Information Retrieval & Data Mining [COMP0084]

Introduction to machine learning & data mining

Vasileios Lampos

Computer Science, UCL

lampos.net

@lampos

Preliminaries

- In this lecture:
  - Data mining; association rule mining (apriori algorithm)
  - Introduction to machine learning; supervised learning (regression, classification), unsupervised learning (clustering) with examples
- Useful additional reads:
  - Chapters 2, 4 of "*Web Data Mining*" by Bing Liu (2006) — cs.uic.edu/~liub/WebMiningBook.html
  - Chapters 3, 4, 14 of "*The Elements of Statistical Learning*" by Hastie, Tibshirani, and Friedman (2008) — hastie.su.domains/ElemStatLearn/
  - Chapter 5 of "*Speech and language processing*" by Jurafsky and Martin (2021) — web.stanford.edu/~jurafsky/slp3/
  - More advanced reading: Paper on estimating influenza prevalence based on Web search activity by Lampos, Miller et al. — nature.com/articles/srep12760
- Some slides were adapted from Bing Liu's course — cs.uic.edu/~liub/teach/cs583-fall-21/cs583.html

Data mining — Definition

- **Data mining** is the process of discovering (*mining*) useful patterns from or conducting inferences based on various types of *data* sources such as structured information repositories (e.g. databases), text, images, sound, video, and so on.
- **Multi-disciplinary**: machine learning (or AI more broadly), statistics, databases, information retrieval — *but the distinction between machine learning and data mining is becoming increasingly difficult, especially from an applications perspective.*
- Strong research community: Knowledge Discovery and Data Mining or **KDD** — kdd.org
- Why? Gaining knowledge from a database is not as simple issue database queries
- Applications include marketing, recommendations, scientific data analysis, and *any task involving large amounts of data*

Data mining — Association rule mining

- Today: a basic look into **Association rule mining** / **learning** — *perhaps the most important task proposed and studied by the data mining community*
- Introduced by Agrawal, Imielinski, and Swami in 1993 — dl.acm.org/doi/pdf/10.1145/170035.170072
- Applicable on categorical / discrete data (e.g. product categories, movies, songs)
- Initially used for market basket analysis to understand how products purchased by customers are related, e.g.

spaghetti → *basil*   [**support** = 0.1%, **confidence** = 25%]

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}

t_2 : {almonds, bananas}

$\dots$

t_n : {cashews, oranges, pistachios}

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\ldots$
t_n : {cashews, oranges, pistachios}

- A set of all the m **items**, $I = \{i_1, i_2, \ldots, i_m\}$
  — e.g. “almonds” is an item

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\ldots$
t_n : {cashews, oranges, pistachios}

- A set of all the m **items**, $I = \{i_1, i_2, \ldots, i_m\}$
  — e.g. "almonds" is an item
- A set of all the n **transactions**, $T = \{t_1, t_2, \ldots, t_n\}$

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\dots$
t_n : {cashews, oranges, pistachios}

- A set of all the m **items**, $I = \{i_1, i_2, \dots, i_m\}$
  — e.g. "almonds" is an item
- A set of all the n **transactions**, $T = \{t_1, t_2, \dots, t_n\}$
- A transaction t_i is a set of items, and hence $t_i \subseteq I$

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\dots$
t_n : {cashews, oranges, pistachios}

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\ldots$
t_n : {cashews, oranges, pistachios}

- An **itemset** is a set of items
  – e.g. $X = \{\text{almonds, cashews}\}$

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\dots$
t_n : {cashews, oranges, pistachios}

- An **itemset** is a set of items
  – e.g. $X = \{\text{almonds, cashews}\}$
- A k-**itemset** is an itemset with k items
  – e.g. $X = \{\text{almonds, cashews, pistachios}\}$ is a 3-itemset

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\ldots$
t_n : {cashews, oranges, pistachios}

- An **itemset** is a set of items
  – e.g. $X = \{\text{almonds, cashews}\}$
- A k-**itemset** is an itemset with k items
  – e.g. $X = \{\text{almonds, cashews, pistachios}\}$ is a 3-itemset
- A transaction t_i contains the set of items (**itemset**) $X \subseteq I$, if $X \subseteq t_i$

Association rule mining — Notation & definitions

market basket transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
$\ldots$
t_n : {cashews, oranges, pistachios}

- An **itemset** is a set of items
  — e.g. $X = \{\text{almonds, cashews}\}$
- A k-**itemset** is an itemset with k items
  — e.g. $X = \{\text{almonds, cashews, pistachios}\}$ is a 3-itemset
- A transaction t_i contains the set of items (**itemset**) $X \subseteq I$, if $X \subseteq t_i$
- An **association rule** between itemsets X, Y is an implication of the form:

$$X \rightarrow Y, \text{ where } X, Y \subset I, \text{ and } X \cap Y = \varnothing$$

Association rule mining — Support & confidence

Association rule mining — Support & confidence

- **Association rule** (a *pattern*): $X \rightarrow Y$
  – when X occurs, Y occurs with a certain *support* and *confidence*

Association rule mining — Support & confidence

- **Association rule** (a *pattern*): $X \rightarrow Y$
  – when X occurs, Y occurs with a certain *support* and *confidence*

- **support** $= \dfrac{(X \cup Y)\,.\,\text{count}}{n}$
  – probability that a transaction will contain both itemsets X and Y, $\Pr(X \cup Y)$
  – how many times X and Y appear together in all (n) transactions in T divided by n

Association rule mining — Support & confidence

- Association rule (a *pattern*): $X \rightarrow Y$
  - when X occurs, Y occurs with a certain *support* and *confidence*

- **support** $= \dfrac{(X \cup Y)\,.\,\text{count}}{n} \quad \sim \quad \Pr(X \cup Y)$

- **confidence** $= \dfrac{(X \cup Y)\,.\,\text{count}}{X\,.\,\text{count}}$
  - conditional probability that a transaction that contains X will also contain Y, $\Pr(Y \mid X)$
  - how many times a transaction that contains X also contains Y divided by the number of transactions that contain X

Association rule mining

- **Goal:** Find all association rules $(X \rightarrow Y)$ that satisfy a pre-specified (*by us!*) **minimum support** (also abbreviated as **`minsup`**) and **minimum confidence** (**`minconf`**)

- Key features
  – **Completeness**, i.e. find all rules
    Note that $X \rightarrow Y$ and $Y \rightarrow X$ are different rules. **Why?**
  – Mining with data on hard disk (*because it is not always feasible to load everything in memory*)

Association rule mining — An example

- Transactions

t_1 : {almonds, cashews, pistachios}
t_2 : {almonds, bananas}
t_3 : {apples, bananas}
t_4 : {almonds, bananas, cashews}
t_5 : {almonds, bananas, cashews, oranges, pistachios}
t_6 : {cashews, oranges, pistachios}
t_7 : {cashews, oranges, pistachios}

- Let's set
  — **`minsup`** = 30%, and
  — **`minconf`** = 80%

- **Frequent itemset** examples:
  — {almonds, cashews} with support 3/7 (> **`minsup`**)
  — {cashews, pistachios} with support 4/7
  — {cashews, oranges, pistachios} with support 3/7

- **Association rule candidates** from the above frequent itemsets
  — almonds → cashews with confidence 3/4 (< **`minconf`**, *rejected*)
  — pistachios → cashews with confidence 4/4 (> **`minconf`**, *accepted*)
  — {cashews, oranges} → pistachios with confidence 3/3 (*accepted*)

Association rule mining — Algorithms

- Large number of different association rule mining algorithms
- Different strategies, data structures, computational efficiency, memory requirements
- But their output can only be the same:
  – Given a transaction data set T, minsup, and minconf, the set of association rules in T is uniquely determined.
- Let's briefly look at a foundational algorithm for association rule mining: **Apriori**

Association rule mining — Apriori

- **Apriori** is perhaps the most popular algorithm in data mining
- "Apriori" probably because it uses "prior" knowledge of frequent itemsets
- Proposed by Agrawal and Srikant in 1994 — vldb.org/conf/1994/P487.pdf
- Same two steps (*that we've seen previously*)
  - find all the itemsets with a minimum support (frequent itemsets)
  - then use the frequent itemsets to generate association rules

Apriori — Identify frequent itemsets

- The key idea of Apriori is the downward closure property (also known as the “Apriori property”):
  — Any subset of a frequent itemset is also a frequent itemset
  = Any subset of an itemset whose support is ≥ `minsup` has also support that is ≥ `minsup`

- If the itemset {a, b, c, d} with 4 items is frequent, then the $\left(2^4 - 2\right)$ non-empty sub-itemsets will also be frequent. These are: {a}, {b}, {c}, {d}, {a, b}, {a, c}, {a, d}, {b, c}, {b, d}, {c, d}, {a, b, c}, {a, b, d}, {a, c, d}, and {b, c, d}.

- **Contraposition:** if an itemset is not frequent, then any of its supersets cannot be frequent

Apriori — The gist of the algorithm

- Apriori is an iterative algorithm
  - given a minimum support
  - find all frequent 1-itemsets (denoted by `F[1]` in the source code)
  - use those to find all frequent 2-itemsets, and so on
    > `C[2]` is a list of frequent 2-itemset candidates based on `F[1]`
    > `F[2]` $\subseteq$ `C[2]` is a list with the frequent 2-itemsets
  - in each iteration k of the algorithm only consider itemsets that contain some frequent $(k-1)$-itemset

Apriori — An important detail (*item ordering*)

- Items should be sorted according to a sorting scheme — i.e. lexicographic order
- This order will be used throughout the algorithm as it helps to reduce redundant passes on the data, e.g. the frequent itemset {a, b, c, d} is identical to the frequent itemsets {c, d, a, b} or {b, a, d, c} — we only need to deal with {a, b, c, d} once.

Apriori — Pseudocode of the algorithm (*part 1*)

```
% T: all the transactions, MINSUP: frequent itemset minimum support
function apriori(T, MINSUP):
  % C[1] count of 1-itemsets, n transactions in T
  C[1], n ← initial-pass(T)
  % F[1] is the set of frequent 1-itemsets
  F[1] ← {f | f in C[1] AND f.count/n ≥ MINSUP}
  for k = 2; F[k-1] ≠ ∅; k++:
    % use the (k-1)—itemsets to generate k-itemset candidates, C[k]
    C[k] ← generate-candidates(F[k-1])
    for each transaction t in T:
      for each candidate c in C[k]:
        if c is in t:
          c.count++
    F[k] ← {c in C[k] | c.count/n ≥ MINSUP}
  return F
```

Apriori — Candidate itemset generation

- The `generate-candidates` function takes the $(k-1)$-frequent itemsets, denoted by `F[k-1]` in the source code, and returns a superset of k-frequent itemset candidates, denoted by `C[k]`

- Two steps
  - **Join**: generate all possible candidate k-itemsets `C[k]` based on `F[k-1]`
  - **Prune**: remove those candidates in `C[k]` that cannot be frequent, i.e. if a candidate itemset has a subset of items that is not already identified as a frequent itemset it should be removed

Apriori — Pseudocode of the algorithm (*part 2*)

```
% using frequent (k-1)-itemsets generate frequent k-itemset candidates
function generate-candidates(F[k-1]):
  C[k] ← ∅
  for every f1, f2 in F[k-1] where:
    a = f1 - f2 AND                         % set difference
    b = f2 - f1 AND                         % set difference
    (a AND b) are both of size 1 AND        % f1 and f2 differ by 1 element
    a < b do:                               % lexicographic comparison
      c ← {f1,b}                            % frequent k-itemset candidate
      C[k] ← {C[k],c}
      for each (k-1)-subset s of c do:
        if s not in F[k-1]:
          delete c from C[k]                % pruning non-frequent candidates

  return C[k]
```

Apriori — An example

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

Let’s use Apriori to identify all frequent itemsets with minimum support of 30%

Apriori — An example

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

```
C[1]:{almonds:4/7, apples:1/7, bananas:4/7, cashews:5/7, oranges:3/7,
      pistachios:4/7}

F[1]:{almonds, bananas, cashews, oranges, pistachios}

C[2]:{ {almonds, bananas}:3/7,      {almonds, cashews}:3/7,
       {almonds, oranges}:1/7,      {almonds, pistachios}:2/7,
       {bananas, cashews}:2/7,      {bananas, oranges}:1/7,
       {bananas, pistachios}:1/7,   {cashews, oranges}:3/7,
       {cashews, pistachios}:4/7,   {oranges, pistachios}:3/7 }
```

Apriori — An example

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

```
C[2]:{ {almonds, bananas}:3/7,       {almonds, cashews}:3/7,
       {almonds, oranges}:1/7,       {almonds, pistachios}:2/7,
       {bananas, cashews}:2/7,       {bananas, oranges}:1/7,
       {bananas, pistachios}:1/7,    {cashews, oranges}:3/7,
       {cashews, pistachios}:4/7,    {oranges, pistachios}:3/7 }

F[2]:{ {almonds, bananas}, {almonds, cashews}, {cashews, oranges},
       {cashews, pistachios}, {oranges, pistachios} }
```

Apriori — An example

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

```
F[2]:{ {almonds, bananas}, {almonds, cashews}, {cashews, oranges},
       {cashews, pistachios}, {oranges, pistachios} }

C[3]:{ {almonds, bananas, cashews}:2/7,
       {cashews, oranges, pistachios}:3/7 }
```

Apriori — An example

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

F[2]:{ {almonds, bananas}, {almonds, cashews}, {cashews, oranges}, {cashews, pistachios}, {oranges, pistachios} }

C[3]:{ ~~{almonds, bananas, cashews}:2/7, {cashews, oranges, pistachios}:3/7~~ } *** Incorrect ***

C[3]:{ {cashews, oranges, pistachios}:3/7 }

entry {almonds, bananas, cashews} will be pruned because {bananas, cashews} is not in F[2]

F[3]:{ {cashews, oranges, pistachios} }

Apriori — An example

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

Apriori identified the following *frequent* itemsets with a minimum support of 30%:

```
F[1]:{almonds:4/7, bananas:4/7, cashews:5/7, oranges:3/7, pistachios:4/7}

F[2]:{ {almonds, bananas}:3/7,      {almonds, cashews}:3/7,
       {cashews, oranges}:3/7,      {cashews, pistachios}:4/7,
       {oranges, pistachios}:3/7 }

F[3]:{ {cashews, oranges, pistachios}:3/7 }
```

Apriori — Generating association rules from frequent itemsets

- Frequent itemsets do not directly provide association rules
- For each frequent itemset F
  For each non-empty subset A of F (*no repetitions*)
  - $B = F - A$
  - $A \rightarrow B$ is an association rule if confidence $(A \rightarrow B) \geq$ **`minconf`**

$$\text{support}\,(A \rightarrow B) = \text{support}\,(A \cup B) = \text{support}\,(F)$$

$$\text{confidence}\,(A \rightarrow B) = \frac{\text{support}\,(A \cup B)}{\text{support}\,(A)}$$

Apriori — Generating association rules (*example*)

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

`minsup` = 30%, **`minconf`** = 80%, let's use `F[3]:{ {cashews, oranges, pistachios}:3/7 }`

```
A = {{cashews, oranges}, {cashews, pistachios}, {oranges, pistachios},
     {cashews}, {oranges}, {pistachios}}
```

<u>A → B</u>		
{cashews, oranges}	→ pistachios	confidence = 1
{cashews, pistachios}	→ oranges	confidence = 0.75
{oranges, pistachios}	→ cashews	confidence = 1
cashews	→ {oranges, pistachios}	confidence = 0.6
oranges	→ {cashews, pistachios}	confidence = 1
pistachios	→ {cashews, oranges}	confidence = 0.75

Apriori — Generating association rules (*example*)

```
t[1]: {almonds, cashews, pistachios}
t[2]: {almonds, bananas}
t[3]: {apples, bananas}
t[4]: {almonds, bananas, cashews}
t[5]: {almonds, bananas, cashews, oranges, pistachios}
t[6]: {cashews, oranges, pistachios}
t[7]: {cashews, oranges, pistachios}
```

`minsup` = 30%, **`minconf`** = 80%, let's use `F[3]:{ {cashews, oranges, pistachios}:3/7 }`

```
A = {{cashews, oranges}, {cashews, pistachios}, {oranges, pistachios},
     {cashews}, {oranges}, {pistachios}}
```

A → B		
`{cashews, oranges}` → `pistachios`		**confidence = 1**
`{cashews, pistachios}` → `oranges`		confidence = 0.75
`{oranges, pistachios}` → `cashews`		**confidence = 1**
`cashews` → `{oranges, pistachios}`		confidence = 0.6
`oranges` → `{cashews, pistachios}`		**confidence = 1**
`pistachios` → `{cashews, oranges}`		confidence = 0.75

Apriori — Generating association rules from frequent itemsets

- To obtain an association rule $A \rightarrow B$, we need to compute the quantities: support $(A \cup B)$ and support (A)
- This information has already been recorded during itemset generation. No need to access the raw transaction data any longer.
- Not as time consuming a frequent itemset generation, although there are efficient algorithms to generate association rules as well

The (*very*) basics of machine learning

— definition
— supervised learning (*regression*, *classification*)
— unsupervised 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

$\|\mathbf{x}\|_k = \left(\sum_{i=1}^{n} |x_i|^k \right)^{\frac{1}{k}}$ denotes the L_k-norm of $\mathbf{x} \in \mathbb{R}^n$

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

$$< \mathbf{x}_1, y_1 >$$
$$< \mathbf{x}_2, y_2 >$$
$$\dots$$
$$< \mathbf{x}_n, y_n >$$

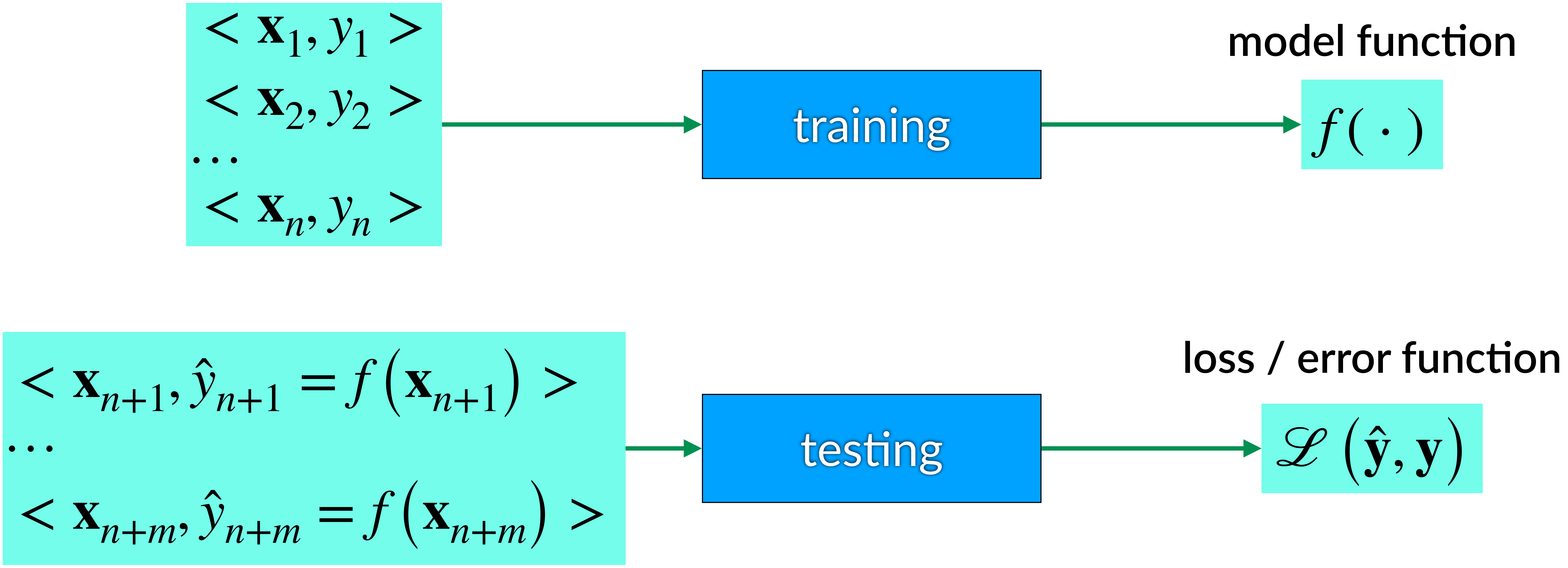
$\rightarrow$ training $\rightarrow$ $f(\cdot)$ (model function)

$$< \mathbf{x}_{n+1}, \hat{y}_{n+1} = f\left(\mathbf{x}_{n+1}\right) >$$
$$\dots$$
$$< \mathbf{x}_{n+m}, \hat{y}_{n+m} = f\left(\mathbf{x}_{n+m}\right) >$$

$\rightarrow$ testing $\rightarrow$ $\mathcal{L}\left(\hat{\mathbf{y}}, \mathbf{y}\right)$ (loss / error function)

- If $\mathscr{L}\left(\hat{\mathbf{y}}, \mathbf{y}\right)$ is “relatively” small, then our model might be learning from experience
- But what makes an error “relatively” small?
  We need to have a solid reference loss value.

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



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

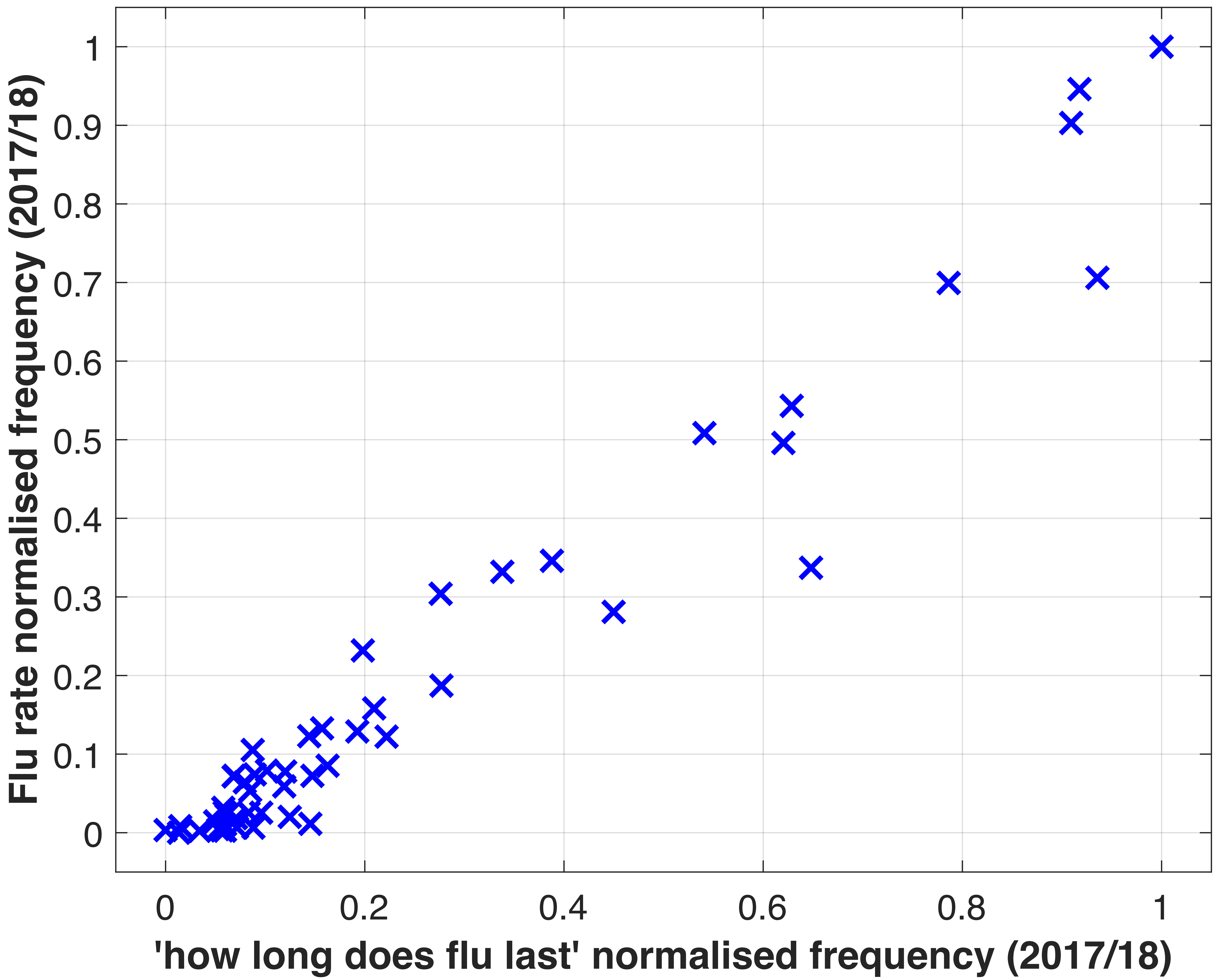
  Examples: predict a time series trend (e.g. in finance), estimate the prevalence of a condition in epidemiology

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

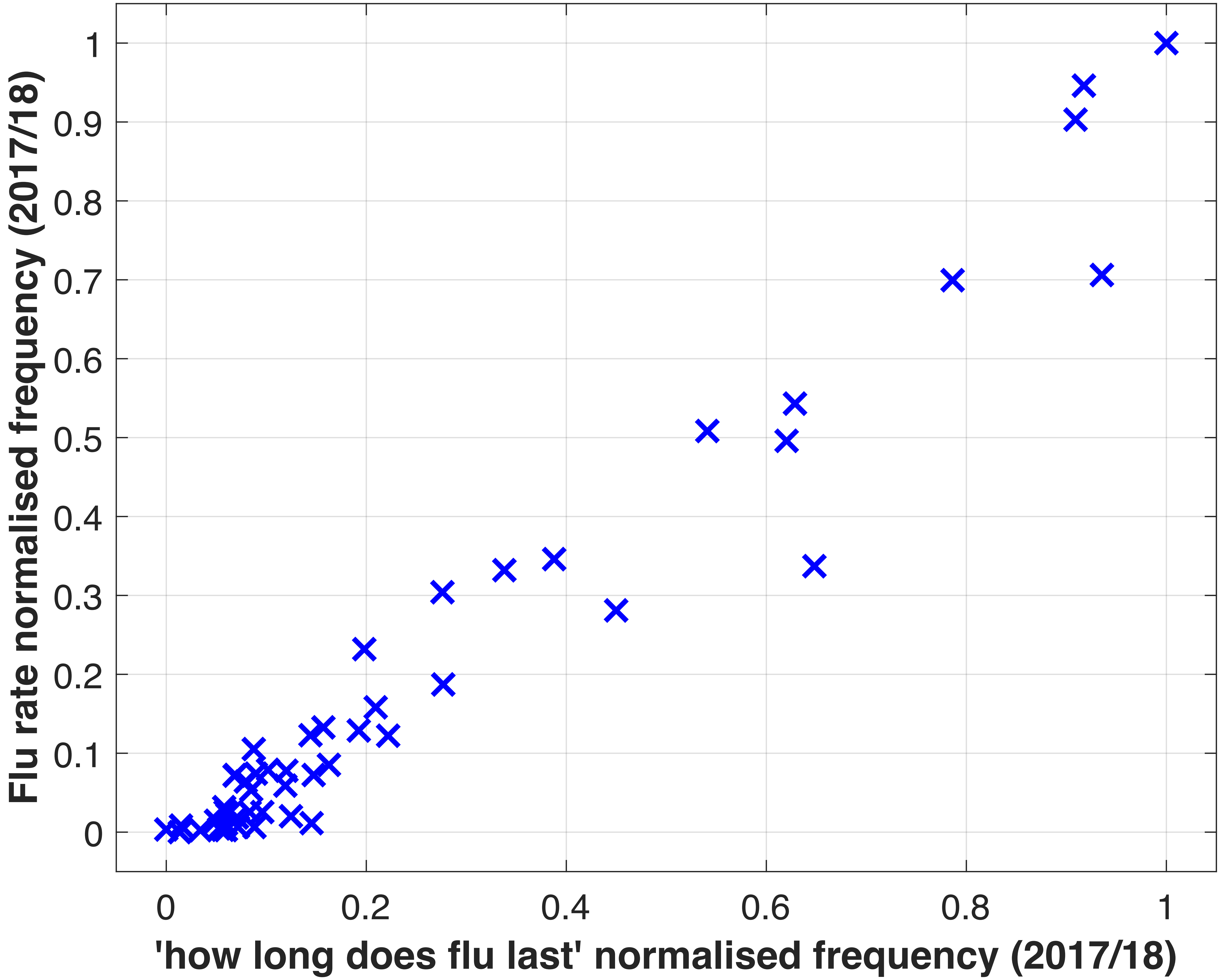
  i.e. learn $f : \mathbf{X} \in \mathbb{R}^{n \times m} \rightarrow \mathbf{y} \in \{1, 2, \ldots, C\}$

  Examples: detect spam email, medical imaging, text classification, language models

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



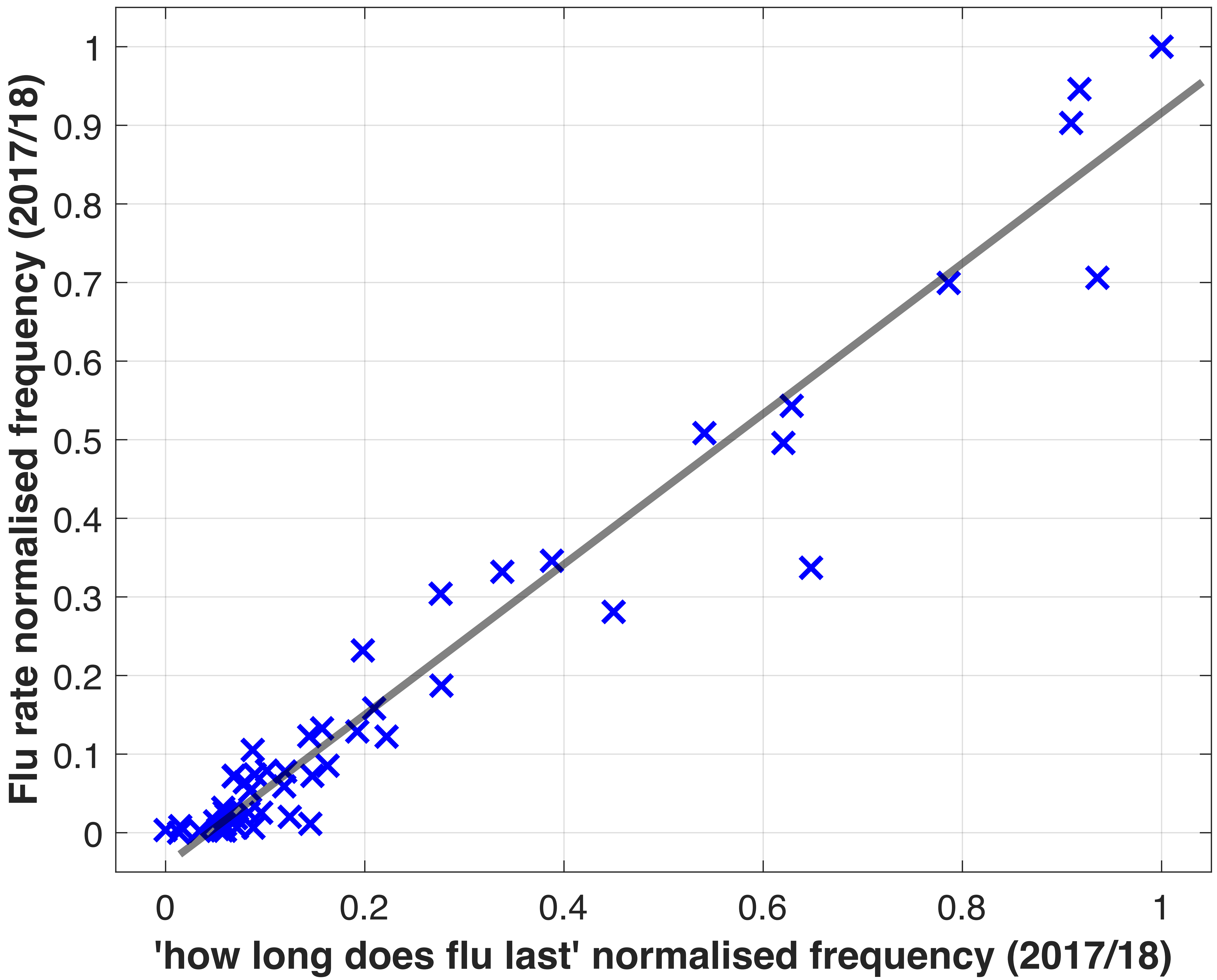
- Estimate the **prevalence of influenza-like illness** in England based on the **frequency of the search query** "*how long does flu last*"
- Linearly related, bivariate correlation of 0.975
- Can we capture this relationship with a straight line?
- **Data:** dropbox.com/s/rgyg190whw26qrj/data-COMP0084-intro-to-ml.zip?dl=0



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

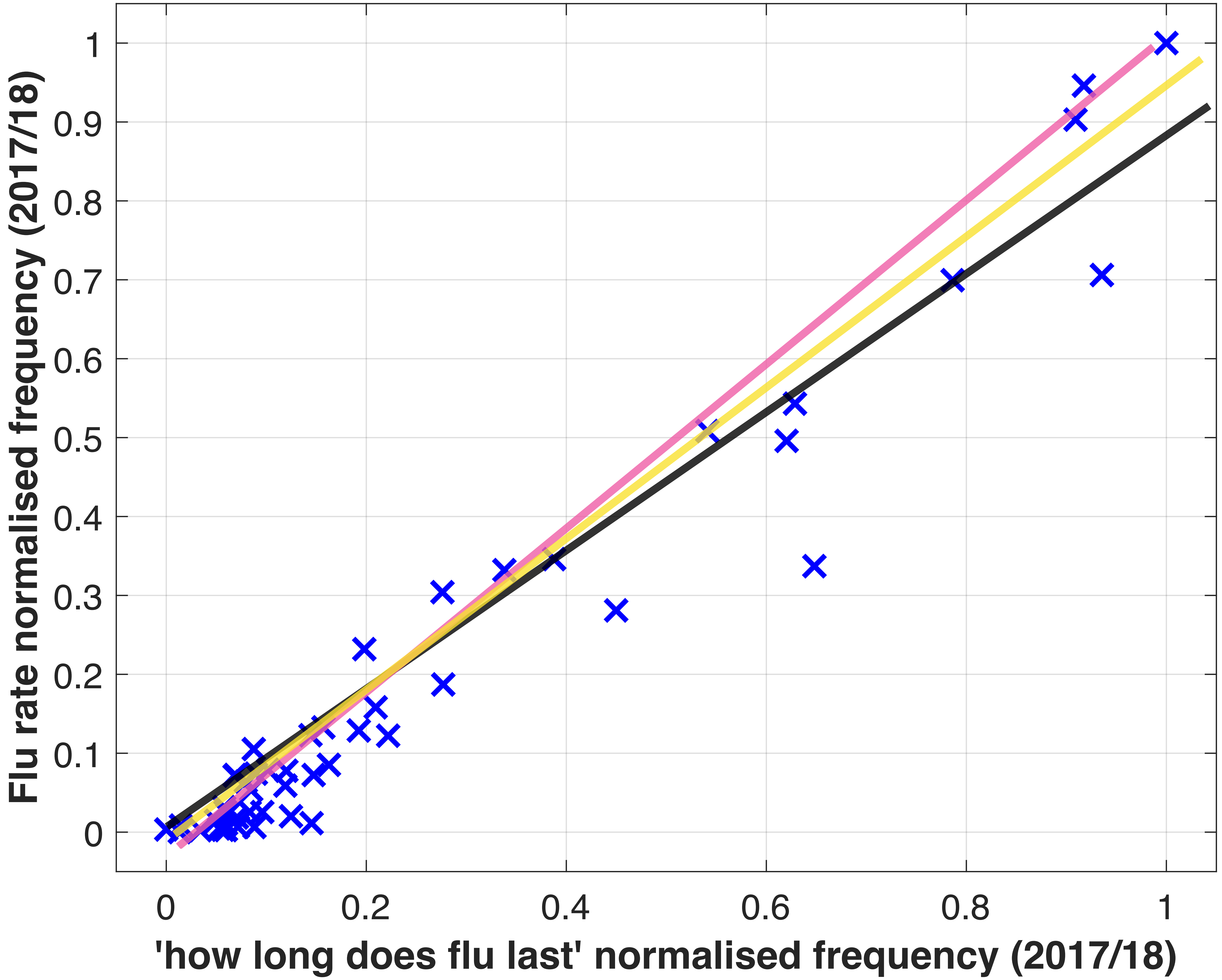
- Linearly related, bivariate correlation of 0.975

- Can we capture this relationship with a straight line?



Supervised learning — Regression

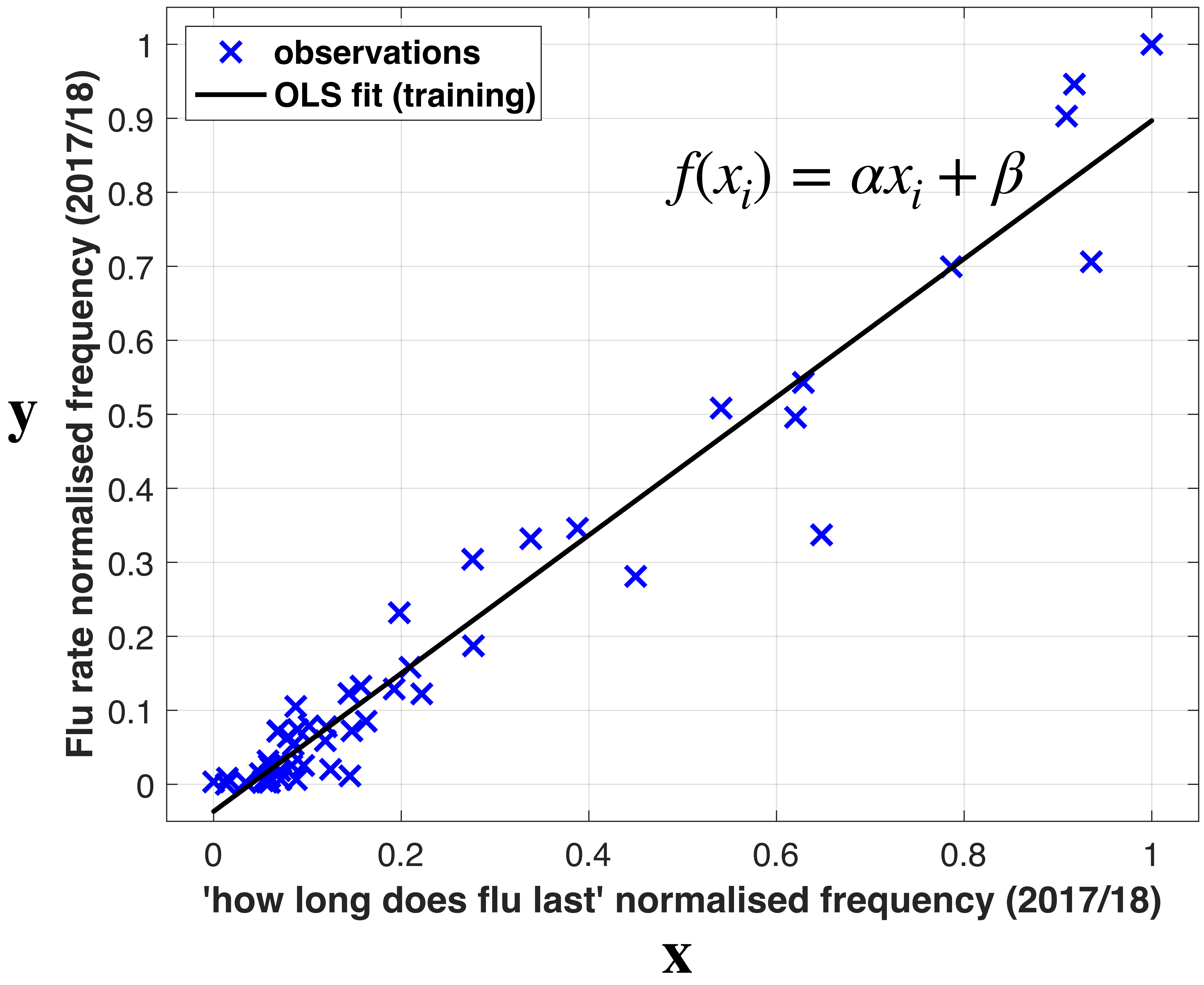
- Estimate the **prevalence of influenza-like illness** in England based on the **frequency of the search query** “*how long does flu last*”
- Linearly related, bivariate correlation of 0.975
- Can we capture this relationship with a straight line?
- Which line is the “best” though?



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

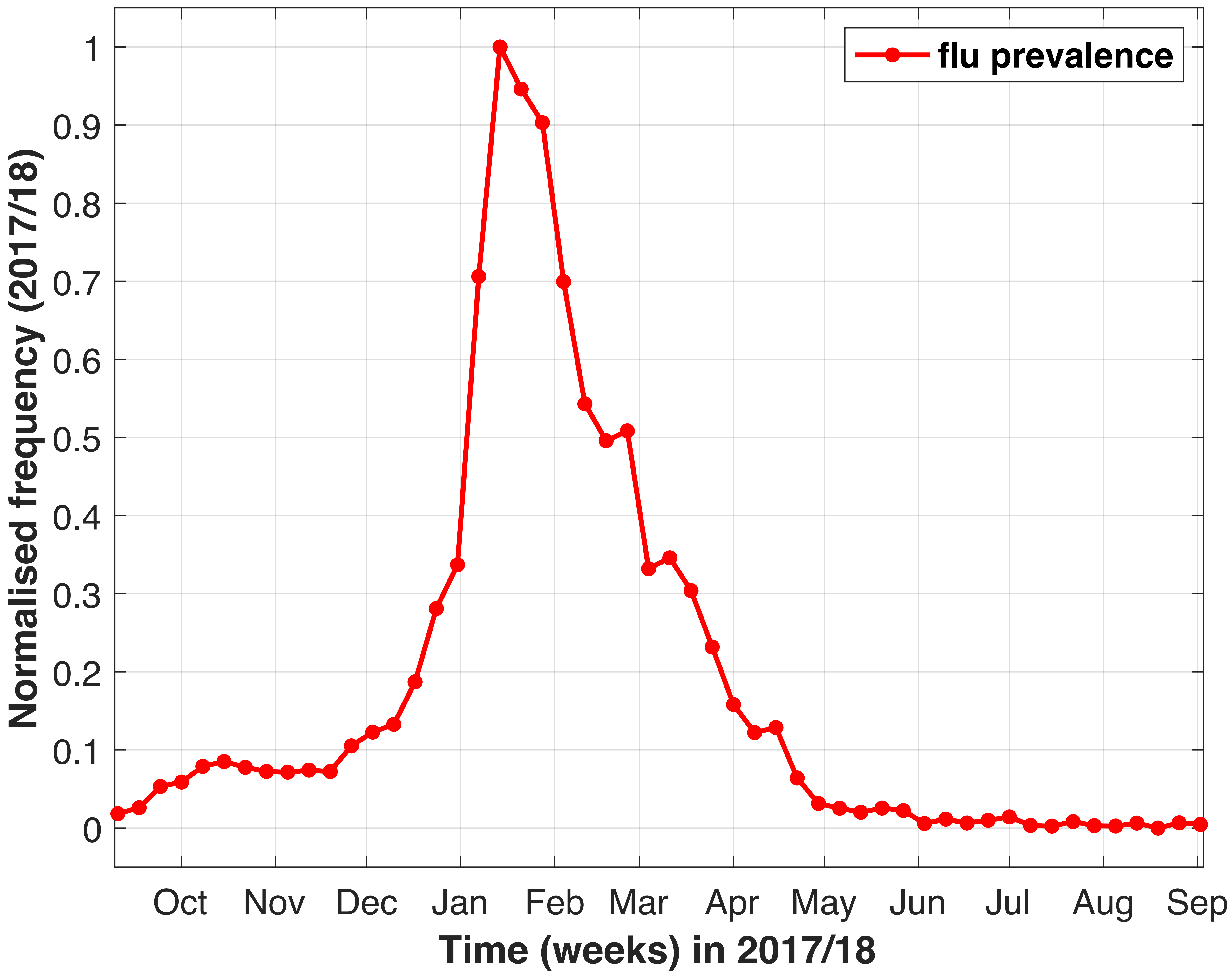
- $\mathbf{y}$ denotes the weekly influenza-like illness prevalence in England from September 2017 until the end of August 2018
- $\mathbf{x}$ denotes the corresponding weekly frequency of the search query "how long does flu last" (Google) for the same time period
- We want to learn a linear mapping 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

- $\mathbf{y}$ ~ 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



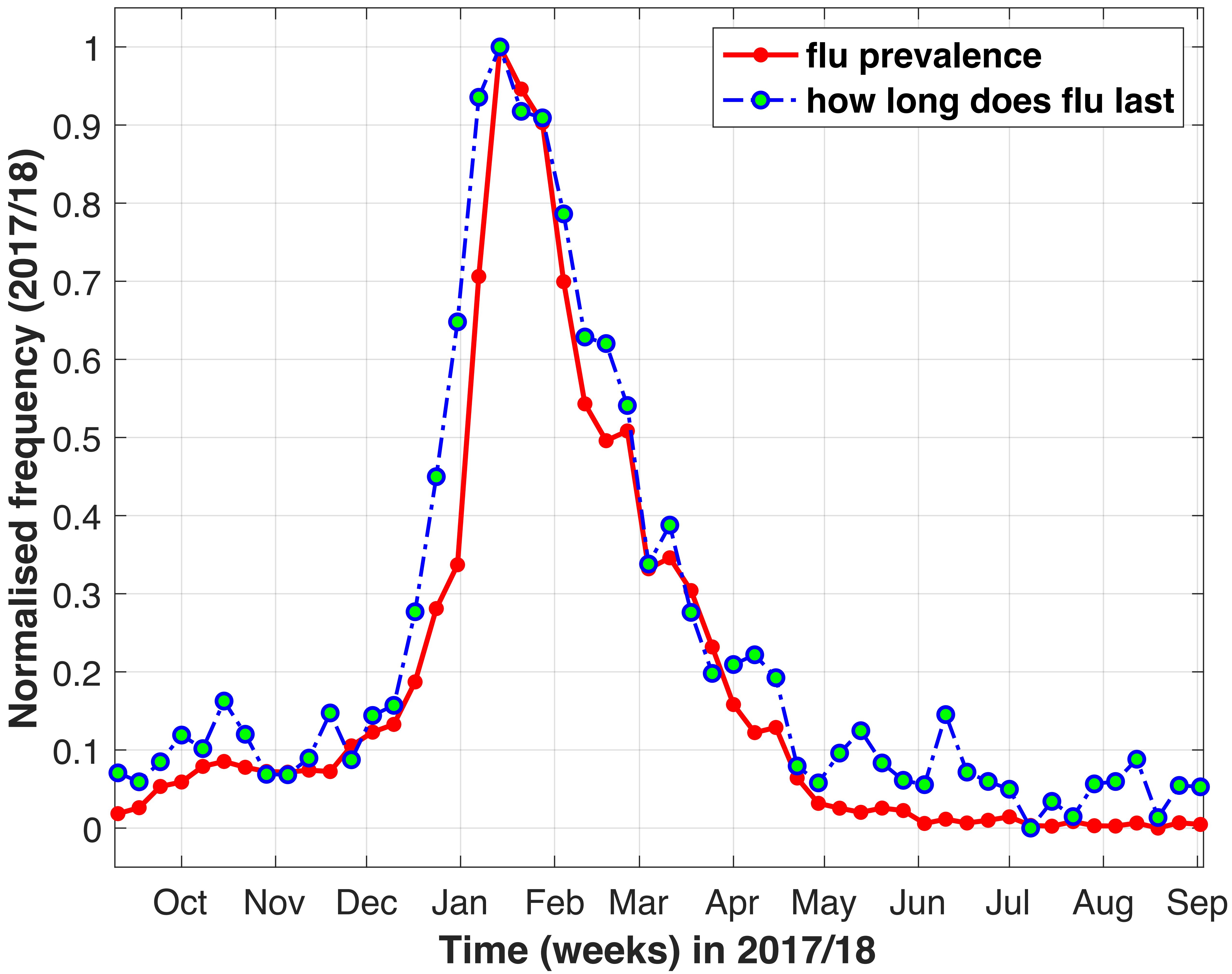
Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ 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



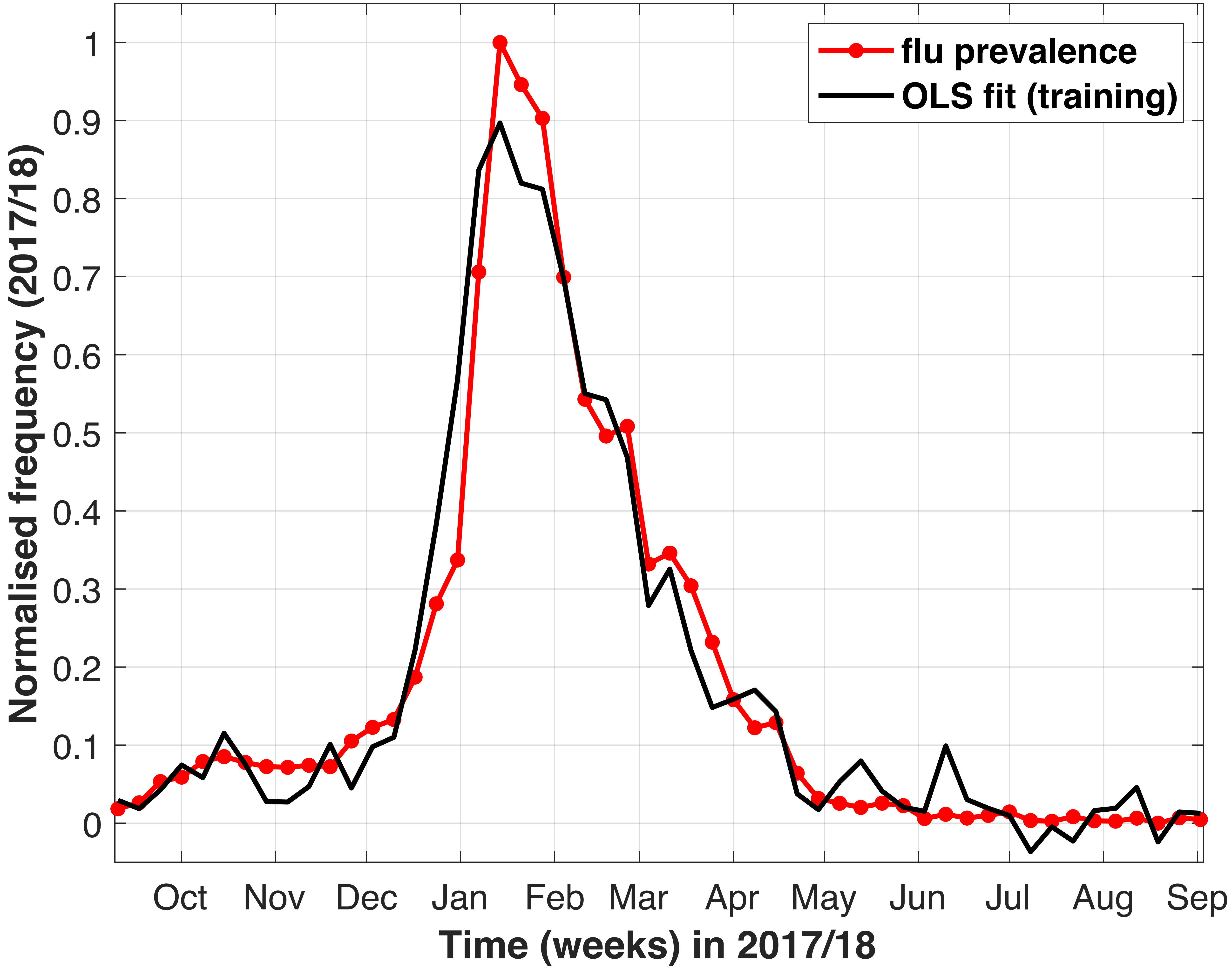
Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ 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

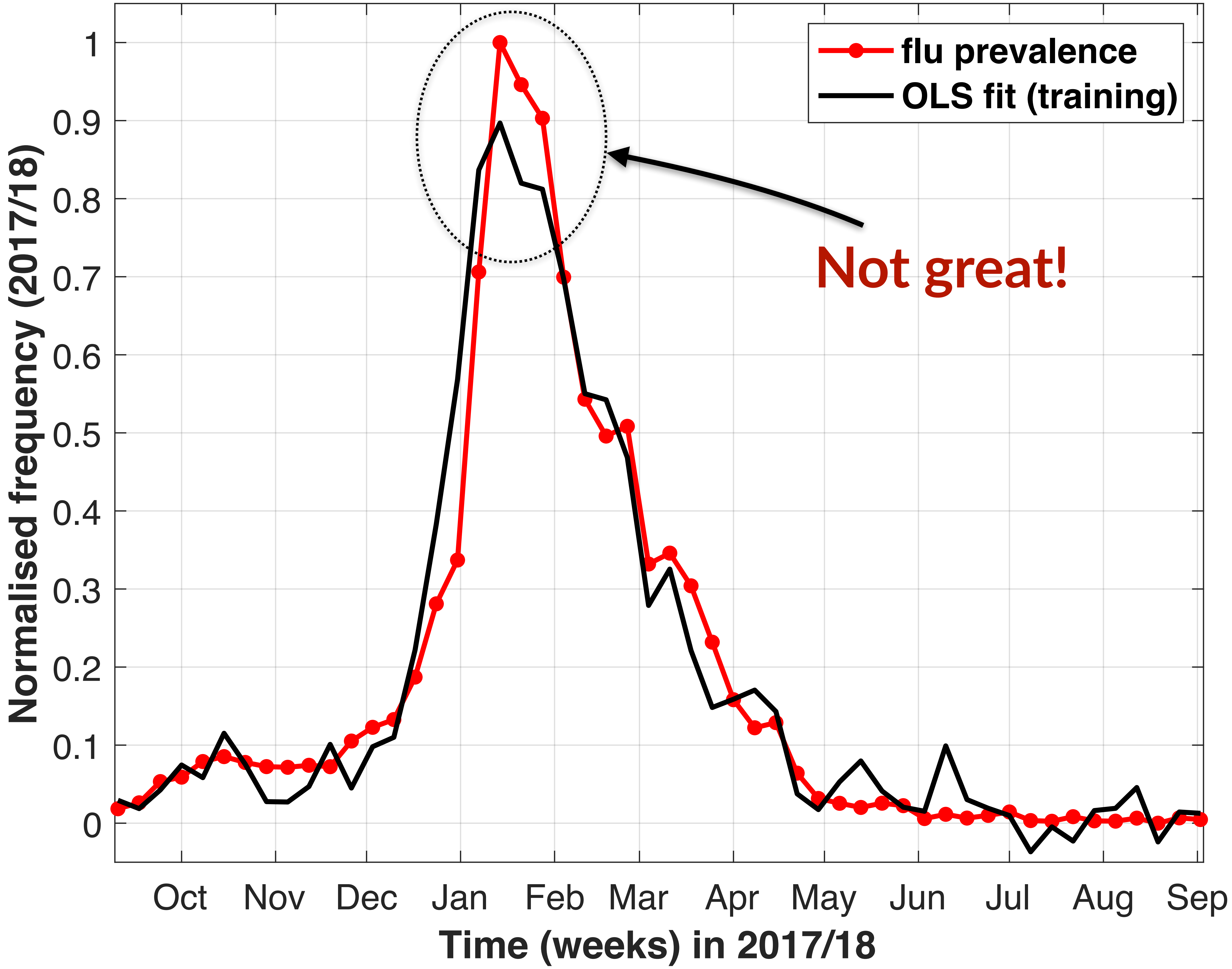


Supervised learning — OLS regression, *alternative point of view*

- $\mathbf{y}$ ~ 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



- $\mathbf{y}$ ~ 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

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

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*): $\mathscr{L}(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$

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*): $\mathscr{L}(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$
- This can also be written as: $\mathscr{L}(\mathbf{w}) = \mathscr{L}\left(\alpha, \beta\right) = \sum_{i=1}^{n}\left(\alpha x_i + \beta - y_i\right)^2$

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 $\mathbf{w}$: $\dfrac{\partial \mathscr{L}}{\partial \mathbf{w}} = -2\mathbf{X}^{\top}\mathbf{y} + 2\mathbf{X}^{\top}\mathbf{X}\mathbf{w}$

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*): $\mathscr{L}\left(\mathbf{w}\right) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$
- 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 $\beta = -0.036631$

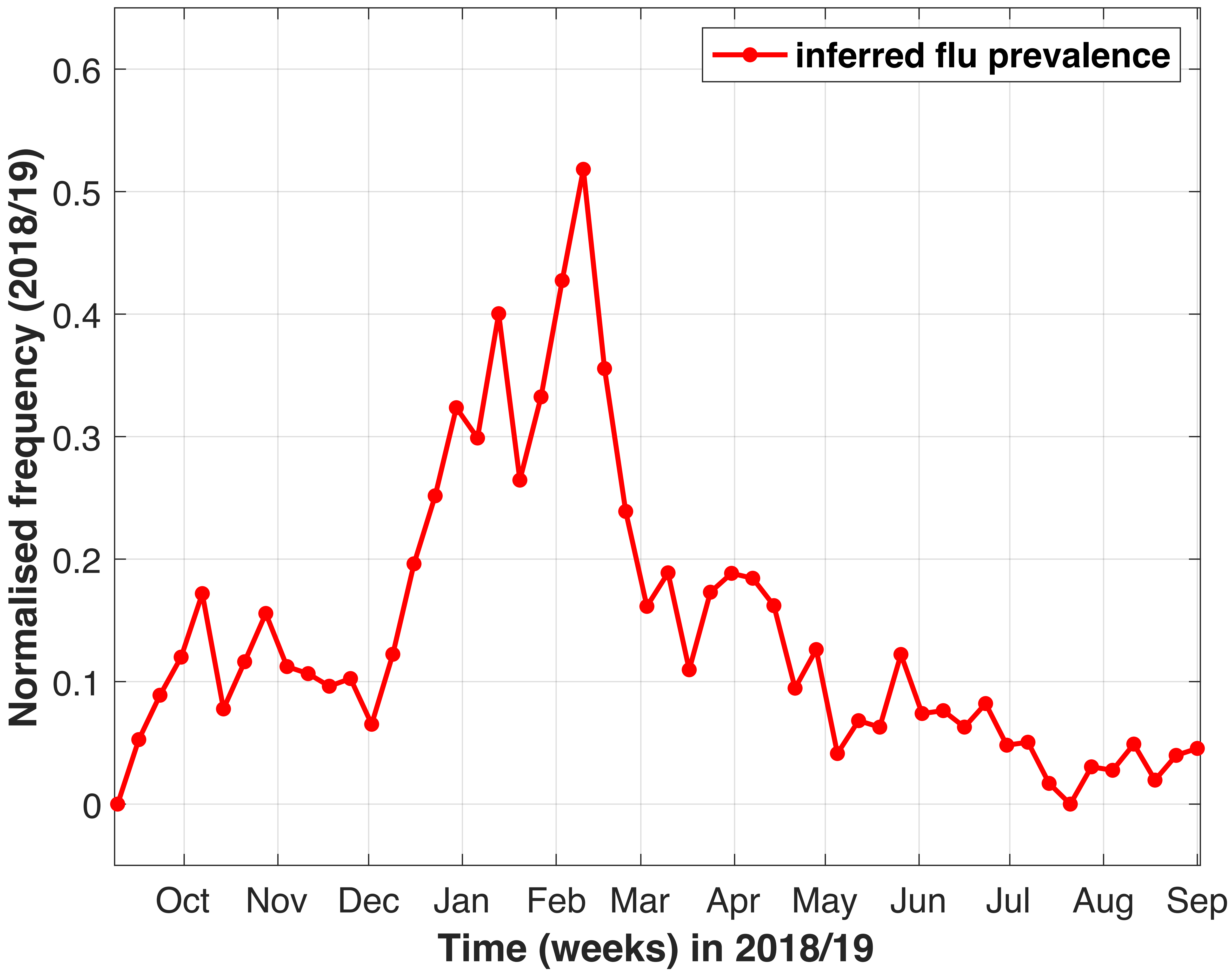
- The question now becomes, **how well will this model do in the next flu season**?

- 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*"

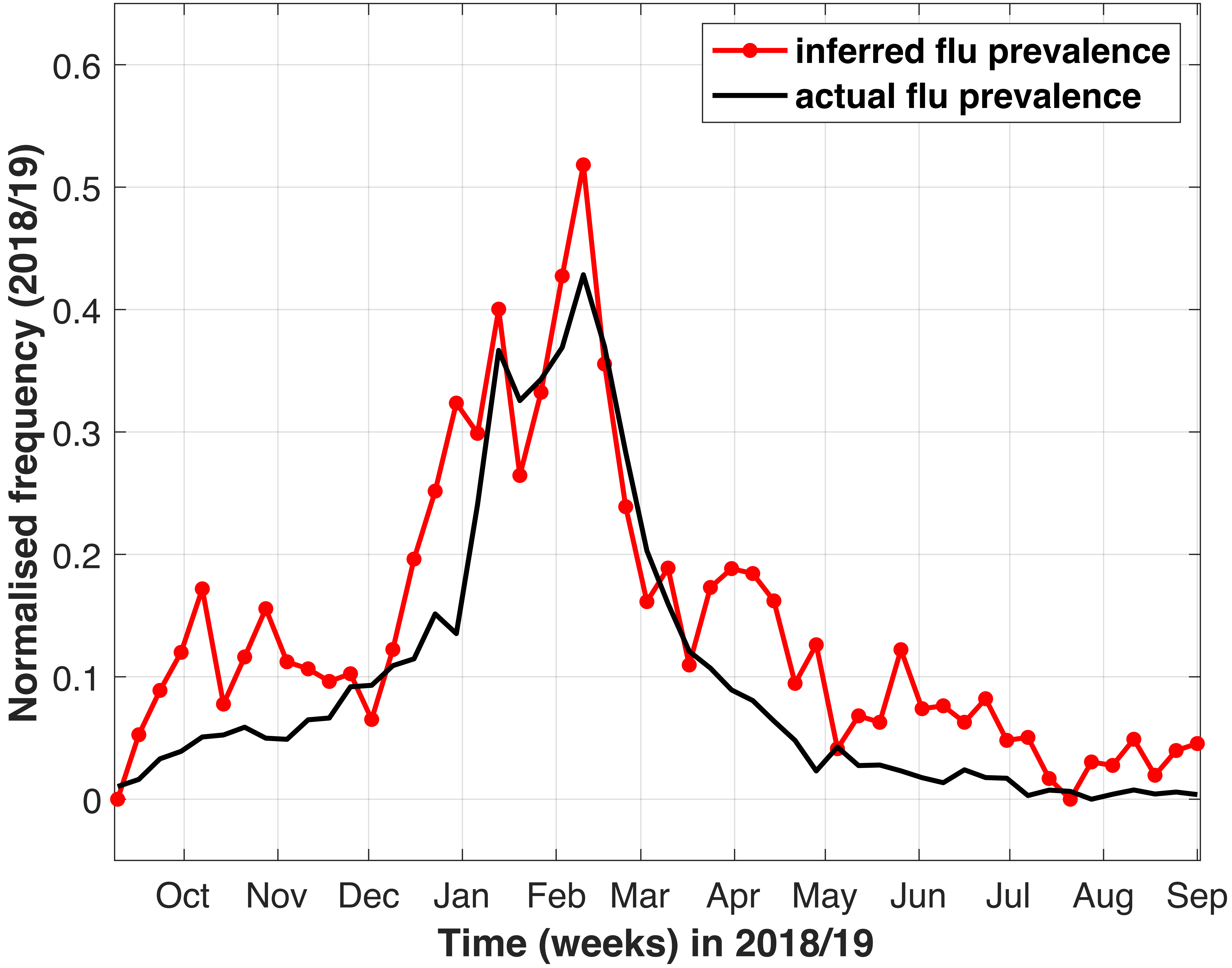
- And then compare it with the actual flu prevalence in England for 2018/19

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*)

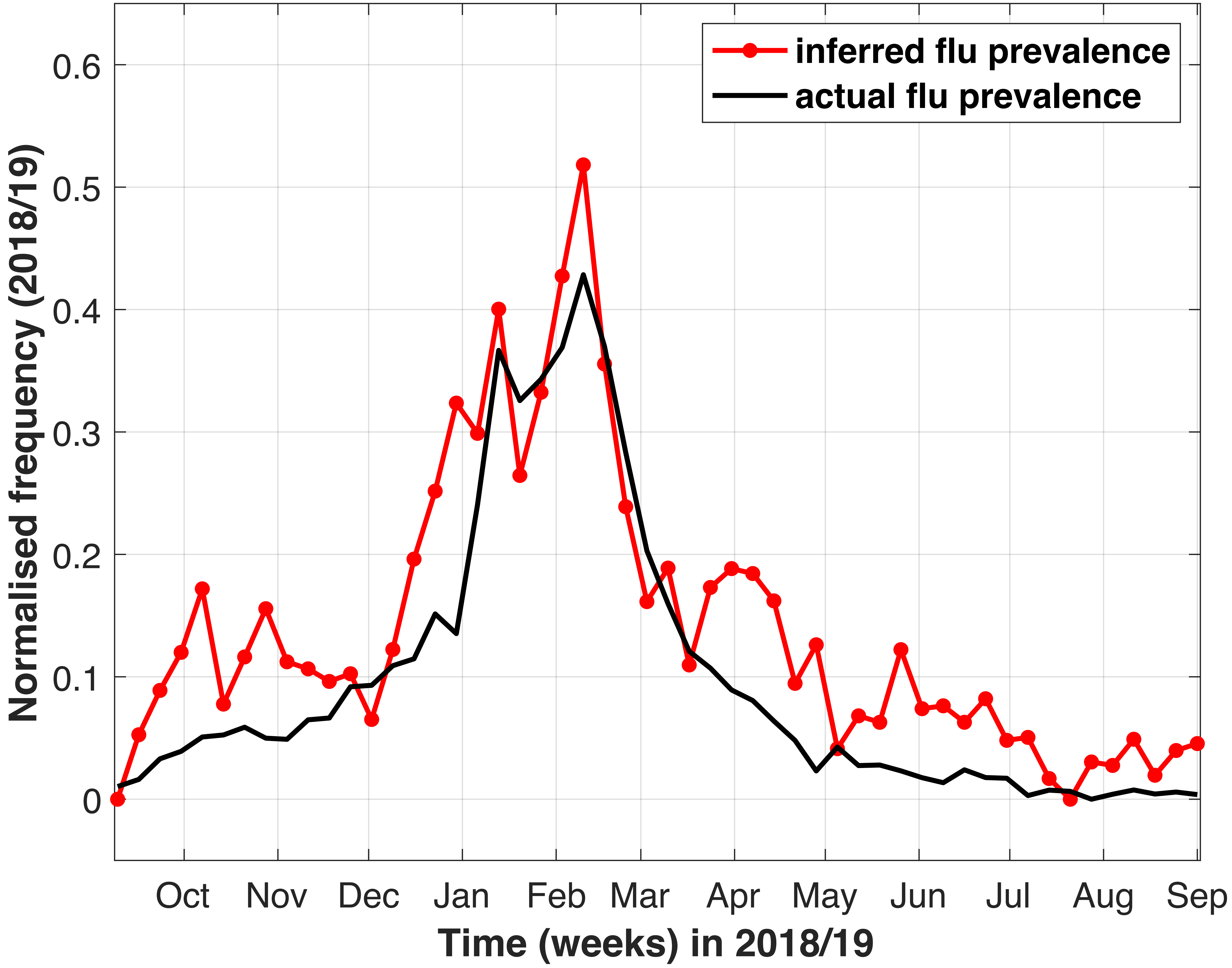


Supervised learning — OLS model training & testing



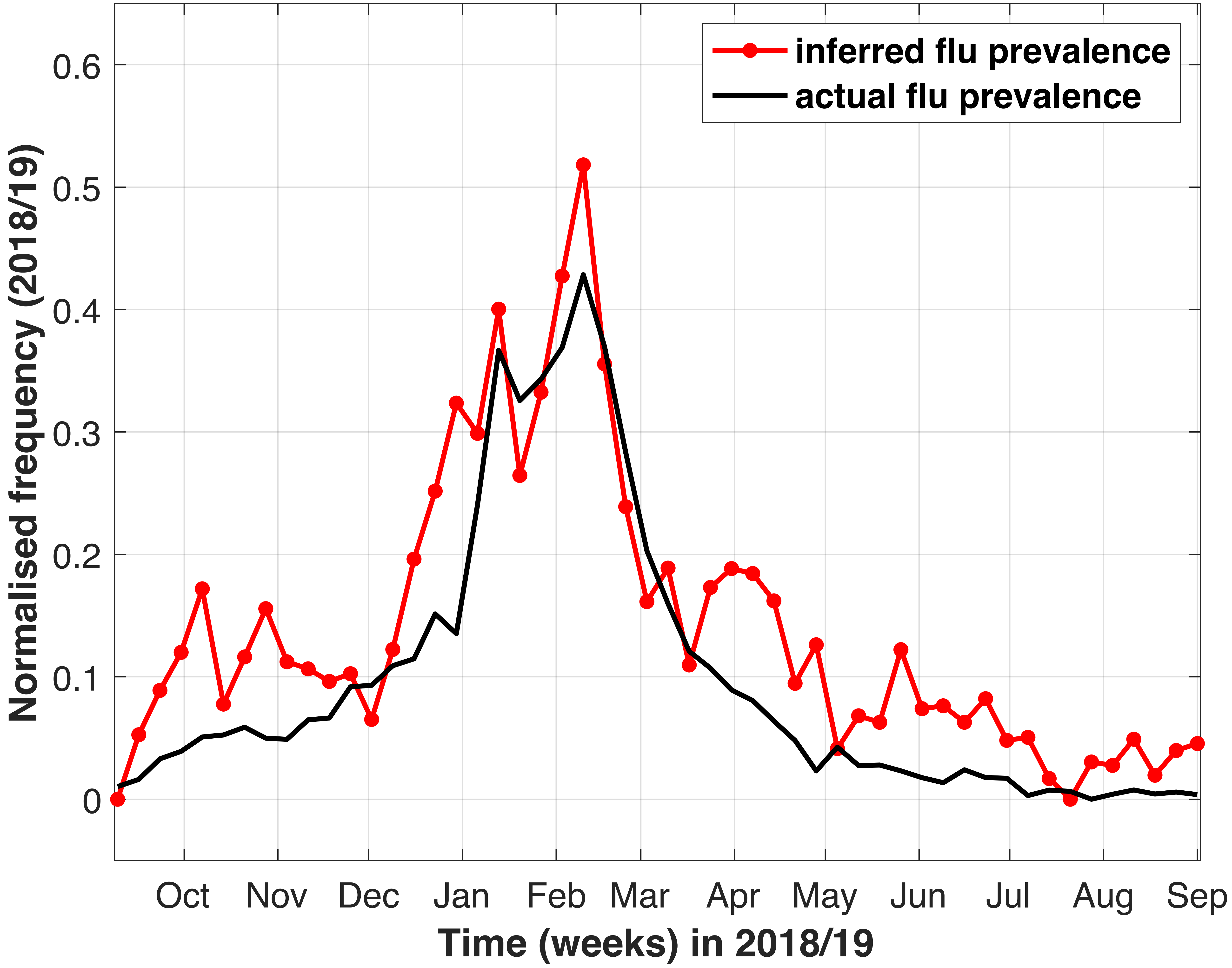
Supervised learning — OLS model training & testing

- 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*)



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 $\mathcal{J}$ 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):

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

- Basic **steps** of gradient descent

  — define a loss function, $\mathcal{J}$

  — compute the partial derivatives of $\mathcal{J}$ 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

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*

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

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:** $\mathscr{J}\left(\alpha, \beta\right) = \dfrac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2$

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:** $\mathcal{J}\left(\alpha, \beta\right) = \dfrac{1}{2n} \displaystyle\sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2$
- **Goal:** $\min\limits_{\alpha,\beta} \mathcal{J}\left(\alpha, \beta\right)$

Supervised learning — OLS with gradient descent

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

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

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:

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

  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

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

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

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

Supervised learning — OLS with gradient descent, the derivatives

Loss function: $$\mathcal{J}(\alpha, \beta) = \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2$$ $$= \frac{1}{2n} \sum_{i=1}^{n} \left(\alpha x_i + \beta - y_i\right)^2$$

n samples, $2n$ is a convention, $\mathcal{J} = \text{MSE}/2$

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

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

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?

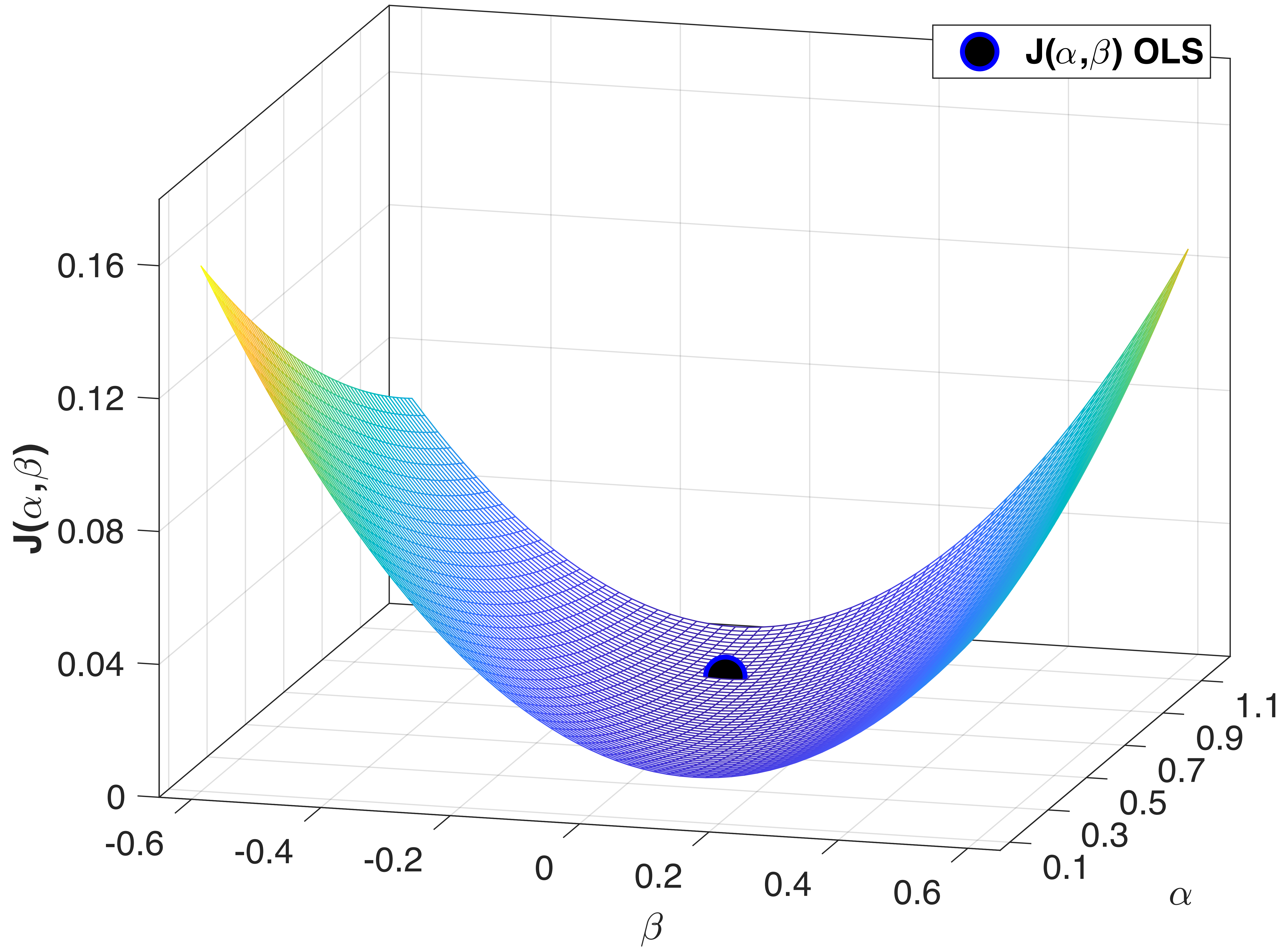
$$= \frac{1}{2n} \sum_{i=1}^{n} \left(w_1 x_{i,1} + \ldots + w_m x_{i,m} + \beta - y_i\right)^2$$

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

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

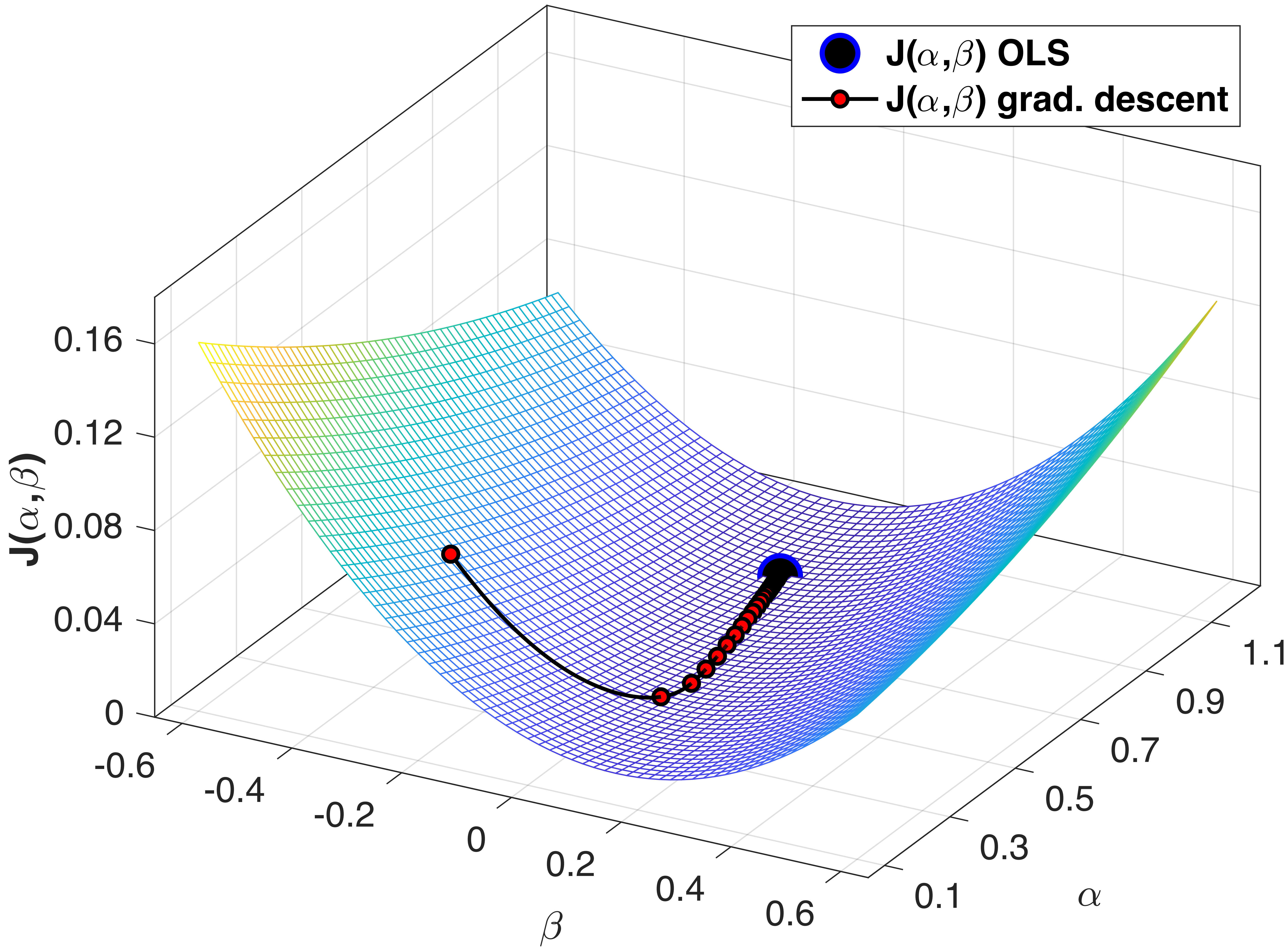
Supervised learning — OLS with gradient descent

- 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*)



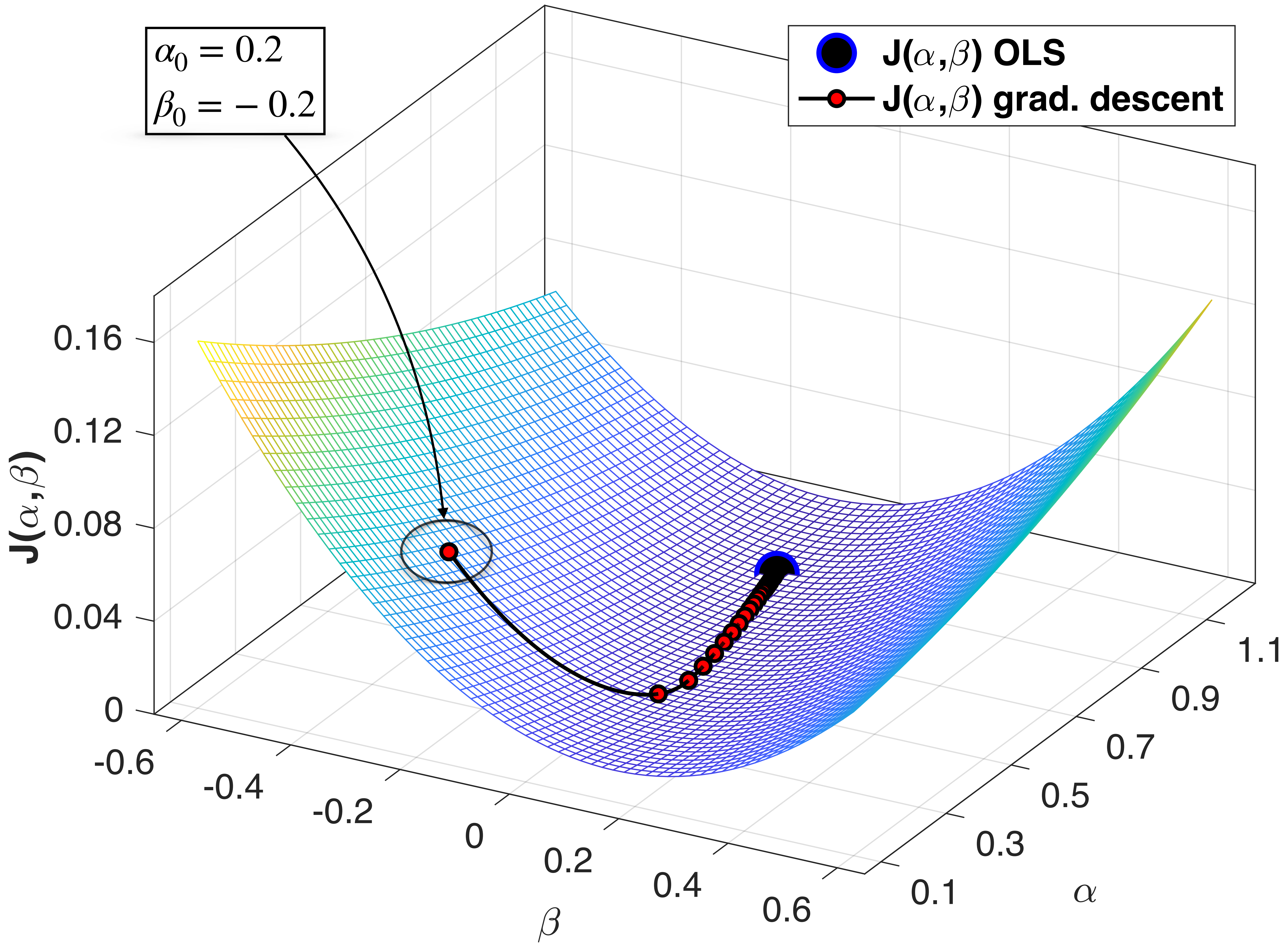
Supervised learning — OLS with gradient descent

- 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 $\mathcal{J}(\alpha, \beta)$ changed in the past k iterations?
- Gradient descent's solution almost identical to exact OLS solution (**expected?**)



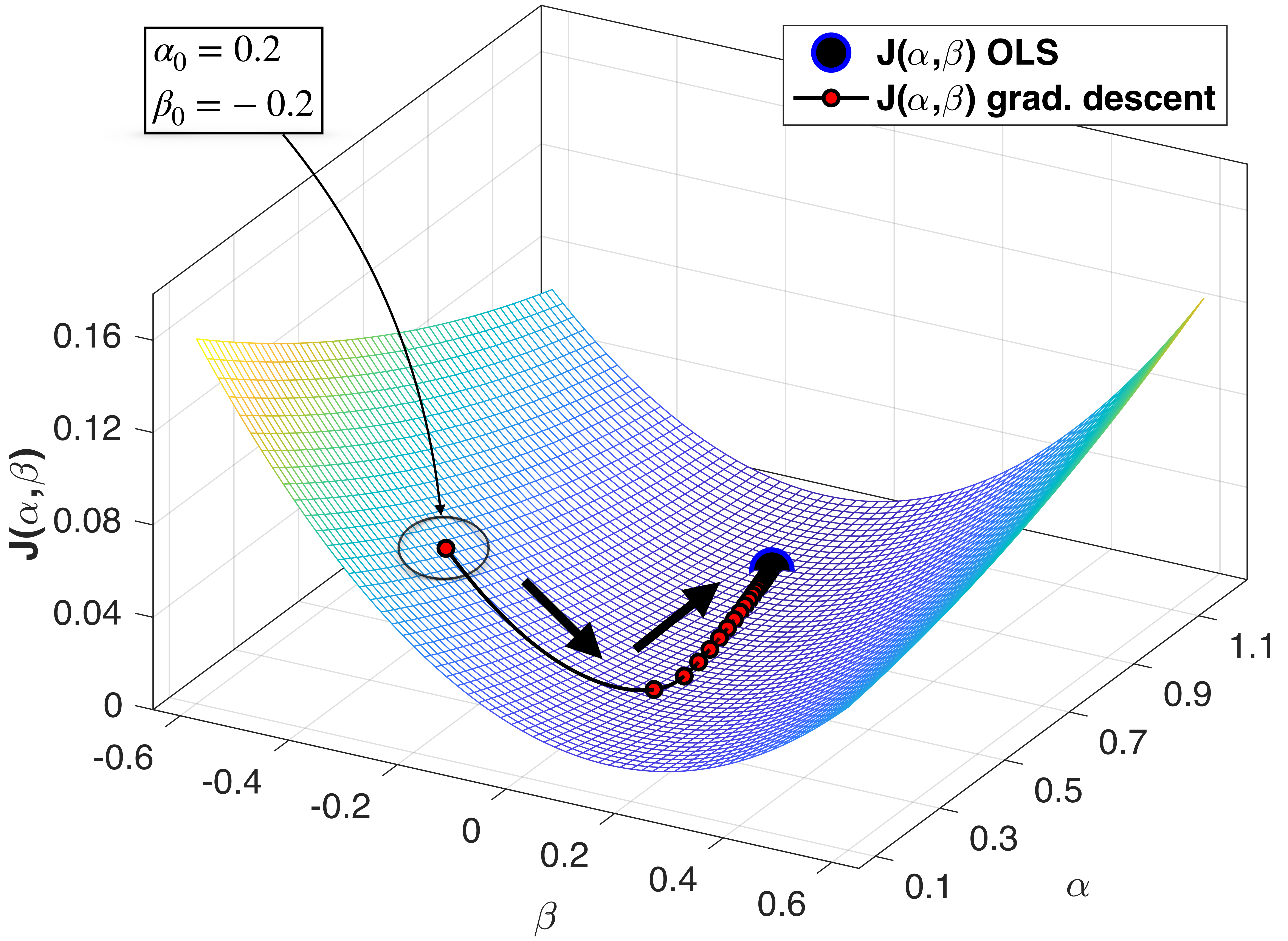
Supervised learning — OLS with gradient descent

- Let's 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?**)

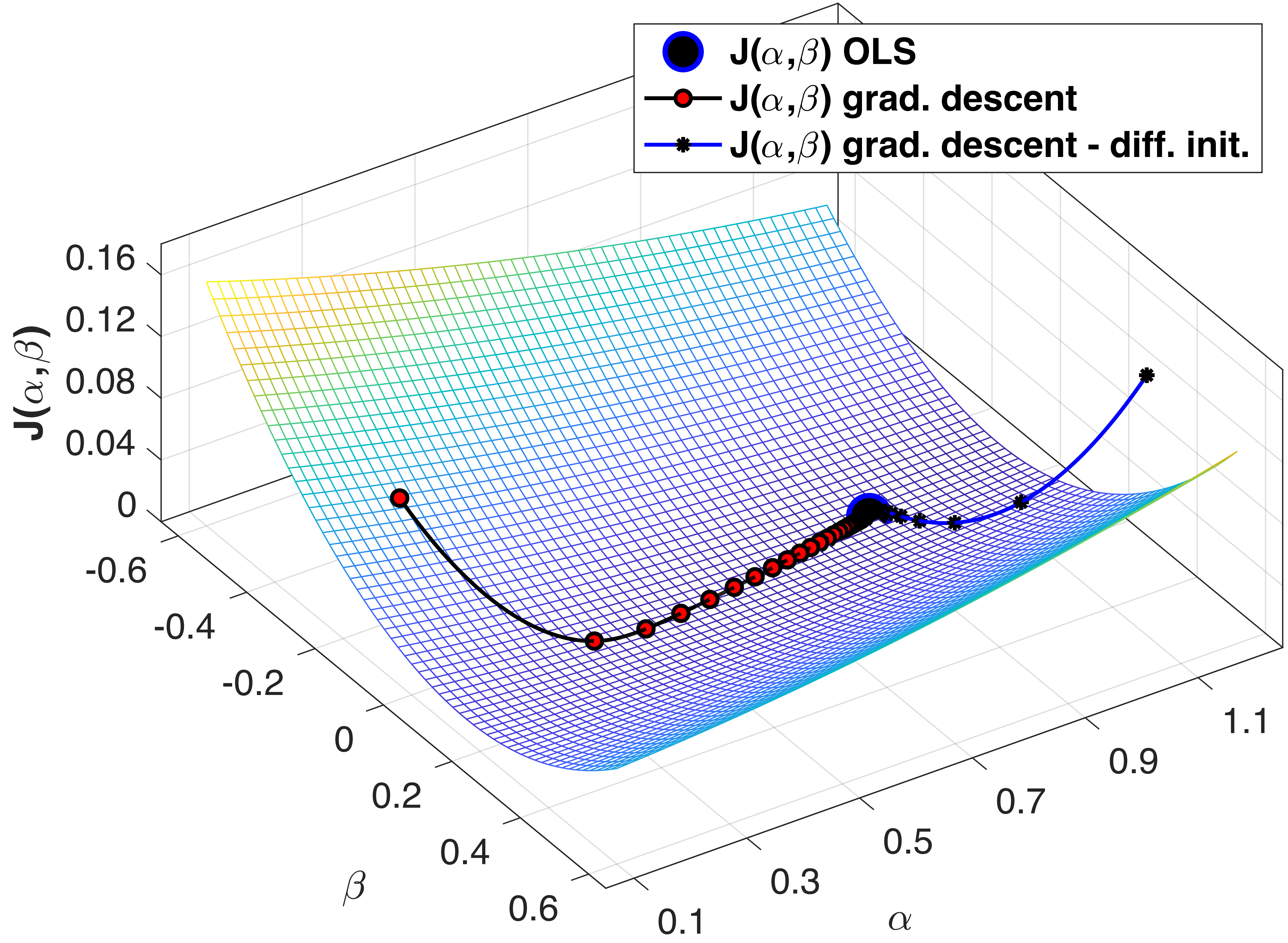


Supervised learning — OLS with gradient descent

- Let's 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 $\mathcal{J}(\alpha, \beta)$ changed in the past k iterations?
- Gradient descent's solution almost identical to exact OLS solution (**expected?**)

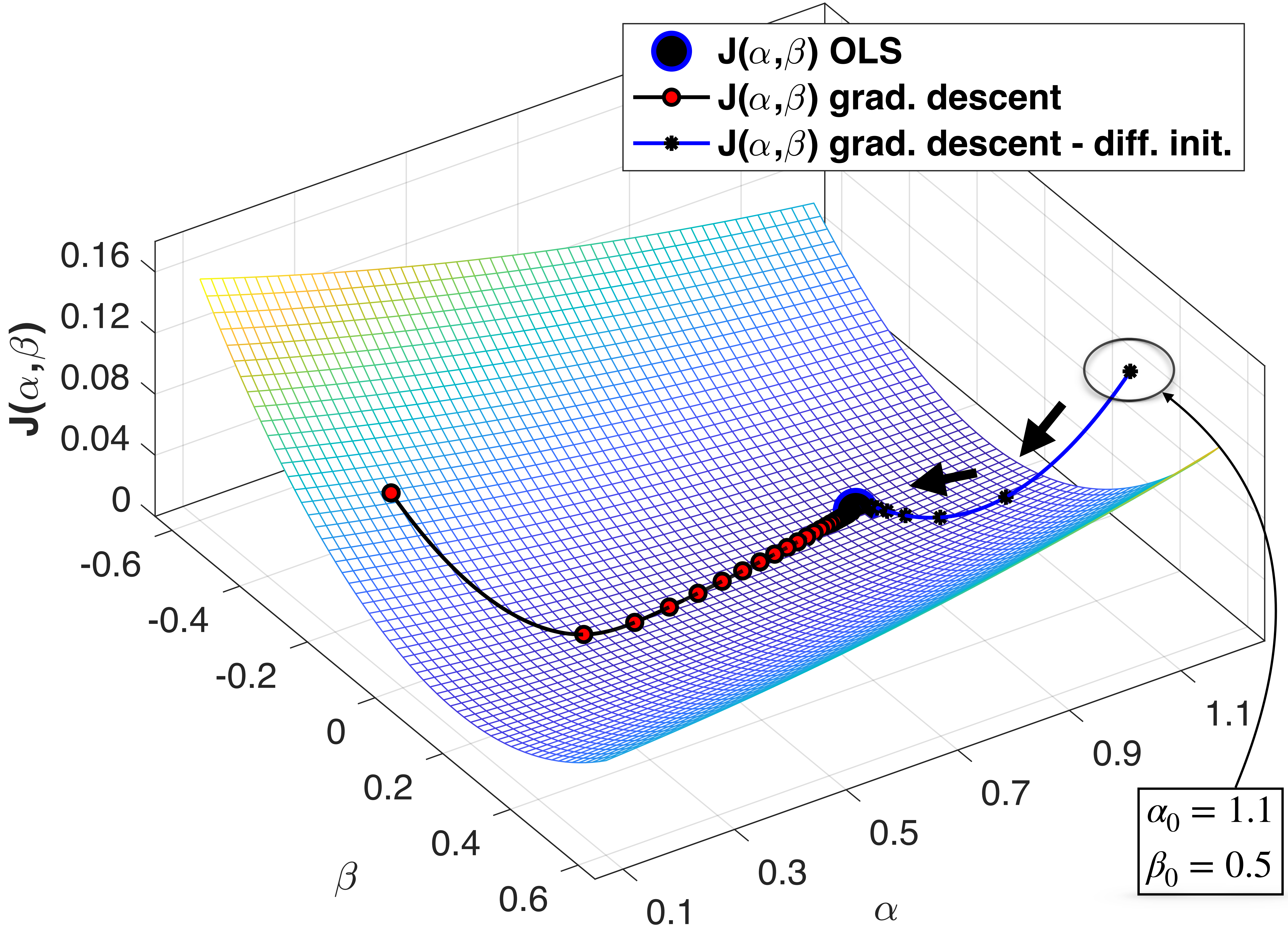


- Let's change the starting point
- $\alpha_0 = 1.1, \beta_0 = 0.5$
- $\ell = 0.02$ (*same learning rate*)
- In this case, it does not affect our solution (**why?**)



Supervised learning — 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)?