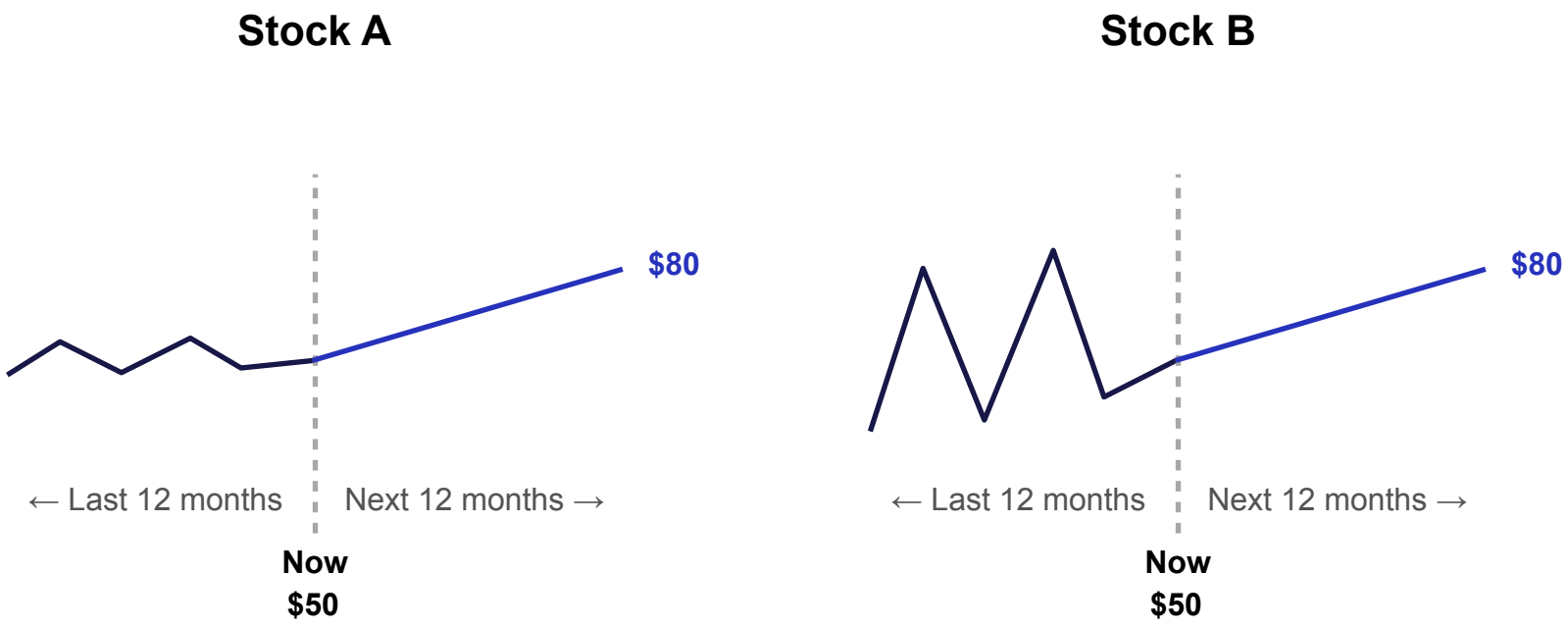
42578 Advanced Business Analytics

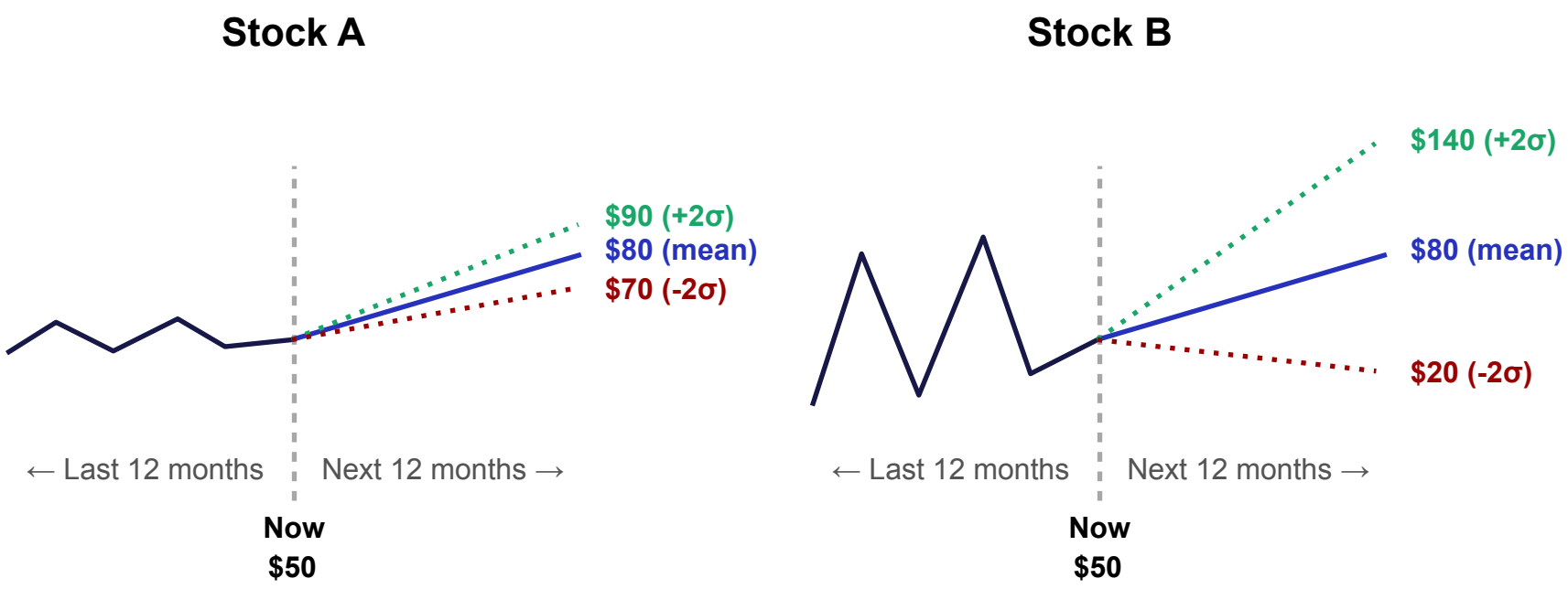
Prediction Uncertainty

Stanislav Borysov [stabo@dtu.dk]

Example: Stock price prediction



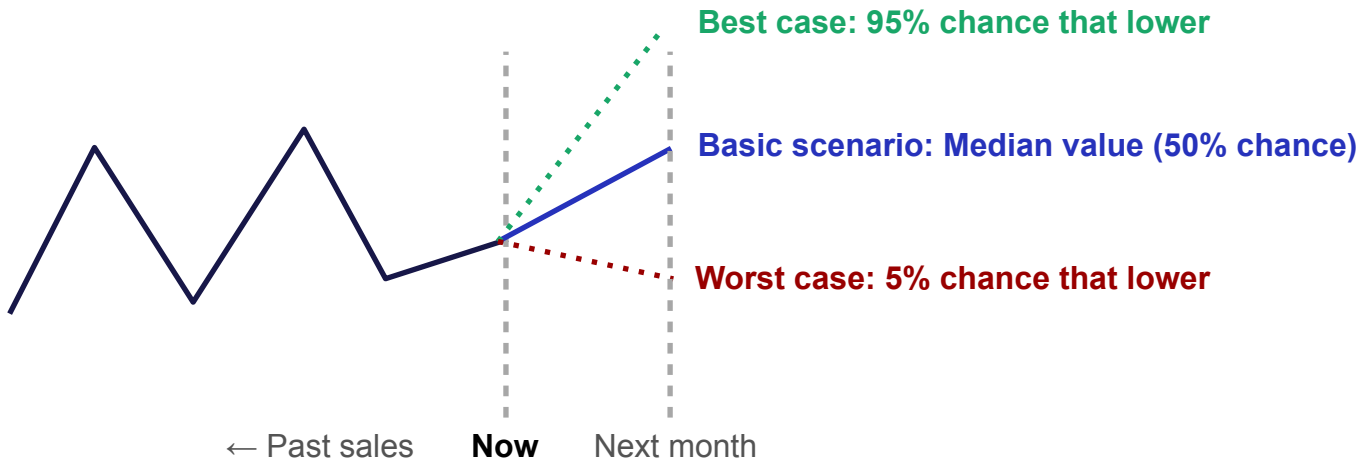
Example: Stock price prediction



- σ is a measure of risk (volatility)
- it is included in pricing models of many financial instruments

Example: Sales forecast

- How many items I do need to order / produce / deliver to shops next month?
- How much revenue can I expect?



Example: Churn prediction

- A classification model predicted that 1000 people might churn next month
- Your company has resources to provide benefits for 100 people only to prevent churns
- Given that the potential churns will lead to equal loss per person, whom to offer special incentives?

Example: Churn prediction

- A classification model predicted that 1000 people might churn next month
- Your company has resources to provide benefits for 100 people only to prevent churns
- Given that the potential churns will lead to equal loss per person, whom to offer special incentives?

- **Check how the model is sure about its prediction for each person!**
  - Check the predicted probabilities:
    p(person 1 will churn) = 0.8
    p(person 2 will churn) = 0.55
    p(person 3 will churn) = 0.99
    …
  - Offer benefits to the most probable churns

The problem

- **The task is to predict the distribution of the target variable**

- **Regression**
  - Usually, only the mean (1st moment) or median (50th percentile) is estimated
  - Uncertainty can be quantified from different perspectives:
    - Variance of the distribution (2nd moment)
    - "X% chance" — quantiles of the distribution
    - Higher-order moments (e.g., skewness, kurtosis, etc)
  - So, we need to specify the full distribution and estimate it

- **Classification**
  - The distribution (Bernoulli / Categorical distribution) is estimated directly; it can be specified with only one parameter for each class
  - Uncertainty is reflected in the predicted class (categorical) probabilities

- **Going deeper for both: Bayesian methods (not covered here)**

Uncertainty in Classification

Binary classification*

Data

$$\mathscr{D} = \left\{ \left(X^{(i)}, \underbrace{y^{(i)}}_{\text{Binary target variable}} \right) \right\}_{i=1}^{N}, y \in \{0,1\}$$

Model

$$p \equiv P\left(y = 1 \,|\, X; \beta\right) = f(X; \beta)$$

Decision function

$$d\left(p\right) \rightarrow \hat{y} = \{0,1\}$$

Examples

- *p=0.05* → most probably negative class (*ŷ=0*)
- *p=0.35* → probably negative class (*ŷ=0*)
- *p=0.99* → almost surely positive class (*ŷ=1*)

** All the material presented can be almost trivially extended to multi-class problems*

Binary classification

- **Examples of the models that directly predict probability** $P\left(y = 1 \mid X\right)$
  - Logistic regression
  - Neural networks
  - Naive Bayes

- **Examples of the models that predict probability-like scores**
  - Support Vector Machines (distance to the separating hyperplane, can be normalised)
  - Decision trees / Random forest / Boosted trees (frequency of the positive class in leaves)
  - k-Nearest Neighbors (fraction of the positive class in the NN)
  - …

Reliability Diagrams (Calibration Curves)

- Reliability diagrams are common aids for illustrating the properties of probabilistic forecast systems. They consist of a plot of the observed relative frequency against the predicted probability, providing a quick visual inter-comparison when tuning probabilistic forecast systems, as well as documenting the performance of the final product

- Specifically, the predicted probabilities are divided up into a fixed number of buckets along the x-axis. The number of events (class=1) is then counted for each bin (e.g. the relative observed frequency). Finally, the counts are normalised. The results are then plotted as a line plot.

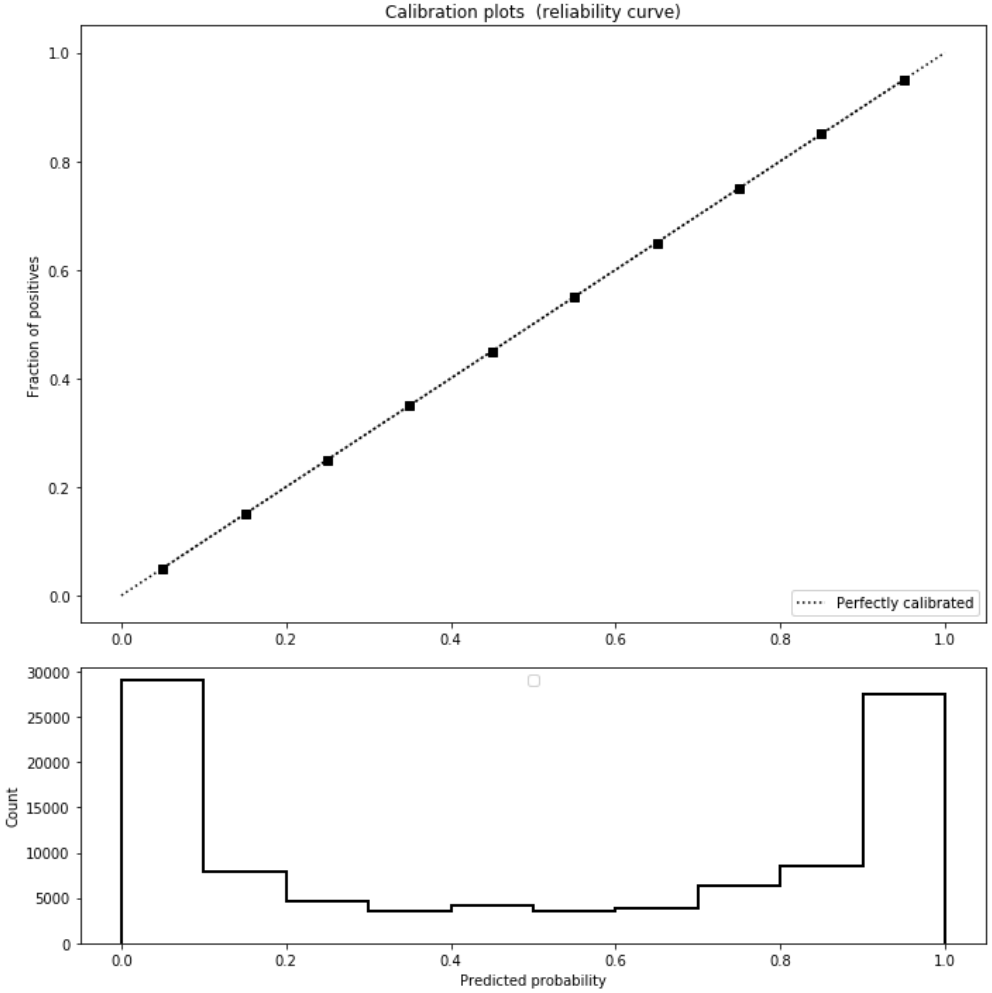
https://machinelearningmastery.com/calibrated-classification-model-in-scikit-learn/

Reliability Diagrams (Calibration Curves)

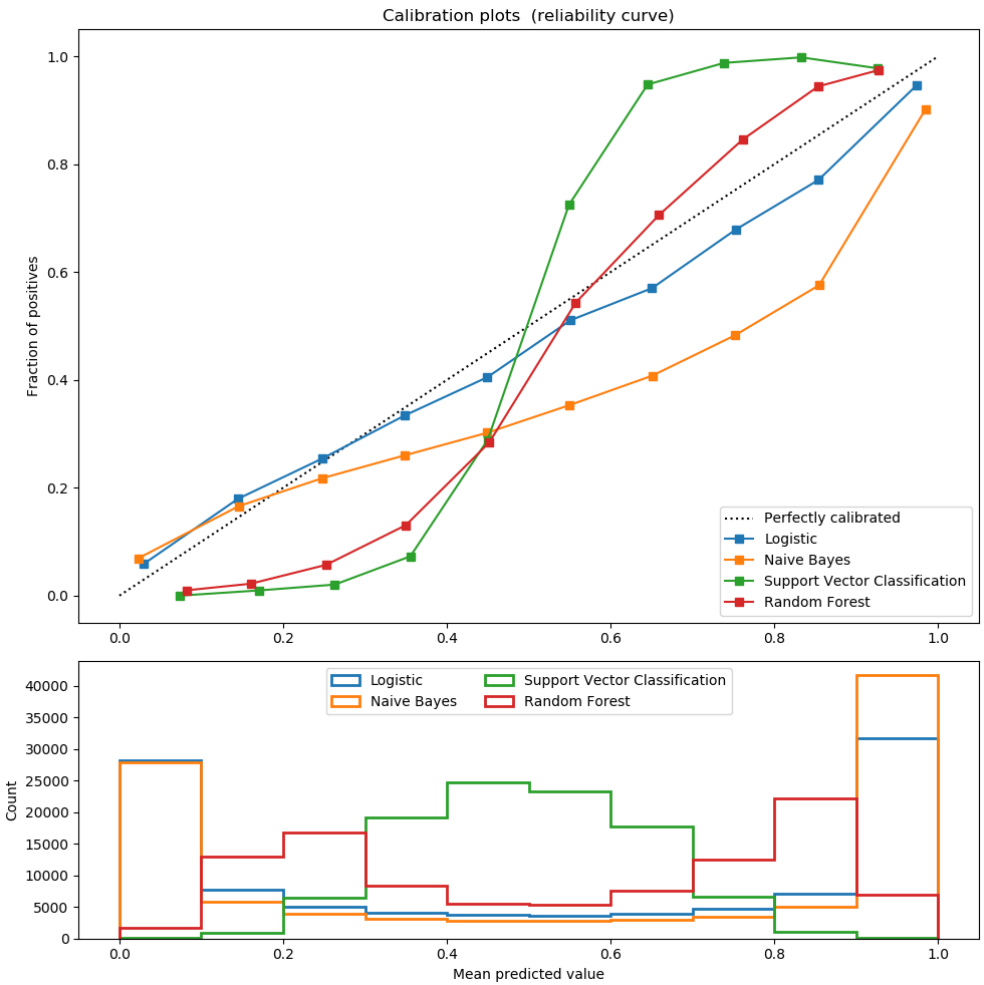
1. Take known data (e.g., training or validation set) and the fitted model
2. Calculate probabilities for the samples $p_i = f(X^{(i)}; \beta)$
   - Normalise to the range [0, 1] if needed
3. Plot histogram of the probabilities

- For example, take all the samples $\left\{ \left(\tilde{X}^{(i)}, \tilde{y}^{(i)} \right) \right\}_{i=1}^{\tilde{N}}$ for which $0.3 \leq p_i < 0.4$
- It means that the frequency of the positive class should be in this range $0.3 \leq \frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} \mathbf{1} \left(\tilde{y}^{(i)} = 1 \right) < 0.4$

Reliability Diagrams (Calibration Curves): Perfect classifier



Reliability Diagrams (Calibration Curves): Reality



Models output

- **Over-confident** (below the diagonal):
  The model has over-forecast; the probabilities are too large.
- **Under-confident** (above the diagonal):
  The model has under-forecast; the probabilities are too small.

- There are two main causes for uncalibrated probabilities:
  - Algorithms not trained using a probabilistic framework.
  - Biases in the training data.

- Usually, the best calibrated model is **Logistic Regression** (LR)

- **The rest:**
  - Neural Nets (~ multilayer LR) — not calibrated due to nonlinearities
  - Naive Bayes (NB) — suffer from redundant features
  - Ensemble models (e.g., RF) — suffer from averaging
  - SVM — problematic due to focus on the “hard” samples only

How to fix: Calibration of the models

1. Fit the main model on the training set $p = f(X; \beta)$
2. Fit a calibration model $p' = g(p; \theta)$ for the predicted probabilities
   - Do not fit on the test set
   - Ideally, the relative order of the predictions should be preserved, i.e., if $p_i < p_j$ then $g(p_i) < g(p_j)$, because it is the original model's job to get it correctly
   - Be careful when the data distribution is different for the training and test sets, for example, due to the balancing of the former. In this case, use a separate non-balanced validation set with a similar (stratified) probability distribution as for the test data

- **Most popular calibration models $g(p)$:**
  - Histogram binning
  - Platt scaling
  - Isotonic regression

https://scikit-learn.org/stable/modules/calibration.html

Histogram binning

1. Define *M* bins b_i: $\left[0; \frac{1}{M}\right), \left[\frac{1}{M}; \frac{2}{M}\right), \ldots, \left[\frac{M-1}{M}; 1\right]$
2. Group the data according to predicted probability p_i for the defined bins, $\mathscr{D}_i = \left\{ \left(X^{(j)}, y^{(j)}\right) : p_j = f\left(X^{(j)}\right) \in b_i \right\}$
3. Calculate the empirical frequency of the positive class for each bin, $\theta_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{1}\left(y^{(j)} = 1\right)$ for $y^{(j)} \in \mathscr{D}_i$ and $N_i = \left|\mathscr{D}_i\right|$ (number of data points in the bin *i*)
4. The calibrated probability p' is simply the empirical probability for the predicted bin: $p' = \theta_i$ when $p \in b_i$

Histogram binning

1. Define *M* bins b_i: $\left[0; \frac{1}{M}\right), \left[\frac{1}{M}; \frac{2}{M}\right), \ldots, \left[\frac{M-1}{M}; 1\right]$
2. Group the data according to predicted probability p_i for the defined bins, $\mathscr{D}_i = \left\{ \left(X^{(j)}, y^{(j)}\right) : p_j = f\left(X^{(j)}\right) \in b_i \right\}$
3. Calculate the empirical frequency of the positive class for each bin, $\theta_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{1}\left(y^{(j)} = 1\right)$ for $y^{(j)} \in \mathscr{D}_i$ and $N_i = \left|\mathscr{D}_i\right|$ (number of data points in the bin *i*)
4. The calibrated probability p' is simply the empirical probability for the predicted bin: $p' = \theta_i$ when $p \in b_i$

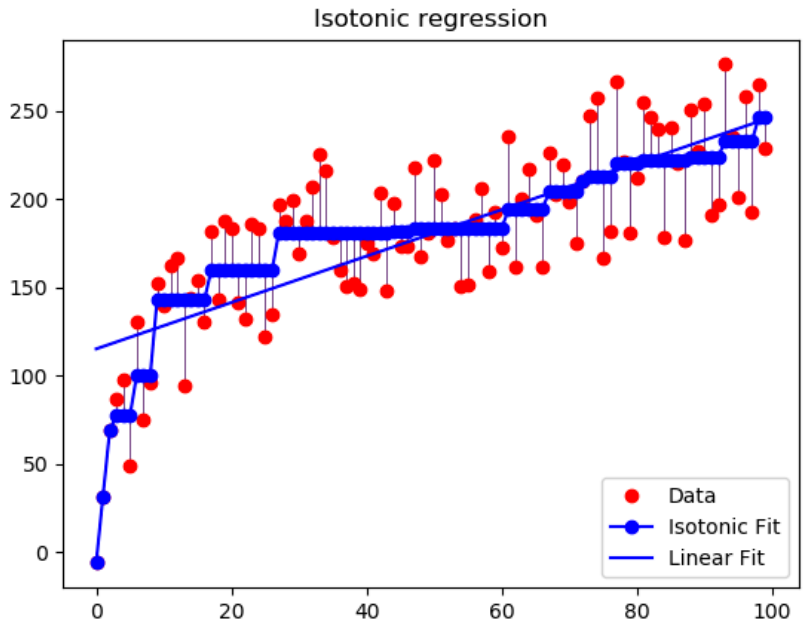
- **The method is rarely used because of discarding the original probabilities within the bins, which leads to poor performance**

Platt ("sigmoid") scaling

- Fit another logistic regression: $p' = g(p; \theta_0, \theta_1) = \dfrac{1}{1 + e^{\theta_0 + \theta_1 p}}$
- Works well for S-shaped uncalibrated probabilities
- Implemented in sklearn
  https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
- **Temperature scaling** – a single-parameter variant ($\theta_0 \equiv 0$) of Platt Scaling – is surprisingly effective at calibrating predictions for NN, especially in a multi-class setting
  https://arxiv.org/pdf/1706.04599.pdf

Isotonic regression

- Fits piece-wise non-decreasing function to the data, $p_i' = g(p_i; \theta) = \theta_i$ with the constraint $\theta_i \leq \theta_{i+1}$
- Number of parameters θ is equal to the number of data points
  - Binning can be considered



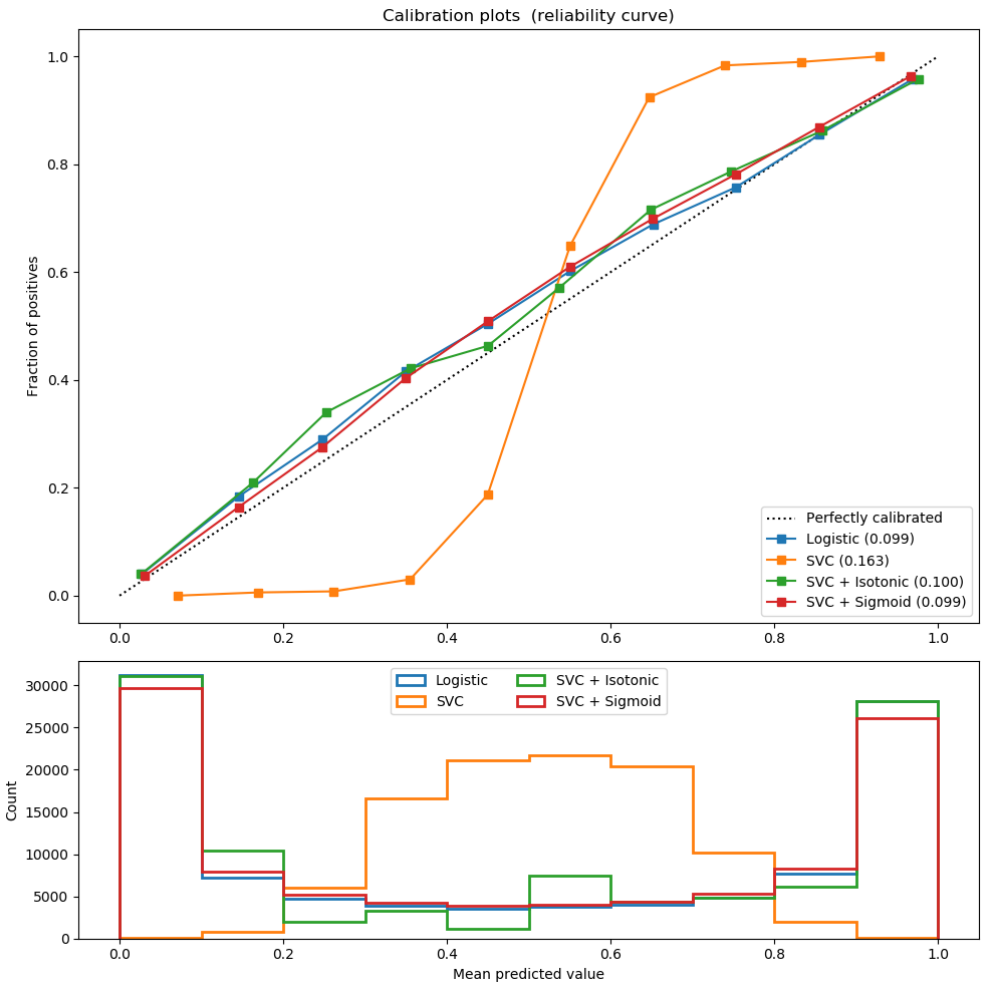
Isotonic regression

- One of the most common non-parametric calibration method: It can correct any monotonic distortion

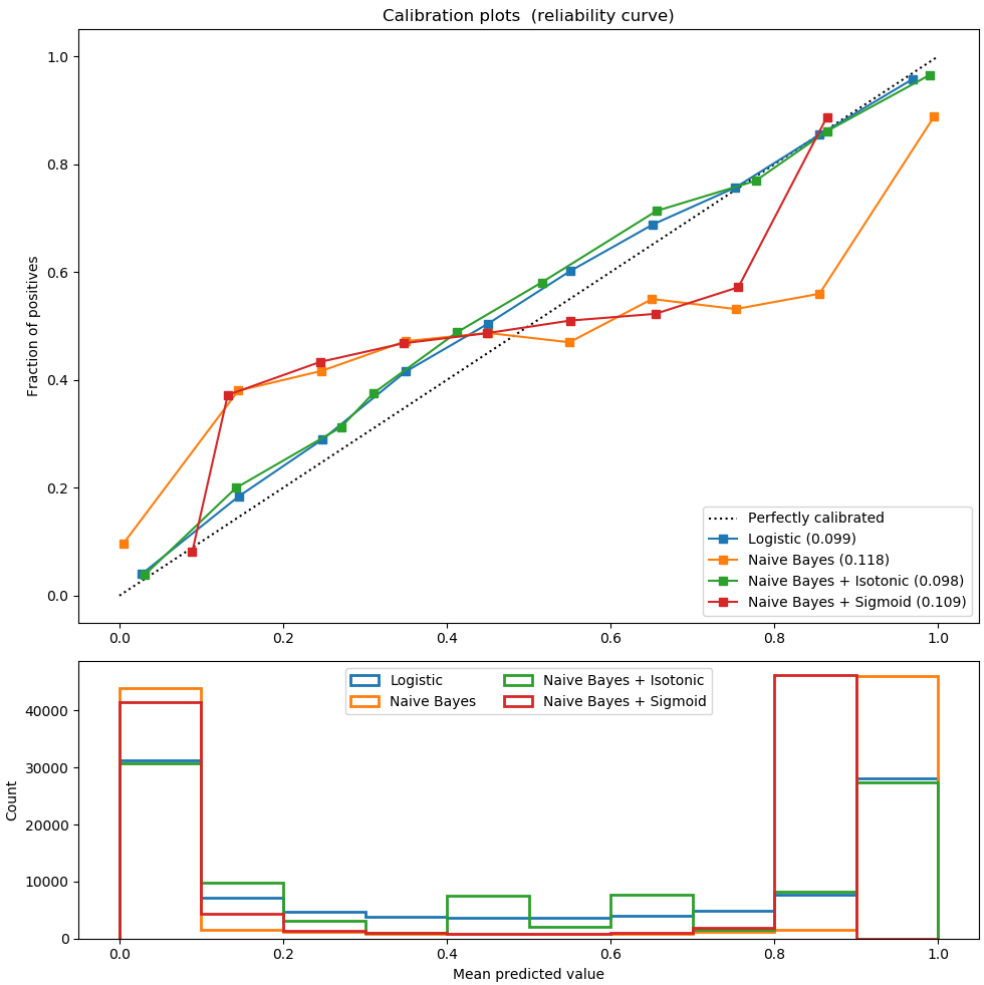
- It is not advised to use isotonic calibration with too few calibration samples (<<1000) since it tends to overfit

- Implemented in sklearn
  https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
  and
  https://scikit-learn.org/stable/modules/generated/sklearn.isotonic.IsotonicRegression.html

Calibration results (SVM)



Calibration results (NB)



Measuring performance of calibrated models

- **Note: Better calibrated probabilities may or may not lead to better class-based or probability-based predictions. It really depends on the specific metric used to evaluate predictions.**

- The predicted probability provides the basis for more granular model evaluation and selection, such as through the use of ROC and Precision-Recall diagnostic plots, metrics like ROC AUC, Brier score, and techniques like threshold moving.

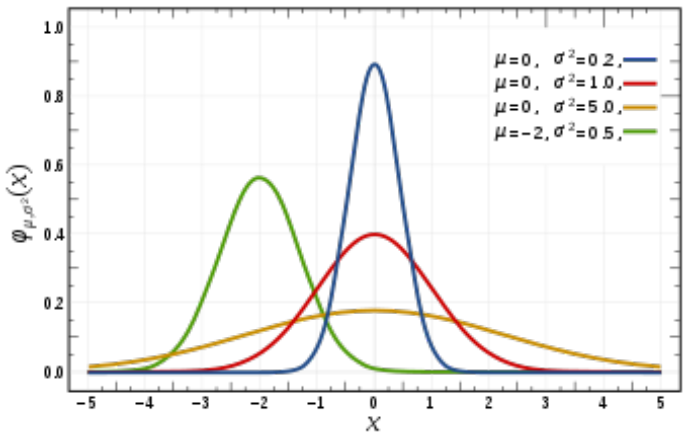
Uncertainty in Regression

How to specify continuous distribution

- **Specify Probability Density Function (PDF)**
  - Known functional form with a few parameters (Normal, exponential, etc)
  - Flexible non-parametric forms (NNs, kernel methods)
  - Moments and cumulants
  - Mixture of distributions

- **Specify Cumulative Distribution Function (CDF)**
  - Parametric forms are usually complicated (involving integrals, etc)
  - Quantile function is easier (= inverse of CDF)

Modelling PDF: Assume parametric form



- Gaussian: N(μ, σ)
- Learn not only μ(x) (OLS) but σ(x) as well!
- Use σ(x) as uncertainty measure

Regression problem: Ordinary least squares

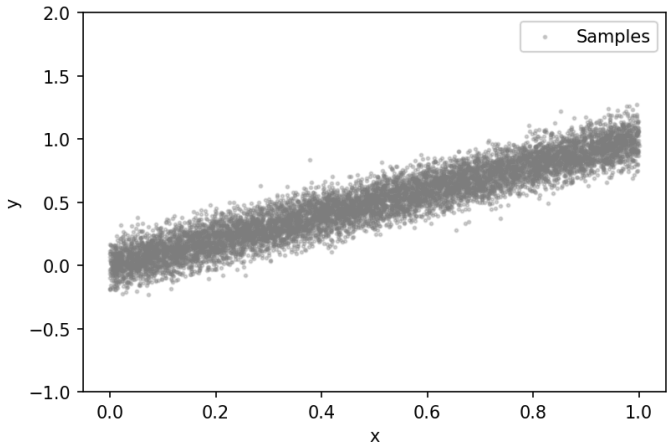
Data

$$\mathscr{D} = \left\{ \left(x^{(i)}, \underline{y^{(i)}} \right) \right\}_{i=1}^{N}$$

Continuous target variable

Model

$$\hat{y} = f(x; \beta)$$

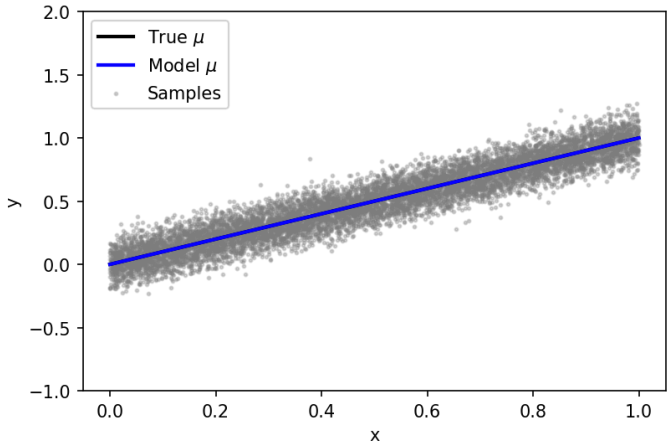


Loss: Mean squared error

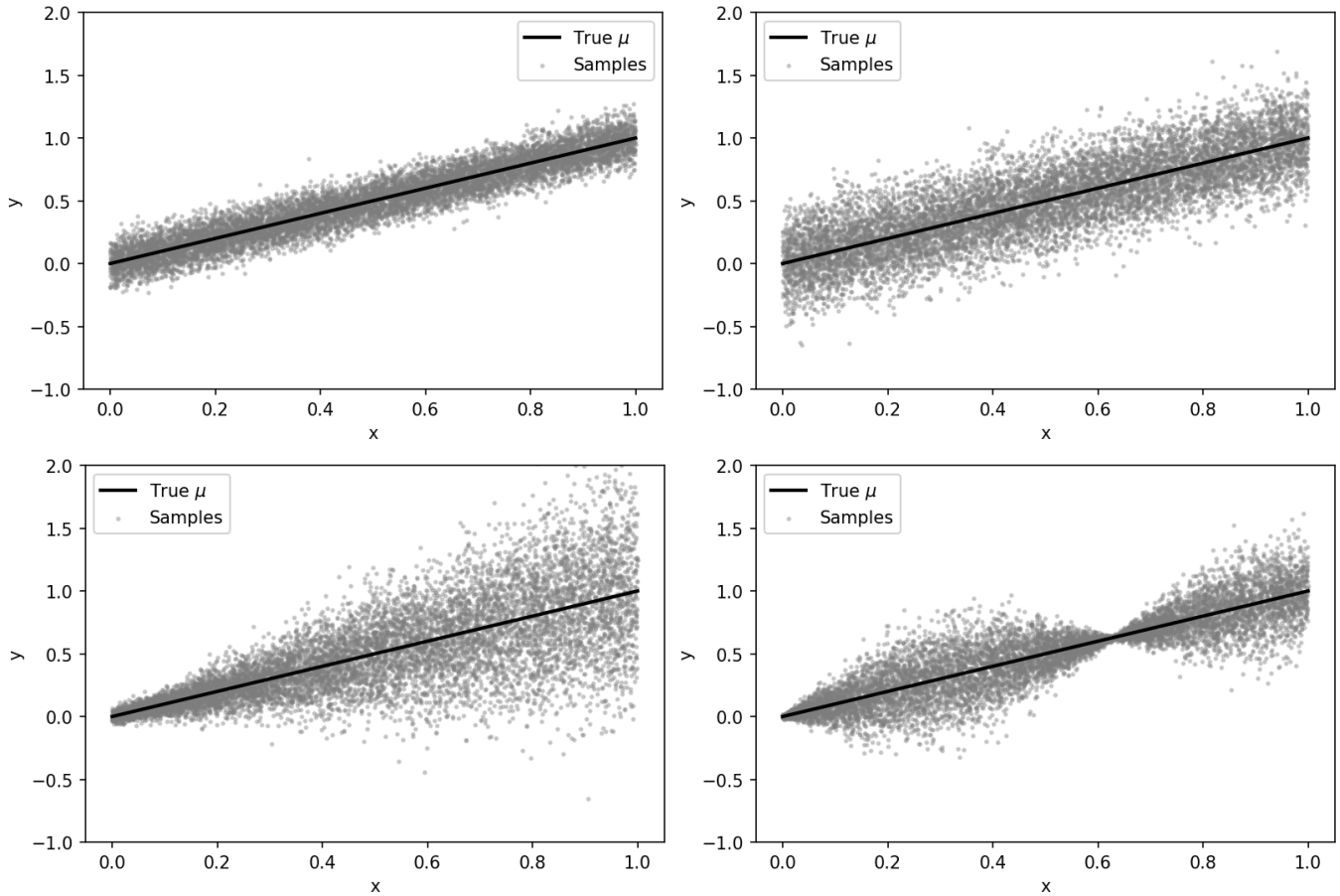
$$\mathrm{MSE}(\beta) = \frac{1}{N} \sum_{\mathscr{D}} \left(y - \hat{y}(x; \beta) \right)^2$$

Optimisation

$$\beta^* = \min_{\beta} \mathrm{MSE}(\beta)$$



The same y(x), different noise



Regression problem: Probabilistic formulation

Data

$$\mathscr{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N}$$

Model

$$\hat{y} = f(x;\beta) \boxed{+ \epsilon(\alpha)}$$

where, *ϵ(σ)* is a random variable (parametrised by *α)*

Thus, *y* becomes a random variable as well!

Loss: Likelihood

$$\mathrm{L} = p\left(\left\{ y^{(i)} \right\}_{i=1}^{N} | \hat{y}\left(\left\{ x^{(i)} \right\}_{i=1}^{N}; \beta, \alpha \right) \right) \equiv p\left(\mathbf{y} \,|\, \mathbf{x}, \beta, \alpha \right)$$

Optimisation: Maximum Likelihood Estimation

$$\beta^*, \alpha^* = \max_{\beta,\theta} \mathrm{L}(\beta, \alpha)$$

Regression problem: Probabilistic formulation

Data

$$\mathscr{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N}$$

Model

$$\hat{y} = f(x;\beta) \boxed{+ \epsilon(\alpha)}$$

where, *ϵ(σ)* is a random variable (parametrised by *α)*

Thus, *y* becomes a random variable as well!

Loss: Likelihood

$$\mathrm{L} = p\left(\left\{ y^{(i)} \right\}_{i=1}^{N} | \hat{y}\left(\left\{ x^{(i)} \right\}_{i=1}^{N}; \beta, \alpha \right) \right) \equiv p\left(\mathbf{y} \,|\, \mathbf{x}, \beta, \alpha \right)$$

Optimisation: Maximum Likelihood Estimation

$$\beta^*, \alpha^* = \max_{\beta,\theta} \mathrm{L}(\beta, \alpha)$$

In practice

i.i.d. assumption

$$p\left(\mathbf{y} \,|\, \mathbf{x}, \beta, \alpha \right) = \prod_{i=1}^{N} p\left(y^{(i)} \,|\, x^{(i)}, \beta, \alpha \right)$$

Log-likelihood

$$\mathrm{LL} = \log \mathrm{L} = \sum_{i=1}^{N} \log p\left(y^{(i)} \,|\, x^{(i)}, \beta, \alpha \right)$$

minimising negative log-likelihood

$$\mathrm{NLL} = -\,\mathrm{LL}$$

$$\beta^*, \alpha^* = \min_{\beta,\alpha} \mathrm{NLL}(\beta, \alpha)$$

Example: Gaussian noise

Gaussian PDF

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

Model

$$\hat{y} = f(x;\beta) + \mathcal{N}\left(\mu = 0, \sigma\right) = \mathcal{N}\left(\mu = f(x;\beta), \sigma\right)$$

LL with i.i.d. assumption

const

$$\mathrm{LL} = \sum_{i=1}^{N} \log p\left(y^{(i)} \mid x^{(i)}, \beta, \sigma\right) = -\frac{N}{2}\cancel{\log 2\pi} - N \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(y^{(i)} - f\left(x^{(i)}; \beta\right)\right)^2$$

Example: Gaussian noise

Gaussian PDF

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

Model

$$\hat{y} = f(x;\beta) + \mathcal{N}\left(\mu = 0, \sigma\right) = \mathcal{N}\left(\mu = f(x;\beta), \sigma\right)$$

LL with i.i.d. assumption

$$\mathrm{LL} = \sum_{i=1}^{N} \log p\left(y^{(i)} \mid x^{(i)}, \beta, \sigma\right) = -\frac{N}{2}\underbrace{\cancel{\log 2\pi}}_{\text{const}} - N \log \sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y^{(i)} - f\left(x^{(i)};\beta\right)\right)^2$$

Given that sigma is const $\quad \beta^* = \max_{\beta} \mathrm{LL}(\beta, \sigma) = \min_{\beta} \mathrm{MSE}(\beta)$

MLE estimates for beta are the same as OLS!

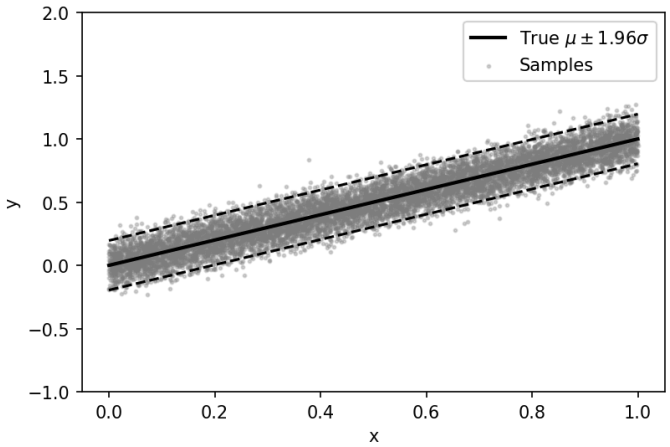
(In fact, LL is a generalisation of OLS)

OLS estimates

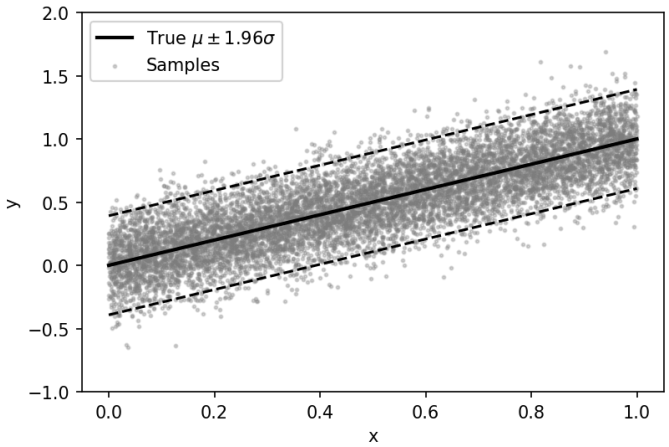
$$\mathbb{E}\left[y \mid x\right] = \mu(x)$$

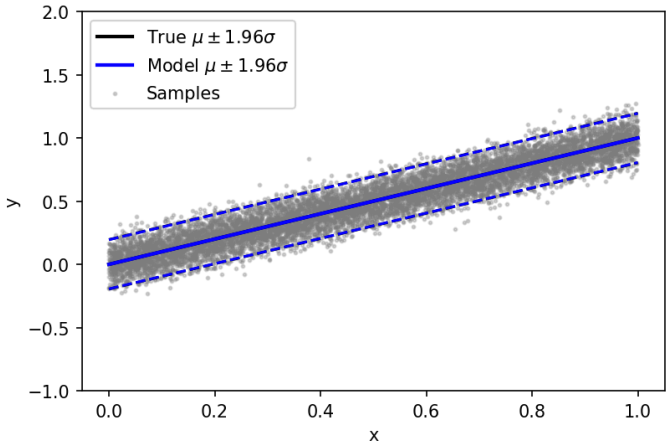
Linear regression: Homoscedastic case (constant noise variance)

$$y \sim \mathcal{N}\left(\mu = x, \sigma = 0.1\right)$$



$$y \sim \mathcal{N}\left(\mu = x, \sigma = 0.2\right)$$

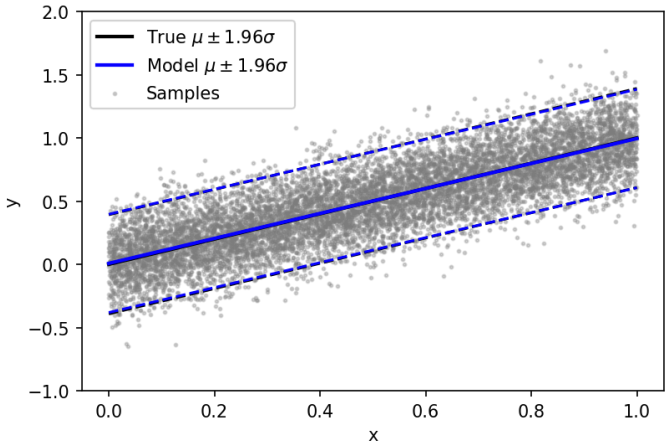




Model

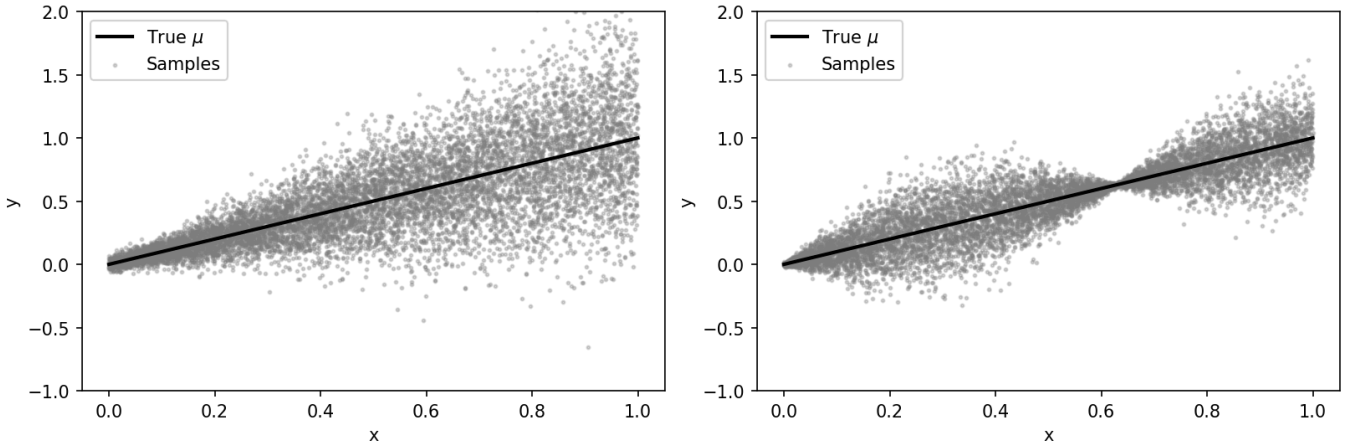
$$\hat{\mu} = \beta_0 + \beta_1 x$$

$$\hat{\sigma} = \alpha_0$$



95% prediction interval = 1.96σ

What about non-constant variance? (a.k.a. heteroscedasticity)



What about non-constant variance? (a.k.a. heteroscedasticity)

But we can define σ(x; α) and optimise LL for both β and α

(estimates for β will remain the same if $\sigma(x; \alpha)=\alpha_0=\text{const}$)

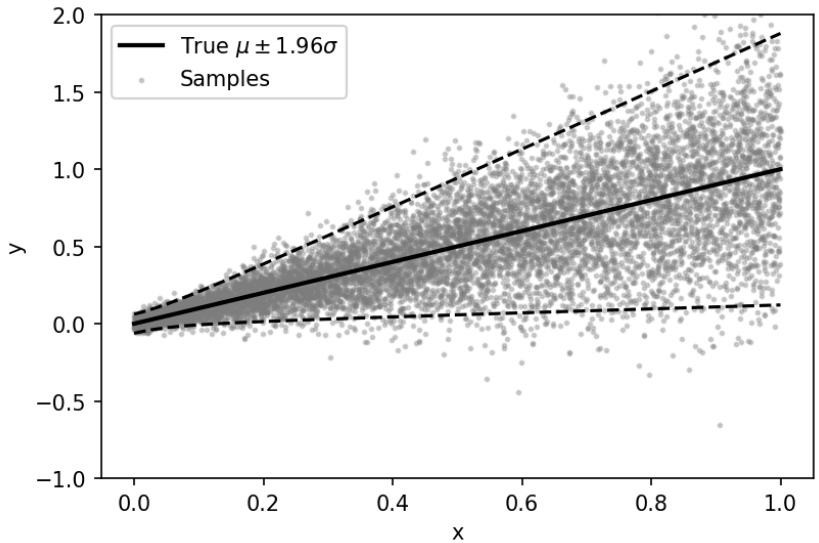
Define another model for σ(x)=g(x; α) and optimise LL for α and β simultaneously

$$\mathrm{LL}\left(\beta, \alpha\right) = -\frac{N}{2}\underbrace{\cancel{\log 2\pi}}_{\text{const}} - N \log g\left(x^{(i)}; \alpha\right) - \frac{1}{2}\sum_{i=1}^{N}\frac{\left(y^{(i)} - f\left(x^{(i)}; \beta\right)\right)^2}{g\left(x^{(i)}; \alpha\right)^2}$$

The problem becomes non-convex 😔

Non-constant variance example

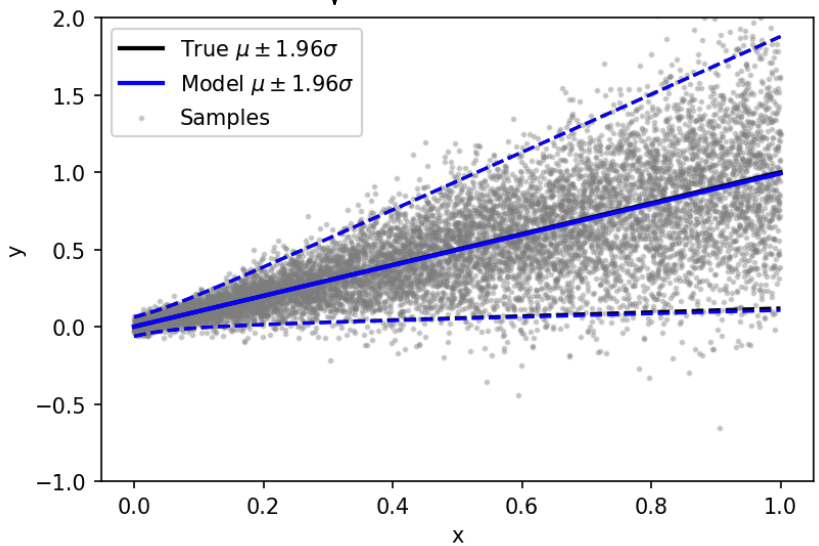
$$y \sim \mathcal{N}\left(\mu = x, \sigma = \sqrt{0.001 + 0.2x^2}\right)$$



Model

$$\hat{\mu} = \beta_0 + \beta_1 x$$

$$\hat{\sigma} = \sqrt{\alpha_0 + \alpha_1 x + \alpha_2 x^2}$$



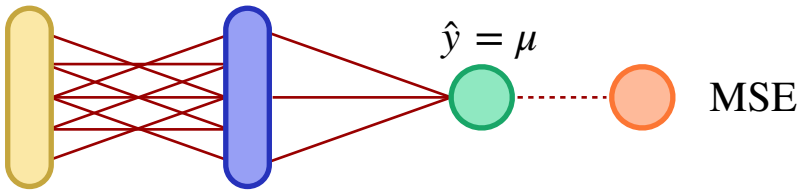
General approach

1. Make assumptions about the target variable distribution $p(y \mid \lambda_1(x), \lambda_2(x), \ldots)$
2. Define NLL
3. Specify models for the distribution parameters $\boldsymbol{\lambda(x)}$
4. Optimise for the models' parameters using your favourite tools
   - include constraints if necessary (e.g., $\sigma>0$ for the normal distribution)
   - or use a transformation of λ (e.g., $\log(\sigma)$)

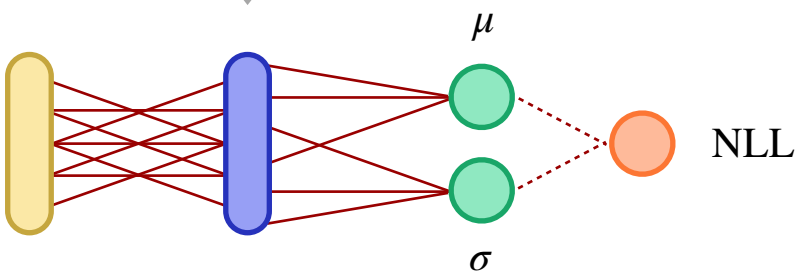
- **ANNs are very convenient for the last two steps!**

MLE using ANN

Mean estimation (OLS)



Mean and variance estimation (MLE)



Practical tips

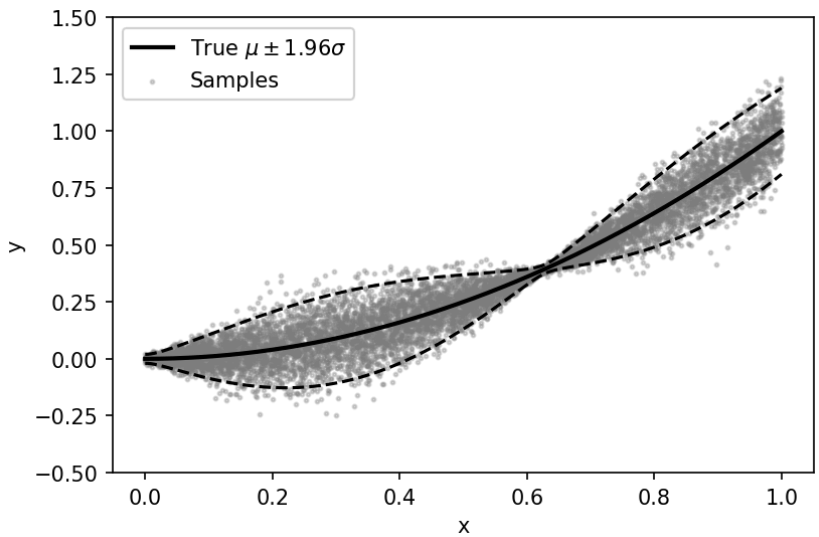
- To ensure positive σ, use ReLU or EXP neurons instead of the linear
- Or learn log(σ) using a linear neuron

Custom NLL loss in Keras

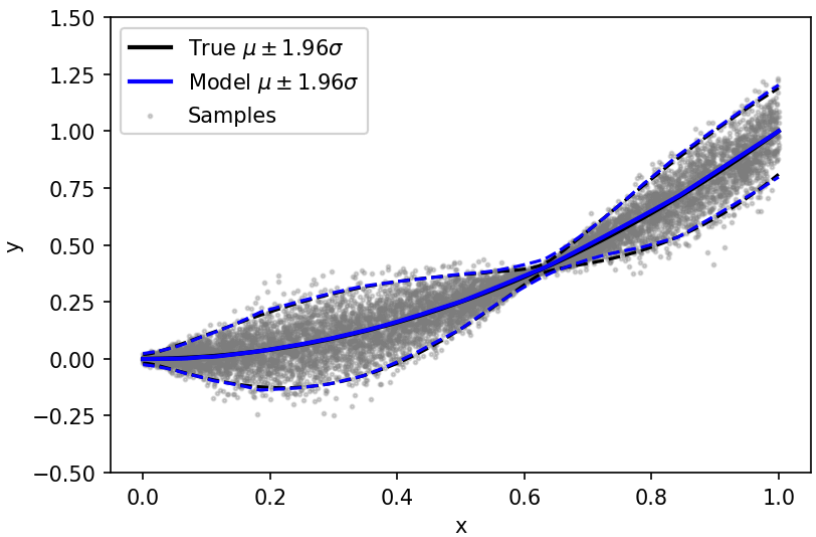
```
from keras import backend as K # TensorFlow functions


def keras_nll_loss(y_true, y_pred):
    mean_true, log_s_true = y_true[:, 0], None
    mean_pred, log_s_pred = y_pred[:, 0], y_pred[:, 1]
    # ensure correct data type
    mean_true = K.cast(mean_true, tf.float32)
    mean_pred = K.cast(mean_pred, tf.float32)
    log_s_pred = K.cast(log_s_pred, tf.float32)
    # calculate loss
    nll = log_s_pred + K.square(mean_true - mean_pred) / K.exp(2. * log_s_pred) / 2.
    return K.sum(nll, axis=-1)
```

MLE using ANN: Example

$$y \sim \mathcal{N}\left(\mu = x^2, \sigma = 0.001 + 0.1\sin(5x)\right)$$



FCNN



FCNN: 1 input, HL1 100 ReLU, HL2 50 ReLU, 2 linear outputs (for μ and log(σ))

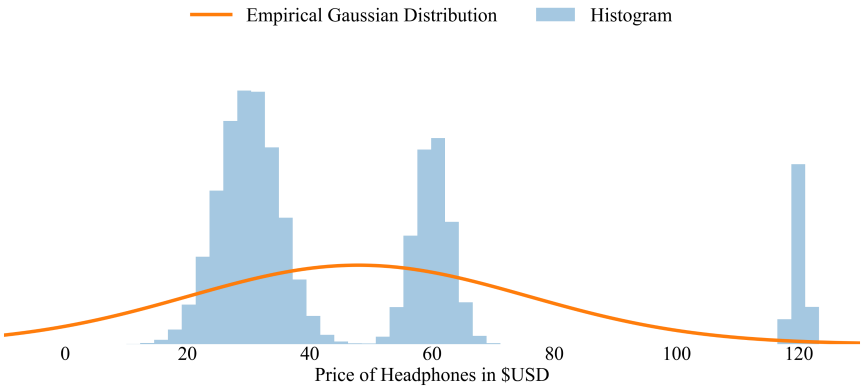
Warning: No CV was used here — the model might overfit!

What if the modelled distribution is not Gaussian?

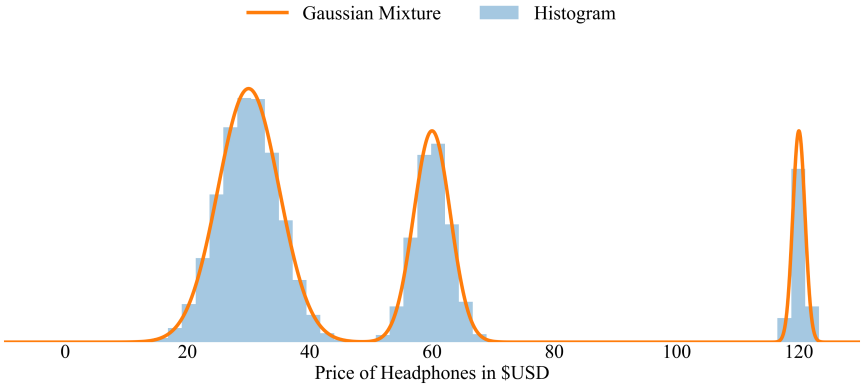
- **Unimodal distributions (e.g., log-normal, Gamma, etc.):**
  - Specify PDF in a parametric form
  - Define a loss function as a log-likelihood and learn parameters

- **Multimodal distributions → use mixture models**
  - Almost the same!
  - Define the number (hyperparameter) and parametric form of the distributions
  - Define a loss function as a log-likelihood and learn parameters of the distributions together with a mixing coefficient

Multimodal distributions: Mixture of Gaussians



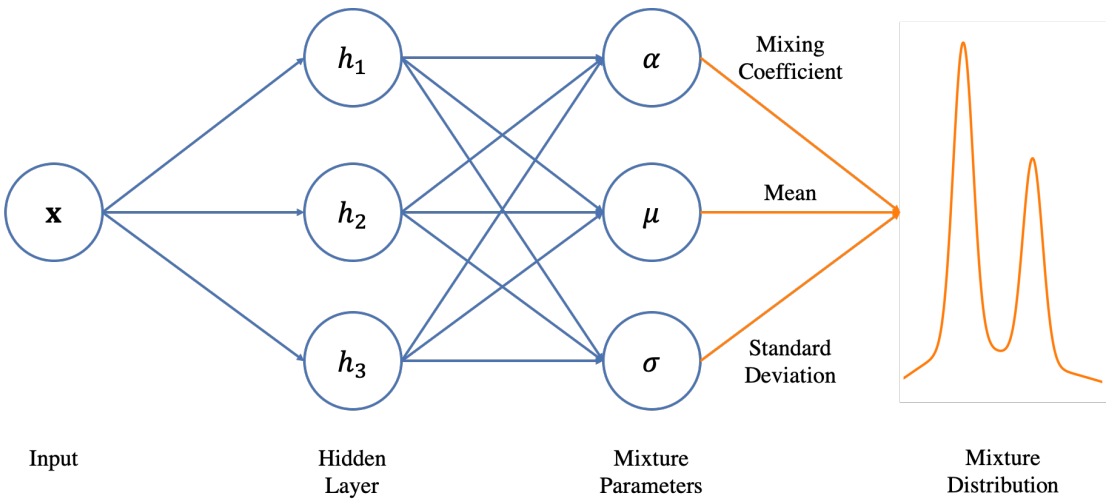
1 Gaussian



3 Gaussians

https://towardsdatascience.com/a-hitchhikers-guide-to-mixture-density-networks-76b435826cca

Mixture Density Network (MDN)



$$p(y|\mathbf{x}) = \sum_{c=1}^{C} \underbrace{\alpha_c(\mathbf{x})}_{\text{Mixing coefficient}} \mathcal{D}\left(y|\underbrace{\lambda_{1,c}(\mathbf{x}), \lambda_{2,c}(\mathbf{x}), \ldots}_{\text{Distribution parameters}}\right)$$

https://towardsdatascience.com/a-hitchhikers-guide-to-mixture-density-networks-76b435826cca

Mixture Density Network (MDN): Example

1 hidden layer NN

$$h_1(\mathbf{x}) = max(\mathbf{W_1}^\top \mathbf{x} + \mathbf{b_1}, 0)$$

$$\alpha(\mathbf{x}) = softmax(\mathbf{W}_\alpha^\top h_1(\mathbf{x}) + \mathbf{b}_\alpha)$$

Mixing coefficient

$$\lambda_1(\mathbf{x}) = (\mathbf{W}_{\lambda_1}^\top h_1(\mathbf{x}) + \mathbf{b}_{\lambda_1})$$

$$\lambda_2(\mathbf{x}) = (\mathbf{W}_{\lambda_2}^\top h_1(\mathbf{x}) + \mathbf{b}_{\lambda_2})$$

Distribution parameters

$$\mu(\mathbf{x}) = \lambda_1(\mathbf{x})$$

$$\sigma(\mathbf{x}) = ELU(\lambda_2(\mathbf{x})) + 1$$

Gaussian parameters

Loss function

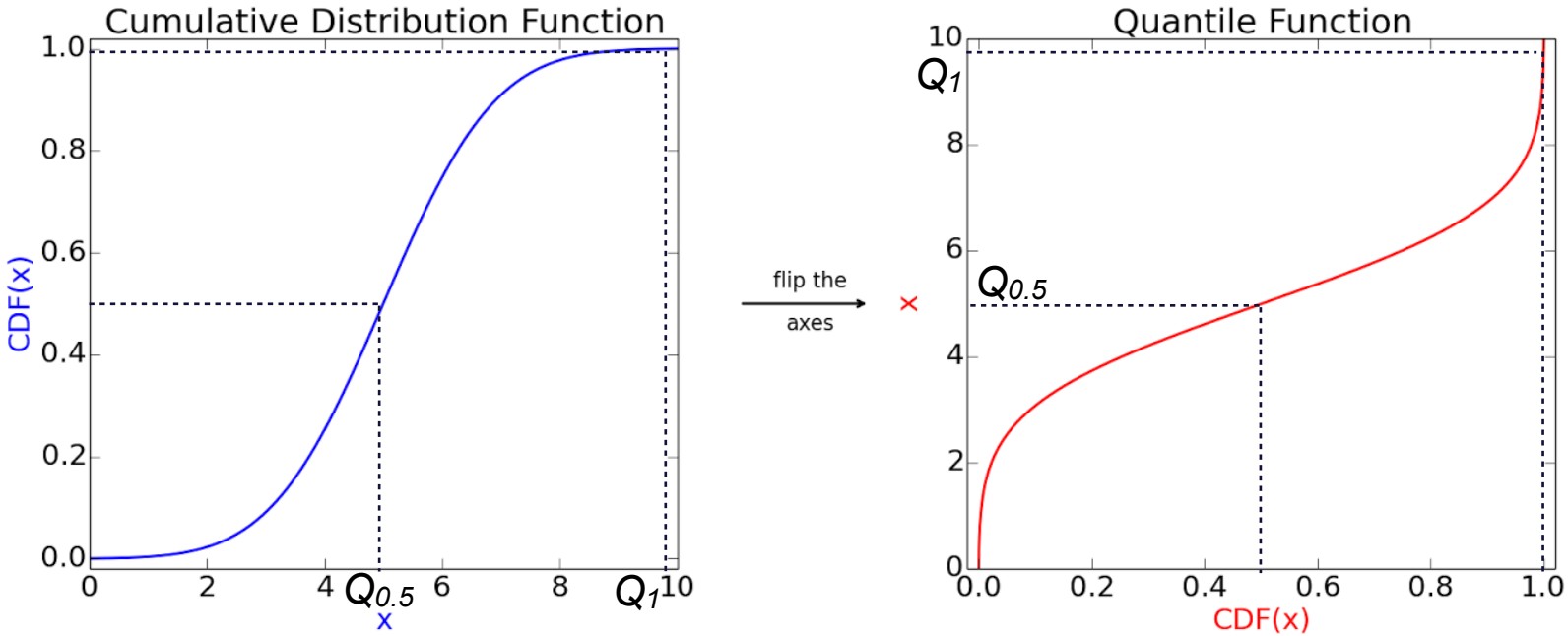
$$-\log p(y|X) = -\sum_{\mathbf{x}} \log \sum_{c=1}^{C} \alpha_c(\mathbf{x}) \underbrace{\mathcal{D}\left(y|\lambda_{1,c}(\mathbf{x}), \lambda_{2,c}(\mathbf{x}), \ldots\right)}_{\text{Gaussian PDF}} =$$

```
gm = tfd.MixtureSameFamily(
    mixture_distribution=tfd.Categorical(probs=alpha),
    components_distribution=tfd.Normal(
        loc=mu,
        scale=sigma))

log_likelihood = gm.log_prob(tf.transpose(y))
```

Check https://towardsdatascience.com/a-hitchhikers-guide-to-mixture-density-networks-76b435826cca

Defining CDF: Quantiles

Quantile function is inverse of CDF



Median = $Q_{0.5}$: 50% of data are less or equal $Q_{0.5}$, i.e. $\Pr(x \leq Q_{0.5}) = 0.5$

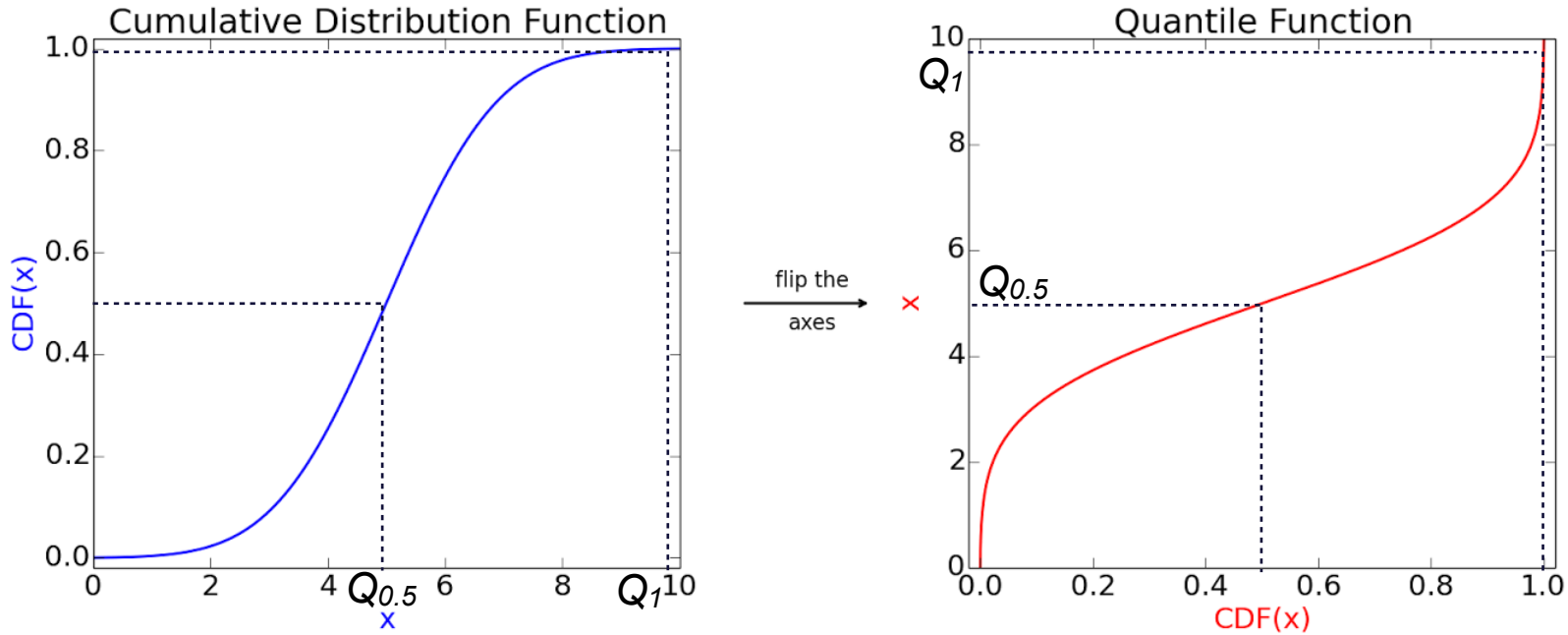
Q_q : q of data are less than Q_q , i.e. $\Pr(x \leq Q_q) = q$

$Q_1 = \max(x)$: 100% of data are less or equal than Q_1 , i.e. $\Pr(x \leq Q_1) = 1$

http://maximum-entropy-blog.blogspot.com/p/glossary.html#quantile-function

Defining CDF: Quantiles

Quantile function is inverse of CDF



Median = $Q_{0.5}$: 50% of data are less or equal $Q_{0.5}$, i.e. $\Pr(x \leq Q_{0.5}) = 0.5$

Q_q : q of data are less than Q_q , i.e. $\Pr(x \leq Q_q) = q$

$Q_1 = \max(x)$: 100% of data are less or equal than Q_1 , i.e. $\Pr(x \leq Q_1) = 1$

To learn Q_q we will use quintile loss (a.k.a. "tilted loss")

http://maximum-entropy-blog.blogspot.com/p/glossary.html#quantile-function

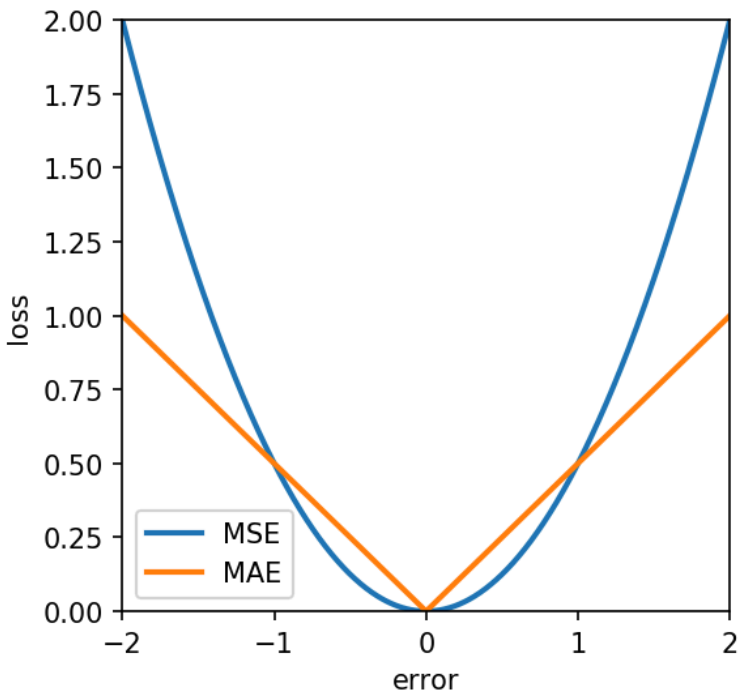
Mean Squared Error vs Mean Absolute Error

$$\epsilon = y - \hat{y}$$

$$\mathrm{SE}(\epsilon) = \frac{\epsilon^2}{2}$$

$$\mathrm{AE}(\epsilon) = \frac{|\epsilon|}{2}$$

$$\text{Mean Loss} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{Loss}(\epsilon_i)$$

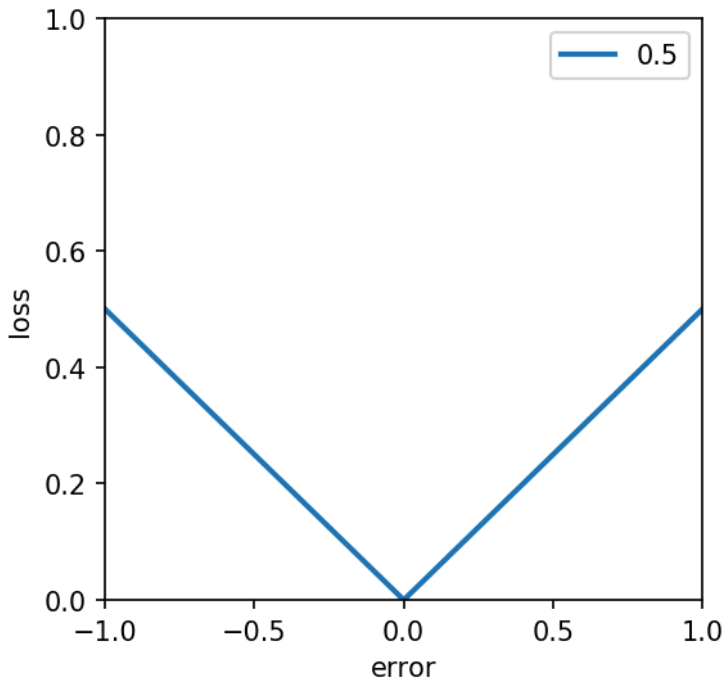


MSE optimises for mean (Gaussian noise assumption)
MAE optimises for median (Laplacian noise assumption)

Sidenote: Mean is more susceptible to outliers, median is more robust

Tilted loss

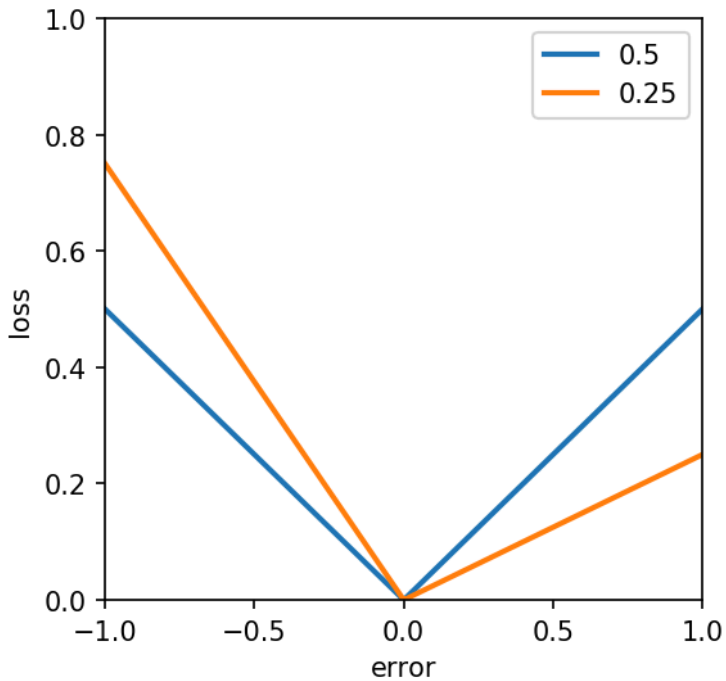
$$\mathrm{QL}\left(\epsilon \,|\, q\right) = \begin{cases} q\,|\,\epsilon\,|, \epsilon > 0 \\ (1-q)\,|\,\epsilon\,|, \epsilon < 0 \end{cases} = \max\left(q\epsilon, (q-1)\epsilon\right)$$



$$\text{Median} = Q_{0.5} : \mathrm{QL}\left(\epsilon \,|\, q = 0.5\right) = \mathrm{AE}(\epsilon) = \frac{|\,\epsilon\,|}{2}$$

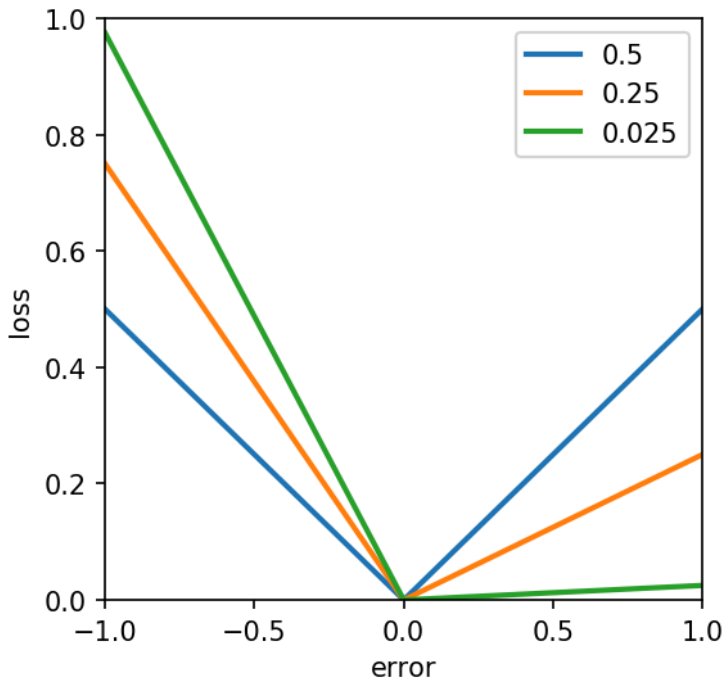
Tilted loss

$$\mathrm{QL}\left(\epsilon \,|\, q\right) = \begin{cases} q\,|\,\epsilon\,|, \epsilon > 0 \\ (1-q)\,|\,\epsilon\,|, \epsilon < 0 \end{cases} = \max\left(q\epsilon, (q-1)\epsilon\right)$$



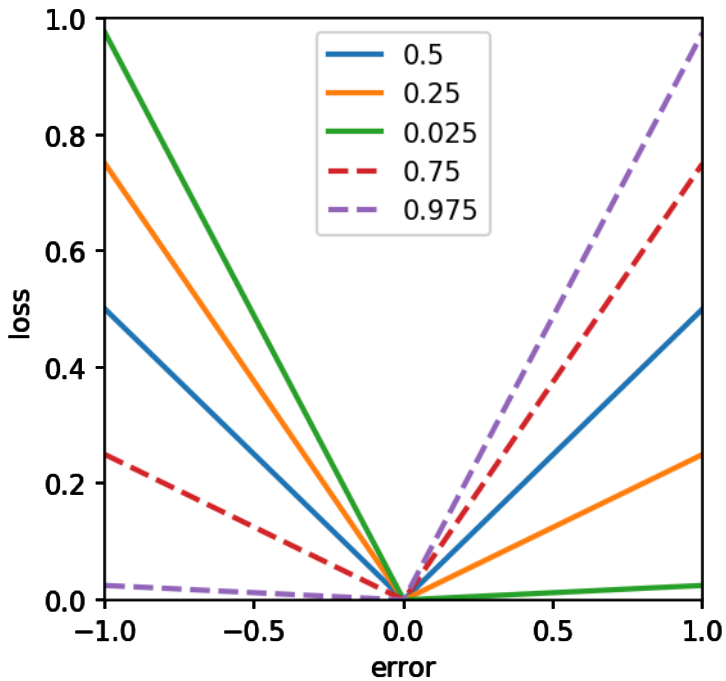
Tilted loss

$$\mathrm{QL}\left(\epsilon \,|\, q\right) = \begin{cases} q\,|\,\epsilon\,|, \epsilon > 0 \\ (1-q)\,|\,\epsilon\,|, \epsilon < 0 \end{cases} = \max\left(q\epsilon, (q-1)\epsilon\right)$$



Tilted loss

$$\mathrm{QL}\left(\epsilon \,|\, q\right) = \begin{cases} q\,|\,\epsilon\,|\,, \epsilon > 0 \\ (1-q)\,|\,\epsilon\,|\,, \epsilon < 0 \end{cases} = \max\left(q\epsilon, (q-1)\epsilon\right)$$



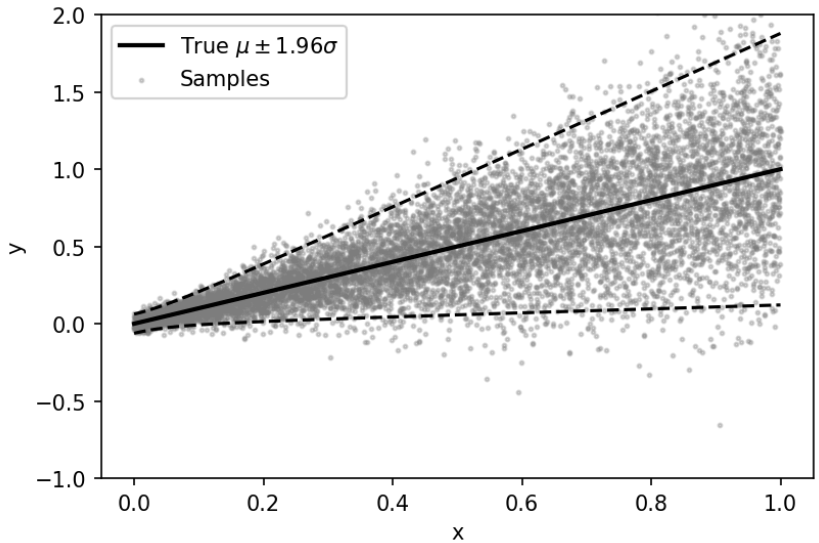
Quantile regression steps

1. Specify a quantile model, $\hat{y}(x;\beta)$
2. Choose q and optimise MQL($\hat{y}(x;\beta)$ | q) to find β

$$\mathrm{MQL} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{QL}(\epsilon_i)$$

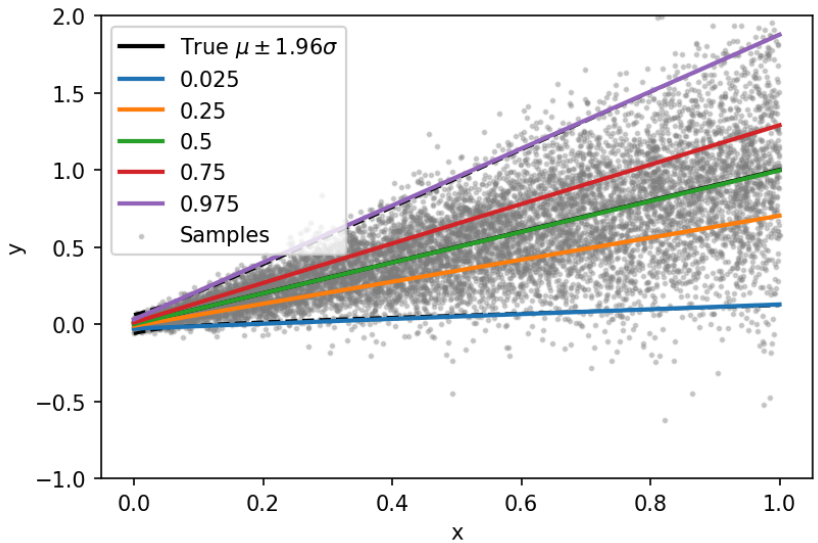
Quantile regression example

$$y \sim \mathcal{N}\left(\mu = x, \sigma = \sqrt{0.001 + 0.2x^2}\right)$$



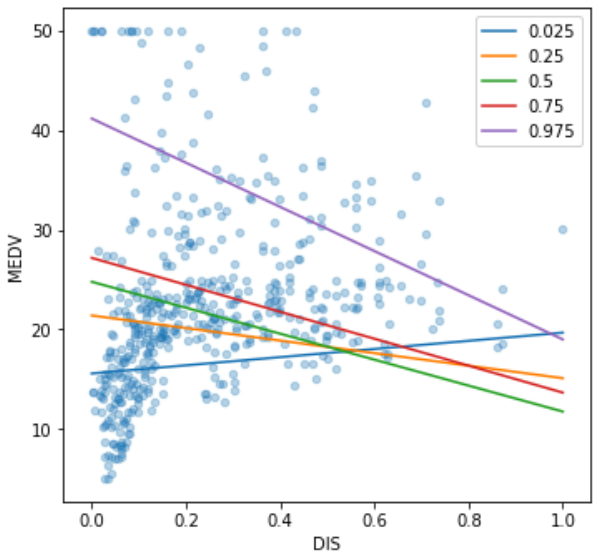
Model

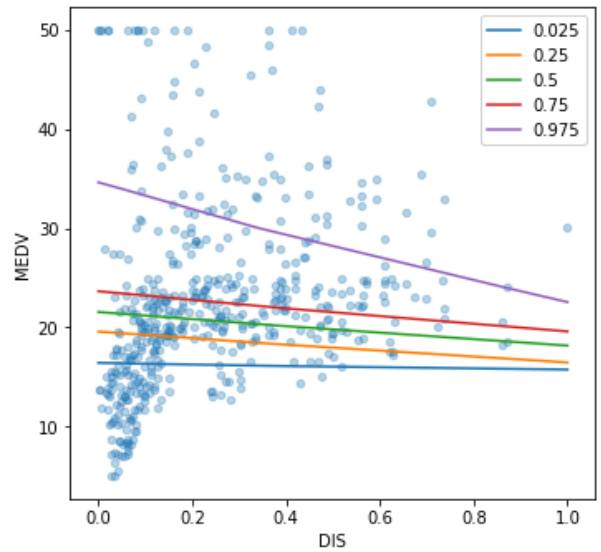
$$\hat{y} = \beta_0 + \beta_1 x$$



Quantile crossings problem

- Number of quantile crossings should be zero, e.g., $\hat{y}_{0.5}(x) < \hat{y}_{0.6}(x), \forall x$

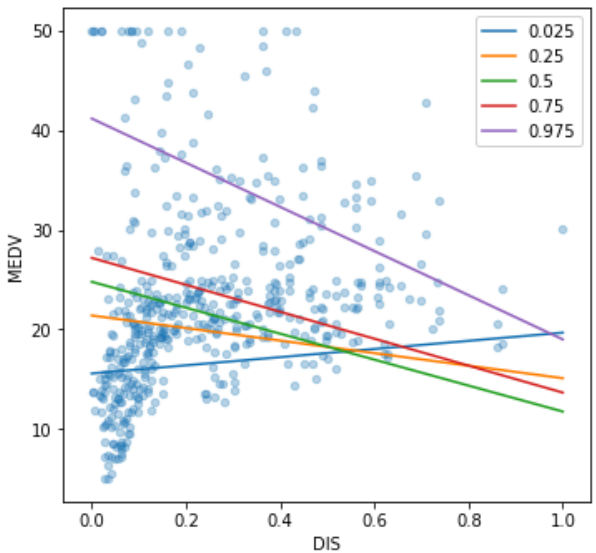


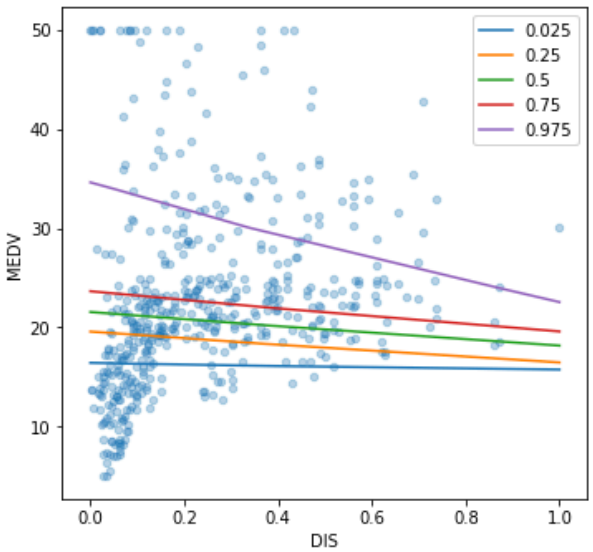


Quantile crossings problem

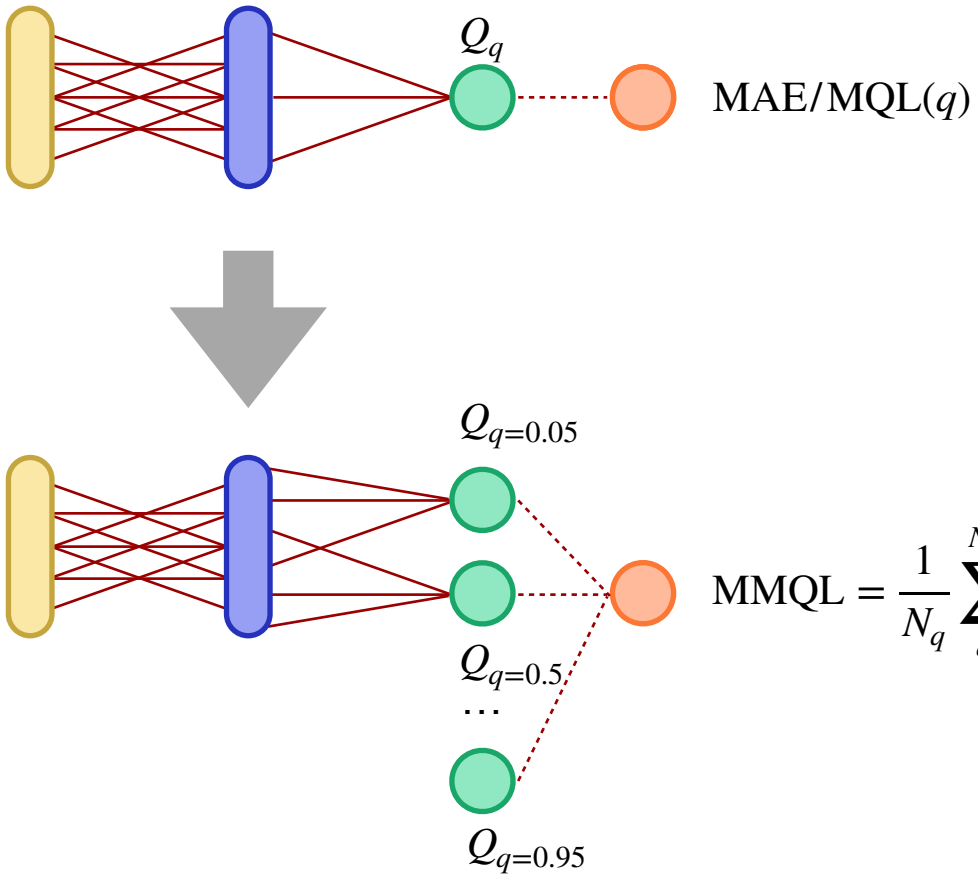
- Number of quantile crossings should be zero, e.g., $\hat{y}_{0.5}(x) < \hat{y}_{0.6}(x), \forall x$
- Many solutions have been proposed, .e.g., “Restricted Regression Quantiles” (RRQ)
- However, multi-output models are usually good enough in practice

- **ANNs are suitable again!**





Quantile regression using ANN



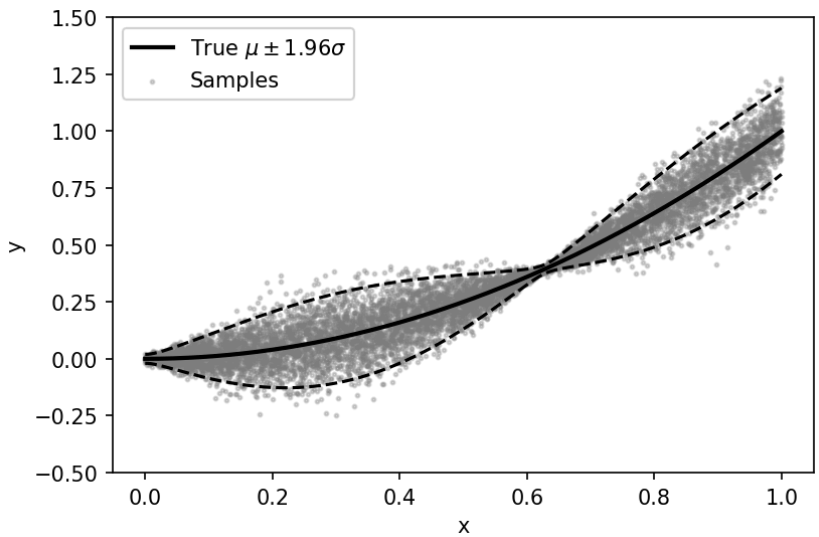
$$\text{MMQL} = \frac{1}{N_q} \sum_{q}^{N_q} \text{MQL}(q)$$

Custom tilted loss in Keras

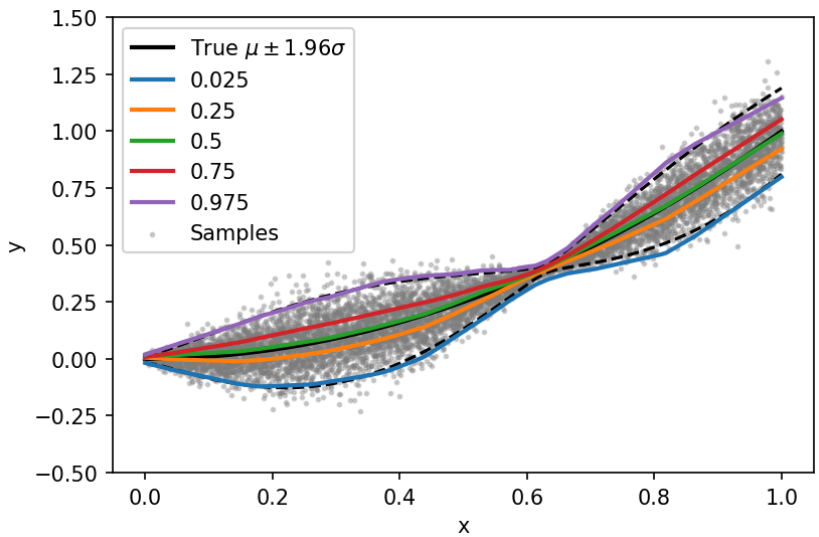
```
from keras import backend as K # TensorFlow functions


def quantile_loss_nn(y_true, y_pred):
    loss = 0
    for q_i, q in enumerate(quantiles):
        e = y_true - y_pred[:, q_i:q_i+1]
        loss += K.mean(K.maximum(q*e, (q-1)*e))
    return loss

model.compile(optimizer='rmsprop', loss=quantile_loss_nn, metrics=['mae'])
```

Quantile regression using ANN: Example

$$y \sim \mathcal{N}\left(\mu = x^2, \sigma = 0.001 + 0.1\sin(5x)\right)$$



FCNN



FCNN: 1 input, HL1 100 ReLU, HL2 50 ReLU, 6 linear outputs

Warning: No CV was used here — the model might overfit!

Exercises

- “Probability calibration in classification.ipynb”
- “Uncertainty in regression.ipynb”

Recommended reading

- “How and When to Use a Calibrated Classification Model with scikit-learn” https://machinelearningmastery.com/calibrated-classification-model-in-scikit-learn/
- Sklearn documentation, “1.16. Probability calibration” https://scikit-learn.org/stable/modules/calibration.html
- “On Calibration of Modern Neural Networks” https://arxiv.org/abs/1706.04599
- “A Hitchhiker’s Guide to Mixture Density Networks” https://towardsdatascience.com/a-hitchhikers-guide-to-mixture-density-networks-76b435826cca
- “Deep Quantile Regression” https://towardsdatascience.com/deep-quantile-regression-c85481548b5a