to exact OLS solution (**expected?**)



Supervised learning — OLS with gradient descent

- Let’s change the starting point
- $\alpha_0 = 1.1, \beta_0 = 0.5$
- $\ell = 0.02$ (*same learning rate*)
- In this case, it does not affect our solution (**why?**)



Supervised learning — Gradient descent, general remarks

- Effect of **learning rate** ℓ
  — if it is too small, gradient descent can be slow
  — if it is too large, gradient descent may fail to converge (overshoots the minimum)
  — adaptive learning rate (*by using line search*)
- Different initialisations might help get past local optima
- **Batch** gradient descent (*presented today*): use the entire training set for gradient updates
  — guaranteed convergence to a local minimum
  — slow on large problems (e.g. neural networks)
- **Stochastic** gradient descent: use one training sample for gradient updates
  — faster convergence on large redundant data sets
  — hard to reach high accuracy
- **Mini-batch** gradient descent: use a subset of the training set for gradient updates
  — very common in neural network training
  — better in avoiding local minima
  — what is the best mini-batch size (number of training samples to use)?

Supervised learning

- **Regression**
  estimate / predict a continuous output / target variable

  i.e. learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \mathbb{R}^{n}$

  Examples: predict a time series trend (e.g. in finance), estimate the prevalence of a condition in epidemiology

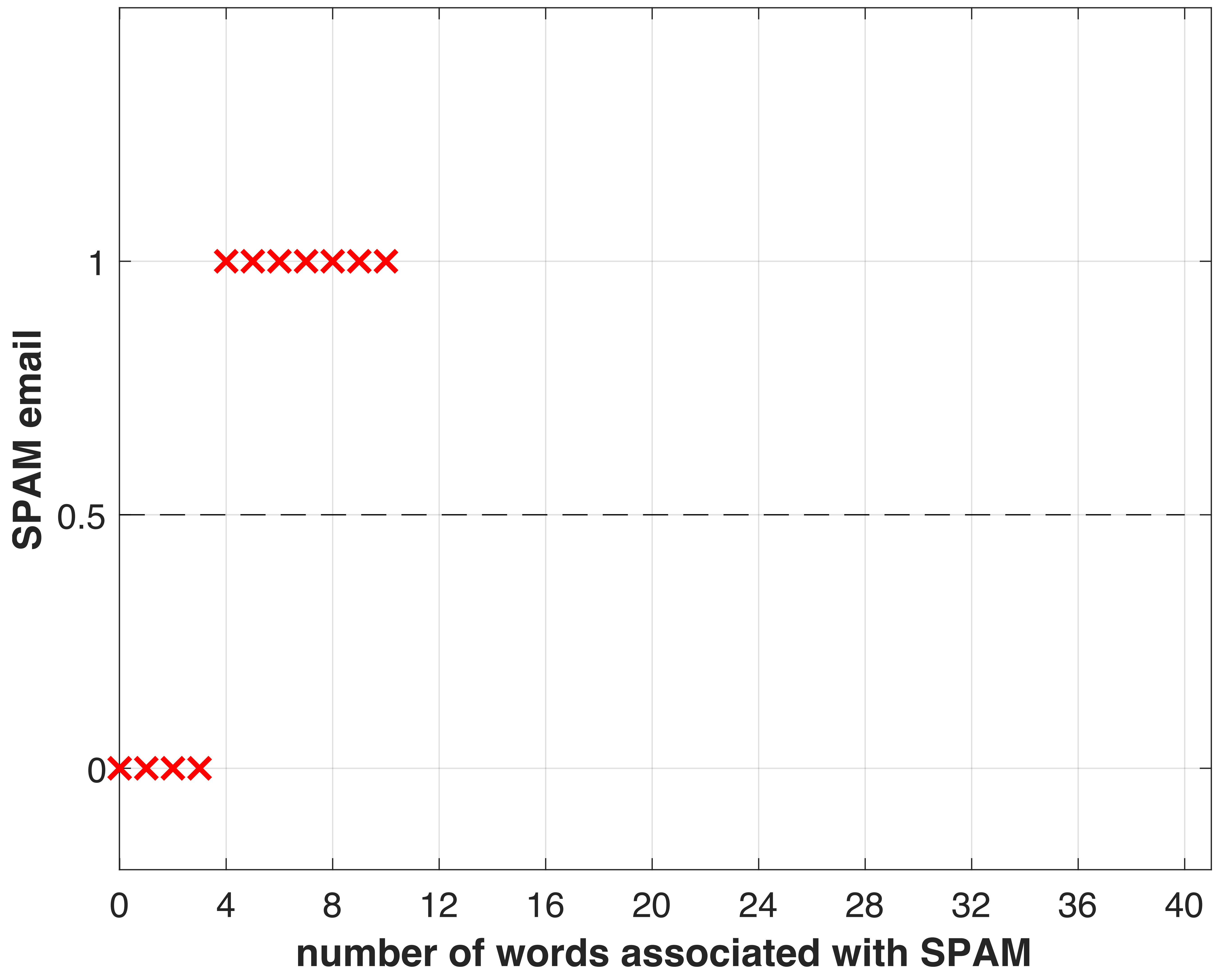
- **Classification**
  estimate a set of C unordered (and mutually exclusive) labels / classes

  i.e. learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \{1, 2, \ldots, C\}$

  Examples: detect spam email, medical imaging, text/document classification

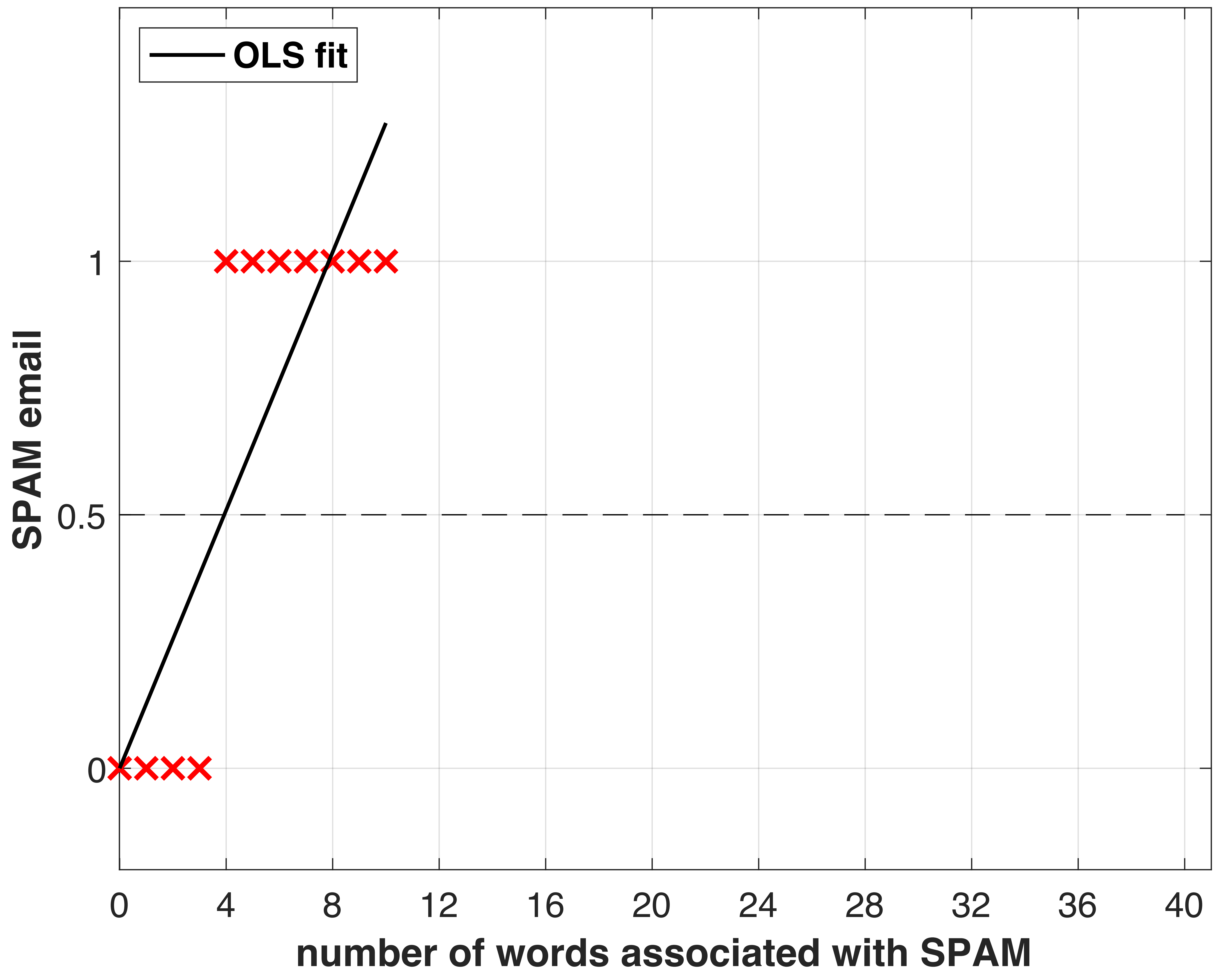
Supervised learning — Binary classification

- **Binary classification** means that we only have two label categories, e.g.
  > *spam* vs. *not spam* email
  > *relevant* vs. *not relevant* document
- if $f_{\mathbf{w}}(x_i) \geq 0.5$, then SPAM
  if $f_{\mathbf{w}}(x_i) < 0.5$, then not SPAM
- What if we used OLS to learn $f_{\mathbf{w}}$?



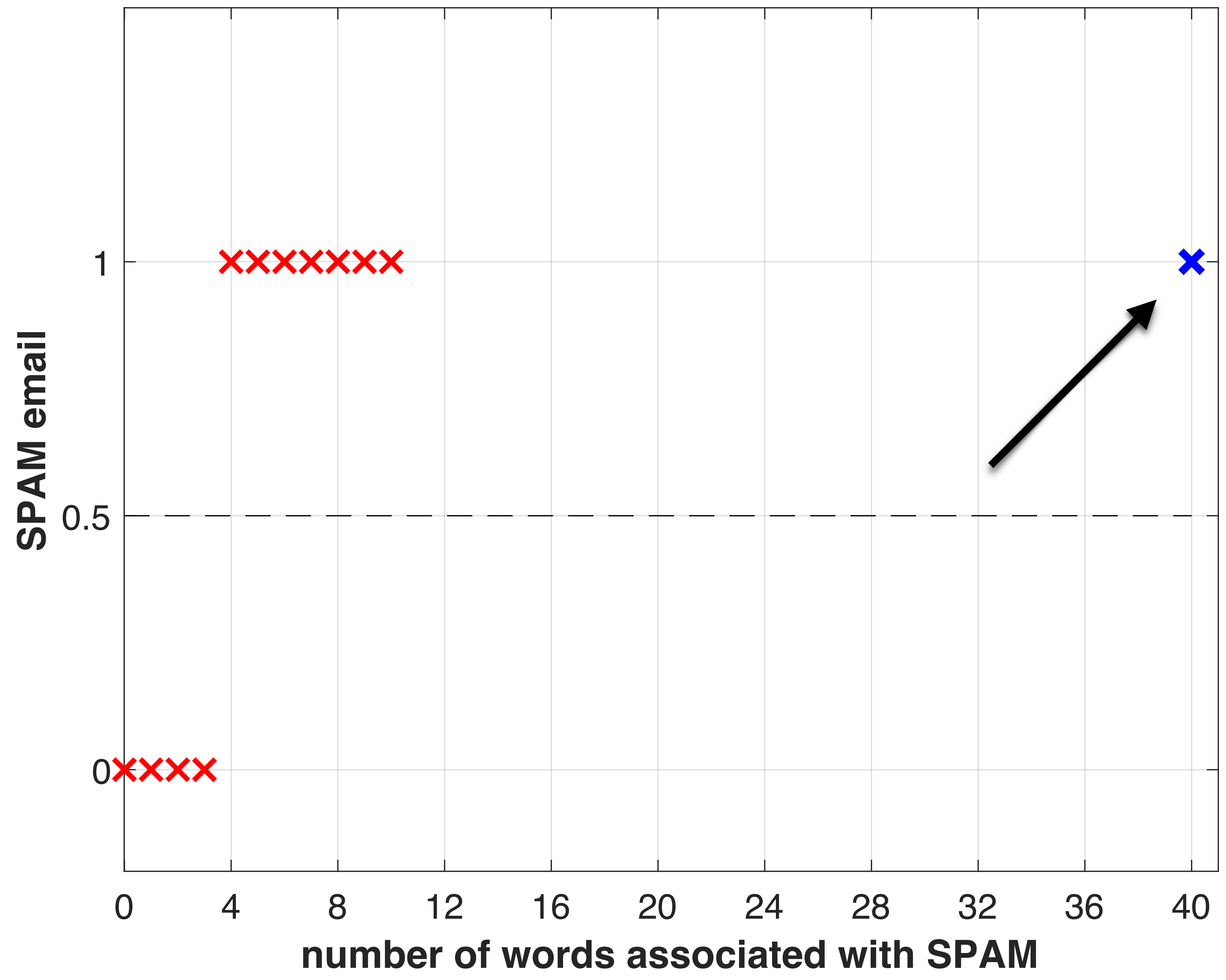
Supervised learning — Binary classification

- Binary classification means that we only have two label categories, e.g.
  > *spam* vs. *not spam* email
  > *relevant* vs. *not relevant* document
- if $f_{\mathbf{w}}\left(x_i\right) \geq 0.5,$ then SPAM
  if $f_{\mathbf{w}}\left(x_i\right) < 0.5,$ then not SPAM
- What if we used OLS to learn $f_{\mathbf{w}}$?
  Looks perfectly fine?



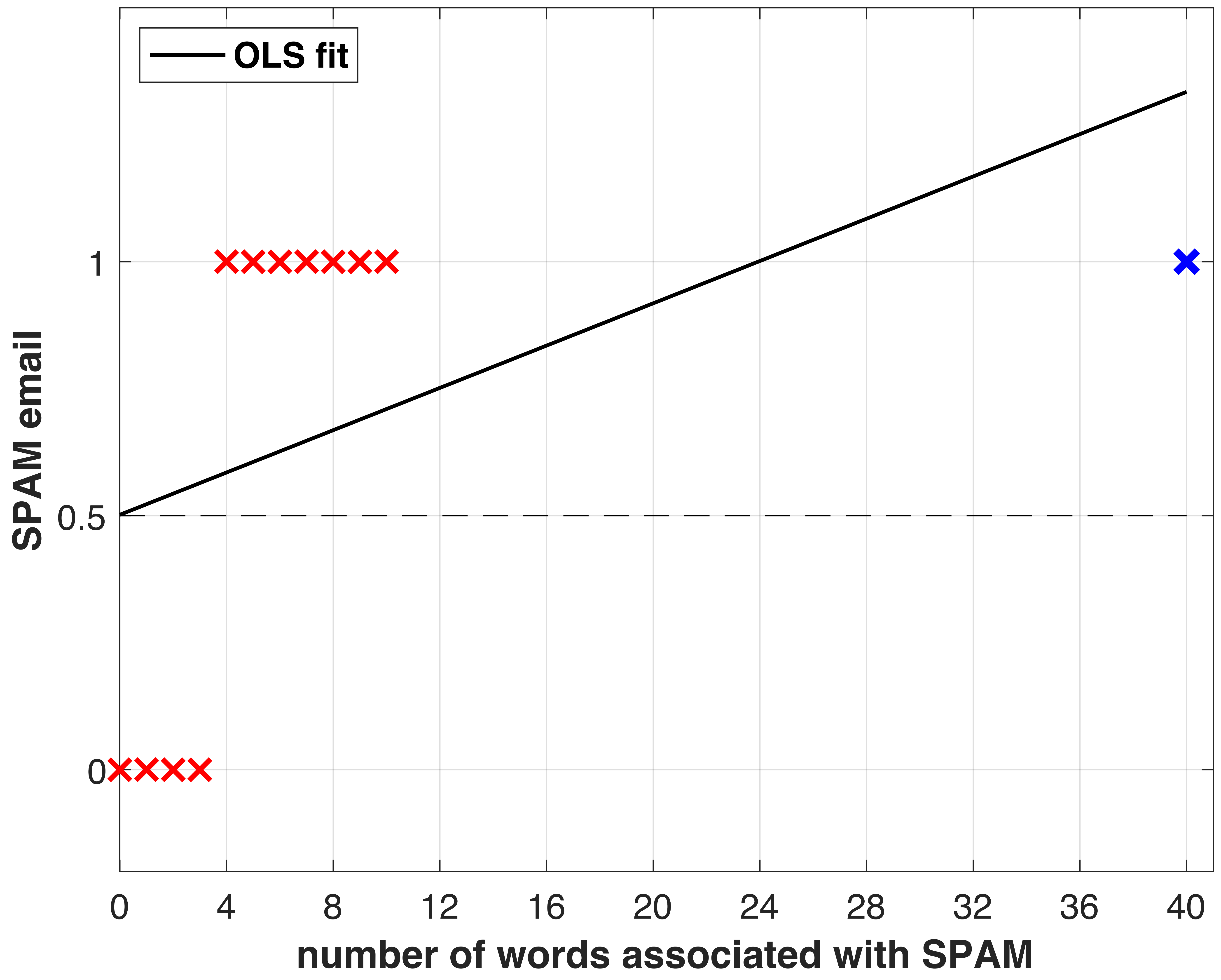
Supervised learning — Binary classification

- **Binary classification** means that we only have two label categories, e.g.
  > *spam* vs. *not spam* email
  > *relevant* vs. *not relevant* document
- if $f_{\mathbf{w}}\left(x_i\right) \geq 0.5,$ then SPAM
  if $f_{\mathbf{w}}\left(x_i\right) < 0.5,$ then not SPAM
- What if we used OLS to learn $f_{\mathbf{w}}$?
  Looks perfectly fine?
- Let's add one more observation to our data. How would that affect our OLS classifier?



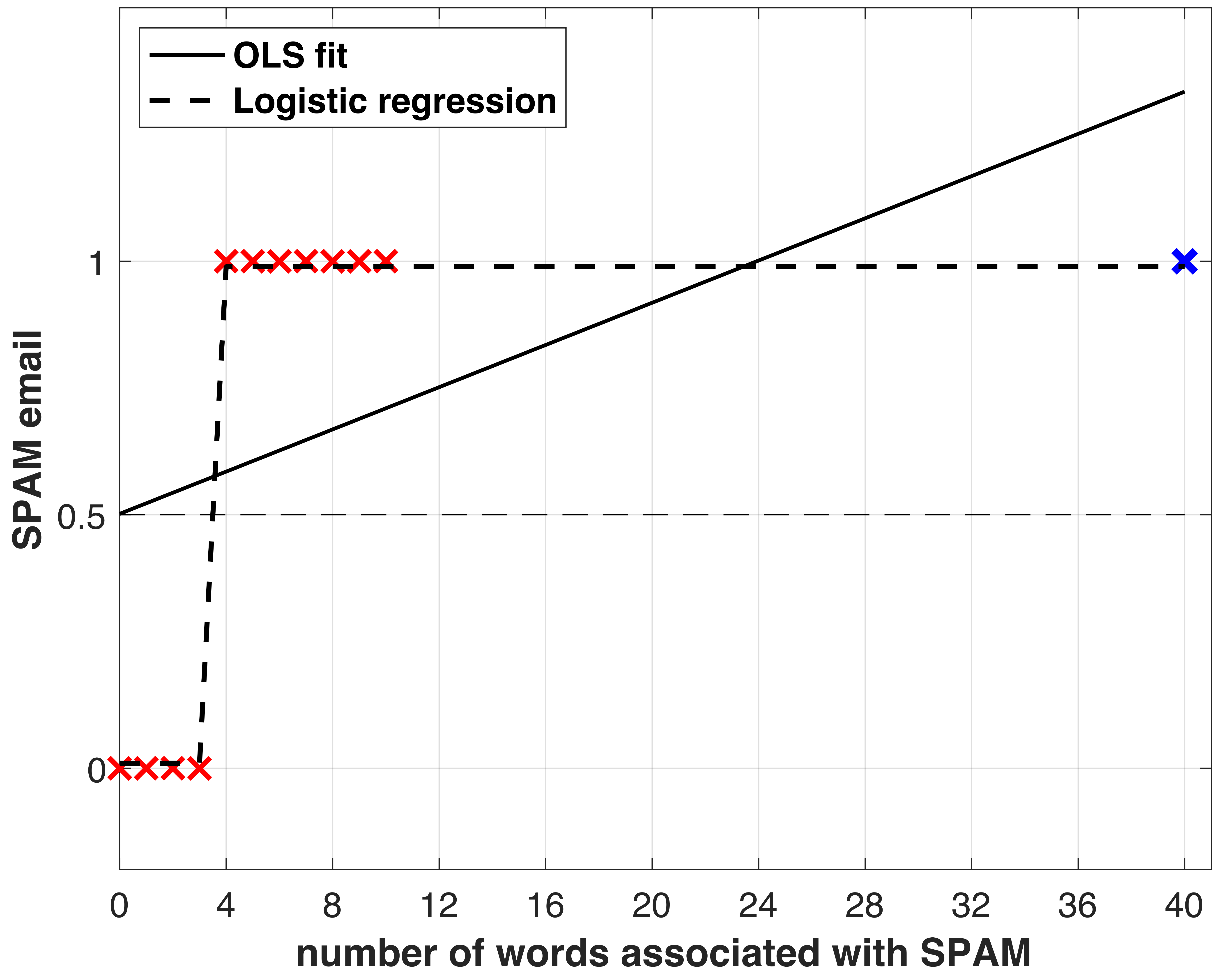
Supervised learning — Binary classification

- **Binary classification** means that we only have two label categories, e.g.
  > *spam* vs. *not spam* email
  > *relevant* vs. *not relevant* document
- if $f_{\mathbf{w}}\left(x_i\right) \geq 0.5,$ then SPAM
  if $f_{\mathbf{w}}\left(x_i\right) < 0.5,$ then not SPAM
- What if we used OLS to learn $f_{\mathbf{w}}$?
  Looks perfectly fine?
- Let's add one more observation to our data. How would that affect our OLS classifier? **Not great!**



Supervised learning — Binary classification

- **Binary classification** means that we only have two label categories, e.g.
  > *spam* vs. *not spam* email
  > *relevant* vs. *not relevant* document
- if $f_{\mathbf{w}}\left(x_i\right) \geq 0.5,$ then SPAM
  if $f_{\mathbf{w}}\left(x_i\right) < 0.5,$ then not SPAM
- What if we used OLS to learn $f_{\mathbf{w}}$?
  Looks perfectly fine?
- Let's add one more observation to our data. How would that affect our OLS classifier? **Not great!**
- It is not impossible to separate these classes — we just need a **different function**.



Supervised learning — Logistic regression

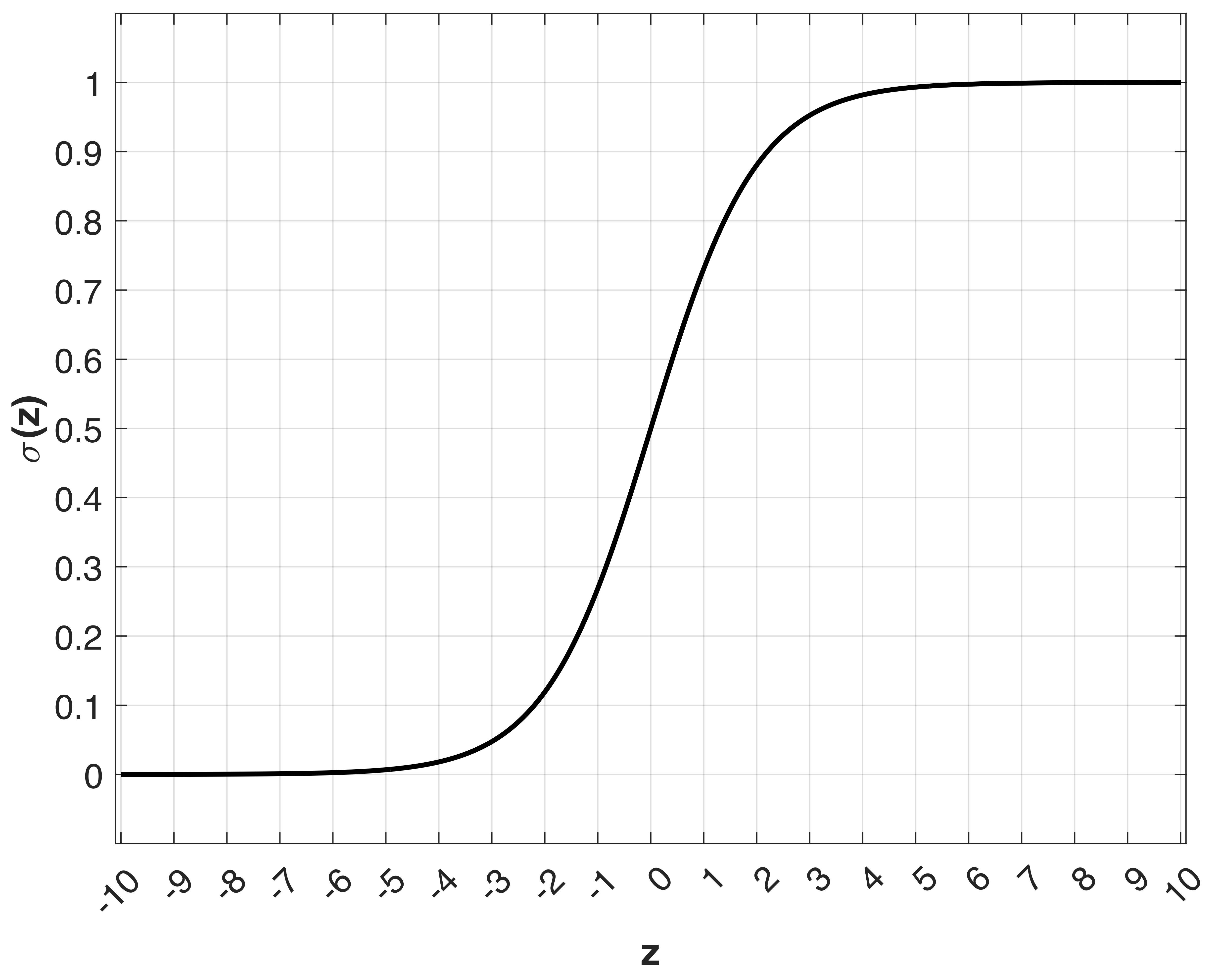
- Assume we have an m-dimensional observation $\mathbf{x} \in \mathbb{R}^m$
- We want $0 \leq f_{\mathbf{w}}(\mathbf{x}) \leq 1$, where $\mathbf{w} \in \mathbb{R}^m$ are the corresponding weights
- Sigmoid or logistic function

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

- $f_{\mathbf{w}}(\mathbf{x}) = \sigma\left(\mathbf{w}^\top \mathbf{x}\right) \in [0,1]$

  it can be seen as a probability

Sigmoid / logistic function



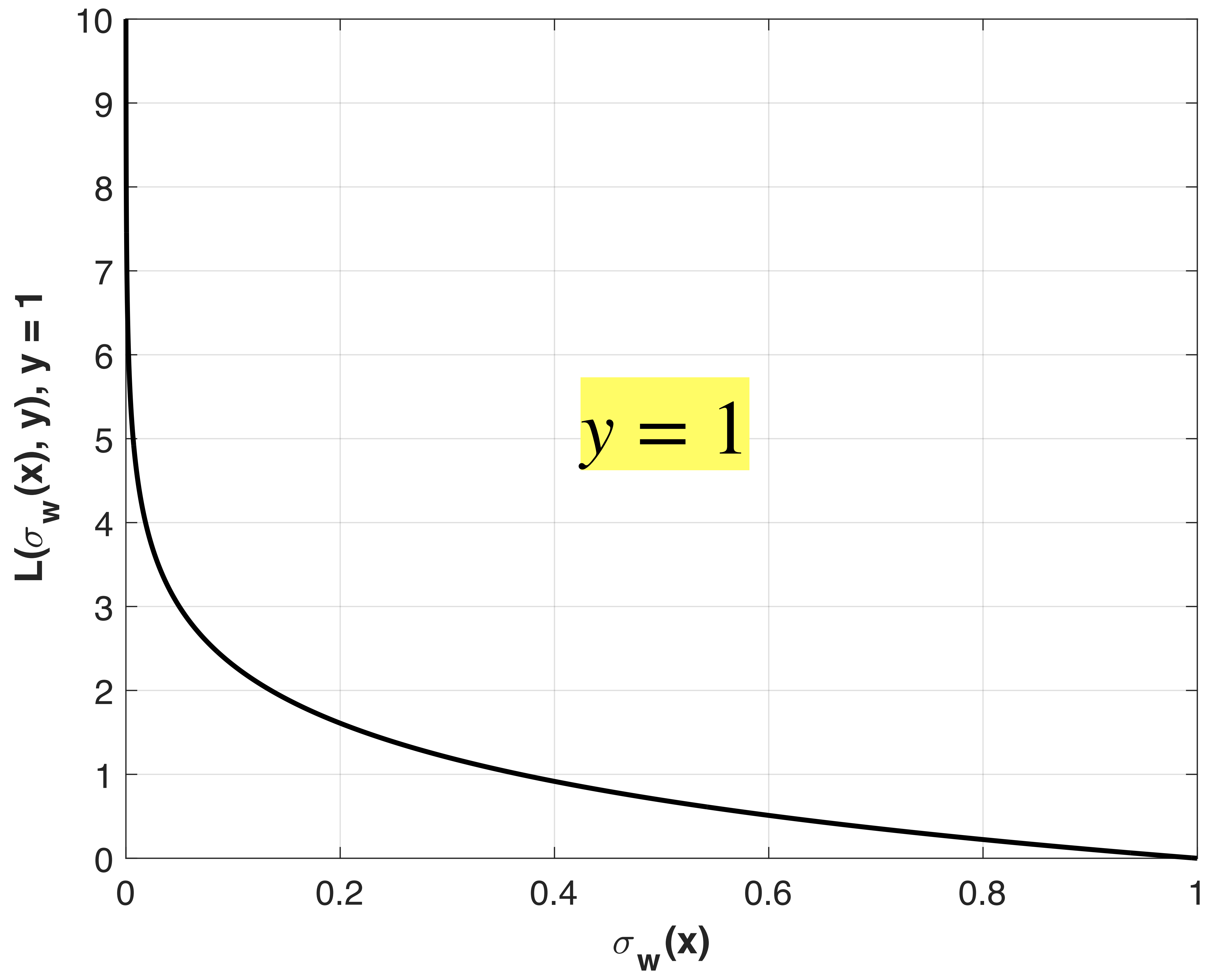
Supervised learning — Logistic regression's loss function

- Cross-entropy loss function between the projection of $\mathbf{x}$ and label $y \in \{0,1\}$

- $\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(\sigma_{\mathbf{w}}(\mathbf{x})\right)$ if $y = 1$
  $\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(1 - \sigma_{\mathbf{w}}(\mathbf{x})\right)$ if $y = 0$

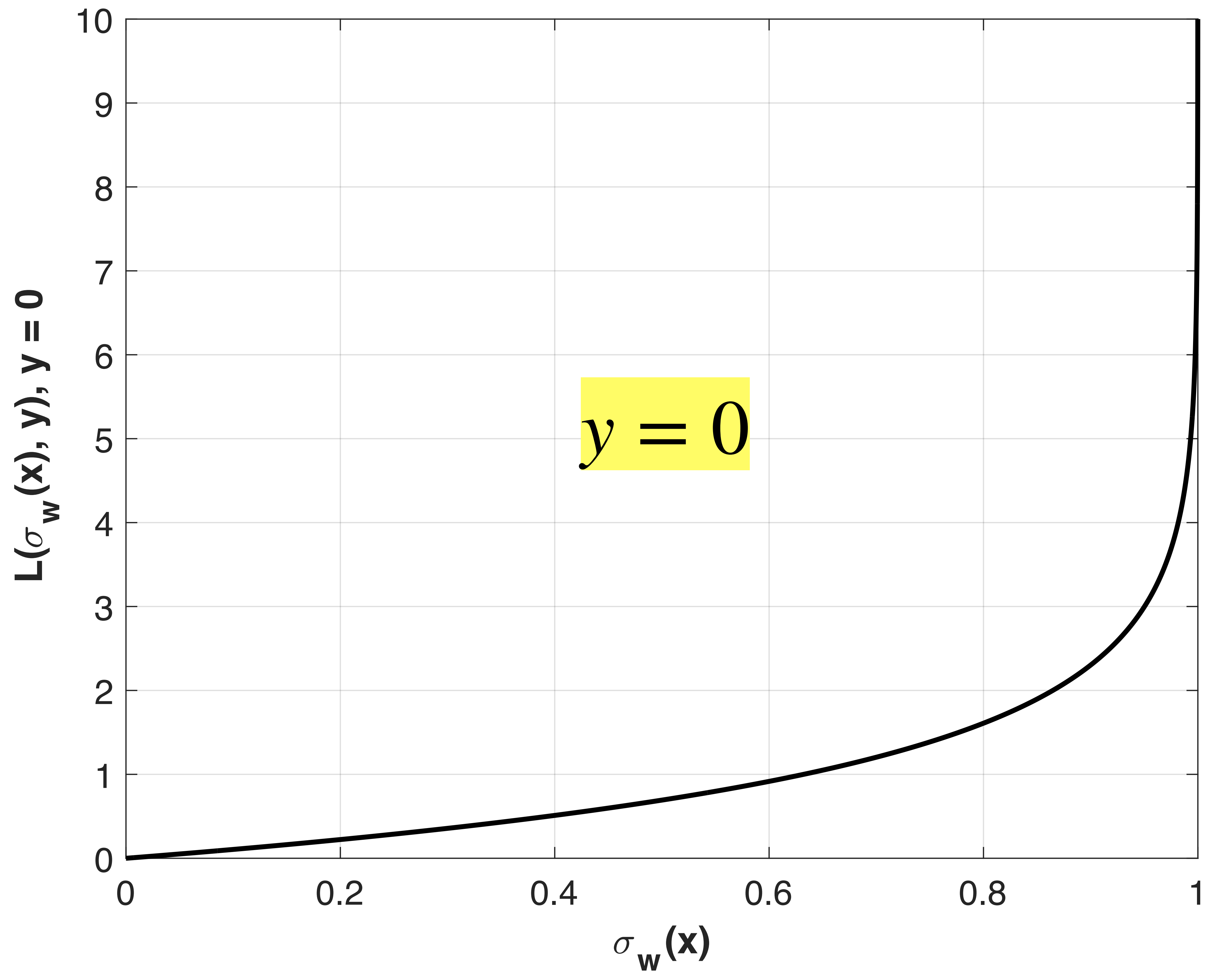
- Derivation from Bernoulli distribution

- Intuitively
  — we want a loss that is easy to differentiate
  – if $y = 1, \sigma_{\mathbf{w}}(\mathbf{x}) \rightarrow 1$: $\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) \rightarrow 0$
  – if $y = 1, \sigma_{\mathbf{w}}(\mathbf{x}) \rightarrow 0$: $\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) \rightarrow \infty$



Supervised learning — Logistic regression's loss function

- Cross-entropy loss function between the projection of $\mathbf{x}$ and label $y \in \{0,1\}$
- $\mathcal{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(\sigma_{\mathbf{w}}(\mathbf{x})\right)$ if $y = 1$
  $\mathcal{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(1 - \sigma_{\mathbf{w}}(\mathbf{x})\right)$ if $y = 0$
- Derivation from Bernoulli distribution
- Intuitively
  — we want a loss that is easy to differentiate
  – if $y = 0$, $\sigma_{\mathbf{w}}(\mathbf{x}) \to 0$: $\mathcal{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) \to 0$
  – if $y = 0$, $\sigma_{\mathbf{w}}(\mathbf{x}) \to 1$: $\mathcal{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) \to \infty$



Supervised learning — Logistic regression with gradient descent

Cross-entropy loss function

$$\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(\sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 1$$
$$\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(1 - \sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 0$$

Logistic (sigmoid) function

$$\sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) = \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)^{-1}$$

Our label y_i is either 1 or 0 for all our observations. So, for each observation only one part of the loss function is activated / used. Since we have n observations the loss function takes the form:

$$\mathscr{J}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) + \left(1 - y_i\right) \ln\left(1 - \sigma_{\mathbf{w}}\left(\mathbf{x}_i\right)\right)\right]$$

Supervised learning — Logistic regression with gradient descent

Cross-entropy loss function

$$\mathcal{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(\sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 1$$
$$\mathcal{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(1 - \sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 0$$

Logistic (sigmoid) function

$$\sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) = \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)^{-1}$$

Combined loss function

$$\mathcal{J}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) + \left(1 - y_i\right) \ln\left(1 - \sigma_{\mathbf{w}}\left(\mathbf{x}_i\right)\right)\right]$$

Let's incorporate the actual value of the sigmoid function and attempt to simplify:

$$\ln\left(\sigma_{\mathbf{w}}\left(\mathbf{x}_i\right)\right) = \ln(1) - \ln\left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right) = -\ln\left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)$$

The loss function becomes:

$$\mathcal{J}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{w}^\top \mathbf{x}_i y_i - \mathbf{w}^\top \mathbf{x}_i - \ln\left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)\right]$$

Supervised learning — Logistic regression with gradient descent

Cross-entropy loss function

$$\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(\sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 1$$
$$\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(1 - \sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 0$$

Logistic (sigmoid) function

$$\sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) = \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)^{-1}$$

Combined loss function

$$\mathscr{J}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{w}^\top \mathbf{x}_i y_i - \mathbf{w}^\top \mathbf{x}_i - \ln\left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)\right]$$

Partial derivative

$$\frac{\partial \mathscr{J}(\mathbf{w})}{\partial \mathbf{w}_j} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i x_{i,j} - x_{i,j} + e^{-\mathbf{w}^\top \mathbf{x}_i}\left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)^{-1} x_{i,j}\right]$$
$$= -\frac{1}{n}\sum_{i=1}^{n}\left[x_{i,j}\left(y_i - \sigma_{\mathbf{w}}\left(\mathbf{x}_i\right)\right)\right]$$

Supervised learning — Logistic regression with gradient descent

Cross-entropy loss function

$$\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(\sigma_{\mathbf{w}}(\mathbf{x})\right) \qquad \text{if } y = 1$$

$$\mathscr{L}\left(\sigma_{\mathbf{w}}(\mathbf{x}), y\right) = -\ln\left(1 - \sigma_{\mathbf{w}}(\mathbf{x})\right) \quad \text{if } y = 0$$

Logistic (sigmoid) function

$$\sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) = \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)^{-1}$$

Combined loss function

$$\mathscr{J}(\mathbf{w}) \quad 1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\Big)\Big]$$

Partial derivative

$$\frac{\partial \mathscr{J}(\mathbf{w})}{\partial \mathbf{w}_j} \quad \sum_{i=1} \Big[\quad + e^{-\mathbf{w}^\top \mathbf{x}_i}\Big)^{-1} x_{i,j}\Big]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[x_{i,j}\left(y_i - \sigma_{\mathbf{w}}\left(\mathbf{x}_i\right)\right)\right]$$

The rest is identical to the least squares example, i.e. initialise $\mathbf{w}$, then compute the partial derivatives for each w_j, then update w_j's using a learning rate, and repeat until convergence.

Supervised learning — Logistic regression, example

- Going back to the application of estimating flu prevalence using web search activity
- Now, we want to use the frequency of 4 search queries to predict whether the flu rate in a population is above a low-epidemic threshold or not
  – binary classification task
  – $y_i = 1$, if the flu rate is above a low-epidemic threshold
  – $y_i = 0$, if the flu rate is below or equal to a low-epidemic threshold
- We have in total 104 weekly observations
  – observation matrix $\mathbf{X} \in \mathbb{R}^{104 \times 4}$
  – queries: "*how long does flu last*", "*flu symptoms*", "*cough flu*", "*flu recovery*"
  – labels $\mathbf{y} \in \{0,1\}^{104}$

Supervised learning — Logistic regression, example

- We have in total 104 weekly observations
  — observation matrix $\mathbf{X} \in \mathbb{R}^{104 \times 4}$
  — queries: "*how long does flu last*", "*flu symptoms*", "*cough flu*", "*flu recovery*"
  — labels $\mathbf{y} \in \{0,1\}^{104}$

- 4-fold cross validation
  — form 4 folds (equally sized baskets) of the data,
  — train a classifier using 3 of them, test (evaluate) on the remaining 1
  — report average performance metrics

Binary classification — Basic performance metrics

- $\hat{\mathbf{y}} \in \{0,1\}^n$ denotes our predictions and $\mathbf{y} \in \{0,1\}^n$ the correct labels
- $$\text{accuracy} = \frac{\text{number of times } \hat{y}_i = y}{n}$$
- $$\text{precision} = \frac{\text{number of times } \hat{y}_i = 1 \text{ AND } \hat{y}_i = y_i}{\text{number of times } \hat{y}_i = 1}$$
- $$\text{recall} = \frac{\text{number of times } \hat{y}_i = 1 \text{ AND } \hat{y}_i = y_i}{\text{number of times } y_i = 1}$$
- F_1 score is the harmonic mean between precision and recall

  $$\text{F}_1 \text{ score} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Supervised learning — Logistic regression, example

logistic regression performance metrics

accuracy $= 0.923\,(0.070)$

precision $= 0.902\,(0.121)$

recall $= 0.844\,(0.120)$

F_1 score $= 0.871(0.116)$

logistic regression weights

flu symptoms: 78.058

how long does flu last: 24.537

flu recovery: 3.8977

cough flu: -14.663

Download the data from
dropbox.com/s/rgyg190whw26qrj/data-COMP0084-intro-to-ml.zip?dl=0
and try it at home!

Multi-class classification

- Binary classification is the simplest classification case — we often have more than two labels, i.e. most tasks require multi-class classification
- We can use different classifiers (machine learning models) that support multi-class classification such as neural network architectures and generative models
- We can also use a binary classifier
  — **one vs. rest** strategy: n classes require n classifiers to be trained
  highest score determines the classification label
  — **one vs. one** strategy: n classes require $\frac{n(n-1)}{2}$ classifiers to be trained
  voting scheme, class with the most votes wins

Unsupervised learning

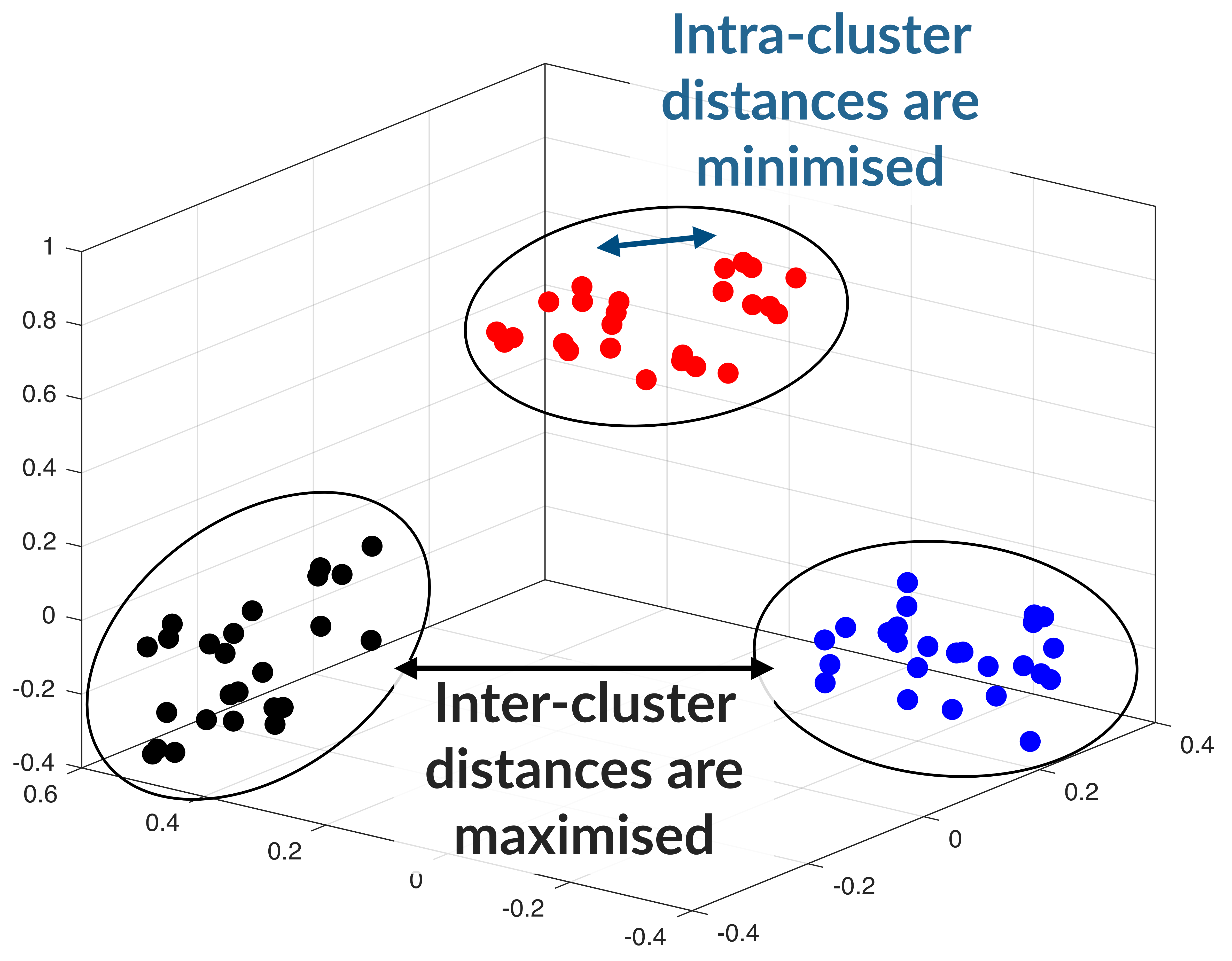
- In the previous machine learning paradigms we had an input $\mathbf{X}$ and an output $\mathbf{y}$ and we wanted to learn $f : \mathbf{X} \rightarrow \mathbf{y}$
- In unsupervised learning, there are no particular outputs or, better, response variables that we can associate our inputs with
- Our goal now is different: we want to extract some kind of pattern (a rule, an intrinsic structure) from a data set (a set of observations $\mathbf{X}$)

Unsupervised learning

- Is association rule mining a form of unsupervised learning?
- Dimensionality reduction, principal component analysis
- Clustering: almost synonymous with unsupervised learning
- Clustering groups similar observations (features) together into **clusters**

Unsupervised learning — Clustering

- A **cluster** contains data instances that are similar to each other (or if you visualise this are very close to each other in a vector space) — in very lay terms, different clusters are supposed to be capturing a different part of this vector space
- So, clustering is a grouping of data objects such that the objects within a group are similar (or related) to one another and different from (or unrelated to) the objects in other groups
- The plot shows 3 very visible clusters



Clustering — Where is it being used?

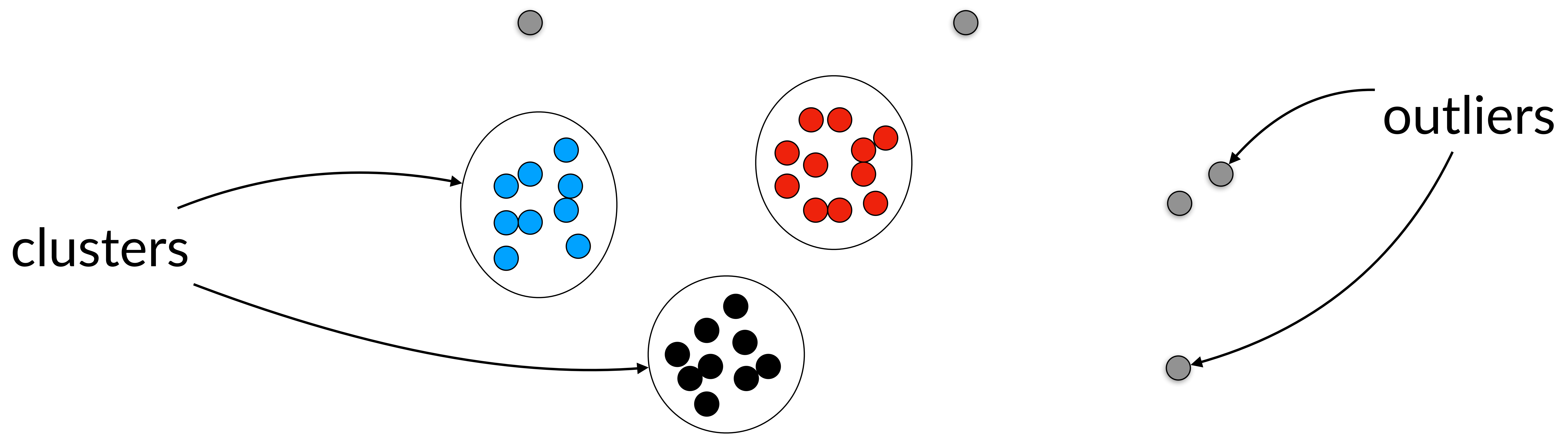
- Computational biology, e.g. understand properties of genes
- Medicine, e.g. in medical imaging
- Marketing, e.g. segment customers according to their underlying characteristics, then conduct targeted marketing
- Document clustering, topic models, text clustering in general
- Applicable to tasks that require "pattern analysis" and in many different research disciplines for analysing outcomes (e.g. in psychology, sociology, computer science, neuroscience)

Clustering — Some further key aspects

- Many different clustering algorithms / methods
— partitional, hierarchical, hard/soft, generative, and even supervised
- A distance (dissimilarity) or a similarity function is often a key component for determining clusters
- Clustering goal is to maximise the distance between different clusters (inter-cluster distance) and at the same time to minimise the distance of elements in a cluster (intra-cluster distance)
- The quality of a clustering outcome depends on the algorithm, the distance function, and eventually the specifics of an application
- However, determining the actual quality of a cluster is not always an easy task given the lack of supervision

Clustering — Outliers

- Outliers are objects that do not belong to any cluster or form clusters of very small cardinality
- In some applications we are actually interested in discovering outliers, not clusters



Clustering — Distance / similarity functions

- Let's assume we want to compare two n-dimensional observations, $\mathbf{x}$ and $\mathbf{z}$
- Let's also assume first that both $\mathbf{x}$ and $\mathbf{z}$ contain discrete values; these can be binary values (0 or 1), or specific element identifiers
- The Jaccard similarity provides a simple way to compare these observations

$$\text{Jsim}\left(\mathbf{x}, \mathbf{z}\right) = \frac{|\mathbf{x} \cap \mathbf{z}|}{|\mathbf{x} \cup \mathbf{z}|}$$

- Comparisons are element-wise
- The Jaccard similarity can (by definition) take values from 0 to 1
- It can be turned to a distance metric: $\text{Jdist}\left(\mathbf{x}, \mathbf{z}\right) = 1 - \text{Jsim}\left(\mathbf{x}, \mathbf{z}\right)$
- if $\mathbf{x} = \left[1\,0\,0\,1\,1\,1\right]$, $\mathbf{z} = \left[0\,1\,1\,0\,1\,0\right]$ then $\text{Jsim}\left(\mathbf{x}, \mathbf{z}\right) = 1/6$ and $\text{Jdist}\left(\mathbf{x}, \mathbf{z}\right) = 5/6$

Clustering — Distance / similarity functions

- Let's assume we want to compare two n-dimensional observations, $\mathbf{x}$ and $\mathbf{z}$
- Let's now assume that both $\mathbf{x}$ and $\mathbf{z} \in \mathbb{R}^n$
- Recall the L_p-norm definition: $\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$
- Popular distance measures stem from this — the input now is the difference of the vectors we want to compare — this is also known as the **Minkowski distance**
$$L_p(\mathbf{x}, \mathbf{z}) = \left(|x_1 - z_1|^p + |x_2 - z_2|^p + \ldots + |x_n - z_n|^p \right)^{1/p} = \|\mathbf{x} - \mathbf{z}\|_p$$

Clustering — Distance / similarity functions

- Let's assume we want to compare two n-dimensional observations, $\mathbf{x}$ and $\mathbf{z} \in \mathbb{R}^n$
- **Minkowski distance**

$$L_p(\mathbf{x}, \mathbf{z}) = \left(|x_1 - z_1|^p + |x_2 - z_2|^p + \ldots + |x_n - z_n|^p \right)^{1/p} = \|\mathbf{x} - \mathbf{z}\|_p$$

- For different values of $p \in \mathbb{N}_{>0}$ we can obtain common distance functions
- $p = 1$, **Manhattan** or **city block distance** or L_1-norm

$$L_1(\mathbf{x}, \mathbf{z}) = |x_1 - z_1| + |x_2 - z_2| + \ldots + |x_n - z_n|$$

- $p = 2$, **Euclidean distance** or L_2-norm

$$L_2(\mathbf{x}, \mathbf{z}) = \left[(x_1 - z_1)^2 + (x_2 - z_2)^2 + \ldots + (x_n - z_n)^2 \right]^{1/2} = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2 + \ldots + (x_n - z_n)^2}$$

Clustering — Distance / similarity functions

- Let's assume we want to compare two n-dimensional observations, $\mathbf{x}$ and $\mathbf{z} \in \mathbb{R}^n$
- **Minkowski distance**

$$L_p(\mathbf{x}, \mathbf{z}) = \left(|x_1 - z_1|^p + |x_2 - z_2|^p + \ldots + |x_n - z_n|^p \right)^{1/p} = \|\mathbf{x} - \mathbf{z}\|_p$$

- For different values of $p \in \mathbb{N}_{>0}$ we can obtain common distance functions
- We can also weight distances, if we want to give more importance to certain features, e.g.

$$L_2(\mathbf{x}, \mathbf{z}) = \sqrt{w_1 \left(x_1 - z_1\right)^2 + w_2 \left(x_2 - z_2\right)^2 + \ldots + w_n \left(x_n - z_n\right)^2}$$

Clustering algorithms — k-means

- Different approaches to clustering
  — relation between objects and classes (exclusive vs. overlapping)
  — relation between classes and classes (ordered vs. flat)

- Today we are going to see k-means: driven by the relationship to cluster representatives (or means), partitional clustering algorithm

- Construct a partition of a set of n features (objects) into a set of k clusters
  — each object belongs to exactly one cluster (hard clustering)
  — the number of clusters (k) is a setting given in advance

Clustering algorithms — k-means

- Let's assume we have a set of n m-dimensional observations, i.e. a matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$
  — the number of dimensions = number of features (m)
  — a feature i is represented by the i-th column of $\mathbf{X}$, the n-dimensional vector $\mathbf{x}_{:,i}$
  — we want to partition the features of $\mathbf{X}$ into k clusters

1. Randomly select k data points (seeds) to be the initial cluster centres. We call these centres **centroids** and in practice they are n-dimensional vectors (same size as the number of observations). Centroid j is denoted by $\mathbf{c}_j \in \mathbb{R}^n$
2. Assign each feature $\mathbf{x}_{:,i}$ to its closest centroid $\mathbf{c}_j$
3. Re-compute centroids by averaging across their members
4. If a convergence criterion is not met, go back to step 2

Clustering algorithms — k-means convergence criteria

- no or minimum re-assignments of data points to different clusters
- no or minimum change of centroids
- minimum decrease in the following cost function — the distance of all features from their centroids has converged to a minimum (C_j denotes cluster j)

$$\sum_{j=1}^{k} \sum_{\mathbf{x}_{:,i} \in C_j} \text{dist}\left(\mathbf{x}_{:,i}, \mathbf{c}_j\right)$$

- we can use different distance functions, the most common being the Euclidean distance squared, i.e.

$$\sum_{j=1}^{k} \sum_{\mathbf{x}_{:,i} \in C_j} \|\mathbf{x}_{:,i} - \mathbf{c}_j\|_2^2$$

Clustering algorithms — k-means

- Strengths
  — simple implementation
  — efficient, time complexity $\mathcal{O}\left(tkn\right)$, t number of iterations, k clusters, n observations
  — finds a local optimum
  — *no definitive evidence that any cluster algorithm performs better* (hard to evaluate anyway)

- Weaknesses
  — we need to specify k (the number of clusters)
  — sensitive to outliers
  — sensitive to initialisation

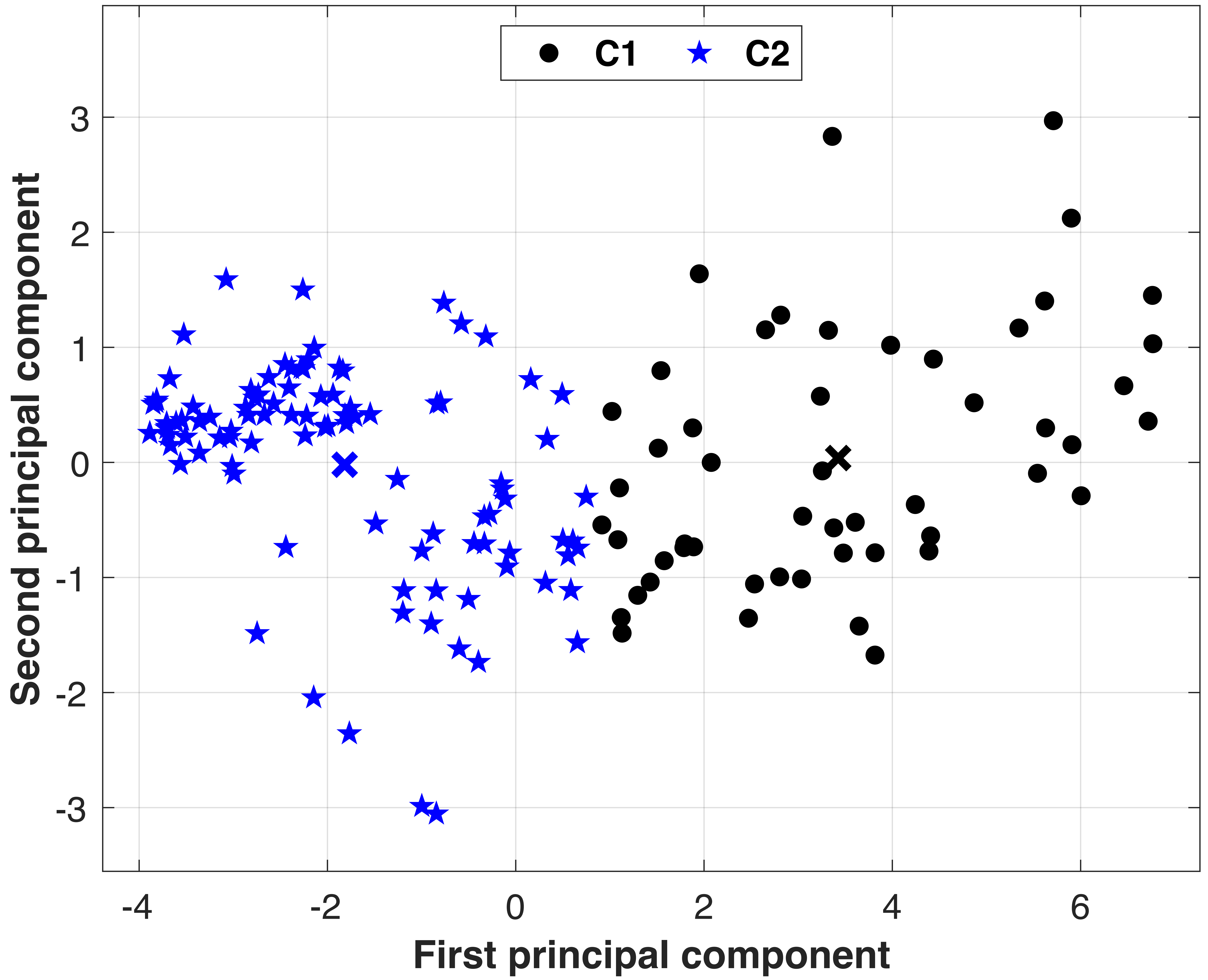
- Workarounds / improvements
  — multiple runs with different initialisations
  — non random initialisation, centroids set to the most distant observations (k-means++)

Clustering algorithms — k-means, an example

- Back to our web search activity data set
- 150 web search queries that are used to model flu rates in England
- Weekly frequency for 674 weeks, i.e. $\mathbf{X} \in \mathbb{R}_{\geq 0}^{674 \times 150}$
- To visualise the k-means clusters, I am using the two principal components (PCA) of $\mathbf{X}$ — not great in this example because they explain ~$70\,\%$ of the data's variance
- So, actually clustering applied on a matrix $\mathbf{Z} \in \mathbb{R}^{2 \times 150}$
- Download data: dropbox.com/s/rgyg190whw26qrj/data-COMP0084-intro-to-ml.zip?dl=0

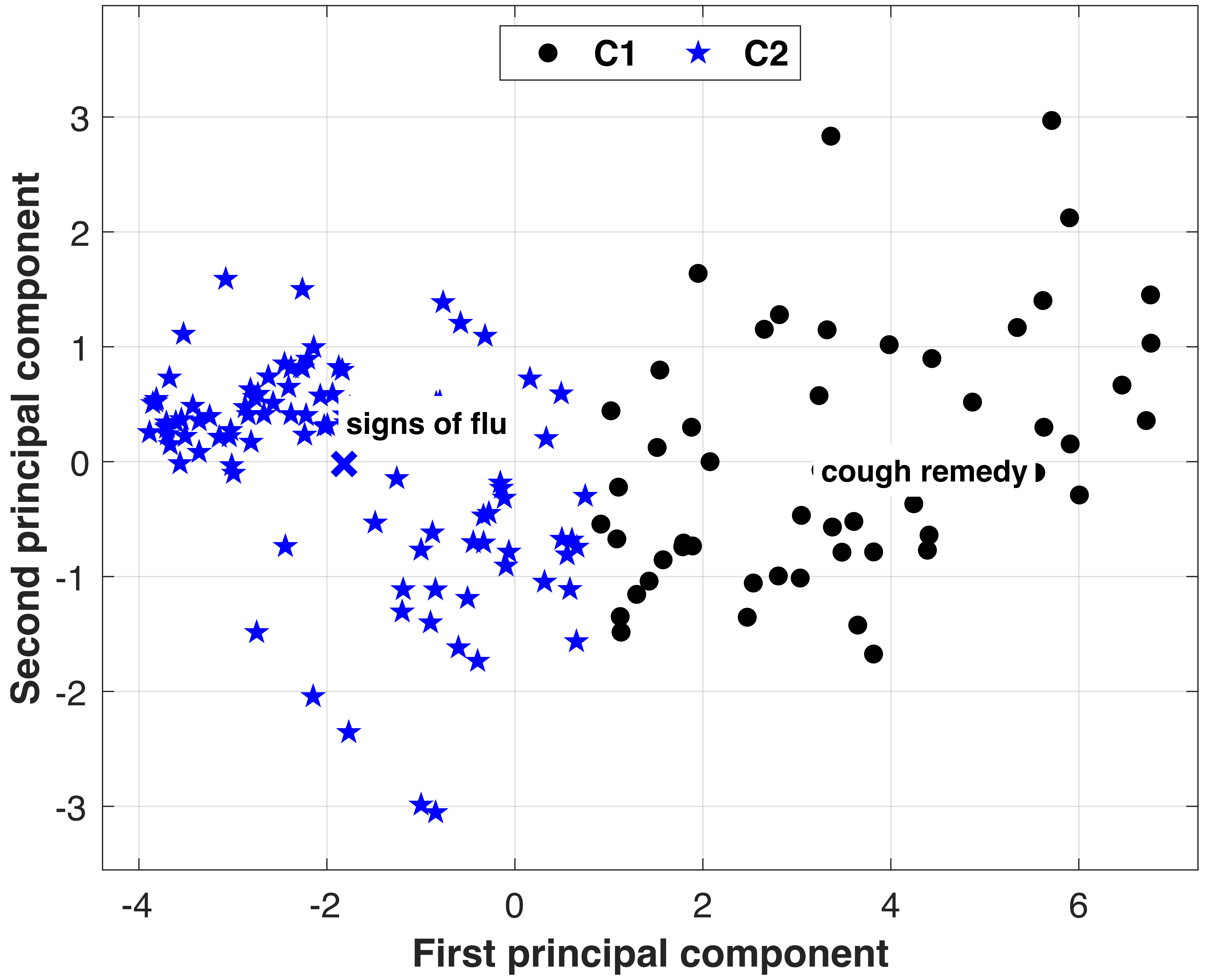
Clustering — k-means, an example

- $k = 2$
- clusters are denoted by Ci
- a cross is used to denote each cluster's centroid



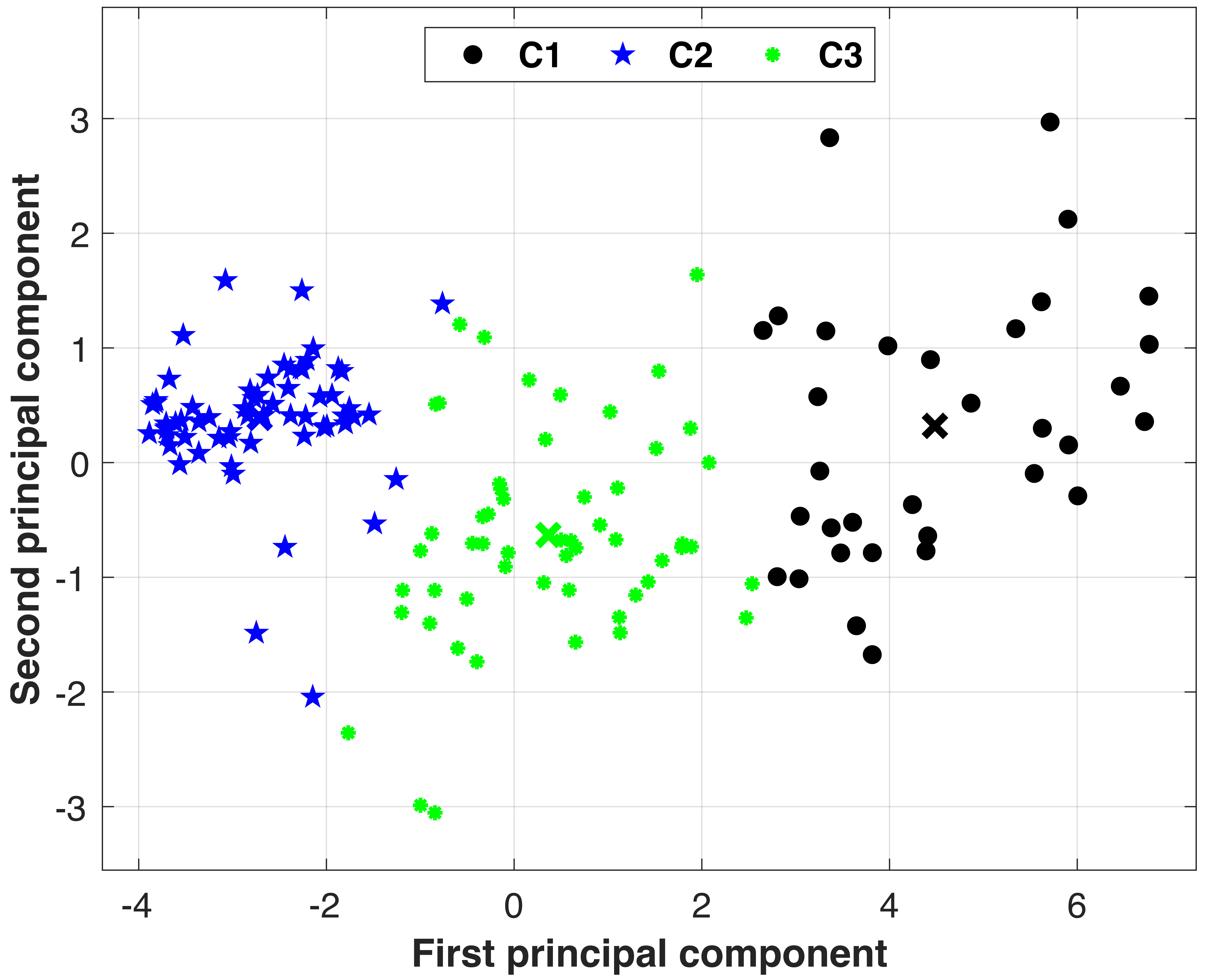
Clustering — k-means, an example

- $k = 2$
- clusters are denoted by Ci
- a cross is used to denote each cluster's centroid
- which search queries are closer to their cluster's centroid?



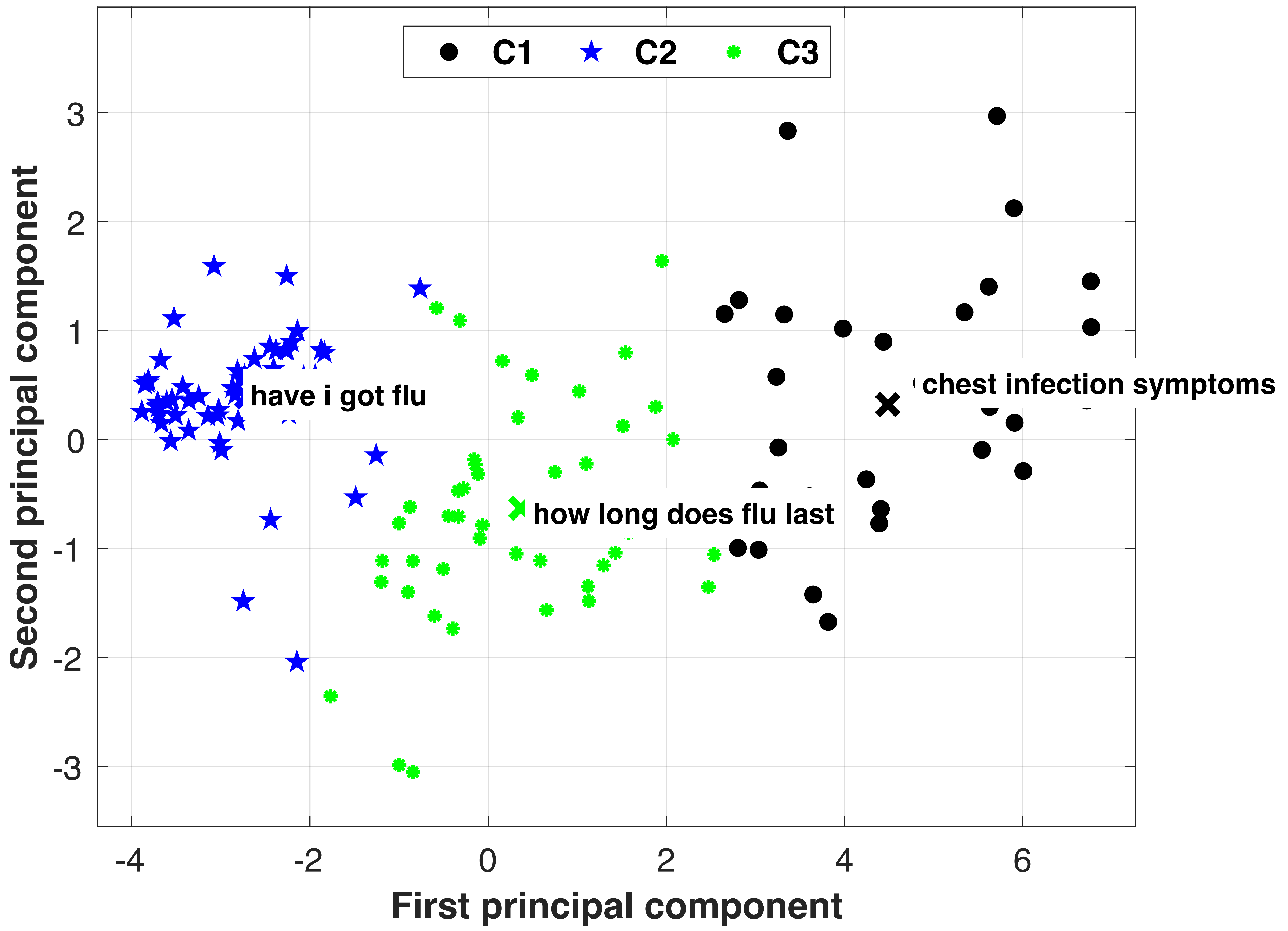
Clustering — k-means, an example

- $k = 3$
- clusters are denoted by Ci
- a cross is used to denote each cluster's centroid
- which search queries are closer to their cluster's centroid?



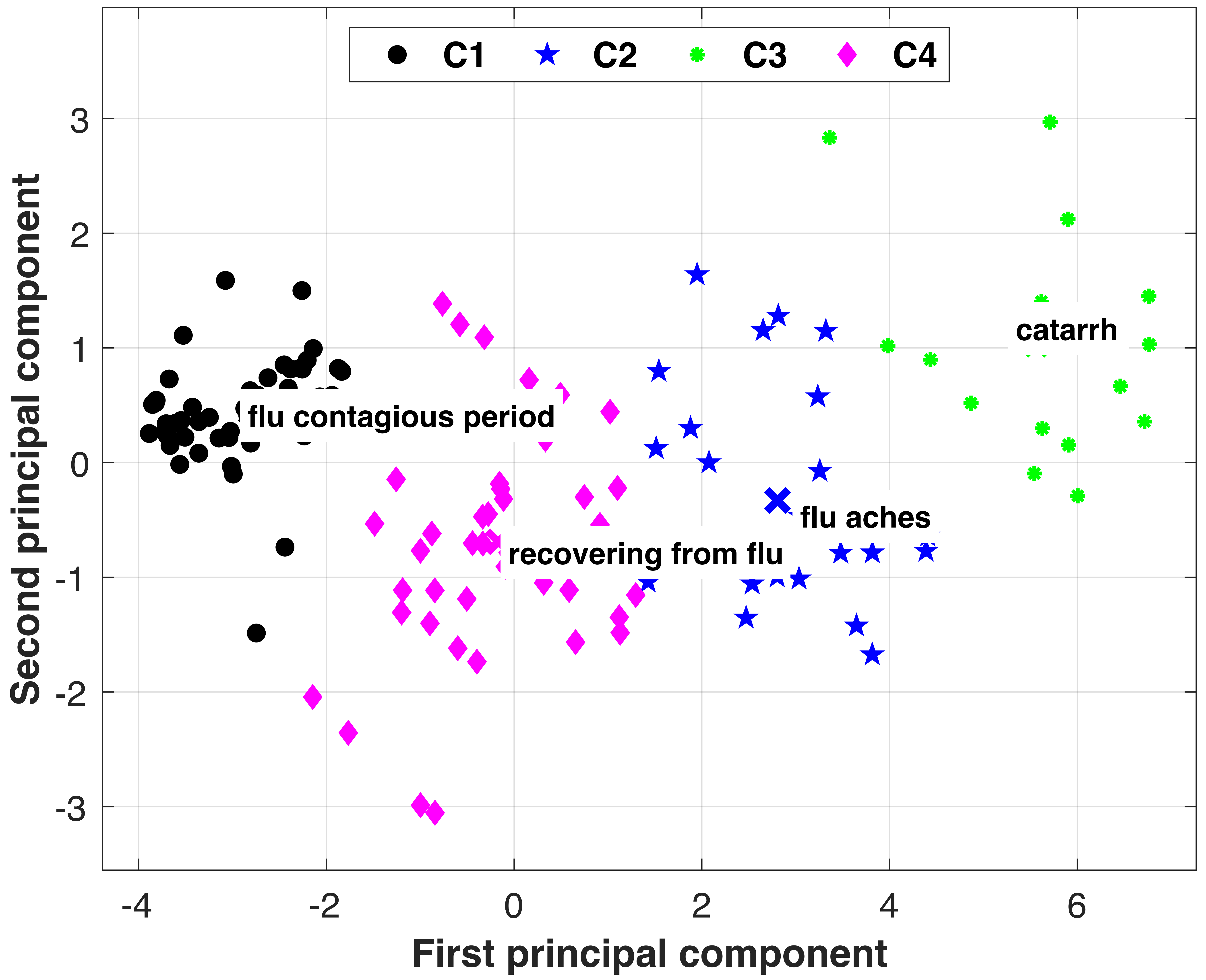
Clustering — k-means, an example

- $k = 3$
- clusters are denoted by Ci
- a cross is used to denote each cluster's centroid
- which search queries are closer to their cluster's centroid?
- does the addition of a cluster change the thematic coverage of the revised clusters?
- central queries have changed



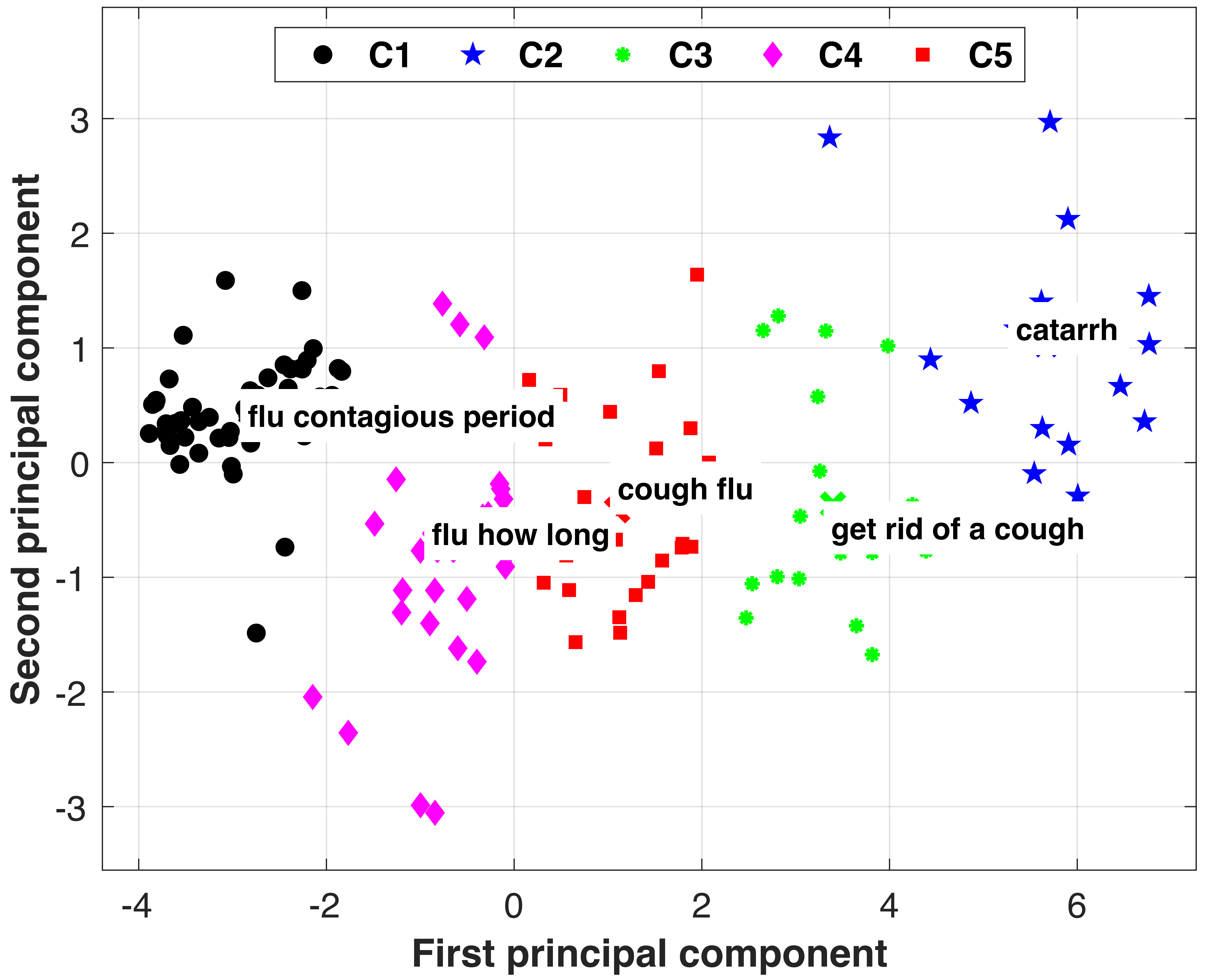
Clustering — k-means, an example

- $k = 4$
- clusters are denoted by Ci
- a cross is used to denote each cluster's centroid
- which search queries are closer to their cluster's centroid?
- does the addition of a cluster change the thematic coverage of the revised clusters?
- central queries have changed



Clustering — k-means, an example

- $k = 5$
- clusters are denoted by Ci
- a cross is used to denote each cluster's centroid
- which search queries are closer to their cluster's centroid?
- does the addition of a cluster change the thematic coverage of the revised clusters?
- central queries have changed — *partially!*



Next lectures with me

Topic models and vector semantics (*word embeddings*)

- March 2 and 3 (2 hours + 1 hour guest lecture)

Guest lecture (*self-invited*) about modelling infectious disease prevalence (*including COVID-19*) using web search activity

- March 16 (1 hour)