Uncertainties in ML

University of Victoria - PHYS-555

ML Challenges: Myths or Realities?

Often machine learning methods are critiqued:

1. they do not incorporate uncertainties
2. they do not deal with heteroscedastic data
3. they are only capable of giving point estimates
4. they do not deal with missing data
5. their predictions are not interpretable
6. they do not generalize

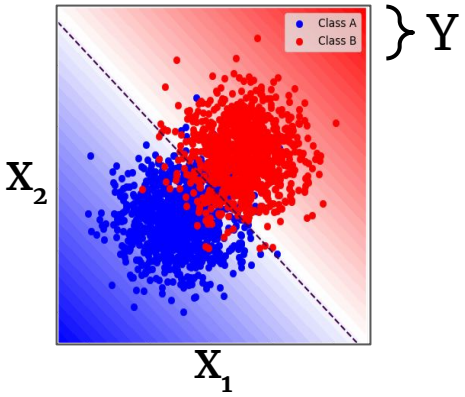
Which ones do you think are correct?

What do we mean by Uncertainty?

Return an uncertainty over predictions rather than a single prediction. Often a distribution.

- ***Classification***: Output label along with its confidence.
- ***Regression***: Output mean along with its variance.

Good uncertainty estimates quantify ***when we can trust the model's predictions***.



$$p(\mathbf{y}|\mathbf{x})$$

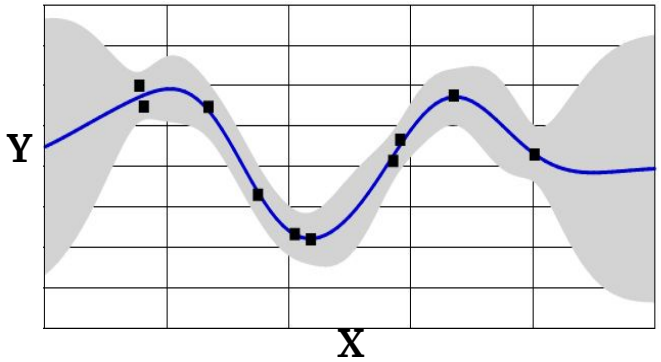


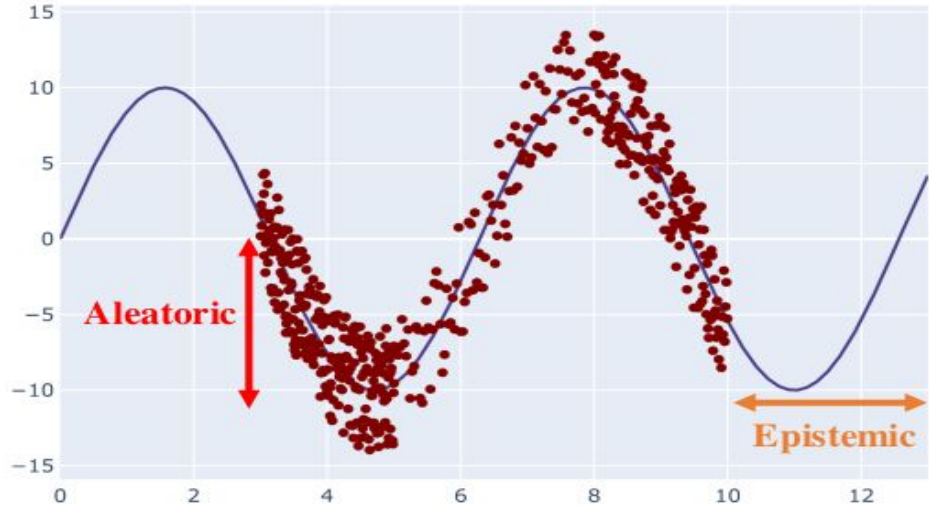
Image credit: Eric Nalisnick

Predictive, aleatoric and epistemic uncertainties

Objective: Predictive uncertainty

Data (Aleatoric) uncertainty

Epistemic (Model) uncertainty



A review of uncertainty quantification in deep learning: Techniques, applications and challenges, M. Abdar et al., https://doi.org/10.1016/j.inffus.2021.05.00

Data Uncertainty

Sources of uncertainty: *Data uncertainty*

- Labeling noise (ex: human disagreement)

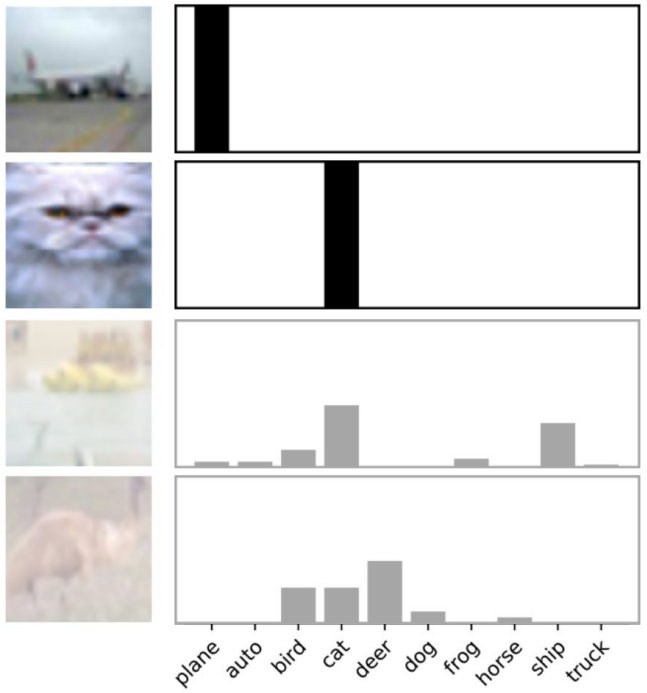


Image source: Battleday et al. 2019 “Improving machine classification using human uncertainty measurements”

Sources of uncertainty: *Data uncertainty*

- Labeling noise (ex: human disagreement)

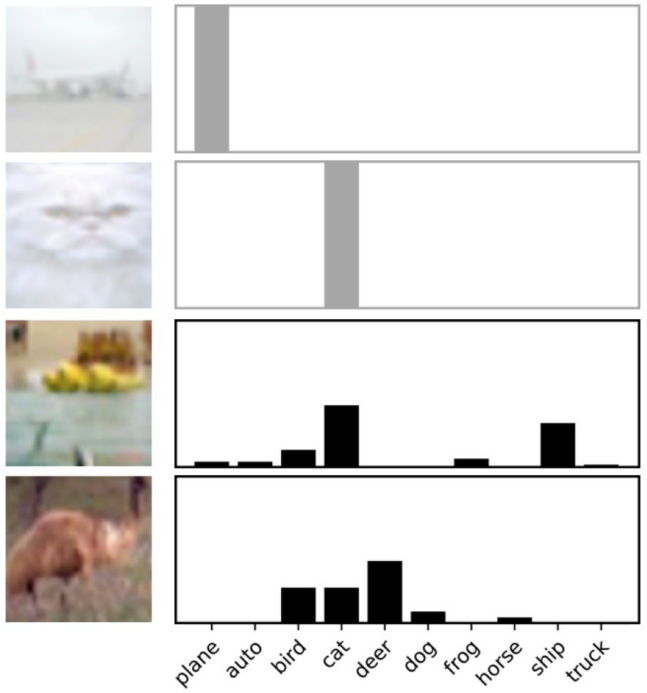


Image source: Battleday et al. 2019 “Improving machine classification using human uncertainty measurements”

Sources of uncertainty: *Data uncertainty*

- Labeling noise (ex: human disagreement)
- Measurement noise (ex: imprecise tools)
- *Missing* data (ex: partially observed features, unobserved confounders)
- Also known as ***aleatoric uncertainty***
- Data uncertainty is "**irreducible***"
  - Persists even in the limit of infinite data
  - *Could be reduced with additional features/views

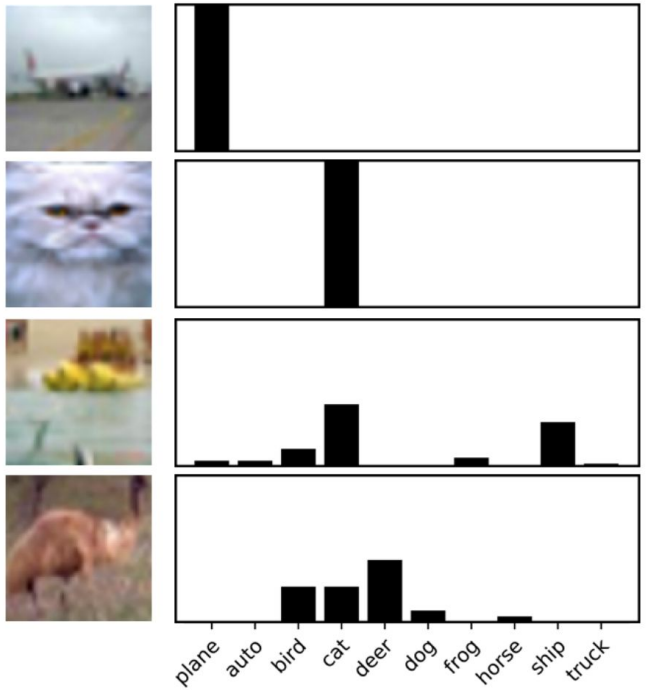
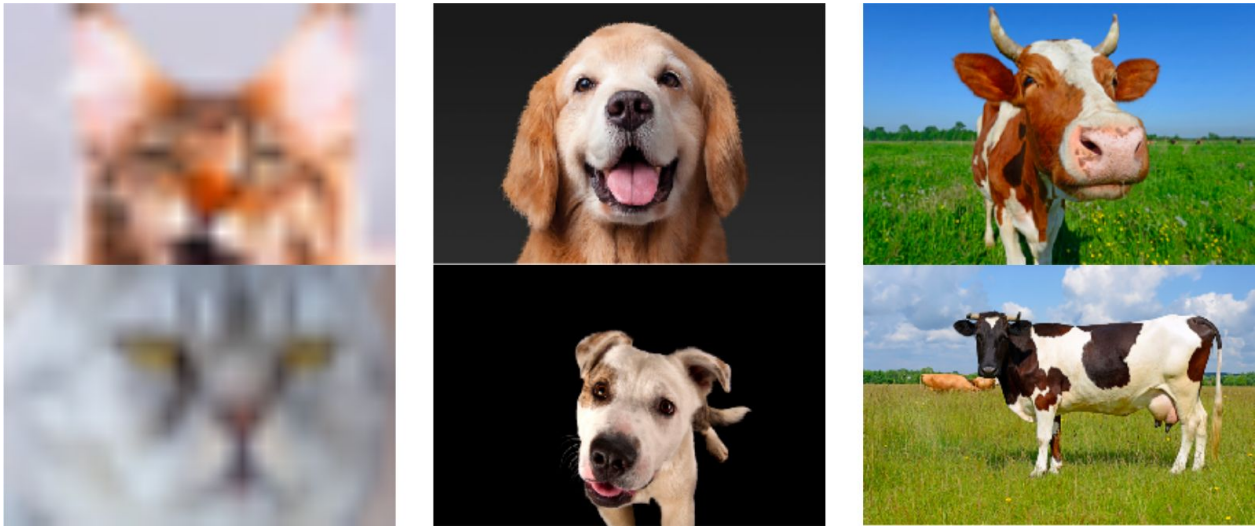


Image source: Battleday et al. 2019 "Improving machine classification using human uncertainty measurements"

Data (Aleatoric) Uncertainty

We have three different types of images to classify, cat, dog, and cow, where only cat images are noisy.



Data (Aleatoric) Uncertainty

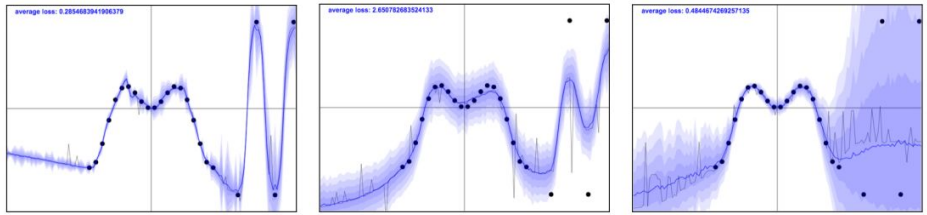
Aleatoric uncertainty captures noise inherent in the observations.

- For example, sensor noise or motion noise result in uncertainty.
- This uncertainty cannot be reduced with more data.
- However, aleatoric could be reduced with better measurements.

In physics, we sometimes call it: statistical or random uncertainty.

Aleatoric uncertainty can further be categorized into homoscedastic and heteroscedastic uncertainties:

- Homoscedastic uncertainty relates to the uncertainty that a particular task might cause. It stays constant for different inputs.
- Heteroscedastic uncertainty depends on the inputs to the model, with some inputs potentially having more noisy outputs than others.



(a) Homoscedastic model with small observation noise.

(b) Homoscedastic model with large observation noise.

(c) Heteroscedastic model with data-dependent observation noise.

Regression with Uncertainty

Consider training data $(\mathbf{x}, y) \sim P(X, Y)$, with

- $\mathbf{x} \in \mathbb{R}^p$,
- $y \in \mathbb{R}$.

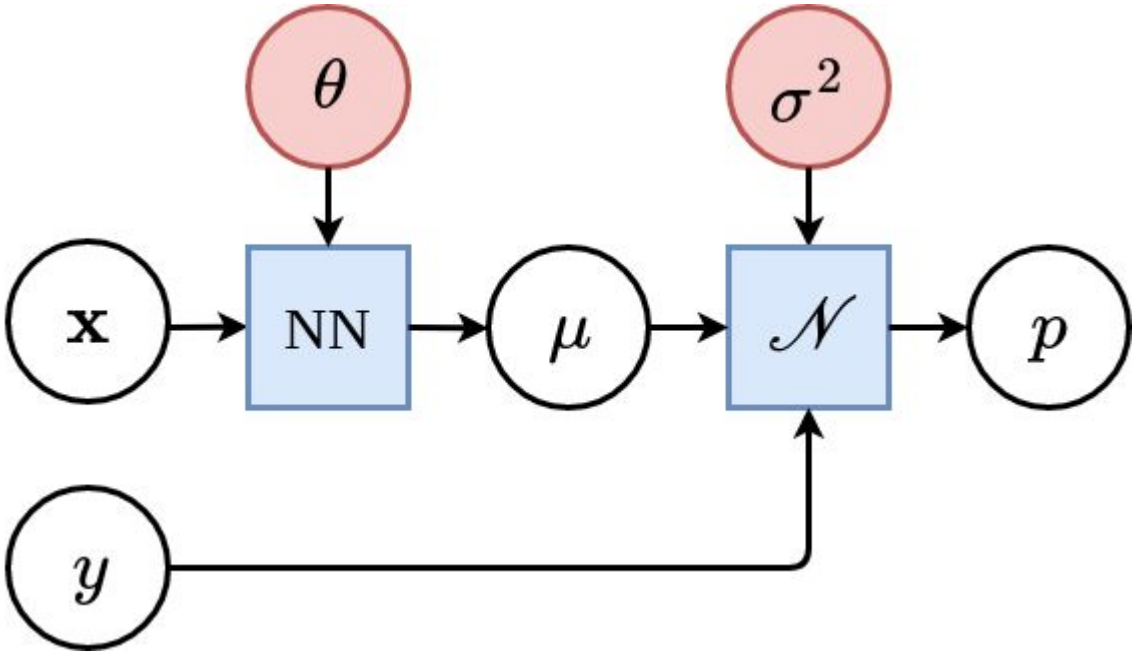
We model aleatoric uncertainty in the output by modelling the conditional distribution as a Normal distribution,

$$p(y|\mathbf{x}) = \mathcal{N}(y; \mu(\mathbf{x}), \sigma^2(\mathbf{x})),$$

where $\mu(x)$ and $\sigma^2(\mathbf{x})$ are parametric functions to be learned, such as neural networks.

In particular, we do not wish to learn a function $\hat{y} = f(\mathbf{x})$ that would only produce point estimates.

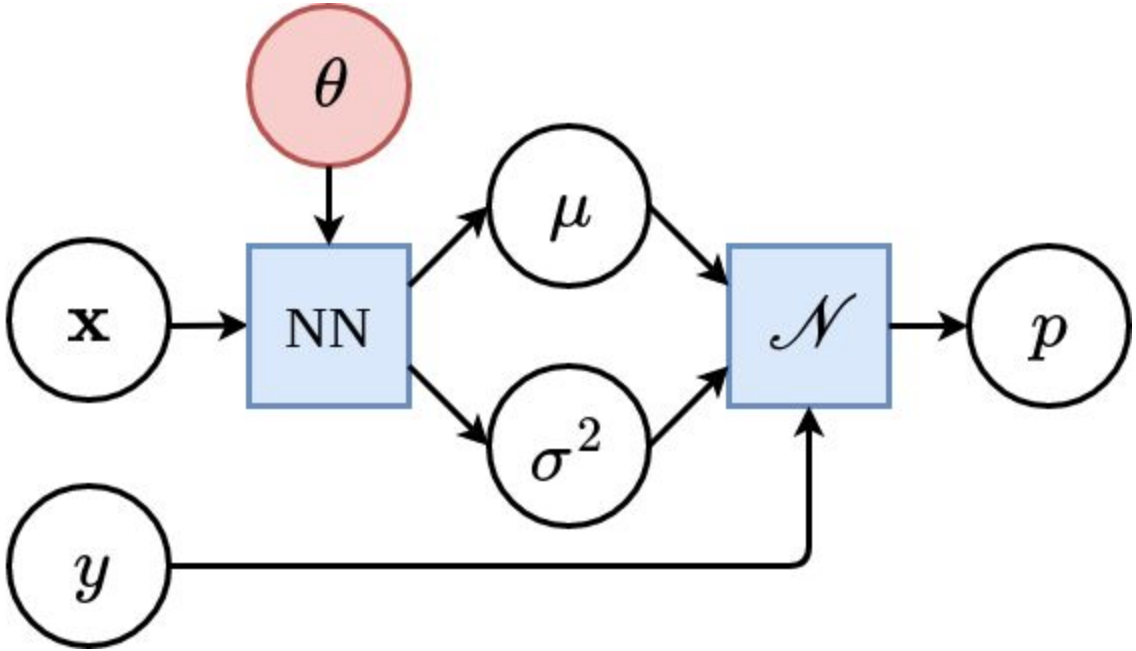
Homoscedastic Aleatoric Uncertainty



We have,

$$
\begin{aligned}
&\arg\max_{\theta,\sigma^2} p(\mathbf{d}|\theta,\sigma^2) \\
&= \arg\max_{\theta,\sigma^2} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} p(y_i|\mathbf{x}_i,\theta,\sigma^2) \\
&= \arg\max_{\theta,\sigma^2} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i-\mu(\mathbf{x}_i))^2}{2\sigma^2}\right) \\
&= \arg\min_{\theta,\sigma^2} \sum_{\mathbf{x}_i,y_i\in\mathbf{d}} \frac{(y_i-\mu(\mathbf{x}_i))^2}{2\sigma^2} + \log(\sigma) + C
\end{aligned}
$$

Heteroscedastic Aleatoric Uncertainty



Same as for the homoscedastic case, except that that σ^2 is now a function of $\mathbf{x}_i$:

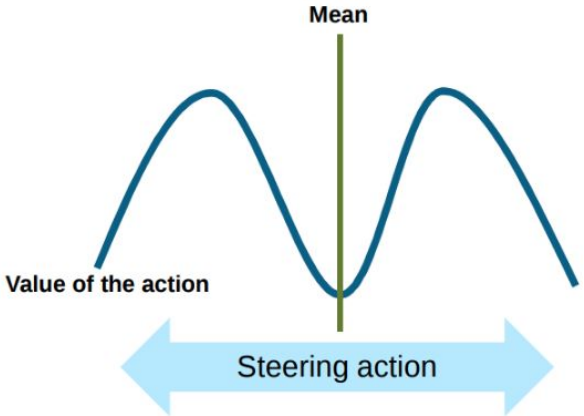
$$
\begin{aligned}
&\arg\max_{\theta} p(\mathbf{d}|\theta) \\
&= \arg\max_{\theta} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} p(y_i|\mathbf{x}_i, \theta) \\
&= \arg\max_{\theta} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} \frac{1}{\sqrt{2\pi}\sigma(\mathbf{x}_i)} \exp\left(-\frac{(y_i - \mu(\mathbf{x}_i))^2}{2\sigma^2(\mathbf{x}_i)}\right) \\
&= \arg\min_{\theta} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \frac{(y_i - \mu(\mathbf{x}_i))^2}{2\sigma^2(\mathbf{x}_i)} + \log(\sigma(\mathbf{x}_i)) + C
\end{aligned}
$$

Posterior Multimodality

Modelling $p(y|\mathbf{x})$ as a unimodal Gaussian is not always a good idea!

(and it would be even worse to have only point estimates for y!)

https://commons.wikimedia.org/wiki/File:Newport_Whitepit_Lane_pot_hole.JPG

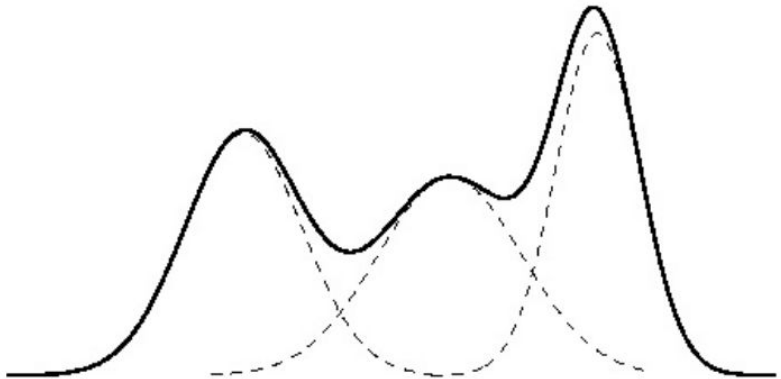


Gaussian Mixture Model

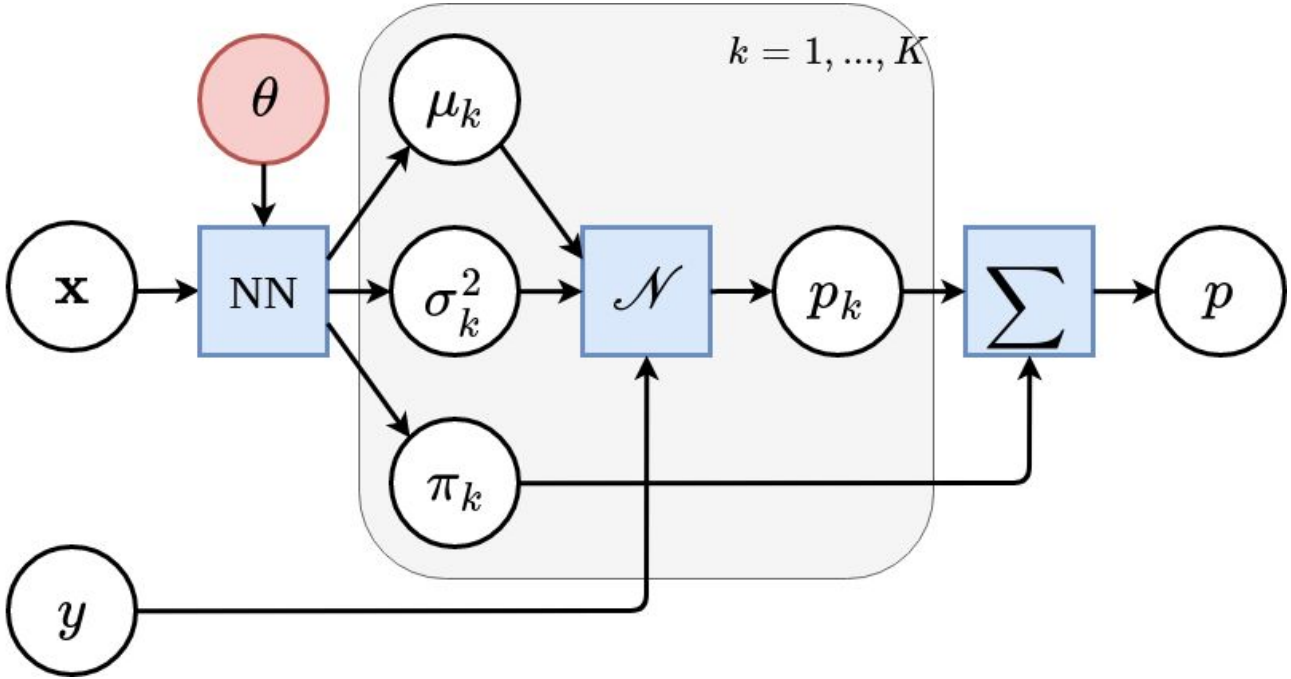
A Gaussian mixture model (GMM) defines instead $p(y|\mathbf{x})$ as a mixture of K Gaussian components,

$$p(y|\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(y; \mu_k, \sigma_k^2),$$

where $0 \leq \pi_k \leq 1$ for all k and $\sum_{k=1}^{K} \pi_k = 1$.



Mixture Density Network

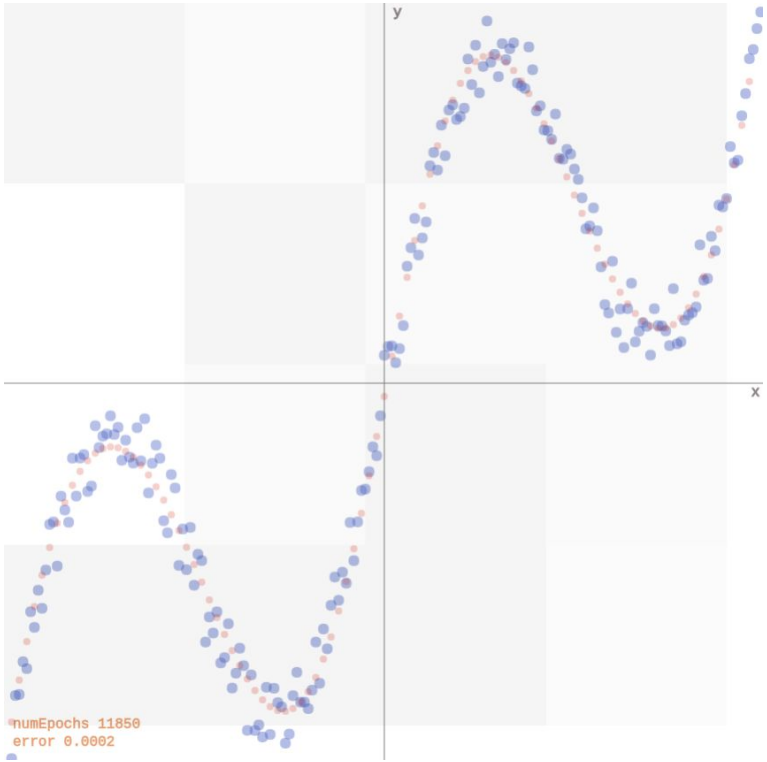


MDN Example

Let us consider training data generated randomly as

$$y_i = \mathbf{x}_i + 0.3 \sin(4\pi \mathbf{x}_i) + \epsilon_i$$

with $\epsilon_i \sim \mathcal{N}$.



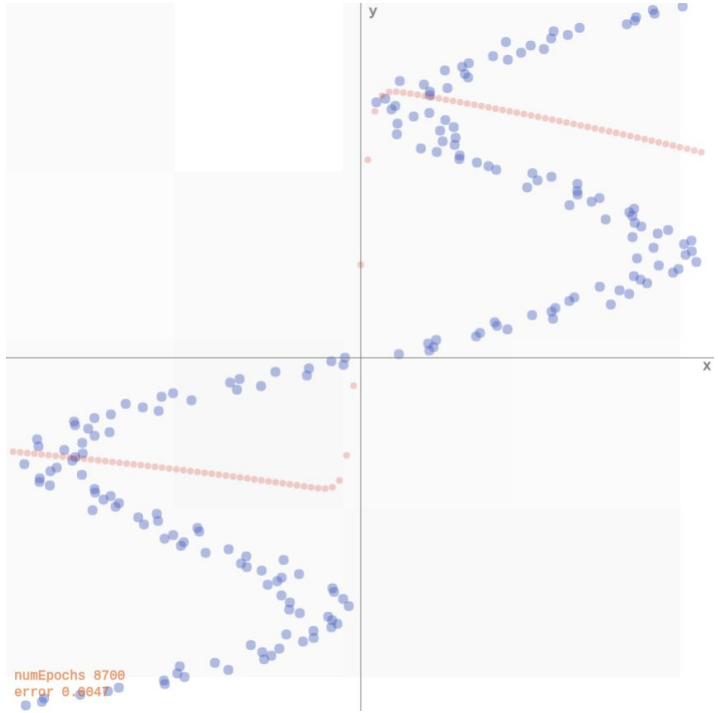
Demo: http://otoro.net/ml/mixture/index.html

MDN Example

Let us consider training data generated randomly as

$$y_i = \mathbf{x}_i + 0.3 \sin(4\pi \mathbf{x}_i) + \epsilon_i$$

with $\epsilon_i \sim \mathcal{N}$.



Flip x and y issue!

http://otoro.net/ml/mixture/inverse.html

MDN Example

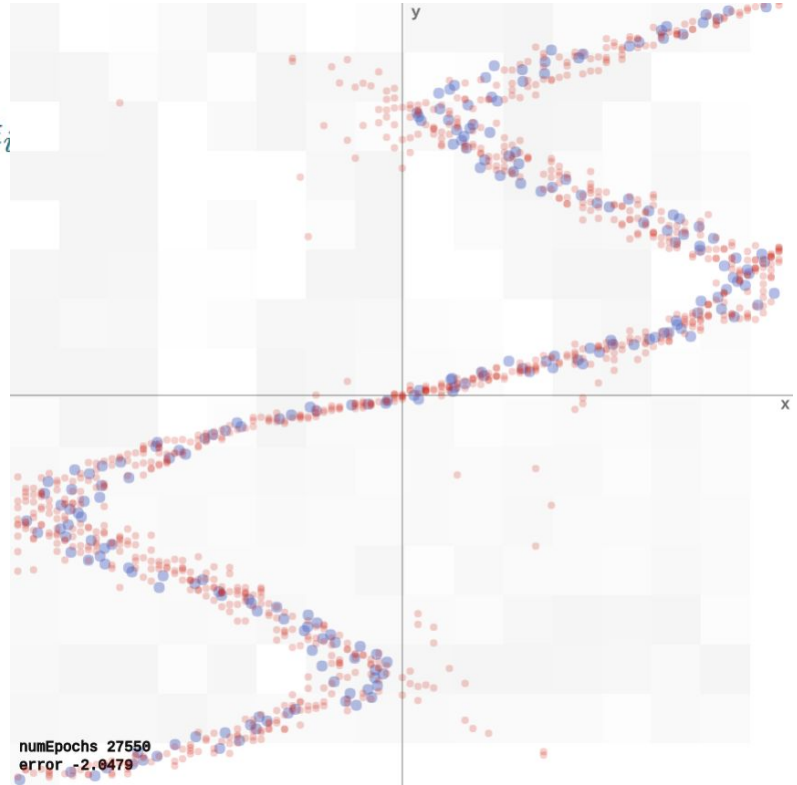
Let us consider training data generated randomly as

$$y_i = \mathbf{x}_i + 0.3\sin(4\pi\mathbf{x}_i) + \epsilon_i$$

with $\epsilon_i \sim \mathcal{N}$.

But MDN fits OK:

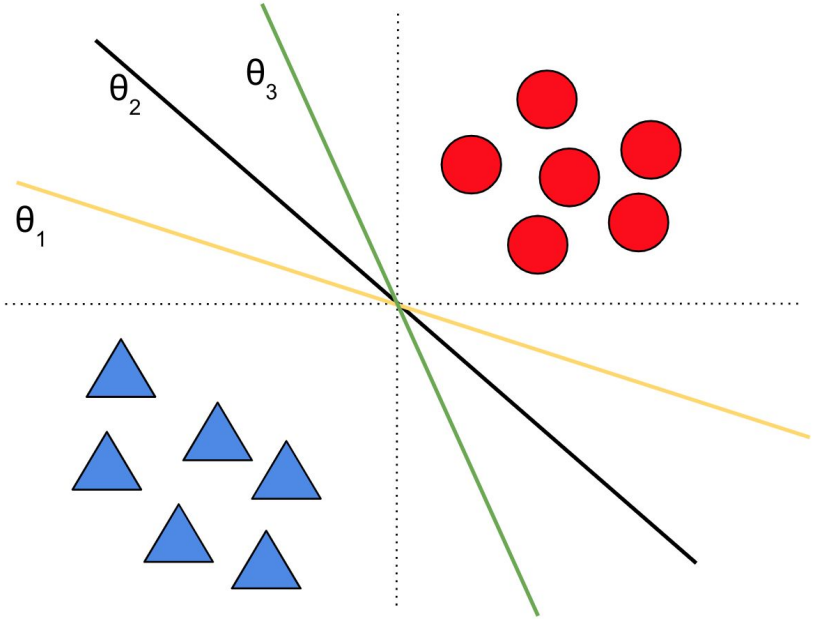
http://otoro.net/ml/mixture/mixture.html



Model Uncertainty

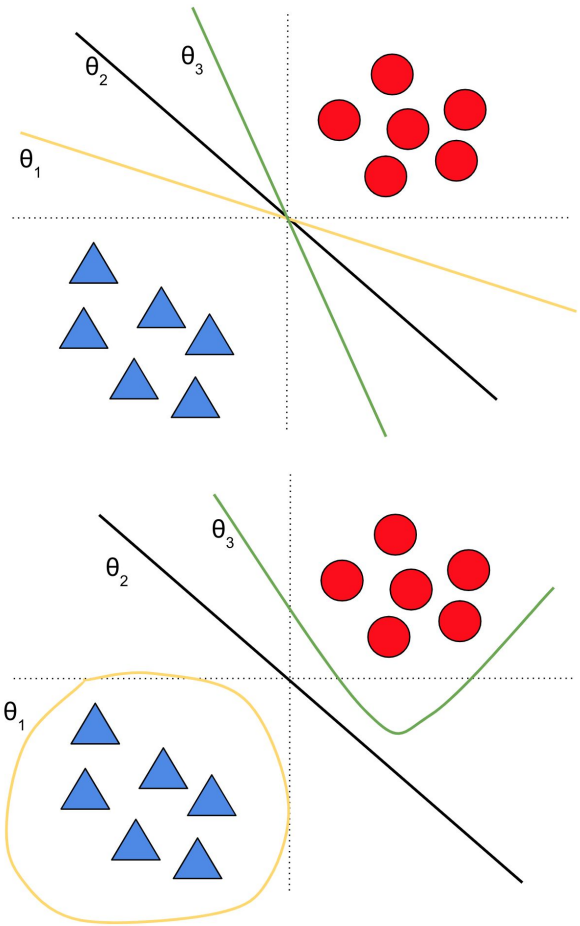
Sources of uncertainty: *Model uncertainty*

- Many models can fit the training data well
- Also known as ***epistemic uncertainty***
- Model uncertainty is "**reducible**"
  - Vanishes in the limit of infinite data (subject to model identifiability)



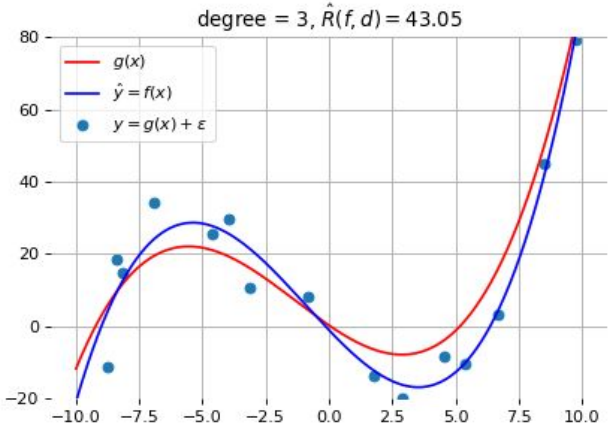
Sources of uncertainty: *Model uncertainty*

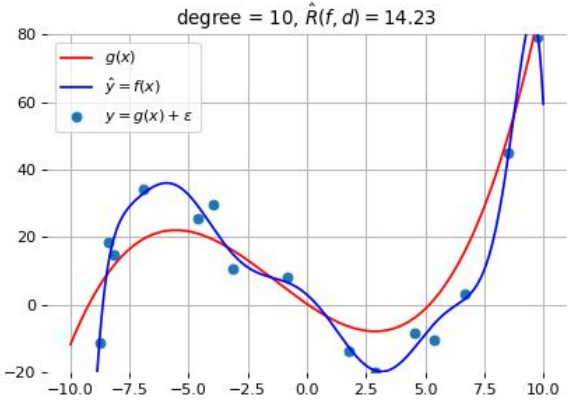
- Many models can fit the training data well
- Also known as ***epistemic uncertainty***
- Model uncertainty is "**reducible**"
  - Vanishes in the limit of infinite data (subject to model identifiability)
- Models can be from same hypotheses class (e.g. linear classifiers in top figure) or belong to different hypotheses classes (bottom figure).

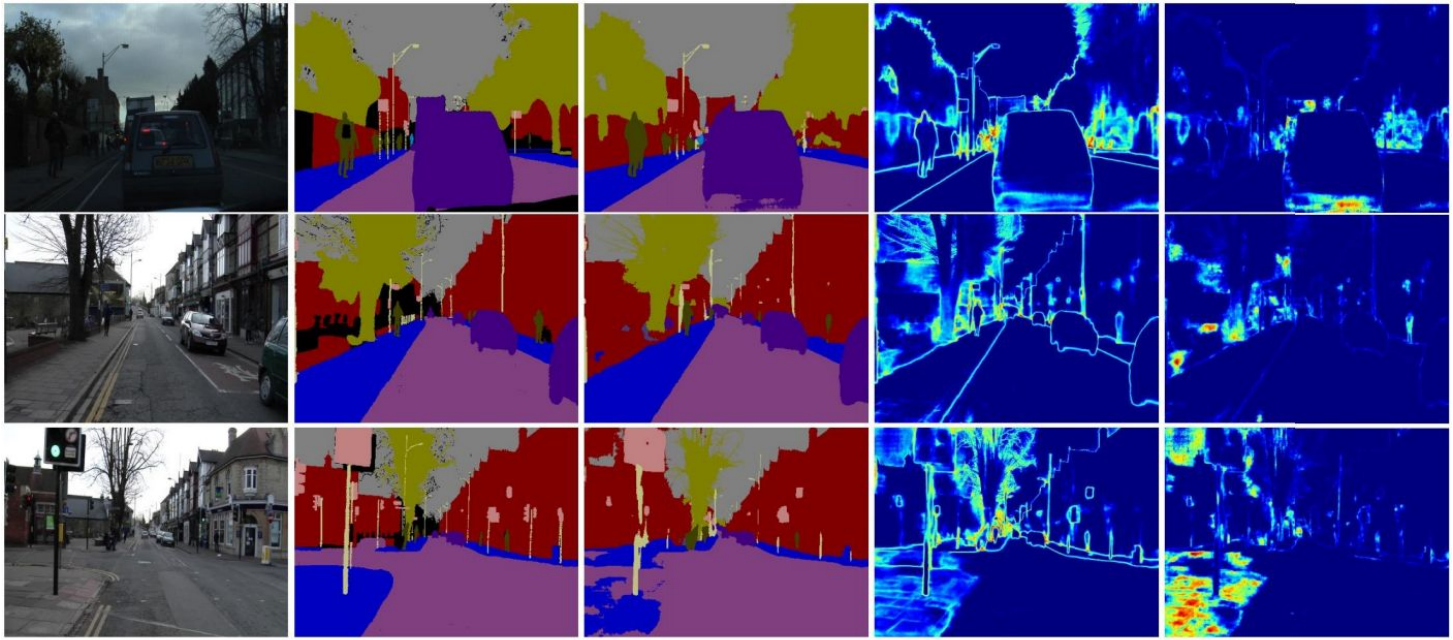


Model (Epistemic) Uncertainty

What is the best model parameters that best explain a given dataset? What model structure should we use?







(a) Input Image (b) Ground Truth (c) Semantic Segmentation (d) Aleatoric Uncertainty (e) Epistemic Uncertainty

[Kendall and Gal (2017)]

*Our model exhibits in (d) increased **aleatoric uncertainty on object boundaries and for objects far from the camera. Epistemic uncertainty accounts for our ignorance about which model generated our collected data**. In (e) our model exhibits increased epistemic uncertainty for semantically and visually challenging pixels. The bottom row shows a failure case of the segmentation model when the model fails to segment the footpath due to increased epistemic uncertainty, but not aleatoric uncertainty."*

Model (Epistemic) Uncertainty

Epistemic uncertainty accounts for uncertainty in the model parameters.

- It captures our ignorance about which model generated the collected data.
- It can be explained away given enough data (why?).

Out of Distribution Uncertainties

Out-of-distribution Uncertainties

Let us consider a neural network model trained with several pictures of dog breeds.

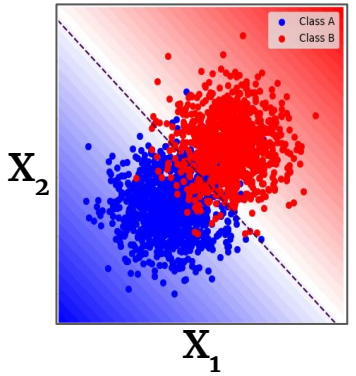
- We ask the model to decide on a dog breed using a photo of a cat.
- What would you want the model to do?

Out-of-Distribution Robustness?

I.I.D. $p_{TEST}(y,x) = p_{TRAIN}(y,x)$

(*Independent and Identically Distributed*)

O.O.D. $p_{TEST}(y,x) \neq p_{TRAIN}(y,x)$



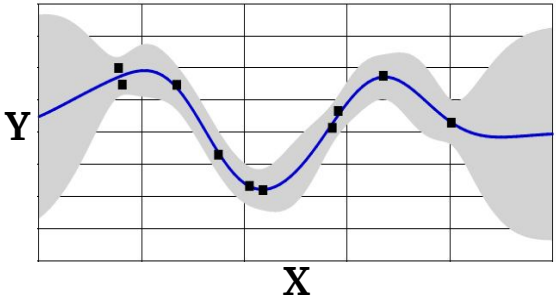


Image credit: Eric Nalisnick

Out-of-Distribution Robustness?

I.I.D. $p_{TEST}(y,x) = p_{TRAIN}(y,x)$

O.O.D. $p_{TEST}(y,x) \neq p_{TRAIN}(y,x)$

Examples of dataset shift:

- ***Covariate shift.*** Distribution of features p(x) changes and p(y|x) is fixed.
- ***Open-set recognition.*** New classes may appear at test time.
- ***Subpopulation shift.*** Frequencies of subpopulation changes
- ***Label Shift.*** Distribution of labels p(y) changes and p(x|y) is fixed

ImageNet-C: Varying Intensity for Dataset Shift

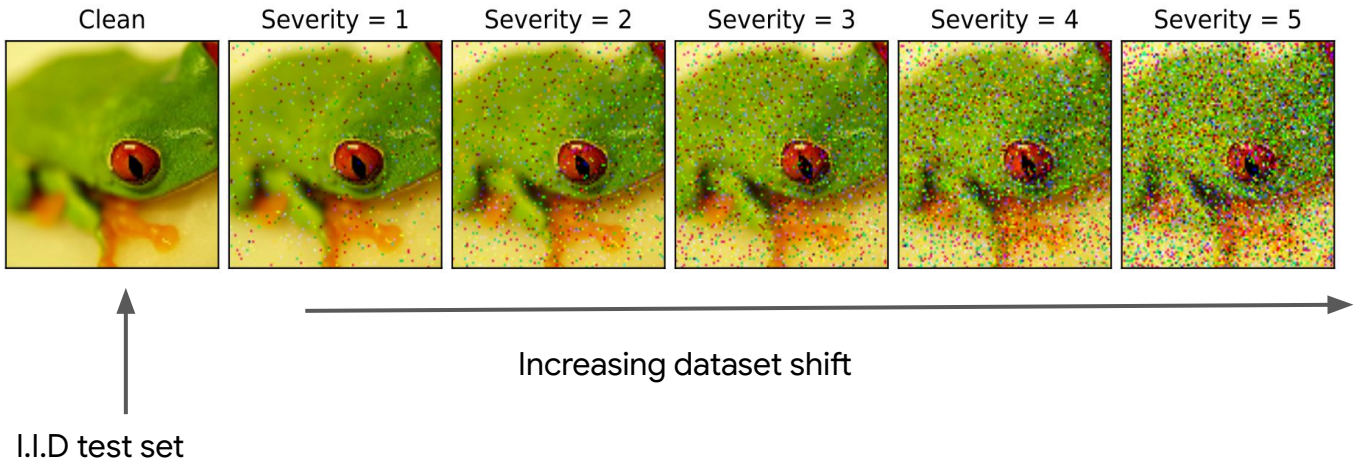
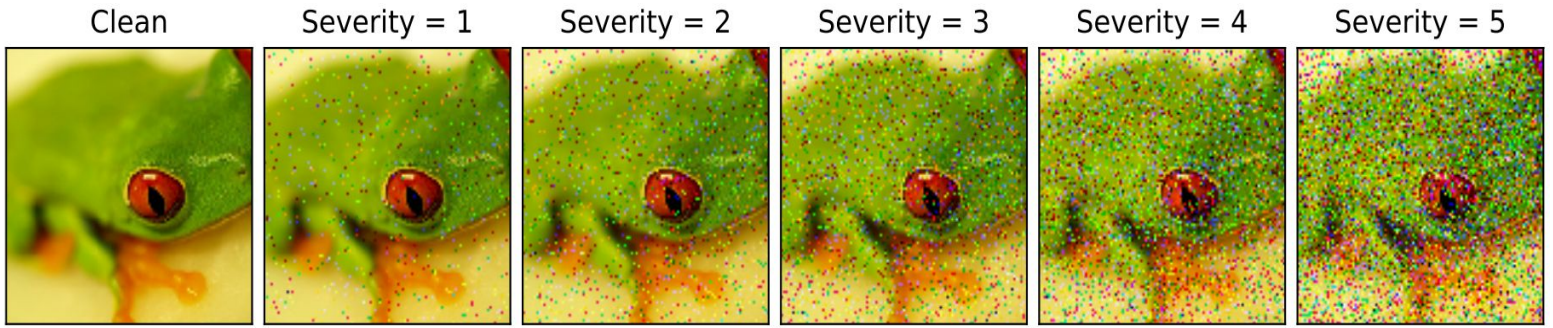


Image source: Benchmarking Neural Network Robustness to Common Corruptions and Perturbations, Hendrycks & Dietterich, 2019.

ImageNet-C: Varying Intensity for Dataset Shift

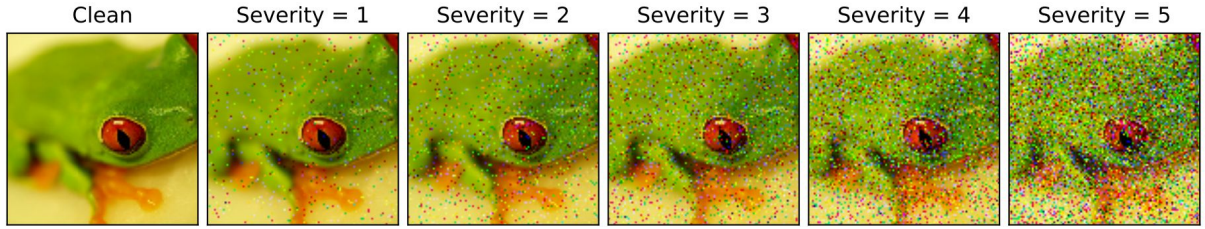


I.I.D test set

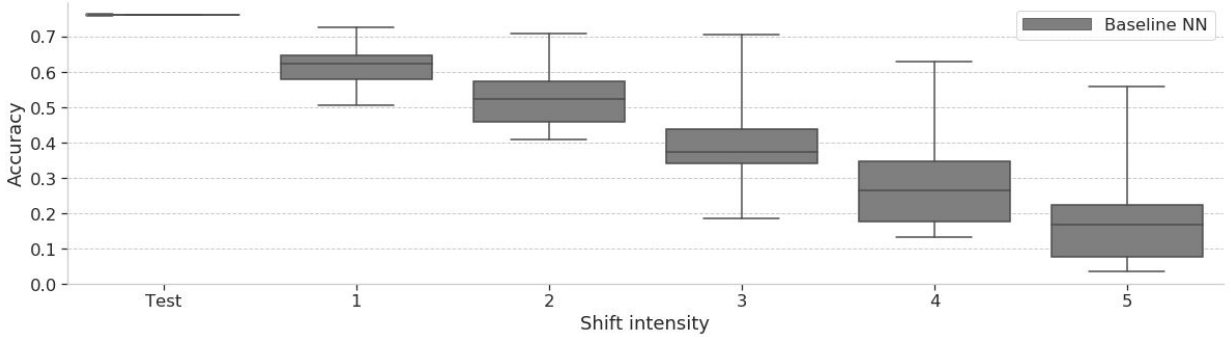
Increasing “OODness”
“Distance” between pTEST(y,x) and pTRAIN(y,x) increases

Image source: Benchmarking Neural Network Robustness to Common Corruptions and Perturbations, Hendrycks & Dietterich, 2019.

Neural networks do not generalize under covariate shift



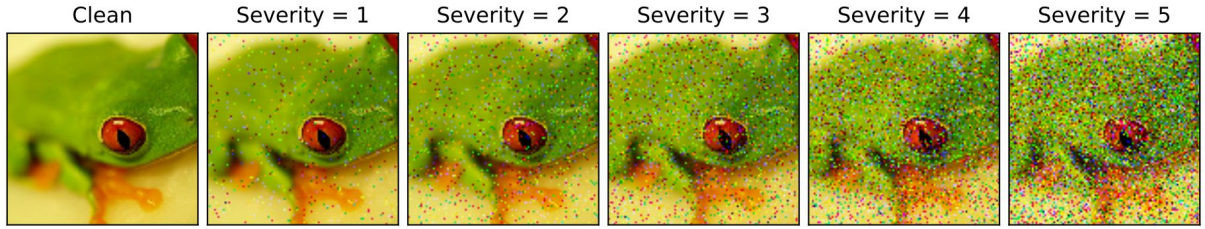
- **Accuracy drops** with increasing shift on Imagenet-C



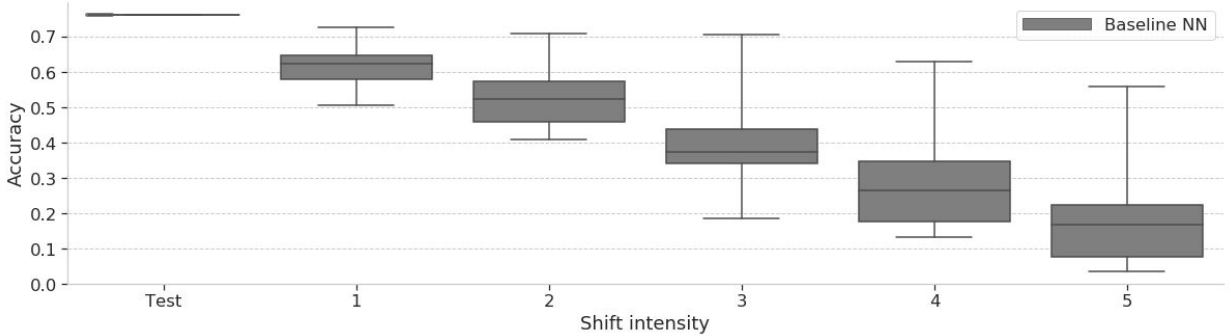
- But do the models know that they are less accurate?

Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift?, Ovadia et al. 2019

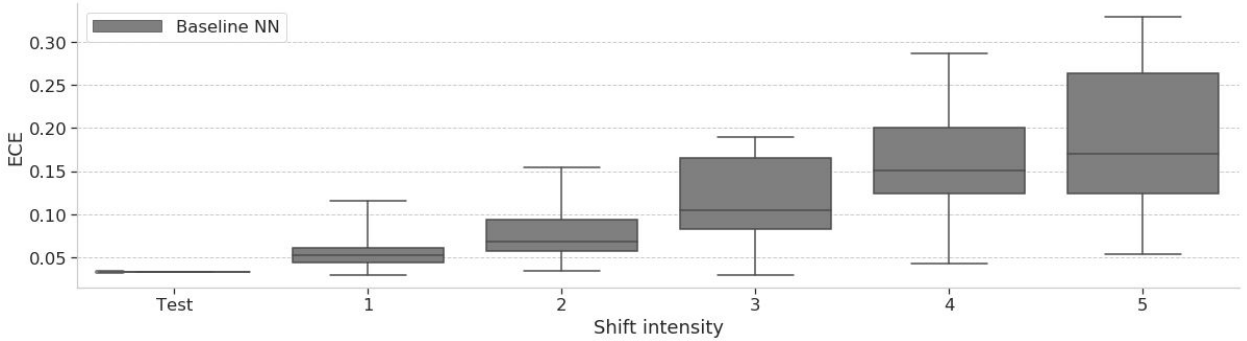
Neural networks *do not know when they don't know*



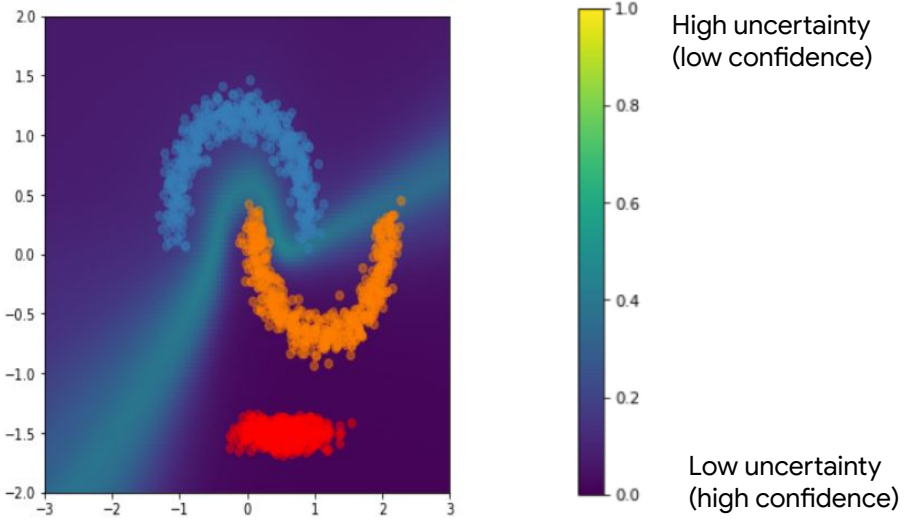
- **Accuracy drops** with increasing shift on Imagenet-C



- **Quality of uncertainty degrades with shift** -> "overconfident mistakes"



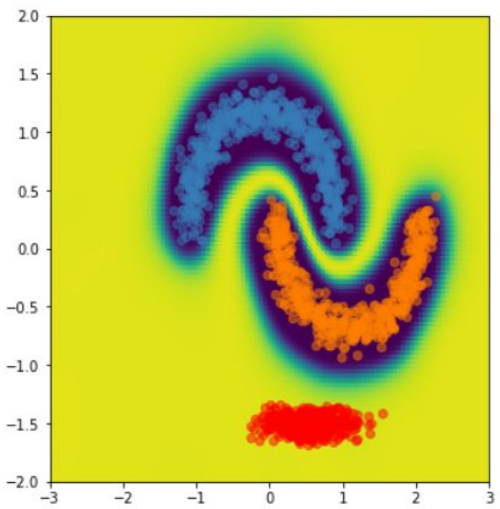
Models assign high confidence predictions to OOD inputs



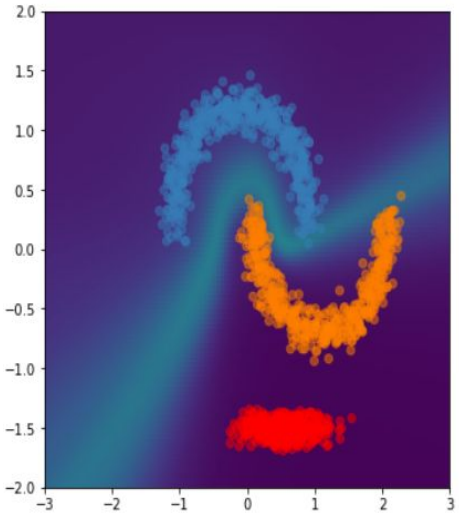
Deep neural networks

Image source: “Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness” Liu et al. 2020

Models assign high confidence predictions to OOD inputs



Ideal behavior



Deep neural networks

High uncertainty
(low confidence)

Low uncertainty
(high confidence)

Trust model when x* is close to $p_{TRAIN}(x,y)$

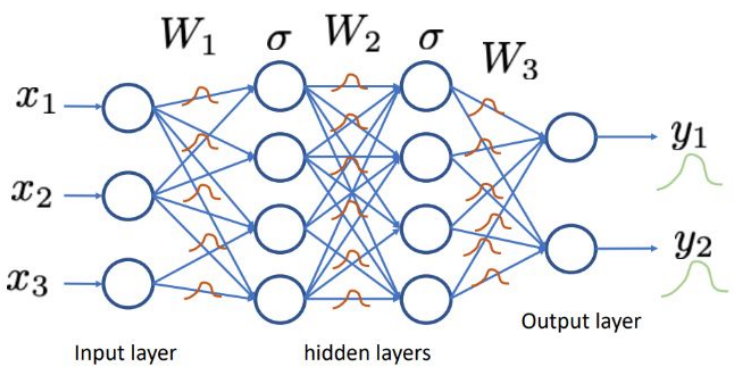
Image source: “Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness” Liu et al. 2020

Uncertainty Methodology

Bayesian Neural Networks

To capture epistemic uncertainty in a neural network, we model our ignorance with a prior distribution $p(\omega)$ over its weights.

Then we invoke Bayes for making predictions.



- The prior predictive distribution at $\mathbf{x}$ is given by integrating over all possible weight configurations,

$$p(y|\mathbf{x}) = \int p(y|\mathbf{x}, \omega)p(\omega)d\omega.$$

- Given training data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and $\mathbf{Y} = \{y_1, ..., y_N\}$, a Bayesian update results in the posterior

$$p(\omega|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \omega)p(\omega)}{p(\mathbf{Y}|\mathbf{X})}.$$

- The posterior predictive distribution is then given by

$$p(y|\mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int p(y|\mathbf{x}, \omega)p(\omega|\mathbf{X}, \mathbf{Y})d\omega.$$

- The prior predictive distribution at $\mathbf{x}$ is given by integrating over all possible weight configurations,

$$p(y|\mathbf{x}) = \int p(y|\mathbf{x}, \omega)p(\omega)d\omega.$$

- Given training data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and $\mathbf{Y} = \{y_1, ..., y_N\}$, a Bayesian update results in the posterior

$$p(\omega|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \omega)p(\omega)}{p(\mathbf{Y}|\mathbf{X})}.$$

- The posterior predictive distribution is then given by

$$p(y|\mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int p(y|\mathbf{x}, \omega)p(\omega|\mathbf{X}, \mathbf{Y})d\omega.$$

Aleatoric part

Epistemic part

Bayesian neural networks are easy to formulate, but notoriously difficult to perform inference in.

- This stems mainly from the fact that the marginal $p(\mathbf{Y} \mid \mathbf{X})$ is intractable to evaluate, which results in the posterior $p(w \mid \mathbf{X}, \mathbf{Y})$ not being tractable either.
- Therefore, we must rely on approximations.

Ensemble Learning

- A prior distribution often involves the complication of approximate inference.
- *Ensemble learning* offers an alternative strategy to aggregate the predictions over a collection of models.
- Often winner of competitions!
- There are two considerations: the collection of models to ensemble; and the aggregation strategy.

Popular approach is to average predictions of independently trained models, forming a mixture distribution.

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}_k)$$

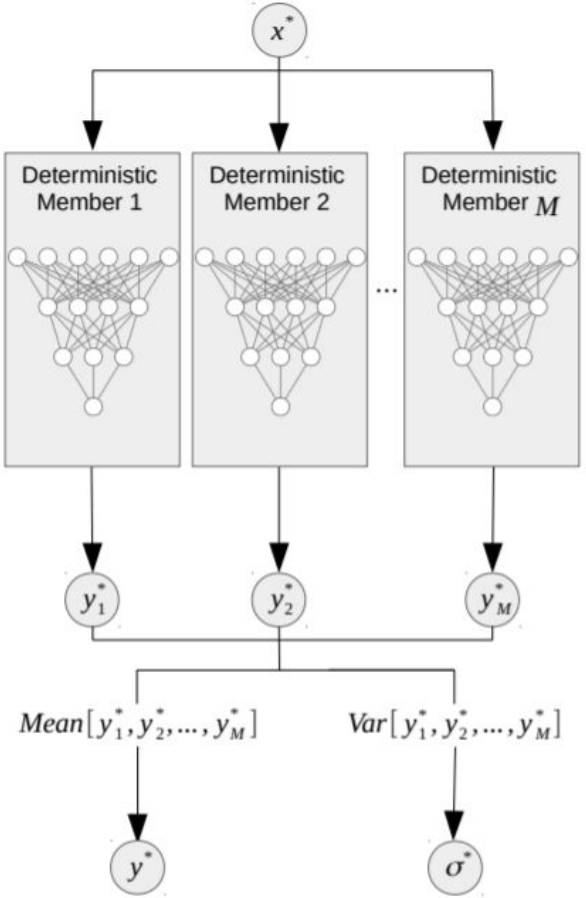
Many approaches exist: bagging, boosting, decision trees, stacking.

[Dietterich 2000]

Simple Baseline: Deep Ensembles

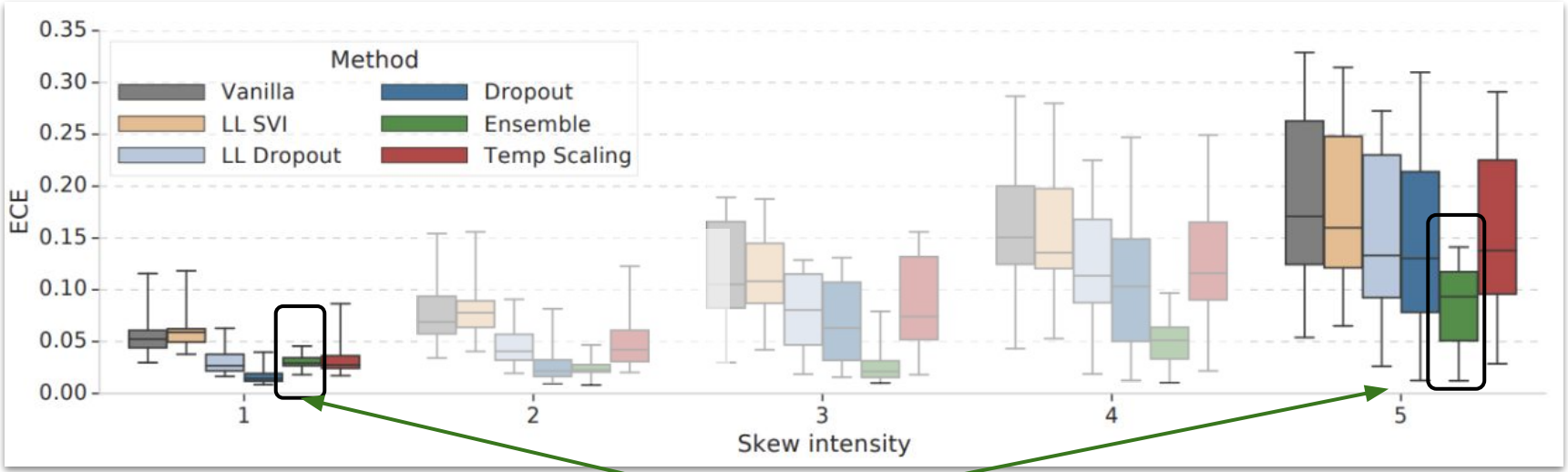
Idea: Just re-run standard SGD training but with different random seeds and average the predictions

- A well known trick for getting better accuracy and Kaggle scores
- We rely on the fact that the loss landscape is non-convex to land at different solutions
    - Rely on different initializations and SGD noise



[Lakshminarayanan+ 2017]

Deep Ensembles work surprisingly well in practice



Deep Ensembles are consistently among the best performing methods, especially under dataset shift

Adversarial Attacks

"We can cause the network to misclassify an image by applying a certain hardly perceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input."

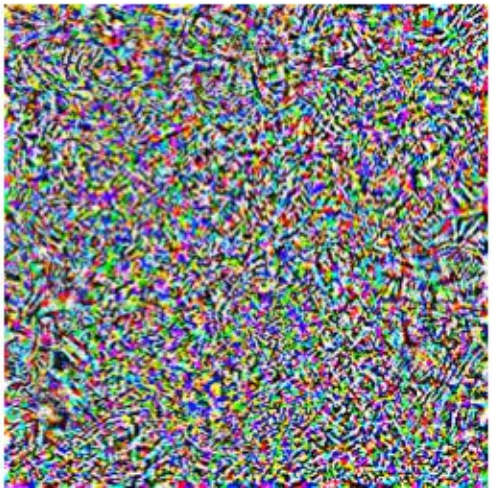
The existence of the adversarial negatives appears to be in contradiction with the network's ability to achieve high generalization performance. Indeed, if the network can generalize well, how can it be confused by these adversarial negatives, which are indistinguishable from the regular examples?"

(Szegedy et al, 2013)

Attacks

“pig”

+ 0.005 x



=

“airliner”

Defences:

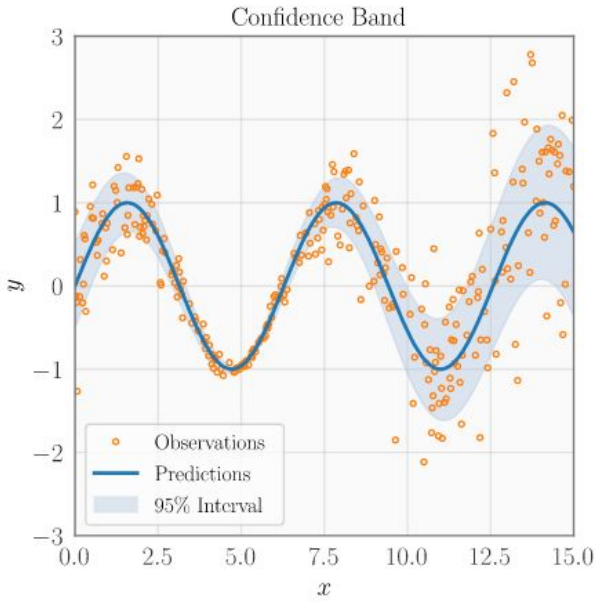
- Data augmentation
- Adversarial training
- Denoising / smoothing

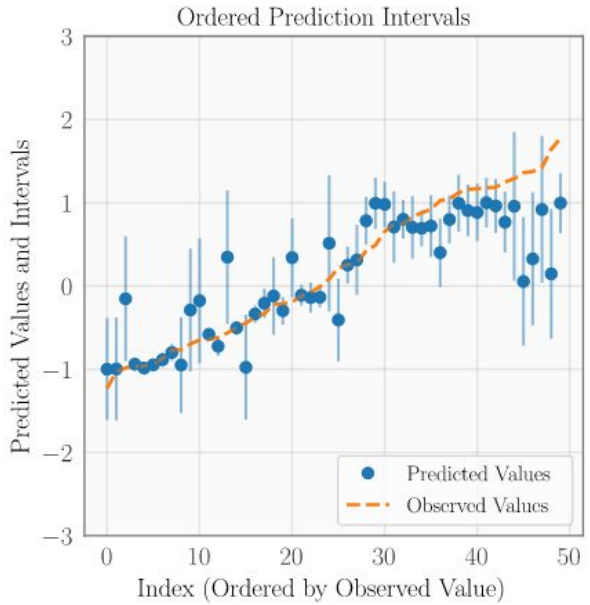
Probability Calibration

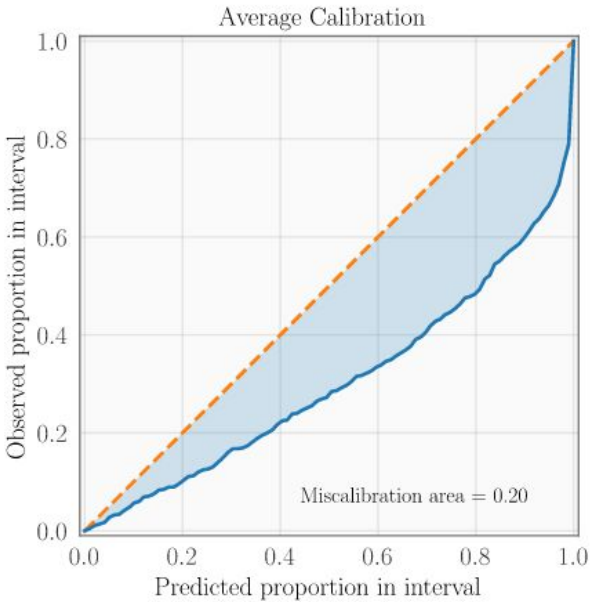
Many ML algorithms DO NOT provide probabilities by construction: the output is more a score of confidence than a true calibrated probability. They need to be calibrated if their outcomes to be used as probabilities.

- Calibrated probabilities match the “true” frequency of events
- Uncalibrated probabilities are either overconfident or underconfident

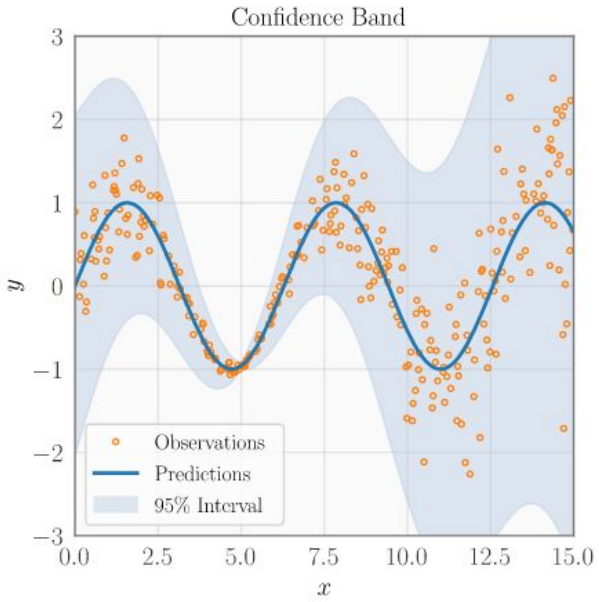
Overconfident (not enough uncertainty)

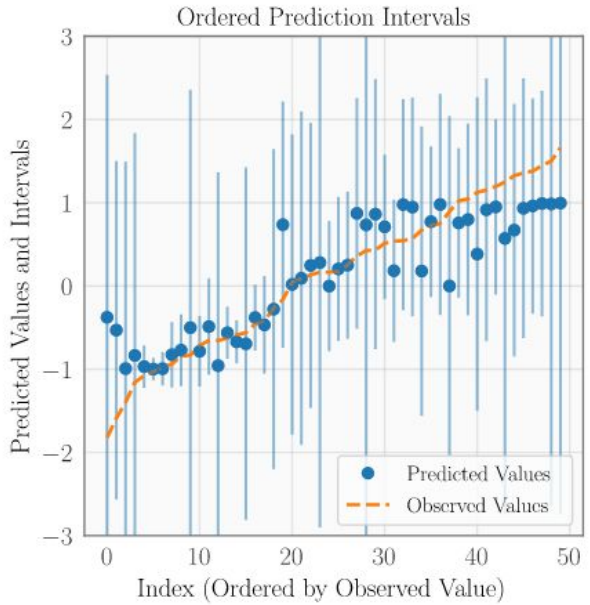


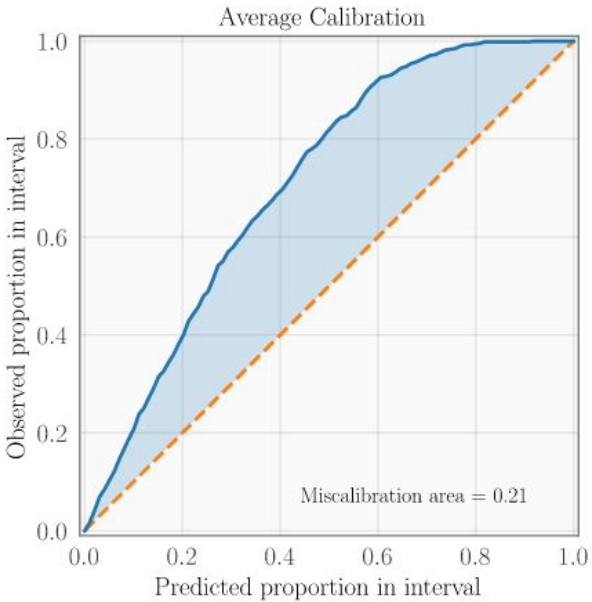




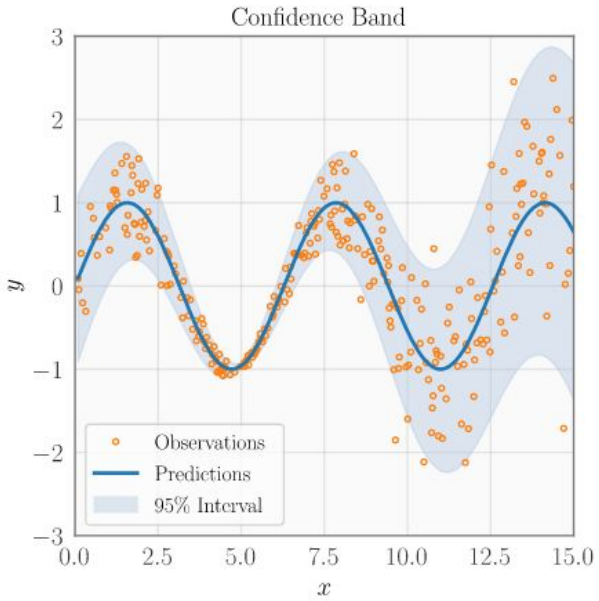
Underconfident (too much uncertainty)

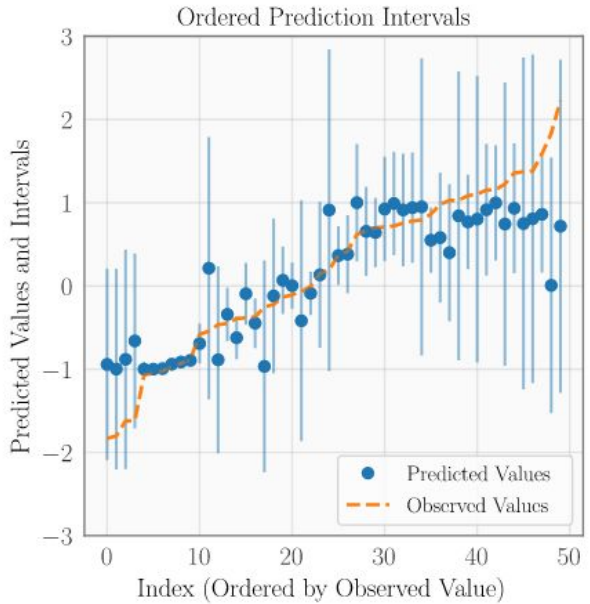


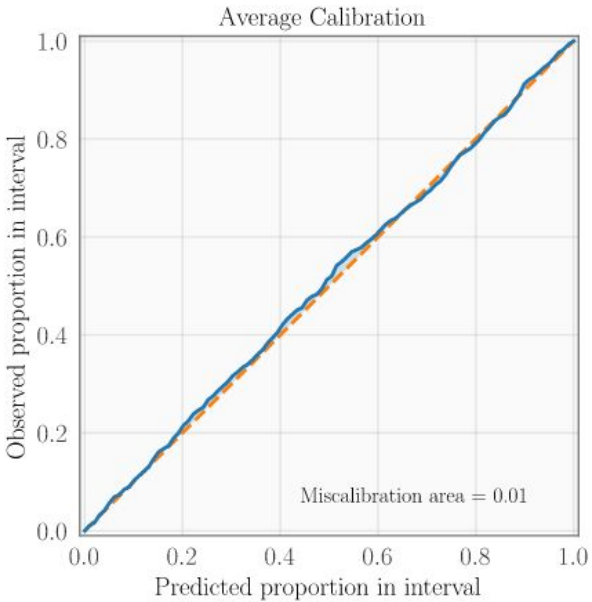




Well-calibrated







credits: [uncertainty-toolbox]

Probability Calibration

Often ML models constructed on probabilistic principles are calibrated. Here is a sample of algorithms and their calibration properties:

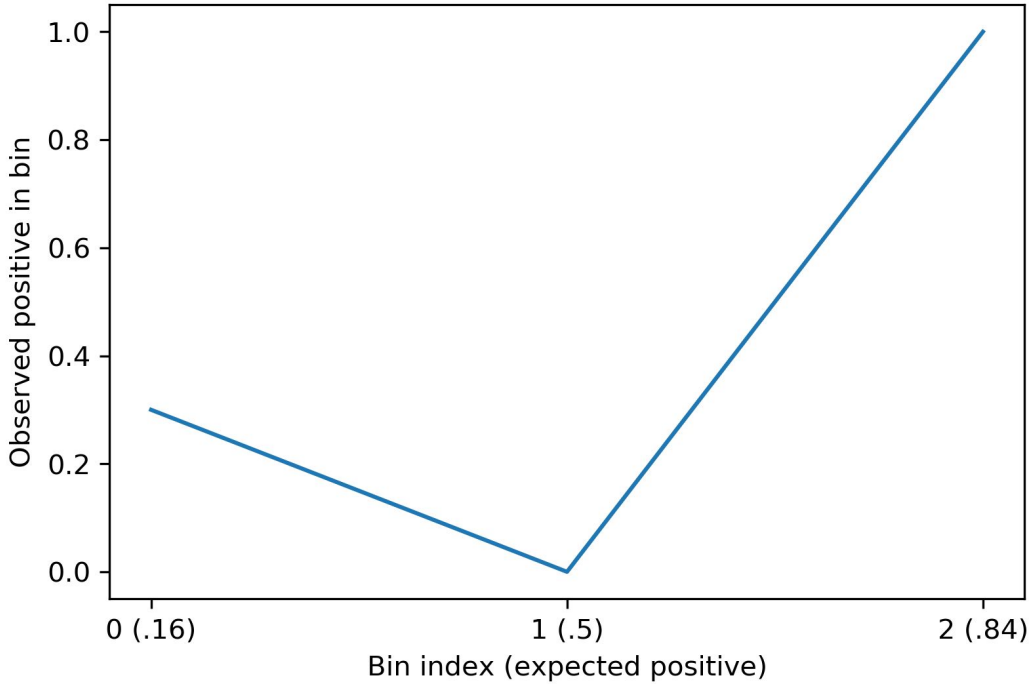
Calibrated Probability by Construction	Uncalibrated Probabilities
Logistic Regression	KNN
Linear Discriminant Analysis	SVM
Naïve Bayes	Decision Trees
ANN (with likelihood based loss)	Ensemble of Decision Trees (RF, GBT)

Calibration Curve

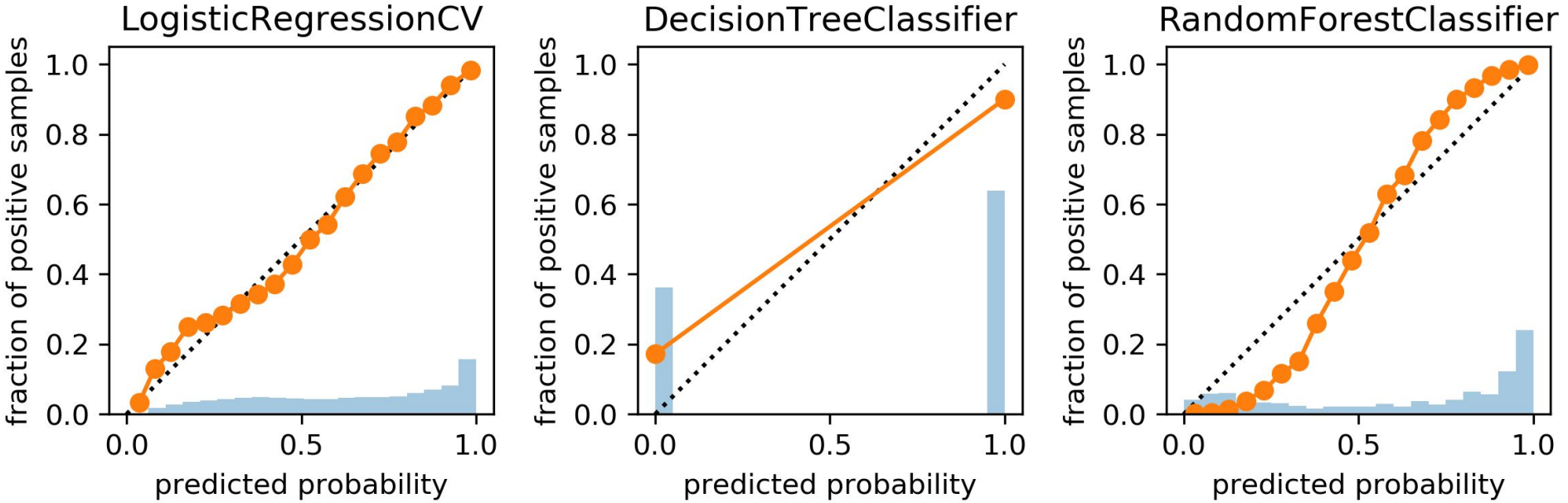
$\hat{p}(y)$	y
0.9	1
0.4	0
0.3	1
0.6	0
0.8	1
0.2	0
0.3	0

sort →

$\hat{p}(y)$	y	bin
0.9	1	}
0.8	1	}
0.6	0	}
0.4	0	}
0.3	1	}
0.3	0	}
0.2	0	}



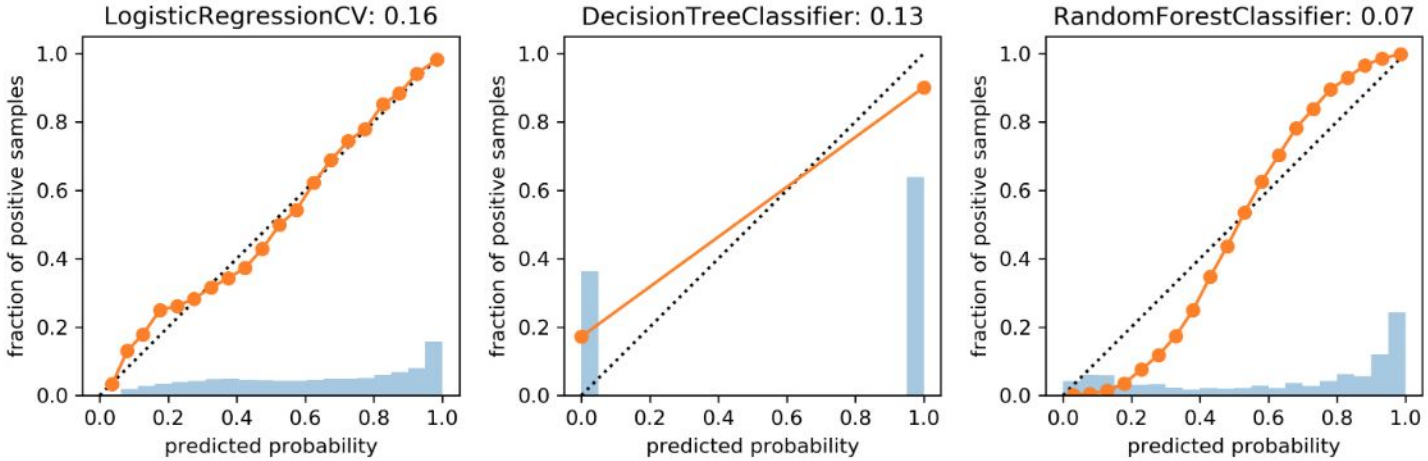
Calibration curve for ML models



Brier Score (Binary Classification)

MSE of proba

$$BS = \frac{\sum_{i=1}^{n} (\hat{p}(y_i) - y_i)^2}{n}$$



Platt Scaling

- Use a logistic sigmoid for f_{calib}
- Basically learning a 1d logistic regression
- (+ some tricks)
- Works well for SVMs

$$f_{platt} = \frac{1}{1 + exp(-ws(x) - b)}$$

Isotonic Regression

- Very flexible way to specify f_{calib}
- Learns arbitrary monotonically increasing step-functions in 1d.
- Groups data into constant parts, steps in between.
- Optimum monotone function on training data (wrt mse).