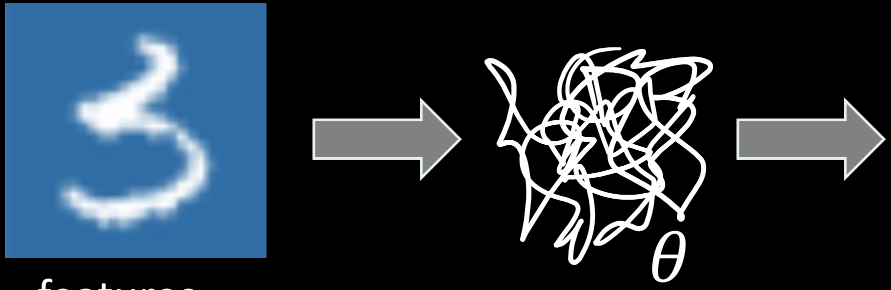
ground truth:
Let y be the actual observed label in the dataset

CONVENTIONAL (ALGORITHMIC) VIEW OF ML

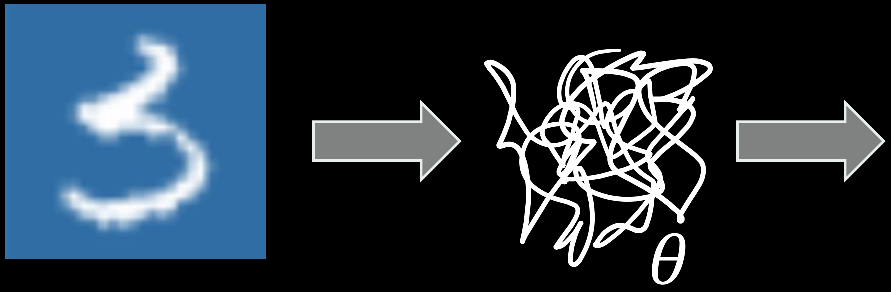


features

5
predicted label $\hat{y}$

evaluation metric:
loss function e.g. $L(y, \hat{y}) = 1_{\hat{y} \neq y}$

PROBABILITY MODELLER'S VIEW



$$\begin{cases} 5 \text{ with prob } 45\% \\ 3 \text{ with prob } 41\% \\ 6 \text{ with prob } \ \ 5\% \\ \quad \dots \end{cases}$$

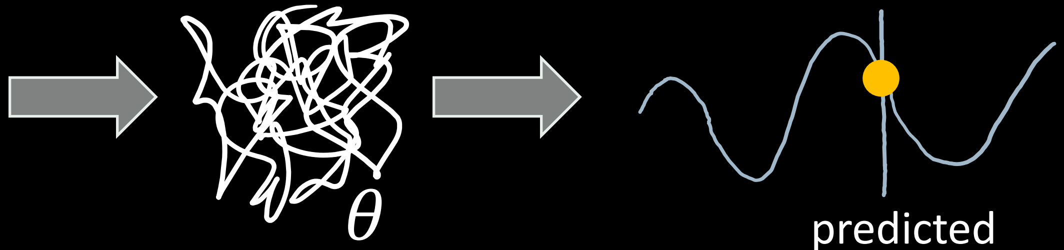
random predicted label Y

evaluation metric:
log likelihood i.e. $\log \Pr_Y(y)$

ground truth:
Let y be the actual observed temperature at time t

CONVENTIONAL (ALGORITHMIC) VIEW OF MODELLING

timepoint t

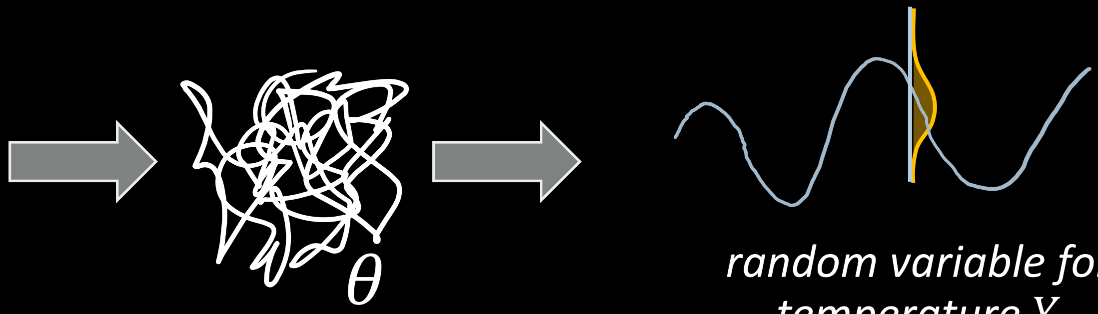


predicted temperature $\hat{y}$ at timepoint t

evaluation metric:
loss function e.g. $L(y, \hat{y}) = (y - \hat{y})^2$

PROBABILITY MODELLER'S VIEW

timepoint t

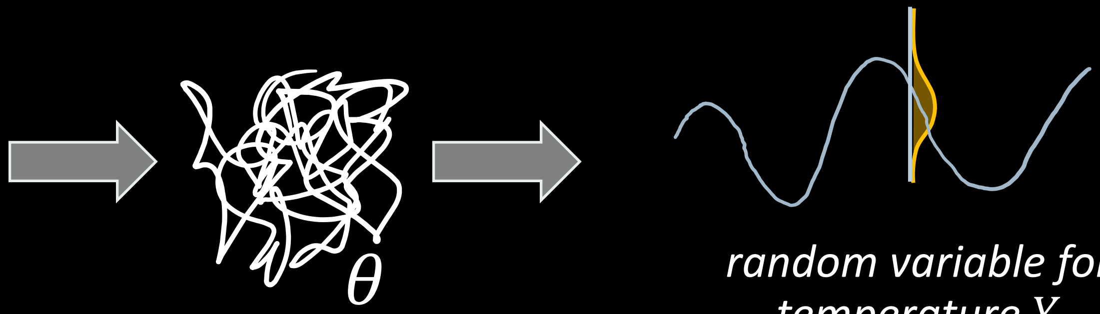


random variable for temperature Y at timepoint t

evaluation metric:
log likelihood i.e. $\log \Pr_Y(y; t)$

Our job is to invent a probability model, specifying the **distribution** of temperature at a given timepoint.

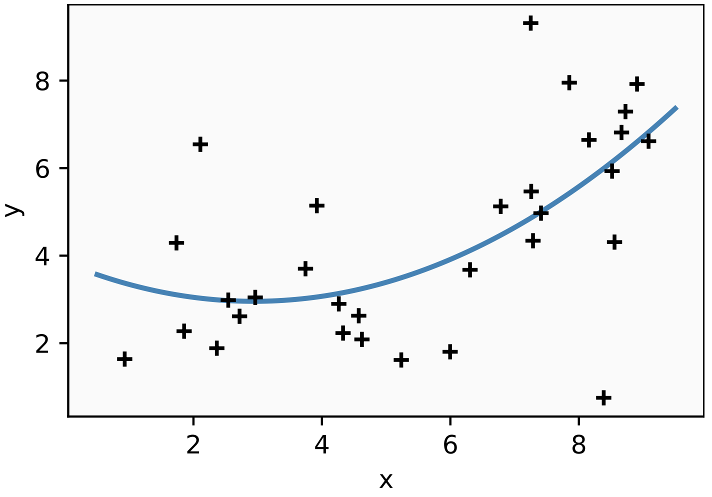
timepoint t



random variable for temperature Y at timepoint t

Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, fit the model $Y_i \sim \alpha + \beta x_i + \gamma x_i^2 + N(0, \sigma^2)$



Model for a single observation:

$$Y \sim \alpha + \beta x + \gamma x^2 + N(0, \sigma^2)$$
$$\sim N(\alpha + \beta x + \gamma x^2, \sigma^2)$$

Likelihood of a single observation:

$$\Pr_Y(y\,; x, \alpha, \beta, \gamma, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-(\alpha+\beta x+\gamma x^2))/2\sigma^2}$$

Log likelihood of the dataset:

$$\log \Pr(y_1, \dots y_n; x_1, \dots, x_n, \alpha, \beta, \gamma, \sigma) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

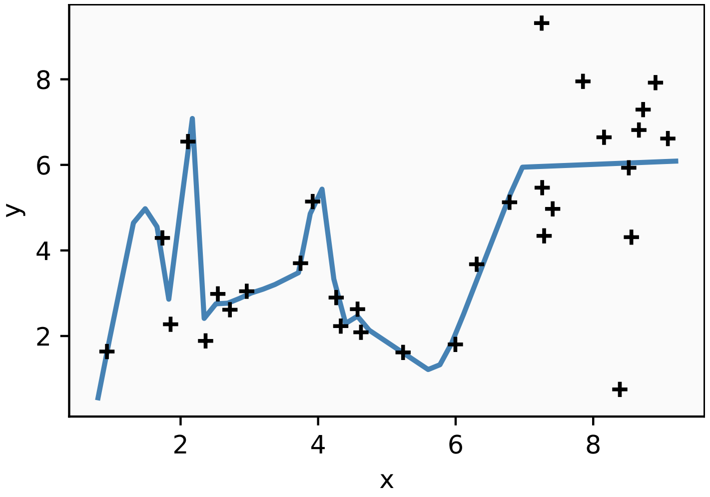
where $\hat{y}_i = \alpha + \beta x_i + \gamma x_i^2$

Optimize over the unknown parameters:

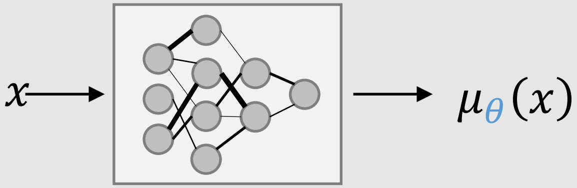
```
def logpr(y, x, α,β,γ,σ):
    pred = α + β*x + γ*(x**2)
    return - 0.5*np.log(2*π*σ²) – (y – pred)**2 / (2*σ²)

x,y = …
def f(θ):
    return - np.sum(logpr(y, x, θ[0], θ[1], θ[2], θ[3]))

scipy.optimize.fmin(f, [3,1,0.1,3])
```

Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, fit the model $Y_i \sim \mu_\theta(x_i) + N(0, \sigma^2)$



The question implies that $\mu_\theta(\cdot)$ is some given function with unknown parameter θ.



edge weights θ

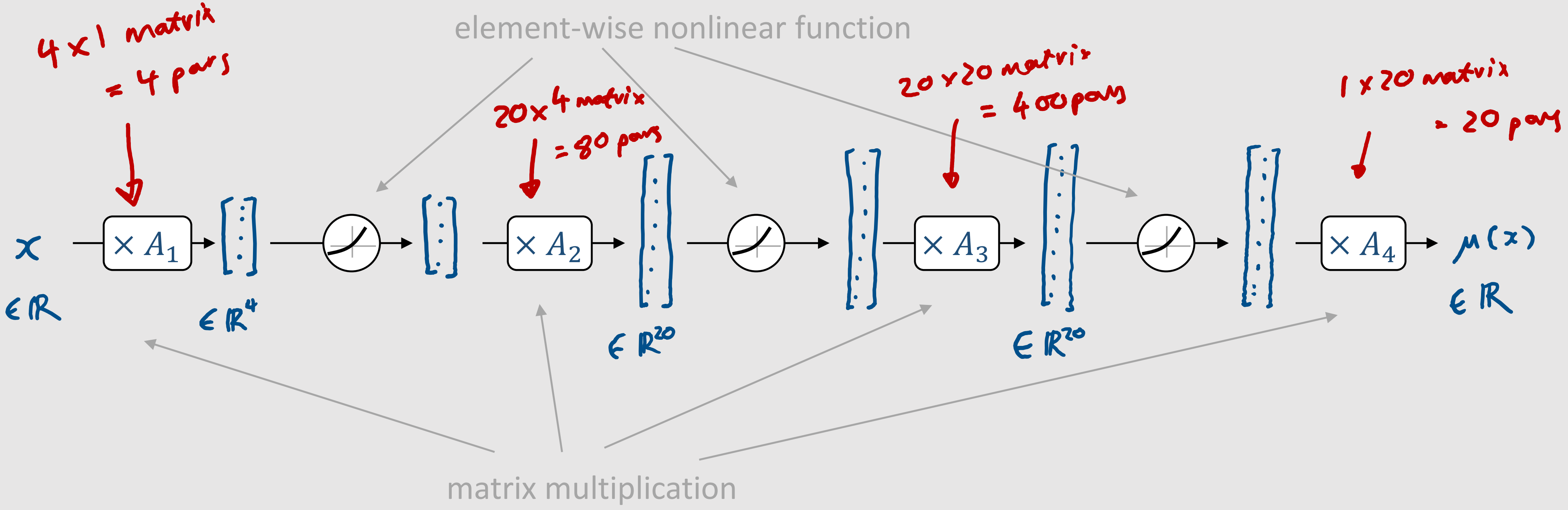
Log likelihood of the dataset:

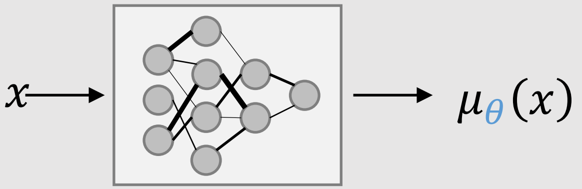
$$\log \Pr(y_1, \dots y_n; x_1, \dots, x_n, \theta, \sigma) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_\theta(x_i))^2$$

Optimize over the unknown parameters θ and σ:

```
class RWiggle(nn.Module):
    def __init__(self):
        super().__init__()
        self.μ = ...
        self.σ = nn.Parameter(torch.tensor(1.0))
    # compute log Pr(y;x)
    def forward(self, y, x):
        return - 0.5*torch.log(2*π*σ²) - torch.pow(y - self.μ(x), 2) / (2*σ²)

x,y = ...
mymodel = RWiggle()

optimizer = optim.Adam(mymodel.parameters())
for epoch in range(10000):
    optimizer.zero_grad()
    loglik = torch.sum(mymodel(y, x))
    (-loglik).backward()
    optimizer.step()
```

See notes section 3.3 for intro to PyTorch





edge weights θ

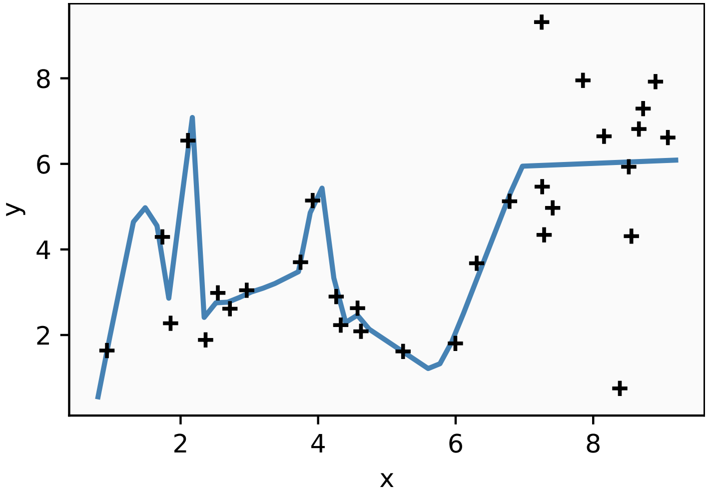
```
self.μ = nn.Sequential(
    nn.Linear(1,4), nn.LeakyReLU(),
    nn.Linear(4,20), nn.LeakyReLU(),
    nn.Linear(20,20), nn.LeakyReLU(),
    nn.Linear(20,1)  )
```

} 504 parameters

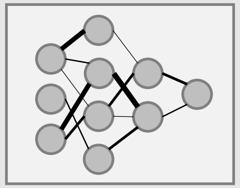
https://xkcd.com/1838

Example (regression)

Given a labelled dataset consisting of pairs (x_i, y_i) of real numbers, fit the model $Y_i \sim \mu_\theta(x_i) + N(0, \sigma^2)$



The question implies that $\mu_\theta(\cdot)$ is some given function with unknown parameter θ.

$x \longrightarrow$  $\longrightarrow \mu_\theta(x)$

edge weights θ

Log likelihood of the dataset:

$$\log \Pr(y_1, \dots y_n; x_1, \dots, x_n, \theta, \sigma) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_\theta(x_i))^2$$

Optimize over the unknown parameters θ and σ:

$$\max_{\theta,\sigma}\left\{-\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_\theta(x_i))^2\right\}$$

$$= \max_{\sigma}\left\{\max_{\theta}\left[-\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_\theta(x_i))^2\right]\right\}$$

$$= \max_{\sigma}\left\{-\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\min_{\theta}\left[\sum_{i=1}^{n}(y_i - \mu_\theta(x_i))^2\right]\right\}$$

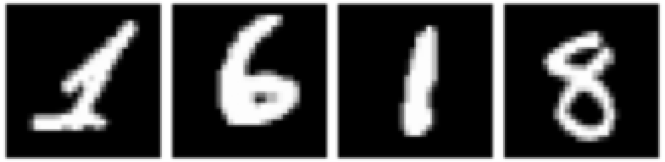
"First, find θ to minimize prediction loss. Next, pick σ."

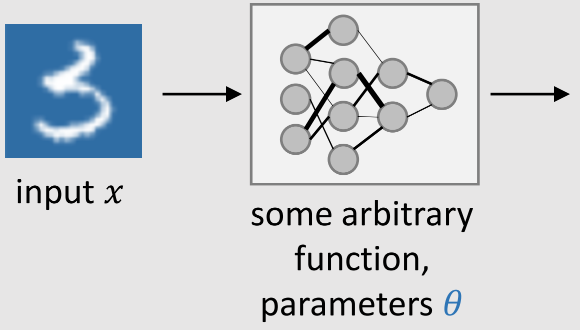
- Any useful “minimize prediction loss” problem can be restated as a maximum likelihood problem
- **Supervised ML *is* fitting a probability model**
- Probability modelling is a more powerful way to think about ML

Example (classification)

The MNIST dataset consists of pairs (x_i, y_i), where each record consists of $x_i \in \mathbb{R}^{28\times 28}$ an image of a handwritten digit and $y_i \in \{0,1,\ldots,9\}$ is its label.

Devise a probabilistic model to predict the label of a given input image, and fit it.



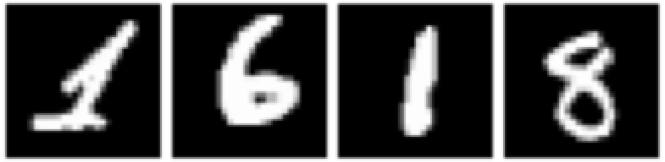


What sort of probability model might we use for the response Y?

Example (classification)

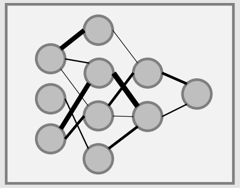
The MNIST dataset consists of pairs (x_i, y_i), where each record consists of $x_i \in \mathbb{R}^{28\times 28}$ an image of a handwritten digit and $y_i \in \{0,1,\dots,9\}$ is its label.

Devise a probabilistic model to predict the label of a given input image, and fit it.



input x

→



some arbitrary function, parameters θ

→ $\vec{p}(x) = [p_0, \dots, p_9]$ → $Y \sim \text{Cat}(\vec{p}(x))$

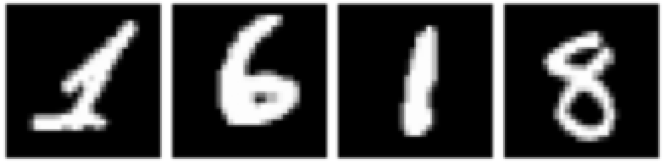
How can we make sure that $\vec{p}$ is a valid probability vector?

(We need $p_i \in [0,1]$ for each i, and $\Sigma_i p_i = 1$.)

Example (classification)

The MNIST dataset consists of pairs (x_i, y_i), where each record consists of $x_i \in \mathbb{R}^{28\times 28}$ an image of a handwritten digit and $y_i \in \{0,1,\ldots,9\}$ is its label.

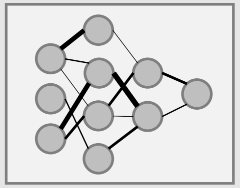
Devise a probabilistic model to predict the label of a given input image, and fit it.



Softmax function:

$$p_k = \frac{e^{s_k}}{\sum_{\ell=0}^{9} e^{s_\ell}}$$

input x



some arbitrary function, parameters θ

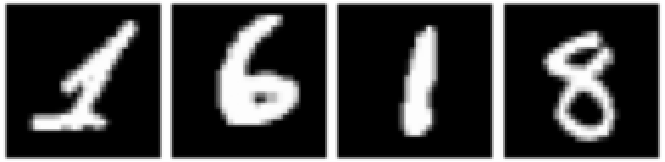
$\vec{s} \in \mathbb{R}^{10} \longrightarrow \vec{p} = \mathrm{softmax}(\vec{s}) \longrightarrow Y \sim \mathrm{Cat}(\vec{p})$

How should we fit the function parameters θ?

Example (classification)

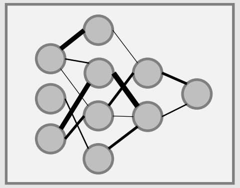
The MNIST dataset consists of pairs (x_i, y_i), where each record consists of $x_i \in \mathbb{R}^{28\times 28}$ an image of a handwritten digit and $y_i \in \{0,1,\dots,9\}$ is its label.

Devise a probabilistic model to predict the label of a given input image, and fit it.



Model for a single datapoint:

input x



some arbitrary function, parameters θ

$$\vec{s} \in \mathbb{R}^{10} \longrightarrow \vec{p} = \text{softmax}(\vec{s}) \longrightarrow Y \sim \text{Cat}(\vec{p})$$

$$1_a = \begin{cases} 1 & \text{if } a \text{ is true} \\ 0 & \text{if } a \text{ is false} \end{cases}$$

Likelihood of a single datapoint y:

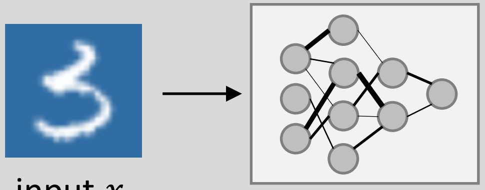
$$\Pr_Y(y; x, \theta) = [\vec{p}(x;\theta)]_y = p_y(x;\theta)$$

where $\vec{p}(x;\theta) = \text{softmax}(\vec{s}(x;\theta\,))$

Log likelihood of the dataset:

$$\log \Pr(y_1,\dots,y_n) = \sum_{i=1}^{n} \log p_{y_i}(x_i;\theta) = \sum_{i=1}^{n}\sum_{k=0}^{9} 1_{y_i=k} \log p_k(x_i;\theta)$$

This is called *softmax cross-entropy*, and it's the standard loss function for classification.



input x

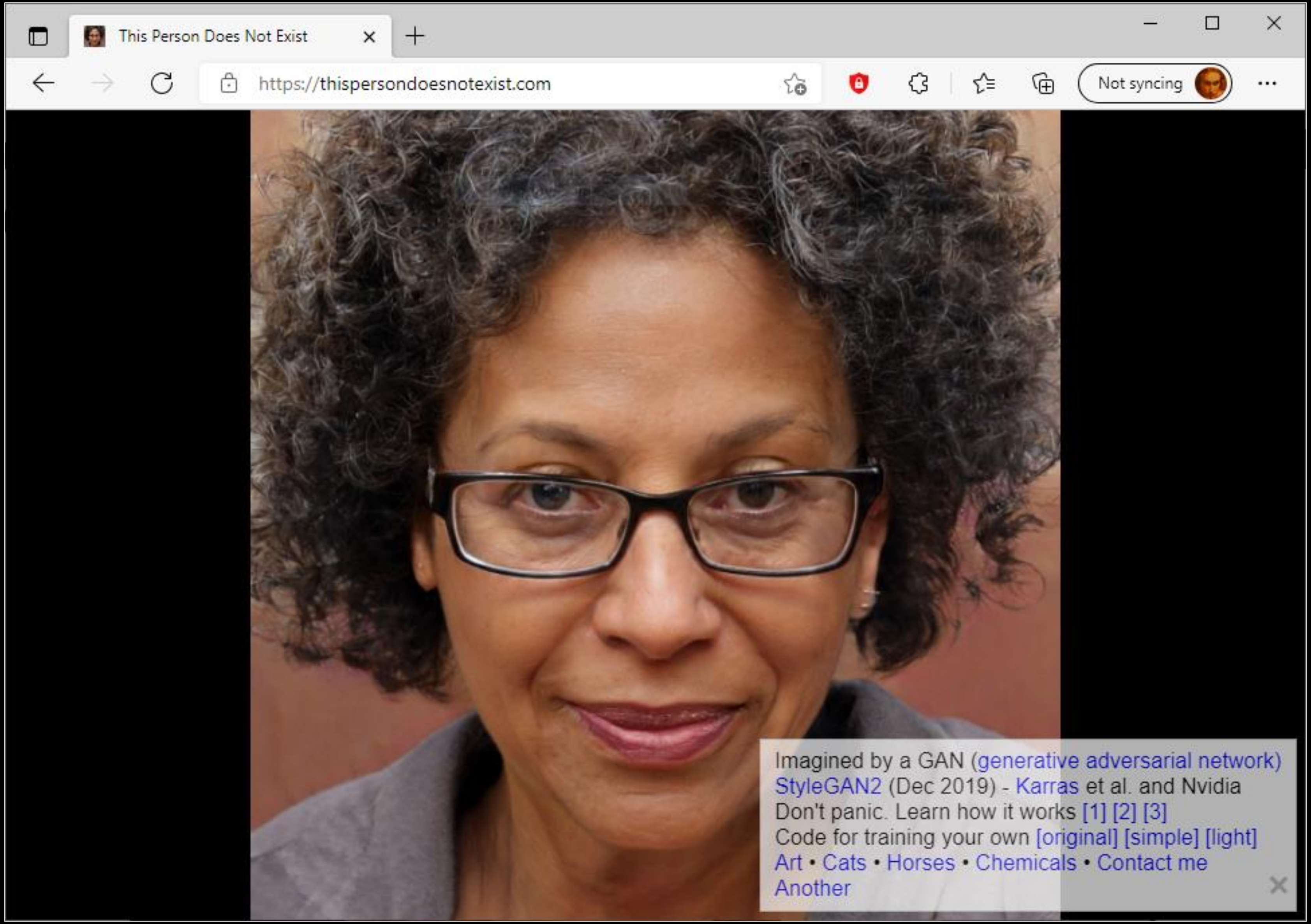
some arbitrary function, parameters θ

$$\vec{s} \in \mathbb{R}^{10} \longrightarrow \vec{p} = \text{softmax}(\vec{s}) \longrightarrow Y \sim \text{Cat}(\vec{p})$$

```python
class MyImageClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.f = nn.Sequential(          # input shape [B*1*28*28]
            nn.Conv2d(1, 32, 3, 1),      # -> [B*32*26*26]
            nn.ReLU(),
            nn.Conv2d(32, 64, 3, 1),     # -> [B*64*24*24]
            nn.MaxPool2d(2),             # -> [B*64*12*12]
            nn.Dropout2d(0.25),
            nn.Flatten(1),               # -> [B*9216]
            nn.Linear(9216, 128),        # -> [B*128]
            nn.ReLU(),
            nn.Dropout2d(0.5),
            nn.Linear(128, 10)           # -> [B*10]
        )

    # compute log likelihood for a batch of data
    def forward(self, y, x):             # x.shape [B*1*28*28], y.shape [B], output.shape [B]
        return - nn.functional.cross_entropy(self.f(x), y, reduction='none')

    # compute the class probabilities for a single image
    def classify(self, x):   # input: [1*28*28] array
        q = self.f(torch.as_tensor(x)[None,...])[0]
        return nn.functional.softmax(q, dim=0)
```

HOMEWORK

- Make sure you can run this code
- For each digit {0,1,...,9}, select some images where the label is very likely, and some where it is very unlikely



This is machine learning, too!

CONVENTIONAL VIEW OF MACHINE LEARNING

Supervised Learning

Data: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

Labels: $y_1, y_2, \ldots, y_n$

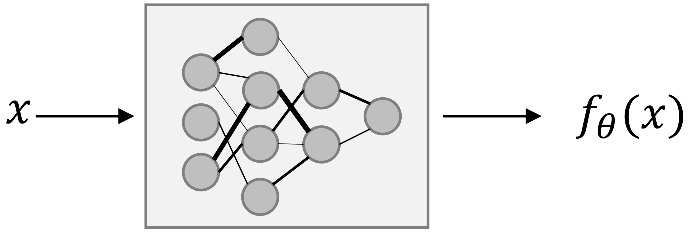
Task: Predict the label

$$y_i \approx f_\theta(x_i)$$

Training goal: Invent a loss function and learn θ to minimize the prediction loss

$$\sum_i L(y_i, f_\theta(x_i))$$

Evaluation: prediction loss on holdout data



edge weights θ

Generative Modelling

Data: $\{x_1, x_2, \ldots, x_n\}$

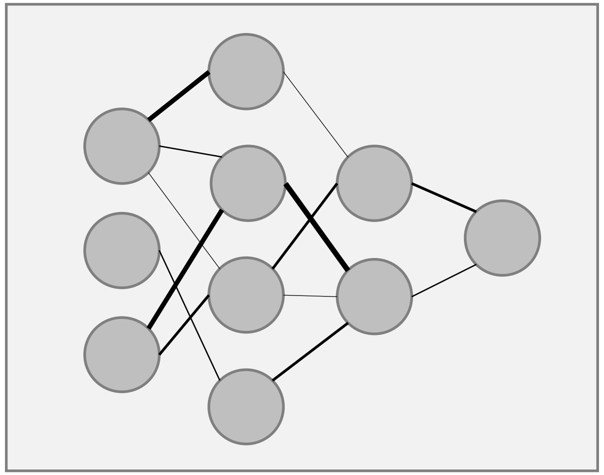
Labels: n/a

Task: learn to synthesize new values similar (but not identical) to those in the dataset, ...

Training goal: ???

Evaluation: ???

random noise Z →



edge weights θ

→ $X = f_\theta(Z)$

The output X is a random variable. It therefore has a likelihood function $\Pr_X(x)$.

QUESTION. How could we even use neural networks to generate novel images? What should the input be?

Supervised Learning

Data: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

Labels: $y_1, y_2, \ldots, y_n$

Task: Fit a probability model

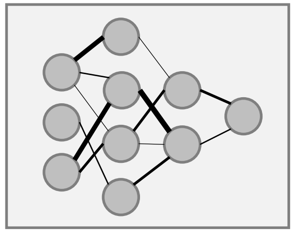
$$\Pr_Y(y_i \,;\, f_\theta(x_i))$$

Training goal: Learn θ to maximize the log likelihood of the dataset

$$\sum_i \log \Pr_Y(y_i \,;\, f_\theta(x_i))$$

Evaluation: log likelihood of holdout data

$x \longrightarrow$



edge weights θ

$\longrightarrow f_\theta(x) \longrightarrow Y$

Generative Modelling

Data: $\{x_1, x_2, \ldots, x_n\}$

Labels: n/a

Task: fit the probability model

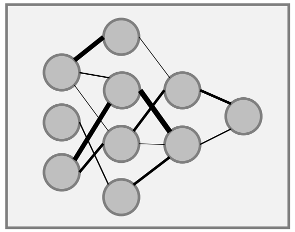
$$\Pr_X(x \,;\, \theta)$$

Training goal: Learn θ to maximize the log likelihood of the dataset

$$\sum_i \log \Pr_X(x_i \,;\, \theta)$$

Evaluation: log likelihood of holdout dataset

random noise Z $\longrightarrow$



edge weights θ

$\longrightarrow X = f_\theta(Z)$

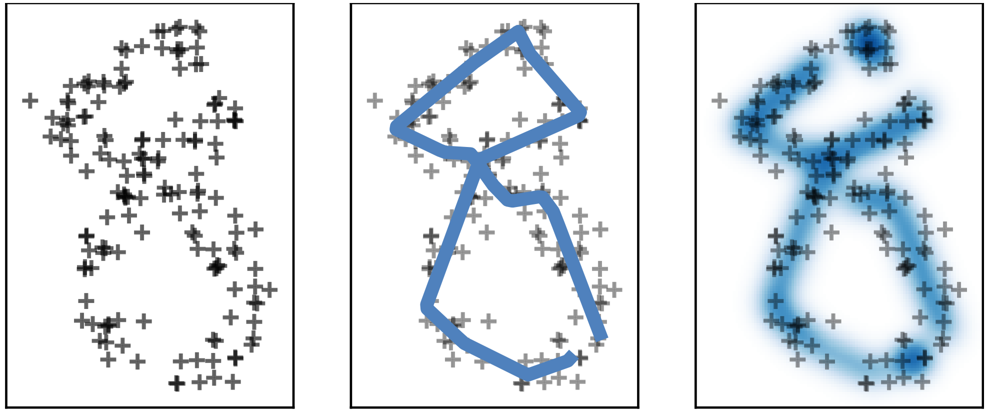
X has a histogram, (if I run it lots of times with different noise).

Exercise (generative modelling).

Train a generative model for a collection of points $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^2$. The model should have the form

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim f(Z) + \begin{bmatrix} N(0,\sigma^2) \\ N(0,\sigma^2) \end{bmatrix}$$

where $Z \sim U[0,1]$ and $f : [0,1] \to \mathbb{R}^2$ is a neural network to be trained.



the path $f(z)$

the likelihood $\Pr_{X_1,X_2}(x_1, x_2)$

Law of Total Probability:

$$\mathbb{P}(X=x) = \sum_z \mathbb{P}(X=x \mid Z=z)\, \mathbb{P}(Z=z)$$

Model for a single observation

$$Z \sim U[0,1]$$
$$X_1 \sim f_1(Z) + N(0,\sigma^2)$$
$$X_2 \sim f_2(Z) + N(0,\sigma^2)$$

Likelihood for a single observation

$$\Pr(x_1, x_2) = \int_{z=0}^{1} \underbrace{\Pr(x_1, x_2 \mid Z=z)}_{} \overbrace{\Pr_Z(z)}^{\Pr_Z(z)=1} \, dz$$

$$\underbrace{\Pr(x_1 \mid Z=z)}_{} \Pr(x_2 \mid Z=z)$$

$$\Pr(x_i \mid Z=z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(x_i - f_i(z))^2/2\sigma^2}$$

Log likelihood of the dataset

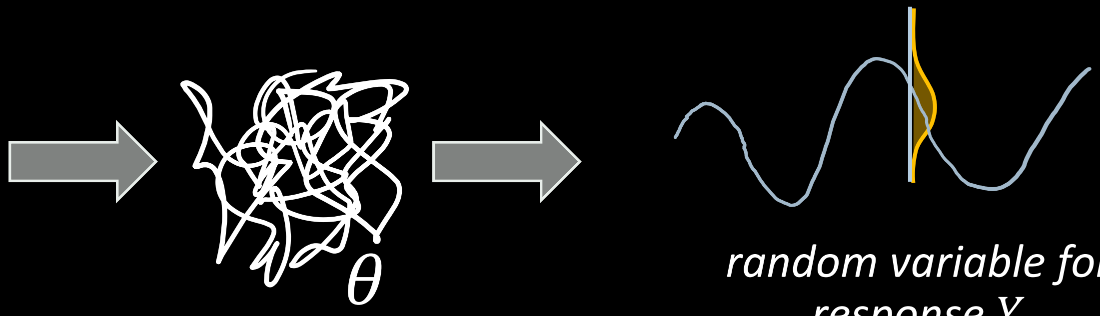
$$\sum_{i=1}^{n} \log \Pr(x_1^{(i)}, x_2^{(i)})$$

Maximize over unknown parameters

(We'll approximate the integral over z by a sum.)

Our job is to invent a probability model, specifying the **distribution** of the response at a given input.

input x
(e.g. timepoint)

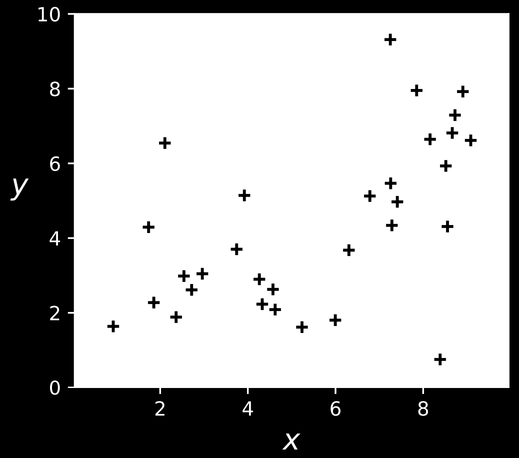


random variable for response Y at input x

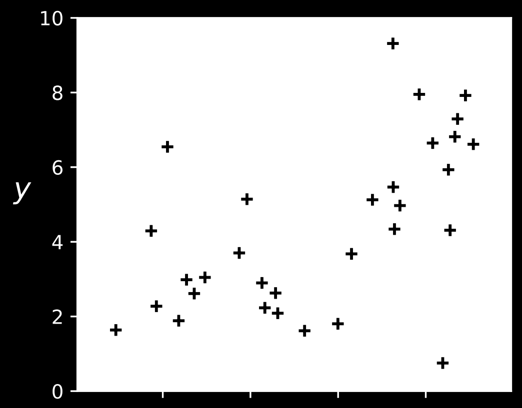
What are we really after,
when we fit a probability model?

What’s a good model?

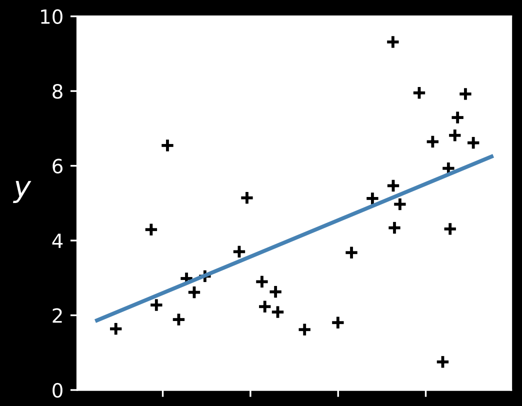
How can we compare models?



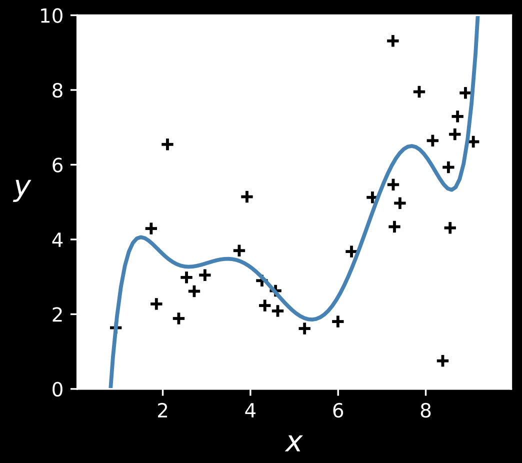
dataset of (x_i, y_i) pairs



dataset of (x_i, y_i) pairs



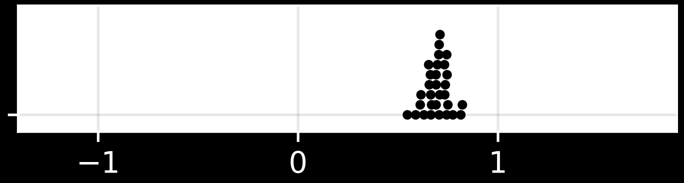
$$Y_i \sim 1.62 + 0.49\, x_i + \text{Normal}(0, 2.39^2)$$



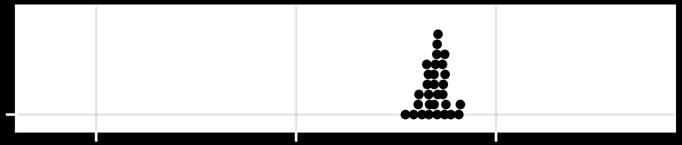
$$Y_i \sim -38.5 + 95.7\, x_i - 84.8\, x_i^2 + 38.3\, x_i^3 - 9.5\, x_i^4 + 1.3\, x_i^5 - 0.09\, x_i^6 + 0.003\, x_i^7 + \text{Normal}(0, 0.31^2)$$

Question

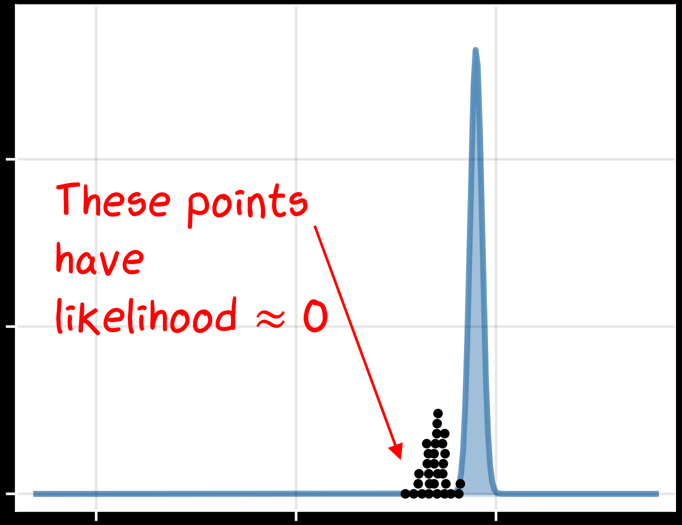
Which of these two models fits the dataset better?



dataset $\{x_1, \ldots, x_n\}$



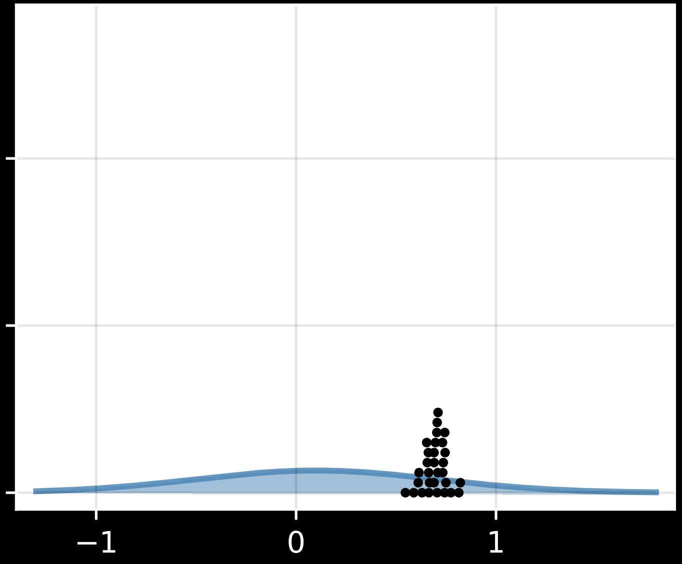
dataset $\{x_1, \dots, x_n\}$



Model A:
IID sample from
$X \sim N(0.9, 0.03^2)$

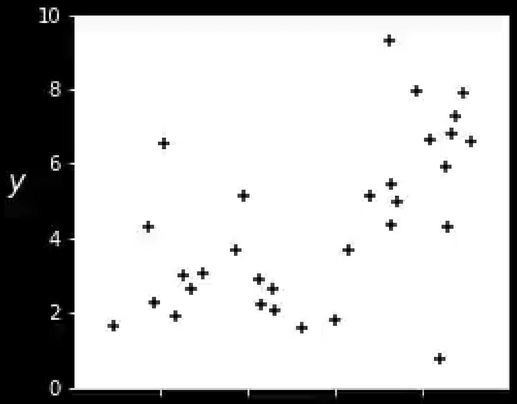
This model is extraordinarily unlikely to generate the dataset, so it's a bad model.
log lik (dataset) = -570.5

> Question
> Which of these two models fits the dataset better?

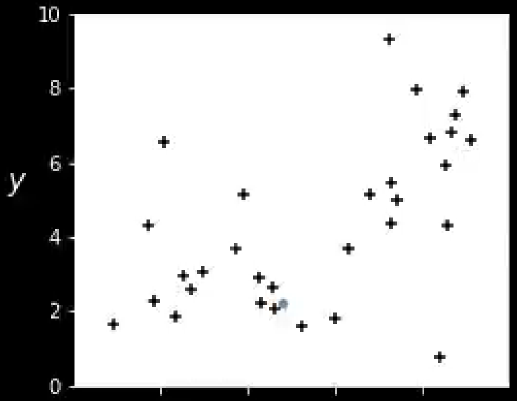


Model B:
IID sample from
$X \sim N(0.1, 0.6^2)$

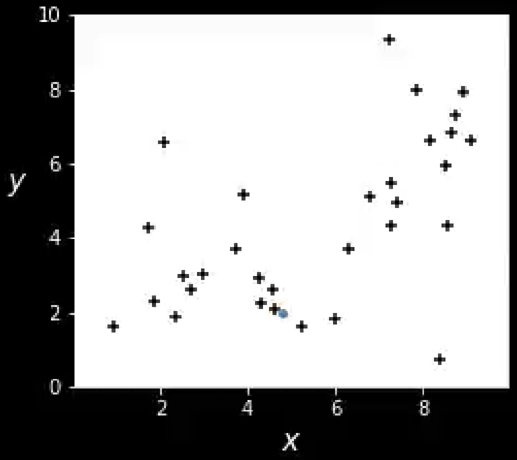
This model might possibly have generated the data (but it's still not great).
log lik (dataset) = -28.0



dataset of (x_i, y_i) pairs



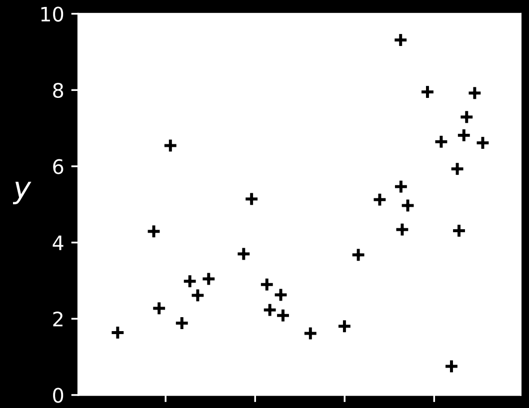
$$Y_i \sim 1.62 + 0.49\, x_i + \text{Normal}(0, 2.39^2)$$



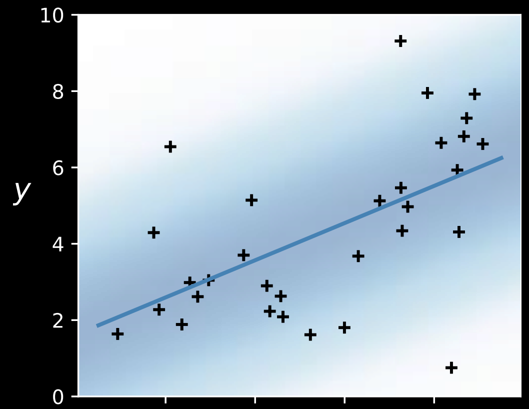
$$Y_i \sim -38.5 + 95.7\, x_i - 84.8\, x_i^2 + 38.3\, x_i^3 - 9.5\, x_i^4 + 1.3\, x_i^5 - 0.09\, x_i^6 + 0.003\, x_i^7 + \text{Normal}(0, 0.31^2)$$

Question

Which of these two models fits the dataset better?



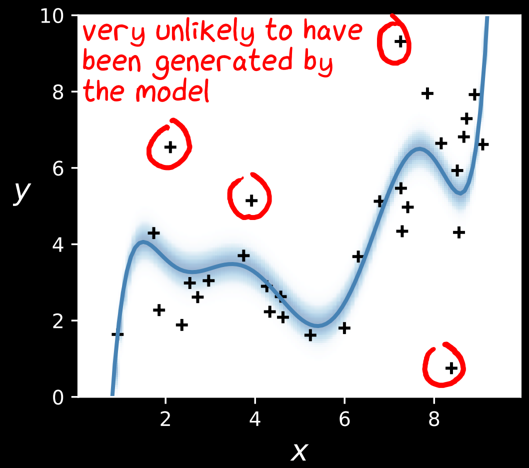
dataset of (x_i, y_i) pairs



$$Y_i \sim 1.62 + 0.49\, x_i + \text{Normal}(0, 2.39^2)$$

log lik (dataset) = -64.6

This is the better model.



$$Y_i \sim -38.5 + 95.7\, x_i - 84.8\, x_i^2 + 38.3\, x_i^3 - 9.5\, x_i^4 + 1.3\, x_i^5 - 0.09\, x_i^6 + 0.003\, x_i^7 + \text{Normal}(0, 0.31^2)$$

log lik (dataset) = -379.3

- The goal of modelling is to find models that fit the dataset well
- A good metric for model fit is: **likelihood of the dataset, according to the model**
- This applies equally to both supervised and generative modelling

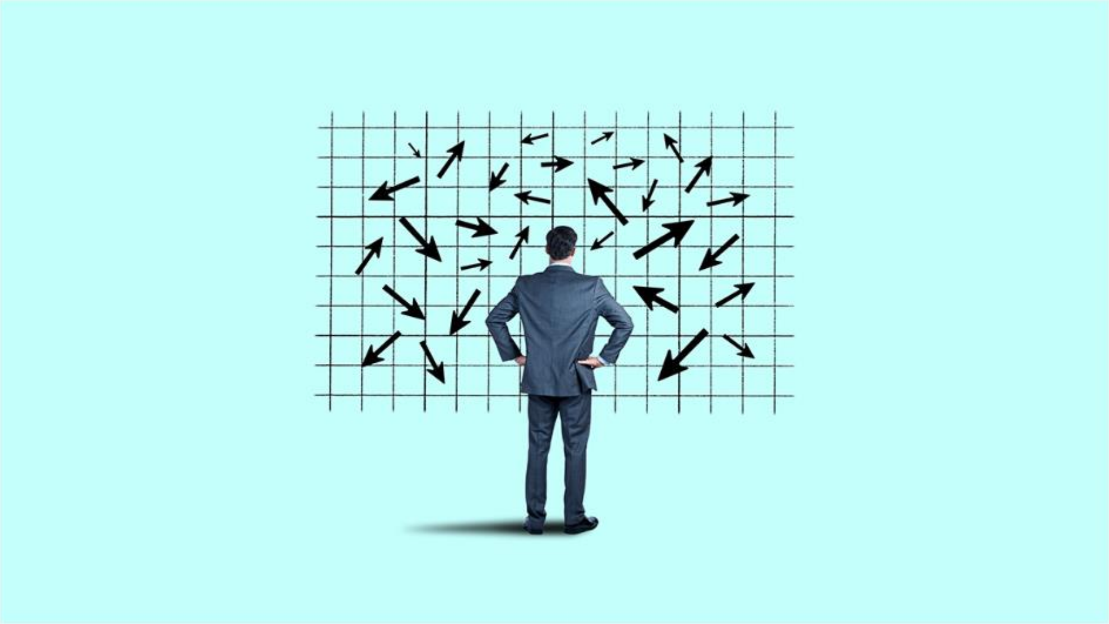
In NLP, log likelihood is called "perplexity"

In sports betting, log likelihood is called "ignorance score"

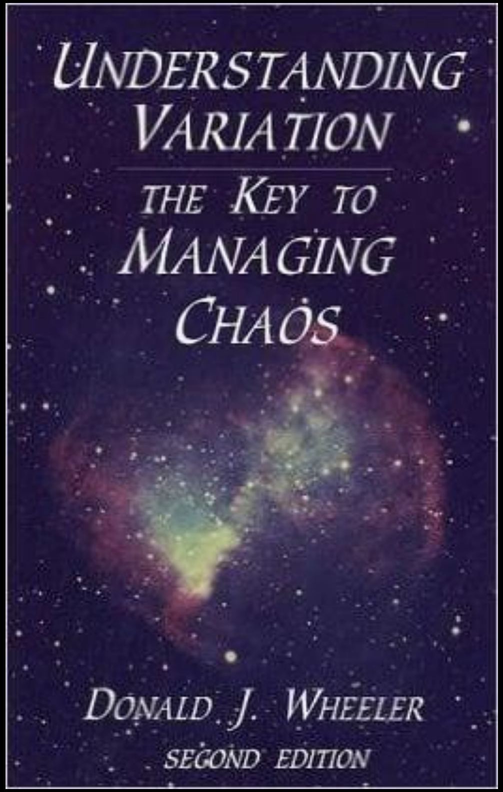
Do You Understand the Variance In Your Data?

by Thomas C. Redman

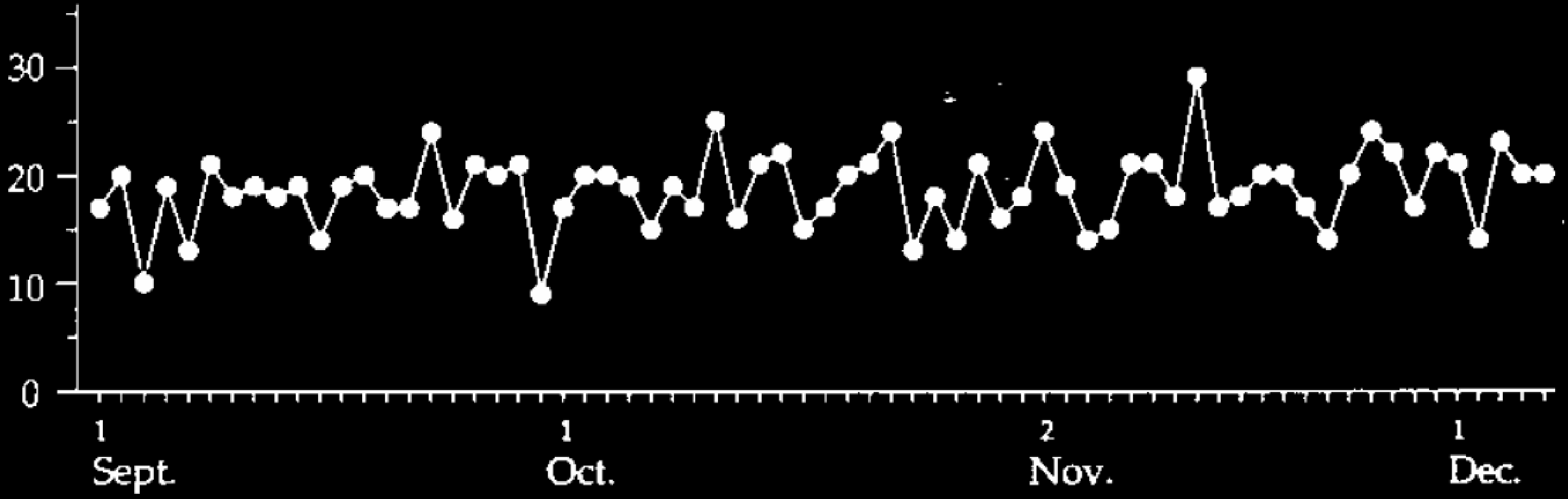
August 16, 2019



DNY59/Getty Images

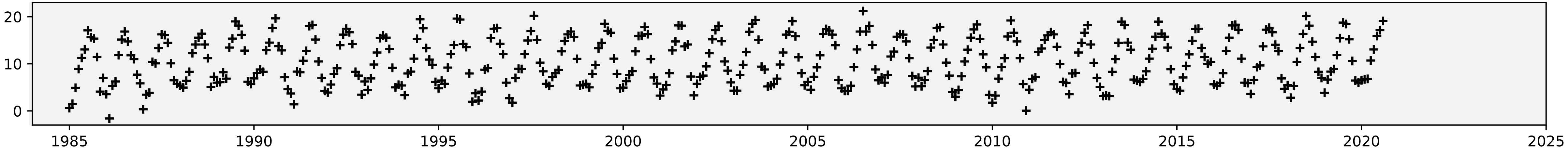


Number of defective pairs of shoes each day



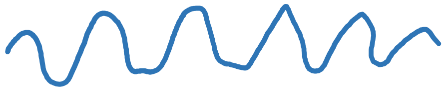
Monthly average temperatures in Cambridge, UK

What's a good model for this dataset?

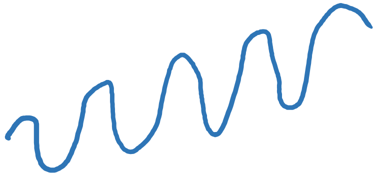


Climate is stable?

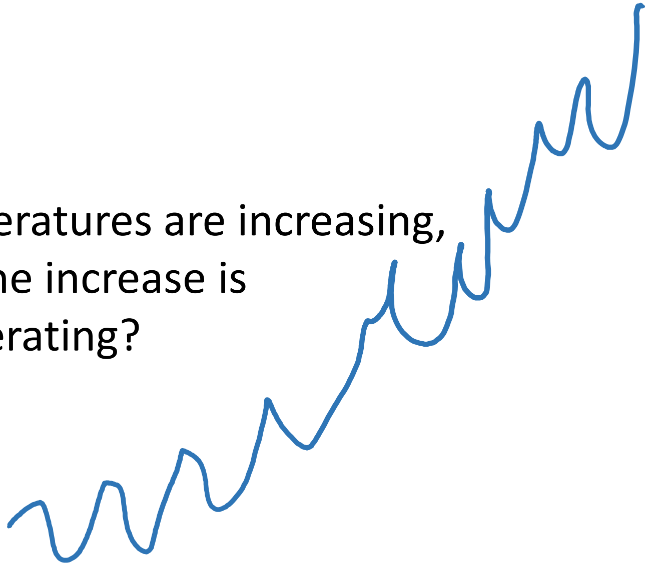
$$\text{Temp}(t) \sim a + b\sin\big(2\pi(t+\phi)\big) + N(0,\sigma^2)$$



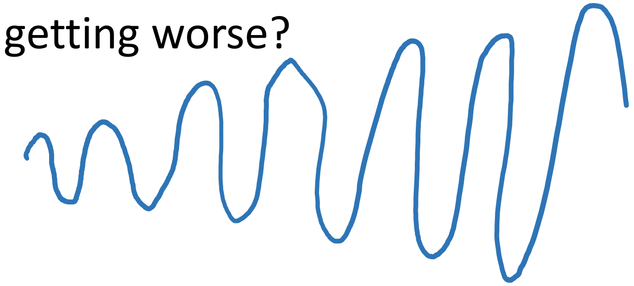
Temperatures are increasing?



Temperatures are increasing, and the increase is accelerating?



The extremes are getting worse?



https://www.metoffice.gov.uk/research/climate/maps-and-data/historic-station-data

You've got to have models in your head. And you've got to array your experience – both vicarious and direct – on this latticework of models.

You may have noticed students who just try to remember and pound back what is remembered. Well, they fail in school and in life. You've got to hang experience on a latticework of models in your head.

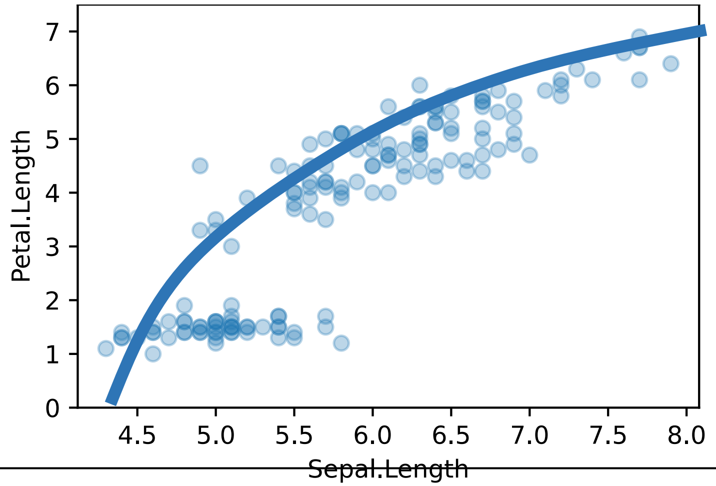
Charlie Munger, *A lesson on elementary, worldly wisdom as it relates to investment management & business.*

Example 2.1.1

The Iris dataset has 50 records of iris measurements, from three species.

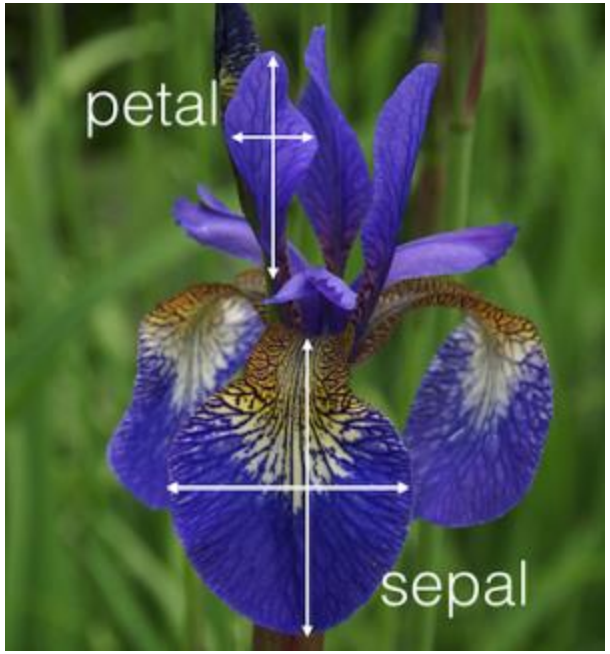
Petal. Length	Petal. Width	Sepal. Length	Sepal. Width	Species
1.0	0.2	4.6	3.6	setosa
5.0	1.9	6.3	2.5	virginica
5.8	1.6	7.2	3.0	virginica
4.2	1.2	5.7	3.0	versicolor
…				

How does `Petal.Length` depend on `Sepal.Length`?



Let's guess that for parameters $\alpha, \beta, \gamma, \sigma$ (to be estimated),

$$\texttt{Petal.Length} \sim \alpha + \beta\ \texttt{Sepal.Length} + \gamma(\texttt{Sepal.Length})^2 + N(0, \sigma^2)$$

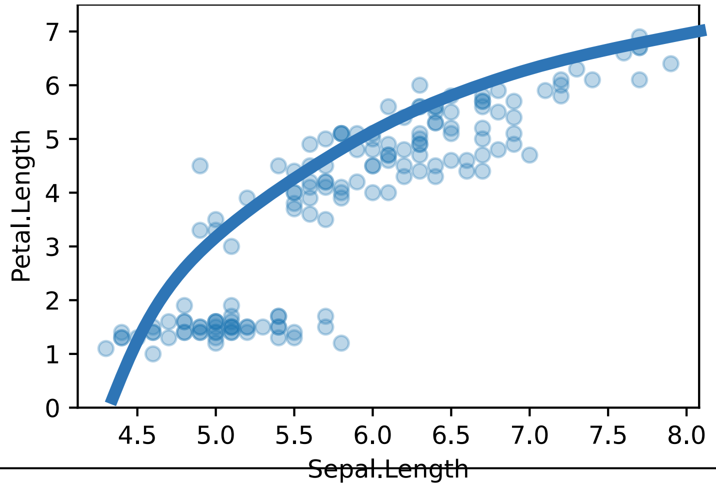


Example 2.1.1

The Iris dataset has 50 records of iris measurements, from three species.

Petal. Length	Petal. Width	Sepal. Length	Sepal. Width	Species
1.0	0.2	4.6	3.6	setosa
5.0	1.9	6.3	2.5	virginica
5.8	1.6	7.2	3.0	virginica
4.2	1.2	5.7	3.0	versicolor
…				

How does `Petal.Length` depend on `Sepal.Length`?



Let's guess that for parameters $\alpha, \beta, \gamma, \sigma$ (to be estimated),

$$\texttt{Petal.Length} \sim \alpha + \beta\ \texttt{Sepal.Length} + \gamma(\texttt{Sepal.Length})^2 + N(0, \sigma^2)$$

A linear model

unknown parameters to be estimated

$$\begin{bmatrix} PL_1 \\ PL_2 \\ \vdots \\ PL_n \end{bmatrix} \sim \alpha \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \beta \begin{bmatrix} SL_1 \\ SL_2 \\ \vdots \\ SL_n \end{bmatrix} + \gamma \begin{bmatrix} (SL_1)^2 \\ (SL_2)^2 \\ \vdots \\ (SL_n)^2 \end{bmatrix} + \begin{bmatrix} N(0,\sigma^2) \\ N(0,\sigma^2) \\ \vdots \\ N(0,\sigma^2) \end{bmatrix}$$

response vector

feature vectors

Not a linear model

$$\text{temp} = \alpha + \beta \sin(2\pi(t+\phi)) + \gamma t + N(0, \sigma^2)$$

2.1. LINEAR MODELS

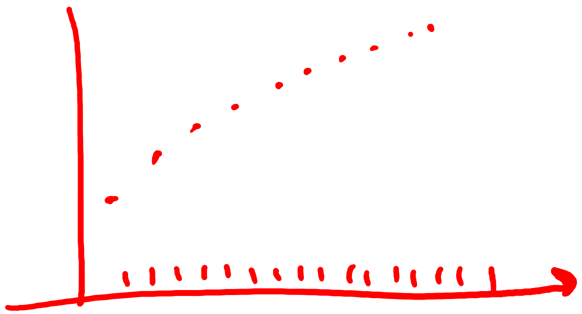
$$\texttt{Petal.Length} \approx \alpha + \beta\, \texttt{Sepal.Length} + \gamma (\texttt{Sepal.Length})^2$$

$$\begin{bmatrix} \mathrm{PL}_1 \\ \mathrm{PL}_2 \\ \vdots \\ \mathrm{PL}_n \end{bmatrix} \approx \alpha \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \beta \begin{bmatrix} \mathrm{SL}_1 \\ \mathrm{SL}_2 \\ \vdots \\ \mathrm{SL}_n \end{bmatrix} + \gamma \begin{bmatrix} (\mathrm{SL}_1)^2 \\ (\mathrm{SL}_2)^2 \\ \vdots \\ (\mathrm{SL}_n)^2 \end{bmatrix}$$

Models of this form are called *linear models* (because they're based on linear algebra).

They are flexible, and very fast to optimize.

We'll assume Gaussian errors. Thus, maximum likelihood estimation is the same as minimizing squared prediction loss. Linear modelling is also called "least squares model-fitting".



$$\texttt{Petal.Length} \approx \alpha + \beta\, \texttt{Sepal.Length} + \gamma (\texttt{Sepal.Length})^2$$

$$\begin{bmatrix} PL_1 \\ PL_2 \\ \vdots \\ PL_n \end{bmatrix} \approx \alpha \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + \beta \begin{bmatrix} SL_1 \\ SL_2 \\ \vdots \\ SL_n \end{bmatrix} + \gamma \begin{bmatrix} (SL_1)^2 \\ (SL_2)^2 \\ \vdots \\ (SL_n)^2 \end{bmatrix}.$$

```
iris = pandas.read_csv(...)
```

Fitting the model

```
one, SL, PL = np.ones(len(iris)), iris['Sepal.Length'], iris['Petal.Length']
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(np.column_stack([one, SL, SL**2]), PL)
(α,β,γ) = model.coef_
```

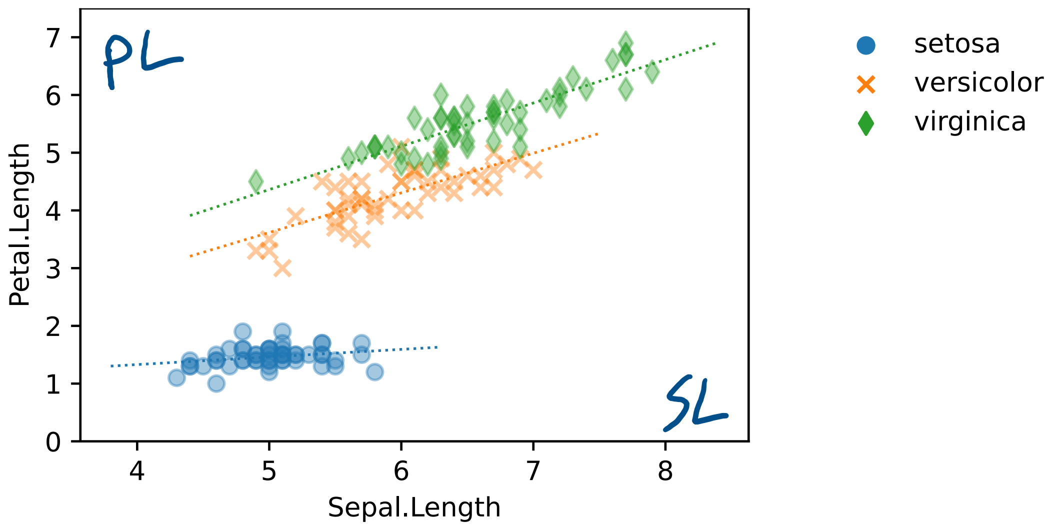
Making predictions / getting fitted values from the model

```
newSL = np.linspace(4.2, 8.2, 20)
predPL = model.predict(np.column_stack([one, newSL, newSL**2]))
```

Feature design

How do we design features, so that linear models answer the questions we want answered?

ONE-HOT CODING



Want to fit three straight lines

$PL \approx \alpha_{species} + \beta_{species} SL$

This has six parameters to fit:

α_{seto} α_{vers} α_{virg}

β_{seto} β_{vers} γ_{virg}

e.g. first row is setosa, and linear model says

$PL_1 \approx \alpha_{seto} + \beta_{seto} SL_1$

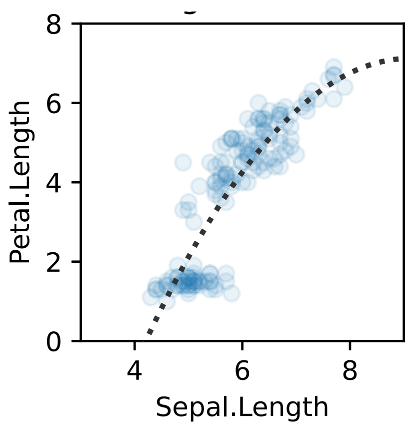
$1_{species=setosa}$ $\quad$ $1_{species=setosa} \times SL$

Linear model form
(linear combination of features, weighted by parameters)

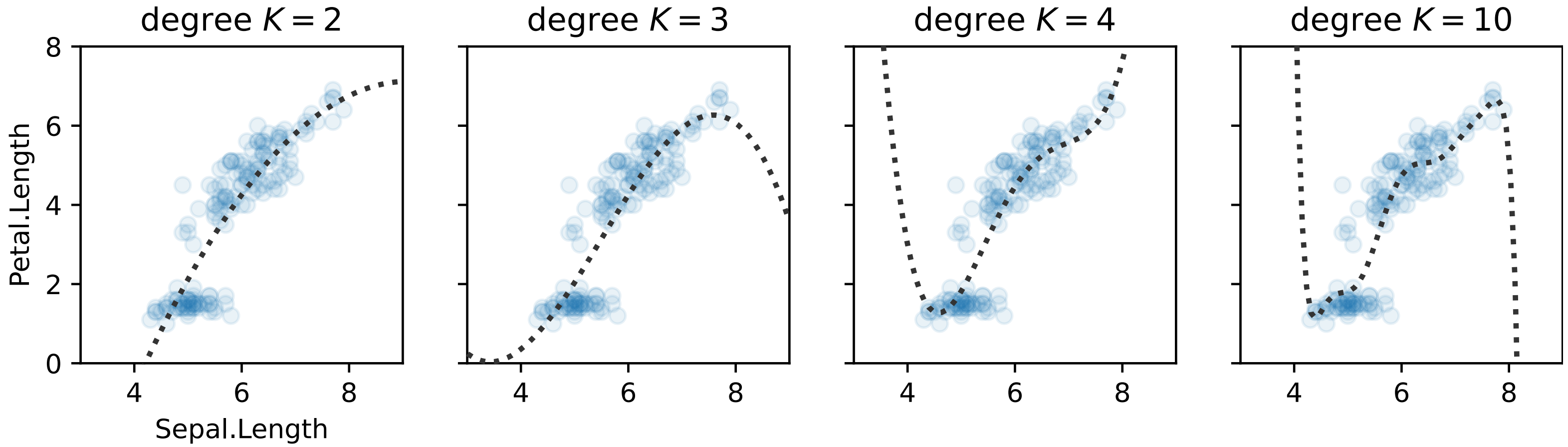
$$
\begin{matrix} \text{seto} \\ \text{virg} \\ \text{virg} \\ \text{seto} \\ \text{vers} \\ \\ \end{matrix}
\begin{bmatrix} PL_1 \\ PL_2 \\ PL_3 \\ PL_4 \\ PL_5 \\ \vdots \end{bmatrix}
\approx \alpha_{\text{seto}} \begin{bmatrix} \vdots \end{bmatrix}
+ \alpha_{\text{virg}} \begin{bmatrix} \vdots \end{bmatrix}
+ \alpha_{\text{vers}} \begin{bmatrix} \vdots \end{bmatrix}
+ \beta_{\text{seto}} \begin{bmatrix} \vdots \end{bmatrix}
+ \beta_{\text{virg}} \begin{bmatrix} \vdots \end{bmatrix}
+ \beta_{\text{vers}} \begin{bmatrix} \vdots \end{bmatrix}
$$

```
species, SL = iris['Species'], iris['Sepal.Length']
PL = iris['Petal.Length']
species_levels = ['setosa','versicolor','virginica']
i1,i2,i3 = [np.where(species==k,1,0) for k in species_levels]
X = np.column_stack([i1, i2, i3, i1*SL, i2*SL, i3*SL])
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, PL)
```

NON-LINEAR RESPONSE

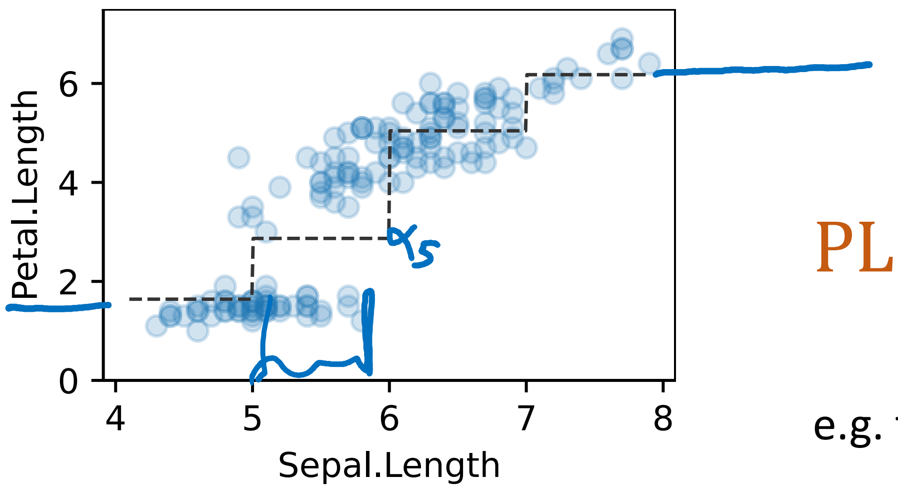


Petal.Length ≈
$$\alpha + \beta\ \text{Sepal.Length} + \gamma(\text{Sepal.Length})^2$$



Petal.Length ≈
$$\beta_0 + \sum_{k=1}^{K} \beta_k (\text{Sepal.Length})^k$$

NON-LINEAR RESPONSE via one-hot coding

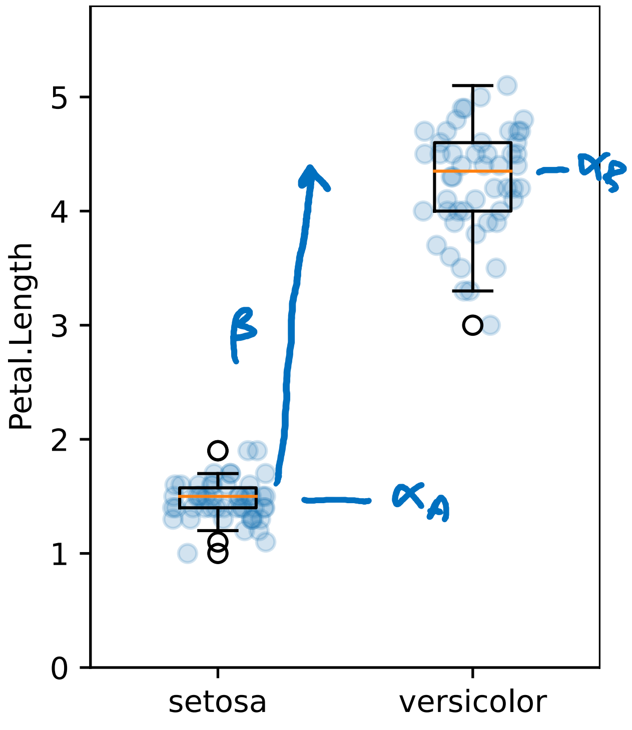


$$PL \approx \alpha_4 1_{SL<4} + \alpha_5 1_{\lfloor SL \rfloor=5} + \alpha_6 1_{\lfloor SL \rfloor=6} + \alpha_7 1_{SL\geq 7}$$

e.g. for an observation with SL=5.3, we predict PL $\approx \alpha_5$

e.g. for an observation with SL=3.1, we predict PL $\approx \alpha_4$

COMPARING GROUPS



Measurements for condition A: $\mathrm{a} = [\mathrm{a}_1, \mathrm{a}_2, \dots, \mathrm{a}_m]$
Measurements for condition B: $\mathrm{b} = [\mathrm{b}_1, \mathrm{b}_2, \dots, \mathrm{b}_n]$

Can we use a linear model to compare A and B?

$$x \approx \alpha_A 1_{cond=A} + \alpha_B 1_{cond=B}$$

$$x \approx \alpha + \beta 1_{cond=B}$$

for an indiv. of type A: $x \approx \alpha$

for an indiv. of type B: $x \approx \alpha + \beta$

condition	x
A	a_1
⋮	⋮
A	a_m
B	b_1
⋮	⋮
B	b_n

resp. vector

MODEL DIAGNOSIS

After we fit a model, how do we learn if it's a good fit?

1. Evaluate its log likelihood
2. Hypothesis testing [next week]
3. Eyeball it!

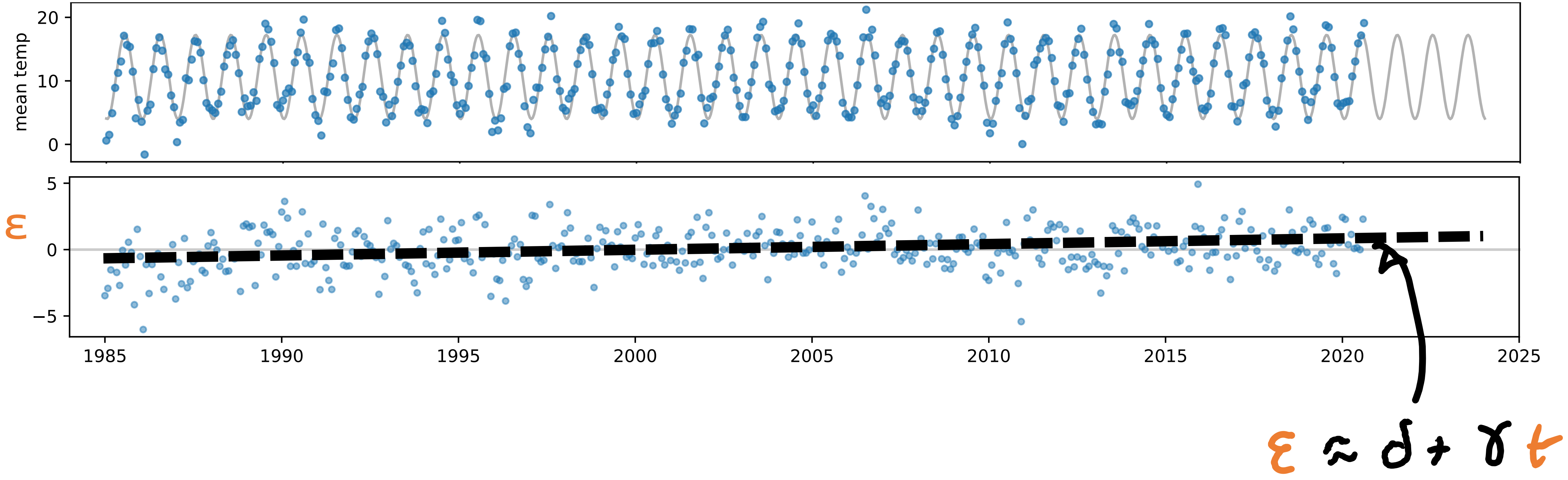
Find the log likelihood of each datapoint, and showcase some datapoints with very low or very high likelihood

Find the prediction error for each datapoint, and plot it every way we can think of

If we hadn't thought to include climate change in our temperature model ...

$$\texttt{temp} \approx \alpha + \beta \sin\big(2\pi(\texttt{t} + \phi)\big)$$

$$\texttt{temp} = \alpha + \beta \sin\big(2\pi(\texttt{t} + \phi)\big) + \varepsilon$$



This suggests a revised model ...

$$\texttt{temp} = \alpha' + \beta' \sin\big(2\pi(\texttt{t} + \phi)\big) + \gamma \texttt{t} + \varepsilon$$

Q. Should we just keep adding more and more features to our model?

A. No. If we did, we'd overfit.