Notes on

Machine Learning Lecture

Taught by
Prof. Dr. Joni Dambre
Prof. Dr. Tom Dhaene

Created by
M.Sc. Student
Karina Chiñas Fuentes

Universiteit Gent

August 18, 2023

Weekly Content

DISCLAIMER

These are my notes and associated further reading that regards my understanding of Machine Learning based on the Universiteit Gent 2022 winter lecture. Non-cited statements come from the lecture. I also make notes to myself. If the reader finds something useful, I encourage them to investigate further. Non-cited charts were created by me, and their nonprofit-related usage is allowed with a citation.

The following text is from me to me; to prepare myself for the examination and enhance my knowledge of related topics. The empty spaces are for me to write once I print these notes.

Before you write:

1. What are the keywords?
2. What are the assumptions of the model or algorithm?
3. What are the main take away?
4. Flow Charts?
5. Advantages and Disadvantages
6. Are they linear or non-linear transformations / models

Recall that the Recap slides from Prof. Dr. Joni Dambre also helps.

1 First Steps in Supervised Learning

1.1 What is Machine Learning?

Machine Learning is the field that generates algorithms that learn from data so they can predict or make decisions based on their learning. One aims to **build a model for data by optimizing the performance criterion using training data**.

To understand machine learning algorithms, it is necessary to know about Probability Theory and Statistics. It is also desired to have efficient programming knowledge, so this is an interdisciplinary field.

Learning algorithms are implemented when no rules are available to establish the relationship between the input and the output. It is also possible that the algorithm is required to adapt to different environments.

1.2 Types of Machine Learning

There are three main different types of machine learning; these are depicted in Figure 1.

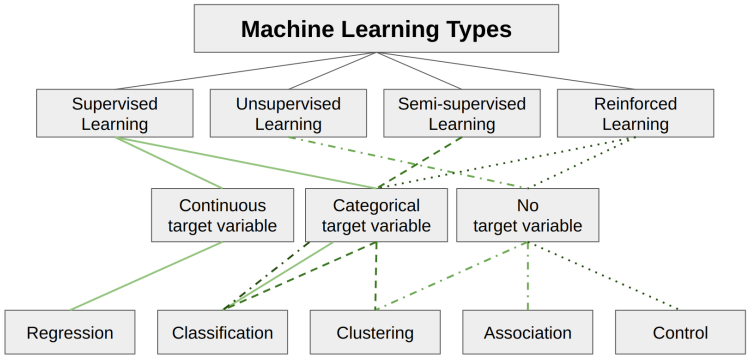


Figure 1: Different types of machine learning. Image inspired from [1].

- **Supervised**: here, the expert gives the algorithm the output data (as labels for classification) so that the input-output relationship can be modeled. The type of tasks covered here are:
  - Regression: when the output is numerical.
  - Classification: when the output is categorical.
- **Unsupervised**: the aim is to learn without an output; this approach allows us to model the structure of the data or the underlying profile of the data; they can become *discriminative* or *generative*[1]; for instance, K-nearest neightbours, random forests, and neural networks are discriminative models. Generative models are often used as pre-processing in deep learning [2].
  - Clustering: hierarchic and nonhierarchic are their sub-branches; more might be investigated. Clustering can be both discriminative and generative.
  - Association / Transformation of Features: the aim is to uncover latent variables or compress information. This can either improve or worsen the interpretability of the model, e.g. Principal Component Analysis (PCA); see Figure 2.

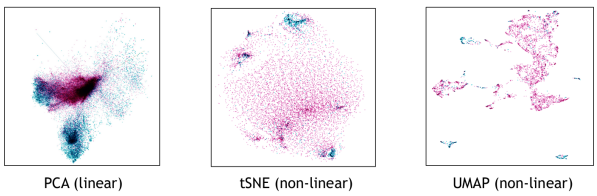


Figure 2: Different types of feature transformation, showing PCA (left), t-Distributed Stochastic Neighbor Embedding (center), and Uniform Manifold Approximation and Projection (right). Image from lecture 1.

- **Semi-supervised Learning**: as the name suggests, this is a mixture of the previous two and is typically implemented when the cost of labeling is high [1].
- **Reinforced Learning**: this method uses gathered information from the interaction with the environment to maximize reward and reduce risks [1]; this can be implemented for real-time decisions.

[1] The algorithm focuses on devising the decision boundary (without underlying assumptions on data) or on modeling the joint distribution of inputs and outputs, respectively.

1.3 Supervised Learning

The notation used in the lecture is the following[2]:

- Ground truth: $\mathbf{y} = f(\mathbf{x}) + \varepsilon$
- Training data: $\mathbf{X}, \mathbf{Y} : (\mathbf{x_1}, y_1), \dots (\mathbf{x_N}, y_N)$
- Hypothesis: $f(\mathbf{x}) \sim g(\mathbf{x}, \mathbf{X}, \boldsymbol{\theta}) + \varepsilon$; best model that can be learned from a certain model family.
- Loss function: $\mathcal{L}(\boldsymbol{\theta} \mid \mathbf{X})$
- Learning: $\hat{\boldsymbol{\theta}} = argmin_\theta \left(\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta} \mid \mathbf{X})\right]\right)$; minimize expected loss on new data.

Notice that $f(\mathbf{x})$ is unknown. In general, finding the optimal parameters that will minimize the expected loss function on new data is necessary. Optimal parameters depend on the assumption made on data. Finding non-parametric algorithms in supervised learning for regression and classification is also possible, e.g., K-nearest neighbors (KNN). See Figure 3 for a better comparison.

! Algs: KNN (regression and classification), decision trees (regression and classification), linear discriminate (regression and classification), and non-linear feature expansion (regression).

Consider mentioning the training/inference time and memory scale w.r.t. dataset size when describing an algorithm. These issues become relevant once the quality of the data has been addressed.

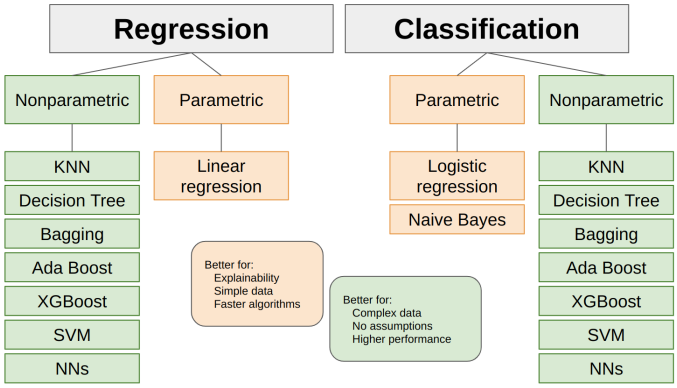


Figure 3: Types of algorithms in supervised learning and some examples. Image inspired from [4].

1.4 Evaluating Models and Conditions for Generalization

Machine learning aims to achieve **generalization**; the algorithm should perform well when new data is introduced. To understand how this is feasible, important concepts will be introduced.

- **Loss function**: this is among the most important metric in machine learning. This function allows the user to tell how good a model's prediction is, based on the *the loss*. This transforms the learning into an optimization problem by defining a loss function and optimizing it to its minimum. More on this shall be explored [5]. ! Different loss functions, different predictions.
- **Empirical Risk Minimization** (ERM): the understanding of ERM permits us to understand the limits of an algorithm, which also allows the development of practical skills in machine learning [6]. The concept of empirical risk arises from the fact that the user cannot access the *true error* since the algorithm only receives a sample of an unknown distribution from the data. However, estimating the *training error* – the error the algorithm incurs over the training sample [7] is possible. In short, ERM is the search for a predictor (or model) that minimizes the training error; this can be depicted in the following expression

$$\hat{\boldsymbol{\theta}} = argmin_\theta \left(\frac{1}{N} \sum_{j=1}^{N} \mathcal{L}(x_j \mid \boldsymbol{\theta}, \mathbf{X}) \right). \tag{1}$$

  Remember that the model represents your **hypothesis**. ERM: the search for the best model that minimizes training error.
- **Test Error**: because the ERM does not tell the user how well the algorithm is capable to predict on new data, what is typically done is to split the available data such that, once the algorithm has been trained (found $\hat{\boldsymbol{\theta}}$), the algorithm can be assessed on the performance on the *test data*. ! WARNING: depending on how you split the data, how likely you are introducing *lekeage* in your model.
- **IID Assumption**: Identically and Independent Assumption:

[2] However, the nomenclature is not always maintained; therefore, you might find the notation following in [3]. To avoid confusion, it is explained in the text what variables mean.

- Indetically: training, validation, test, and new incoming data are drawn from the same underlying joint distribution. In other words, all datasets must have the same statistical properties: mean, standard deviation, and other characteristics [8].

  Violating this assumption creates

  * prediction bias; predictions are not characteristic of the real world due to lack of representative groups, e.g., training facial recognition on engineering students: population not very diverse, very few women, no children, no elderly people. This can also result from a lack of balanced classes.

    Feature and label bias can be considered a violation of the "identically distributed" part of the i.i.d. assumption.

- Independent: no subgroups of samples correlated in any type of data subset (training, validation, and test); in other words, any data point should not provide information on the occurrence or value of another data point [8]. Usually, The correlation is attributed to **latent varibales**, a.k.a. unknown or unobserved variables.

  Violating this assumption creates

  * bias;

Violation of any assumption leads to poorer generalization, and they can be interconnected. !!! I think what is important in this discussion is identifying the source of a violation rather than the independent consequence because the violation of each might result in a similar consequence. FOCUS ON THE SOURCE RATHER THAN THE CONSEQUENCE.

—

For instance, the characteristic clusters **in STDB5** for the spherical and non-spherical tokamaks violate the identical assumption. This might not necessarily mean that spherical tokamaks must have their own scaling law (?). In reality, there is no continuum in aspect ratio to effectively assess whether they are *identical* machines. This implies too much money to discover.

1.5 Model Selection: hyperparameter analysis

When one changes the hyperparameters of an algorithm, it can be considered a different model than before; for instance, the number of features one uses is a hyperparameter. How can one tell which model is better compared to another? It is recommended to follow the diagram shown in Figure 4, left.

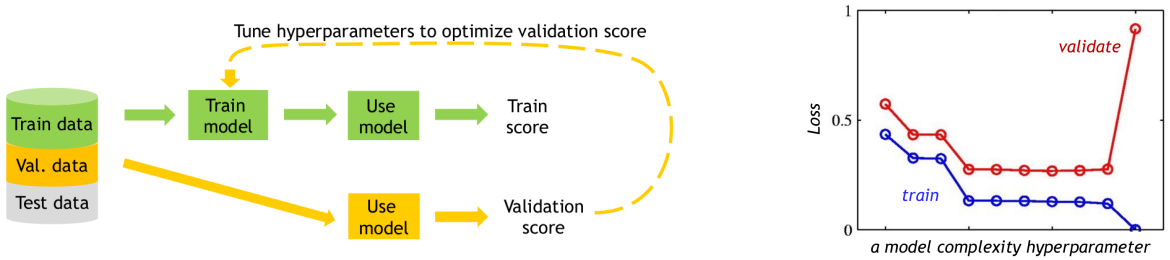


Figure 4: Left: data split for model assessment, all data subsets are assumed to follow IID assumption. Right: validation curve; if the IID assumption is not fulfilled, the curves will not look similar; this is why it is recommended to look at this plot at the beginning. Sometimes, the validation curve is plotted with accuracy or another metric, like F_1-score. Images from lecture 1.

When you tune the model's hyperparameters, you also change the model complexity; the validation curve is assessed from this analysis. The validation curve plots a model performance metric vs. the algorithm's complexity [9]. The typical behavior is that the validation error goes down as the complexity increases, but the error goes up again at some point; see Figure 4, right. **More on this will be explained when the bias-variance decomposition in a model is studied.** Once the ideal model has been found, the train and validation datasets are merged to obtain the overall train and test errors without further modification. This is depicted in Figure 5

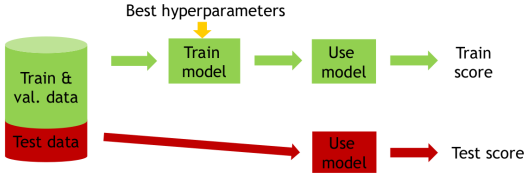


Figure 5: Data split for model assessment, final stage. Image from lecture 1.

If the validation and training data are too small, the model will likely have a poor choice of hyperparameters and might overfit. Again, beware of how you split the data, as you might introduce a source of leakage.

Another discussion brought on the analysis of model complexity is the choosing of features. In general, it is not ideal to have too many features due to what is known as *the curse of dimensionality*. The reasoning is that a higher number of features implies a more complex model, and the more complex the model, the more data it requires to perform well. For instance, consider the case of fitting data with a polynomial of degree d with F total number of features. The complexity of the said model is $\mathcal{O}(F^d)$; from this, it is possible to observe that the selection of features becomes a serious matter for optimal performance, directly affecting its generalization capability.

Here are some forms of **regularization** for optimal model selection:

- Spend time searching for the optimal subset of features to avoid the "curse of dimensionality".
- If possible, reduce the number of hyperparameters (this is a matter in neural networks).
- Constrain parameter space: the idea is that you assess for a region in parameter space where you know solutions will lead to an optimal generalization, e.g. L_1 and L_2 regularization, more on this in the incoming chapter.
- When working with optimization algorithms, such as gradient descent, avoid over-tuning in training data (e.g. early stopping).

1.6 Summary

Key words: empirical risk minimization (ERM), validation curves, curse of dimensionality.

2 Supervised Parametric Linear Models and Introduction to SVM

Three central concepts must be understood when discussing parametric models: loss function, regularization, and optimization [3]. To understand these concepts, let us have a look at the equation that *models* the true input-output relationship

$$y = f(\mathbf{x}) + \varepsilon. \tag{2}$$

Depending on the assumptions, is the algorithm; for instance, if $f(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x}$ is linear, and the noise is Gaussian $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, you retrieve ordinary least squares [3]. Derive the previous statement:

Of course, $f(\mathbf{x})$ can be non-linear, and other assumptions can be imposed over the irreducible noise, depending on the problem; for instance, the assumption of $\varepsilon \sim$ Laplace distribution, one obtains the *absolute error loss* [3]. However, if one keeps the assumption of Gaussian noise, one will obtain a Gaussian joint likelihood [3], which eases the interpretation and working of the model's predictions.

The main objective of the learning algorithm is to learn enough from the training data to properly predict or make decisions on unseen data; this is known as *generalization*. A model may also learn the noise from the training data rather than the underlying properties of it; this is known as *overfitting*. The loss function is implemented to prevent overfitting so that the model does not mimic the training data.

2.1 Loss Functions

Recall that, the learning of an algorithm means that one is dealing with an optimization algorithm [3] of the parameters $\boldsymbol{\theta}$. As previously mentioned, the optimization problem is formulated as follows

$$\hat{\boldsymbol{\theta}} = argmin_\theta \left(\frac{1}{N} \sum_{t=1}^{N} \mathcal{L}\left(y_i, f_\theta(\mathbf{x_i})\right) \right), \tag{3}$$

here, one observes

- the **loss function** $\mathcal{L}(y_i, f_\theta(\mathbf{x_i}))$, and
- the **cost function** $J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{t=1}^{N}\mathcal{L}\left(y_i, f_\theta(\mathbf{x_i})\right)$.

It is possible that a solution to the optimization problem is not exactly and is not directly computable; this is particularly common in non-linear functions, e.g. no closed-form solution exists for logistic regression. Therefore, a solution might be found through numerical optimization [3]. However, the ultimate goal is not solving Eq. (3), but rather

$$\hat{\boldsymbol{\theta}} = argmin_\theta\left(E_{new}(\boldsymbol{\theta})\right), \tag{4}$$

with $E_{new}(\boldsymbol{\theta}) = \mathbb{E}_*\left[E(\hat{y}, y_*)\right]$, being the expected error on predictions w.r.t. unseen data. Nevertheless, one does not have access to this information; for this reason that **the loss minimization is the representative for generalization** [3]. Therefore, it is of crucial importance that the machine learning practitioner understands that the training objective, Eq. (3), is just a proxy for the actual objective, Eq. (4). The authors in [3] stress on the following observations

- optimization accuracy $\neq$ statistical accuracy,
- loss function $\neq$ error function, and
- explicit vs implicit regularization (explain *asymptotic minimizer* and *strictly proper*[3] functions).

It is important to understand that selecting the loss and error functions comes as part of the model's design. There is no right nor wrong selection, but rather, what represents best your data. For instance, one has

- linear regression: linear in the parameter model and $\mathcal{L}$ being the squared error loss; and,
- support vector classification: linear in the parameter model and $\mathcal{L}$ being the Hinge loss [3].

IMPORTANT: a loss function is said to be robust if the training data containing a considerable amount of outliers only has a minor impact on the learned model [3].

2.1.1 Regression

As already discussed, some loss functions come naturally from the assumptions made on the data, such as the *squared error loss*

$$\mathcal{L}(y, \hat{y}) = (\hat{y} - y)^2, \tag{5}$$

coming from a Gaussian noise in Eq. (2); and, the *absolute error loss*

$$\mathcal{L}(y, \hat{y}) = |\hat{y} - y|, \tag{6}$$

if the noise in Eq. (2) is $\varepsilon \sim L(0, b_\varepsilon)$ Laplacian [3]. However, not all loss functions come from an instinctive derivation from data assumption; some are designed to achieve a superior generalization. For instance, the *Huber loss*

$$\mathcal{L}(y, \hat{y}) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2 & \text{if } |\hat{y} - y| < 1 \\ |\hat{y} - y| - \frac{1}{2} & \text{else.} \end{cases} \tag{7}$$

This is a combination of the previous two; this is because the absolute error is moss robust to outliers than the squared error [3]. Another variation to this idea is depicted with the *ϵ-insensitive loss*

$$\mathcal{L}(y, \hat{y}) = \begin{cases} 0 & \text{if } |\hat{y} - y| < \epsilon \\ |\hat{y} - y| - \epsilon & \text{else,} \end{cases} \tag{8}$$

[3]Hint: the downside of Hinge loss; remedy: squared Hinge loss. Do not forget to see p.105 in [3].

where ϵ is left to be chosen for the design, notice that the *ϵ-insensitive loss* leaves a tolerance width 2ϵ around the observed y and, outside the region, behaves like the absolute error loss [3]. This loss function proves particularly useful for support vector regression [3]. See Figure 6 to compare the robustness of the functions mentioned in this section.

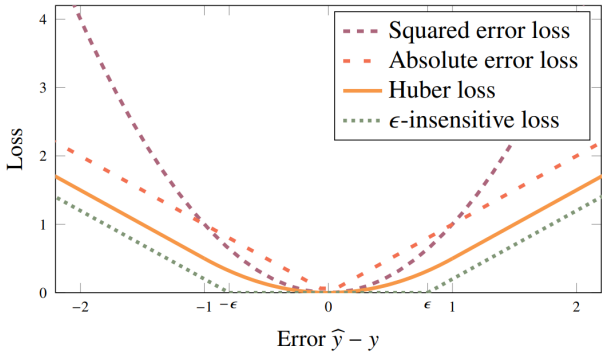


Figure 6: Different loss functions for regression given a specific error function. Image from [3].

2.1.2 Classification

An instinctive loss function for binary classification could have similar behaviour to the Heaviside function; there is one known as the *misclassification loss* written as

$$\mathcal{L}(y, \hat{y}) = \begin{cases} 0 & \text{if } \hat{y} = y \\ 1 & \text{else.} \end{cases} \tag{9}$$

Despite that one might have the goal of suppressing the misclassification rate the most, this is not the commonly chosen loss function for three main reasons (explain). The usage of the *cross-entropy loss*

$$\mathcal{L}(y, g(\mathbf{x})) = \begin{cases} \ln(g(\mathbf{x})) & \text{if } y = 1 \\ \ln(1 - g(\mathbf{x})) & \text{if } y = -1 \end{cases} \tag{10}$$

is more used. Here, $g(\mathbf{x})$ is the probability that the observation $\mathbf{x}$ belongs to a class; this allows a complete statistical description of the output-input conditional distribution [3]. To understand other families of loss functions, it is essential to comprehend the concept of **margins**. In general, it is defined that **the margin of a classifier for a data point** $(\mathbf{x}, y)$ **is** $y \cdot f(\mathbf{x})$; so that if the margin has the same sign, the classification is considered correct, otherwise incorrect [3]. This construction helps when the classifier does not have a probabilistic interpretation. Still, rather it is just being represented with an underlying function $f(\mathbf{x})$; it can be seen as a measure of certainty in a prediction [3] (!!! think: similarity between error in regression loss functions and margin in classification loss functions). From the construction of the margin, it is possible to assign a small loss to a positive (correct classification) margin and a considerable loss to a negative margin. Figure 7 shows diverse loss functions for classification with different robustness to outliers.

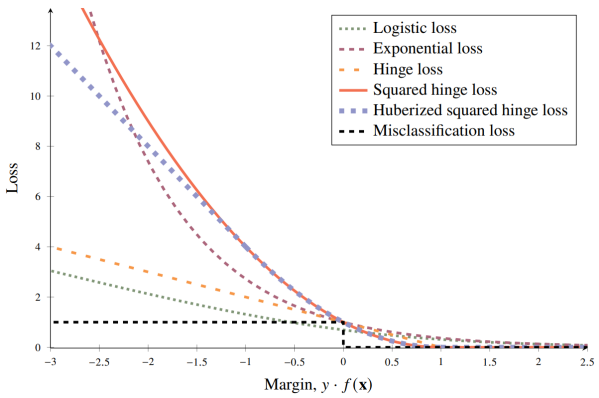


Figure 7: Different loss functions for classification given the margin. Image from [3].

Write the expressions for the different loss functions shown in the image with their respective properties and algorithms usage[4].

[4] Interesting: loss for multiclass classification?

2.2 Learning parameters for Specific Cases

Explain the solution and main characteristics when minimizing the loss functions for regression and classification using MSE (or SSE) and log-loss, respectively. Explain gradient descent and compare it with coordinate descent.

2.3 Overfitting and Explicit Regularization

As mentioned, generalization is when an algorithm performs well on unseen data. Overfitting happens when proper generalization is not achieved, and the algorithm only performs well on the training dataset; there are many causes of overfitting, one being a model being too complex (e.g. number of features being used).

The goal of regularization in parametric models is that: "if a model with small values of the parameter $\hat{\boldsymbol{\theta}}$ fits the data almost as well as a model with larger parameter values, the one with small parameter values should be preferred." [3]

Regularization is the act of preventing overfitting. For implicit regularization, one can modify the expression of the loss function; for instance, **Ridge regression** or $\mathbf{L}^2$ **regularization** is the addition of a function that penalizes high weights in polynomial regression (because high values of weights mean high complexity algorithm). The resultant expression is the following

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2 + \lambda \cdot ||\mathbf{w}||_{(11)}^2$$

With $\lambda \geq 0$, one always has a unique solution to regression – the best value is obtained through cross-validation. $\lambda = 0$ gives OLS solution. Notice that the modified equation presents a trade-off between having the perfect fit and enforcing the regressor parameters to being close to zero; the greater λ, the closer to zero the $\hat{\boldsymbol{\theta}}$ values. Given the loss function with Ridge regression, one obtains [3]

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^T\mathbf{X} + n\lambda\mathbf{I}_{p+1}\right)^{-1}\mathbf{X}^T\mathbf{y}, \tag{12}$$

with n denoting the number of observations in the dataset. Another type of regularization is known as **LASSO** or $\mathbf{L}^1$ **regularization**, and it is of the following form

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2 + \lambda \cdot ||\mathbf{w}||_1, \tag{13}$$

where $||\mathbf{w}||_1$ is the 1-norm $||\mathbf{w}||_1 = |w_1| + |w_2| \ldots + |w_p|$ [3]. It is worth noting that there is no closed-form solution to this loss function; numerical optimization algorithms must be used to solve the LASSO regression. The effect of this type of regularization is that it can *switch-off* some coefficients (by setting specific weights to zero) and provide sparse solutions; it is for this reason that, sometimes, this is a method of feature selection [3].

Following images from the book [3].

General Explicit Regularisation

L^1 and L^2 regularisation are two common examples of what we refer to as explicit regularisation since they are both formulated as modifications of the cost function. They suggest a general pattern on which explicit regularisation can be formulated:

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \underbrace{J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y})}_{\text{(i)}} + \underbrace{\lambda}_{\text{(iii)}} \underbrace{R(\boldsymbol{\theta})}_{\text{(ii)}}. \tag{5.26}$$

This expression contains three important elements:

(i) the cost function, which encourages a good fit to the training data;

(ii) the regularisation term, which encourages small parameter values; and

(iii) the regularisation parameter λ, which determines the trade-off between (i) and (ii).

Method	***What is optimisation used for?***			***What type of optimisation?***			
	Training	*Hyper-parameters*	*Nothing*	*Closed-form**	*Grid search*	*Gradient-based*	*Stochastic gradient descent*
k-NN							
Trees							
Linear regression							
Linear regression with L^2-regularisation							
Linear regression with L^1-regularisation							
Logistic regression							
Deep learning							
Random forests							
AdaBoost							
Gradient boosting							
Gaussian processes							

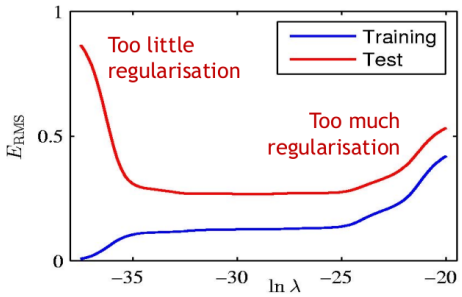
**including coordinate descent*

Figure 8: Images from [3]

Generalisation: perform well on unseen data!

Train model on training data

More complex model: *training error* decreases



Evaluate model on separate test data: at some point test error goes up again.

⇒ *Overfitting to the training data!*

⇒ *Select best degree based on unseen data!*

Figure 9: Image from lecture 2

2.4 Classification: Linear Support Vector Machine

2.5 Classification: Soft Margin Support Vector Machine

2.6 Summary

Keywords: asymptotic minimizer, strictly proper, margins, optimization, open- and closed-form solutions.

3 Reasoning About Models and Data

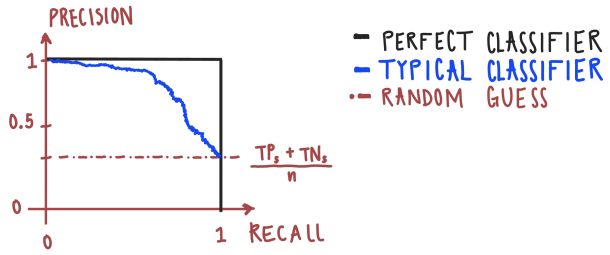
3.1 Error Functions for Model Selection

Describe the confusion matrix and why it is needed.

For the following, add comments based on slides.

Table 1: Some of the obtainable metrics from a confusion matrix [3], [10]. Remember that n is a dataset's total number of observations.

Metric	Formula	Description
Misclassification Rate	$\frac{\text{FNs} + \text{FPs}}{n}$	Fraction of predictions being incorrect
Accuracy	$\frac{\text{TNs} + \text{TPs}}{n}$	Complement of misclassification rate
Precision	$\frac{\text{TPs}}{\text{TPs}+\text{FPs}}$	Fraction of predicted positives actually being positive
Recall	$\frac{\text{TPs}}{\text{TPs}+\text{FNs}}$	Fraction of actual positives correctly predicted
Fall-out	$\frac{\text{FPs}}{\text{FPs}+\text{TNs}}$	Probability of false alarm
Specificity	$\frac{\text{TPs}}{\text{FPs}+\text{TNs}}$	Compliment of fall-out
False discovery rate	$\frac{\text{FPs}}{\text{FPs}+\text{TPs}}$	Fraction of incorrect positive predictions
False negative rate	$\frac{\text{FNs}}{\text{FNs}+\text{TPs}}$	Fraction of actual positive incorrectly classified
False omission rate	$\frac{\text{FNs}}{\text{FNs}+\text{FPs}}$	Fraction of incorrect negative relative to tall incorrect classifications
Prevalence	$\frac{\text{FNs} + \text{TPs}}{n}$	Proportion of actual positive instances in the dataset
F_1-score	$\frac{2 \cdot precision \cdot recall}{precision + recall}$	Harmonic mean of precision and recall
F_β-score	$\frac{(1+\beta^2) \cdot precision \cdot recall}{\beta^2 (precision + recall)}$	Used to account that recall is β-times as important as precision



Explain RoC curve beside the PR curve.

Explain hypothesis testing and its importance.

3.2 Bias and Variance

For this discussion, the **IID assumption is conserved**. This discussion centers on *What happens if you train a model multiple times on different IID data sets of the same size?* The answer is that one gets **different models**, with the difference being more notorious if the complexity of the model is high; however, the difference decreases if the amount of data increases in each case. This is depicted in Figure 10

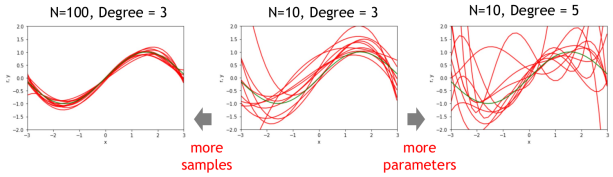


Figure 10: Image from lecture 3.

Explain what bias and variance are **in statistics**.

An **estimator** or a model $g()$ is an instance of a model family being fed with a fixed number of observations in the dataset.

Derive an estimator's bias and variance decomposition (see the summary slide of bias and variance).

Explain the theoretical plot of generalization error [3]:

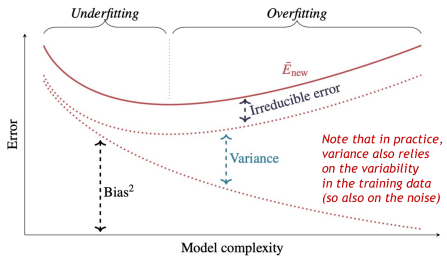


Figure 11: Image from [3].

The trade-off between model complexity and dataset size opens the discussion on

- **MODEL BIAS**: this theoretical concept cannot be measured because the user cannot access the underlying data distribution. This characterizes being
  - **the expectation** across various models trained with different data subsets of the expected model error on unseen data, compared to the ground truth.
  - **the capability** of the chosen estimator to approximate the ground truth.

  High bias implies that the model is too simple to extract suitable information from the features; this is also referred to as **underfitting**. The result of high bias can be a high observation of error; however, this could also be from a large amount of irreducible error (see plot above), meaning that features do not carry the necessary information for estimation.
- **MODEL VARIANCE**: this theoretical concept cannot be measured because the user cannot access the underlying data distribution. This characterizes being
  - **the variance** across various models trained with different data subsets of the expected model error on unseen data, compared to the ground truth.
  - **the sensitivity** of the chosen estimator to variability in training data.

  When high variance is observed, this is because the model is **overfitting** the training data, so it is suggested to either obtain more data or make the model less complex.

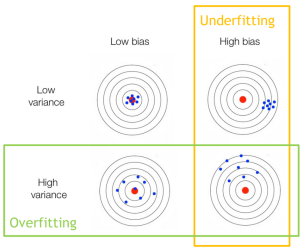


Figure 12: In the bull's eye plots, each blue dot represents the average error on unseen data of a model trained on a different dataset; the red mark represents the perfect model. Without better data, model generalization error trades-off between high variance and low bias and vice versa. Image from lecture 3.

Bias and variance trade-off if you keep the amount of data and the model type fixed but tune the model complexity.

A model can be complex enough to overfit the training data but still be unable to capture the ground truth due to a large irreducible error; hence, **it is possible to have both in a model**.

Figure 13 considers how the generalization error and variance change as the dataset size increases for a theoretical example with known ground truth and noise-free error.

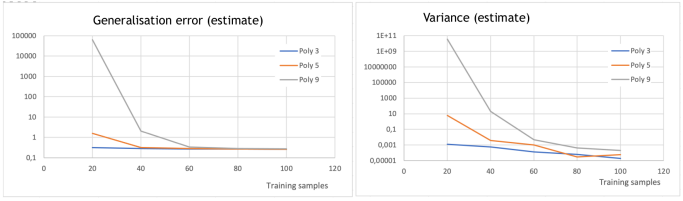


Figure 13: Variation on training dataset size and generalization error with variance with known ground truth and controlled noise; thus, this is an ideal case. Image from lecture 3.

From this, it is possible to observe that, for greater datasets, the generalization error decreases and converges to the irreducible error while the variance converges to zero. It is important to understand that **convergence is slower for more complex models or complex noise**.

LEARNING CURVES: when ground-truth is not available – sampling impact from training dataset
In practice, bias and variance cannot be estimated since one cannot access the underlying data distribution. For this, it is possible to obtain the learning curves that yield information on whether the model has high variance and/or bias by sampling the impact of training dataset size from the total training set. Figure 14 shows the learning curves for two situations.

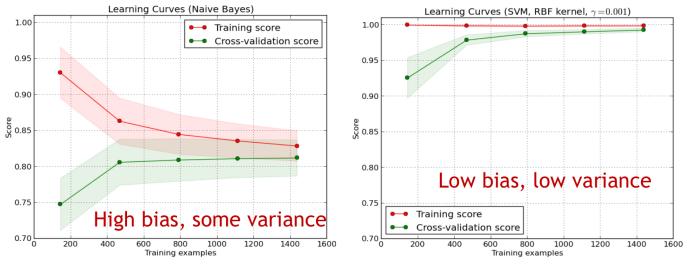


Figure 14: Learning curves with accuracy score; finding learning curves with the loss instead of the score is also possible. On the left-hand side (low N, both plots): insufficient data, severe overfitting, training performance close to perfect, validation performance very bad. On the right-hand side (high N, both plots): hopefully enough data and less overfitting. Limit to infinity: training and validation curves converge to the same value. Image from lecture 3.

Here are a couple of observations:

- If training and validation sets are identically distributed, **and** validation set is large enough: scores should converge to the same value.
- Extrapolation towards *converged value* estimates the best possible generalization error, i.e. model bias + irreducible error.
- More complex models should give lower errors: lower model bias, same irreducible; if not: irreducible error dominates (or other data problem).
- Variability of scores does NOT reflect variance.

Learning vs (Cross)-Validation Curves

- Learning:
  - Plot shows the average training loss/score in the y-axis.
  - Number of samples per training in the x-axis.
- Cross-Validation:
  - Plot shows the average validation loss/score in the y-axis.
  - Serves as an estimate for loss on unseen data.
  - Hyperparameter variation in the x-axis (e.g., regularisation parameter λ, number of features, polynomial degree, training time (#batches) in gradient descent, number of neurons in a neural network, etc.)

3.3 Risks of Data Splitting and IID Assumption Fulfillment

There are some issues when one performs data splitting for model development; for instance, if the training set is too small, there is a risk of overfitting. If the validation set is too small, poor hyperparameter choice is possible. If the test set is too small, the test score might not represent its generalization capacity well. Nevertheless, these are not the only risks of data splitting; one must ensure that the IID assumption is fulfilled as best as possible.

As mentioned, **non-independent data can lead to leakage** of information from validation or test sets into training, resulting in optimistic scores but disappointing performance in the real world. Here are some of the causes:

- Validation and/or test data not sampled independently from train data (the split was not independent). Think of an example!
- Some model or preprocessing parameters are tuned on validation and/or test data. Example: normalizing data based on all data's average and standard deviation (instead of just training data).
- Not enough data to average on latent variables and give identically distributed sets to the three splits.

Example: suppose you have 440 data points from 8 different hospitals; for splitting this dataset, in practice, one has two options

- one randomly shuffles all data, then splits into 3 sets.
  - Positive aspect: three datasets come from the sale distribution; hence, one IID was fulfilled w.r.t. local dataset.
  - Negative aspects: they are not identically w.r.t. the underlying distribution or ground truth; hence, there is a risk of overfitting on these eight hospitals. Furthermore, there is a risk of leakage: records from a single patient could end in training and test sets.
- one randomly selects hospitals to create the three datasets (e.g. 3 for the train, 2 for the validation, and 3 for the test) – this is a **grouped split**.
  - Positive aspect: there is no risk of leakage as **they are independent datasets** (one IID fulfilled w.r.t. underlying distribution).
  - Negative aspect: sets not identically distributed w.r.t. local dataset and underlying distribution.

Mention the other splitting forms and when they are used for specific cases.

3.4 Robust Validation Strategies

When dealing with too little data, it is possible to make use of cross-validation, which consists in averaging across results for multiple train-validate splits given a single dataset. The idea is that one trains the model multiple times (with different train/validation split each time) to then average the validation score across the training runs for model selection. Here are some of the benefits:

- more robust estimate of the error on unseen data,
- allows reducing the size of the validation set (more data for training),
- allows a more in-depth analysis of distribution issues, and
- shows variability across validation errors that combine different training sets (variance) and different validation sets.

Here are some of the types of cross-validation:

- **k-fold cross-validation**: split training data into k folds (typically 5 to 10) considering the IID assumption.

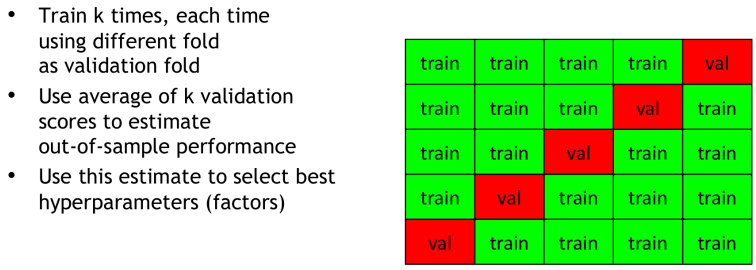


Figure 15: Image from lecture 3.

After final parameter selection: use all train & validation data to train final model – final evaluation on the test set (stage 2 – refit option = True)!

- **Grouped kCV**: assures independence (one of IID) by splitting per group. The number of folds is upper-bounded by the number of groups in the dataset.
- **Stratified kCV**: assures identically distributed (one of IID) by splitting with the assurance of the same amounts of classes in all splits. The number of folds upper-bounded by the number of samples in the smallest class.
- **Stratified-Grouped kCV**: assures both identically distributed and independence (two of IID) by assuring the previous two characteristics.
- Explain multiple random splits and bootstrapping.

The same loss function (for training) and error function (for hyperparameter turning) should be used in all cases. Here are some characteristics according to the number of folds:

- **Fewer folds**: less averaging (more sensitive to the variability of the training set), BUT each fold is more representative because there is more data. This is okay if the given dataset is large. Also, this implies less training time.
- **More folds**: more robust for small datasets and optimally use data for training; however, it takes more time for training.
  - EXTREME CASE: leave-one-out: only one data point is left for validation, and everything else is used for training. **Use LOOCV for small datasets or when estimated model performance is critical!**

3.5 Analysis of Model Performance

When reporting the model scores (train, validation, and test), it is natural to observe two types of gaps: train-validation and validation-test; how large these gaps are will tell the user the possible problems within the model and suggest possible actions to mitigate them. If

- **train-validation gap is too large**: this may be an overfitting issue; here, it is recommended to plot the validation curves and check whether the hyperparameter tuning was correctly done. It is also possible that there are too many features, and feature engineering might be required. Finally, one can also try a simpler model. **If overfitting is not the issue**, it is possible that the corresponding datasets are not identically distributed; for this, plotting learning curves and fold analysis is recommended. If possible, try to increment the size of the validation dataset. The following image shows how to determine whether this is an overfitting or IID issue.

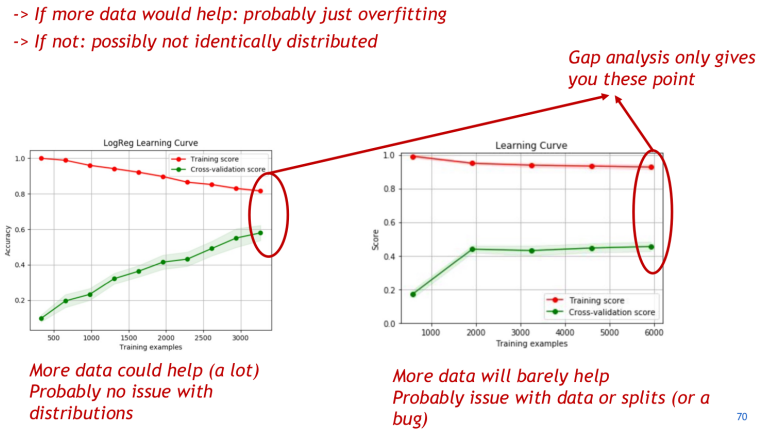


Figure 16: Image from lecture 3.

Explain what exactly fold analysis means.

- **validation-test gap is too large**: leakage may be the main cause of this gap; for this, it is recommended to enhance the splitting of the datasets to ensure independence. If splitting is not the source of leakage, then it is possible that it comes from incorrect preprocessing: everything computed from data MUST be refitted for each fold. It is possible that overfitting in the validation set is the cause due to extreme hyperparameter tunning; for this, the tunning of fewer hyperparameters could help. If possible, adding more data to the validation set is recommended. A violation of the identically distributed can also be the cause; maybe test data was collected separately.
- In both scenarios, falling back on domain knowledge and thinking is recommended.

Gaps indicate unreliable estimation of the algorithm's performance on unseen data; if they remain, it is recommended to attempt a simpler model: higher bias and lower variance can avoid 'unpleasant surprises' in the real world. It is also possible to implement **error analysis**, which requires the implementation of diverse confusion matrices to gain a complete overview of what could be wrong. Mention and explain the three different confusion matrices.

Explain which other types of examinations could be implemented on the given dataset.

3.6 Augmentation

Explain augmentation: why is it needed? How does it differentiate from data cleaning? advantages and disadvantages. How does/can each affect your model? What happens if you do too much or too little of either?

3.7 Summary

Chapter 11: gaps, learning curves, error analysis, model debugging, and augmentation.

Keywords: estimator, model bias and model variance, learning curves, leakage, grouped split, validation strategy,

4 Clustering – K-Means and Clustering Mixture Models

There are instances where data presents specific clusters with some features. It is possible to characterize these clusters to obtain their intra- and inter-cluster distances. With this approach, one can obtain the correlation of random variables, compress the data, or detect anomalies.

4.1 Latent Variables

Latent variables are those that cannot be observed but are relevant to the problem; thus, they are also referred to as hidden variables. These variables can appear naturally (for instance, because of faulty sensors) or be introduced intentionally to model complex dependencies, as shown in the following image.

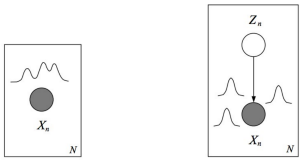


Figure 17: Image from lecture 4.

When latent variables deal with categorical data, it is a clustering problem – which is finding the missing data labels. Otherwise, it is a dimensionality reduction issue – which is finding the missing data inputs.

When there are latent variables present, the algorithm's learning because harder. Why? In a fully observed IID scenario, the probability of a model is just the product of the associated distributions (joint distribution); hence, the logarithm of the likelihood decouples the terms of the associated observed variables.

$$\begin{aligned}\ell(\theta;\mathcal{D}) &= \log p(\mathbf{x},\mathbf{z}|\theta)\\ &= \log p(\mathbf{z}|\theta_z) + \log p(\mathbf{x}|\mathbf{z},\theta_x)\end{aligned}$$

Figure 18: Image from lecture 4.

With hidden variables, the likelihood depends on a sum from the marginalized latent variables, so the log-likelihood still has all parameters coupled together

$$\begin{aligned}\ell(\theta;\mathcal{D}) &= \log \sum_{\mathbf{z}} p(\mathbf{x},\mathbf{z}|\theta)\\ &= \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z)p(\mathbf{x}|\mathbf{z},\theta_x)\end{aligned}$$

Figure 19: Image from lecture 4.

Not only is learning harder but also this is usually perceived as a black box if latent variables do not have an intuitive interpretation. Furthermore, statistically, hard to guarantee convergence to a correct model with more data. Finally, it usually is more computationally expensive due to its non-closed form for Maximum Likelihood estimates; however, the Expectation-Maximization (EM) algorithm provides a general-purpose local search algorithm for learning parameters in probabilistic models with latent variables.

Nevertheless, not everything is bad when latent variables are present in a situation, as their presence can enhance generalization with their applications; some are learning with missing data, discovering new features, and more.

Professor Q.: *Discuss latent variables in the continuous case vs discrete case, and discuss pros/cons of latent variables.*
Professor Q.: *How to deal with unobserved/latent variables?*

4.2 K-Means Clustering

Intuitively, a cluster is a group of points close together and far from others.

Professor Qs.:*Explain the K-Means clustering algorithm. Is (global) convergence guaranteed? What is the computational complexity of K-Means?*

For the following algorithm, annotate the computational complexity per step.

- **K-Means Clustering**
  - **Initialize** K centroids (cluster means), then **iterate** between two phases:
    - **step 1** (E): given centers, **assign points** to nearest "cluster"
    - **step 2** (M): given points, **update centers** to be the new cluster means

Write and explain the cost function of this problem.

Because KMeans is a (Euclidean) distance-based algorithm, it is strongly suggested to first rescale the data. Make notes to the following image indicating the steps from the algorithm.

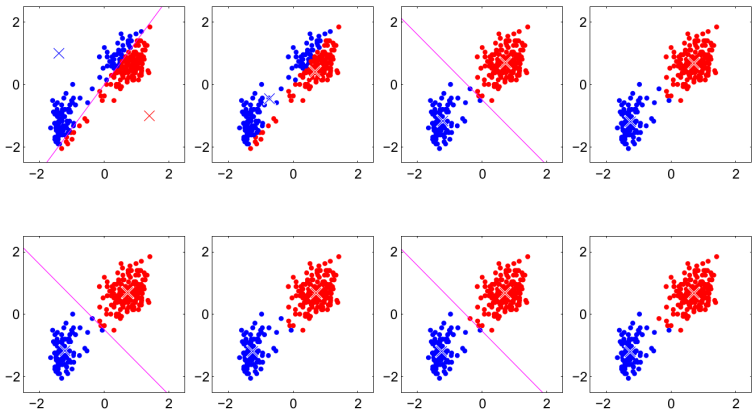


Figure 20: Image from lecture 4.

The algorithm is **guaranteed to converge** after a finite number of iterations; if it converges in a local minimum or a global minimum is strongly dependent on the initialization of the K centroids.

Professor Q.: *What is vector quantization?*

One of the common applications of KMeans is Image Compression (also referred to as vector quantization). For instance, in the image below, the results vary after compressing the image with K colors.

Each pixel (color value) is one data point in RGB space; considering this, all pixels in a cluster can be colored by the cluster's mean. This is known as hard assignments of data points to clusters.

Figure 21: Image from lecture 4.

Professor Q.: *How to select the optimal number "K"?*

The problem at hand can determine the selection of the best K. For instance, if there is a meaning to K, it is possible to manually set this value. When the meaning is not relevant, one can make use of the **incremental (leader-cluster) algorithm**, which consists in adding one-at-a-time "K" until *elbow* is shown in the objective (error) function; see the following image.

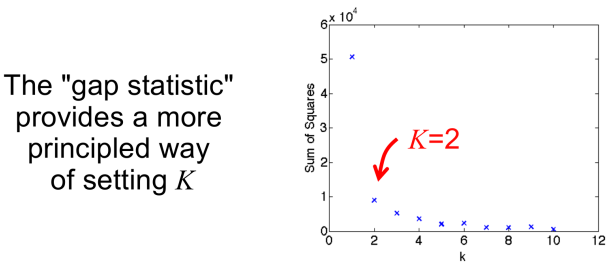


Figure 22: Image from lecture 4.

Professor Q.: *Discuss pros/problem cases/extensions of K-means.*

The positive aspect of this algorithm is that it is fast and converges to a local minimum of within-cluster squared error. However, it is extremely sensitive to the initialization of the centroids and outliers; furthermore, it can only detect spherical/circular clusters.

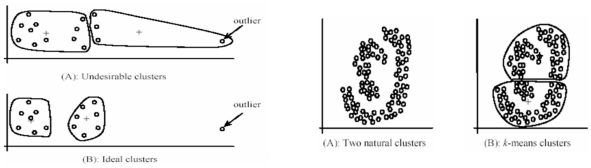


Figure 23: Image from lecture 4.

For extensions, there are

- Speed-ups: possible through efficient search structures, and
- General distance measures: k-medoids.

4.3 Mixture Distributions: Mixture of Gaussians

The Mixture of Gaussians algorithm attempts to approximate an observed smooth distribution as a sum of M individual latent normal distributions. Additive mixture models are the most basic possible latent variable model, having one single discrete latent variable.

To understand how the Gaussian mixture model works, let us first discuss the supervised case, where GMMs become discriminant analysis.

As mentioned before, a discriminative model aims to model $p(y \mid x)$, while a generative model is more ambitious and aims to model the joint probability distribution $p(x, y)$. GMMs is a generative model. From Probability Theory, it is well known that

$$p(x, y) = p(y)p(y \mid x) \tag{14}$$

Gaussian mixture models characterize for assuming that $p(y \mid x) \sim \mathcal{N}(x \mid \mu_y, \Sigma_y)$ is Gaussian. This model is focused on classification/clustering; hence, $p(y = m) = \pi_m$, where $m = 1, \ldots, M$ is denoting for the m-th cluster – each cluster or category is assumed to have its own marginalized discrete probability density distribution. With this, it is possible to note that the parameters of this model are $\boldsymbol{\theta} = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m\}_{m=1}^{M}$. By setting the marginal class probabilities as

$$\pi_m = \frac{n_m}{n}, \tag{15}$$

where n_m is the total number of observations in the m-th cluster, one finds that the problem has a closed-form solution! Notice that this implies that the probability that an observation belongs to class m is proportional to the amount of this class in the training data [3]. It is straightforward that the mean vector of each class $\boldsymbol{\mu}$ is just the empirical mean

$$\boldsymbol{\mu}_m = \frac{1}{n_m} \cdot \sum_{i \in C_m} \mathbf{x}_i. \tag{16}$$

Similarly, the covariance matrix is[5]

$$\boldsymbol{\Sigma}_m = \frac{1}{n_m} \sum_{i \in C_m} (\mathbf{x}_i - \boldsymbol{\mu}_m) (\mathbf{x}_i - \boldsymbol{\mu}_m)^T. \tag{17}$$

From this, one can make predictions from the conditional distribution as

$$p(y \mid X) = \frac{p(x, y)}{p(x)} = \frac{p(x, y)}{\sum_m p(x, y = m)} \tag{18}$$

so that

$$p(y \mid X) = \frac{\pi_m \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{19}$$

From this, it follows the well-known hard assignation of class based on the greatest probability, namely

[5] Think: what does it mean that it is divided by n_m and not $n_m - 1$?

$$\hat{y}(\mathbf{x}_*) = argmax_m \left[p(y = m \mid \mathbf{x}_*) \right]. \tag{20}$$

The maximization on the joint probability and the log-probability yields the same result, with the latter being easier to manage, mathematically speaking. When taking the logarithm, it is easy to see that there will be a quadratic dependence for the observations on $\mathbf{x}$. It is for this reason that the supervised version of Gaussian Mixture Models is referred to as *Quadratic Discriminant Analysis*, and *Linear Discriminant Analysis* on the assumption that all covariance matrices are the same for all classes [3].

Professor Q.: *Does the MoG model has (a) discrete and/or continuous latent variable(s)? Explain.*

The unsupervised case means that the target data is absent, which is categorical; hence, the latent variables are discrete, which belongs to the labeling of the clusters. It is also, for this reason, the assumption that each cluster has its own discrete probability distribution function.

Professor Q.: *Is a closed-form analytical Maximum Likelihood estimation (MLE) possible for a MoG model? Explain.* Do the mathematics.

Unsupervised Mixture of Gaussians
Mixture of Gaussians can be used as a clustering algorithm when the labeled data is accessible. Recall that the model is as follows

$$p(x, y) = p(y)p(y \mid \mathbf{x}) = \pi_y \mathcal{N}(y \mid \mathbf{x}, \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y) \tag{21}$$

The issue is that we do not know what π_m is since the target variable is a latent variable. This results in a non-closed-form conditional probability $p(y \mid \mathbf{x})$; for this reason, the Expectation-Maximization (EM) is used. The EM algorithm is a general numerical tool (NOT a cost function) to solve a maximum likelihood probabilistic problem when there are latent variables [3]. In this case, the challenge is that the likelihood is not accessible; hence, evaluation of the log-likelihood requires the marginalization of the latent variables, here y. The EM algorithm addresses the challenge by alternating between

- the expected log-likelihood (a.k.a. **credit assignment problem**), and
- maximizing this value to update the parameters (**a.k.a. mixture density estimation**)

until there is convergence [3].

Professor Q.: *Discuss the EM algorithm for MoG: discuss (1) the "Credit Assignment Problem" and (2) the "Mixture Density Estimation."*

The overall algorithm is as follows:

REQUIEREMENTS: Data; M, number of clusters; and, $\boldsymbol{\theta}_0 = \{\pi_m, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m\}_{m=1}^M$, initialization of all centroids.

Repeat Until Convergence:

- **Expectation**: compute the conditional probability

$$p(y = m \mid \mathbf{x}) = \frac{\pi_m \cdot \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)}{\sum_{j=1}^M \pi_j \cdot \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{22}$$

- set: $w_i(m) \leftarrow p(y_i = m \mid \mathbf{x}_i)$.
- **Maximization**: maximize the likelihood by updating the parameters

$$\begin{aligned} \pi_m &\leftarrow \frac{1}{n} \sum_{i=1}^N w_i(m) \\ \mu_m &\leftarrow \frac{1}{\sum_{i=1}^N w_i(m)} \sum_{i=1}^N w_i(m) \cdot x_i \\ \Sigma_m &\leftarrow \frac{1}{\sum_{i=1}^N w_i(m)} \sum_{i=1}^N w_i(m) \cdot (x_i - \mu_m)(x_i - \mu_m)^T. \end{aligned} \tag{23}$$

The only difference with the supervised learning to this is that instead of doing a hard clustering assignment based on the targets, a soft assignment is done through the weights $w_i(m)$ [3].

Professor Q.: *Is regularization needed in EM? Why? How? In practice, how to calculate Mixture of Gaussians? Give tips/tricks (technical advice).*

Given this technique, it is possible that the algorithm assigns a single cluster to a single data point, making its likelihood tend to infinity; there is a way to prevent this by adding a small constant to all the diagonal elements to the covariance matrix; such as, $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma} + \sigma_{min}\mathbb{I}$; this is a form of regularization. In this manner, one enforces a minimum width to the Gaussians.

Since the maximum-likelihood problem **is a non-convex optimization problem**, the EM algorithm may converge in a poor solution if the initialization of the cluster's centroids is poor **ML**. There is the technical advice of first utilizing

$k-means$ as **initialization** to improve results. k-Means is initialized randomly and will also find a local optimum, but its convergence is much faster.

Typical EM procedure:

- Run k-Means M times (e.g. M = 10-100)
- Pick the best result (lowest error J)
- Use this result to initialize EM
  - Set μ_j to the corresponding cluster mean from k-Means
  - Initialize Σ_j to the sample covariance of the associated data points

Figure 24: Image from lecture 4.

Here are some general characteristics of the EM algorithm:

- EM = general optimization scheme (for parameter estimation) when working with latent variables – NOT a cost function (such as ML) and NOT a model (such as mixture model).
- EM can potentially make large steps with **ensured improvement of likelihood in each iteration** until convergence, contrary to gradient descent.
- EM is slower w.r.t. gradient descent if there is still much uncertainty about latent variables because it has to be conservative about the bounds.
- EM can get stuck in local minima, and there is the risk of singularities when no regularization is done.
- **AGAIN:** EM = extension of Maximum Likelihood parameter estimation (frequentist approach) if latent variables are explicit.

Professor Qs.: *Explain the key differences between MoG and K-Means clustering. Discuss MoG properties and problems. Draw and explain a MoG generative model as a directed graphical model. Explain all variables.*

K-Means clustering essentially corresponds to a Gaussian Mixture Model (MoG/GMM) estimation with EM whenever:

- The covariances of the K Gaussians are set to $\Sigma_j = \sigma^2 I$ ("spheres", no "ellipsoids")
- For some small, fixed σ^2 ("hard" boundaries)

Gaussian Mixture Models - Summary

- **Properties ☺**
  - Very general, can represent any (continuous) distribution
  - Once trained, very fast to evaluate
  - Can be updated online
- **Problems / Caveats ☹**
  - Some numerical issues in the implementation
    - ➔ Need to apply regularization in order to avoid singularities
  - EM for MoG is computationally expensive
    - Especially for high-dimensional problems!
    - More computational overhead and slower convergence than k-Means
    - Results very sensitive to initialization
    - ➔ Run k-Means for some iterations as initialization!
  - Need to select the number of mixture components K
    - ➔ Model selection problem

Figure 25: Image from lecture 4.

5 The Data Pipeline (DP) and Ramp-Up Towards Non-Linear Models

When having categorical data, one part of the data pipeline is to convert them to numerical values by using *OrdinalEncoders* (1, 2, 3, ...) or *OneHotEncoders* (0,1; 00, 01, 10, 11; 100, 010, 001, ...). A general pipeline is of the following form

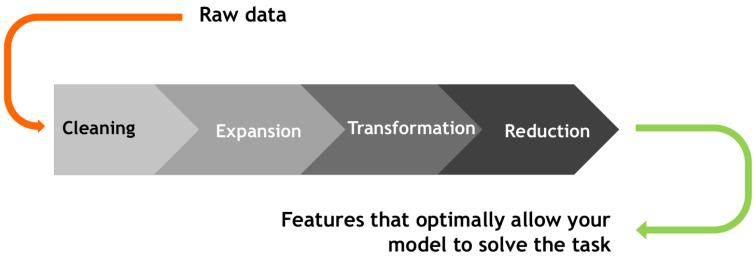
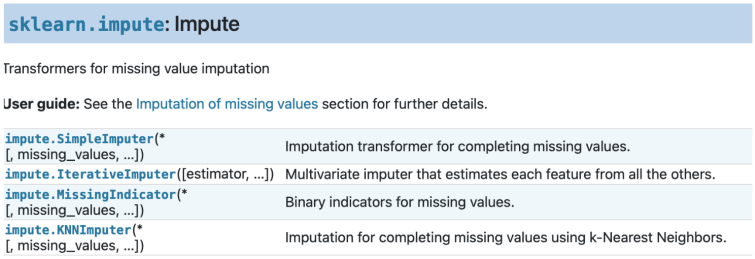


Figure 26: Image from lecture 5.

Sometimes, augmentation might be required at the beginning to increase the size. Remember that many steps in the data-pipeline use parameters that are extracted from the whole training set, these MUST be included in your sklearn cross-validation pipeline to avoid leakage!

5.1 DP: Cleaning

Sometimes, with outliers, missing or zero values, action must be taken for optimal algorithm performance; for instance, filling missing values with the mean of a specific group and/or date-time. The data cleaning will always depend on the specific problem at hand. Here are some options implemented by sklearn.



sklearn.impute: Impute

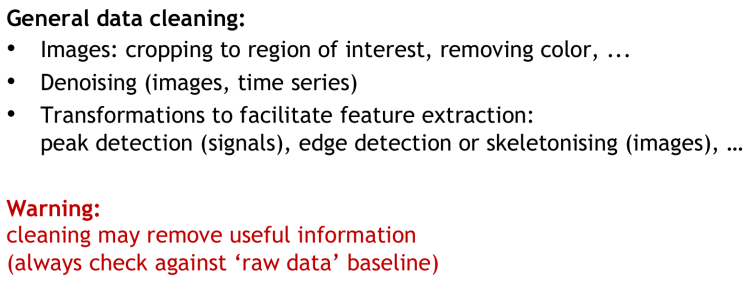
Transformers for missing value imputation

User guide: See the Imputation of missing values section for further details.

impute.SimpleImputer(* [, missing_values, ...])	Imputation transformer for completing missing values.
impute.IterativeImputer([estimator, ...])	Multivariate imputer that estimates each feature from all the others.
impute.MissingIndicator(* [, missing_values, ...])	Binary indicators for missing values.
impute.KNNImputer(* [, missing_values, ...])	Imputation for completing missing values using k-Nearest Neighbors.

Figure 27: Image from lecture 5.

While outliers can damage the algorithm's performance, sometimes it is not correct to remove them from the data; sometimes, the correct approach is to keep them and aim for a more robust algorithm. This depends strongly on the problem at hand. The cleaning part is the opposite of augmentation, as the former aims to ease the complexity of the data while the latter increases it.



General data cleaning:

- Images: cropping to region of interest, removing color, ...
- Denoising (images, time series)
- Transformations to facilitate feature extraction:
  peak detection (signals), edge detection or skeletonising (images), …

Warning:
cleaning may remove useful information
(always check against 'raw data' baseline)

Figure 28: Image from lecture 5.

Data cleaning = removing (unnecessary) information. Data augmentation = adding (necessary) information.

Professor Q.: *What is (or can be) the impact of outliers on your model? What are ways to deal with outliers? Give examples of situations where you would make different choices.*

Professor Q.: *Preprocessing: what is "standardization" or "feature normalization" in a machine learning pipeline? Why/when do you need it? What is the difference between sklearn "scalers", "normalizers" and power transformers? When would you use which?*

5.2 DP: Feature Extraction / Expansion

Think: how does it compare to cleaning vs preprocessing vs transformations?

When features are not good enough for a model, sometimes it is required that they get transformed (non-linearly or linearly, e.g. polynomials) to obtain their useful characteristics. It is good to remember that many models do not like sparse features. Unsupervised learning can also be utilized to extract new features.

Many 'traditional' image processing feature extractors exist (see, e.g., Wikipedia for descriptions):
- HOG, SIFT, SURF, GLOH, Hough transform
- some detect lines, or edge orientations
- some detect (or help to detect) specific shapes (e.g. circles)

Pretrained deep neural networks
- can also be used as feature extractors
- mostly yield higher-level features

5.3 DP: Transformations and Embeddings

"An embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words." [11]

Professor Q.: *What is feature "orthogonalization"? How/why can it help? What is the difference between orthogonalization and dimensionality reduction?*

5.3.1 Principal Component Analysis: unsupervised

The curse of dimensionality is when there is a term for the exponential growth in computational efforts in processing and/or analysis of data [12]. For this reason, it is encouraged to find suitable ways to represent high-dimensional data in a lower dimension, PCA is a technique that achieves that by considering the information encoded in the data's variance. The algorithm is the following

Learn the PCA model

Data: Training data $\mathcal{T} = \{\mathbf{x}_i\}_{i=1}^n$
Result: Principal axes $\mathbf{V}$ and scores $\mathbf{Z}_0$

1 Compute the mean vector $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$
2 Centre the data, $\mathbf{x}_{0,i} = \mathbf{x}_i - \bar{\mathbf{x}}$, for $i = 1, \ldots, n$
3 Construct the data matrix $\mathbf{X}_0$ according to (10.35)
4 Perform SVD on $\mathbf{X}_0$ to obtain the factorisation $\mathbf{X}_0 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$
5 Compute principal components $\mathbf{Z}_0 = \mathbf{U}\mathbf{\Sigma}$

Method 10.5: Principal component analysis

Figure 29: Image from [3]

The algorithm is just simply performing singular value decomposition (SVD) to the **centered data matrix $\mathbf{X}_0$**. The orthogonal matrix $\mathbf{V}$ corresponds to a change of basis in $\mathbb{R}^p$ (p being the total number of features). Here $\mathbf{Z}_0 = \mathbf{X}_0\mathbf{V}$ is the alternative way to represent the data without a loss of information [3]. The essence of this analysis lies in the fact that **the columns of $\mathbf{V}$ are ordered in terms of relevance w.r.t. encoding q-dimensional data,** $q < p$. In other words, by simply keeping the first q columns of $\mathbf{Z}_0$, one obtains a low-dimensional representation of $\mathbf{X}_0$. The first principal axis is the direction with the largest variance, the second (orthogonal w.r.t. the previous) is the second direction with the most variance, and so on [3].

REMARKS

- PCA is sensitive to scaling: small but relevant features may be suppressed!
- Output of PCA: features scaled according to eigenvalues → may yield features with very different magnitudes; hence, it is safest to use a scaler before AND after PCA.
- The Proportion of Variance (PoV) is explained by the components.
- If you keep all PCA components, PCA is invertible.

BENEFITS

- can be used as feature selection
- improves explainability of the model: correlated features are often exchangeable, decorrelated features are not
- PCA identifies dimensions with low variance:
  - if all samples are very similar in a given linear subspace, this may be an indication that this subspace is less useful
  - but: PCA is unsupervised(!!) so be careful with conclusions here PCA feature importance (eigenvalue) not necessarily related to task importance
- PCA is useful to visually decide whether a feature subset is useful:
  - Plot first 2 or 3 principal components of feature set
  - Color code with labels
  - Check if labels are well-separated

It is possible to check whether a cut in features is necessary or useful with *the elbow method*: plot PoV vs eigenvalues.

5.3.2 Linear Discriminant Analysis: supervised

LDA is a supervised generative model that allows computing the class probabilities using linear discriminants. Since probabilities add up to 1: only k-1 discriminants for k classes

Properties

- Each discriminant is a linear combination of original features
- So can use these as linearly transformed features z
- These features are **not orthogonal**

1.2.2.2. LDA

LDA is a special case of QDA, where the Gaussians for each class are assumed to share the same covariance matrix: $\Sigma_k = \Sigma$ for all k. This reduces the log posterior to:

$$\log P(y=k|x) = -\frac{1}{2}(x-\mu_k)^t\Sigma^{-1}(x-\mu_k) + \log P(y=k) + Cst.$$

The term $(x-\mu_k)^t\Sigma^{-1}(x-\mu_k)$ corresponds to the Mahalanobis Distance between the sample x and the mean μ_k. The Mahalanobis distance tells how close x is from μ_k, while also accounting for the variance of each feature. We can thus interpret LDA as assigning x to the class whose mean is the closest in terms of Mahalanobis distance, while also accounting for the class prior probabilities.

The log-posterior of LDA can also be written [3] as:

$$\log P(y=k|x) = \omega_k^t x + \omega_{k0} + Cst.$$

where $\omega_k = \Sigma^{-1}\mu_k$ and $\omega_{k0} = -\frac{1}{2}\mu_k^t\Sigma^{-1}\mu_k + \log P(y=k)$. These quantities correspond to the `coef_` and `intercept_` attributes, respectively.

From the above formula, it is clear that LDA has a linear decision surface. In the case of QDA, there are no assumptions on the covariance matrices Σ_k of the Gaussians, leading to quadratic decision surfaces. See [1] for more details.

1.2.3. Mathematical formulation of LDA dimensionality reduction

First note that the K means μ_k are vectors in $\mathcal{R}^d$, and they lie in an affine subspace H of dimension at most $K-1$ (2 points lie on a line, 3 points lie on a plane, etc.).

As mentioned above, we can interpret LDA as assigning x to the class whose mean μ_k is the closest in terms of Mahalanobis distance, while also accounting for the class prior probabilities. Alternatively, LDA is equivalent to first *sphering* the data so that the covariance matrix is the identity, and then assigning x to the closest mean in terms of Euclidean distance (still accounting for the class priors).

Computing Euclidean distances in this d-dimensional space is equivalent to first projecting the data points into H, and computing the distances there (since the other dimensions will contribute equally to each class in terms of distance). In other words, if x is closest to μ_k in the original space, it will also be the case in H. This shows that, implicit in the LDA classifier, there is a dimensionality reduction by linear projection onto a $K-1$ dimensional space.

We can reduce the dimension even more, to a chosen L, by projecting onto the linear subspace H_L which maximizes the variance of the μ_k^* after projection (in effect, we are doing a form of PCA for the transformed class means μ_k^*). This L corresponds to the `n_components` parameter used in the `transform` method. See [1] for more details.

Figure 30: LDA info based on [13].

PCA:
- unsupervised, only looks at directions of strongest variability in data
- optimised target = reconstruction of original data
- can be used without reducing number of features
- linear transformation
- new features 'orthogonal'

LDA:
- supervised , takes labels into account
- optimised target = optimal classification(s)
- returns at most k-1 features for k classes!
- linear transformations
- new features not 'orthogonal'! *(just happens to be the case in example above)*

Figure 31: Image from lecture 5.

5.4 DP: Dimensionality Reduction

Professor Q.: *What is dimensionality reduction? What is its purpose? What are the trade-offs you need to make? Dimensionality reduction: unsupervised vs. supervised techniques, "kbest" subset selection, recursive feature addition/removal,...*

Most important: simpler models are more robust to overfitting (on too small datasets) – lower variance! It also reduces the computational time in improves inference (curse of dimensionality).

Dimensionality reduction means considering a subset of all the available features or using a 'compressive' transformation (e.g. LDA or PCA + cut-off).

Typical workflow:

- first consider many features,
- transform them to compress information,
- select the best ones (subset selection)

IMPORTANT: there are 2^F subsets of F features.

To study features, you can add the best features one by one (forward analysis) or start with all and remove them one by one (backward analysis). One can also use **feature importance** by permutations; other methods could be implemented like L_1 regularization, and so on. There are many methods to investigate feature importance and effectively reduce dimensionality. Sklearn offers many built-in methods.

sklearn.feature_selection: Feature Selection

The sklearn.feature_selection module implements feature selection algorithms. It currently includes univariate filter selection methods and the recursive feature elimination algorithm.

User guide: See the Feature selection section for further details.

feature_selection.GenericUnivariateSelect([...])	Univariate feature selector with configurable strategy.
feature_selection.SelectPercentile([...])	Select features according to a percentile of the highest scores.
feature_selection.SelectKBest([score_func, k])	Select features according to the k highest scores.
feature_selection.SelectFpr([score_func, alpha])	Filter: Select the pvalues below alpha based on a FPR test.
feature_selection.SelectFdr([score_func, alpha])	Filter: Select the p-values for an estimated false discovery rate.
feature_selection.SelectFromModel(estimator, *)	Meta-transformer for selecting features based on importance weights.
feature_selection.SelectFwe([score_func, alpha])	Filter: Select the p-values corresponding to Family-wise error rate.
feature_selection.SequentialFeatureSelector([...])	Transformer that performs Sequential Feature Selection.
feature_selection.RFE(estimator, *[, ...])	Feature ranking with recursive feature elimination.
feature_selection.RFECV(estimator, *[, ...])	Recursive feature elimination with cross-validation to select the number of features.
feature_selection.VarianceThreshold([threshold])	Feature selector that removes all low-variance features.
feature_selection.chi2(X, y)	Compute chi-squared stats between each non-negative feature and class.
feature_selection.f_classif(X, y)	Compute the ANOVA F-value for the provided sample.
feature_selection.f_regression(X, y, *[, center])	Univariate linear regression tests returning F-statistic and p-values.
feature_selection.r_regression(X, y, *[, center])	Compute Pearson's r for each features and the target.
feature_selection.mutual_info_classif(X, y, *)	Estimate mutual information for a discrete target variable.
feature_selection.mutual_info_regression(X, y, *)	Estimate mutual information for a continuous target variable.

Figure 32: Image from lecture 5.

5.5 Introduction to Kernels

As already discussed, it is possible that a non-linear boundary exists for classified data in a given dimension and a linear boundary for the same dataset for another dimension, as shown in the following example.

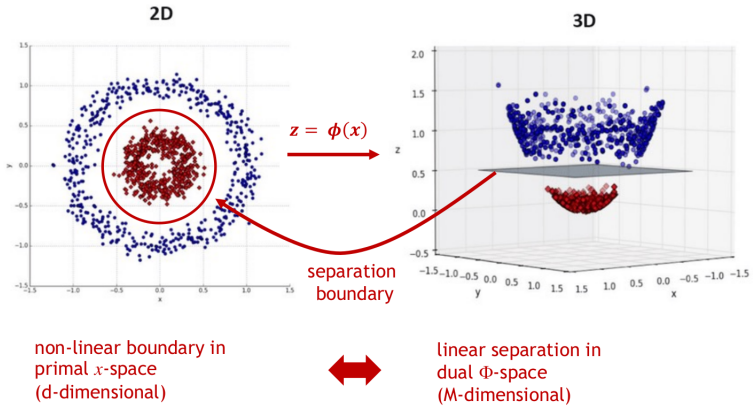


Figure 33: Image from lecture 5.

Here, the primal space d-dimensional space of the original data $\mathbf{x}$ (d is the number of features). The dual space is the M-dimensional space of $\Phi(\mathbf{x})$ where the linear separation exists. One can perform regression or classification in either the primal or dual space.

In several machine learning algorithms, one can find the explicit expression of the inner product between the training examples and the new data when computing $\hat{y}(\mathbf{x'})$. If the tasks require the algorithm's learning on the transformed features, then one would need to compute the inner product of transformed feature vectors $\Phi(\mathbf{x})^T\Phi(\mathbf{x'})$; here is where the concept of kernels comes in hand.

A kernel $K(\mathbf{x}, \mathbf{x'})$ is an inner product between two vectors in some vector space in which the vectors $\Phi(\mathbf{x})$ are transformed versions of $\mathbf{x}$. The only requirement is that the kernel can be expressed as an inner product; for instance

$$K(\mathbf{x}, \mathbf{x'}) = \Phi(\mathbf{x})^T\Phi(\mathbf{x'}) = \sum_{j=1}^{M} \phi_j(\mathbf{x})\phi_j(\mathbf{x'}) \tag{24}$$

which implies that it is always a symmetric function of two d-dimensional vectors; hence, its result is a scalar, not a vector. Have a look at *the kernel trick* for linear regression on transformed features

$$y(\boldsymbol{x}') = \sum_{t=1}^{N} \alpha_t \; r_t\big(\boldsymbol{\phi}(\boldsymbol{x}_t)^T\boldsymbol{\phi}(\boldsymbol{x}')\big) + w_0 \; = \sum_{t=1}^{N} \alpha_t \; r_t \, \mathbf{K}(\boldsymbol{x}_t, \boldsymbol{x}') + w_0$$

By rewriting a linear model in inner product form, it no longer depends on $\boldsymbol{\phi}(\boldsymbol{x}')$, but only on the kernels

If we know the kernel, we don't need to know the feature space explicitly, ***we only need to know it exists!***

Figure 34: Image from lecture 5.

This means there is no need to compute the dual space for effective learning! This property makes inference faster and better; for instance, polynomial regression is more effective with a q-polynomial kernel $K(\mathbf{x}, \mathbf{x'}) = (\mathbf{x}^T\mathbf{x'} + 1)^q$ because computing the kernel function is less costly than computing the transformation and the inner product.

In short: a positive definite symmetric function is a kernel, meaning: $k(v, w) = k(w, v)$.

Let X be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called positive definite kernel function, iff

k is symmetric, i.e. $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$, and for any set of points $x_1, \ldots, x_n \in \mathcal{X}$, the matrix

$$K_{ij} = (k(x_i, x_j))_{i,j}$$

is positive (semi-)definite, i.e. for all vectors $\mathbf{x} \in \mathbb{R}^n$:

$$\sum_{i,j=1}^{N} \mathbf{x}_i K_{ij} \mathbf{x}_j \geq 0$$

Figure 35: Image from lecture 5.

Give examples of other kernels and their properties.

Explain how comparing angles is faster with kernels than its computation in dual space.

Give memory storage of kernels and dual space and mention the main drawbacks of its implications.

Kernels can be used to introduce non-linearity whenever **a model only depends on the data through the inner products** between the new feature vector and the training feature vectors.

Mention the main difference between KNN, SVM, and kernel SVM.

5.6 Summary

6 Directed and Undirected Graphical Models

Professor Q.: *Describe the key properties of graphical models. Why useful?*

This lecture outline:

- **Generative Models**: model the likelihood and prior distributions separately.
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference
- **Discriminative Approaches**: model the posterior distribution
  - Ensemble methods and boosting
  - Random Forests and Decision Trees

To understand the content of this lecture, it is necessary to review the requirements in Probabilistic Models.

A probability is an expression of belief about an uncertain event; this proves useful for decision-making and combining sources of information. A key quantity in probabilistic reasoning is the **joint distribution** $p(x_1, \ldots, x_k)$ of k random variables in a model. It is possible to classify machine learning tasks in terms of the joint distribution:

- **inference**: given the joint distribution, queries are done. When considering a boolean case with k entries, then summations take $\mathcal{O}(2^k)$ and inference is slow for a large k;
- **learning**: modeling the joint distribution (requires inference). For the same boolean case, it implies that 2^k parameters must be learned.

Both require the manipulation of the joint distribution. Here are the main rules of probability that must be kept in mind:

- **Sum rule**: is the marginalization over discrete or continuous random variables

$$p(X) = \sum_y p(X, Y = y); \quad \text{or} \quad p(X) = \int p(X, Y = y) dy \tag{25}$$

- **Product rule**:

$$p(X, Y) = p(X)p(Y \mid X) \tag{26}$$

- **Bayes rule**:

$$p(Y \mid X) = \frac{p(X \mid Y)p(Y)}{p(X)} \tag{27}$$

- **Conditional Independence**: X and Y are independent iff
  - $p(X \mid Y) = p(X)$ and $p(Y \mid X) = p(Y)$, or
  - $p(X, Y) = p(X)p(Y)$

  In many cases, the use of conditional independence greatly reduces the size of the representation of the joint distribution, which directly reduces the number of parameters to be learned in inference or increases the computational speed in inference.

6.1 Introduction to Graphical Models

Professor Q.: *What is "Causal inference"? What is "Diagnostic inference"?*

Graphical models characterize for merging Probability Theory with Graph Theory, providing a natural tool for uncertainty and complexity problems by giving a graphical representation of probability distributions.

Graphical models are composed by two elements:

- **nodes** (also called vertices) represent observed or unobserved**random variables** or groups of random variables;

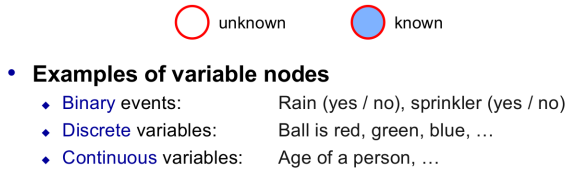


Figure 36: Image from lecture 6.

and,

- **edges** (also called links) represent **conditioning**; they can be directed (known as Bayesian Networks) or undirected (Markov Random Fields).

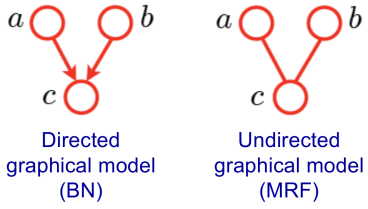


Figure 37: Image from lecture 6.

6.2 Directed Graphical Models: Bayesian Networks

Bayesian networks are based on a directed acyclic graph (DAG); however, they can be fully connected, known as associated graphs. Because this type of model links are directed, they represent causal dependencies among the (observed or unobserved) random variables. The structure of the network qualitatively describes the dependencies of the random variables.

Professor Q.: *How to get a factorized representation of the joint distribution in directed graphical models?*

Joint Probability in Directed Graphs
Consider the node A directed to node B; this means that the value of B depends on A; this dependency is expressed as a conditional probability $p(B \mid A)$; and the probability of A is given by the prior probability $p(A)$. **The complete graphical model describes the joint probability distribution** $p(A, B) = p(A)p(B \mid A)$. Given the joint probability, it is possible to derive the marginalization of A in B and obtain the other conditional distribution; namely,

$$p(B) = \sum_A p(A, B) = \sum_A p(B|A)p(A); \quad \text{and} \quad p(A \mid B) = \frac{p(A, B)}{p(B)}, \tag{28}$$

with the latter being Baye's rule.

Factorization in Directed Graphs
Consider a Bayesian Network consisting of a set of variables $U = x_1, \ldots, x_n$ forming an acyclic-directed graph. The joint distribution of said model can be expressed as

$$p(x_1 \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid \{x_j \mid j \in \text{pa}_i\}), \tag{29}$$

with pa_i denoting the parent nodes of x_i, a factorized representation of the joint. In other words, one can express the joint as a product of all the conditional distributions from the parent-child relations in the graph.

Exercise: Computing the joint probability

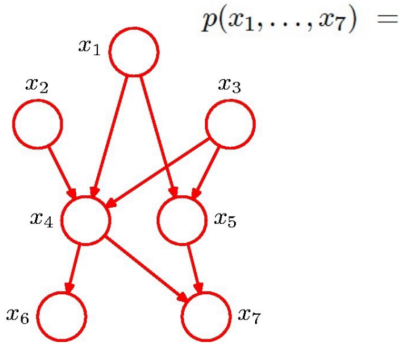


Figure 38: Image from lecture 6.

Overall, having the factorized version of the joint probability allows reducing the complexity of the problem from $\mathcal{O}(2^n)$ to $\mathcal{O}(n \cdot 2^k)$, with k denoting the maximum number of parents of a node in the model. Here is an example of the Intensive Care Unit's alarm network for monitoring patients.

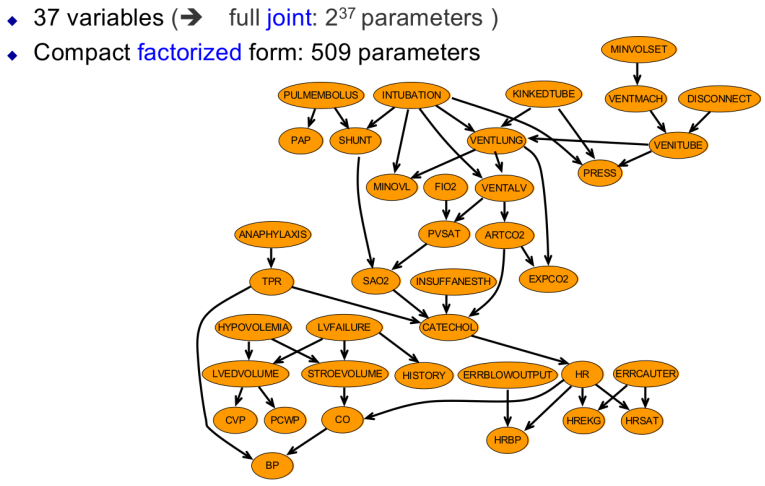


Figure 39: Image from lecture 6.

Professor Q.: *Discuss "conditional independence" in directed graphical models. Discuss the Divergent, Chain and Convergent cases.*

Conditional Independence
The notion of conditional independence means that other variables become independent given the observation of a certain variable; e.g. $p(x_2 \mid x_{0,x}\, 2) = p(x_2 \mid_x 1)$, means that x_2 is conditionally independent from x_0 when x_1 is observed. The nomenclature for conditional independence is the following

- X **is conditionally independent of** Y **given** V

  Equivalence: $X \perp\!\!\!\perp Y|V \Leftrightarrow p(X|Y,V) = p(X|V)$

  Also: $X \perp\!\!\!\perp Y|V \Leftrightarrow p(X,Y|V) = p(X|V)\,p(Y|V)$

- Special case: (**marginal**) **independence**

$$X \perp\!\!\!\perp Y \Leftrightarrow X \perp\!\!\!\perp Y|\emptyset \Leftrightarrow p(X,Y) = p(X)\,p(Y)$$

Figure 40: Image from lecture 6.

In other words, by observing V, the observation of Y is irrelevant for the estimation of X, and vice versa. Furthermore, knowing V explains any observed dependence between X and Y.

It is possible to read the conditional independence from directed graphical models; it is typical to encounter three canonical cases, each composed of three nodes (A, B, C):

- **Tail-to-Tail** (divergent): this model has C as the only parent of A and B ($A \leftarrow C \rightarrow B$). To tell whether A and B are independent, one needs to marginalize out C; namely,

$$\begin{aligned} p(A,B) &= \sum_C p(A,B,C) \\ &= \sum_C p(A \mid C)p(B \mid C)p(C) \\ &\neq p(A)p(B) \rightarrow A \not\perp\!\!\!\perp B \mid \emptyset \end{aligned} \tag{30}$$

  If C is not observed, then one can express the conditional probability as

$$p(A,B \mid C) = \frac{p(A,B,C)}{p(C)} = \frac{p(A \mid C)p(B \mid C)p(C)}{p(C)} = p(A \mid C)p(B \mid C) \rightarrow A \perp\!\!\!\perp B \mid C \tag{31}$$

  The latter is because there is no link between B and A; with the former equation, one observes that A and B are conditionally independent!

- **Head-to-Tail** (chain): this model has A as the unique partent of C and C the unique parent of B ($A \rightarrow C \rightarrow B$). Again, are A and B generally independent?

$$\begin{aligned} p(A,B) &= \sum_C p(A,B,C) \\ &= \sum_C p(A)p(C \mid A)p(B \mid C) \\ &\neq p(A)p(B) \rightarrow A \not\perp\!\!\!\perp B \mid \emptyset \end{aligned} \tag{32}$$

  What happens if C is observed?

$$p(A,B \mid C) = \frac{p(A,B,C)}{p(C)} = \frac{p(A)p(C \mid A)p(B \mid C)}{p(C)}, \text{ with } p(C \mid A) = \frac{p(A \mid C)p(C)}{p(A)} \rightarrow A \perp\!\!\!\perp B \mid C \tag{33}$$

  it is possible to observe that A and B are conditionally independent when C is observed.

- **Head-to-Head** (Convergent): this model has A and B being the two parents for C ($A \rightarrow C \leftarrow B$). Are A and B independent?

$$p(A,B) = \sum_C p(A,B,C) = \sum_C p(A)p(B)p(C \mid A,B) = p(A)p(B) \rightarrow A \perp\!\!\!\perp B \mid \emptyset \tag{34}$$

  What happens if C is observed?

$$p(A,B \mid C)\frac{p(A,B,C)}{p(C)} = \frac{p(A)p(B)p(C \mid A,B)}{p(C)} \rightarrow A \not\perp\!\!\!\perp B \mid C \tag{35}$$

  Therefore A and B are **not** conditionally independent when C is observed; this also holds when any of C's descendants is observed. This case is the opposite of the previous cases.

  Visit the explanation on conditional independence given the example in the lecture (cloudy days).

Explaining Away
Professor Q.: *What is "explaining away"?*

There are instances where observing a child node changes the probability of a parent(s) node(s) but do not block them. **The phenomenon of explaining away means that observations of the child nodes will not block paths to the co-parents.**

In other words, Bayesian Networks have the capability to explain away hypotheses by new evidence.

The Markov Blanket
Professor Q.: *What is a "Markov Blanket" in directed graphical models?*

A Markov Blanket, or Markov boundary, of a node A is the minimal set of nodes that isolates A from the rest of the graph, comprising the set of parents, children, and co-parents (the other parents of its children) of A.

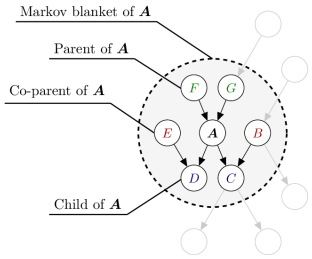


Figure 41: Image from [14].

6.3 Undirected Graphical Models: Markov Random Fields

Markov Random Fields (MRFs) are undirected graphical models which allow easier reading of the conditional independence on their nodes. The Markov Blanket of MRFs are just the neighboring nodes (no need to worry of co-parents).

Professor Q.: *What is a "Markov Blanket" in Undirected graphical models?*

Conditional Independence
If every path from any node in set A to set B passes through at least one node in set C, then $A \perp\!\!\!\perp B \mid C$.

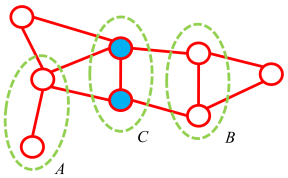


Figure 42: Image from lecture 6.

The analysis just requires simple graph separation. Checking all the paths between sets can tell whether a set its conditional independence. Here is another example between two computers and two hubs.

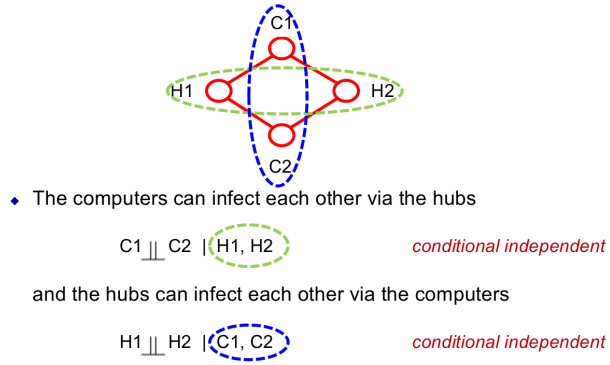


Figure 43: Image from lecture 6.

Professor Qs.: *ow to factorize an undirected graph? What is a (max) clique? Why is a normalization constant needed?*

Factorization in Undirected Graphs
To understand how factorization is done in MRFs, it is important to understand the concept of **clique**. A clique is a

fully connected subset of the nodes; a link exists between all pairs of nodes in the subset. The **maximal clique** is the biggest possible such clique in a given graph.

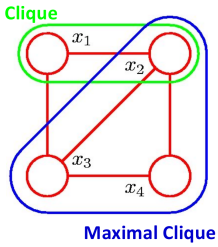


Figure 44: Note that nodes in a clique cannot be made conditionally independent from each other. Image from lecture 6.

The joint distribution of a non-directed graph is expressed as a product of non-negative potential functions $\psi_c(\mathbf{x}_c)$ over maximal cliques in the graph

$$p(\mathbf{x}) = \frac{1}{Z}\prod_c \psi_c(\mathbf{x}_c), \tag{36}$$

with Z being the partition function

$$Z = \sum_{\mathbf{x}}\prod_c \psi_c(\mathbf{x}_c), \tag{37}$$

that normalizes the probability. One of the main limitations of MRFs is the partition function since its evaluation is $\mathcal{O}(K^M)$, where M is the total number of nodes and K is the number of nodes in a given clique (?). Expressing the potential function as an exponential function is convenient due to their similarity in Statistical Physics for the Boltzmann distribution. Furthermore, it is convenient because the joint distribution depends on the product of these components; hence, taking the logarithm to the resultant expression is suitable.

One can observe that Bayesian Networks are automatically normalized, while MRFs are not.

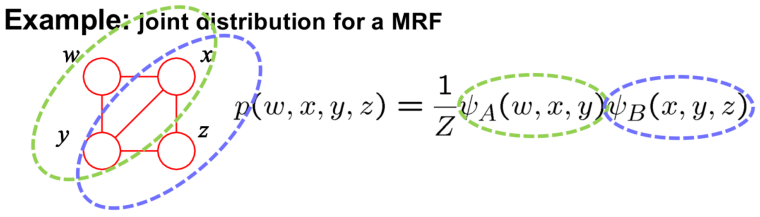


Figure 45: Image from lecture 6.

Example of usage: Ising model for image restoration, segmentation, inpainting, and denoising. MRFs can also be used to convert a low-resolution image into a high-resolution image or create disparity maps.

Professor Q.: *Compare Directed vs. Undirected Graphs: pros/cons*

Professor Qs.: *How to convert a directed to an undirected graph? What is "moralization"? What is a "moral graph"?*

It is possible to convert a directed graph into an undirected graph by considering the necessary cliques to replace the directed links. Furthermore, one must *marry the parents*, or introduce additional links, to account for normalization in the directed graph. This process is known as **moralization**.

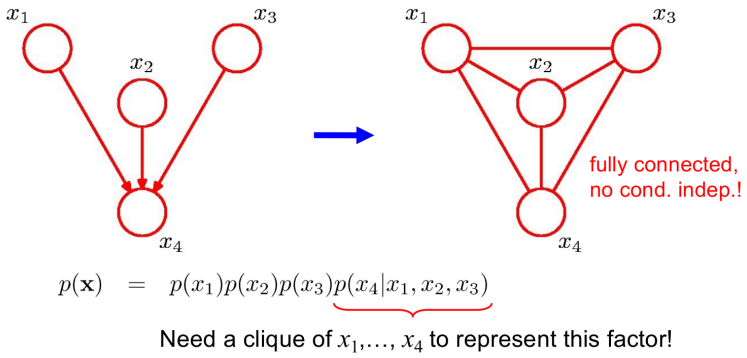


Figure 46: Image from lecture 6.

A **moral graph** is a graph that results from transforming a Bayesian Network into a MRF by moralizing it. Here is the general procedure:

1. Moralize the graph (marry the parents).
2. Drop the arrows of the original graph (obtain a moral graph).
3. Find maximal cliques for each node and initialize all clique potentials to 1.
4. Take each conditional distribution factor of the original directed graph and multiply it into one clique potential.

WARNING: conditional independence properties are often lost and moralization results in additional connections and larger cliques.

EXERCISE: take the Bayesian network in Figure 38 and transform it into a Markov Random Field.

6.4 Exact Inference in Graphical Models

Professor Q.: *What is inference in graphical models (only high-level description: general definition and general question formulation $p(Q \mid E)$: max 10 lines)?* Watch the lecture, if possible.

6.5 Summary

Keywords: associated graphs, Markov blanket, clique, potential function, partition function, moralization

7 Bayesian Estimation

7.1 Recap: Regression

Briefly explain linear regression and Ridge regression (L_2 regularization).

NEXT: Bayesian approach:

Professor Qs.: *How can we prevent overfitting systematically? How can we avoid the need for validation on separate test data?*

7.2 Probability Concepts

Recall the sum rule, product rule, Baye's rule, and marginalization – the basic rules of probability. The following terms must be understood:

- **Prior**: a priori probability; namely, what do we know before seeing the data?
- **Conditional probability**: the probability of observing a random variable given the observation of another one.
- **Interpretation of Bayes Theorem/Rule**:

$$posterior = \frac{likelihood \times prior}{Normalization\ factor} \tag{38}$$

- **One-dimensional Gaussian Distribution**:

- Mean μ
- Variance σ^2

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

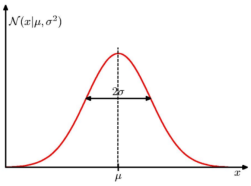
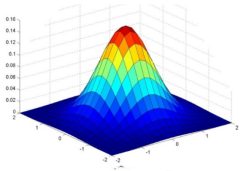


Figure 47: Image from lecture 7.

- **Multi-dimensional Gaussian Distribution**:

- Mean $\boldsymbol{\mu}$
- Covariance $\boldsymbol{\Sigma}$



$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

Figure 48: Image from lecture 7.

IMPORTANT: the inverse of the covariance matrix is known as the precision matrix: $\Lambda = \Sigma^{-1}$. Write the terms of the one- and multi-dimensional Gaussian distribution in terms of precision beside their images.

- **Special Cases of Gaussian Distributions** Draw them!

- **Central Limit Theorem**: The distribution of the sum of N i.i.d. random variables becomes increasingly more Gaussian as N grows.
- **Maximum Likelihood**: estimation of the parameters $\boldsymbol{\theta}$ through the minimization of the negative log-likelihood. Applying maximum likelihood to estimate the parameters of a Gaussian yields

$$\begin{aligned} \mu_{ML} &= \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{sample mean} \\ \sigma^2_{ML} &= \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{ML})^2 \quad \text{(biased) sample variance} \\ \tilde{\sigma}^2_{ML} &= \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_{ML})^2 \quad \text{(unbiased) sample variance} \end{aligned} \tag{39}$$

One of this approach's limitations is that **it systematically underestimates the variance of the distribution**, resulting in a tendency to overfit the training data.

Professor Qs.: *Does Maximum Likelihood (ML) estimation for the parameters of a Gaussian distribution under or overestimate the true mean and/or variance? Explain + give math expressions. What happens if there is only one sample?*

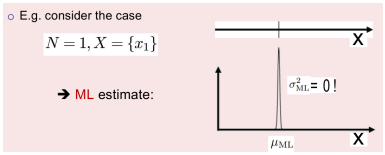


Figure 49: Image from lecture 7.

This method is a Frequentist concept, meaning that probabilities are the frequencies of random, repeatable events – it refers to *past* events; these frequencies are fixed but can be estimated more precisely when more data is available. **This is in contrast to the Bayesian interpretation**. In the Bayesian view, probabilities quantify the uncertainty about certain states or events – it refers to the *future*. This uncertainty (measures of belief) can be revised in the light of new evidence.

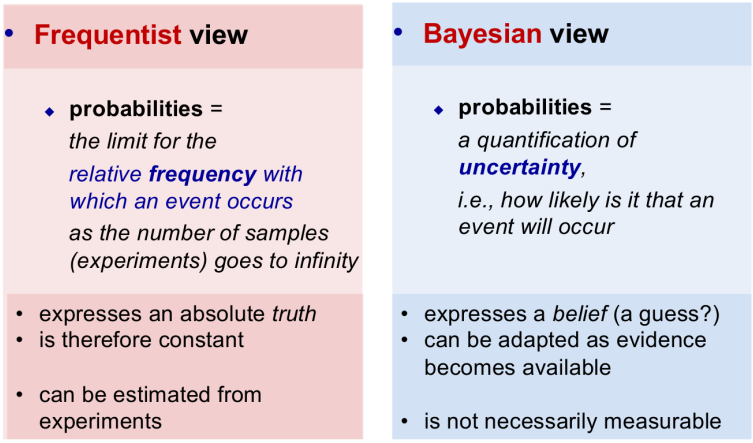


Figure 50: Image from lecture 7.

Bayesian Learning Approach
One of the main differences between Bayesian Learning and Frequentist Learning is that in the former visualizes $\boldsymbol{\theta}$ as a random variable, while the latter perceives it as a fixed parameter that can be estimated. In Bayesian learning, $p(\boldsymbol{\theta})$ is the prior distribution. The probability density of the parameters after the training data is observed $p(\theta \mid x)$ is the posterior probability density.

Bayesian Learning is a Generative Model since the prediction accounts for finding the probability distribution for a new input value x conditioned on the observed training data. It is possible to analyze this without labels, such as

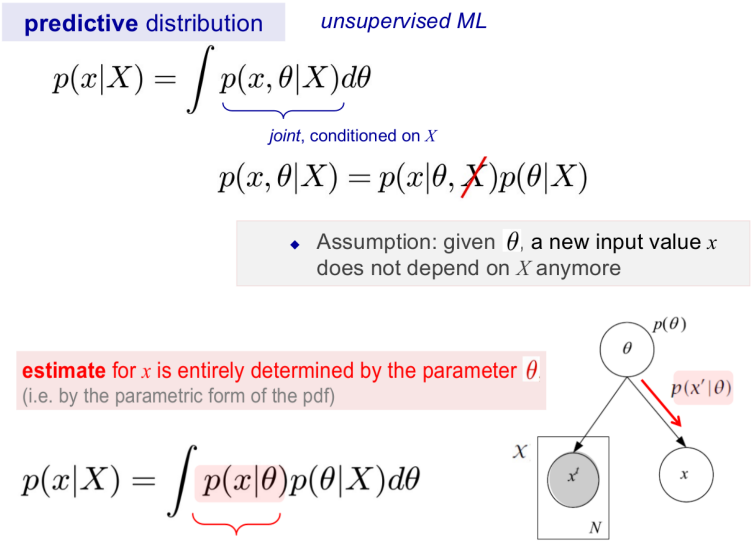


Figure 51: Image from lecture 7.

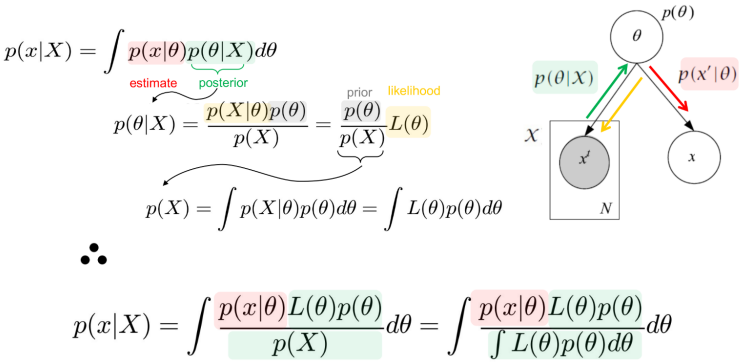


Figure 52: Image from lecture 7.

The wrap-up of Bayesian Learning is that the more uncertain one is about θ, the more one averages over all possible parameter values.

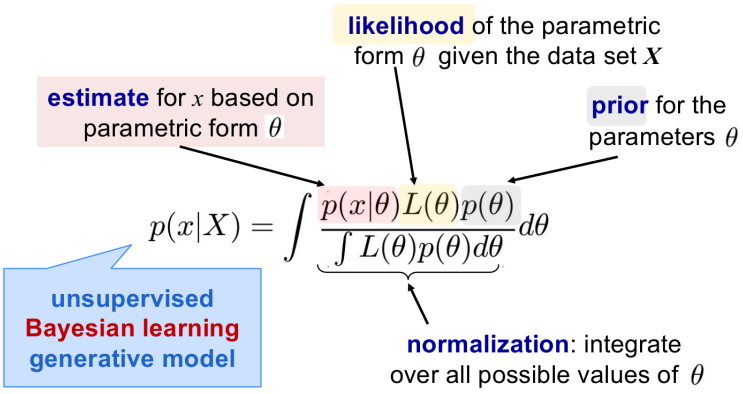


Figure 53: Image from lecture 7.

Professor Q.: *Bayesian learning: Draw and explain a generative model (as a directed graphical model) in the unsupervised case, and give the predictive probability $p(x' \mid x)$ for a new data sample x', conditioned on the observed (training) data x*

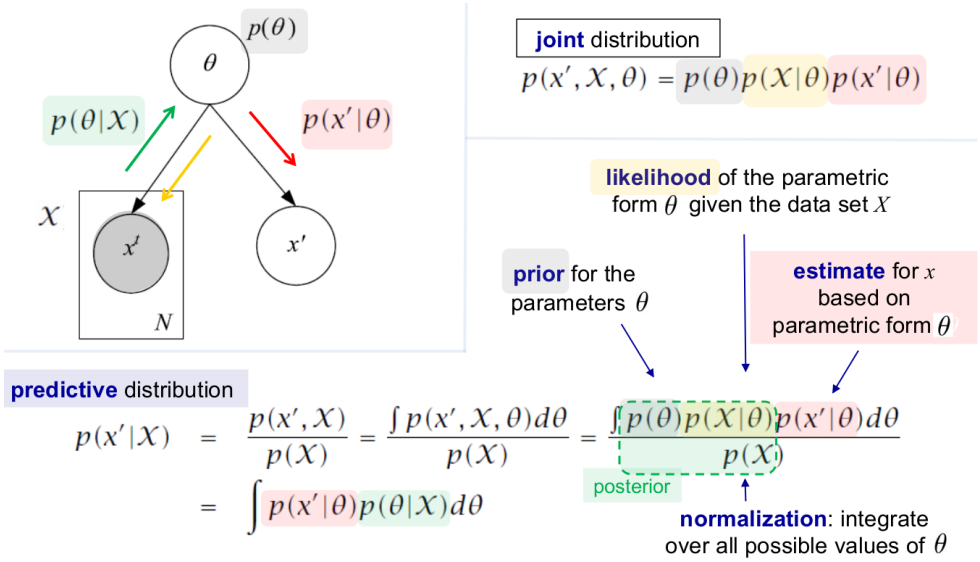


Figure 54: Image from lecture 7.

7.3 A Probabilistic View on Regression

7.3.1 Least-Squares Estimation as Maximum Likelihood

!! **Professor Q.:** *Linear regression: Prove and explain that Least-Squares regression is equivalent to Maximum Likelihood (ML) estimation under the assumption of Gaussian output noise. Derive and formulate the ML solution* w_{ML}.

Solution:

7.3.2 Maximum-A-Priori (MAP) Estimation

!! **Professor Q.:** *Linear regression: Prove and explain that L2-regularized linear regression (ridge regression) is equivalent to Maximum-A-Posteriori (MAP) estimation under the assumption of Gaussian output noise and with a Gaussian prior on the coefficients* w

- *Derive and formulate the MAP solution* w_{MAP}
- *assume that the output noise is Gaussian distributed –* $1/\beta$ *noise variance*
- *introduce a prior Gaussian distribution over the coefficients* w *–* $1/\alpha$ *variance*
- *express the posterior distribution over* w*,*
- *and calculate the MAP solution* w_{MAP}

Is the MAP w_{MAP} *solution a fixed value or a distribution?*

Guide: https://agustinus.kristia.de/techblog/2017/01/05/bayesian-regression/

Solution:

7.3.3 Bayesian Curve Fitting

!! **Professor Q.:** *Bayesian learning: Draw and explain a generative model (as a directed graphical model) in the supervised case, and give the predictive probability* $p(t' \mid x', x, t)$ *for a new output* t'*, given a new input value* x'*, and conditioned on the observed (training) data* x*,* t.

Solution:

!! **Professor Q.:** *Draw and explain the Bayesian curve fitting (Bayesian linear regression) generative model as a directed graphical model.*

- *Focus on understanding the graphical model representation for "learning" and "prediction".*
- *What are the key differences between Maximum Likelihood (ML) estimation and the Bayesian learning approach to estimate model weights vector?*

Solution:

7.4 Summary

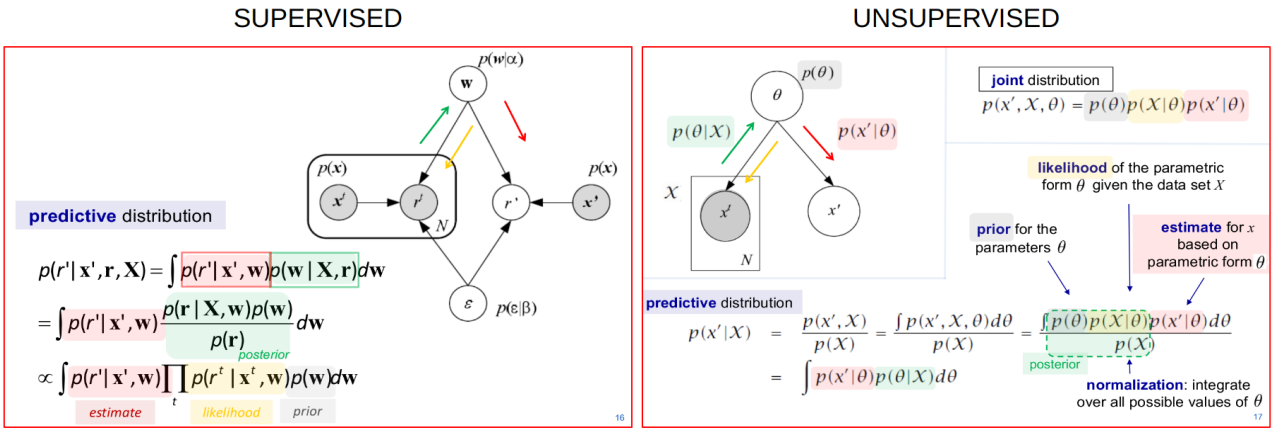


Figure 55: Images from lecture 7.

8 Hidden Markov Models and Gaussian Processes

"Markov chains are sequences of random variables in which the future variable is determined by the present variable but is independent of the way in which the present state arose from its predecessors." This work launched the theory of stochastic processes.

In the following, we will look at sequential problems from a Graphical Models perspective (remember, sequential data is not IID; however, it is possible to treat all observations as independent). The Markov assumption is that each observation only depends on the most recent previous observation (regardless of states in previous times). E.g., the simplest model

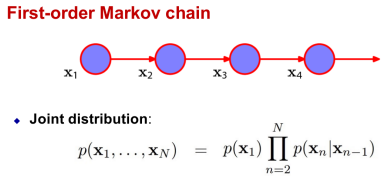


Figure 56: Images from lecture 8.

A second order would connect up to their second neighbor. When having N observations that can connect their first k neighbors, the parameters that the model must learn are $(k-1)k^N$; complexity time grows exponentially. It is possible to model the dependency of all previous observations with limited parameters by introducing a **state-space model**.

8.1 Hidden Markov Models

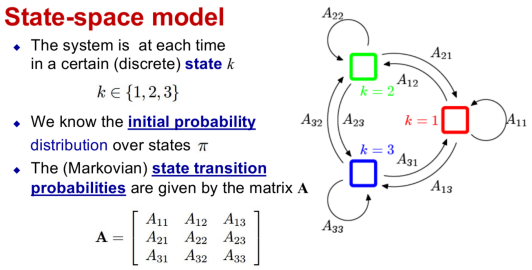


Figure 57: Image from lecture 8.

Notice that one cannot observe the states directly; they are hidden (latent). Furthermore, each state produces a characteristic output, given by an observation probability distribution over outputs φ.

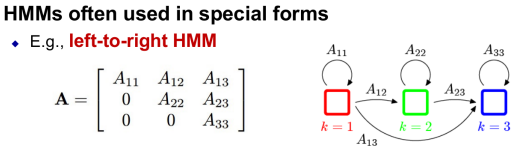


Figure 58: Image from lecture 8.

To be able to encode HMMs as graphical models, one needs to introduce latent variables z for the current system state so that the observed output x is conditioned on this state z.

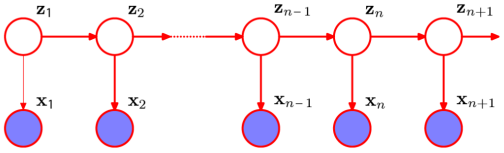


Figure 59: Image from lecture 8.

The (Markovian) state transition probabilities $p(z_n \mid z_{n-1})$ are given by the entries of the state transition matrix A; $p(z_{n,k} = 1 \mid z_{n-1,j} = 1) = A_{ij}$. With this, the number of parameters reduces to $nk(k-1) + const.$

General formulation of a Hidden Markov Model Recommend to watch lecture again

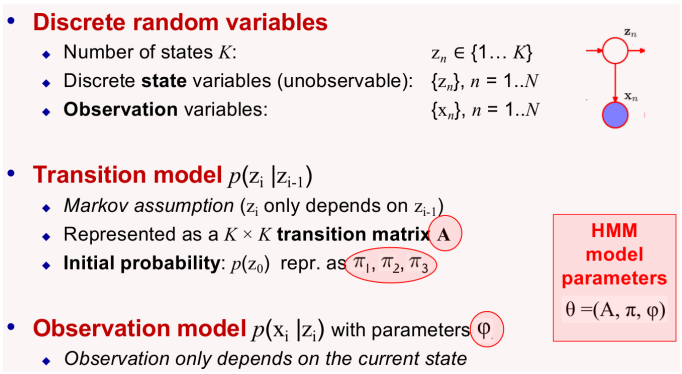


Figure 60: Image from lecture 8.

Hidden Markov Models prove particularly useful for sequential data and considering temporal correlations. Markov models work well if they have short and fixed-time dependencies. Three possible operations can be performed with HMMs:

- **Data likelihood** given a model and an observation (message passing scheme can be used).
- **Most likely state sequence**, just like the former, requires a model and an observation; furthermore, a message-passing scheme can be used.
- **Optimal HMM parameters**, Expectation Maximization is used for training the parameters.

Professor Q.: *Discuss the general formulation of a Hidden Markov Model (HMM) graphical model + max 15 lines. Wrap-up Slide.*

8.2 Gaussian Processes

Introduction
Professor Q.: *Explain differences: Generative vs Discriminative models and Parametric vs Non-parametric models.*

Professor Qs.: *Give the definition of a Gaussian Process (GP). What is needed to fully specify a GP? Give the predictive distribution $p(y_*|x_*, X, y)$ of a GP (in the noise-less and noisy cases). List and describe all parameters or variables. Discuss the computational complexity of a GP (i.e., the computational complexity of the training, the calculation of the prediction mean, and the prediction variance). Describe the key properties (i.e., pros/cons) of a GP (rather brief, max 20 lines, a summary of this chapter).*

Professor Qs.: *Give and discuss the Squared Exponential (SE) kernel. What are the hyperparameters of the Squared Exponential (SE) kernel? How to estimate the hyperparameters of a GP (brief, max 20 lines)? Do we use cross-validation*

to estimate the hyperparameters?

Professor Q.: *Draw and discuss the flowchart of Bayesian Optimization (BO) (no math required, flowchart + max 20 lines). What is surrogate-based optimization (SBO)? What is an infill criterion, a sampling policy? What is an acquisition function? How do we name the loss function of GP- based optimization, or Bayesian Optimization (BO), in the engineering community and in the machine learning community?*

8.3 Summary

Keywords: state-space model,

9 Combining Multiple Learners – Ensembles

Recall that there are two types of approaches: **discriminative** (either explicitly or implicitly, study the posterior distribution directly) or **generative** (models the likelihood and prior separately – Bayesian Networks and MRFs). In this lecture, we will focus on discriminative models.

9.1 Ensembles

Professor Q.: *Explain ensembles: key properties, strengths, and weaknesses.*

Various models could be used to make decisions or predict based on

- **bagging**: the majority voting is taken from all ensembles, and
- **boosting**: there is a weighted opinion on the best algorithms.

Each model in the ensemble is assumed to be independent (uncorrelated errors), with each having an error probability of $p < 0.5$; this assures that a simple majority vote of the ensemble has a lower error than each individual model.

- **Advantages**: the ensemble typically outperforms single models, is easy to implement, and does not require too much parameter tuning.
- **Disadvantages**: there is no compact representation, it can have learning time and memory constraints, and it can behave as a black box as the learned concept could be difficult to understand.

When designing ensembles, there are two main questions to be addressed:

1. How to construct the ensemble such that each has uncorrelated errors?
   - The simplest case is to train the same model on different data (but this can be expensive), or
   - sub-sample the training data and train the same model.

   This works particularly well on **unstable algorithms** (where small differences in training data can produce very different responses, e.g.: decision trees, neural networks, or rule learning algorithms. Some stable algorithms are k-nearest neighbor, linear regression, SVMs, and others.)
2. How to combine the individual hypotheses?

Professor Q.: *Explain: k-fold Cross-Validation.* (A way of constructing an ensemble, not for hyperparameter tunning.)

9.2 Bagging, Stacking, Boosting, and AdaBoost

Professor Qs.: *Explain: stacking, bagging (with algorithm and its schematic representation), and boosting (with algorithm and its schematic representation). Explain the differences between boosting/bagging and boosting/AdaBoost. Explain the advantages/disadvantages of AdaBoost and how AdaBoost can be used for feature selection.*

BAGGING: an answer to the first design question.
Bagging is an acronym for "Bootstrap aggregating" [Breiman 1994, 1996]

- Bootstrapping: randomly select K samples (with replacement) from the full set of N training data points (e.g.: a,b,c,d,e → a,c,c,d,d). WARNING: If K = N, then on average, 63.2% of the training points will be represented; the rest are duplicates. Each model in the ensemble is trained on different sampling.
- Aggregation: majority vote for classification or average for regression.

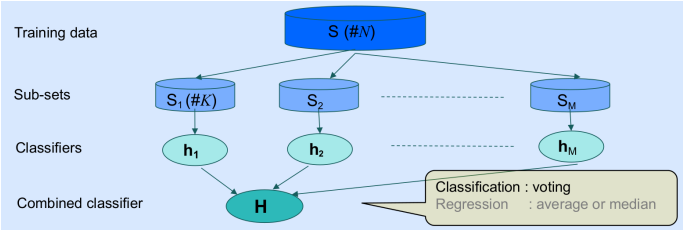


Figure 61: Image from lecture 9.

- Main characteristics: resamples data points. Improves stability and classification accuracy. **Reduces variance** and helps to avoid overfitting.

  One can also inject randomness per sampling with different seeds (e.g. Gaussian noise) to reduce error correlation.

STACKING: an answer to the second design question.
Stacking accounts for taking L learned models (based on the training data) and finding a meta-classifier that inputs the output of the L first-level models as input.

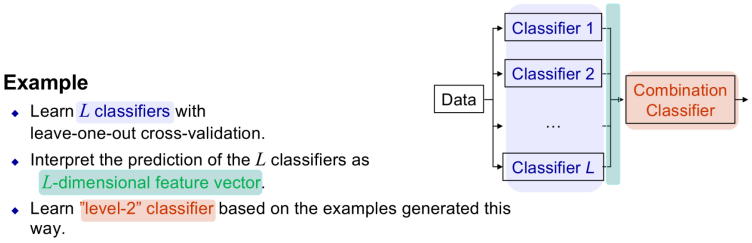


Figure 62: Image from lecture 9.

The idea of this technique lies in "Divide-and-Conquer": Train each of L classifiers on its own input data. Only the combination classifier needs to be trained on combined input. This proves effective for missing data – instead of *cleaning* it, assign it its own classifier. It gives better results than Naïve Bayes (NB) estimator.

BAGGING: an answer to the second design question.
It has been discussed how bagging can be created, and its schematic algorithm has been shown. However, it is possible to show that the average error of a model can be reduced by a factor of M simply by averaging M versions of the model. Nevertheless, this result depends on the assumption that the errors are all uncorrelated – in practice, this is impossible; in fact, the contrary – even with that, it still can be demonstrated that the ensemble error will be lower than a singular model. Bagging is the primary variance-reducing method.

BOOSTING: an answer to the first and second design questions.
The main idea in boosting is to fit each model one at a time with randomly sampled data (without replacement); then, the chosen error metric is checked so that the incoming model will be fitted again on the incorrect training examples. This is a Sequential model selection. In the early iterations, **boosting is primarily a bias-reducing method**, and in later iterations, it appears to be **primarily a variance-reducing method**, resulting in interesting properties. The output can be done with majority voting or average.

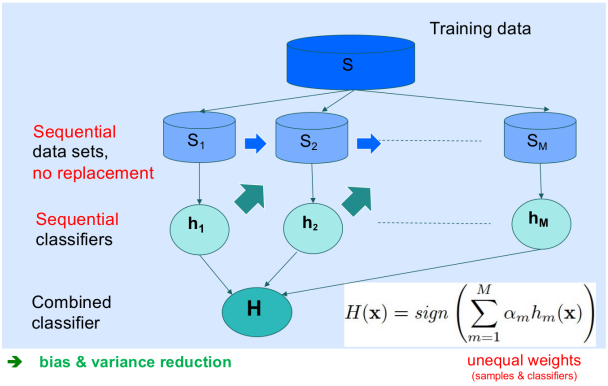


Figure 63: Image from lecture 9.

One interesting property of this algorithm is that it can be used with many different types of classifiers (even weak ones, meaning none need to be too good on its own).

ADABOOST: an answer to the first and second design questions.
AdaBosst is the acronym for "Adaptative Boosting" and, in essence, it has the same main characteristics of boosting; but, instead of increasing the chance of the incorrect output being selected again, it increases the cost when training on the full set.

Basic steps : In each iteration *m*, AdaBoost

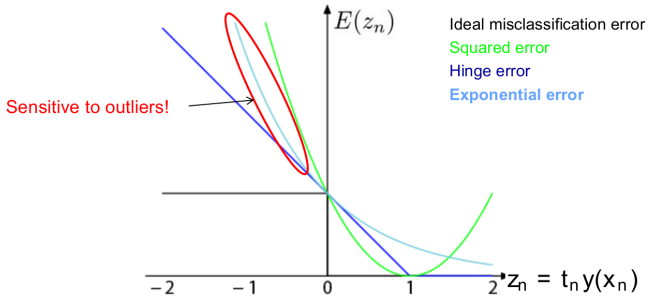
- (1) First, trains a new weak classifier $h_m(\mathbf{x})$ based on the current weighting coefficients $\mathbf{W}^{(m)}$.
- (2) Next, adapts the weighting coefficients w_n for each point
  - Increase w_n if x_n was misclassified by $h_m(\mathbf{x})$.
  - Decrease w_n if x_n was classified correctly by $h_m(\mathbf{x})$.
- (3) Then, makes predictions using the final combined model

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

Figure 64: AdaBoost for classification using accuracy as score. Image from lecture 9.

Professor Q.: *Explain: loss functions*

Historically, it was empirically observed that AdaBoost often tends not to overfit. **AdaBoost minimizes an exponential error function** in a sequential fashion; from this, it is possible to derive that AdaBoost is sensitive to noise, outliers, and mislabeled data. Have a look a t the following image



- **Exponential error** used in AdaBoost
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Disadvantage: exponential penalty for large negative values!
  - ➔ Less robust to outliers or misclassified data points!

Figure 65: Image from lecture 9.

It is possible to make AdaBoost more robust to outliers by switching to the cross-entropy error (found in logistic regression), which, for negative values, it is situated below the Hinge error (hence, being more robust than Hinge used in SVMs). This type of algorithm is known as GentleBoost.

- Advantages in AdaBoost: it is easy to implement, and there are few parameters to tune. The time and space grow linearly with the number of examples. It naturally combines feature selection with learning. It is robust to overfitting.
- Disadvantages in AdaBoost: it is sensitive to (uniform) noise and outliers. It can perform poorly when there is insufficient training data relative to the complexity of the base classifiers. Finally, due to their sequential fitting, this algorithm cannot be parallelized (nor boosting, contrary to bagging).

9.3 Viola-Jones Face Detector

Professor Q.: *Explain key steps in Viola-Jones Face Detector. Explain "attentional cascade" and how boosting can be used for feature selection.*

This algorithm [P. Viola, M. Jones, 2004] characterizes by using

- **Boosting**: for feature selection, and
- **Attentional cascade**: for fast rejection of negative windows.

The tasks of the algorithm are the following:

1. hypothesizes: try all possible rectangle locations and sizes.
2. test: Based on**wavelet-like features**, classify if a rectangle contains a face.

The face detection accounts for feature extraction by considering rectangular filters (similar to Haar wavelets [Papageorgiou e.a.]); instead of pixels, simple non-linear functions sensitive to edges and other critical features of the image are utilized. To find the simple features, AdaBoost is used as follows

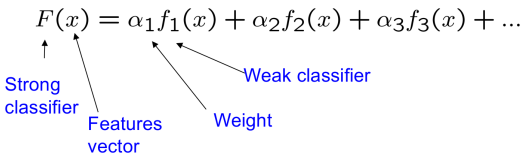


Figure 66: Image from lecture 9.

This analysis is implemented in different rounds by reweighting the examples according to errors, choosing another filter/threshold combo. After AdaBoost, the attentional cascade is implemented for fast rejection of negative windows.

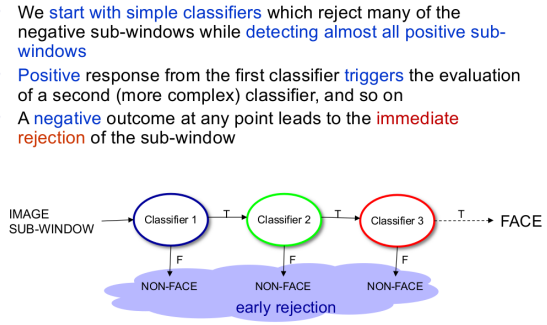


Figure 67: Image from lecture 9.

This algorithm can be used in real-time object detection. REMARK: cascade is just another form of answering the second design question.

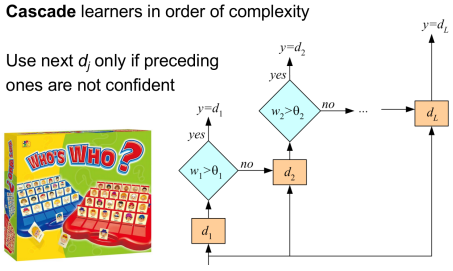


Figure 68: Image from lecture 9.

ChatGPT answer: *Can you explain the Viola-Jones Face Detector algorithm in simple terms, please?*

Of course! The Viola-Jones Face Detector is an algorithm used to detect faces in images. Imagine you're looking at a picture, and you want to find all the faces in it. This algorithm helps computers do that automatically.

Here's how it works in simple terms:

- **Dividing the Image**: First, the algorithm breaks down the image into smaller pieces, like puzzle pieces. These smaller pieces are called "sub-windows." Each sub-window is like a little square area of the image.
- **Haar Features**: The algorithm uses something called "Haar features" to figure out if a sub-window might contain a face. Haar features are like patterns or characteristics that might appear in faces. For example, a Haar feature could be a dark area (like an eye) surrounded by lighter areas (like the forehead and cheek).
- **Calculating Feature Values**: The algorithm checks each sub-window to see if these Haar features match. It calculates a value for each feature in each sub-window. If the value matches a certain pattern (like dark-light-dark), it thinks there might be a face there.
- **Integral Images**: To speed up calculations, the algorithm uses a trick called "integral images." These are special versions of the image that help quickly calculate the sums of pixel values in rectangular areas.
- **AdaBoost Training**: The algorithm has been taught to recognize which Haar features are more important for identifying faces. It uses a technique called "AdaBoost" to combine many weak classifiers (those that use a single Haar feature) into a strong classifier.
- **Cascade of Classifiers**: The algorithm doesn't check every sub-window with all the features right away. Instead, it uses a "cascade" of classifiers. Each stage of the cascade rules out non-face sub-windows quickly, so the algorithm doesn't waste time on them.
- **Adjustable Thresholds**: The algorithm uses thresholds to decide whether a sub-window contains a face or not. These thresholds can be adjusted to control how sensitive or specific the detector is. Higher thresholds make it more specific but might miss some faces, while lower thresholds make it more sensitive but might also give more false positives.
- **Final Detection**: After going through all the stages and checking different sub-windows, the algorithm marks the sub-windows where it thinks there might be faces. These marked areas are the detected faces!

So, in simple terms, the Viola-Jones Face Detector breaks down an image into pieces, looks for certain patterns that resemble face features, and uses smart techniques to quickly decide where faces are likely to be. It's like a digital detective for finding faces in pictures.

9.4 Decision Trees: CART Framework

CART = Classification And Regression Trees [Breiman et al. 1984].

Decision trees (typically univariate) are relatively old algorithms that can be implemented with numerical or categorical data. They are popular due to their interpretable results, which can be written as sets of if-then rules. Important termonology are

- Node: specifies a test for some attribute,
- Branch: corresponds to a possible attribute value.
- Leaf: corresponds to a possible class.

Finding the optimal decision tree is NP-hard.

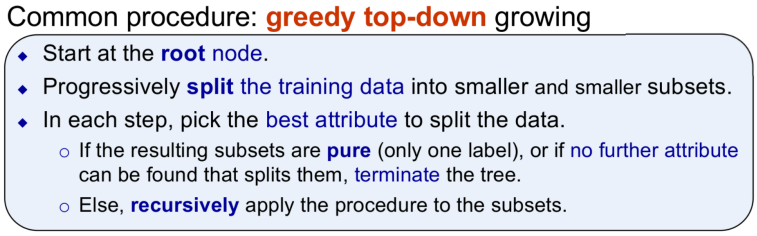


Figure 69: Image from lecture 9.

Six general questions must be approached when working with trees.

1. Binary or multi-valued problem? (i.e. how many splits should there be at each node?)
    - It is common to encounter binary trees as a multi-valued tree can be converted into an equivalent binary tree.

2. **Which property should be tested at a node?** (i.e. how to select the query attribute?)
   - The goal is to have a tree that is as simple/small as possible (this ideology on simplicity is known as Occam's razor).

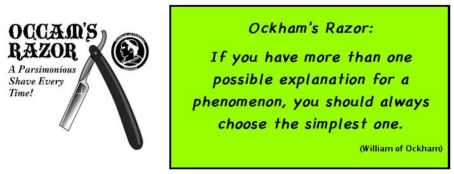


Figure 70: Image from lecture 9.

However, finding a minimal tree is an NP-hard optimization problem. Despite that the greedy top-down search is efficient, it does not always converge on the smallest tree. Here is where the concept of **impurity** $i(node)$ comes into hand. There are three different ways to compute the impurity of a node N:

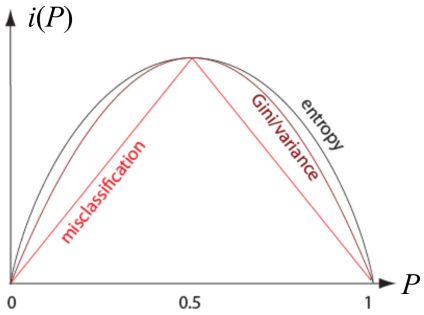


Figure 71: Image from lecture 9.

Write de expressions of the Gini and entropy impurity.

The goal is to select node-query that decreases impurity the most. One can impose this idea until the tree is fully grown; but, this will always lead to overfitting. There are two ways to prevent this:

- Prepruning: stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
- Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence (this is often best strategy).

The most expensive part is the training algorithm, with $\mathcal{O}(DN^2 \log N)$, where D is the number of dimensions (features) in data. Trees also have stability issues since they can be very sensitive to the details of the training points.

3. When should a node be declared a leaf? (i.e. when to stop growing the tree?)
4. How can a grown tree be simplified or pruned? (Goal: reduce overfitting.)
5. How to deal with impure nodes? (i.e. when the data itself is ambiguous.)
6. How should missing attributes be handled?

Professor Q.: *What is the CART framework (only high-level description: goal + key steps + max 10 lines)? Explain Occam's razor (max 5 lines). Explain Entropy impurity (high-level description: goal = pick good splitting features + max 5 lines).*

Professor Q.: *Explain: Decision Trees: Properties/Limitations*

9.5 Random Forests

The idea is to create an ensemble of many very simple and fully grown binary trees, so there is little effort to find the best split. Trees are fitted with Gini impurity. Each split is only made based on a random subset of the available attributes (features). Empirically, they have shown very good results (often as good as SVMs and Boosting). **The secret is in the injection of the right kind of randomness**.

Algorithmic Goals

1. Create many fully-grown trees (50 – 1000).
2. Inject randomness into trees such that each tree has maximal strength and has a minimum correlation with the other trees.
3. Create the ensemble and implement majority vote/ averaging.

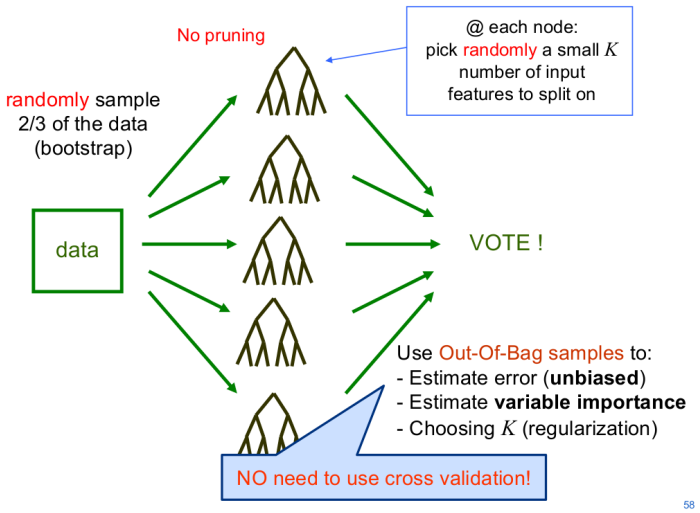


Figure 72: Image from lecture 9.

The *right injection oh randomness* happens at the bootstrap sampling (with replacement) process from all N available training examples. On average, each tree is grown on only ∼63% of the original training data. The remaining 37% **"out-of-bag" (OOB) data** is used as the validation set. In this manner **the error estimate is unbiased and behaves as if we had an independent test sample of the same size as the training sample.**

WARNING: sampling is not the only way to inject randomness; feature selection is another way!

For each node, randomly choose a subset of k attributes on which the split is based, typically $k = \sqrt{total\ num\ features}$. This approach leads to a faster training procedure and minimizes inter-tree dependence. The overall approach is more robust than AdaBoost. Since each tree is fully grown, this method can be comparable to the k-nearest-neighbor predictor.

Random forests are outstanding algorithms because they are simple to implement and robust. Furthermore, they are fast in training and can be easily parallelized. They can yield estimates of feature importance in classification through the impurity concept. However, one downside is that they consume a lot of memory and require large training data.

Professor Qs.: *Random Forests – describe key features. How can RFs possibly ever work? Is cross-validation used in RFs? Explain OOB, and RFs Properties/Limitation.*

MORE (professor's) QUESTIONS:

1. Explain differences:
   - Generative vs Discriminative models.
   - Parametric vs Non-parametric models.
   - Supervised vs Unsupervised learning.
   - Stable vs Unstable learners.
   - Bias vs Variance.
   - Bagging vs Boosting.
   - Boosting vs AdaBoost.

9.6 Summary

10 Neural Networks and Feature Learning

Instead of slide by slide, I will answer the following questions regarding Neural Networks.

Professor Qs.: *Explain basic principles. Give the conceptual drawing. Mathematical expression for the model equation. Which type of model is it? Which hyperparameters can you tune to decrease bias and/or variance? Any other important hyperparameters? Principles of training approach(es). Computational complexity of training and inference, scaling with amount of data. Memory requirements for inference (+ scaling behavior). Specific properties, e.g. sensitivities to feature properties and number of features.*

A neural network is composed of nodes (also called neurons) that are connected to other neurons creating layers; biological neurons inspired their design. The following image is an example of a fully connected neural network.

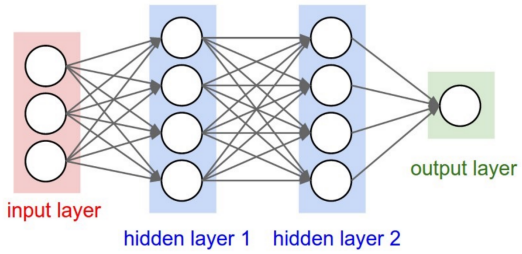


Figure 73: How many hyperparameters are there to tune in a NN? Image from lecture 10.

Their structured form allows parallelization. Perceptrons were the earliest form of artificial neural networks, which basically consist in a linear model + output non-linearity; it can be represented as shown below

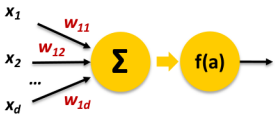


Figure 74: Image from lecture 10.

Then, it was found that multilayer perceptions (MLP) with one hidden layer are a universal approximator for classification. MLP are powerful tools that allows to capture non-linear properties in data, but the more layers they have, the more weights there are to train; besides, they are not differentiable with hard thresholds. The following image shows a MLP with soft thresholds (with a differentiable activation function, e.g., sigmoid or tanh).

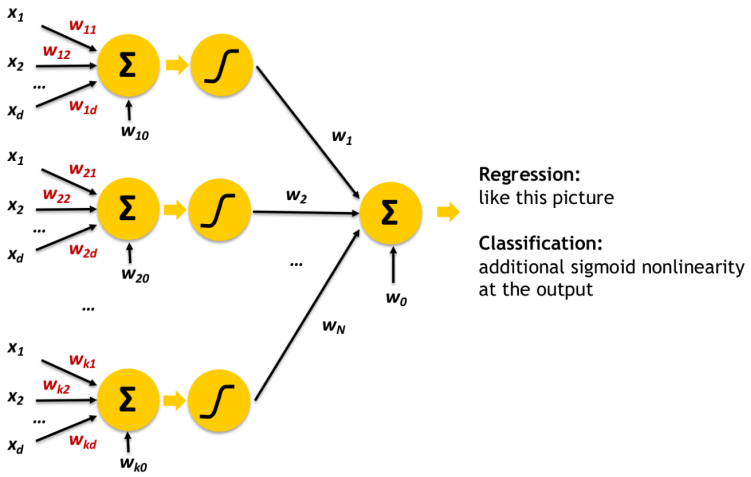


Figure 75: Image from lecture 10.

MLP are the foundations of more complex neural networks.

ChatGPT answers mixed with lecture:

Forward Propagation:

1. Think of it like: Looking at a picture and trying to guess if it's a cat or a dog.
2. What happens: The picture's pixels are the input, like the information your eyes send to your brain. Each neuron (like a mini-brain cell) in the network looks at the input and decides whether it looks more like a cat or a dog.
3. How it works: The input pixels are multiplied by weights (like giving more importance to certain features) and added up. Then, this sum goes through a "decision-making" function (activation function) to determine if it's more like a cat or a dog.
4. Outcome: At the end of the process, the network gives you its guess: "It's probably a cat!" or "It's likely a dog!"

- **Input and Weights**: Let's say you have an input data vector x and weights w for each connection between neurons in the network. Each neuron's input is the weighted sum of the inputs from the previous layer, plus a bias term b:

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b$$

- **Activation Function**: The weighted sum z is then passed through an activation function $f(z)$, which introduces non-linearity to the network. Common activation functions include sigmoid, ReLU, and tanh:

$$a = f(z)$$

- **Propagation**: This a becomes the output of the current neuron and is used as input for the next layer's neurons. This process of feeding inputs forward through the network layer by layer is called forward propagation.

Backpropagation:

1. Think of it like: Learning from your mistakes to improve your guesses.
2. What happens: After making a guess, you find out if it was right or wrong (by checking with the actual answer). If the guess was wrong, you want to figure out which neurons were responsible for the mistake.
3. How it works: The difference between the guess and the actual answer is used to adjust the weights of the neurons. Neurons that contributed to the wrong guess get adjusted more, so they'll do better next time.
4. Outcome: This process is like teaching your brain which features in the picture matter most for telling cats from dogs. Over time, the network gets better at making accurate guesses.

- **Calculating Error (Loss)**: After forward propagation, you compare the network's output a with the actual target value y to compute the error (also known as loss) using a loss function $L(a, y)$.
- **Gradient Descent**: Backpropagation aims to adjust the weights w and biases b to minimize the error. To do this, you calculate the gradient of the loss with respect to the weights and biases. This gradient tells you how much changing each weight and bias would affect the error.

- **Error Backpropagation**: You then propagate this error gradient backward through the network. This involves applying the chain rule of calculus to compute how much each neuron contributed to the error in the final prediction.
- **Weight Update**: Once you have the gradient, you update the weights and biases in the opposite direction of the gradient to reduce the error. This step is typically performed using optimization algorithms like stochastic gradient descent (SGD) or its variants.
- **Iterative Process**: You repeat the forward propagation and backpropagation process for multiple rounds (epochs), adjusting the weights and biases at each step to gradually minimize the error and improve the network's performance.

In summary, forward propagation involves calculating the output of the network given inputs and weights, while backpropagation calculates how much each weight and bias contributed to the error and updates them accordingly. Through many iterations of this process, the network learns to make better predictions and improve its performance on the given task.

One can avoid overfitting by giving more data (either getting more IID, augmenting it or adding noise). However, there is a neural network-specific technique known as **dropout**. Dropout **randomly "drops out" (deactivates) a certain percentage of neurons** in a layer. It's like temporarily removing some brain cells from the network. By doing this, the network is forced to not rely too heavily on any particular neuron because it never knows which ones will be turned off. This prevents certain neurons from becoming overly specialized and makes the network more robust. Final network is like an ensemble of a large number of pruned networks.

Neural networks can be used as autoencoders and embeddings.

Deep Neural Networks:

Deep neural networks (DNNs) are a type of artificial neural network that consist of multiple layers of interconnected neurons. The term "deep" in this context refers to the presence of multiple hidden layers between the input and output layers. These layers allow DNNs to learn and represent complex patterns and relationships in data, making them well-suited for tasks like image recognition, natural language processing, and more.

Each layer of a deep neural network consists of multiple neurons, and connections between neurons in adjacent layers have associated weights. DNNs use forward propagation to process input data through the layers, transforming it into a final output. The introduction of non-linear activation functions in each neuron's processing helps the network capture and learn complex relationships in the data.

Vanishing Gradients in Deep Networks: Vanishing gradients is a problem that can occur when training deep neural networks with multiple layers, especially those using certain activation functions like the sigmoid or hyperbolic tangent (tanh). The term "gradient" refers to the rate at which the error changes with respect to the weights and biases in the network during training. Gradients are used to update these parameters and improve the network's performance.

In deep networks, during backpropagation (the process of calculating gradients to update the network's parameters), the gradients can become very small as they are propagated backward from the output layer to the earlier layers. When the gradients become extremely small, the weights and biases of the earlier layers receive very tiny updates, causing these layers to learn very slowly or not at all.

This phenomenon is called "vanishing gradients" because the gradients "vanish" or become negligible as they propagate backward through the layers. As a result, the early layers of the network may not learn meaningful representations of the data, and the network's overall performance can suffer. Deep networks with many layers are particularly vulnerable to this issue.

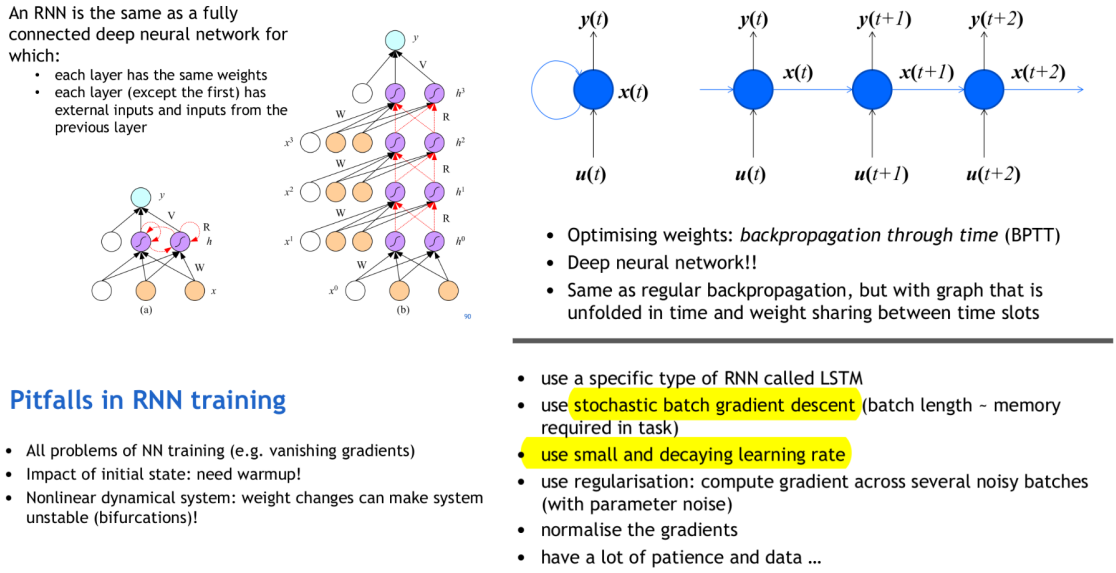
One way to mitigate vanishing gradients is to use activation functions that do not saturate as quickly as sigmoid or tanh, such as Rectified Linear Units (ReLU). Additionally, techniques like batch normalization and skip connections (used in architectures like **ResNet**) help address vanishing gradients by allowing smoother gradient flow during training.

In summary, deep neural networks are networks with multiple hidden layers that can capture complex patterns in data. Vanishing gradients refer to the problem of gradients becoming too small during backpropagation, which can slow down learning and hinder the training of deep networks.

Exploding gradients can also occur in Deep Neural Networks (DNNs), especially when the network architecture is deep and the gradients during training become too large. Exploding gradients can lead to numerical instability during training and cause the network's parameters (weights and biases) to be updated excessively, making the training process diverge or become very slow. The phenomenon of exploding gradients is more commonly associated with RNNs due to their sequential nature and the potential for information to be multiplied over multiple time steps. However, it can still happen in DNNs with many layers, particularly when the network's activation functions, weight initialization, and learning rate are not properly managed.When the gradients during backpropagation are large, they can cause the weights to increase significantly in each iteration, pushing the network's parameters towards very high values.

Recurrent Neural Networks (RNNs) are a type of neural network architecture designed to work with sequential data, where the order of data points matters. RNNs have loops in their architecture that allow information to be carried

from one step to the next, making them particularly useful for tasks like language modeling, time series prediction, machine translation, and more.



Pitfalls in RNN training

- All problems of NN training (e.g. vanishing gradients)
- Impact of initial state: need warmup!
- Nonlinear dynamical system: weight changes can make system unstable (bifurcations)!

- use a specific type of RNN called LSTM
- use stochastic batch gradient descent (batch length ~ memory required in task)
- use small and decaying learning rate
- use regularisation: compute gradient across several noisy batches (with parameter noise)
- normalise the gradients
- have a lot of patience and data …

Figure 76: RNN info. Images from lecture 10.

Weight sharing is a technique used in neural networks to reduce the number of parameters that need to be tuned, especially in cases where there is limited data available or when the network architecture is constrained by computational resources. Weight sharing involves using the same set of weights for multiple units within a neural network layer. This approach can be particularly useful in situations where certain patterns or features are expected to be similar or shared across different parts of the input.

Here's how weight sharing works and how it reduces the number of hyperparameters:

- **Shared Weights**: In weight sharing, instead of assigning unique weights to each connection, a single set of weights is shared across multiple connections or units in a layer.
- **Pattern Recognition**: Weight sharing is often used when the same feature or pattern should be detected in different parts of the input. For instance, in image processing tasks, weight sharing can be applied to detect similar features like edges, corners, or textures across different regions of the image.
- **Reduced Parameters**: By sharing weights, the number of learnable parameters in the network is significantly reduced. This reduction can lead to several benefits: - **Reduced Overfitting**: With fewer parameters, the network is less likely to overfit the training data, especially when the dataset is small. - **Faster Training**: Fewer parameters mean fewer computations, leading to faster training times. - **Improved Generalization**: Weight sharing encourages the network to learn more general and shared features, which can lead to better generalization to unseen data.
- **Convolutional Neural Networks (CNNs)**: Weight sharing is a fundamental concept in CNNs, a type of neural network specifically designed for image analysis. In CNNs, the convolutional layers use shared weights (kernels) to scan across the input image, detecting features irrespective of their location.
- **Siamese Networks**: Another application of weight sharing is in Siamese networks, which are used for tasks like image similarity and verification. In Siamese networks, two or more identical subnetworks share weights, enabling the network to learn and compare patterns in a more efficient manner.

Weight sharing is a powerful technique that reduces the complexity of neural networks and improves their ability to generalize. By leveraging the shared information across different parts of the input, weight sharing allows networks to learn common features more effectively, making them more suitable for tasks with limited data or specific spatial relationships.

Convolutional Neural Networks (CNNs) are a special type of neural network designed for tasks that involve images or grid-like data, like pictures, videos, and even some types of text data. They're great at automatically finding patterns and features in these kinds of data.

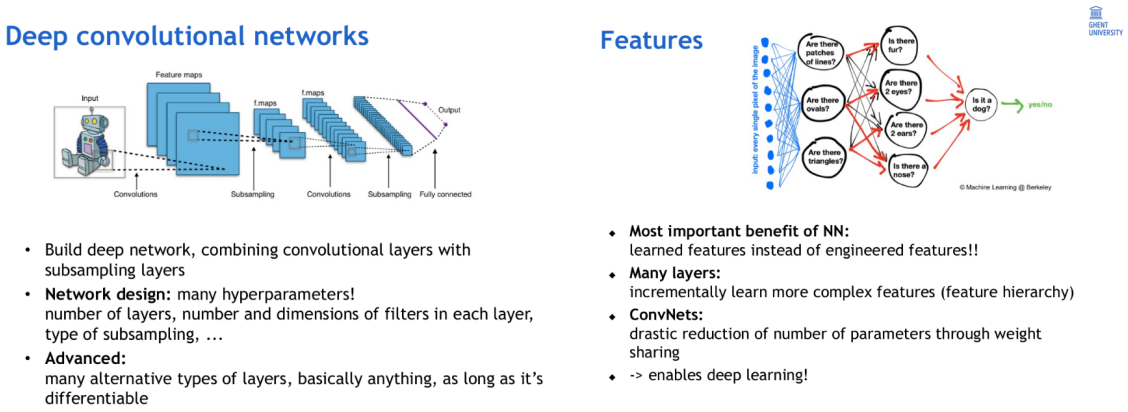


Figure 77: CNN info. Images from lecture 10.

The computational complexity of a fully connected neural network, also known as a multi-layer perceptron (MLP), depends on several factors including the number of layers, the number of neurons in each layer, and the batch size used during training or inference. The most significant contributors to computational complexity are the number of parameters (weights and biases) and the number of operations required for forward and backward passes.

Here's a rough breakdown of the computational complexity of a fully connected neural network:

- **Number of Parameters**: The total number of parameters in a fully connected neural network is determined by the number of neurons in each layer and the connections between layers. For example, if you have n input neurons, m hidden neurons in each layer, and p output neurons, the number of parameters can be approximated as $n \times m + m \times m + m \times p$.
- **Forward Pass**: In the forward pass, the input data is multiplied by the weights and biases for each layer, and then the activation function is applied. The computational complexity of the forward pass depends on the number of neurons, layers, and the batch size. If we consider L layers, N neurons per layer, and B batch size, the complexity of a single forward pass is roughly $O(L \times N \times B)$.
- **Backward Pass (Backpropagation)**: In the backward pass, gradients are calculated with respect to the weights and biases for each layer. The complexity is similar to the forward pass, so it's also roughly $O(L \times N \times B)$.
- **Training**: Training a neural network involves multiple forward and backward passes for each batch of data, and it usually requires multiple epochs. The total training complexity is determined by the number of epochs, batches, and the number of forward and backward passes in each batch.

It's important to note that the above complexities are rough approximations and can vary based on factors such as the specific operations used in calculations, the optimization algorithms, and the hardware (CPU, GPU) being used.

As neural networks become deeper and larger, the computational complexity can become quite high. This is one of the reasons why more efficient architectures, like convolutional neural networks (CNNs) and techniques like model pruning and quantization, have been developed to reduce the computational demands of neural network training and inference.

10.1 Summary

11 Ethical Aspects in Machine Learning

Requirements for ethical AI:

1. Willingness to act ethically
2. Awareness of data-related risks
3. Awareness of model-related risks
4. Awareness of existing analysis and mitigation techniques (and their limitations)
5. Correct and nuanced communication towards non-technical audiences

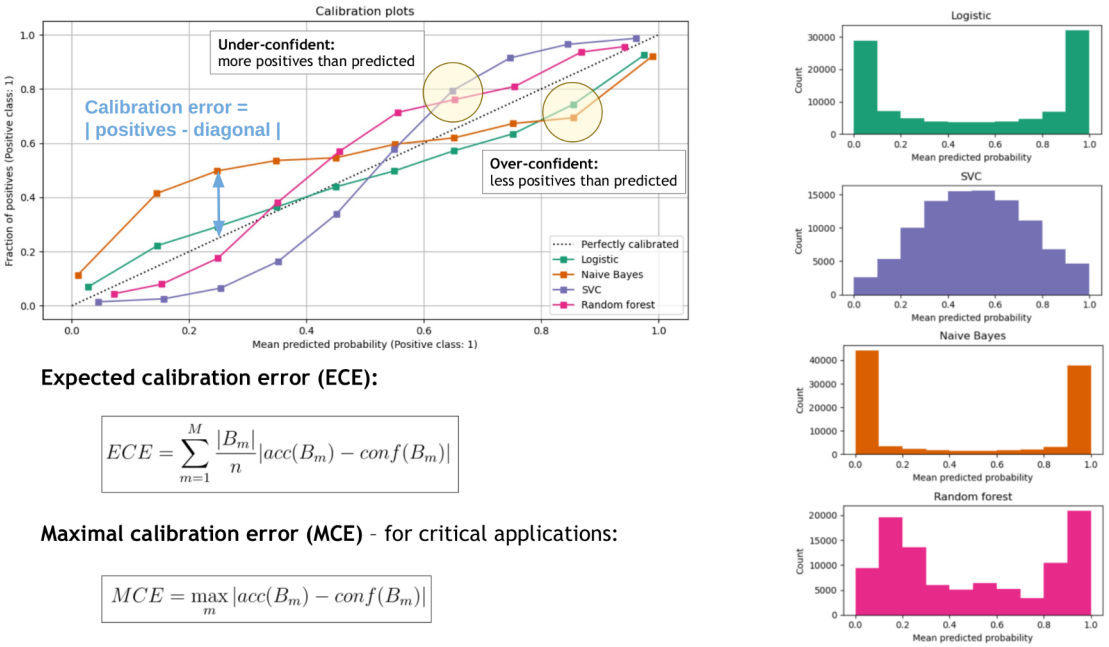
Probabilistic predictions = any approach tany approach that allows your model to assign a probability densityhat allows your model to assign a probability density. Examples:

- Classification with cross-entropy error
- Regression with Squared error loss

11.1 Overconfidence and Unreliability of Models

Professor Q.: *Model calibration: what are calibrated probabilities? Sketch calibration curves for an over-/underconfident model. Why are powerful models more likely to be over-confident?* ***Details of specific calibration techniques are not necessary!***

The concept is a bit harder in regression, but for classification, it is easier. Think about: what is the assumption that a model output means? We assume that the output is really close to the real value. In other words, the output probability given by the model is always assumed to reflect the likelihood of the true events. **If the model output for all classes is close to the likelihood of their true events, then the model output is said to be calibrated.** One can check the calibration of a model with calibration curves (normal in binary classification).



Expected calibration error (ECE):

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)|$$

Maximal calibration error (MCE) - for critical applications:

$$MCE = \max_m |acc(B_m) - conf(B_m)|$$

Figure 78: Images from lecture 11.

More complex models tend to be over-confident because they do not learn to be unsure when they are unsure. This happens due to their almost perfect performance with the training dataset; they have been barely penalized for mistakes

because powerful models are complex enough to learn well from training data; this results in typical poorly calibrated predictions. The models cannot say "This is something I have never seen before, I don't know". Have a look at the following image:

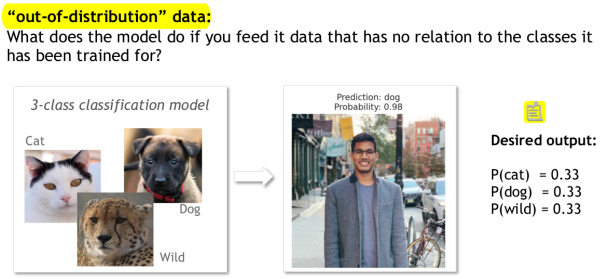


Figure 79: You want to make your model be able to say "I have never seen before this, so I will assign equal probability to all existing classes". Image from lecture 11.

ChatGPT answer:

Calibration plots, also known as reliability diagrams or calibration curves, are graphical representations that help assess the calibration performance of a classification model's predicted probabilities. These plots are used to visualize how well the predicted probabilities from a model align with the actual observed frequencies of positive outcomes.

In a perfectly calibrated model, the predicted probabilities should match the true probabilities across different ranges of predicted probabilities. This means that if the model predicts a probability of 0.7 for a certain class, that class should actually occur about 70% of the time among instances with similar predicted probabilities.

Calibration plots are particularly useful for evaluating the reliability of predicted probabilities when making binary classification decisions. Here's how a calibration plot is typically constructed:

1. **Data Binning**: The range of predicted probabilities is divided into multiple bins or intervals. For example, you might have bins like [0.0-0.1), [0.1-0.2), ..., [0.9-1.0], where each bin corresponds to a range of predicted probabilities.
2. **Average Predicted Probability**: Within each bin, the average predicted probability is computed. This is calculated by taking the average of the predicted probabilities of instances falling into that bin.
3. **Observed Frequency**: The actual observed frequency of positive outcomes (e.g., instances where the true class is positive) is calculated for each bin.
4. **Plotting**: The average predicted probabilities are plotted on the x-axis, while the observed frequencies (or proportions) are plotted on the y-axis.

In a well-calibrated model, the plotted points should closely follow the diagonal line (y = x). Deviations from the diagonal indicate calibration issues. For instance, if the points consistently lie above the diagonal, it indicates that the model's predicted probabilities are overconfident and tend to be higher than the actual probabilities. Conversely, if the points consistently lie below the diagonal, the model's predicted probabilities are underconfident and tend to be lower than the actual probabilities.

Calibration plots provide a visual way to assess the calibration of a model's predicted probabilities and help identify areas where the model might need further improvement. They can be used to decide whether post-processing techniques like Platt scaling or isotonic regression are necessary to better calibrate the model's output probabilities.

In summary, calibration plots are an important tool for evaluating the reliability of classification models' probability estimates and ensuring that these estimates accurately reflect the underlying true probabilities.

Calibrated probabilities in machine learning refer to the process of adjusting the raw output probabilities of a classification model to provide more accurate estimates of the true likelihood of the predicted classes. In many classification algorithms, especially those based on complex models like support vector machines, random forests, or neural networks, the raw predicted probabilities might not accurately represent the actual probability that a particular instance belongs to a certain class.

Calibration is important because it enables the model's predicted probabilities to reflect the true underlying probability distribution of the data. Calibrated probabilities are particularly useful in scenarios where not only predictions are important, but also the confidence or probability estimates associated with those predictions. This is often the case in applications like medical diagnosis, fraud detection, and risk assessment.

Two common calibration methods are:

Platt Scaling (Sigmoid Calibration)

This method involves training a separate calibration model, often a logistic regression, on top of the outputs of the original classification model. The purpose of this calibration model is to map the raw model outputs to calibrated probabilities. This is typically done by fitting the logistic regression model on the raw scores and using it to transform the scores into calibrated probabilities.

Isotonic Regression

Instead of fitting a logistic regression, isotonic regression fits a piecewise non-decreasing function to the predicted probabilities. This function ensures that higher predicted probabilities are associated with higher true probabilities. This method is particularly useful when the relationship between raw scores and true probabilities is not linear.

Both of these methods aim to adjust the raw predicted probabilities so that they better reflect the actual likelihood of the classes. Calibrated probabilities can lead to more reliable and informative probability estimates, which can be especially useful in scenarios where making confident decisions based on those probabilities is crucial.

Professor Q.: *Reject: what is meant by a model with a "reject option"? Why is that useful? How is it different from using a "garbage" (or 'unknown') class?*

Risk = expected value of some error function. For instance, the accuracy/misclassification rate has the same weight in every mistake. In this case, the risk is 1 − probability of choosing the other class – the decision threshold is in the middle. The concept of risk becomes important when a false positive or a false negative becomes more severe; hence, there must be different weights in error cost for the algorithm depending on the situation.

The reject option comes in hand when the class with highest probability is lower than a certain threshold, then it is possible to *reject* the observation = "I am not sure enough to make a decision". The implementation of the reject option does not require re-fitting the model, it can be directly implemented. For the following image, write the missing equations:

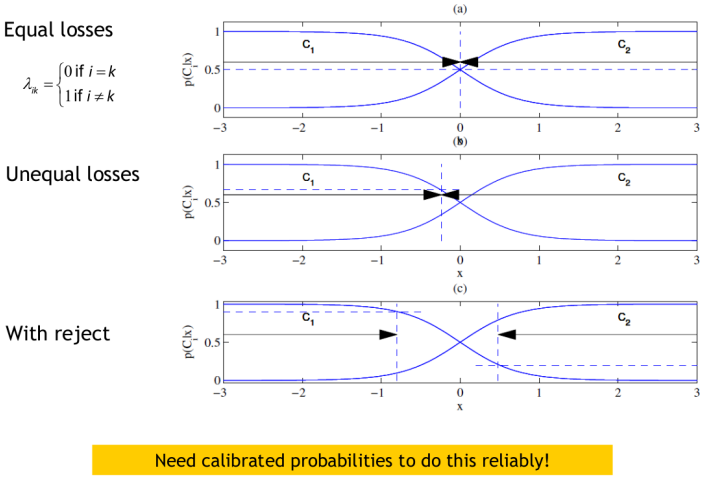


Figure 80: Image from lecture 11.

11.2 Model Explainability

Professor Q.: *Explainability: principles of permutation importance and partial dependence plots. How are they different?*

Some tools try to quantify how (much) impact features had on the model's decision; this can be interpreted as an attempt at explainability.

Explainability in machine learning refers to the ability to understand and interpret the decisions and predictions made by a machine learning model in a human-understandable and transparent manner. It involves providing insights into why a model arrived at a particular decision, what features or factors influenced that decision, and how those factors were weighted. Explainability is crucial for several reasons:

1. **Transparency and Accountability:** In many applications, especially those with high stakes such as healthcare, finance, and legal systems, it's important to understand why a model made a certain decision. This transparency helps ensure accountability and allows users to challenge or question decisions if they appear incorrect or biased.
2. **Trust:** Machine learning models can be complex and treated as black boxes, making it difficult for users to trust their predictions. If the inner workings of the model are transparent and interpretable, users are more likely to trust the model's outputs.

3. **Insights:** Explainability provides insights into the relationships between input features and predictions. This can lead to a better understanding of the underlying data and domain, helping to uncover hidden patterns, correlations, and outliers.
4. **Bias and Fairness:** Explainability helps identify biases and potential discrimination in model predictions. By understanding how the model uses different features to make decisions, it becomes possible to address and mitigate biases.
5. **Regulatory Compliance:** Some industries and jurisdictions require machine learning models to be explainable to meet regulatory or ethical standards.

There are various levels of explainability, ranging from simple methods like feature importance scores to more complex techniques that provide detailed insights into individual predictions. Some common approaches to achieving explainability include:

1. **Feature Importance:** Determining the importance of input features in influencing model predictions. Methods like permutation importance and feature contribution scores fall under this category.
2. **Local Explanations:** Explaining individual predictions by identifying which features contributed to a specific outcome for a particular instance. Techniques like LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) provide local explanations.
3. **Global Explanations:** Providing insights into the overall behavior of the model across the entire dataset. This might involve visualizations, summary statistics, or aggregated feature importance.
4. **Simpler Models:** Using simpler, interpretable models like decision trees or linear regression to approximate the behavior of complex models. These simpler models can serve as proxies for understanding the complex model's decisions.

It's important to note that there's often a trade-off between model complexity and explainability. As models become more complex (e.g., deep neural networks), they might sacrifice some level of interpretability. Striking the right balance between model performance and explainability depends on the specific application and regulatory requirements.

Permutation Importance
A simple tool to evaluate individual importance of each feature:

1. evaluate the model on unseen data
2. (with unseen data): randomly permute values of one feature and
3. re-evaluate (without re-train) and report the change in loss function for each feature
4. features with the largest performance loss were the most important

Benefits:

- model-agnostic: can be used for all models without constraints
- simple to implement, easy to interpret

Drawbacks:

- very 'brute-force', no insight in feature-interactions
- misleading results for strongly correlated features!
- gives global results for the model, **not for an individual instance** (it does not tell you anything about why **a specific sample** was classified like that.)

Partial Dependence Plots
This is the preferred option for model explainability as it allows for specific individual instance analyses; however, it is more complicated than the previous method. This technique is characterized to allow the investigation of individual features with specific samples.

1. sweep 1 feature of a sample while keeping all others equal; namely, swipe x_{iF} across a wide range for the specific F feature
2. make a plot to show the impact of decision output of the overall model
3. can extend to 2D-plots to combine two features (investigate the interaction between features)
4. can average across multiple samples to look for trends

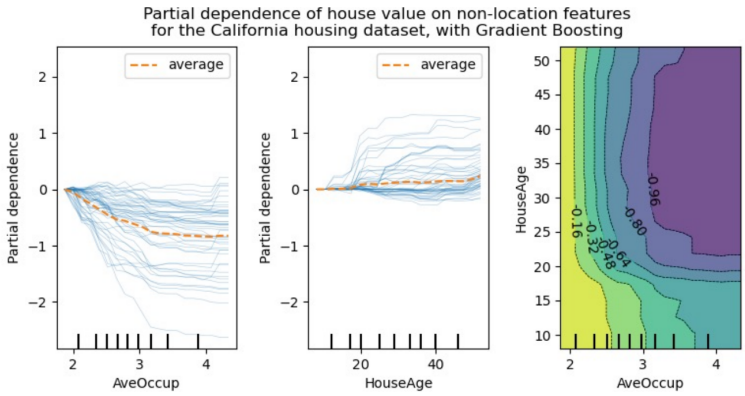


Figure 81: Each blue line is an observation, orange dashed line is the average of all. Image from lecture 11.

SHAP-values
This is another, more complicated, technique that allows analyses for individual samples. It is based on Game Theory and is complicated to further understand.

11.3 FAIRNESS: Criteria, Mitigation & "Fairness by Awareness"

Professor Q.: *Fairness: why is complete fairness generally not possible in machine learning? What would be required to make it possible?*

Fairness: "Absence of any prejudice or favoritism toward an individual or group based on their inherent or acquired characteristics".

In machine learning, fairness is quantified depending on the problem at hand. Accuracy or misclassification rate sometimes do not reflect the actual impact on the algorithm's fairness. Sometimes, the **false negative rate** is more suitable in quantifying fairness at a specific application; e.g., consider a recommendation algorithm, it can tell you how many times is a specific group more likely not to be recommended a certain product compared to the other group. The **false positive rate** would allow the user to know how much a group would be undesirably exposed to recommendations compared to the other group. The ideal scenario is that false positive and false negative rates have the same numbers for all classes; however, this is almost impossible.

The question is: Can we achieve perfect fairness for the false positive rate, precision, and true negative rate (recall)?

Consider the confusion matrix:

Category 1	Negative $y = -1$	Positive $y = 1$
Predicted negative $(\widehat{y}(\mathbf{x}) = -1)$	$n_1 - f_1$	$p_1 - t_1$
Predicted positive $(\widehat{y}(\mathbf{x}) = 1)$	f_1	t_1
Category 2	Negative $y = -1$	Positive $y = 1$
Predicted negative $(\widehat{y}(\mathbf{x}) = -1)$	$n_2 - f_2$	$p_2 - t_2$
Predicted positive $(\widehat{y}(\mathbf{x}) = 1)$	f_2	t_2

Figure 82: Image from lecture 11.

We have:

- **Parameters given by the real world**: n_i, individuals in negative class; p_i individuals in positive class.
- **Parameters given by the algorithm**: f_i, false positive, and t_i true positive.

1. Make sure that your model has the same rate of false positives in both categories:
$$\frac{f_1}{n_1} = \frac{f_2}{n_2} \rightarrow f_1 = \frac{n_1 f_2}{n_2} \tag{40}$$
2. Make sure that your model has the same true positive rate (recall):
$$\frac{t_1}{n_1} = \frac{t_2}{n_2} \rightarrow t_1 = \frac{n_1 t_2}{n_2} \tag{41}$$
Of course, this is in practice super hard, but still "achievable".
3. Make sure that your model has equal precision for both categories:
$$\frac{t_1}{t_1 + f_1} = \frac{t_2}{t_2 + f_2} \tag{42}$$

4. Substitute t_1 and f_1 with previous equations to obtain

$$\frac{t_2}{t_2 + \frac{p_2 n_1 f_2}{p_1 n_2}} = \frac{t_2}{t_2 + f_2} \tag{43}$$

Which **only holds true iff**

$$f_1 = f_2 = 0 \quad \text{or} \quad \frac{p_1}{n_1} = \frac{p_2}{n_2} \tag{44}$$

This means that either the model is **perfect**, or the two categories come from the same underlying distribution; namely, both groups have identical statistics, which we are out of control for modelers.

To improve fairness, one can modify the predictions of the model – e.g. use different thresholds for privileged/unprivileged groups, change the loss function, or try to give more representation to the lower class (in other words, get more data for that class).

11.4 Summary

References

[1] J. Fumo, "Types of machine learning algorithms you should know," *Medium*, Jun. 2017, [Accessed: Jul. 16, 2023]. [Online]. Available: *https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861*.

[2] "Metacademy." [Accessed: Jul. 16, 2023], Metacademy.org. (2023), [Online]. Available: *https://metacademy.org/graphs/concepts/generative_vs_discriminative*.

[3] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. [Online]. Available: *https://smlbook.org*.

[4] S. Vignesh, "Parametric vs non parametric model selection for regression and classification based on statistical test," *Medium*, Aug. 2021, [Accessed: Jul. 16, 2023]. [Online]. Available: *https://22vignesh97.medium.com/parametric-vs-non-parametric-model-selection-for-regression-and-classification-based-on-statistical-152fddd4b384*.

[5] A. Rakhecha, "Importance of loss function in machine learning - towards data science," *Medium*, Sep. 2019, [Accessed: Jul. 16, 2023]. [Online]. Available: *https://towardsdatascience.com/importance-of-loss-function-in-machine-learning-eddaaec69519*.

[6] M. Vlastelica, "Learning theory: Empirical risk minimization - towards data science," *Medium*, Feb. 2019, [Accessed: Jul. 17, 2023]. [Online]. Available: *https://towardsdatascience.com/learning-theory-empirical-risk-minimization-d3573f90ff77*.

[7] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, Jul. 2014. DOI: *10.1017/CBO9781107298019*. [Online]. Available: *https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf*.

[8] P. Gomede Everton, "Independent and identically distributed (iid) in machine learning: Assumptions and implications," *Medium*, May 2023, [Accessed: Jul. 17, 2023]. [Online]. Available: *https://medium.com/@evertongomede/independent-and-identically-distributed-iid-in-machine-learning-assumptions-and-implications-930ee9821e14*.

[9] *Validation curve*, GeeksforGeeks, [Accessed: Jul. 18, 2023], Jul. 2020. [Online]. Available: *https://www.geeksforgeeks.org/validation-curve/*.

[10] J. Brownlee, *Step-by-step framework for imbalanced classification projects - machinelearningmastery.com*, Accessed May 07, 2023, 2020. [Online]. Available: *https://machinelearningmastery.com/framework-for-imbalanced-classification-projects/*.

[11] "Embeddings." [Accessed: Aug. 10, 2023], Google for Developers. (2022), [Online]. Available: *https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture#:~:text=An%20embedding%20is%20a%20relatively,like%20sparse%20vectors%20representing%20words.*.

[12] S. Karanam, "Curse of dimensionality — a 'curse' to machine learning," *Medium*, Aug. 2021, [Accessed: Aug. 17, 2023]. [Online]. Available: *https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb*.

[13] "1.2. linear and quadratic discriminant analysis." [Accessed: Aug. 17, 2023], scikit-learn. (2023), [Online]. Available: *https://scikit-learn.org/stable/modules/lda_qda.html#lda-qda*.

[14] T. Champion, M. Grześ, and H. Bowman, *Realising active inference in variational message passing: The outcome-blind certainty seeker*, 2021. arXiv: *2104.11798* [cs.LG].