Practical classifier design

Gustav Eje Henter

ghe@kth.se

Division of Speech, Music and Hearing (TMH),
School of Electrical Engineering and Computer Science (EECS),
KTH Royal Institute of Technology, Stockholm, Sweden

2020-05-13

Perspective

Course book overview:

1. Preliminary topics
2. Optimal decision theory
3. Practical classifier design
4. Sequence classification and hidden Markov models
5. Bayesian learning

Why is this interesting?

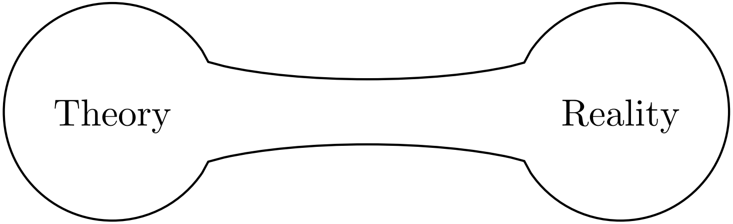
You already know optimal decision theory, how to make optimal decisions. Why do you need to know more?

The problem is that the theory requires you to know the conditional class probabilities $P(C = i \mid \boldsymbol{X} = \boldsymbol{x})$. In practice, these are typically unknown!

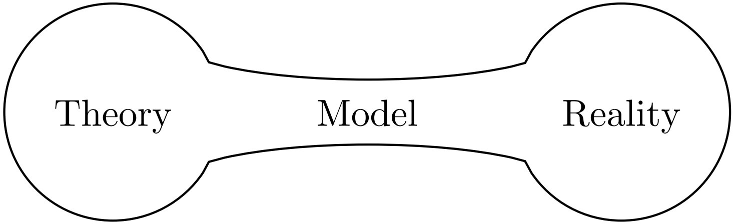
Visual illustration

Theory
Reality

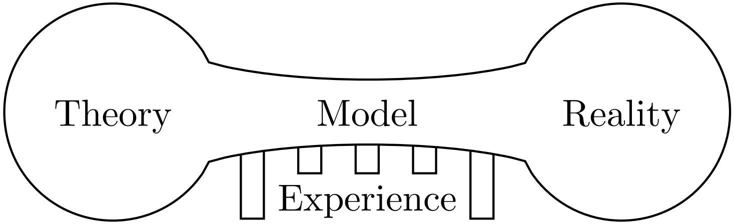
Visual illustration



Visual illustration



Visual illustration



Why is this interesting?

You already know optimal decision theory, how to make optimal decisions. Why do you need to know more?

The problem is that the theory requires you to know the conditional class probabilities $P(C = i \mid \boldsymbol{X} = \boldsymbol{x})$. In practice, these are typically unknown!

We need to create a bridge between reality and decision theory. This is called a *model*. It has to be based on assumptions, but can also use data.

- To create a good model, we rely on experience

Classifier design

Today's topic: **How to design good classifiers**

- . . . to the best of my knowledge, of course
- Much of today's content I have learned from teaching the course

The lecture has two parts:

1. The standard classifier design procedure
2. Troubleshooting and improving classifiers

Classifier design

Today's topic: **How to design good classifiers**

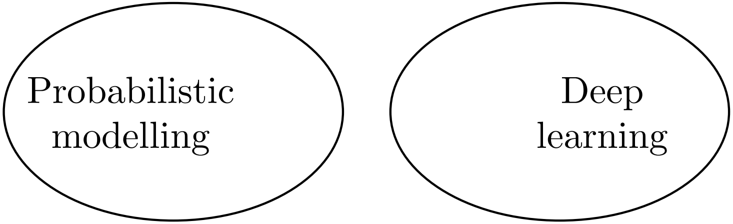
- . . . to the best of my knowledge, of course
- Much of today's content I have learned from teaching the course

The lecture has two parts:

1. The standard classifier design procedure
2. Troubleshooting and improving classifiers

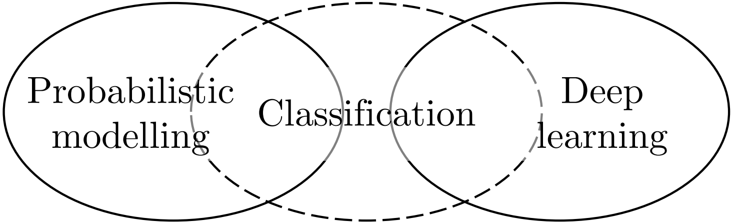
A common misconception

Many university courses can give an impression that probabilistic classifiers and deep-learning-based classifiers are two separate areas:



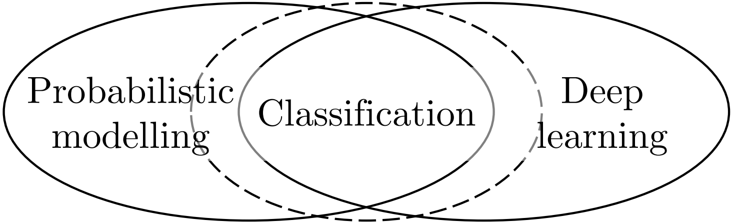
A common misconception

Many university courses can give an impression that probabilistic classifiers and deep-learning-based classifiers are two separate areas:



Probability and deep learning

In reality, most classifiers used today are built on probability (for the loss function) *and* deep learning (for the architecture), together:



Different terminologies, same result

Probability and deep learning use different words for the same thing:

- Deep learning: "Use a softmax output layer and train the model to minimise the cross-entropy loss"
- Probabilistic modelling: "Assume a categorical distribution and estimate parameters using maximum likelihood"

The approaches described above are *mathematically equivalent!*

- The softmax outputs of the trained deep-learning model can directly be interpreted as a-posteriori class probabilities $P\left(C = i \,\middle|\, \boldsymbol{X} = \boldsymbol{x}\right)$

Classifier types

- Probabilistic methods
  - *Generative:* Describe the joint label-feature distribution

$$\begin{aligned} P(C = i, \boldsymbol{X} = \boldsymbol{x}) &= P(\boldsymbol{X} = \boldsymbol{x} \mid C = i)\, P(C = i) \\ &= P(C = i \mid \boldsymbol{X} = \boldsymbol{x})\, P(\boldsymbol{X} = \boldsymbol{x}) \end{aligned}$$

  - *Discriminative:* Only model conditional class probabilities

$$P(C = i \mid \boldsymbol{X} = \boldsymbol{x})$$

- Non-probabilistic methods
  - Discriminative: Only model the decision function $d(\boldsymbol{x}) = \widehat{c}(\boldsymbol{x})$
    - Random variation is not explicitly accounted for
    - Often a special case of probabilistic methods
    - Model assumptions are not visible upfront

Generative or discriminative?

Generative classifiers rely on Bayes' law

$$P(C = i \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{P(\boldsymbol{X} = \boldsymbol{x} \mid C = i)\, P(C = i)}{P(\boldsymbol{X} = \boldsymbol{x})}$$

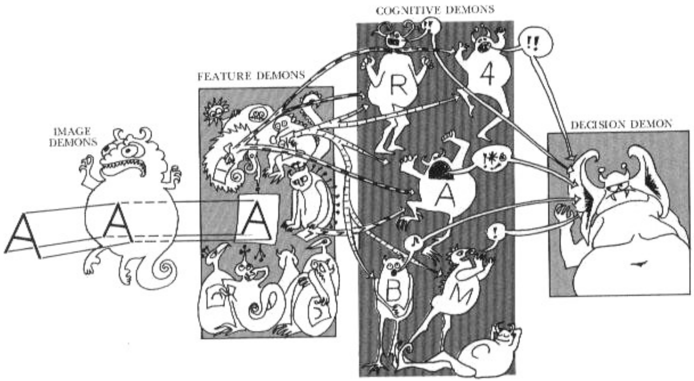
- This "inverts" our model of data-generation to compute class probabilities
- Proposing good models for $P(\boldsymbol{X} = \boldsymbol{x} \mid C = i)$ is hard
  - Classic pattern recognition tends to rely on simple distributions that make optimisation easy, but this tends to limit classifier accuracy

Discriminative classifiers

- Do not require inversion
- Example: Deep neural networks (DNNs) with softmax output layers
  - This probabilistic and deep methodology is often state-of-the-art today

How a classifier is built

Pandemonium architecture (1959): An early cognitive model



(Image by Lindsay & Norman, 1972)

Probabilistic classifiers

Probabilistic classification systems have three key parts:

1. One or more *feature extractors* $\boldsymbol{x} = \boldsymbol{f}\left(\widetilde{\boldsymbol{x}}\right)$
2. *Probabilistic model(s)* of the label/feature distribution

$$P\left(C = i, \boldsymbol{X} = \boldsymbol{x}\right) = P\left(\boldsymbol{X} = \boldsymbol{x} \,|\, C = i\right) P\left(C = i\right)$$

3. A discriminant-function based *decision rule* (e.g., MAP)

$$g_i\left(\boldsymbol{x}\right) = g_i'\left(P\left(C = i \,|\, \boldsymbol{X} = \boldsymbol{x}\right)\right)$$

To build a classification system, we need these three components

Probabilistic classifier design

Traditional, non-deep classifier design has several stages:

1. Create a feature extractor
2. Propose a parametric model
3. Estimate model parameters from data
4. Choose a decision rule
5. Evaluate performance

Probabilistic classifier design

Traditional, non-deep classifier design has several stages:

1. Create a feature extractor
2. Propose a parametric model
3. Estimate model parameters from data
4. Choose a decision rule
5. Evaluate performance

Feature extraction

Feature extraction transforms the raw input data $\widetilde{\boldsymbol{x}}$ to a form $\boldsymbol{x}$ useful for classification

- Information management
  - Concentrate relevant information
  - Remove irrelevant information (*dimensionality reduction*)
- Separate unrelated aspects of the data
- Create data that has a well-behaved distribution $P(\boldsymbol{X} = \boldsymbol{x} \mid C = i)$

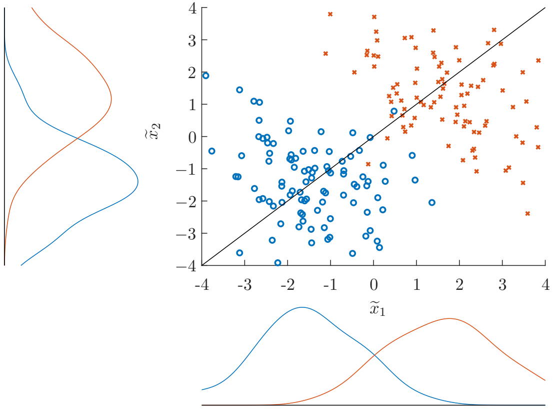
It's OK to have variation and uncertainty!

- The idea is not to solve the problem, but to make it easier for the classifier to solve the problem

Dimensionality reduction example

Dimensionality reduction can be used to create low dimensional data $\boldsymbol{x}$ that contains most information from the original input $\widetilde{\boldsymbol{x}}$

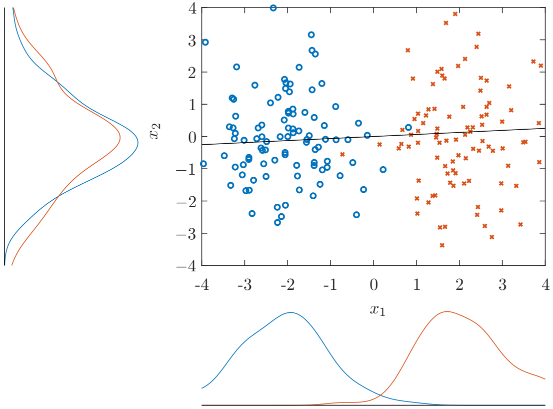
- Example: Principal component analysis (PCA), which retains most of the variance of the data



Dimensionality reduction example

Dimensionality reduction can be used to create low dimensional data $\boldsymbol{x}$ that contains most information from the original input $\widetilde{\boldsymbol{x}}$

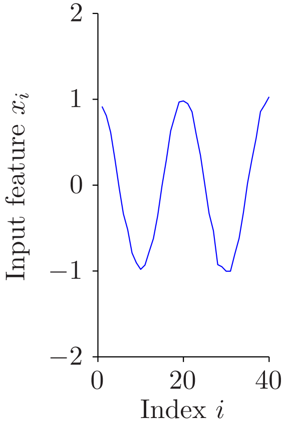
- Example: Principal component analysis (PCA), which retains most of the variance of the data

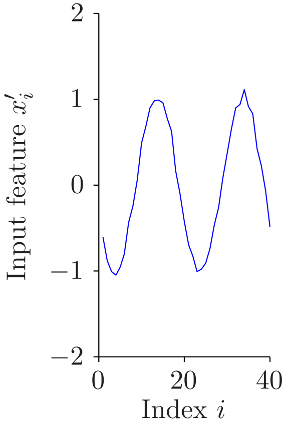


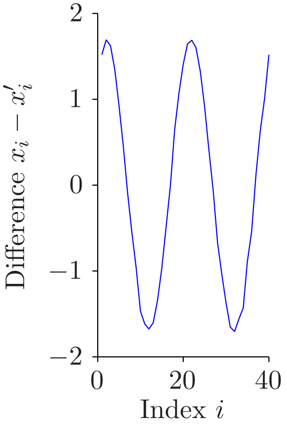
Information separation example

Fourier analysis extracts and separates different frequencies in the data

- Useful in speech or music recognition, for instance
- Example: Tones in noise (raw waveforms)



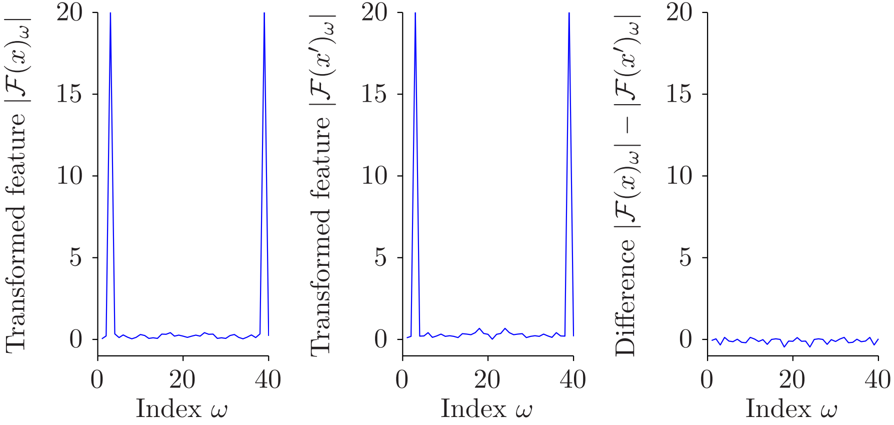




Information separation example

Fourier analysis extracts and separates different frequencies in the data

- Useful in speech or music recognition, for instance
- Example: Tones in noise (Discrete Fourier transform power spectra)



Feature extraction principles

To design the extractor, we often use our domain knowledge about the problem

- Inspiration from nature/biology is common
- Example: Perceptual (auditory) features for speech recognition

The hidden layers in a deep neural network can be seen as a learned feature extractor that feeds into a very simple classifier

- But an initial, non-learned feature extractor is still common
- Bottleneck layers are like dimensionality reduction

Probabilistic classifier design

Classifier design has several stages:

1. Create a feature extractor
2. Propose a parametric model
3. Estimate model parameters from data
4. Choose a decision rule
5. Evaluate performance

Model building

Since we do not know the true data distribution, we have to make assumptions

- Assume the features of class i follow a distribution $\widehat{f}_i$:

$$P\left(\boldsymbol{X} = \boldsymbol{x} \mid C = i\right) = \widehat{f}_i\left(\boldsymbol{x}\right)$$

- Think: How do we believe class-conditional features are distributed?
- Example: Use a non-negative distribution for non-negative data
- Example: Features are sums of many independent terms $\Rightarrow$ Gaussian by central limit theorem

$$\begin{aligned} P\left(\boldsymbol{X} = \boldsymbol{x} \mid C = i\right) &\approx \widehat{f}_i\left(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) \\ &= \frac{1}{(2\pi)^{\frac{D}{2}} \det \boldsymbol{\Sigma}_i} \exp\left(-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)^T \boldsymbol{\Sigma}_i^{-1}\left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)\right) \end{aligned}$$

The model is an assumption

Again: The expression for $\widehat{f_i}$ is *only an assumption* of $P\left(\boldsymbol{X} = \boldsymbol{x} \mid C = i\right)$

- Ideally, $\widehat{f_i}$ should match the true behaviour of the features
- This is impossible in practice
- It is most important to describe the aspects of the features that distinguish different classes

Parametric models

Fortunately, we do not need to specify every aspect of the probabilistic model manually

- Typically, we assume a *parametric* functional form $\widehat{f_i}(\boldsymbol{x} \mid \boldsymbol{\theta}_i)$
  - Different classes can use the same form, but different parameters $\boldsymbol{\theta}_i$
- The parameters $\boldsymbol{\theta}_i$ represent unknown aspects of $P(\boldsymbol{X} = \boldsymbol{x} \mid C = i)$
  - Example: Mean and/or variance if $\widehat{f_i}$ is Gaussian
- The unknown $\boldsymbol{\theta}_i$ can be determined using data from the problem!

Probabilistic classifier design

Classifier design has several stages:

1. Create a feature extractor
2. Propose a parametric model
3. Estimate model parameters from data
4. Choose a decision rule
5. Evaluate performance

Learning from data

By estimating the unknown $\boldsymbol{\theta}_i$ we fill in the blanks in the model $\widehat{f_i}\left(\boldsymbol{x} \mid \boldsymbol{\theta}_i\right)$

- This is known as *parameter estimation* or *model fitting* or *training*
- This approach is very powerful since it learns from observations

To train the models, we require training examples – feature vectors paired with their corresponding class label

- $\mathcal{D} = \{(\boldsymbol{x}_n,\, c_n)\}_{n=1}^{N}$ is the set of (independent) labelled training examples
- $\mathcal{D}_i$ is the subset of training examples from class i
- $\boldsymbol{\theta} = \{\boldsymbol{\theta}_i\}$ is the set of model parameters over all classes

Parameter-estimation techniques

- The traditional approach is *maximum likelihood estimation* (MLE)
  - Find the parameters $\widehat{\boldsymbol{\theta}}_{\mathrm{ML}}$ for which the data is most (log-)probable under the model:

$$\widehat{\boldsymbol{\theta}}_{\mathrm{ML}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \widehat{P}\left(\mathcal{D} \,|\, \boldsymbol{\theta}\right) = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \widehat{f}_{c_n}\left(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}_{c_n}\right)$$

- If we have some prior knowledge about $\boldsymbol{\theta}$, we can insert it as a prior $\widehat{f}_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}\right)$, and use *maximum a-posteriori* (MAP) estimation:

$$\begin{aligned}\widehat{\boldsymbol{\theta}}_{\mathrm{MAP}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \widehat{P}\left(\boldsymbol{\theta} \,|\, \mathcal{D}\right) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \left(\log \widehat{f}_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}\right) + \sum_{n=1}^{N} \log \widehat{f}_{c_n}\left(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}_{c_n}\right)\right)\end{aligned}$$

  - $\log \widehat{f}_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}\right)$ acts as a kind of regularisation term

MLE for Gaussian distribution

Example: Find $\widehat{\mu}_{\mathrm{ML}}$ that maximises the (log) likelihood of a scalar normal distribution with known standard deviation σ, given data $\{x_1, \ldots, x_N\}$

Solution: The log likelihood is

$$\begin{aligned}\mathcal{L}\left(\mu \,\middle|\, x_1, \ldots, x_N\right) &= \log\left(\prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_n-\mu)^2}{2\sigma^2}\right)\right) \\ &= -\frac{N}{2}\log(2\pi) - N\log\sigma - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2\end{aligned}$$

which is equivalent to minimising the mean squared error (MSE) and has its optimum at the sample mean

$$\widehat{\mu}_{\mathrm{ML}} = \overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

Probabilistic classifier design

Classifier design has several stages:

1. Create a feature extractor
2. Propose a parametric model
3. Estimate model parameters from data
4. Choose a decision rule
5. Evaluate performance

Decision rule

To make a decision, we use the Bayes minimum-risk rule

- This is the optimal choice for minimising expected cost
- We need to know or guess the costs of different misclassifications
- The most general cost function depends on the true class c_n, the guess $\widehat{c}_n(\boldsymbol{x})$, and on $\boldsymbol{x}$
- Often all errors are weighted equally, in which case we minimise classification error rate

The system can now perform classification!

- Done?

Probabilistic classifier design

Classifier design has several stages:

1. Create a feature extractor
2. Propose a parametric model
3. Estimate model parameters from data
4. Choose a decision rule
5. Evaluate performance

Performance evaluation

Only a fool would use a classifier before checking how well it performs!

- If $\widehat{f}_i\left(\boldsymbol{x}\,\middle|\,\widehat{\boldsymbol{\theta}}_i\right) \approx P\left(\boldsymbol{X} = \boldsymbol{x} \mid C = i\right)$, the classifier is likely be close to minimum cost for these features
- We cannot be sure this holds; many assumptions have been made!

Bad idea: Look at how the classifier performs for the training data

- Example: Compute the estimated probability of $\mathcal{D}$ (the likelihood)
- Example: Compute the average cost over the training data
  - Special case: Frequency of misclassifications
- On training data, the cost is often misleadingly low!
  - The model was optimised to make this number look good

Held-out data

Better idea: Estimate performance on previously unseen data

If we cannot get new data, we can "fake it" using *held-out data*

- Prior to training, partition the data $\mathcal{D}$ into a training set $\mathcal{D}^{(train)}$, a validation set $\mathcal{D}^{(val)}$, and a test set $\mathcal{D}^{(test)}$
  1. Estimate parameters on the training set only
  2. Refine the approach (tune hyperparameters) on the validation set
  3. Estimate final performance on the test set once all models are fixed
- A more accurate number might be obtained by averaging over several different partitionings, so-called *cross-validation*
- The partitioning scheme can emulate the classifier use case
  - Example: Using the older data in $\mathcal{D}$ to predict the newer data
- The use of validation data is part of the course project

Iterative improvement

Now we have a classifier, and a method to estimate its performance

By tweaking the classifier and looking at how the estimated performance changes, we can iteratively improve the classifier

This is the road from a mere classifier, to a *good* classifier

- The faster you can iterate, the faster you can make progress
- Improving classifiers is part two of this lecture

Intermission

See you in 15 minutes!

Classifier design

Today's lecture:

1. The standard classifier design procedure
2. Troubleshooting and improving classifiers

Classifier performance hierarchy

Both what we do and how we do it limits the theoretically and practically optimal performance. The effects can be separated into a number of steps:

Level	**Description**	**Upper limit**	**Limiting factor**
Problem /task	"Predict C from $\boldsymbol{X}$"	Bayes optimal classifier	Class overlap, raw feature info
Model	"…using the parametric model $\widehat{f}_i(\boldsymbol{x} \mid \boldsymbol{\theta}_i)$"	Best classifier $\boldsymbol{\theta}^\star$	Model assumptions
Achievable	"…from data $\mathcal{D}$ in practice"	Best classifier we can find $\widehat{\boldsymbol{\theta}}$	Data sample, optimisation
Bottom line	"…without looking at $\boldsymbol{x}$"	Best a-priori classifier	-

In practice, we do not know the values of the different upper limits

- Performance as well as parameters can only be estimated

Classifier design choices

Design choices at every stage affect end performance:

1. Data acquisition
2. Feature extraction
3. Model building
4. Parameter estimation
5. Decision rule
6. Performance evaluation

The stages are not independent, and different methods call for different choices

- Example: Deep neural networks vs. classical pattern recognition

Classifier problems

Bad design choices can cause poor classification performance

Some common problem categories include:

1. Inappropriate features
2. Inappropriate model
3. Inappropriate optimisation
4. Inappropriate data

Classifier problems

Bad design choices can cause poor classification performance

Some common problem categories include:

1. Inappropriate features
2. Inappropriate model
3. Inappropriate optimisation
4. Inappropriate data

Feature engineering

In principle, the feature extractor should keep the information that helps classification, and remove information that doesn't help

- What this means in practice depends on the model
    - Feature engineering is of high importance in classical pattern recognition, but less so in deep learning

Both too little and too much information can be problematic:

- With too little information, we don't know enough for a good guess
    - The Bayes optimal classifier suffers
- For high-dimensional features, we don't know *where* the information is
    - This is closely related to the *curse of dimensionality*
    - A big problem for nearest-neighbour approaches
    - Seems less problematic for SVMs and, especially, DNNs

Classifier problems

Problem categories:

1. Inappropriate features
2. Inappropriate model
3. Inappropriate optimisation
4. Inappropriate data

Overfitting

It is often better to oversimplify than to overcomplicate

- At least at first
- Bias-variance trade-off

Complicated models require too much data to be estimated accurately

- They often fit the training data well, but predict poorly (*overfitting*)
- They have learned the noise, rather than the general tendencies
- This problem can often be detected using cross-validation
    - This simulates the behaviour on unseen data
    - Often more reliable than model-selection criteria such as AIC

Simple models are preferable

Only data aspects relevant to classification actually need to be modelled

- Example: HMM-based speech recognition models are notoriously unlike speech, but still perform OK

Good $\neq$ perfect

- "All models are wrong, but some are useful"
- Start from a simple model, and improve it as needed

In contrast, deep neural networks can often generalise well despite being highly complex

- Stop optimisation when validation-set performance begins to decrease (*early stopping*)

Mixture models

There's no simple rule for finding a good model for every situation

- Domain knowledge is often very helpful
- If we don't know what to expect, we may opt for a generic model that works in many cases and where the complexity can be adapted

Mixture models are very common for creating more flexible $\widehat{f_i}(\boldsymbol{x} \mid \boldsymbol{\theta}_i)$

1. Many phenomena are mixtures of different aspects
    - Example: Speech audio is a mixture of different voices
    - Example: Written texts are a mixture of different genres
2. Theoretically, GMMs can approximate *any* distribution
    - Many components may be necessary for a good approximation

Discriminative training

Interestingly, an inappropriate parametric model $\widehat{f}_i$ may be compensated for by choosing the parameters differently

- MLE is essentially optimal *if* the model is correctly specified
- If the data distribution does not agree with the assumptions, i.e.,

$$\widehat{f}_i\left(\boldsymbol{x}\,\middle|\,\boldsymbol{\theta}_i=\widehat{\boldsymbol{\theta}}\right)\neq P\left(\boldsymbol{X}=\boldsymbol{x}\mid C=i\right)\ \forall\widehat{\boldsymbol{\theta}},$$

  for some i, other parameter-estimation approaches can be better

- **Insight:** We should concentrate on classifying as well as possible, not on describing $P\left(\boldsymbol{X}=\boldsymbol{x}\mid C=i\right)$ as well as possible
  - It's the decision regions that matter!

Discriminative training approaches

- Try to optimise classification performance directly
  - *Minimum classification error* (MCE)

$$\widehat{\boldsymbol{\theta}}_{\mathrm{MCE}} = \underset{\boldsymbol{\theta}}{\mathrm{argmin}} \frac{1}{N} \sum_{n=1}^{N} I\left(\max_{i \neq c_n} \widehat{f}_{C \mid \boldsymbol{X}}\left(i \mid \boldsymbol{x}_n,\, \boldsymbol{\theta}\right) \geq \widehat{f}_{C \mid \boldsymbol{X}}\left(c_n \mid \boldsymbol{x}_n,\, \boldsymbol{\theta}\right) \right)$$

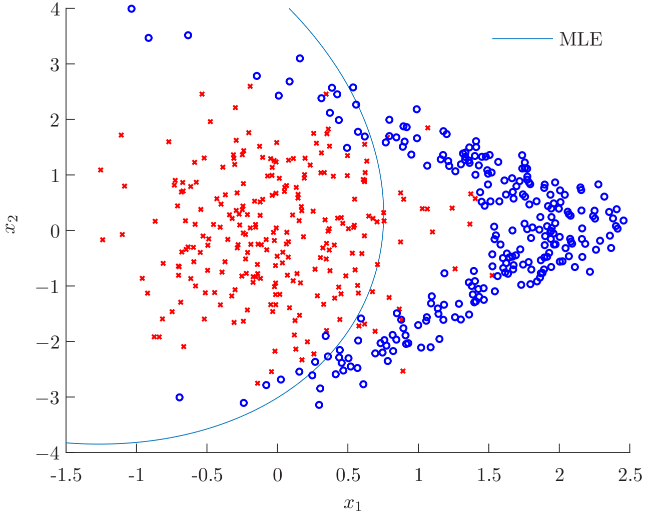
  - Not a continuous function (piecewise constant)
- Maximise the probability of the correct class
  - *Maximum mutual information* (MMI) training or *conditional maximum likelihood*

$$\widehat{\boldsymbol{\theta}}_{\mathrm{MMI}} = \underset{\boldsymbol{\theta}}{\mathrm{argmax}} \sum_{n=1}^{N} \log \widehat{f}_{C \mid \boldsymbol{X}}\left(c_n \mid \boldsymbol{x}_n,\, \boldsymbol{\theta}\right)$$

  - Also cumbersome to optimise classically
  - The typical approach with discriminative DNN classifiers

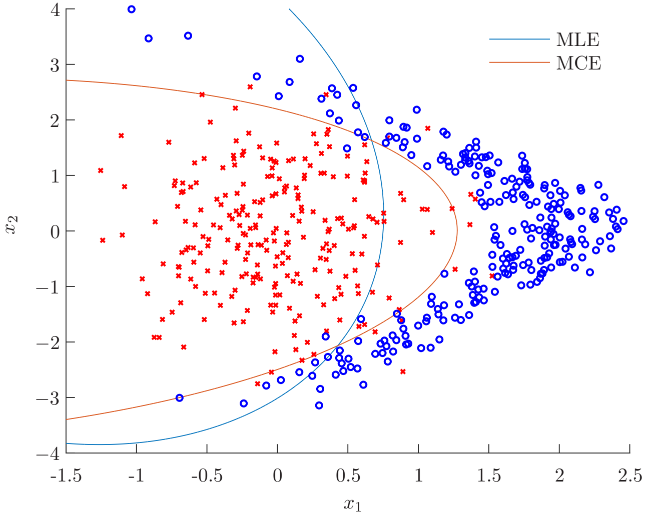
Comparison of approaches

Example: $\widehat{f}_i$ incorrectly assumes both classes are Gaussian
Maximum-likelihood fit (validation error 10.6%):



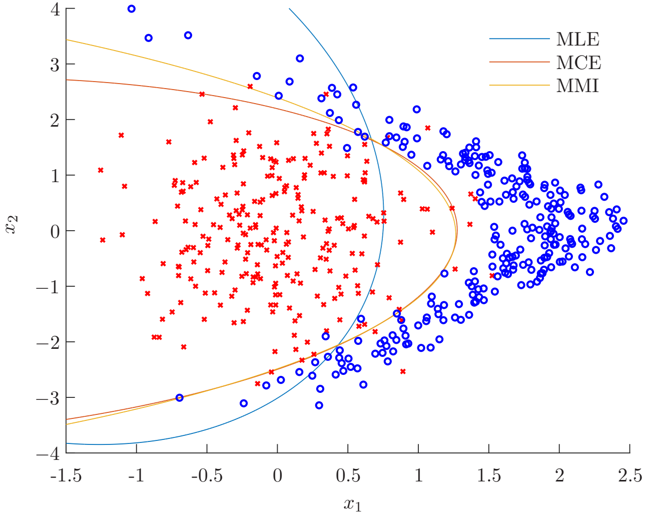
Comparison of approaches

Example: $\widehat{f}_i$ incorrectly assumes both classes are Gaussian
Minimum classification error (validation error 4.4%):



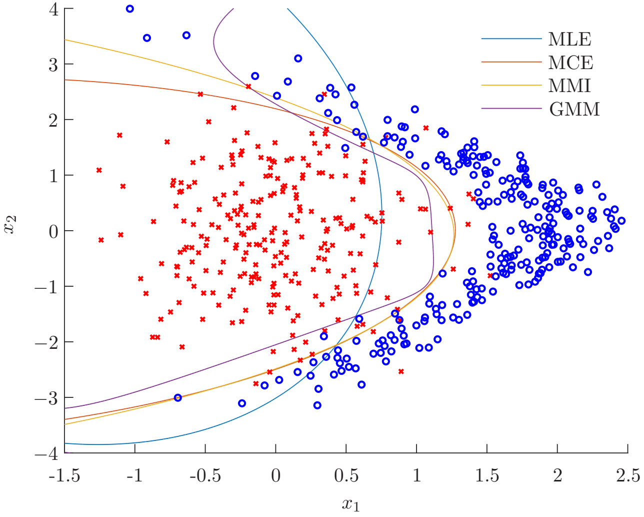
Comparison of approaches

Example: $\widehat{f}_i$ incorrectly assumes both classes are Gaussian
Maximum mutual information (validation error 4.4%):



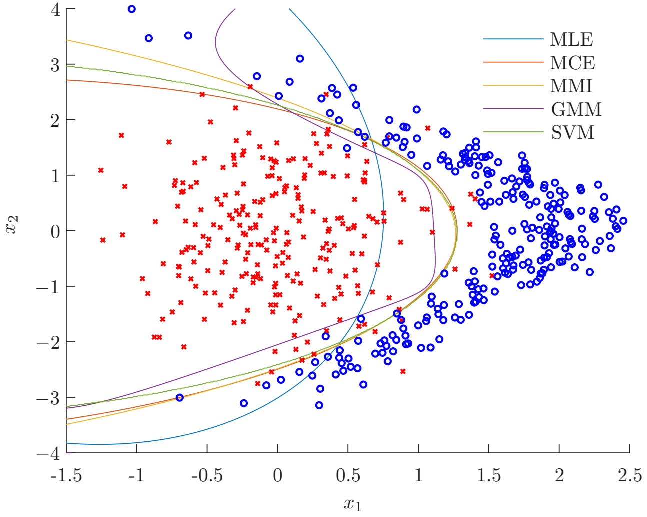
Comparison of approaches

Example: $\widehat{f}_i$ incorrectly assumes both classes are Gaussian
Maximum-likelihood GMM fit (validation error 5.5%):



Comparison of approaches

Example: $\widehat{f}_i$ incorrectly assumes both classes are Gaussian
Discriminative method (validation error 4.2%):



Comparison of approaches

Findings from the example summarised in table form:

Model or classifier	**Probabilistic?**	**Optimisation objective**	**Validation error**
Gaussian	✓	Maximum likelihood	10.6%
"	✓	Minimum classification error	4.4%
"	✓	Maximum mutual information	4.4%
GMM	✓	Maximum likelihood	5.5%
SVM	✗	Soft maximum margin	4.2%

Classifier problems

Problem categories:

1. Inappropriate features
2. Inappropriate model
3. Inappropriate optimisation
4. Inappropriate data

Optimisation issues

Optimisation is often the least well-understood aspect of model building

The more complex and more powerful the model class, the more demanding it tends to be to optimise

- A hierarchy of optimisation techniques ordered by growing computational demands, with example models that use them:
  - Analytic solution (e.g., single Gaussian) $<$ convex optimisation (e.g., SVMs) $<$ guaranteed-ascent iteration (e.g., GMMs) $<$ gradient-based optimisation (e.g., DNNs) $<$ gradient-free methods (e.g., simulators)
- In practice, the goal is not to perfectly optimise the objective function, but to find a model that generalises well
  - Example: Early stopping
- More computation often beats simpler, easy-to-optimise models
  - Trend to move toward complex models optimised on GPUs

Optimisation tips

Factors that can be varied to improve optimisation include:

- Initialisation
  - Starting close to a good solution often helps
  - Backup solution: Neutral initialisation close to the a-priori distribution
  - Try several different random initialisations
- How long the optimisation runs
  - Generalisation first improves but then decreases as overfitting begins
    - Use early stopping
  - For DNNs on complex, computationally-demanding problems, the real limiter is often how long we are willing to wait
    - Example: OpenAI Five learning to play DotA 2
- Optimiser and hyperparameters
  - Methods with momentum often do better than just gradient ascent
  - Analytic optimisation (if possible) is often faster than gradients

Classifier problems

Problem categories:

1. Inappropriate features
2. Inappropriate model
3. Inappropriate optimisation
4. Inappropriate data

Scarce data

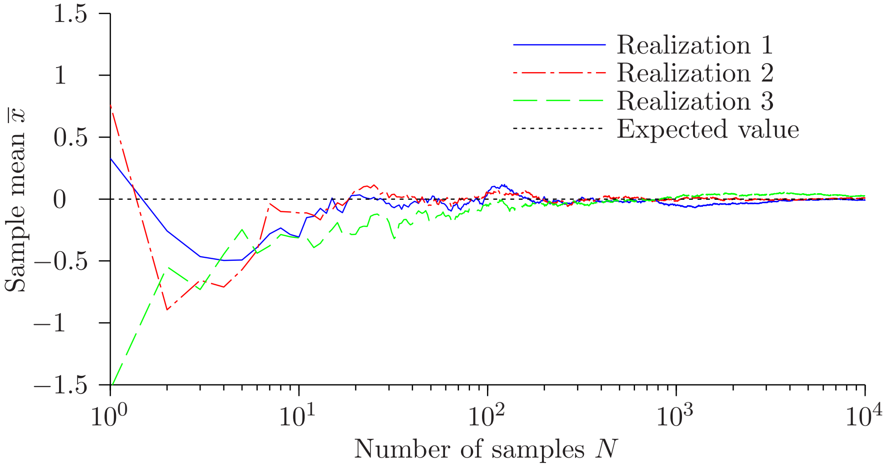
Many things can go wrong when the training data is gathered

Too little data leads to poor classifiers. Solutions include:

- Obtain more data
    - If more real data is not an option, it might be possible to "fake it"
    - *Augment* data by manually adding variation that should not affect the decision
        - Mirroring, exposure, scaling, rotation, noise, reverberation, ...
- Use a simpler model with fewer parameters
- Post-process models to anticipate possibilities not seen in the data
    - Example: Smoothing (pseudocounts) make zero-probabilities nonzero
        - For unseen words or word combinations in text classification tasks
        - Important for discrete feature distributions in the course project

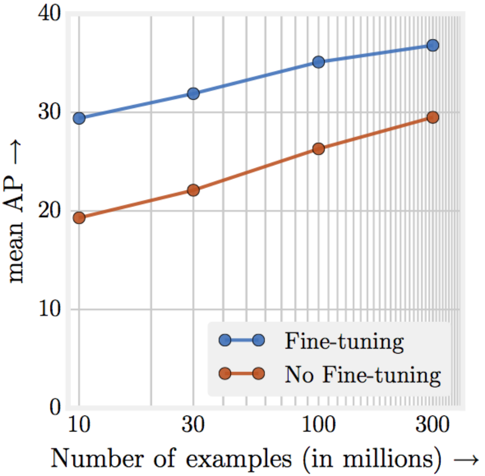
Law of large numbers

Estimates improve with more data. For complex problems, more data will also reveal more things reality might do.



Unreasonable effectiveness of data

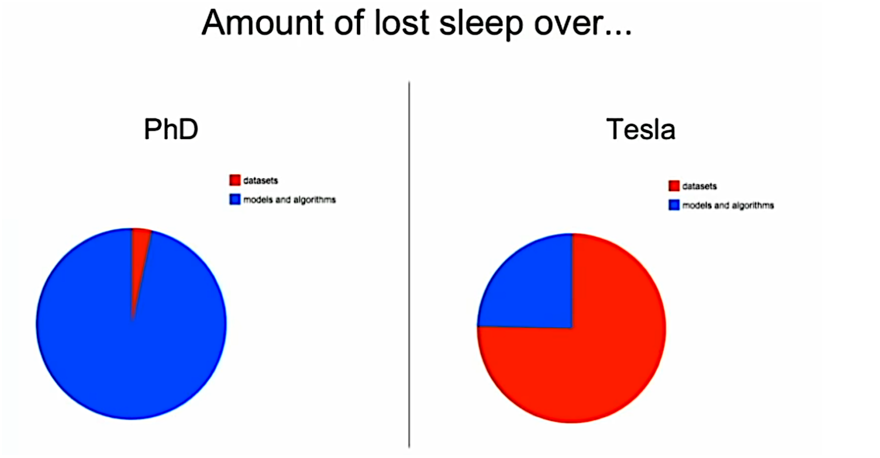
With enough computation, greater amounts of appropriate data usually beats improving models and algorithms, at least with deep learning



(Image by Sun et al., 2017)

Unreasonable effectiveness of data

With enough computation, greater amounts of appropriate data usually beats improving models and algorithms, at least with deep learning



(Image by A. Karpathy, 2018)

Systematic bias

If data is scarce or absent only for certain conditions, it is said that the data is *biased*

- Example: Self-driving car trained on video data from real driver did not learn to recover from errors
    - This particular scenario (the model iteratively "predicting itself" into a previously unseen state) is called *exposure bias*

It is often difficult or impossible to cover all possibilities equally well in practice

- Example: Stock-market crashes and other extreme events

Overcoming bias

Solutions to combat bias include:

- Gathering more data from under-represented conditions
- Simulating under-represented conditions using augmentation
  - Be careful, since this is likely to change and possibly distort the overall class probabilities in the training data
- Simply weighting the datapoints
  - If we have an idea of the nature of the bias (e.g., class imbalance)

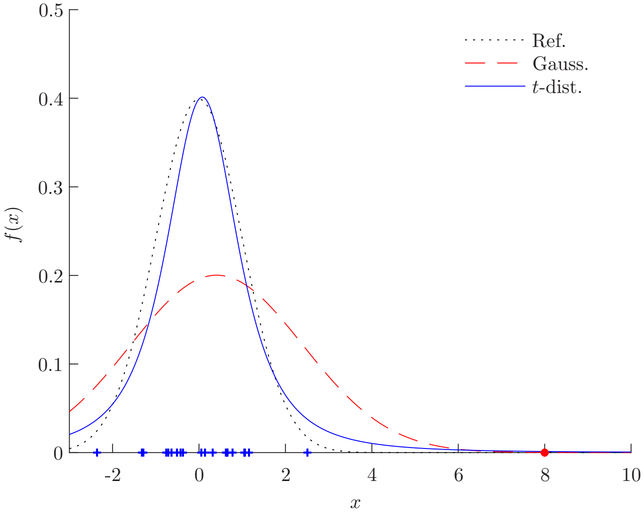
Outliers

Data or features may be noisy, or even completely wrong (*label noise*)

- Abundant data often has low quality
    - Example: Found data from the Internet
    - If we know what's bad or unreliable, we can reduce its weight
- There may be *outliers*, points that behave very different from all others
    - Outliers can be errors or weird but genuine behaviour
    - These can be highly influential on the trained models
- Use (statistically) *robust* approaches
    - Methods designed to cope with adverse conditions
    - Can provide protection against disturbances or outliers
    - Trade-off: Better when data behaves badly, suboptimal when data is well-behaved

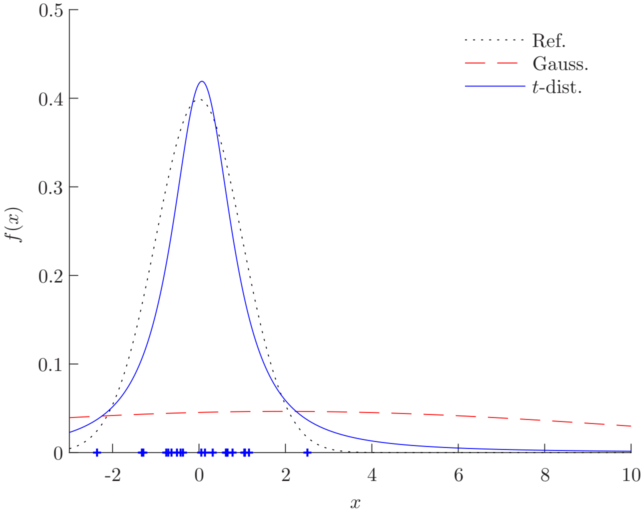
Robust model example

Data from a standard normal distribution, with one outlier.
Gaussian vs. Student's t-distribution with outlier at $x = 8$:



Robust model example

Data from a standard normal distribution, with one outlier.
Gaussian vs. Student's t-distribution with outlier at $x = 40$:



Classifier problems

Problem categories:

1. Inappropriate features
2. Inappropriate model
3. Inappropriate optimisation
4. Inappropriate data

By changing one aspect at a time, it is usually easier to identify where the performance limitation sits

Combining models

Finally, one can combine several models to achieve better results than any individual model (*ensemble methods*)

- This is a powerful approach in practice
  - Used to win prediction contests like the Netflix Prize and on Kaggle
- *Boosting*
  - Theoretically appealing
  - Many bad classifiers become one strong
- *Bayesian learning*
  - Discussed in the last chapters of the course book
  - The fully Bayesian approach is an average over different models
- *Dropout* in neural networks
- Diverse ensembles often perform best
  - Pattern recognition and deep learning might be stronger together!

Summary

How to solve a practical classification problem:

1. Gather data and information
2. Create a feature extractor
3. Propose a parametric model
4. Estimate model parameters from data
5. Choose a decision rule
6. Evaluate performance
7. Repeat from 1 or 2 while improving approach
8. Use solution (if ethical and sufficiently good)

The end

The end

Good luck with the rest of the course!