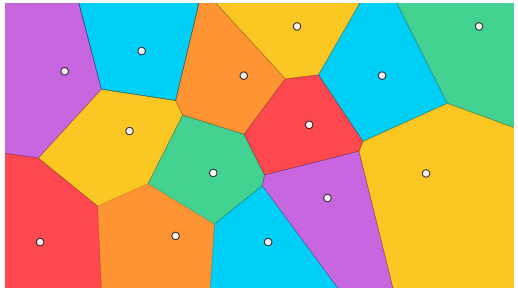
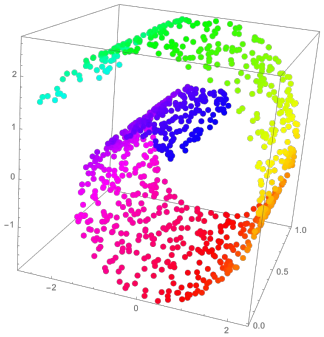
NYU CS-GY 6923 Machine Learning

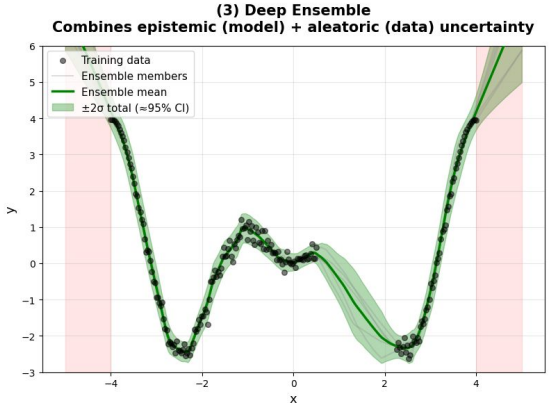
Prof. Pavel Izmailov

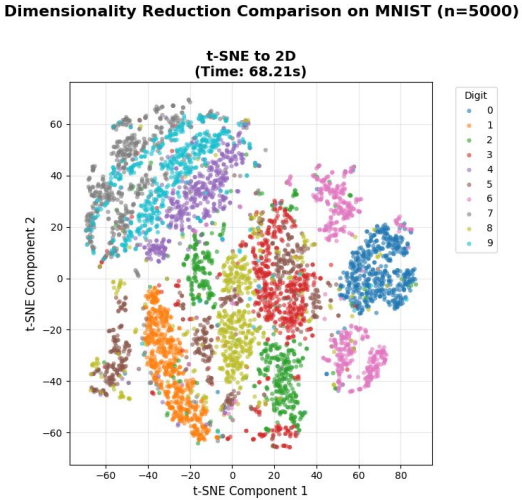
Today

- Uncertainty in Machine Learning
- K-Nearest Neighbors
- Dimensionality Reduction









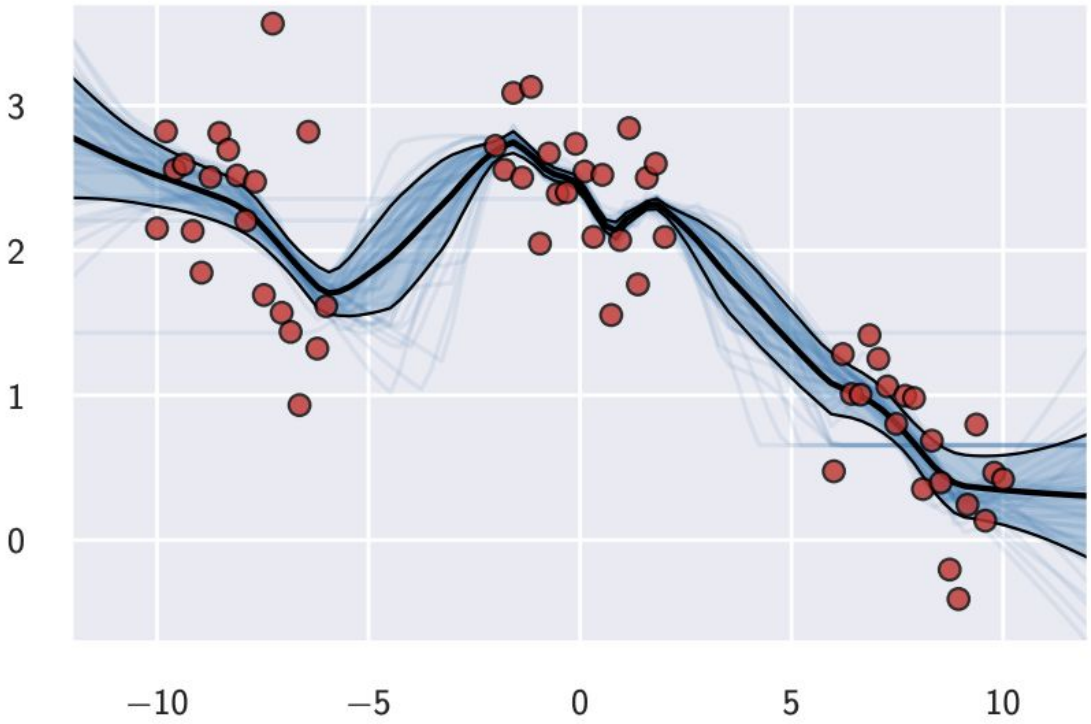

Uncertainty

Uncertainty

Uncertainty is important when we make decisions based on model predictions:

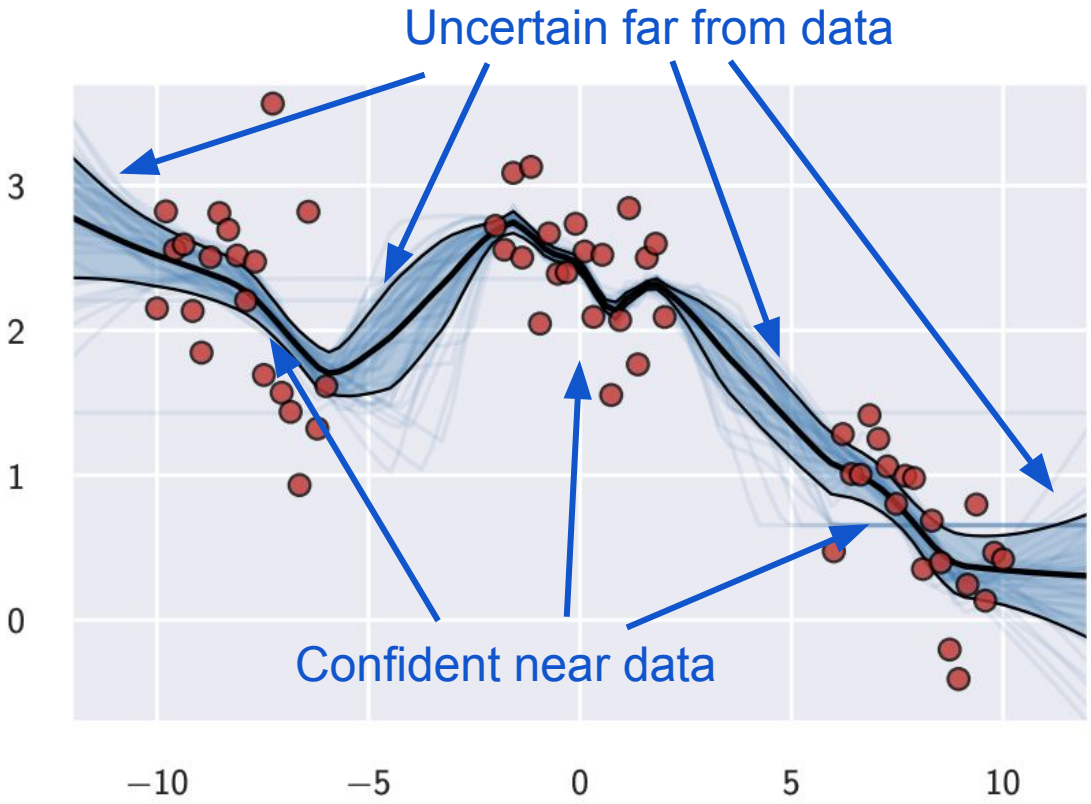
- It will rain tomorrow with probability X%
- Stock price will go up with probability X%
- Based on the X-ray, you are healthy with probability X%
- The time to get to the airport is within range [A, B]
- …

Uncertainty in Regression



In addition to accurate predictions, we often want to know how confident we should be.

Uncertainty in Regression

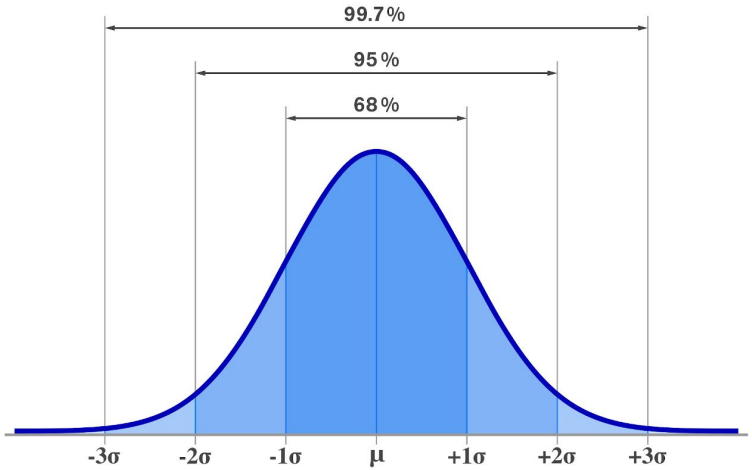


In addition to accurate predictions, we often want to know how confident we should be.

Uncertainty in Regression

We will work with *likelihoods*.
Remember normal distribution:

$$\mathcal{N}(y|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$



$$\log \mathcal{N}(y|\mu,\sigma^2) = -\frac{(y-\mu)^2}{2\sigma^2} + C$$

Normal likelihood corresponds to MSE loss.

Uncertainty in Regression

Normal likelihood corresponds to MSE loss.

$$\log \mathcal{N}(y|\mu, \sigma^2) = -\frac{(y-\mu)^2}{2\sigma^2} + C$$

Observations independent given the model

$$\log p(y|X, w) = \log \prod_{i=1}^{N} p(y_i|x_i, w) = \sum_{i=1}^{N} \log p(y_i|x_i, w) =$$

Uncertainty in Regression

Normal likelihood corresponds to MSE loss.

$$\log \mathcal{N}(y|\mu, \sigma^2) = -\frac{(y-\mu)^2}{2\sigma^2} + C$$

Observations independent given the model

$$\log p(y|X, w) = \log \prod_{i=1}^{N} p(y_i|x_i, w) = \sum_{i=1}^{N} \log p(y_i|x_i, w) =$$

$$\sum_{i=1}^{N} \log \mathcal{N}(y_i|f(x_i, w), \sigma^2) = \sum_{i=1}^{N} -\frac{(y_i - f(x_i, w))^2}{2\sigma^2} + C$$

Model prediction is the mean of the normal distribution, σ represents uncertainty

Uncertainty in Regression

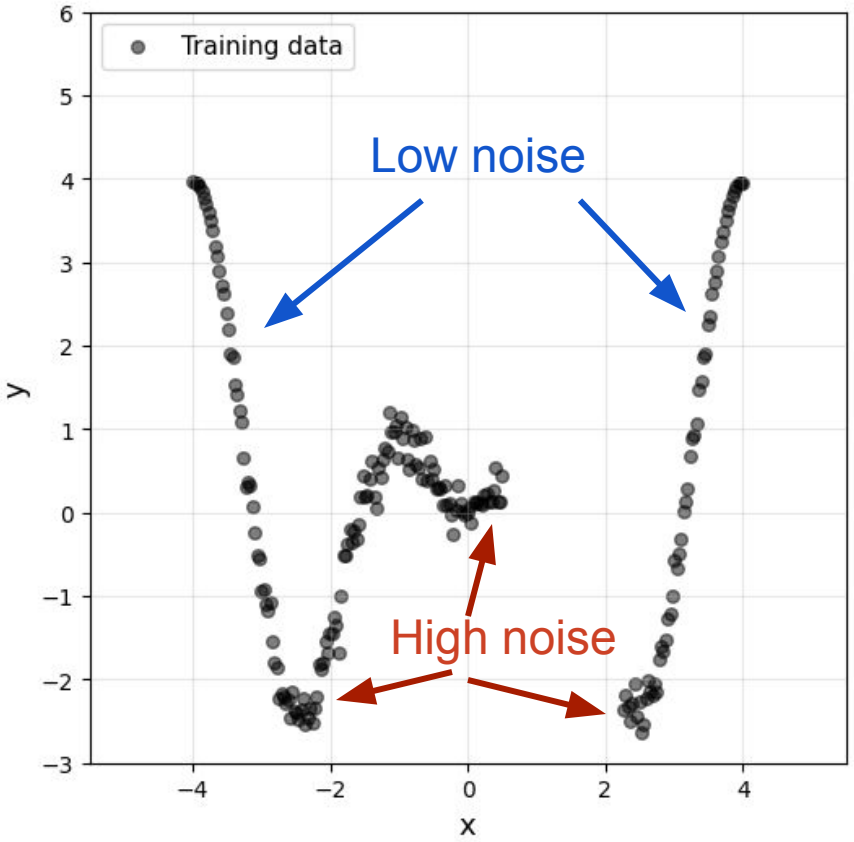
Normal likelihood corresponds to MSE loss.

$$\log \mathcal{N}(y|\mu, \sigma^2) = -\frac{(y-\mu)^2}{2\sigma^2} + C$$

$$\arg\max_w \sum_{i=1}^{N} -\frac{(y_i - f(x_i, w))^2}{2\sigma^2} + C = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i, w))^2$$

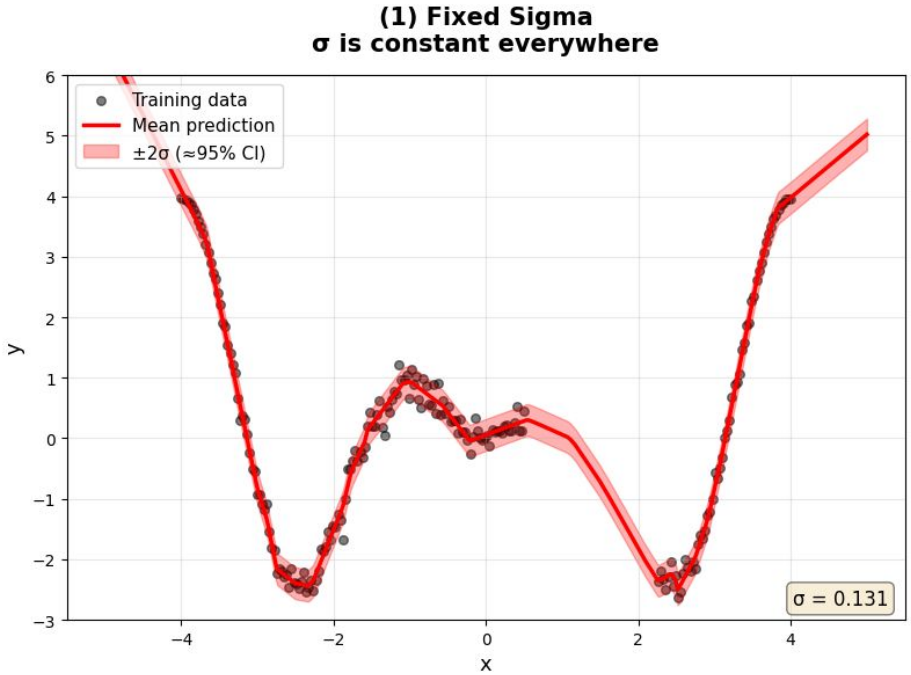
With any fixed noise variance σ^2 maximizing the likelihood is equivalent to minimizing mean squared error.

Uncertainty in Regression



Data with *heteroscedastic* noise: different amount of noise at different inputs.

Uncertainty in Regression



Data with *heteroscedastic* noise: different amount of noise at different inputs.

- We can use a constant σ, a hyperparameter
- But then we get constant uncertainty everywhere

Uncertainty in Regression

Let's make the model output σ as well!

$$\frac{1}{N}\log p(y|X,w) = \frac{1}{N}\sum_{i=1}^{N}\log \mathcal{N}(y_i|f(x_i,w),\sigma(x_i,w)^2) =$$

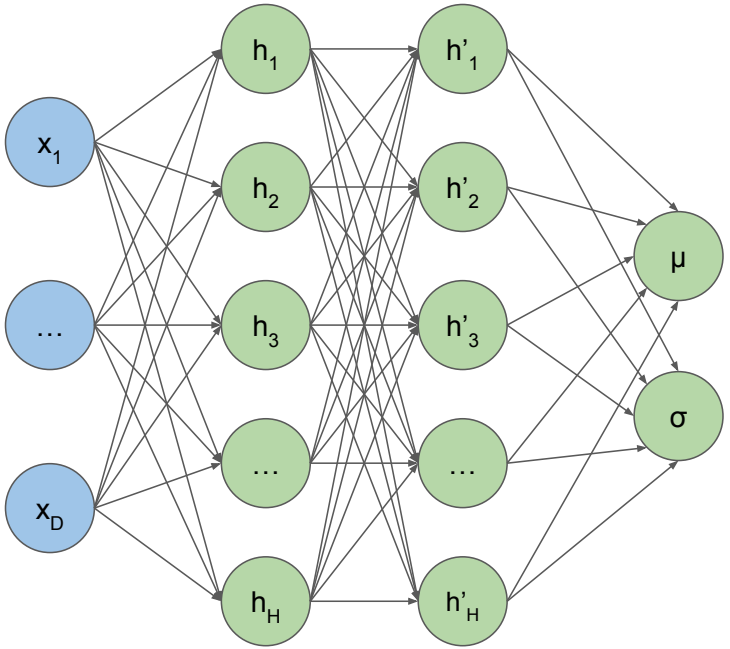
$$\frac{1}{N}\sum_{i=1}^{N}\left[-\frac{(y_i - f(x_i,w))^2}{2\sigma(x_i,w)^2} - \log\left(\sigma(x_i,w)\sqrt{2\pi}\right)\right]$$

We can no longer ignore this term

- Now the model can learn to have a non-constant uncertainty that changes from input to input!
- The likelihood formulation allows us to naturally derive a training objective

Uncertainty in Regression

- We can modify the model to have two outputs: prediction and uncertainty
- Linear output layer with two dimensions
- The variance needs to be positive, so we can specify it as *exp(log_sigma),* where the model outputs *log_sigma*

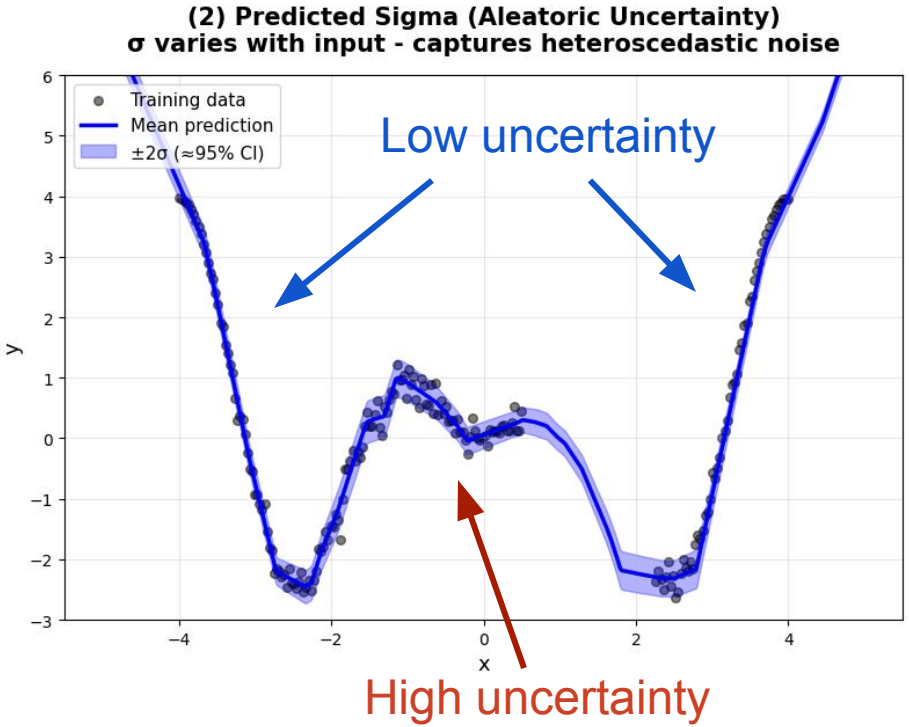


Uncertainty in Regression

- We can modify the model to have two outputs: prediction and uncertainty
- Linear output layer with two dimensions
- The variance needs to be positive, so we can specify it as *exp(log_sigma),* where the model outputs *log_sigma*

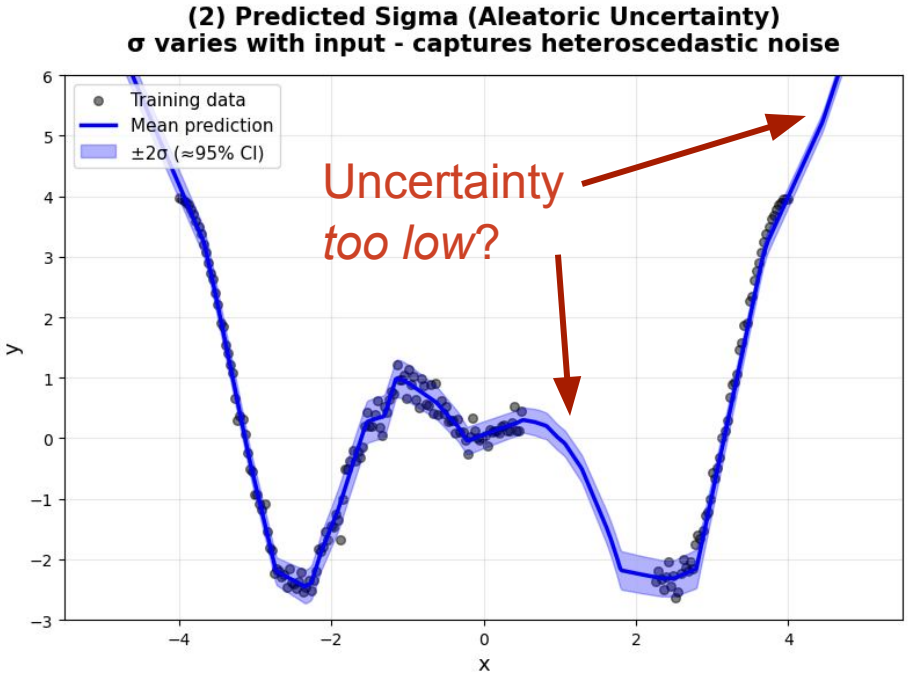
```
class MLPPredictedSigma(nn.Module):
    def __init__(self, hidden_size=50):
        super().__init__()
        # Shared feature extraction layers
        self.shared = nn.Sequential(
            nn.Linear(1, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
        )
        # Separate heads for mean and variance
        self.mu_head = nn.Linear(hidden_size, 1)
        self.log_sigma_head = nn.Linear(hidden_size, 1)

    def forward(self, x):
        features = self.shared(x)
        mu = self.mu_head(features)
        log_sigma = self.log_sigma_head(features)
        sigma = torch.exp(log_sigma) # Ensure sigma >
0
        return mu, sigma
```

Uncertainty in Regression



- Now, uncertainty varies with inputs!
- We managed to capture the structure of noise in the data

Uncertainty in Regression



- Now, uncertainty varies with inputs!
- We managed to capture the structure of noise in the data
- But we get low uncertainty even in parts of the input space where we have no data…

Aleatoric and Epistemic Uncertainty

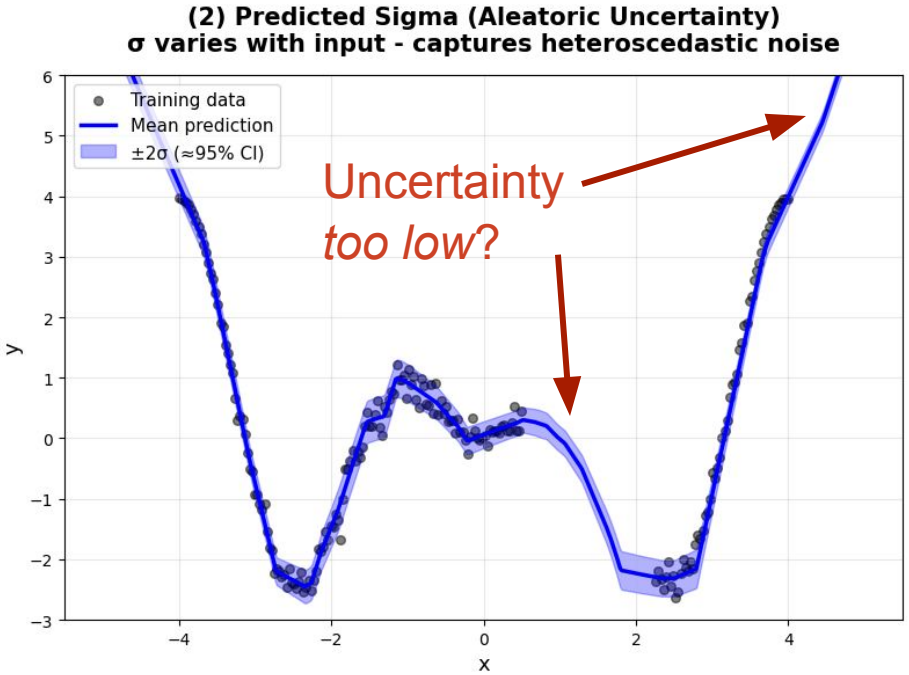
Aleatoric uncertainty:

- Property of the data
- Irreducible, no matter how much data we collect
- E.g. measurement noise
- High when data is noisy
- Uncertain what the observations are given the latent function

Epistemic uncertainty:

- Property of the observer
- Reducible with more data
- E.g. uncertain if $y(x)$ is linear or quadratic
- High when far from training data
- Uncertain what the latent data-generating function is

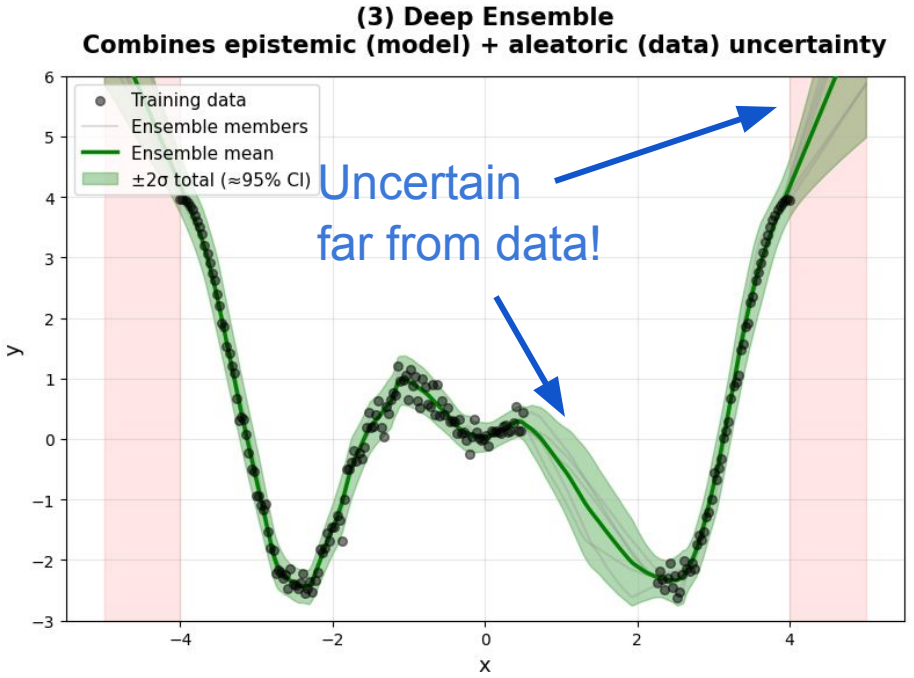
Uncertainty in Regression



- Now, uncertainty varies with inputs!
- We managed to capture the structure of noise in the data
- But we get low uncertainty even in parts of the input space where we have no data…
- So far, we captured aleatoric but not epistemic uncertainty

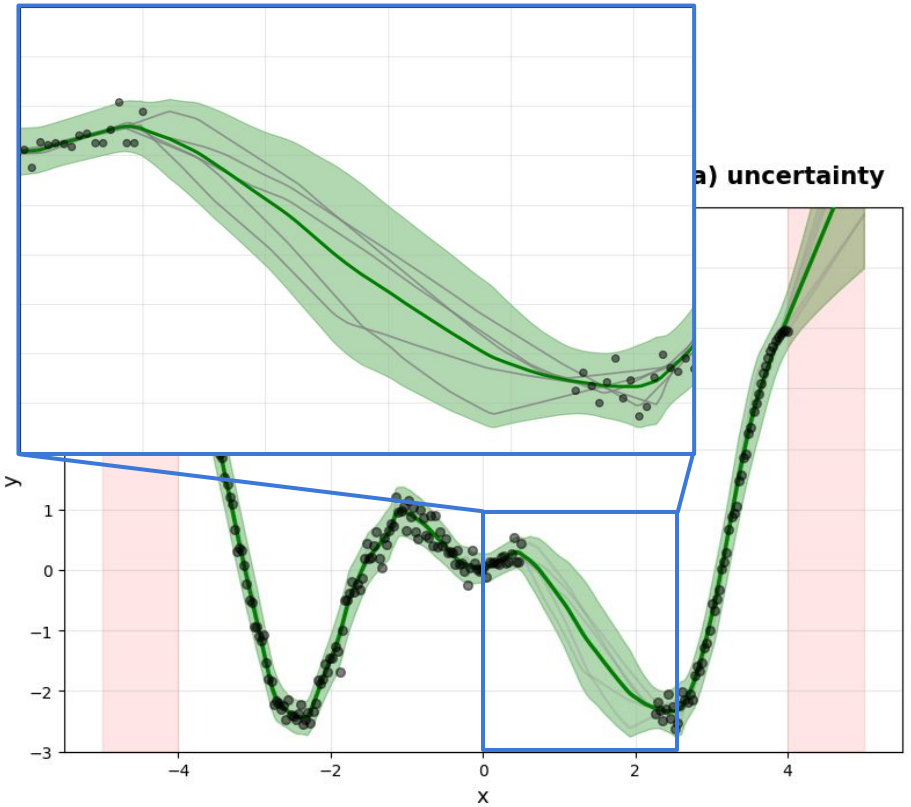
Uncertainty in Regression

- Let's train a *deep ensemble*!
- The models in the ensemble are not constrained far from the data
- Because each model fits the data differently, we capture uncertainty

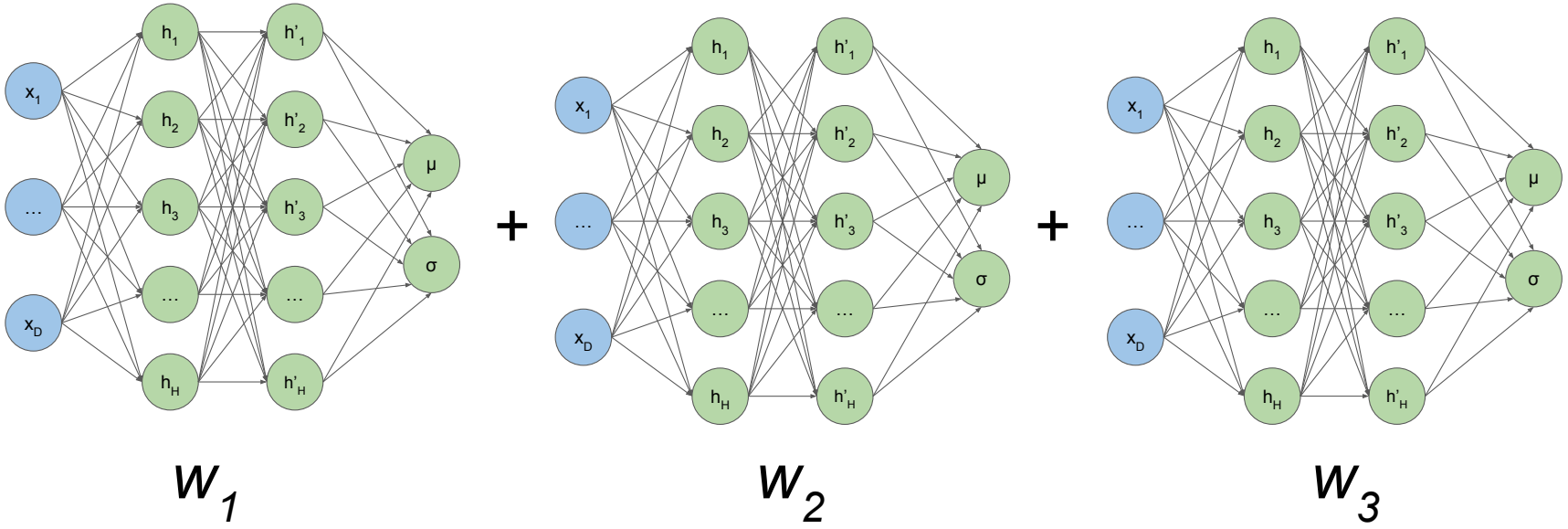


Uncertainty in Regression

- Let's train a *deep ensemble*!
- The models in the ensemble are not constrained far from the data
- Because each model fits the data differently, we capture uncertainty
- Ensembling captures *epistemic* uncertainty



Deep Ensembles for Regression



- We simply train the model *M* times using different random initializations
- Combine predictions

Deep Ensembles for Regression

$$f_{\text{ens}}(x) = \frac{1}{M}\sum_{i=1}^{M} f(x, w_i)$$

Mean prediction is just the mean over ensemble members

$$(\sigma_{\text{ens}}^{\text{ale}})^2 = \frac{1}{M}\sum_{i=1}^{M} \sigma(x, w_i)^2$$

Aleatoric uncertainty is the mean uncertainty over the ensemble

$$(\sigma_{\text{ens}}^{\text{epi}})^2 = \frac{1}{M}\sum_{i=1}^{M} \big(f(x, w_i) - f_{\text{ens}}(x)\big)^2$$

Epistemic uncertainty is the uncertainty in the mean prediction

$$(\sigma_{\text{ens}}^{\text{total}})^2 = (\sigma_{\text{ens}}^{\text{epi}})^2 + (\sigma_{\text{ens}}^{\text{ale}})^2$$

Total uncertainty is the sum of the two

Deep Ensembles for Regression

$$f_{\text{ens}}(x) = \frac{1}{M}\sum_{i=1}^{M} f(x, w_i) \approx \mathbb{E}_f \mathbb{E}[x|f]$$

$$(\sigma_{\text{ens}}^{\text{ale}})^2 = \frac{1}{M}\sum_{i=1}^{M} \sigma(x, w_i)^2 \approx \mathbb{E}_f \text{Var}[x|f]$$

$$(\sigma_{\text{ens}}^{\text{epi}})^2 = \frac{1}{M}\sum_{i=1}^{M} \big(f(x, w_i) - f_{\text{ens}}(x)\big)^2 \approx \text{Var}_f \mathbb{E}[x|f]$$

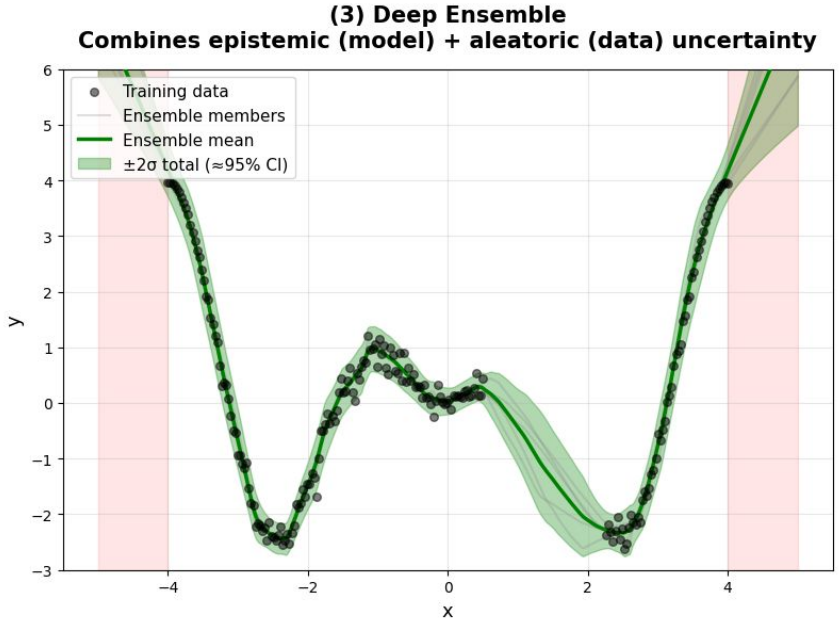
$$(\sigma_{\text{ens}}^{\text{total}})^2 = (\sigma_{\text{ens}}^{\text{epi}})^2 + (\sigma_{\text{ens}}^{\text{ale}})^2 \approx \text{Var}[x]$$

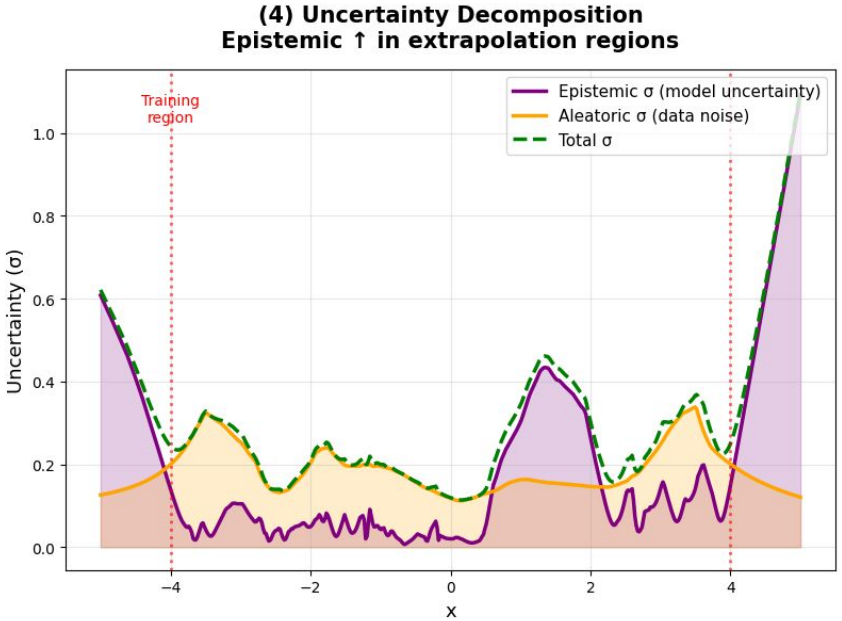
These formulas come from probability theory (total variance formula)...

Don't worry if these are not intuitive

- Ensembles can capture uncertainty over the latent functions
- We can combine predictions across the ensemble members to get a better uncertainty estimate

Aleatoric and Epistemic Uncertainty





- We can visualize aleatoric and epistemic uncertainty separately
- Our deep ensemble formulation captures both types of uncertainty

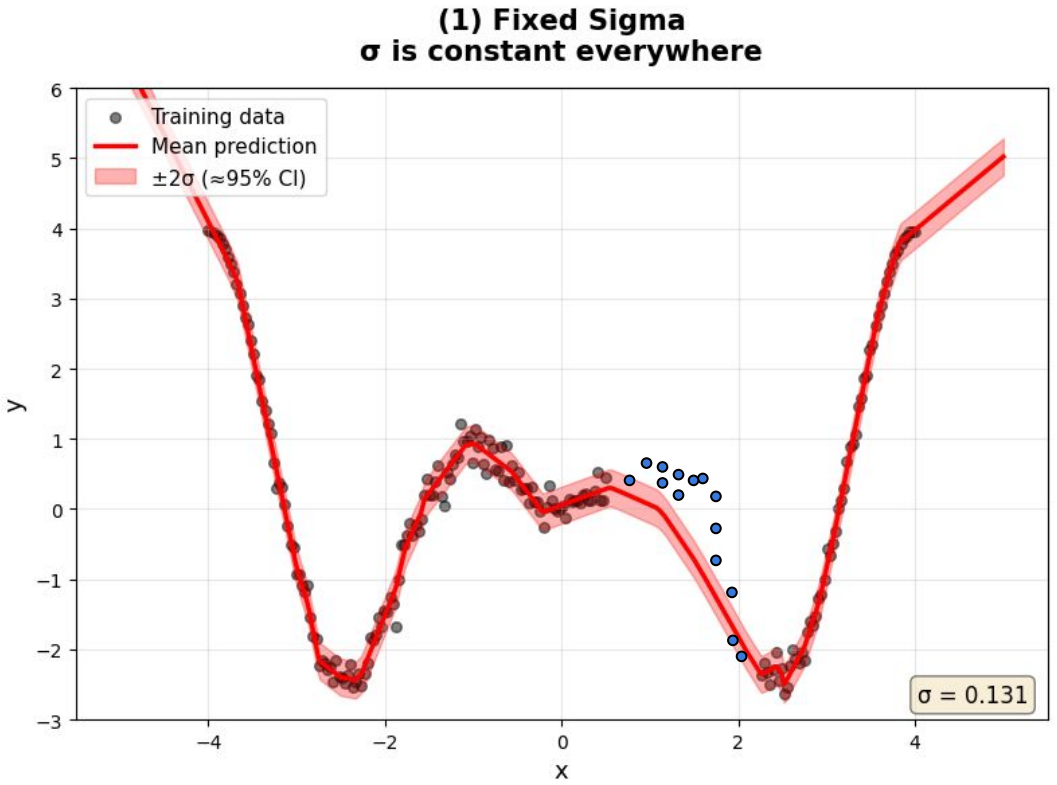
Measuring the Quality of Uncertainty

$$\frac{1}{N}\log p(y|X,w) = \frac{1}{N}\sum_{i=1}^{N}\log \mathcal{N}(y_i|f(x_i,w),\sigma(x_i,w)^2) =$$

$$\frac{1}{N}\sum_{i=1}^{N}\left[-\frac{(y_i - f(x_i,w))^2}{2\sigma(x_i,w)^2} - \log\left(\sigma(x_i,w)\sqrt{2\pi}\right)\right]$$

We can estimate likelihood on held-out data. High likelihood ⇒ good predictions and uncertainty.

Measuring the Quality of Uncertainty

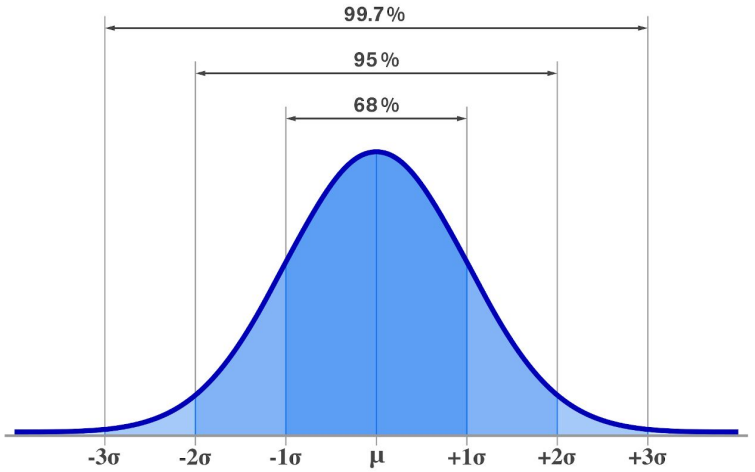


When the model cannot predict the test data well, it needs to be uncertain. Predicting incorrectly and confidently ⇒ very low likelihood.

Measuring the Quality of Uncertainty

Uncertainty calibration: We expect the data to be distributed similar to the model predictions. Suppose the model predicts (μ(x), σ(x)) at each test point

- Among all test datapoints, we should expect ~70% to lie in 1-σ region around μ
- ~ 95% to lie in 2-σ region
- …



Remember μ(x), σ(x) are different for each datapoint here

Uncertainty in Classification

Cat: 98%

Dog: 98%

- Imagine we train a classifier on binary classification: cats vs dogs
- Predicted class probabilities represent confidence ↔ uncertainty

Uncertainty in Classification

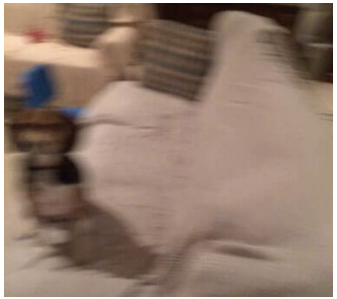
Cat: 98%

Dog: 98%

????

????

????



- Imagine we train a classifier on binary classification: cats vs dogs
- Predicted class probabilities represent confidence ↔ uncertainty
- We may want the model to be uncertain when at test time the input is ambiguous

Uncertainty in Classification

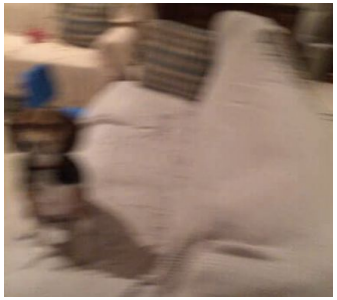
Cat: 98%

Dog: 98%

Cat: 85%

Dog: 92%

Cat: 72%



- Imagine we train a classifier on binary classification: cats vs dogs
- Predicted class probabilities represent confidence ↔ uncertainty
- We may want the model to be uncertain when at test time the input is ambiguous
- Naively, models tend to be *overconfident*

Aleatoric and Epistemic Uncertainty

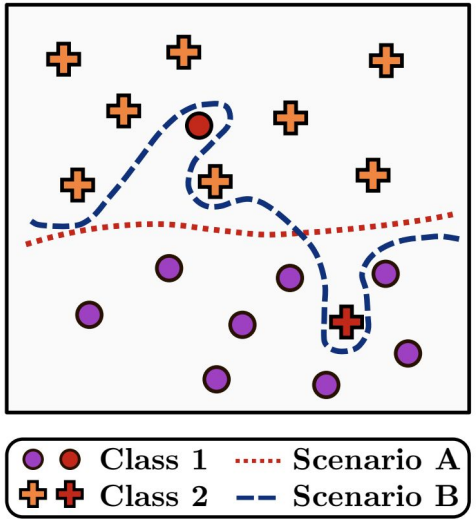
Aleatoric uncertainty in classification:

- Label noise: errors in data

Epistemic uncertainty in classification:

- Uncertainty in the classification rule

- **Scenario A:** You expect noise in labels ⇒ prefer simpler classification rule
- **Scenario B:** You expect noise in labels ⇒ prefer more complex classification rule



Uncertainty in Classification

- 100-class classification with CNNs
- Given input x, predict a distribution over classes c:

$$0 \leq p_{\text{CNN}}(c|x) \leq 1, \qquad \sum_{c=1}^{100} p_{\text{CNN}}(c|x) = 1$$

- Confidence for a test input *x*: maximum predicted class probability

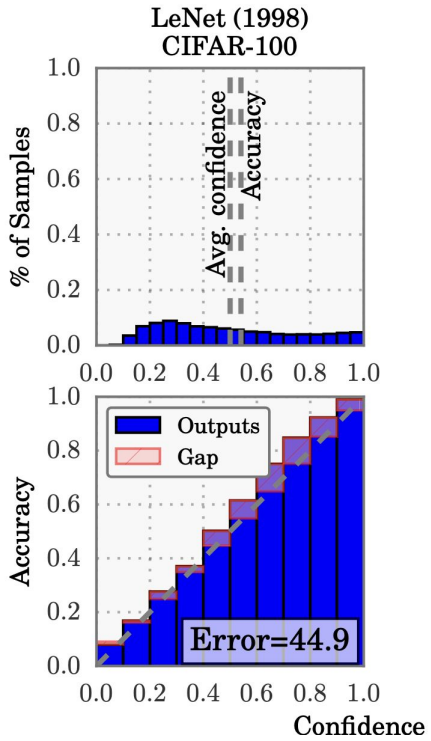
$$\max_{c \in \{1 \ldots 100\}} p_{\text{CNN}}(c|x)$$

Cat: 98%

Dog: 96%

Duck: 18%

Uncertainty in Classification

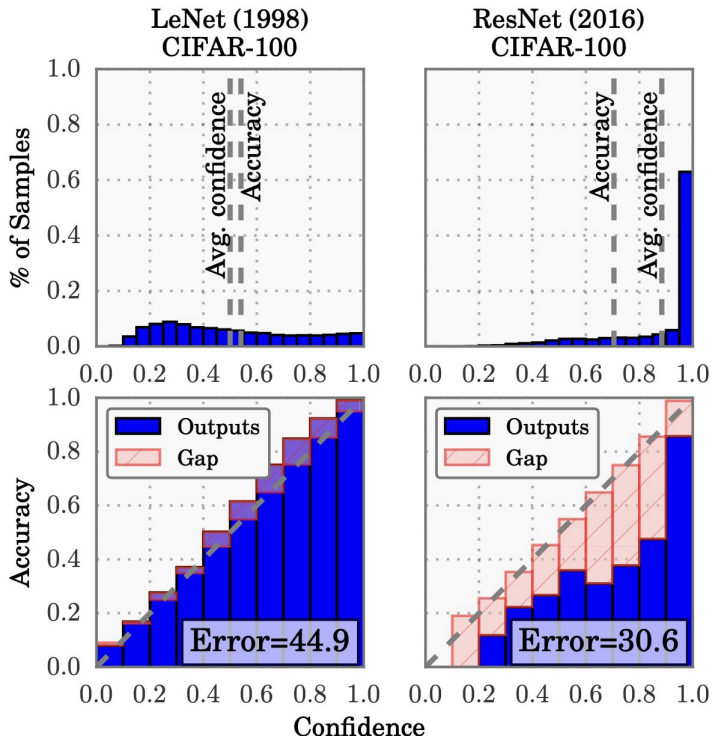


- **Calibration:** model confidence matches model accuracy
- If we take all test datapoints where the confidence is ≈X%, the accuracy on these datapoints should be ≈X%
- Bin all datapoints according to the confidence; compute accuracy within each bin
- Expected calibration error (ECE): average difference between confidence and accuracy

[1706.04599] On Calibration of Modern Neural Networks

Uncertainty in Classification

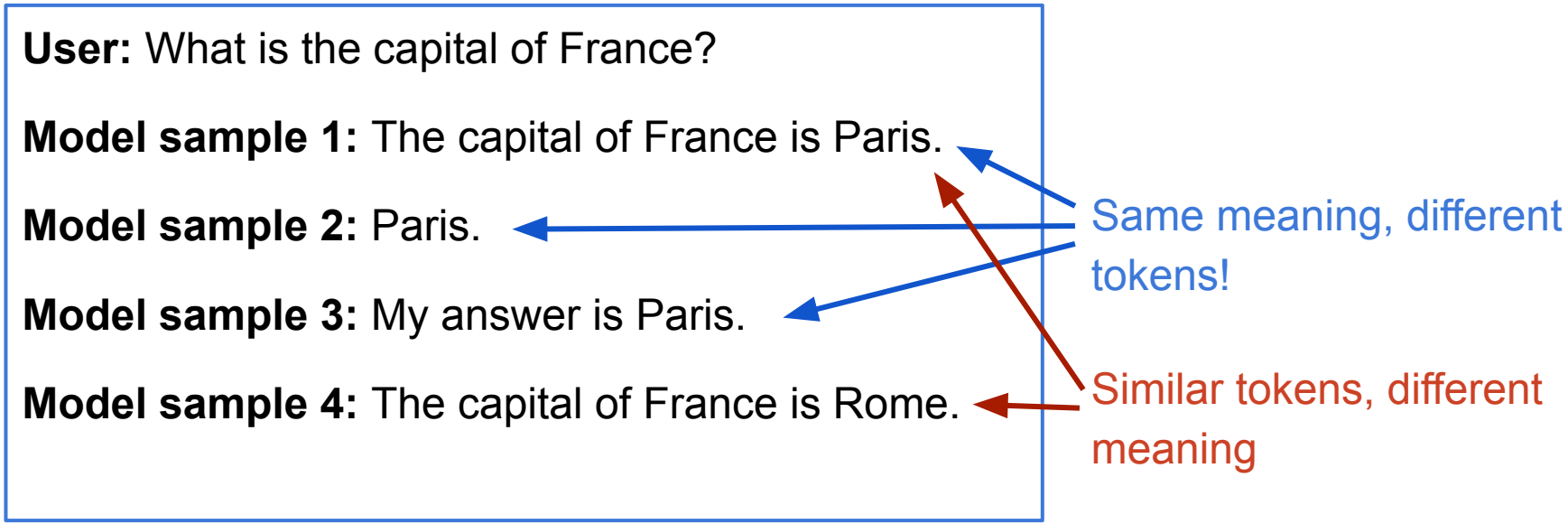
ResNet has much better accuracy but much worse calibration!



- **Calibration:** model confidence matches model accuracy
- If we take all test datapoints where the confidence is ≈X%, the accuracy on these datapoints should be ≈X%
- Bin all datapoints according to the confidence; compute accuracy within each bin
- Expected calibration error (ECE): average difference between confidence and accuracy

[1706.04599] On Calibration of Modern Neural Networks

Uncertainty in Language Models



In natural language generation, we care about *semantic* uncertainty, not uncertainty over individual tokens.

Uncertainty in Language Models

Name papers by AI researcher Pavel Izmailov. Do not search, just use your memory

Here are the papers I can list **from prior knowledge**, without searching, that are authored or co-authored by you (Pavel Izmailov). I'll group them by topic so you can check if anything is missing.

Bayesian Deep Learning & Generalization

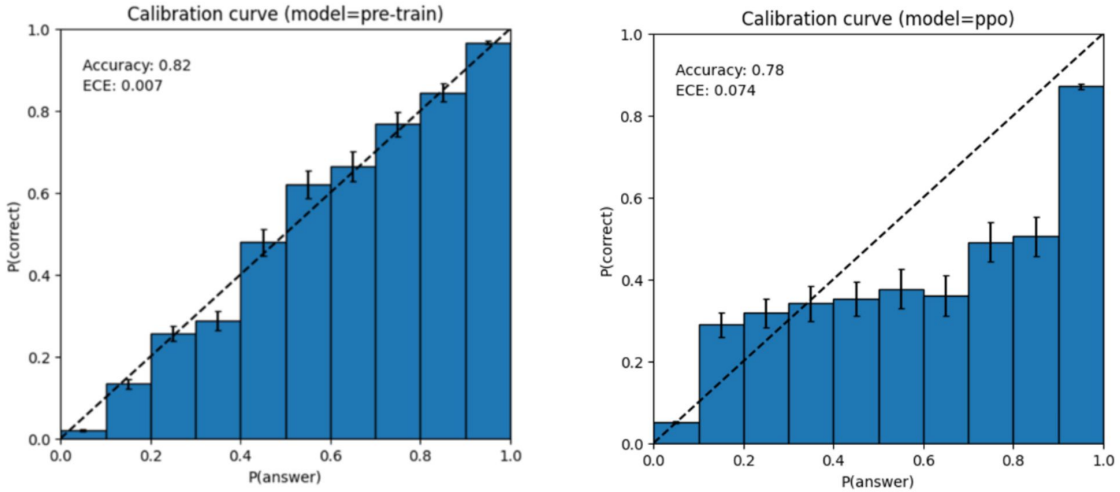
Stochastic Weight Averaging (SWA)

- **Averaging Weights Leads to Wid… …er Generalization**
  Pavel Izmailov, Dmitrii Podoprikhin, Trevor Cushman, Wesley Maddox, Andrew Gordon Wilson
  (Introduced SWA)

Ask ChatGPT

Hallucinations are a related problem: models confidently give incorrect information.

Uncertainty in Language Models

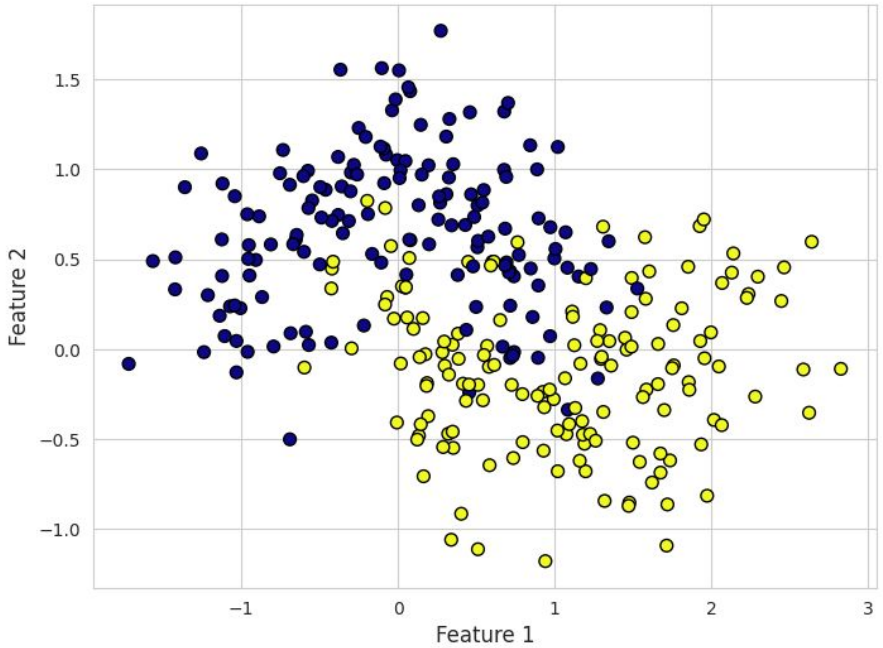


Left: Calibration plot of the pre-trained GPT-4 model on an MMLU subset. The model's confidence in its prediction closely matches the probability of being correct. The dotted diagonal line represents perfect calibration. Right: Calibration plot of post-trained PPO GPT-4 model on the same MMLU subset. Our current process hurts the calibration quite a bit.

https://openai.com/index/gpt-4-research/

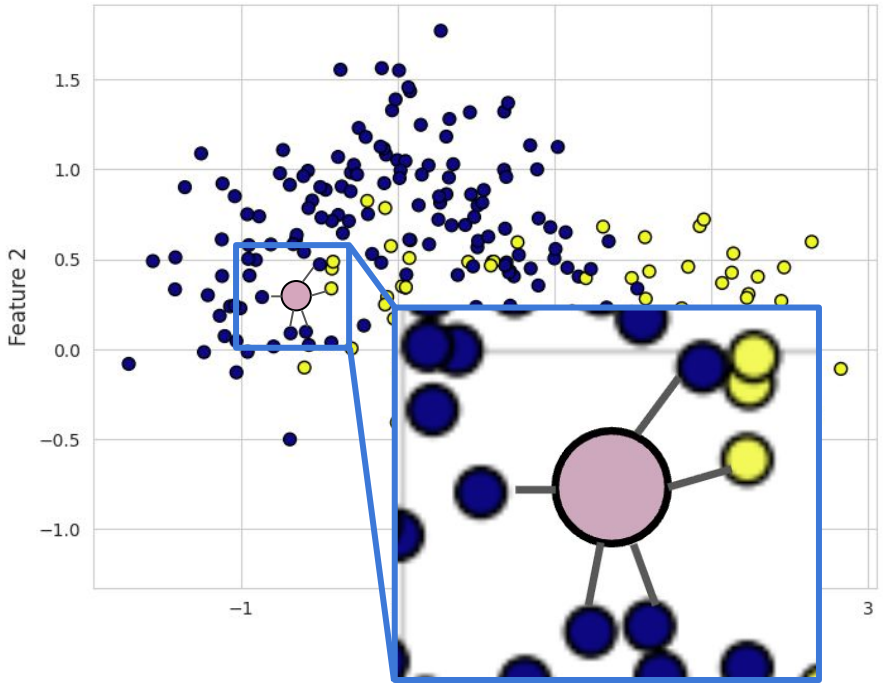
k-Nearest Neighbors

k-Nearest Neighbors



We will now discuss another method for classification and regression: kNN.

k-Nearest Neighbors



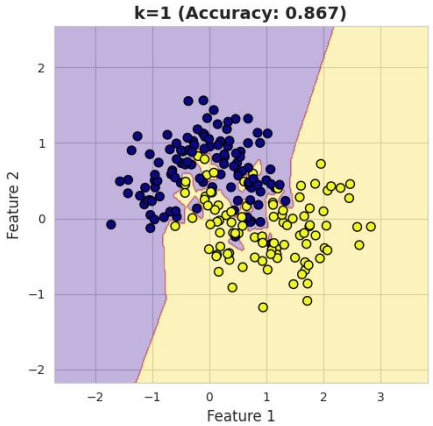
We will now discuss another method for classification and regression: kNN.

For a test datapoint, take its *k* nearest neighbors. Predict the class frequency among the neighbors:

$$p(y = c|\boldsymbol{x}, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_n = c)$$

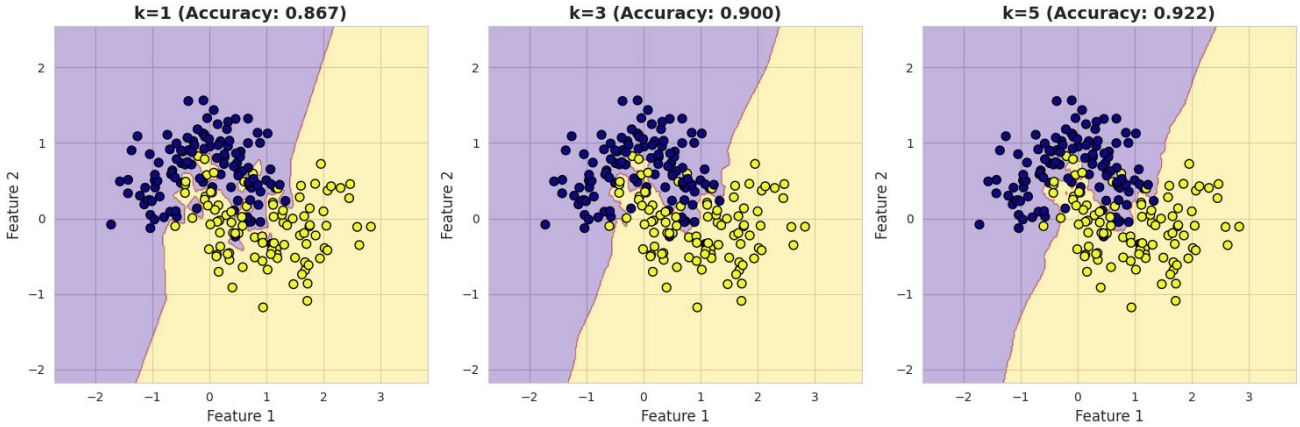
5 neighbors, p(●) = ⅘, p(●) = ⅕

k-Nearest Neighbors



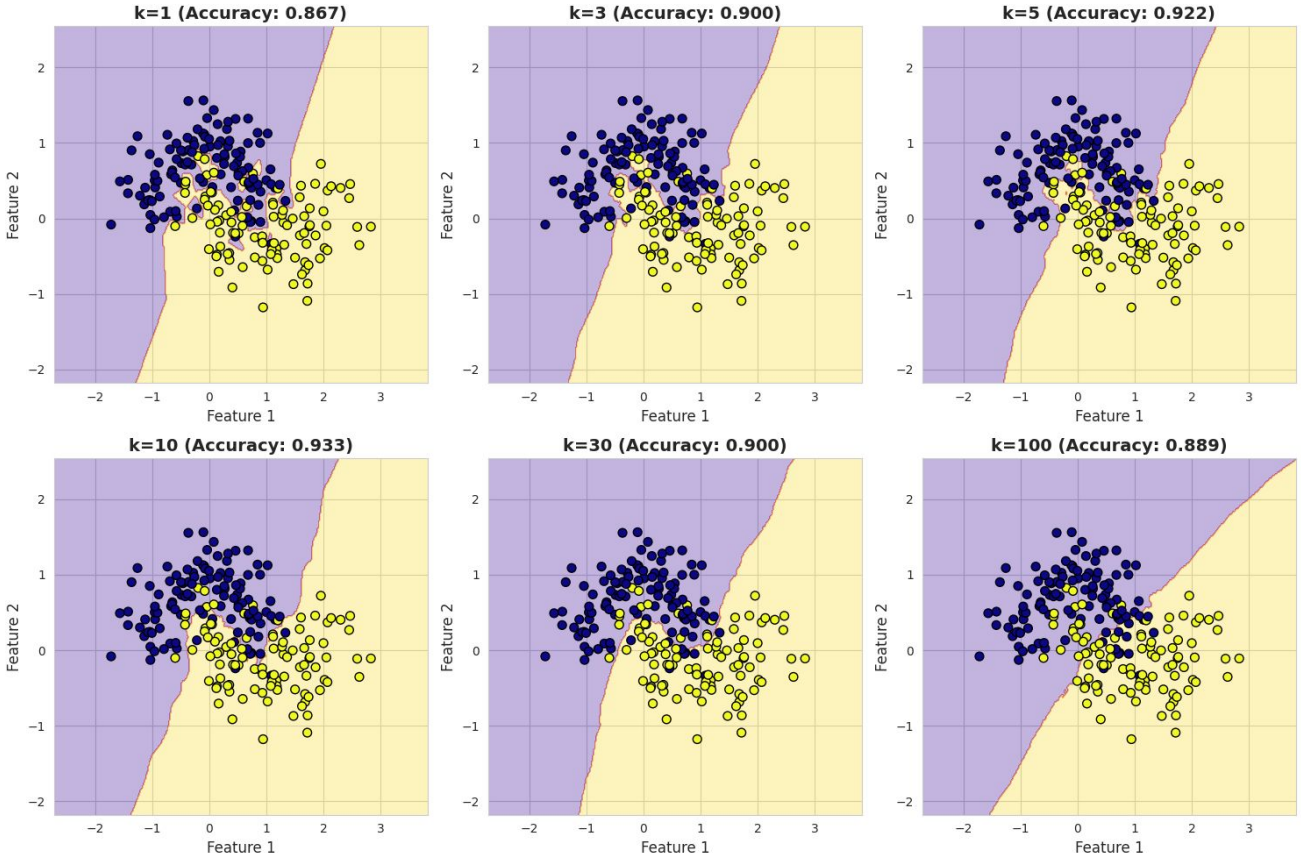
- Small $k \Rightarrow$ complicated decision boundary, fits training data too much

k-Nearest Neighbors



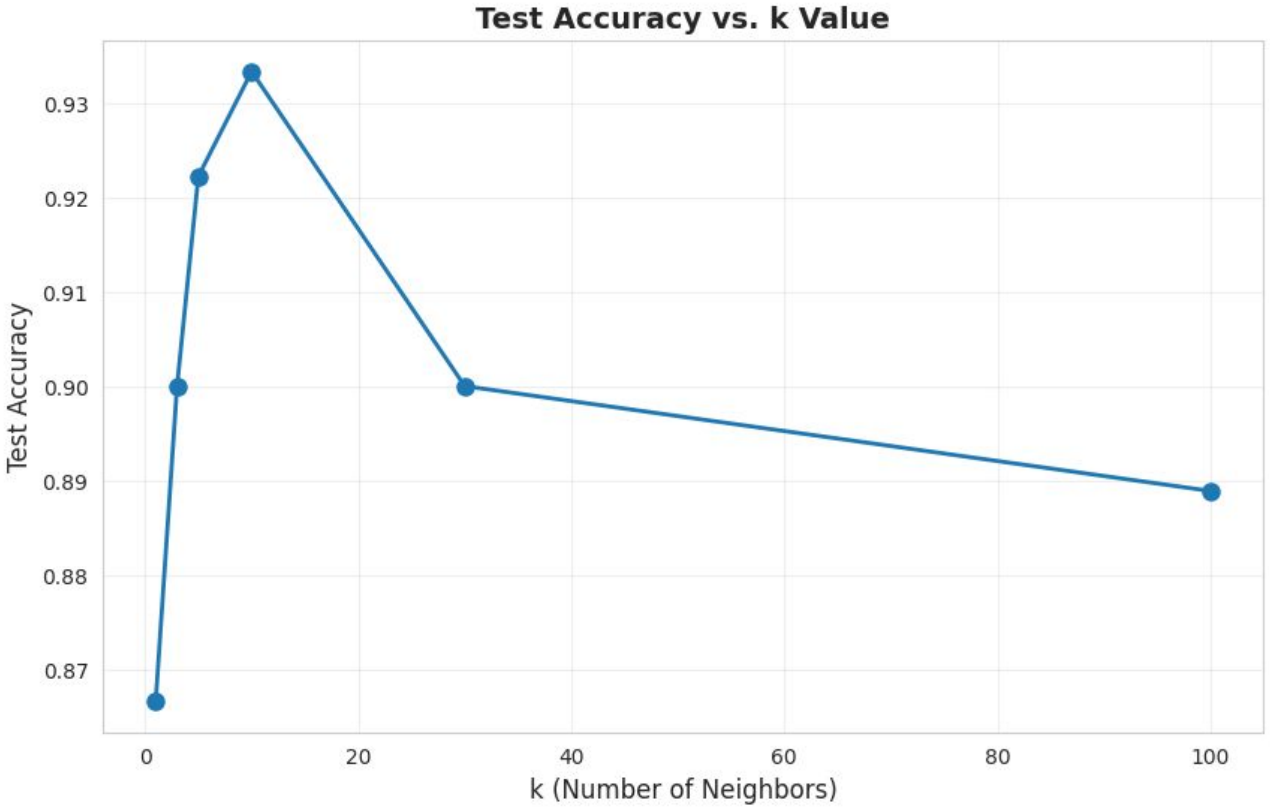
- Small $k \Rightarrow$ complicated decision boundary, fits training data too much

k-Nearest Neighbors



- Small $k \Rightarrow$ complicated decision boundary, fits training data too much
- Very large $k \Rightarrow$ underfitting, overly simple decision boundary

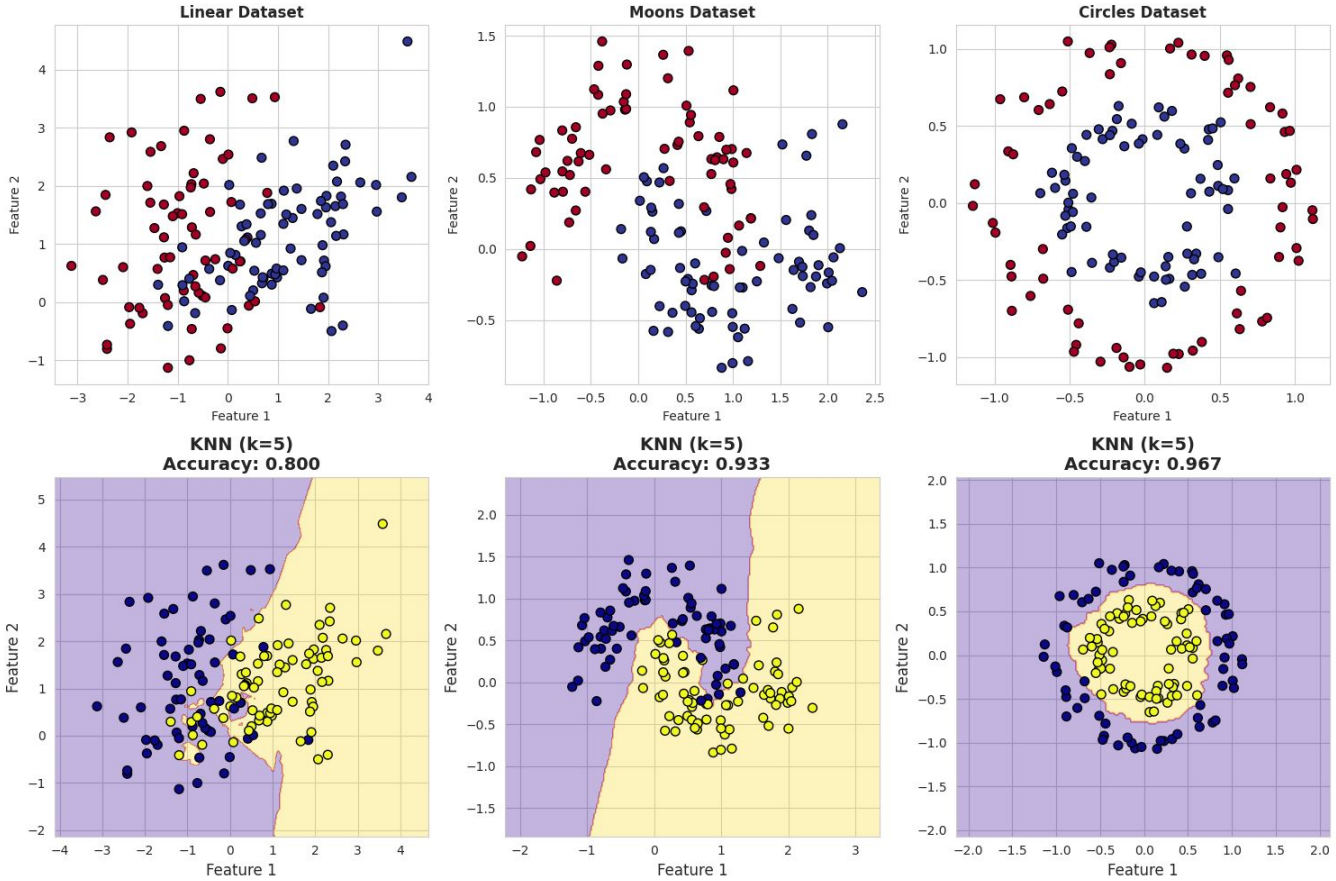
k-Nearest Neighbors



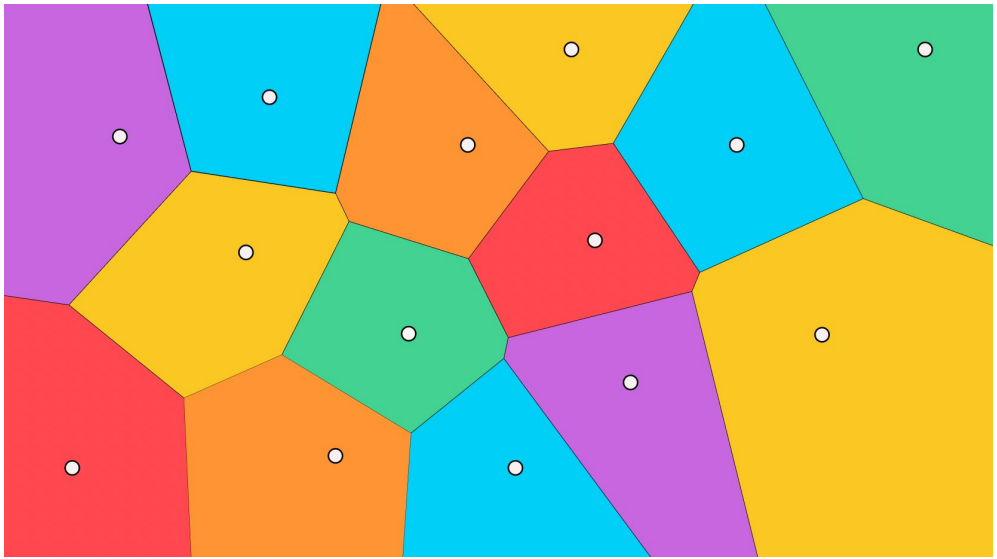
- Small $k \Rightarrow$ complicated decision boundary, fits training data too much
- Very large $k \Rightarrow$ underfitting, overly simple decision boundary

k-Nearest Neighbors

- kNN is flexible and can fit non-linear data well
- It is prone to overfitting



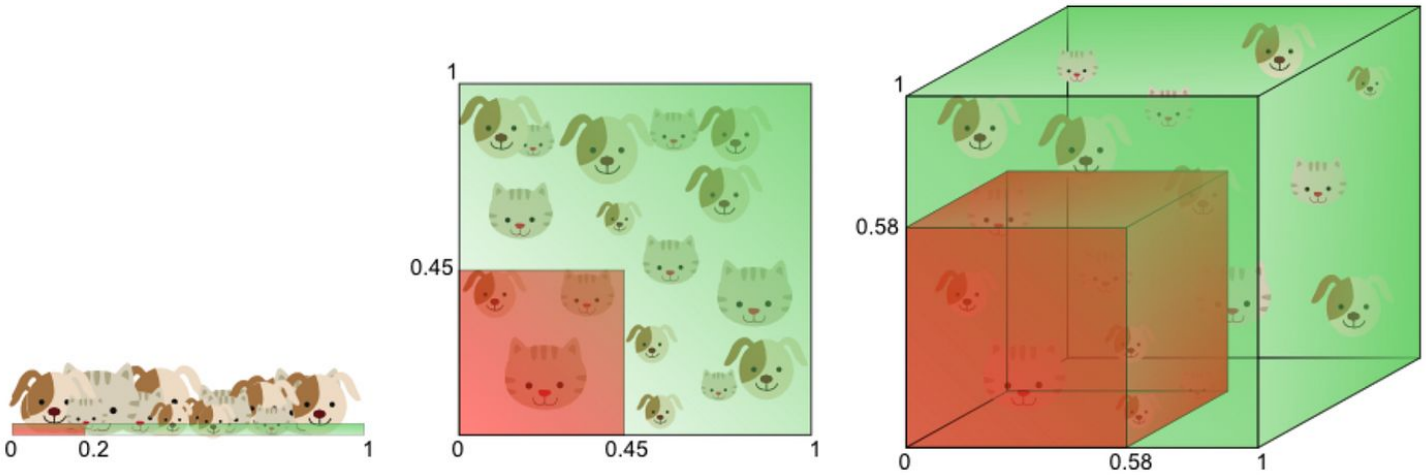
k-Nearest Neighbors



The kNN decision boundaries are known as Voronoi tessellations.

The image is for *k=1*.

Curse of Dimensionality



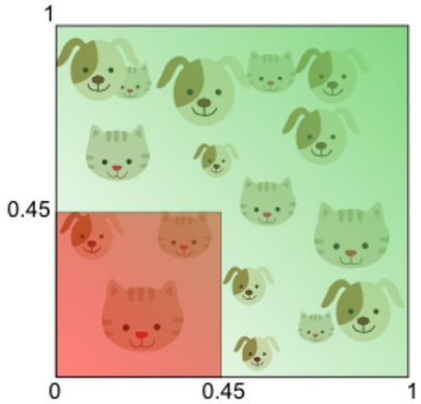
Curse of dimensionality: data becomes more sparse in high dimensions.

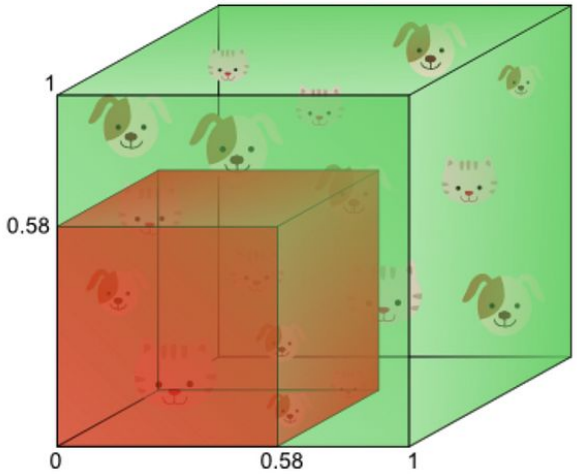
The nearest neighbors can be very far, not very predictive.

https://medium.com/@gokcenazakyol/what-is-curse-of-dimensionality-machine-learning-2-739131962faf

Curse of Dimensionality

Grid of 10 points along each dimension ⇒ 10^d points





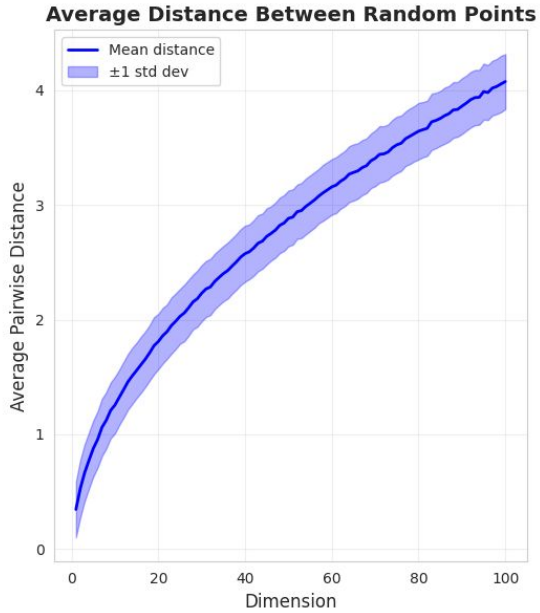
Curse of dimensionality: data becomes more sparse in high dimensions.

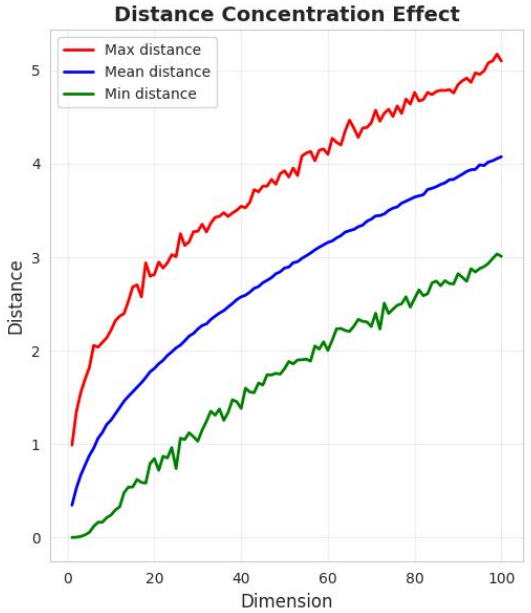
The nearest neighbors can be very far, not very predictive.

https://medium.com/@gokcenazakyol/what-is-curse-of-dimensionality-machine-learning-2-739131962faf

Curse of Dimensionality

The distances between points grow a lot in high dimensions!





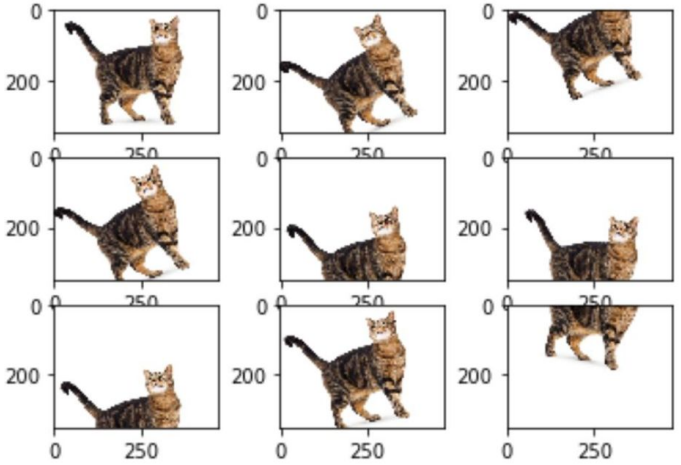
We sample 1000 datapoints uniformly in (0, 1) along d dimensions:

```
np.random.uniform(0, 1, size=(n_samples, d))
```

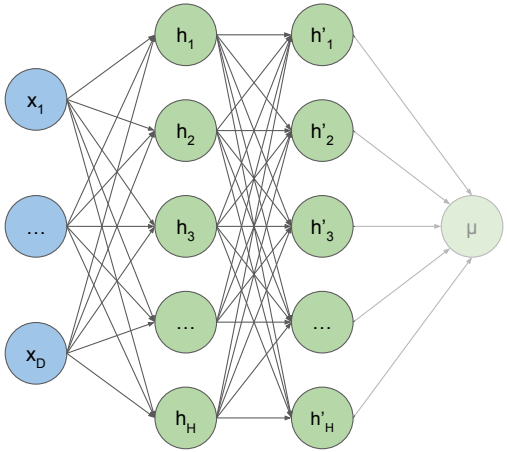
Curse of Dimensionality

Defining distance in structured high-dimensional data is non-trivial, e.g.

- Distance in pixel space between images is a bad measure
- Defining a *semantic* distance metric for images is very difficult



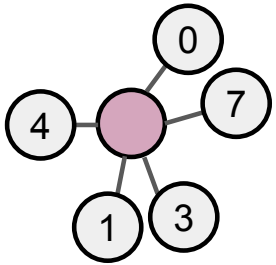
Neural Networks + kNN



- We can use a neural network (MLP, CNN, RNN, Transformer) to map an input to a latent embedding
- Then, we can do kNN using distances in the latent embedding space
- This way we can do kNN on images, text, graphs, …
- Neural networks often learn good distance metrics

kNN Regression

For regression we average the labels across the *k* neighbors



5 neighbors, prediction is (0 + 1 + 3 + 4 + 7) / 5 = 3

k-Nearest Neighbors

- Two main hyper-parameters:
  - Number of neighbors *k*
  - Distance metric
- Non-parametric method
  - Infinitely flexible: can adapt to any number of datapoints
- Naturally handles regression, multiclass classification, …
- No training needed
- Very simple
- Interpretable: can provide an explanation for any prediction
- Only does interpolation, no extrapolation
- In practice, not as good for high-dimensional data
- Can be slow

k-Nearest Neighbors

Use if:

- Small to medium datasets (<100k)
- Low-dimensional (<20 features)
- Non-linear decision boundaries
- Need an interpretable baseline

Don’t use if:

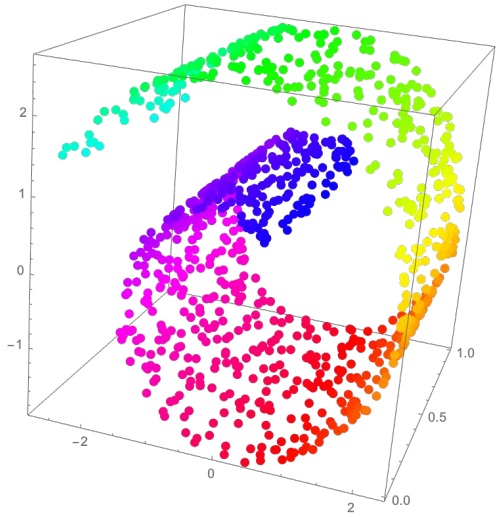
- Large datasets (slow predictions)
- High-dimensional data (curse of dimensionality)
- Real-time predictions required (too slow)
- Many irrelevant features
- Severe class imbalance


Dimensionality
Reduction

Dimensionality Reduction

High-dimensional data is hard to work with:

- Cannot visualize
- Distances are counter-intuitive
- Curse of dimensionality



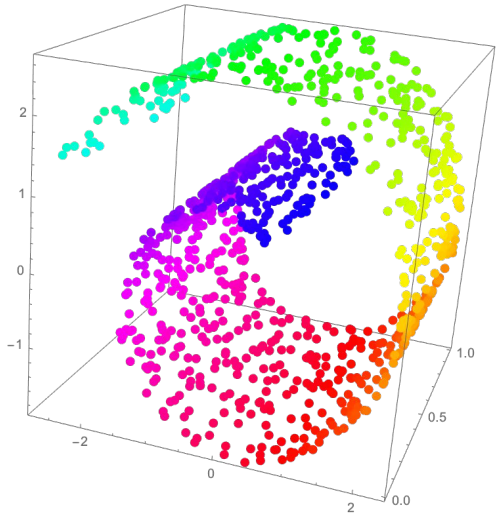
Manifold assumption:

- Data lives on a lower-dimensional manifold, doesn’t span full space
- If we can discover the manifold, we can reduce dimension

Dimensionality Reduction

High-dimensional data is hard to work with:

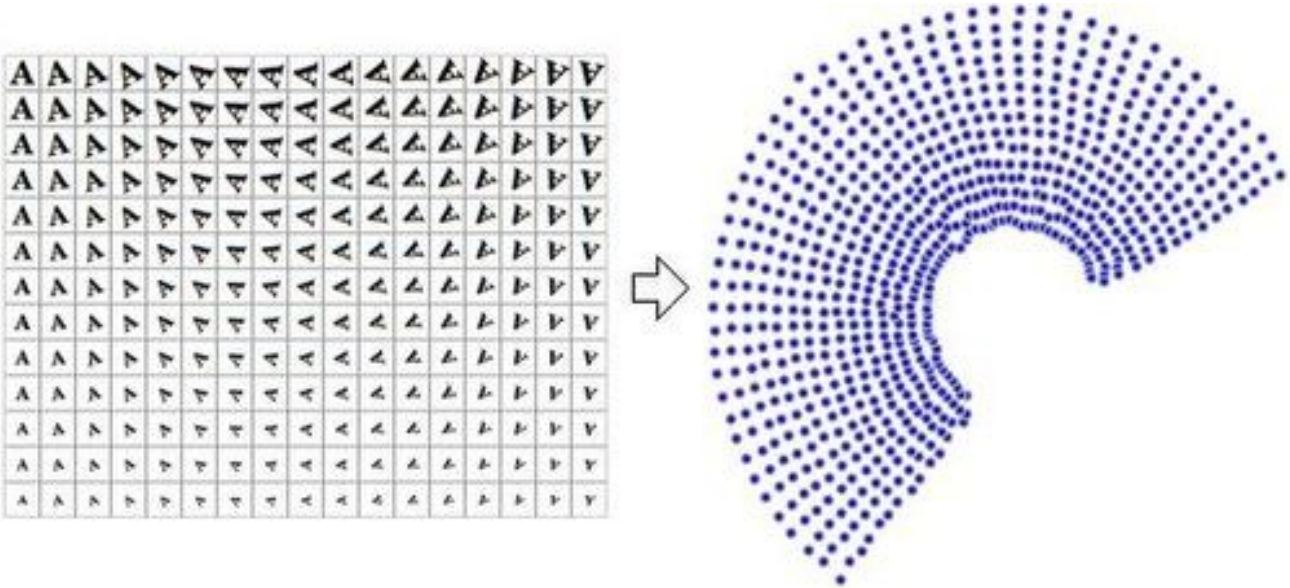
- Cannot visualize
- Distances are counter-intuitive
- Curse of dimensionality



Manifold assumption:

- Data lives on a lower-dimensional manifold, doesn’t span full space
- If we can discover the manifold, we can reduce dimension

Dimensionality Reduction: Example



- Images of the same letter “A”, differ in size and rotation
- Original images might be 28 * 28 dimensional
- Can represent with just two natural coordinates

Representation Learning

A lot of machine learning can be viewed as learning a representation for data:

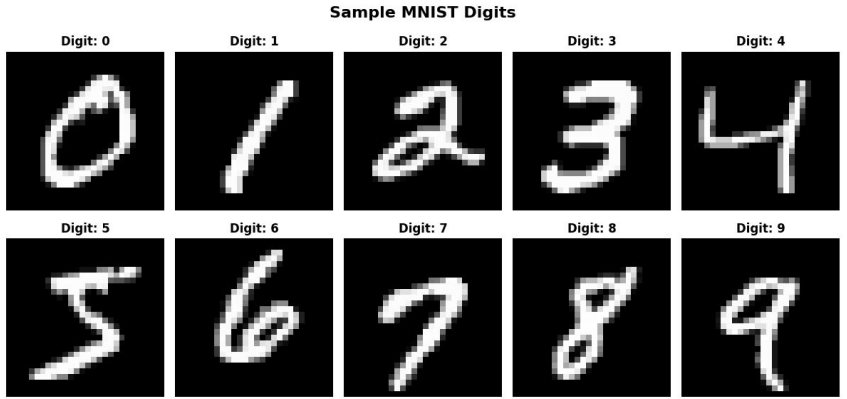
- Image classification models are doing a *linear* classification on top of features extracted by multiple layers of CNN
- LLMs are doing *linear* classification based on transformer representations

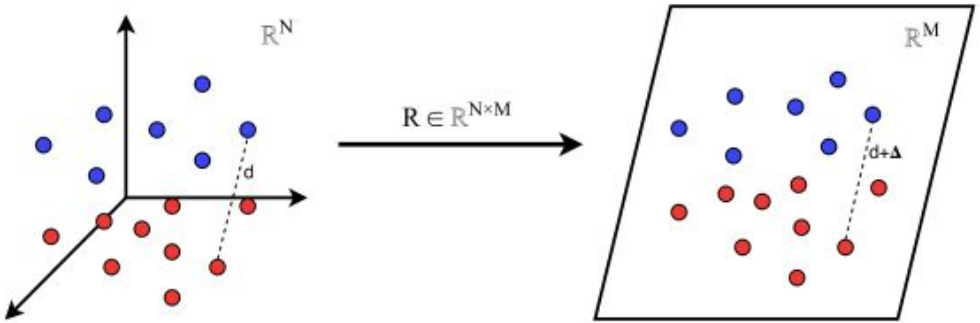
Dimensionality Reduction

Remember MNIST

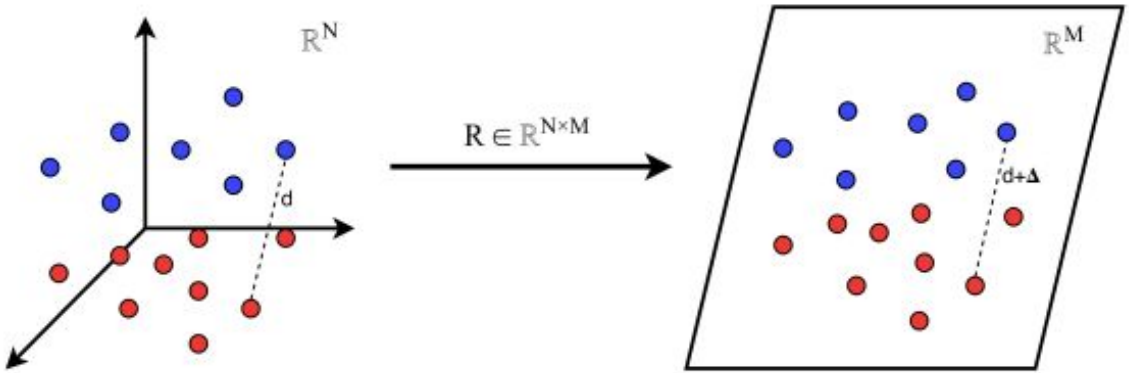
- 28 x 28 pixels, images of digits
- 10 classes

We will try to reduce the dimension to 2!





Random projection

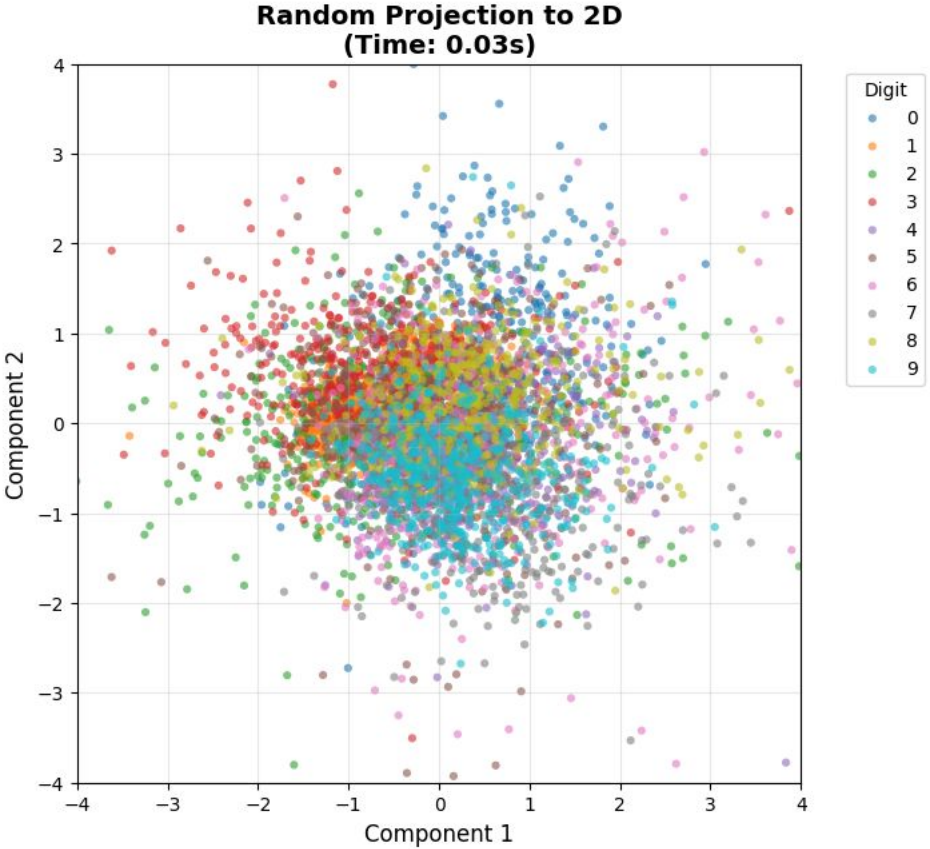


Random projection

- Choose an arbitrary 2d plane
- Project data on this plane

[demo-tsne.ipynb]

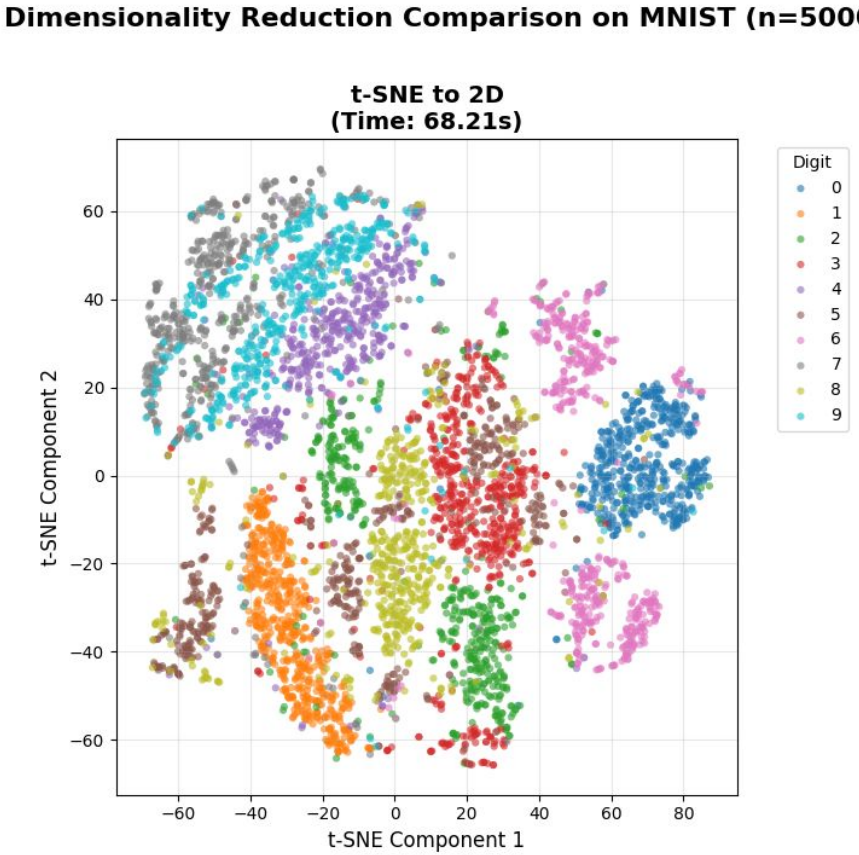
Random projection



Random projection does not work well for us. All classes are mixed up, not much structure.

You can still see some clustering.

T-SNE



Stochastic Neighbor Embedding

Geoffrey Hinton and Sam Roweis
Department of Computer Science, University of Toronto
10 King's College Road, Toronto, M5S 3G5 Canada
{*hinton,roweis*}*@cs.toronto.edu*

T-SNE is a non-linear dimensionality reduction method that works quite well!