Contents

Todo list

References:

General:
- Advanced Machine Learning Course 2020, ETHZ
- Standord CS229 course: http://cs229.stanford.edu/syllabus-fall2020.html

Ensemble methods:
- Images of Bagging, Boosting, Random Forest: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Clustering
- k-means vs EM https://towardsdatascience.com/a-comparison-between-k-means-clustering-and-expectation-maximization-estimation-for-clustering-8c75a1193eb7

Advanced Machine Learning HS 2020

Véronique Kaufmann

February 15, 2021

2 Representations

Goal of learning is to minimize the expected classification error, while maximizing generalization (avoid overfitting!). We can't really do this. But we can minimize the *empirical classification error* and maximize the *estimated empirical generalization performance* by cross validation.

2.1 The learning Problem

- Find function $f : \mathcal{X} \to \mathcal{Y}$
- Loss function Q measures the deviation between y and $f(x)$
  - $(y - f(x))^2$ quadratic loss (regression)
  - $\mathbb{I}_{\{y \neq f(x)\}}$ 0-1 loss (classification)
  - $\exp(-\beta y f(x))$ exponential loss (classification)
- Conditional expected risk: $R(f, X) = \int_{\mathcal{Y}} Q(Y, f(X)) P(Y|X) dY$
- Total Expected Risk:

$$R(f) = \mathbb{E}_X [R(f, X)] = \int_{\mathcal{X}} R(f, X) P(X) dY$$
$$= \int_{\mathcal{Y}} \int_{\mathcal{X}} Q(Y, f(X)) P(X, Y) dX dY$$

We divide the data intro training and test data to make sure we don't use the test data for calidation.

- Empirical Risk Minimizer $\hat{f} \in \arg\min \hat{R}(f, \mathcal{Z}^{\text{train}})$
- Training Error: $\hat{R}(\hat{f}, \mathcal{Z}^{\text{train}}) = \frac{1}{n} \sum_{i=1}^{n} Q(Y_i, \hat{f}(X_i))$
- Test Error: $\hat{R}(\hat{f}, \mathcal{Z}^{\text{test}}) = \frac{1}{m} \sum_{i=n+1}^{n+m} Q(Y_i, \hat{f}(X_i))$

2.1.1 Typical Approach

1. Feature Extraction (transform raw data to reduce information)
2. Classifier Design
3. Post-processing of classification results (adapt classifier output to enable stable scoring)
4. Scoring

2.2 Taxonomy of Data

Pattern Analysis = Find structure in sets of object representations.

- Object Space $\mathcal{O}$
- Measurement mapping x into a domain $\mathbb{K}$

2.2.1 Data

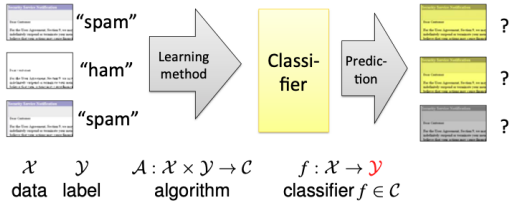
- Monadic data: $X \to \mathbb{R}^d, o \mapsto X_o$ (water depth, temperature, pressure, intensity, ...)
- dyadic data $\mathcal{O}^{(1)} \times \mathcal{O}^{(1)} \to \mathbb{R}, (o_1, o_2) \mapsto X_{o_1, o_2}$ (e.g. (Users, Websites)).
  Pairwise Data: $\mathcal{O} \times \mathcal{O} \to \mathbb{R}, (o_1, o_2) \mapsto X_{o_1, o_2}$ (e.g. image patches x image patches)
- polyadic data
- ...

Data type	design space $\mathcal{O}$	measurement space $\mathcal{X}$
vectorial data	monadic, $\mathcal{O}^{(1)}$	multi scale, $\mathbb{R}^D$
contingency tables	monadic, $\mathcal{O}^{(1)}$	categorial, $\bigotimes_{d=1}^{D} \{1, \ldots, n_d\}$
histogram data	monadic, $\mathcal{O}^{(1)}$	d-dim. probability simplex,
similarity data	pairwise, $\mathcal{O}^{(1)} \times \mathcal{O}^{(1)}$	quant. or ordinal, univariate $\mathbb{R}$
multiway similarity data	3–adic, $\mathcal{O}^{(1)} \times \mathcal{O}^{(2)} \times \mathcal{O}^{(2)}$	quant. or ordinal, univariate $\mathbb{R}$
co-occurrence	dyadic, $\mathcal{O}^{(1)} \times \mathcal{O}^{(2)}$	absolute, $\mathbb{N}$
n-th × occurrence	polyadic, $\mathcal{O}^{R}$	absolute, $\mathbb{N}$
single stimulus	dyadic, $\mathcal{O}^{(1)} \times \mathcal{O}^{(2)}$	quantitative, $\mathbb{R}$
rank preference	dyadic, $\mathcal{O}^{(1)} \times \mathcal{O}^{(2)}$	ordinal, $\mathbb{N}$
preferential choice	3–adic, $\mathcal{O}^{(1)} \times \mathcal{O}^{(2)} \times \mathcal{O}^{(2)}$	categorial, $\{-1, 0, 1\}$

2.3 Scales

- Nominal / Categorical: qualitattive, but without quantitative measurements (e.g. categories, binary, ...)
- Ordinal Scale: Values only meaningful w.r.t other measurements (rank order carries information, not the numerical differences)
- Quantitative Scale:
  - Interval scale: relation of numerical differences carries the information
  - Ratio scale: zero value of the scale carries information, but not the measurement unit
  - Absolute scale: Absolute value is meaningful

2.4 Learning Pipeline:



2.5 Mathematical Spaces

A pair of objects $(\mathcal{X}, d)$ consisting of a non-empty set $\mathcal{X}$ and a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **metric space**, if:

1. Positivity: $d(x, y) \geq 0 \forall x, y \in \mathcal{X}$
2. Uniqueness $d(x, y) = 0 \iff x = y$
3. Symmetry $d(x, y) = d(y, x)$
4. $\triangle$-inequality: $d(x, z) \leq d(x, y) + d(y, z), x, y, z \in \mathcal{X}$

2.5.1 Euclidean Vector Space

Let $\mathcal{V} = (\mathcal{X}, +, \cdot)$ a vector space. $x, y \in \mathcal{V}$. $\phi : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called **scalar product** if:

1. Distributivity: $\phi(x_1. + x_2, y) = \phi(x_1, y) + \phi(x_2, y)$
2. Commutativity: $\phi(x, y) = \phi(y, x)$
3. Homogeneity: $\phi(\alpha x, y) = \alpha \phi(x, y)$
4. Positive Definiteness: $\phi(x, x) > 0 \quad \forall x \neq 0$

A vector space with a scalar product is called **euclidean vector space**.

2.5.2 Probability Spaces

- Elementary event: $\omega_1, \ldots k, \omega_N$ are sample points
- Sample space: $\Omega = \{\omega_1, \ldots, \omega_N\}$
- Family of Sets: Event A of an experiment is a set of elementary events with $A \subset \Omega$, result of experiment is $\omega \in A$ or $\omega \notin A$
- Algebra of events $\mathcal{A}$ = set of subsets $A \subset \Omega$ with $\Omega \in \mathcal{A}$ and $(A \in \mathcal{A} \wedge B \in \mathcal{A} \implies A \cup B \in \mathcal{A} \wedge A \cap B \in \mathcal{A} \wedge A \backslash B \in \mathcal{A})$

Assign weights to elementary events with $0 \leq p(\omega_i) \leq 1$ and $\sum_i p(w_i) = 1$. The **Probability of an event** $A \in \mathcal{A}$ with $P(A) = \sum_{\{i : w_i \in A\}} p(w_i)$

A **probability model** is a triple $(\Omega, \mathcal{A}, P)$, with $\mathcal{P} = \{P(A) \mid A \in \mathcal{A}\}$

3 Density Estimation with Parametric Models

Overview:

- Maximum Likelihood estimation

$$\theta_{MLE} = \arg\max p(X \mid \theta) = \arg\max \sum_i p(x_i \mid \theta)$$

- Maximum a Posteriori estimation

$$\begin{aligned}\theta_{MAP} &= \arg\max P(\theta \mid X) = \arg\max P(X \mid \theta)p(\theta) \\ &= \arg\max \sum_i (\log p(x_i \mid \theta) + \log p(\theta))\end{aligned}$$

- If we have uniform distribution in the prior, the last term of the MAP sum goes away and we have $\theta_{MAP} = \theta_{MLE}$.

3.1 Bayesianism: MAP estimation

Assume $p(x)$ is unknown, $p(x \mid \theta)$ is known. We want to copmute $p(X = x \mid \mathcal{X})$

- Prior P(model)
- likelihood P(data | model)
- Posterior: P(model | data)
- evidence: P(data)
- Bayes Rule:

$$P(model \mid data) = \frac{P(data \mid model)P(model)}{P(data)}$$

3.2 Frequentism (Fisher): ML estimation

1. Define parametric model (e.g. $\mathcal{N}(\theta, 1)$)
2. Define the likelihood as a function of the parametric model (probability of the observations given the parameter θ), e.g.

$$\mathbf{P}(y_1, ..., y_n \mid \theta) = \prod_{i \leq n} \mathbf{P}(y_i \mid \theta) = \prod_{i \leq n} \mathcal{N}(y_i, \theta, 1)$$

3. compute an estimator by maximizing the likehood $\hat{\theta}_{ML} = \arg\max_\theta \mathbf{P}(y_1, ..., y_n \mid \theta)$, usually using the log-likelihood.

Properties of ML Estimators:

- Consistent ($\theta_{ML} \to \theta_0$) as $n \to \infty$
- Asymptotically normal: $1/\sqrt{n}(\theta_{ML} - \theta_0)$ converges in distribution to a random variable with distribution $\mathcal{N}(0, J^{-1}(\theta)I(\theta)J^{-1}(\theta)$
- Asymptotically efficient: θ_{ML} minimizes $\mathbb{E}\left[(\theta_{ML} - \theta_0)^2\right]$. I.e. $\mathbb{E}\left[(\theta_{ML} - \theta_0)^2\right] = \frac{1}{I_n(\theta_0)}$ (See Rao-Cramer Bound)

3.2.1 Rao Cramer Bound

There exists no estimator such that $\mathbb{E}\left[(\hat{\theta}^* - \theta_0)^2\right] = 0$,

$$\mathbb{E}\left[(\hat{\theta} - \theta_0)^2\right] \geq \frac{1}{I_n(\theta_0)} \text{ for any unbiased estimator } \hat{\theta} \text{ of } \theta_0.$$

where $I_n(\theta_0)$ is the Fisher Information:

$$I_n(\theta_0) = n\mathbb{V}_{\theta_0}\left[\left[\frac{\partial}{\partial\theta}\log \mathbf{P}(Y \mid \theta)\right]_{\theta_0}\right]$$

3.2.2 Stein estimator

For finite samples, the ML estimator is not necessarily efficient.

$$\hat{\theta}_{JS} = \left(1 - \frac{(d-2)\sigma^2}{\|y\|^2}\right)\mathbf{y}$$

is better for a multivariate random variable with distribution $\mathcal{N}(\theta_0, \sigma^2 I)$ with range $\mathbb{R}^d, d \geq 3$.

3.3 Conclusion

Bayesian Method	Frequentist method
Allows priors	Does not allow priods
Provides distribution when estimating parameters	Provides a single-point when estimating parameters
Requires efficient integration methods (when computing posteriors)	Only requires differentiation methods when estimating parameters
The prior often induces a regularization term	consistent, equivariant, asympt. normal + efficient

3.4 Summary ML Estimators

- Bias of an estimator: $\mathbf{bias}(\hat{\theta}_n) = \mathbb{E}[\hat{\theta}_n] - \theta$: deviation from the true parameter
- consistent estimator: $\forall\epsilon > 0, \mathbb{P}\{|\hat{\theta}_n - \theta| > \epsilon\} \xrightarrow{n\to\infty} 0$
- Efficiency: ML estimators are **asymptotically efficient**:

$$\lim_{n\to\infty}\left(\mathbb{V}[\hat{\theta}^{ML}(x_1, ..., x_n)]I(\theta)\right)^{-1} = 1$$

$$I(\theta) = \mathbb{V}\left[\frac{\partial \log P(x_1, ..., x_n|\theta)}{\partial\theta}\right]$$

-

3.5 ML vs. Bayes Estimation (MAP)

- They are asymptotically equivalent
- Advantage of ML: Only need differential calculus and gradient descent techniques. Bayes needs integration.
- A "flat" prior yields little information, since the model uncertainty is not significantly reduced.

4 Regression

- Optimal solution for regression $\arg\min_f \mathbb{E}(Y - f(X))^2$ given by $f^*(x) = \mathbb{E}(Y \mid X = x)$
  However, $\mathbf{P}(Y \mid X)$ and $\mathbf{P}(X)$ are unknown
- **(Parametric) maximum likelihood:**
  1. Assume $Y \mid X \sim \mathcal{N}(f(X), \sigma^2\mathbf{I})$
  2. Solve $\arg\max_f \sum_{i=1}^n \log \mathbf{P}(Y = y_i \mid X = x_i, \sigma^2)$
- **Statistical learning theory:**
  Directly minimize empirical risk $\arg\min_f \sum_{i=1}^n (y_i - f(x_i))^2$

4.1 Linear Regression

$$Y = \beta_0 + \sum_{j=1}^d X_j\beta_j = X^T\beta, Y \in \mathbb{R}$$

β_0 is called bias, $X, \beta \in \mathbb{R}^{d+1}$.

Ordinary least squares regression model: least-squares cost function.

$\Rightarrow$ Minimization either through gradient descent or closed form solution

4.1.1 Residual Sum of Squares (RSS)

$RSS(\beta) = \sum_{i=1}^n (y_i - x_i^T\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$
$\nabla_\beta RSS(\beta) = 0 \implies \hat{\beta} = (\mathbf{X^T X})^{-1}\mathbf{X}^T\mathbf{y}$

4.1.2 Prediction

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

where $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is an orthogonal projectin on the space spanned by the columns of $\mathbf{X}$.

- Assume additive gaussian noise: $Y = \beta_0 + \sum_{j=1}^d X_j\beta_j + \epsilon = X^T\beta + \epsilon$ with conditional mean $\mathbb{E}(Y \mid X_1, ..., x_d) = X^T\beta$
- Distribution: $\hat{\beta} \sim \mathcal{N}(\beta, (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2)$

4.1.3 Local weighting

We can add weights to the features, such that not all of them are treated equally:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i w_i (y_i - {x_i}^\top \beta)^2$$

If w_i is large, this feature will become very important, if w_i is small it will be pretty much ignored.
Predict: $y = x\beta$

4.2 Bias / Variance Dilemma

Goal: Minimize bias and variance simultaneously (usually impossible).

- Data $D = \{(x_i, y_i)_{i=1}^n\}, y_i \in \mathbb{R}$
- Objective: Find regression function $f \in \mathcal{C}$ such that $\mathbb{E}(Y - f(X))^2$ is minimal.
- Optimum $f^*(x) = \mathbb{E}(Y \mid X = x)$
- Estimator $\hat{f}(X)$ depends on random variable X and data $\mathcal{D}$
- **Problems:** we only have a finite training set, and the complexity of hypothesis class $\mathcal{C}$ is unknown.

Tradeoff:

- Small data sets and large $\mathcal{C}$: Variance large, bias small
- Large data set, small $\mathcal{C}$: Variance small, bias large

The optimal tradeoff between bias and variance is achieved when we avoid both underfitting (large bias) and overfitting (large variance)

Prediction error:

$$\begin{aligned}\mathbb{E}_D \mathbb{E}_{Y|X=x} \left(\hat{f}(x) - Y\right)^2 = \\ \text{(variance)} \quad & \mathbb{E}_D(\hat{f}(x) - \mathbb{E}_D(\hat{f}(x))^2 \\ \text{(bias}^2\text{)} \quad & + \left(\mathbb{E}_D(\hat{f}(x)) - \mathbb{E}(Y \mid X = x)\right)^2 \\ \text{(noise)} \quad & + \mathbb{E}(Y - \mathbb{E}(Y \mid X = x))^2\end{aligned}$$

4.3 Overfitting

- **Regularization**: Add model complexity term to cost function $\arg\min_\theta \sum_{i=1}^n l(f(x_i, \theta, y_i) + R(\theta)$
  Often equivalent to choosing a prior in a bayesian framework and using a MAP estimator.
- Model selection based on **generalization error estimate** (e.g. cross-validation)
- **Ensembles** of classifiers (See Section 9)

4.4 Ridge Regression, LASSO

	Ridge Regression	LASSO
Reg. R	$\lambda \beta^T \beta$	$\lambda \|\beta\|_1$
Bay. view:	$Y\|(X,\beta) \sim \mathcal{N}(x^T\beta, \sigma^2\mathbf{I})$, prior $\beta : \beta \sim \mathcal{N}(0, \sigma^2/\lambda\mathbb{I})$	$Y\|(X,\beta) \sim \mathcal{N}(x^T\beta, \sigma^2\mathbf{I})$, Laplace prior: $p(\beta_i) = \frac{\lambda}{4\sigma^2} \exp(-\|\beta\| \frac{\lambda}{2\sigma^2})$
Solution	$\hat{\beta}_\lambda^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$	Optimization techniques (e.g. LARS), $\|\beta\|_1$ is not differentiable.

4.4.1 Singular Value Decomposition

- Take centered X and perform SVD: $X = UDV^T$ (U, V have orthonormal columns)
- SVD of least squares fitted vector:
  $\mathbf{X}\hat{\beta}^{ls} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{U}\mathbf{U}^T\mathbf{y}$

4.4.2 Ridge regression:

If the β_j are unconstrained, they can explode (high variance). We can **control the variance by regularizing the coefficients:**

Constraint:

$$\min \sum_{i=1}^n (y_i - \beta^\top x_i)^2 \text{ s.t. } \sum_{j=1}^d \beta_j^2 \le t$$

$$\iff \min (y - \mathbf{X}\beta)^\top (y - \mathbf{X}\beta) \text{ s.t. } \sum_{j=1}^d \beta_j^2 \le t$$

Assume $\mathbf{X}$ standardized, $\mathbf{y}$ centered

We can rewrite this as

$$\begin{aligned} PRSS(\beta)_{l_2} &= \sum_{i=1}^n (y_i - {x_i}^\top \beta)^2 + \lambda \sum_{j=1}^d \beta_j^2 \\ &= (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) + \lambda \|\beta\|^2 \end{aligned}$$

$$\begin{aligned} \mathbf{X}\hat{\beta}^{\text{ridge}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{y} \\ &= \sum_{i=1}^d \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda} \mathbf{u}_j^T \mathbf{y} \end{aligned}$$

$\frac{d_j^2}{d_j^2+\lambda}$ is small for small singular values d_j and goes to 1 for large SV.
This suppresses contributions of small eigenvalues.

- $\hat{\beta}^{ls} = \beta + (\textit{Inflammation term dep. on } \Sigma^{-1})\epsilon$ (Derivation)
- With multicollinearity (i.e. columns of $\mathbf{X}$ are not independent), some d_i, d_j will be close to 0. The diagonal element $1/d_j$ will then be huge.
- This leads to large inflammation term and therefore **great deviation in the least squares weights from the true weights.**
- Ridge regression projects $\mathbf{y}$ onto the components with large d_j
- This shrinks the coefficients of low-variance components.

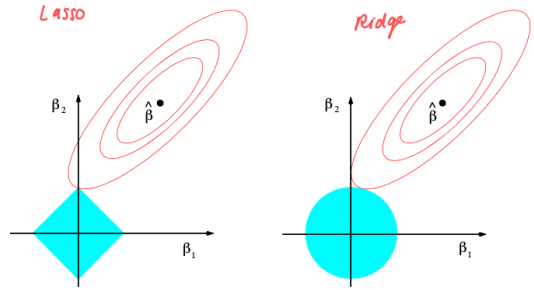
4.4.3 The LASSO (Least Absolute Shrinkage and Selection Operator)

$$\begin{aligned} \hat{\beta}^{\text{LASSO}} &= \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^d x_{i,j}\beta_j)^2 \\ &= \underset{\beta}{\operatorname{argmin}} (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) \\ &\text{subject to } \sum_{j=1}^d |\beta_j| \le s. \end{aligned}$$

We can rewrite this as

$$\begin{aligned} PRSS(\beta)_{l_1} &= \sum_{i=1}^n (y_i - {x_i}^\top \beta)^2 + \lambda \sum_{j=1}^d |\beta_j| \\ &= (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) + \lambda|\beta_j| \end{aligned}$$

LASSO estimates are known to be sparse with few coefficients non-vanishing (Reason: LSE error surface hits the corners of the constraint surface often).



Contours of error and constraint functions. Blue = constraint regions, red=contours of least squares error function.

- Often, we believe that some β_j should be 0.
- Large λ will set some coefficients equal to $0 \rightarrow$ sparse solution
- **LASSO will perform model selection for us**

4.5 nonlinear regression with basis expansion

Idea: Transform variables X nonlinearly and fit a linear model in the resulting feature space.

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$
$$h_m(X) : \mathbb{R}^d \mapsto \mathbb{R}, 1 \leq m \leq M$$

Smoothing Splines: Cubic splines are a common choice for h.

- Know selection: Use the maximal number of knots and control the smoothness by regularization:

$$RSS(f, \lambda) = \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int (f''(x))^2 dx$$

4.6 Regression with Wavelets

Many phenomena are local perturbations (e.g. face muscles) and can hence be modeled by wavelets (?).
Haar wavelet, symmlet-8 wavelet, denoising by wavelet shrinkage

5 Gaussian Processes

Different way to go nonlinear.
Remember linear regression: $Y = X^T\beta + \epsilon, \epsilon \sim \mathcal{N}(\epsilon, 0, \sigma^2)$ or equivalently:

$$p(Y \mid X, \beta, \sigma) = \mathcal{N}(Y \mid X^T\beta, \sigma^2) \propto \exp\left(-\frac{1}{2\sigma^2}(Y - X^T\beta)^2\right)$$

5.1 Multivariate Gaussians

A vector $x \in \mathbb{R}^d$ has multivariate normal distribution with mean $\mu \in \mathbb{R}^d$ and covariance $\Sigma \in \mathbf{S}^d$ (symmetric positive definite $n \times d$ matrix) $x \sim \mathcal{N}(\mu, \Sigma)$, if

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T\Sigma^{-1}(x - \mu)\right)$$

5.1.1 Why Gaussians are useful

- Common when modeling noise in statistical algorithms (can be considered as accumulation of small independent random perturbations);
  Summation of independent random variables tend to look gaussian.
- Convenient for many analytical manipulations, because many of the integrals that arise have closed form solutions.

5.2 Bayesian Linear Regression

Assume a prior distribution over parameters and obtain posterior using Bayes's rule.

We can turn conventional multiple linear regression model into a bayesian regression model by defining a **prior** over the regression coefficients.
Goal: Probabilistic approach; find a distribution over the parameters that gets updated whenever new data points are observed.

$$p(\beta \mid \Lambda) = \mathcal{N}_d(\beta \mid \mathbf{0}, \Lambda^{-1}) \propto \exp(-1/2\beta^T\Lambda\beta),$$

where $\Lambda = \Sigma^{-1}$ is the precision matrix.
This favors β to be 0.
The **Posterior** then becomes:

$$p(\beta \mid \mathbf{X}, \mathbf{y}, \Lambda) = \mathcal{N}(\beta \mid \mu_\beta, \Sigma_\beta)$$
$$\mu_\beta = (\mathbf{X}^T\mathbf{X} + \sigma^2\Lambda)^{-1}\mathbf{X}^T\mathbf{y}$$
$$\Sigma_\beta = \sigma^2(\mathbf{X}^T\mathbf{X} + \sigma^2\Lambda)^{-1}$$

Bayesian linear regression with gaussian prior over the coefficients is equivalent to ridge regression for $\Lambda = \lambda\mathbb{I}_d, \sigma = 1$

5.3 Gaussian Processes

Gaussian processes are the extension of multivariate Gaussians to infinite-sized collections of real-values variables (Gaussian processes are distributions over random functions and not just random vectors).

Goal: Non-parametric approach: Find a distribution over the possible functions $f(x)$ that are consistent with the observed data (is also a Bayesian approach: Start with a prior, update it whenever new data points are observed; produce posterior).

- Moments of joint Gaussian:
  $\mathbb{E}[\mathbf{y}] = \mathbf{0}, Cov[\mathbf{y}] = \mathbf{X}\Lambda^{-1}\mathbf{X}^T + \sigma^2\mathbb{I}_n$
- Consider the joint distribution over all $\mathbf{y}$:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{N}\left(\mathbf{y} \middle| \mathbf{0}, \begin{bmatrix} k_{1,1} + \sigma^2 & k_{1,2} & \dots & k_{1.n} \\ k_{2,1} & k_{2,2} + \sigma^2 & \dots & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n,1} & k_{n,2} & \dots & k_{n,n} + \sigma^2 \end{bmatrix}\right)$$

  where $k_{i,j} = k(x_i, x_j))x_i^T\Lambda^{-1}x_j$ is a Kernel function (here, any kernel function could be used, which gives it power and flexibility).
- This means, that outputs of points whose inputs are similar to each other, have a high covariance (non-axis-sligned ellipse level set of joint distributions)

5.3.1 Kernel Functions

Kernel functions specify the degree of similarity between any 2 datapoints. They encode our assumptions about the function we wish to learn (different kernels can obtain very different models).
Properties:

1. Symmetry: $k(x, x') = k(x', x)$
2. Positive Semi-Definiteness:
   $k(x, x')f(x)f(x')dxdx' \geq 0 \forall f \in L - 2, \Omega \subset \mathbb{R}^d$

The **Gram Matrix** $\mathbf{K}$ is positive semi-definite $/iffx^T\mathbf{K}x \geq 0$
Examples of kernel functions:

- Linear kernel: $k(x, x') = x^Tx'$
- Polynomial Kernel $k(x, x') = (x^Tx' + 1)^p$
- Gaussian (RBF) kernel: $k(x, x') = \exp(-\|x - x'\|_2^2 (h^2)$ (judges how different x and x' are.
- Sigmoid (tanh) kernel: $k(x, x') = \tanh \kappa x^Tx' - b$

Different kernels have different invariance properties (e.g. invariance to rotation or translation)

5.3.2 Prediction by Gaussian Processes

The predictive density $p(y_{n+1} \mid x_{n+1}, \mathbf{X}, \mathbf{y})$ can be obtained analytically.

$$p\left(\begin{bmatrix} \mathbf{y} \\ y_{n+1} \end{bmatrix} \middle| x_{n+1}, \mathbf{X}, \sigma\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{y} \\ y_{n+1} \end{bmatrix} \middle| \mathbf{0}, \begin{bmatrix} \mathbf{C}_n & \mathbf{k} \\ \mathbf{k}^T & c \end{bmatrix}\right)$$
$$\mathbf{C}_n = \mathbf{K} + \sigma^2\mathbf{I}$$
$$c = k(x_{n+1}, x_{n+1}) + \sigma^2$$
$$\mathbf{k} = k(x_{n+1}, \mathbf{X})$$
$$\mathbf{K} = k(\mathbf{X}, \mathbf{X})$$

Algorithm 1: Prediction with Gaussian processes

Require:
n observed data $\mathbf{X} = (x_1, ..., x_n)^\top \in \mathbb{R}^{n \times d}$,
k kernel function
σ^2 noise variance $x_{n+1} \in \mathbb{R}^d$ new data point

$\mathbf{K} \leftarrow (k(x_i, x_j))_{1 \leq i,j \leq n}$; // Compute kernel matrix
$\mathbf{k} \leftarrow (k(x_{n+1}, x_i))_{1 \leq i \leq n}$; // Similarity of new and observed data
$\mu_{y_{n+1}} \geq \mathbf{k}^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$; // Mean of predictive distribution
$\sigma^2_{y_{n+1}} \geq k(x_{n+1}, x_{n+1}) - \mathbf{k}^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}$; // Var of predictive distribution
return $\mathcal{N}(y_{n+1}|\mu_{y_{n+1}}, \sigma^2_{n+1})$; // Return predictive distribution

5.3.3 Reasons to use Gaussian processes for Regression

1. Like Bayesian methods, GP models allow to quantify uncertainty from errors in the parameter estimation procedure (not just instrinsic noise).
2. Non-parametric and can hence model essentially arbitrary functions of the input points
3. Natural ways to introduce kernels into a regressino modeling framework
4. Simple and straightforward linear algebra implementations

5.3.4 Wisdom of Crowds

Averaging over multiple estimators:
- unbiased estimators remain unbiased after averaging
- Variance is reduced

6 Linear Discriminant Methods

Discriminant methods aim at estimating $P(X|Y)$ and $P(Y)$ and then use the bayes rule to estimate $P(Y|X)$, instead of directly estimating the latter.
- Tend to model a decision boundary between classes / labels. Goal is to find the boundary separating one class from abother

6.1 Discriminative / Generative Models

Discriminative models model the decision boundary between the classes, while generative model explicitly model the distribution of each class. Reference: Medium Blogpost.

6.1.1 Models

Generative model: statistical model of the joint probability distribution $p(x, y)$ **Important:** Lecture's notation: $p(x, y \mid \theta)$,where θ is the assumed model.
Examples: Hidden Markov Models, Naive Bayes, Bayesian Networks, LDA/QDA

Discriminative model: Model of the conditional distribution $p(y|x)$
Examples: Threshold functions (perceptron), Fisher's linear discriminant, SVMs, Traditional Neural Networks

6.1.2 Classifiers

Probabilistic Generative Classifier
Based on the generative model. Called generative because the joint probability distribution P(X, Y) can be used to generate samples.
1. Assume distribution of labels $(Y|\theta)$
2. Assume distribution of samples conditioned on labels: $P(X|Y = y)$
3. Perform MLE over likelihood

$$P(\mathbf{X}, \mathbf{y}|\theta) = P(\mathbf{y})P(\mathbf{X}|\mathbf{y}, \theta)$$

4. Compute posterior $P(Y|X)$ with LDA or QDA
5. Create classifier from $P(y|X)$ using Bayes decision theory

$$y = \underset{y}{\operatorname{argmax}}\, P(y|X) = \underset{y}{\operatorname{argmax}}\, P(y) \prod_{i=1}^{n} p(x_i|y)$$

Probabilistic Discriminative Classifier
Based on the discriminative models. Called discriminative because classifier can be used to discriminate the value of the target variable Y.
1. Assume the posterior has the form $P(Y|X) = \sigma(w^\top x + w_0)$. After normalization $P(Y|X) = \sigma(\tilde{w}^\top x)$
2. Use MLE over likelihood

$$P(\mathbf{y}|\mathbf{w}, \mathbf{X}) = P(\mathbf{y}|\mathbf{X}, \mathbf{w})$$

to get

$$\begin{aligned} L(w) &= \log P(\mathbf{y}|\mathbf{w}, \mathbf{X}) \\ &= c + \sum_{i \le n} \left[y_i \log \sigma(\mathbf{w}^\top) x_i + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top x_i) \right] \end{aligned}$$

3. Do gradient Descent or Newton's method over $NL(w) = -L(w)$, since $L(w)$ is intractable.
4. Use weight vector w^* to predict.

Discriminative Classifier *The discriminative classifier is not based on any model, but classifies directly.*
1. Choose a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$
2. Approximate expected risk $\mathbb{E}_{X,Y}[\mathcal{L}(y, c(X))]$ with the empirical loss $\hat{R} = \frac{1}{n} \sum_{i \le n} \mathcal{L}(y_i, c(x_i))$ *(abuse of notation)*
3. Find the optimal classifier c^*, such that $c^* = \operatorname{argmin}_c \hat{R}$

6.2 Marginal Distributions Recap

From $P(X, Y)$, we can compute

$$\begin{aligned} P(X) &= \sum_y P(X, Y = y) \\ P(Y) &= \int_x P(Y, X = x) \\ P(X|Y) &= P(X, Y)/P(Y) \\ P(Y|X) &= P(X, Y)/P(X) \end{aligned}$$

6.3 Discriminant Functions

Function that takes an input vector x and assigns it to one of the k classes $\mathcal{C}_k$

6.3.1 Least Squares (LDA, QDA)

Make the model predictions as close as possible to a set of target values.
- **Linear Discriminant Analysis (LDA):** Assume the classes have the same covariances (i.e. $\Sigma_0 = \Sigma_1$)
  $p(y \mid x) = \sigma(\mathbf{w}^T x + w_0)$
- **Quadratic discriminant Analysis (QDA):** General (no cov. assumption)
  $p(y \mid x) = \sigma(x^T \mathbf{W} x + x^T \mathbf{w} + w_0)$

6.3.2 Fisher's Linear Discriminant

Classification as dimensionality reduction. Goal is to maximize class separation in the output space.
- choose $\mathbf{w}$ to maximize $m_2 - m_1 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1)$ with $m_k = \mathbf{w}^T \mathbf{w}_k$ ($\mathbf{m}_i$ is the mean of class i).

$$w^* \propto S_w^{-1}(\overline{x}_0 - \overline{x}_1)$$

- Within-class variance: $s_k^2 = \sum_{n \in \mathcal{C}_k} (w^T(x_n - \overline{x}_k))^2$
- **Fisher criterion:** between class variance vs within-class variance:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T(\overline{x}_1 - \overline{x_2})(\overline{x}_1 - \overline{x_2})^T \mathbf{w}}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

, where $\mathbf{S}_B$ is the between-class variance and $\mathbf{S}_W$ is the within class variance.

$$\begin{aligned} \mathbf{S}_B &= (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \\ \mathbf{S}_W &= \sum_{n \in \mathcal{C}_1} (x_n - \mathbf{m}_1)(x_n - \mathbf{m}_1)^T \\ &\quad + \sum_{n \in \mathcal{C}_2} (x_n - \mathbf{m}_2)(x_n - \mathbf{m}_2)^T \\ &= \mathit{Cov}(\mathcal{C}_1) + \mathit{Cov}(\mathcal{C}_2) \end{aligned}$$

For multiple classes, Fisher works as well. See *Pattern Recognition and Machine Learning, p.192*.
Classification with fisher: $\mathbf{w}^T x = \sum_i w[i]x[i]$
1. Fisher's projection w^*
2. Fit mix of gaussians
3. Bayes decision theory

6.4 Perceptron Algorithm

- **Goal:** Compute $w \in \mathbb{R}^d : \begin{cases} w^T x_i > 0 & \textit{if } y_i = +1 \\ w^T x_i < 0 & \textit{if } y_i = -1 \end{cases}$

  Threshold function: $c(x) = \textit{sgn}(w^T x)$
- **Cost Function:**

$$\begin{aligned} \mathcal{L}(y, c(x)) &= \begin{cases} 0 & \textit{if } c(x) = y \begin{cases} w^T x > 0 \wedge y = +1 \\ \textit{or} \\ w^T x < 0 \wedge y = -1 \end{cases} \\ |w^T x| & \textit{if } c(x) \neq y \end{cases} \\ &= \begin{cases} 0 & \textit{if } y w^T x > 0 \\ |w^T x| & \textit{if } y w^T x < 0 \end{cases} \\ &= \sum_{i \in \mathcal{M}} -y_i w^T x_i \end{aligned}$$

Algorithm 2: Variable increment perceptron

```
k ← 0
w^(k) ← $
do
    if y_k w^(k)^T x_k < 0 then
        w^(k+1) ← w^(k) + η(k) y_k x_k
    end
    k ← k + 1
until ||η(k) Σ_i y_i x_i|| < ε;
```

Variable increment perceptron converges if
- Train set is linearly separable
- $\eta(k) \geq 0$
- $\sum_{k=0}^{t} \eta(k) \to \infty$ for $t \to \infty$
- $\frac{\sum_{k \leq t} \eta^2(k)}{\left(\sum_{k \leq t} \eta(k)\right)^2} \to 0$ for $t \to \infty$

6.5 Loss-Functions

0-1 Loss: Piecewise continuous, not differentiable

$$\mathcal{L}^{0-1}(y, c(x)) = \begin{cases} 0 & \textit{if } c(x) = y \\ 1 & \textit{if } c(x) \neq y \end{cases}$$

exponential Loss: Used in AdaBoost

$$\mathcal{L}^{\exp}(y, c(x)) = \exp(-y c(x)) \begin{cases} e^{-1} & \textit{if } c(x) = y \\ e & \textit{if } c(x) \neq y \end{cases}$$

Hinge Loss: Used in SVMs

$$\mathcal{L}^{\text{hinge}}(y, c(x)) = \begin{cases} 0 & \textit{if } y w^T x \geq 1 \\ -y w^T x & \textit{otherwise} \end{cases}$$

Perceptron Loss:

$$\mathcal{L}^{\text{perc}}(y, c(x)) = \begin{cases} 0 & \textit{if } y w^T x \geq 0 \\ -y w^T x & \textit{otherwise} \end{cases}$$

6.5.1 Gradient Descent

If the loss function is analytically tractable and differentiable, we can use gradient descent to find $\max_w \log p(data \mid w) = \max_w L(w)$
We define

$$\textit{NL}(w) := -L(w) \qquad \textit{The loss function criterium}$$

Algorithm 3: Gradient Descent

```
k ← 0
w^(k) ←
do
    w^(k+1) ← w^(k) − η(k) ∇NL(w^(k))
    k ← k + 1
until ||η(k) ∇NL(w^(k))|| < ε;
```

How to choose learning rate?
- $\eta(k) = \arg\min_\eta \textit{NL}(w^{(k+1)})$
- Approximate NL at $w^{(k)}$ with an analytically tractable approximation

$$\begin{aligned} \textit{NL}(w^{(k+1)}) \approx \textit{NL}(w^{(k)}) &+ (w^{(k+1)} - w^{(k)})^T \nabla \textit{NL}(w^{(k)}) \\ &+ \frac{1}{2}(w^{(k+1)} - w^{(k)})^T \mathbf{H}_{\textit{NL}}(w^{(k)})(w^{(k+1)} - w^{(k)}) \end{aligned}$$

to get

$$\eta(k) = \frac{\left\| \textit{NL}(w^{(k)}) \right\|^2}{\nabla \textit{NL}^T(w^{(k)}) \mathbf{H}_{\textit{NL}} \nabla \textit{NL}(w^{(k)})}$$

6.5.2 Newton's Method

Alternative approach to gradient descent:
Choose $w^{(k+1)} = \arg\min \textit{NL}(w)$ s.t. $w \in$ *neighborhood of* $w^{(k)}$

Algorithm 4: Newton's Method

```
k ← 0
w^(k) ← $
do
    w^(k+1) ← w^(k) − H_NL^{-1}(w^(k)) ∇NL(w^(k))
    k ← k + 1
until ||η(k) ∇NL(w^(k))|| < ε;
```

Comparison:
- Gradient Descent: Depends on η, but is computationally easier
- Newton's Method: Requires $\mathbf{H}_{\textit{NL}}^{-1}$ but gets better updates and does not require a learning rate.

7 Support Vector Machines

Perceptron is instable: It does not choose the best classifier, but just any.

7.1 Maximum Margin Classifiers

Maximum-margin criterion: Maximize the margin between the classes.

$$\begin{aligned} &\min_{\mathbf{w}, w0} \frac{1}{2} \|\mathbf{w}\|^2 \\ &\textit{s.t.} : 1 - y_i(\mathbf{w}^T x_i + w_0) \leq 0 \end{aligned}$$

7.2 Constrained Convec Optimization using Lagrange

7.2.1 Lagrange

Solve $\min_w f(w)$ s.t. $g_i(w) = 0$ by solving

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{l} \alpha_i g_i(w)$$

where α_i are called lagrangian multipliers.

7.2.2 Lagrange Dual Formulation

$\min_w f(w)$ *s.t.* $g_i(w) = 0, h_j(w) \leq 0$

1. Generalized Lagrangian: $\mathcal{L}(\mathbf{w}, \lambda, \alpha) = f(\mathbf{w}) + \sum_i \lambda_i g_i(\mathbf{w}) + \sum_j \alpha_j h_j(\mathbf{w}), \alpha_j \geq 0$
2. $$d^* = \max_{\lambda, \alpha : \alpha_j \geq 0} \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \lambda, \alpha) \leq \min_{\mathbf{w}} \max_{\lambda, \alpha : \alpha_j \geq 0} \mathcal{L}(\mathbf{w}, \lambda, \alpha) = p^*$$

   Strong Duality: $d^* = p^*$.
   Slaters Conditions: is there a $\mathbf{w}$ such that $g_i(\mathbf{w}) = 0$ and $h_j(\mathbf{w}) < 0$? Sufficient for Strong duality
3. solve dual $\max_{\alpha, \lambda} \min_w \mathcal{L}$, get α_i
4. compute α_i to compute the weights. This way, we don't have to compute the scalar product $w^\top x$, which may be expensive if d is large.

$\alpha_i = 0$ for all but a few vectors (the support vectors). We can then compute:

7.2.3 Lecture:

$$\min_w f(w) \quad \textit{s.t. } g_i(w) = 0, h_j(w) \leq 0$$

("primal optimization problem")

1. Solve using generalized Lagrangian:

$$\mathcal{L}(\mathbf{w}, \lambda, \alpha) = f(\mathbf{w}) + \sum_i \lambda_i g_i(\mathbf{w}) + \sum_j \alpha_j h_j(\mathbf{w}), \alpha_j \geq 0$$

   λ_i, α_i are the lagrangian multipliers.
2. Check **Slaters condition:** *Is there* w *s.t.* $g_i(w) = 0$ *and* $h_j(w) < 0$ *for* $i \leq n, j \leq m$ (conditions strictly fullfilled).
   If $\mathbf{w}$ violates the constraints, we can verify that $\max \mathcal{L} = \infty$. Otherwise, $\max \mathcal{L} = f(\mathbf{w})$.
   We can hence consider

$$\min_w \max_{\alpha,\lambda} \mathcal{L}(\mathbf{w}, \lambda, \alpha)$$

   We have

$$\underbrace{\max_{\alpha,\lambda} \min_w \mathcal{L}(\mathbf{w}, \lambda, \alpha)}_{\textit{Dual Formulation}} \leq \min_w \max_{\alpha,\lambda} \mathcal{L}(\mathbf{w}, \lambda, \alpha)$$

3. Solve

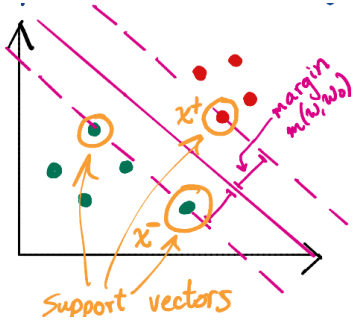
$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0, g_i(\mathbf{w}) = 0, g_j(\mathbf{w}) \leq 0$$
$$\alpha_j \geq 0 \alpha_j h_j(\mathbf{w}) = 0$$
(compl. slackness)

7.2.4 Strong duality

- Strong duality: primal optimal objective and the dual optimal objective are equal.
  $\max_{\lambda,\alpha} \min_w \mathcal{L}(w, \lambda, \alpha) = \min_w f(w)$
- Slater's condition is a suffcient condition for strong duality.
- Weak duality $\max_{\lambda,\alpha} \min_w \mathcal{L}(w, \lambda, \alpha) \leq \min_w f(w)$

7.3 SVM: Compute a maximum margin classifier

Assume the 2 classes are linearly separable.



Goal:

$$\max_{\mathbf{w},w_0} 2m(\mathbf{w}, w_0) \quad \textit{s.t.}$$
$$\mathbf{w}^T x_i + w_0 > 0 \iff y_i = +1$$
$$\mathbf{w}^T x_i + w_0 < 0 \iff y_i = -1$$
$$\equiv y_i(\mathbf{w}^T x_i + w_0) \geq b > 0$$

$$\begin{aligned} 2m(\mathbf{w}, w_0) &= \left\| \textit{proj}_{\mathbf{w}} x^+ - \textit{proj}_{\mathbf{w}} x^- \right\| \\ &= \left\| \frac{\mathbf{w}^T x^+}{\|\mathbf{w}\|^2} \mathbf{w} - \frac{\mathbf{w}^T x^-}{\|\mathbf{w}\|^2} \mathbf{w} \right\| \\ &= \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T x^+ - \mathbf{w}^T x^-| \end{aligned}$$

$m(\mathbf{w}, w_0)$ does not depend on $\|\mathbf{w}\|$ nor w_0: Let $\gamma \in \mathbb{R}$:

$$m(\gamma \mathbf{w}, w_0) = \frac{1}{\gamma \|\mathbf{w}\|} |\gamma \mathbf{w}^T x^+ - \gamma \mathbf{w}^T x^-| = m(\mathbf{w}, w_0)$$

There is w^*, w_0^* optimal such that $w^{*T} x^+ + w_o^* = 1$ and $w^{*T} x^- + w_0^* = -1$:

$$2m(\mathbf{w}^*, w_0^*) = \frac{1}{\|\mathbf{w}^*\|} |\mathbf{w}^{*T} x^+ - \mathbf{w}^{*T} x^-| = \frac{2}{\|\mathbf{w}^*\|}$$
$$w^{*T} x_i + w_0^* \geq w^{*T} x^+ + w_0^* = 1 \textit{ if } y_i = +1$$
$$w^{*T} x_i + w_0^* \leq w^{*T} x^- + w_0^* = 1 \textit{ if } y_i = -1$$

SVM formulation:

$$\min_{\mathbf{w},w0} \frac{1}{2} \|\mathbf{w}\|^2$$
$$\textit{s.t.} : y_i(\mathbf{w}^T x_i + w_0) \geq 1$$

Slaters Condition

- By linear separation: $\exists \mathbf{w}, w_0 : y_i(\mathbf{w}^T x_i + w_0) \geq 1, i \leq n$
- Take $\gamma > 1 \implies (\gamma \mathbf{w}, \gamma w_0)$

$$\begin{aligned} y_i(\gamma \mathbf{w}^T x_i + \gamma w_0) &= \gamma y_i(\mathbf{w}^T x_i + w_0) \\ &> y_i(\mathbf{w}^T x_i + w_0 \\ &\geq 1 \end{aligned}$$

Dual

$$\begin{aligned} \mathcal{L}(\mathbf{w}, w_0, \alpha) &= \frac{1}{2} \|w\|^2 + \sum_i \alpha_i (1.yi(\mathbf{w}^T x_i + w_0)), \alpha_i \geq 0 \\ &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_i \alpha_i \\ &\quad - \sum_i \alpha_i y_i \mathbf{w}^T x_i - \sum_i \alpha_i y_i w_0 \end{aligned}$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} + 0 - \sum_i \alpha_i y_i x_i + 0$$
$$\frac{\partial \mathcal{L}}{\partial w_0} = 0 + 0 + 0 - \sum_i \alpha_i y_i$$

Setting the derivatives to 0 yields:

$$w^* = \sum_i \alpha_i y_i x_i, \quad \sum_i \alpha_i y_i = 0$$

Plugging this into the dual formulation, we get:

$$\max_\alpha \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$
$$\alpha_i \geq 0, \sum_i \alpha_i y_i = 0$$

which simplifies to

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$
$$\alpha_i \geq 0, \sum_i \alpha_i y_i = 0$$

7.4 Shockfish

Shockfish aims at predicting parking lot occupancy. Classification task: Occupied vs. non-occupied?

Idea:

- Sense earth magnetic fields and measure deviations
- Deviations occur when a large mass (e.g. truck) is close to the sensor
- Sensor at each parking lot, use readings to detect presence of vehicle.

Sensor Readings:

- Center Values: Temperature and other physical variations
- Data set: $\textit{signal}(t) = \textit{reading}(t) - \textit{center}(t)$

Problem: Truck also influences sensor measurements of neighboring sensors

⇒ Weak coupling of neighboring sensors

7.4.1 Challenges

- Design new algorithm
- Label the readings of three weeks based on camera imagery
- find approp. data preprocessing filters
- Performance measures: Accuracy, Generalization (over sensors and parking lots).
- Have to carefully specify conditioning on sensor or parking lot.

7.4.2 Problems in Real World Applications

- Non uniformity of sensors (not aligned, calibration errors, ...)
- Signals for occupied lots are not homogeneous (every truck generates different signal depending on steel mass and relative position)

7.4.3 Data Preprocessing

Goal: Fina transformation of the data such that the readings are as comparable as possible and the signal discriminability between occupied and non-occupied is maintained.

Comparison of readings when the entire lot is empty (rest readings) to compare different sensors.

Preprocessing:

1. Center Values
2. Subtract median of rest readings per sensor
3. Subtract minimal rest readings accross sensors
4. Transform to spherical coordinates

7.4.4 Modeling: Classify parking space occupancy

Types of information: Spatial (individual sensors, neighborhood), transition information (changes between consecutive readings)

Classifiers: SVM, Random Forest, Graphical Models

Features:

- Measurements (Z-axis informative, X-axis not informative)
- Spherical coordinates: r very informative, angles exhibit high variations in consecutive readings in the non occ. state
- Time of the day (likelihood varies)

8 SVMs for non-linear Classification

Solutions for not linearly separable training sets:

- Soft-margin SVM
- Kernels

8.1 Soft-Margin SVMs

Train a max-margin classifier (see Section 7), but neglect some samples.

$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i\leq n}\xi_i$$

$$\textit{s.t.} \quad y_i(\mathbf{w}^T x_i + w_0) \geq 1 - \xi_i, \xi_i \geq 0$$

Tradeoff: wide. margin C = many samples neglected, narrow margin= few samples neglected. ξ_i lowers the bar for each neglected example.

Dual:

$$\max_\alpha \sum_i -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

$$\textit{s.t.} \quad 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0$$

Optimum:

$$w^* = \sum_i \alpha_i^* y_i x_i$$

$$\xi_i^* = \max(0, 1 - y_i(w^{*T}x_i + w_0^*))$$

8.2 Kernels

Kernels transform the data from a not linearly separable space to a linearly separable space.

Polynomial Transformation:

$$\varphi : (t, c) \mapsto (1, t, c, t^2, c^2, tc, t^3, t^2c, tc^2, c^3, ...)$$

Using this, we can go from polynomial classification to linear classification:

$$\sum_{j=1}^{\infty}\sum_{n_1+n_2=j} w_{n.n_2}t^{n_1}c^{n_2} = w_0 0 + w_1 t + w_2 c + ... = w^T\varphi(t,c)$$

8.2.1 SVMs and kernels

1. Training:

$$\max_\alpha \sum_{i\leq n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \varphi(x_i)^T\varphi(x_j)$$

$$\textit{s.t.} \quad 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0$$

2. Classification:

$$w^{*T}\varphi(x) = \left(\sum_i \alpha_i^* y_i \varphi(x_i)\right)^T \varphi(x) = \sum_i \alpha_i^* y_i \varphi(x_i)^T\varphi(x)$$

We do not need $\varphi(x_i)$ or $\varphi(x)$. We just need $\varphi(x_i)^T\varphi(x)$.

8.2.2 Important Kernels

$$\mathcal{K} : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$$
$$(x, y) \mapsto \varphi^T(x)\varphi(y)$$

$$\mathcal{K}(x,y) = \exp(-\gamma\|x-y\|^2) \qquad \textit{RBF-kernel}$$
$$\mathcal{K}(x,y) = \tanh(\gamma xy) - b \qquad \textit{sigmoid}$$
$$\mathcal{K}(x,y) = (x^T y)^d$$
$$\mathcal{K}(x,y) = (x^T y + 1)^d$$

(RBF = radial-basis function)

8.2.3 Kernel engineering

if k_1, k_2 are valid kernels,$c > 0$, q polynomial, f function, $\phi : \mathcal{X} \to \mathbb{R}^m$, A symmetric and pos. sem. def. Then the following are also valid kernels:

$ck_1(x,x')$	$k_1(x,x')+k_2(x,x')$	$k_a(x_a,x'_a)k_b(x_b,x'_b)$
$f(x)k_1(x,x')f(x')$	$k_1(x,x')k_2(x,x')$	$k_a(x_a,x'_a) +$
$q(k_1(x,x')$	$k_3(\phi(x),\phi(x'))$	$k_b(x_b,x'_b)$
$\exp(k_1(x.x')$	$x^T Ax'$	

8.2.4 Mercer's Theorem:

$\mathcal{K} : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$
If $\mathcal{K}(x,y) = \mathcal{K}(y,x)$ and $\int\int f(x)\mathcal{K}(x,y)f(y)dxdy \geq 0$ for any $f \in L^2$, then $\mathcal{K}$ is a Kernel.

8.3 Structural SVMs

Structural SVMs can be seen as a generalization of SVM, where we can predict general structured output

8.3.1 Multi-Class Classification

One-vs-rest classification: $c(x) = \arg\max_y \text{score}_y(x)$
We can't do this for structural SVMs: there are too many classes and no inter-class learning.

8.3.2 Joint feature maps:

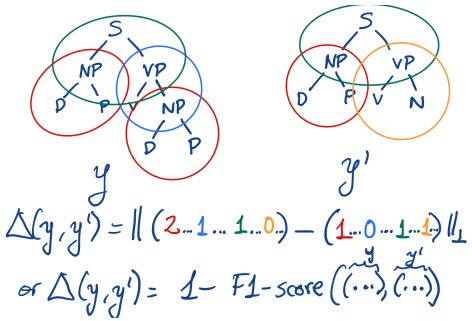
$$\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^m \times \mathbb{Z}^n$$
$$\Psi(\text{"the dog chases the cat"}, \langle \text{tree structure} \rangle) \mapsto w$$

- One weight vector for all classes
- $\mathbf{w}^T \Psi(x, y) =$ *compatibility score between* x *and* y
- $c(x) = \arg\max_y \mathbf{w}^T \Psi(x, y)$

SVM formulation: We do not want to require the same "gap" for all pairs of classes (make a difference between small and large mistakes: $\Delta(y, y') \in \mathbb{R}^+$)

$$\min_w \frac{1}{2} \|w\|^2$$
$$\text{s.t. } w^T \Psi(x_i, y_i) \geq \Delta(y_i, y') + w^T \Psi(x_i, y')$$
$$\forall y' \neq y_i, i \leq n$$

Loss function:



Soft-margin Formulation

$$\min_{w,\xi} \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i \leq n} \xi_i$$
$$\text{s.t. } w^T \Psi(x_i, y_i) \geq \Delta(y_i, y') + w^T \Psi(x_i, y') - \xi_i$$
$$\xi_i \geq 0, \forall y' \neq y_i, i \leq n$$

Why the n?
Theorem: If $w^*.\xi^*$ are optimal, then the empirical risk of w^* w.r.t Δ is

$$\mathbb{E}_{x,y}[\Delta(Y, c_w(x))] = \frac{1}{n} \sum_{i \leq n} \Delta(y_i, c_w(x_i))$$
$$\leq \frac{1}{n} \sum_{i \leq n} \xi_i$$

Proof on slide 42

The conditions can also be written as:

$$\text{s.t. } w^T \Psi(x_i, y_i) \geq 1 + + w^T \Psi(x_i, y') - \frac{\xi_i}{\Delta(y_i, y')}$$
$$\xi_i \geq 0$$

This formulation is **invariant to rescaling of Δ**

8.3.3 Training Algorithm

Algorithm 5: SVM training algorithm

input: tolerance threshold $\epsilon > 0$
$\mathbf{w} \leftarrow 0, \xi \leftarrow 0, W \leftarrow \emptyset$
do
  for $i \leq n$ **do**
    $y' \leftarrow \arg\max_{y \neq y_i} \{\Delta(y_i, y) + w^T \Psi(x_i, y)\}$
    if $(w^T \Psi(x_i, y_i) \not\geq \Delta(y_i, y') + w^T(x_i, y') - \xi_i - \epsilon)$ **then**
      $W \leftarrow W \cup \{w^T \Psi(x_i, y_i) \geq \Delta(y_i, y') - \xi_i + w^T \Psi(x_i, y')\}$
    end
  end
  $w, \xi \leftarrow$ *solve* $\left(\min_w \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_i \xi_i \text{ s.t. } W\right)$
until W *does not change*;

8.3.4 Prediction:

$$c(x) = \arg\max_y w^T \Psi(x, y)$$

8.4 Advantages and Disadvantages

Advantages	Disadvantages
- Works well with infinitely dimensional representations	- requires careful model selection
- Adapted to structured classification	- Requires feature Engineering
- Formulated as QP, for which efficient procedures are available	- Use of kernels make training algos succeptible to curse of dim

Examples on page 51 - 64 (page ranking, diseases, ...)

9 Ensemble Methods

Bagging vs. Boosting

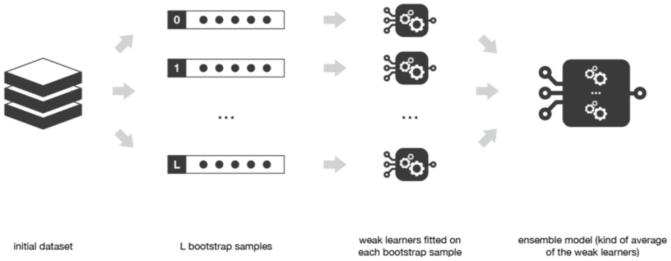
Bagging	Boosting
trains learners in parallel and combines them using deterministic averaging.	trains learners sequentially in an adaptive way and combines them using a deterministic strategy.
less variance than the components	less biased than the components

Combination of multiple weak learners to get a stronger learner. (Wisdom of crowds)

9.1 Bagging

1. Bootstrap sets: Draw M bootstrap sets
2. Train M base models $b^{(1)}, ..., b^{(M)}$
3. Aggregate

$$\bar{b} = \begin{cases} \frac{1}{M} \sum_{t \leq M} b^{(t)}(x) & \text{regression} \\ \textit{majority}\left(b^{(t)}(x)\right)_{t \leq M} & \text{classification} \end{cases}$$

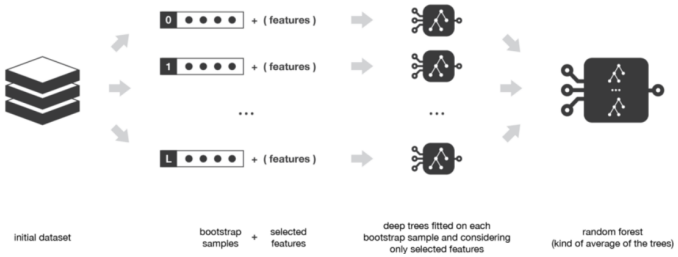


Theorem: For $x \in \mathcal{X}$ if $range(y) < \infty$, then there is a sufficiently large M s.t. $\mathbb{E}[(y - \bar{b}^{(M)}(x))^2] \leq \mathbb{E}[(y - b(x))^2]$ for some base model b.
Proof in Slides, p.20-21
Properties of the base models

- Diversity
- Independence: Bootstrap sets should be independent, but **they are not** (but correlation is small).

9.1.1 Random forests

The Random forest method is a bagging method with trees as weak learners. Each tree is fitted on a bootstrap sample considering only a ***subset of variables*** *randomly chosen. This helps* ***reduce the correlation*** *between trained base trees in the ensemble.*



9.2 Boosting

Key: Learning from previous mistakes.

Fit models iteratively, such that the training of the model at a given step depends on the models fitted in previous steps. Each model gives higher weight to the observations that were handled badly in the previous steps.

9.2.1 Ada Boost (Adaptive Boosting)

Learn from previous mistakes by increasing the weight of misclassified datapoints. ***Outliers*** *will get very high weights and can be detected by that.*

- Loss function: 0-1 Loss
- Base models (stumps)
- place high weights on samples that are very hard to classify

Let $\mathcal{L}^{(w)} = \sum_{i \leq n} w_i * \mathbb{1}\{b(x_i) \neq y_i\}$ (the 0-1 loss)

Algorithm 6: AdaBoost Algorithm

```
// Initialize
```
$\flat^{(0)} \leftarrow 0$

$w_i \leftarrow 1/n$ for $i \leq n$

for *t=1..M* **do**

```
    // Train
```
$b^{(t)} \leftarrow \arg\min_b \mathcal{L}^{(w)}(b)$

```
    // Evaluate
```
$err_t \leftarrow \mathcal{L}^{(w)}(b^{(t)})$

```
    // Reweigh
```
$$w_i \leftarrow \begin{cases} \tilde{\alpha}_t w_i & \text{if } b^{(t)}(x_i) \neq y_i \text{ for } i \leq n, \tilde{\alpha}_t = \frac{1}{err_t} - 1 \\ w_i & \text{otherwise} \end{cases}$$

```
    // Add to set
```
$\flat \leftarrow \flat^{(t-1)} + \tilde{\alpha}_t b^{(t)}$

normalize$(w_1, ..., w_n)$

end

AdaBoost for Classification:

1. Train $\alpha_1, b^{(1)}, ..., \alpha_M, b^{(M)}$

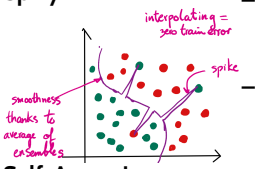
$$\alpha_1, b^{(1)}, ..., \alpha_M, b^{(M)}$$
$$\alpha_t \in [0, \infty)$$
$$b^{(t)} : \mathcal{X} \to \{-1, +1\}$$

2. predict $\flat^{(M)} = sgn(\sum_{t \leq M} \alpha_t b^{(t)}(x))$

Why is AdaBoost so successful? *Theoretical results: Slides p. 57-*

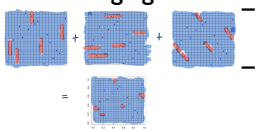
- Friedmann: AdaBoost is equal to forward additive stepwise modeling (FSAM) with exponential loss
- Boosting trains max-margin classifiers
- Wyner et al: AdaBoost and random forest train spiky, interpolating and self-averaging models.

Spiky:



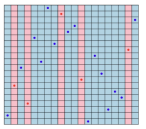
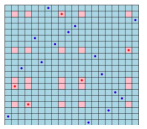
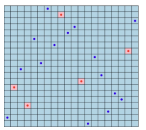
– Mostly smooth with Sharp localized changes to interpolate noisy examples
– This prevents influence from noise and allows fitting of complex signals.

Self-Averaging:



– Trained model is an average of diverse models
– Averaging spiky interpolation helps localise noisy effects

AdaBoost with complex base classifiers trains spiky interpolation classifiers.

base classif.	Vertical Stumps	Stumps	Full Trees
Trained Ensemble			
gen. error	high	medium	low
Interp.	✓	✓	✓
spiky	×	∼	✓

9.2.2 Gradient Boosting

Similarly to AdaBoost, gradient Boosting is a ***method to greedily approximate*** f *using gradient descent based on the additive form. Gradient Boosting learns directly from the residual error instead of updating the weights.*

$$f_M(x) = \sum_{i=1}^{M} \beta_i h_i(x), \textit{where } h_i \textit{ are the weak learners.}$$

Algorithm 7: Gradient Boosting algorithm

$\hat{f}_0(\mathbf{x}) = \text{argmin}_h \sum_{i=1}^n L(y_i, h(\mathbf{x}_i))$; `// Init`

for *t=1..M* **do**

```
    // Compute the negative gradient :
```
$-g_m(\mathbf{x}_i) = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f=\hat{f}_{m-1}(\mathbf{x}_i)}, i = 1...n$

```
    // Fit function h_m to negative gradient by
       least squares:
```
$h_m = \text{argmin}_h \sum_{i=1}^n (-g_m(\mathbf{x}_i - h(\mathbf{x}_i))^2$

```
    // Find β_m to minimize the loss
```
$\beta_m = \text{argmin}_\beta \sum_{i=1}^n L(y_i, \hat{f}_{m-1}(\mathbf{x}_i) - h(\mathbf{x}_i))^2$

```
    // Update f̂
```
$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1} + \beta_m h_m(\mathbf{x})$

end

Differences AdaBoost / Gradient Boosting:

- AdaBoost learns from mistake by increasing the weight of misclassified data-points.
- Gradient Boosting learns from the residual error (mistake) directly instead of updating weights.

9.2.3 Forward Stagewise Additive Modeling (Exercise 6)

Method to approximately compute a classifier of the form $c(x) = sgn(\sum_t \alpha_t b^{(t)})$ that approximately minimizes the empirical loss $\sum_{i \leq n} L(y_i, c(x_i))$.

Algorithm 8: Forward stagewise additive modeling

input :
- $\{(x_1, y_1), ..., (x_n, y_n)\} \subseteq \mathbb{R}^D \times \{-1, 1\}$
- $L : \{1, -1\} \times \{1, -1\} \to \mathbb{R}$
- $M \in \mathbb{N}$

output: $\hat{c} \approx \text{argmin}_c \sum_{i=1}^n L(y_i, c(x_i))$

$f_0(x) \leftarrow 0 \forall x \in \mathbb{R}^D$
for $t = 1$ *to* M **do**
  $(\alpha_t, b^{(t)}) \leftarrow \text{argmin}_{\substack{\alpha>0 \\ b \in \mathcal{H}}} \sum_{i=1}^n L(y_i, \alpha b(x_i) + f_{t-1}(x_i))$
  $f_t(x) \leftarrow \alpha_t b^{(t)}(x) + f_{t-1}(x) \forall x \in \mathbb{R}^D$
end
return $\hat{c} \leftarrow sgn(f_M(x)) \forall x \in \mathbb{R}^D$

⇒ **Exercise 1: AdaBoost is equivalent to forward stagewise additive modeling**

10 Deep Learning

10.1 Feed-Forward Neural Networks

Function that applies linear transformation and activation functions.
- (L)inear functions $L : x \mapsto Wx + b$
- (α)ctivation function: $\alpha : x \mapsto (\alpha(x_1), ..., \alpha(x_n))$

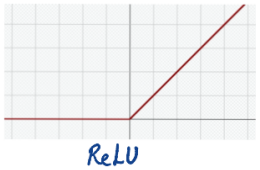
$$\begin{aligned} NN(x) &= \alpha^{(d)}(L^{(d)}(...\alpha^{(2)}(L^{(2)}((\alpha^{(1)}(L^{(1)}))))) \\ &= \alpha^{(d)} \circ L^{(d)} \ldots \alpha^{(2)} \circ L^{(2)} \circ \alpha^{(1)} \circ L^{(1)}(x) \end{aligned}$$

10.1.1 Activation Functions

Activation functions make the NN function non-linear.

ReLu:

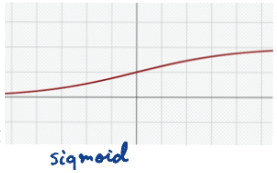
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



Danger of vanishing gradient on the left.

Sigmoid:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

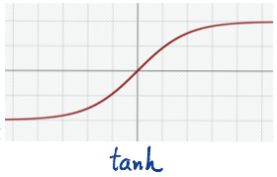


Danger of vanishing gradient on both sides.

$$\sigma'(x) = \begin{cases} \sigma(x)(1 - \sigma(x)) & \text{if } dim(x) = 1 \\ \sigma(x) \circ (1 - \sigma(x)) & \text{if } dim(x) > 1 \end{cases}$$

Tanh:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$



Danger of vanishing gradient on both sides

Theorem Cybenko:
If $f : [0, 1]^n \to \mathbb{R}$ is continuous, $\epsilon > 0$, then there is a neural net s.t.
1. $NN(x) = \sum_{i \leq m} \alpha_i \sigma(w_i^T x + b_i)$
2. $\max_x |f(x) - NN(x)| < \epsilon$

Pooling Layers: average the results from previous layer.
Output Layers:
- Binary classifier: One output, $\sigma(z)$ (sigmoid)
- Multi-Class: Soft-max layer to get probabilities from scores: *"Winner takes it all"*

$$y_i = \frac{\exp)\beta z_i)}{\sum_{j \leq 3} \exp(\beta z_j) t}$$

Training Neural Networks

$$\min_\theta \underbrace{\sum_{i \leq n} \mathcal{L}(y_i, NN_\theta(x_i))}_{\mathcal{L}(\theta)}, \quad \theta = (w^{(1)}; b^{(1)}; w^{(2)}; b^{(2)}, ...),$$

where θ are the parameters of the neural network.This function is analytically intractable, but easy to differentiate.

Forward propagation:

$$a^{(l)} = \sigma(W a^{(l-1)} + b)$$

Back-propagation:
The only purpose of back-propagation is to compute the gradients $\frac{\partial NN}{\partial w^l}$ *(not optimize them, that is what optimizers are for).*
- Start at the output and propagate backwards, updating weights and biases for each layer.
- For wach layer, back propagate the weights and biases using back-propagation algorithm:

$$\frac{\partial C}{\partial w^l} = \frac{\partial C}{\partial z^L} \prod_{i=L \ldots l+1} \left(\frac{\partial z^i}{\partial a^{i-1}} \frac{\partial a^{i-1}}{\partial z^{i-1}} \right) \frac{\partial z^l}{\partial w^l},$$

where L last layer, z value before activation function, a value after activation function.

Backpropagation in Multi-Layer Perceptron (Assignment 7, Ex. 1)

To compute the values for back-propagation, we use the chain rule.

$$\frac{\partial C}{\partial w_r^{(l)}} = \frac{\partial C}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w_r^{(l)}}$$
$$\frac{\partial C}{\partial b^{(l)}} = \frac{C}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial b^{(l)}}$$

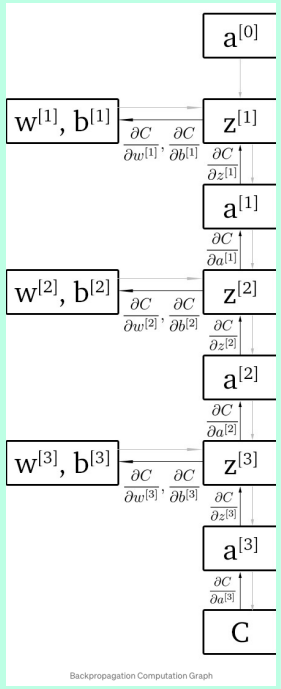
For example, we got:

$$\frac{\partial C}{\partial z^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$$

Computing and substituting all of these values will result in the iteration step of stochastic gradient descent to update the weights matrix $w_t^{(l)}$ and the biase vectors $b_t^{(l)}$ as

$$w_t^{(l)} = w_{t-1}^{(l)} - \eta \frac{\partial C}{\partial w^{(l)}}$$
$$b_t^{(l)} = b_{t-1}^{(l)} - \eta \frac{\partial C}{\partial b^{(l)}}$$

where $w_t^{(l)}$ is the t-th row of the weight matrix $w^{(l)}$ and C is the cost of the last layers output (i.e. $\mathcal{L}$)



10.1.2 Gradient Descent

Algorithm 9: General Form of Gradient Descent

$k \leftarrow 0$
$w_r^{(k)} \leftarrow \text{Unif}([-\sqrt{\frac{1}{n}}, \sqrt{\frac{1}{n}}])$ for $l \leq d, r \leq$#rows $W^{(l)}$
do
  $w^{(k+1)} \leftarrow w^{(k)} - \eta(k)\nabla_{w_r^{(l)}}\mathcal{L}(\theta)$ for $l \leq d, \forall r$
  $k \leftarrow k+1$
until *difference between test-cost and train-cost starts increasing*;

Algorithm 10: Gradient Descent

$k \leftarrow 0$
$w_r^{(k)} \leftarrow \text{Unif}([-\sqrt{\frac{1}{n}}, \sqrt{\frac{1}{n}}])$ for $l \leq d, r \leq$#rows $W^{(l)}$
repeat
  do fwd and back propagation to compute $\frac{\partial NN}{\partial w_r^{(l)}}|_{\theta,x_i}$ for each $l, r, i \leq n$
  $\frac{\partial \mathcal{L}}{w_r^{(l)}} = \sum_{i \leq n} \frac{\partial \mathcal{L}(y_i, NN_\theta(x_i))}{\partial w_r^{(l)}}$
  $w_r^{(l)} \leftarrow w_r^{(l)} - \eta(k)\frac{\partial \mathcal{L}}{\partial w_r^{(l)}}$
until ...;

Algorithm 11: Mini-Batch (stochastic) Gradient Descent

$k \leftarrow 0$
$w_r^{(k)} \leftarrow \text{Unif}([-\sqrt{\frac{1}{n}}, \sqrt{\frac{1}{n}}])$ for $l \leq d, r \leq$#rows $W^{(l)}$
repeat
  do fwd and back propagation to compute $\frac{\partial NN}{\partial w_r^{(l)}}|_{\theta,x_i}$ for each $l, r, i \in S$, where S is a sample from $\{1...n\}$
  $\frac{\partial \mathcal{L}}{w_r^{(l)}} = \sum_{i \leq n} \frac{\partial \mathcal{L}(y_i, NN_\theta(x_i))}{\partial w_r^{(l)}} \approx \sum_{i \in S} \frac{\partial \mathcal{L}(y_i, NN_\theta(x_i))}{\partial w_r^{(l)}}$
  $w_r^{(l)} \leftarrow w_r^{(l)} - \eta(k)\frac{\partial \mathcal{L}}{\partial w_r^{(l)}}$
until ...;

SGD as stochastic optimization

$$\min_\theta \sum_{i \leq n} \mathcal{L}(y_i, NN_\theta(x_i))$$

$$\begin{aligned} \iff 0 &= \sum_{i \leq n} \nabla_\theta \mathcal{L}(y_i, NN_\theta(x_i)) \\ &= \hat{\mathbb{E}}_{x,y}[\nabla_\theta \mathcal{L}(Y, NN_\theta(x)] \\ &\approx \mathbb{E}_{x,y}[\nabla_\theta \mathcal{L}(Y, NN_\theta(x)] \\ &= \mathbb{E}_{x,y}[\nabla_\theta f(X, Y; \theta)] \end{aligned}$$

$$\Leftarrow \mathbb{E}_{x,y}[\nabla_\theta f(X, Y; \theta)] = 0$$

Comparison Gradient Descent

Batch GD	Mini-batch / stochastic GD
• More precise gradient • Larger generalization error	• Can handle large training sets • Faster improvements • Escapes local minimum

10.1.3 Robbins-Monro algorithm

Methodology for solving a root finding problem (of the expected value $\mathbb{E}$). Provides convergence guarantee for SGD (see below).

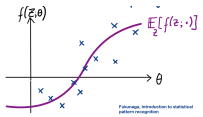
Algorithm 12: Robbin-Monro algorithm

input : Learning rate function η
  sample $z_1, z_2, ...$
output: θ s.t. $\mathbb{E}_Z[f(Z;\theta)] = 0$
$\theta^{(0)} \leftarrow \$$
for $k = 0...$ **do**
  $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta(k) f(z_k; \theta^{(k-1)})$
end

Convergence: If $\mathbb{E}_Z[f(Z;\theta)]$ satisfies some regularity conditions, $\eta(k) \geq 0, \sum_k \eta(k) = \infty, \sum_k \eta^2(k) < \infty$, then

$$\mathbb{E}[(\theta^{(k)} - \theta^*)^2] \xrightarrow{k\to\infty} 0$$
$$P[\theta^{(k)} = \theta^*)] \xrightarrow{k\to\infty} 1$$

Regularity Conditions:
$\mathbb{E}_Z[f(Z;\theta^*)] < \mathbb{E}_Z[f(Z;\theta)], \forall \theta > \theta^*$
$\mathbb{E}_Z[f(Z;\theta^*)] > \mathbb{E}_Z[f(Z;\theta)], \forall \theta < \theta^*$
There are other formulations (see Slides p.32)



10.2 Regularization in DNNs (Exercise 6.3) Problems + Solutions

Dying ReLu Problem: Once a value becomes negative, it will always output 0. Since the gradient for neg. values is 0 in ReLu, it will not become active again.

⇒ *ELU or leaky ReLU activation function that has a non-zero gradient for arguments smaller than zero.*

$$ELU_\alpha = \begin{cases} \alpha(\exp(z) - 1) & z < 0 \\ z & z \geq 0 \end{cases}$$

with hyperparamter α.

Slow learning of a DNN with sigmoid function using Gradient Descent algorithm and large initialized weights:) The sigmoid function saturates quickly and hence, for large values of z the gradient updates are very small, slowing down the learning process.

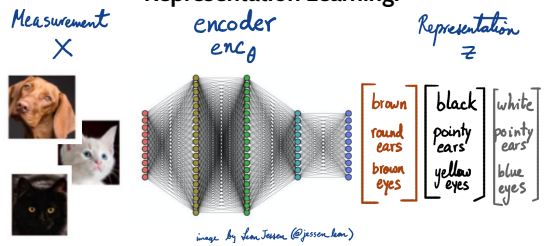
⇒ *Use random initialization weights*
⇒ *Apply different activation function(such as ReLU or its variations), that is nonsaturating for positive values*

Applying dropout layers to avoid overfitting: This will increase the training time, since parameter updates become noisy due to omitting units in each training iteration.
Inference time is not influenced, since dropout is only applied during training

10.3 Variational Autoencoders (Notes 11)

Learn meaningful representations without supervision.

Representation Learning:



10.3.1 Objective

Derive parametrized function enc_θ that maps measurements in $\mathcal{X}$ to probability distributions over a representation space $\mathcal{Z}$

$$enc_\theta : x \in \mathcal{X} \mapsto p_\theta(\cdot|x) \text{ over } \mathcal{Z}$$

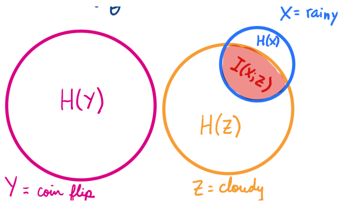
Requirements for Autoencoder:

- Informative: Given representation, it should be easy to guess the measurement
- Disentangled: Every component in the representation is associated with a distinguished feature
- Robust: Noisy perturbations in the measurement should not substantially affect the representation and vice versa

10.3.2 The info max principle (Linsker 1988)

$Z = enc_\theta(X)$ maximizes $I(X;Z)$
Information $(X;Z)$ is defined as

$$I(X;Z) = \mathbb{E}_{X,Z}\left[\log\left(\frac{p(X,Z)}{p(X)p(Z)}\right)\right]$$



$$H(x) = \mathbb{E}_x\left[-\log p(x)\right]]$$

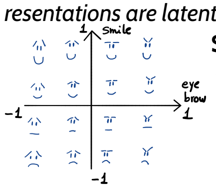
Formalization
Let X and Z be a measurement and a representation space. Let $F = \{enc_\theta : \theta \in \Theta\}$ be a parametric family of functions with enc_θ mapping X to distributions over Z.
The encoder function enc_{θ^*} is defined by

$$\theta^* = \arg\max_\theta I(X;Z)$$

where Z is a random variable with distribution $enc_\theta(X)$
This is informative, but not disentangled or robust. If enc_θ is complex enough then enc_θ maximizes $I(X;z)$ becoming an injective function from X to Z.

10.3.3 Variational Autoencoders (VAE)

A VAE fits a generative probabilistic model to the dataset, where the representations are latent variables.



Smileys Example
- Disentangled: Z_0, Z_1 along axes; in either direction only the smile OR the eye brows change
- Diagonal lines would be entangled.

Blogpost Link
Autoencoders:
- General Idea: Set an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimization process.
- At each iteration, compare encoded-decoded to initial data.

Variational Autoencoders: *Pick a family of distributions over the latent variables with its own variational parameters $q(z_{1:m}|\mathcal{V})$. Then find the parameters that makes q close to the desired posterior and use q with the fitted parameters as a proxy for the posterior.* (Princeton Advanced Methods in Probabilistic Modeling).

- General Idea: autoencoder whose training is regularized to avoid overfitting and ensure that the latent space has good properties that enable generative process
- Autoencoders train encoder-decoder, but there is no way **generating new data**
- instead of encoding an input as a single point, encode it as a distribution over the latent space.
- In order to use an autoencoder for content generation, we have to make sure the distribution is regular (e.g. by using regularization terms).

Variational Inference: The process of finding an approximate posterior.

1. Define prior and calculate likelihood (decoder)
2. Find approximate posterior (encoder)

Variational Autoencoders

1. Define
   - $\{p'_\theta(z) : \theta' \in \Theta'\}$ pram. family of priors
   - $\{p_\theta(x|z) : \theta \in \Theta\}$ pram. family of likelihoods
   - $\{q_\phi(z) : \phi \in \Phi'\}$ pram. f. of approx. posteriors
2. The variational auto encoder is trained by solving

$$\max_{\theta',\theta,\phi} \log p_{\theta',\theta,\phi}(x_1,...,x_n), \{x_1,...,x_n\} \textit{ training set}$$

3. The representation Z of a measurement x is a random variable with pdf $q_\phi(z|x)$.
   The measurement X encoded by a representation Z is a random variable with pdf $p_\theta(x|z)$

How to train a variational autoencoder?

$$\arg\max_{\theta',\theta,\phi} \sum_{i\leq n} \log p_{\theta',\theta}(x_i)$$

$$\begin{aligned}
\log_{\theta',\theta} p(x_i) &= \mathbb{E}_{Z\sim q_\phi(\cdot|x_i)}\left[\log p_{\theta',\theta}(x_i)\right] \\
&= \mathbb{E}_{Z\sim q_\phi(\cdot|x_i)}\left[\log\left(\frac{p_{\theta',\theta}(x_i,Z)}{p_{\theta',\theta}(Z|x_i)}\frac{q_\phi(Z|x_i)}{q_\phi(Z|x_i)}\right)\right] \\
&= \underbrace{\mathbb{E}_{Z\sim q_\phi(\cdot|x_i)}\left[\log\left(\frac{p_{\theta',\theta}(x_i,Z)}{q_\phi(Z|x_i)}\right)\right]}_{elbo_{\theta',\theta,\phi}(x_i)} \\
&\quad + \underbrace{E_{Z\sim q_\phi(\cdot|x_i)}\left[\log\left(\frac{q_\phi(Z|x_i)}{p_{\theta',\theta}(Z|x_i)}\right)\right]}_{KL\left(q_\phi(\cdot|x_i)\|p_{\theta',\theta}(\cdot|x_i)\right)\geq 0} \\
\implies \log_{\theta',\theta} p(x_i) &\geq elbo_{\theta',\theta,\phi}(x_i) \\
&= \underbrace{\mathbb{E}_{Z\sim q_\phi(\cdot|x_i)}\left[\log\left(p_\theta(x_i|Z)\right)\right]}_{Infomax!} \\
&\quad + \underbrace{\underbrace{E_{Z\sim q_\phi(\cdot|x_i)}\left[\log\left(\frac{p_{\theta'}(Z)}{q_\phi(Z|x_i)}\right)\right]}_{-KL\left(q_\phi(\cdot|x_i)\|p_{\theta',\theta}(\cdot)\right)}}_{Regularization\,Term}
\end{aligned}$$

This is informative, disentangled and robust by the choice of $p_\theta(\cdot|Z)$ and $q_\phi(\cdot|x)$.

10.3.4 Kullback-Leibler Divergence KL

Used to measure the closeness of 2 distributions. Usually one of them is the approximation for the other. In our case: q and p.

$$KL(q\|p) = \mathbb{E}_q\left[\log\frac{q(Z)}{p(Z|x)}\right]$$

10.3.5 The evidence lower bound $elbo$

We can't minimize the KL divergence exactly. But we can minimize a function that is equal to it up to a constant: The $elbo$-function
The elbo is optimized using
- Gradient Descent
- Monte-Carlo Sampling
- Analytical Reparametrization tricks

10.3.6 Denoising an Autoencoder

Blank out parts of the input image during training to create a more robust encoder.

10.3.7 Modeling Invariances

If we want the autoencoder to be robust against rotations or scaling:
⇒ Augmentation of training set
⇒ Special preprocessing of images
⇒ Implementation of invariances into multi-layer perceptrons (CNN)

10.4 Deep Generative Modeling

We want to use generative modeling with DNN.

10.4.1 Generative Adversarial Networks (GANs)

*Approach to generative modeling using deep learning methods (e.g. convolutional neural networks). In contrast to VAE, we do not model the density, but instead directly define a **sampler**.*
(Not covered in exam)

10.4.2 VAEs vs. GANs

VAE	GAN
Pros: • Principled approach • Allows inference of the encoder which might be useful for other tasks	**Pros:** • Beautiful • State of the Art
Cons: • Maximizes the lower bound of the likelihood (quality is not as good, blurry)	**Cons:** • Tricky, unstable training • mode collapse • cannot solve inference queries as $p(x)$

11 Clustering

Given a training set, group the data into a few clusters.
Data Representation:
- Vector data: n vecotrs in $\mathbb{R}^d$
- Histogram data: n histograms in $\mathbb{R}^d$
- Proximity data: $n \times n$ pairwise proximity matrix. Much harder problem (structure hidden in n^2 pairwise relations

11.1 k-means vs. EM

k-means	EM
Objective: minimize inter-cluster variance	
Hard assignment	Soft assignment
works well for homog. clusters (assumes spherical clusters, with equal covariance matrices)	Can constrain algorithm to get different shapes of cov-matrices (not limited to spherical shapes)

Neither of the algorithms can detect outliers! This would need a preprocessing step. In the presence of outliers, EM is more sensitive, since there are no constraints on the covariance matrix.

11.2 k-means

11.2.1 k-means Problem

Group data into k groups.
- Given: d-dimensional sample vectors $\mathbf{X}$
- Assignment function

$$c : \mathbb{R}^d \to \{1, ..., k\}$$
$$\mathbf{x} \mapsto c(\mathbf{x})$$

- Prototypes $\mu_c \in \mathcal{Y} \subset \mathbb{R}^d$
- **Problem:** find $c(.)$ and $\mathcal{Y}$ that minimize

$$\mathcal{R}^{km}(c, \mathcal{Y}) = \sum_{x \in X} \left\| x - \mu_{c(x)} \right\|^2$$

11.2.2 k-means algorithm

Algorithm 13: k-means algorithm

input: $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$
init : $\mu_c = \mathbf{x}_c$ for $1 \leq c \leq k$
1 **repeat**
2 Keep prototypes $\mathcal{Y}$ fixed and assign sample vector $\mathbf{x}$ to neareast prototype

$$c(\mathbf{x}) \in \underset{c \in \{1,...,k\}}{\operatorname{argmin}} \left\| \mathbf{x} - \mu_c \right\|^2$$

3 Keep assignments $c(\mathbf{x})$ fixed and estimate prototypes

$$\mu_\alpha = \frac{1}{n_\alpha} \sum_{\mathbf{x}: c(\mathbf{x}) = \alpha} \mathbf{x}, \text{ with } n_\alpha = \#\{\mathbf{x} : c(\mathbf{x}) = \alpha\}$$

4 **until** *Changes of $c(\mathbf{x}), \mathcal{Y}$ vanish*;
5 **return** $c(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$ *and the prototypes* $\mathcal{Y}$

11.3 Mixture Model

Data are assumed to be distributed according to a density. When multiple sources are considered as potential causes for an observed example, this is a mixture model with the density for a feature vector $\mathbf{x}$:

$$p(\mathbf{x}|\pi_1, ..., \pi_k, \theta_1, ..., \theta_k) = \sum_{c \leq k} \pi_c p(\mathbf{x}|\theta_c)$$

the mixture weight π_c is the prior probability that a sample is generated by the mixture component c with parameters θ_c (i.e. $\pi_c = p(c(\mathbf{x}) = c, \theta_c)$.

11.3.1 Gaussian Mixtures

$\mathbf{x}$ *was drawn from one of k gaussians, depending on the class .*
- Parameters $\theta = (\mu, \Sigma)$

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{\sqrt{2\pi}^d} \frac{1}{\sqrt{\|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right)$$

- Estimate $\hat{\theta}$ that maximizes the likelihood of sample feature vectors $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$

$$p(\mathcal{X}|\pi_1, ..., \pi_k, \theta_1, ..., \theta_k) = \prod_{x \in \mathcal{X}} \sum_{c \leq k} \pi_c p(\mathbf{x}|\theta_c)$$

- Log-Likelihood

$$L(\mathcal{X}|\pi_1, ..., \pi_k, \theta_1, ..., \theta_k) = \sum_{x \in \mathcal{X}} \log \sum_{c \leq k} \pi_c p(\mathbf{x}|\theta_c)$$

Direct optimization of the log-likelihood is intractable due to sum within the logarithm (i.e. there is **no closed form solution**).
EM Mixture models solve this by introducing latent indicator variables for mode assignments and maximizing the joint likelihood of the observable and latent variables.

11.4 Expectation - Maximization algorithm

Principle:
1. Calculate $Q(\theta; \theta^{(j)}) = \mathbb{E}_{\mathcal{X}_L}\left[L(\mathcal{X}, \mathcal{X}_L|\theta) \middle| \mathcal{X}, \theta^{(j)}\right]$
   Function of parameters given the parameters in the step before.
2. Estimate new parameters by maximizing the log-likelihood of Q:

$$\theta^{(j+1)} \in \underset{\theta}{\operatorname{argmax}}\, Q(\theta; \theta^{(j)})$$

3. Repeat until convergence

Derivations: *(Full derivations on p. 11 - 16, Lecture 12 - clustering)*

$$M_{\mathbf{x}c} = \begin{cases} 1 & \textit{Mode } c \textit{ has generated vector } \mathbf{x} \\ 0 & \textit{Mode } c \textit{ has not generated vector } \mathbf{x} \end{cases}$$

This gives

$$P(\mathcal{X}, M|\theta) = \prod_{x \in \mathcal{X}} \prod_{c=1}^{k} (\pi_c P(\mathbf{x}|\theta_c))^{M_{\mathbf{x}c}}$$

$$L(\mathcal{X}, M|\theta) = \log P(\mathcal{X}, M|\theta) = \sum_{x \in \mathcal{X}} \sum_{c=1}^{k} M_{\mathbf{x}c} (\pi_c P(\mathbf{x}|\theta_c))$$

We define $\gamma_{\mathbf{x}c} := \mathbb{E}_M\left[M_{\mathbf{x}c}\middle|\mathcal{X}, \theta^{(j)}\right]$

Algorithm 14: Expectation-Maximization Algorithm

1 **repeat**

2 **E-Step: For all** c

$$\gamma_{\mathbf{x}c} = \frac{P(\mathbf{x}|c, \theta^{(j)}P(c|\theta^{(j)})}{P(\mathbf{x}|\theta^{(j)})}$$

3 **M-Step: For all** c

$$\mu_c^{(j+1)} = \frac{\sum_{\mathbf{x}\in\mathcal{X}} \gamma_{\mathbf{x}c}\mathbf{x}}{\sum_{\mathbf{x}\in\mathcal{X}} \gamma_{\mathbf{x}c}}$$

$$(\sigma_c^2)^{(j+1)} = \frac{\sum_{\mathbf{x}\in\mathcal{X}} \gamma_{\mathbf{x}c}(\mathbf{x}-\mu_c)^2}{\sum_{\mathbf{x}\in\mathcal{X}} \gamma_{\mathbf{x}c}}$$

$$\pi_c^{(j+1)} = \frac{1}{|\mathcal{X}|}\sum_{\mathbf{x}\in\mathcal{X}} \gamma_{\mathbf{x}c}$$

4 **until** *Changes are small enough*;

11.5 Problems with Mixtures of Gaussians

- Computation time: Time to estimate the model parameters may be large
- Number of free parameters: Scales with $O(d^2)$ (d data dimension)

⇒ Param. estimation problematic if dimension of features space is high and number of samples is slow

⇒ **Solution:** Use only hard assignments $c(x) \in \{1, ..., k\}$ as in k-means clustering.

12 Non-Parametric Bayesian Methods

- **Beta Distribution**

$$Beta(x|a, b) = \frac{1}{B(a, b)} \cdot x^{a-1}(1-x)^{b-1}, x \in [0, 1]; a, b > 0$$

where

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

$$\Gamma(a) = \int_0^\infty e^{-x}x^{a-1}dx$$

Probability of a Bernoulli process after observing $a-1$ successes and $b-1$ failures

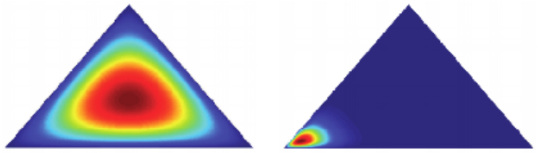
- **Dirichlet Distribution:** Multivariate generalization of the beta distribution.

Given $\mathbf{x} = x_1, ..., x_n, \alpha = \alpha_1, ..., \alpha_n$ *where* $x_i \in [0, 1], \alpha_i > 0$

$$Dir(\mathbf{x}|\alpha) = \frac{1}{B(\alpha)} \cdot \prod_{k=1}^{n} x_k^{\alpha_k - 1}$$

where $B(\alpha)$ is the multivariate generalization of the beta function:

$$B(\alpha) = \frac{\prod_{k=1}^{n}\Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{n}\alpha_k)}$$



(a) $\boldsymbol{\alpha} = (2, 2, 2)$ (b) $\boldsymbol{\alpha} = (20, 2, 2)$

12.1 Finite and infinite mixtures

12.1.1 Finite Gaussian Mixture Model:

Fixed, finite number of clusters K

- Center of cluisters $\mu_k \sim \mathcal{N}(\mu_0, \sigma_0)$
- Prob. of the clusters (parameters) $\rho_{1...K} \sim Dir(\alpha_{1...K})$
- Assignments to clusters $z_i \sim Categorical(\rho_{1...K})$
- Coordinates of data points $x_i \sim \mathcal{N}(\mu_{z_i}, \sigma_{z_i})$

12.1.2 Selecting K:

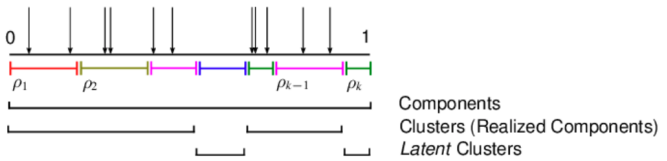
Issues:

- There might be multiple possible ways to cluster data (hence multiple K that would fit)

Adaptation:

- Number of clusters might be unknown in advance (movie genres, topics of documents, image segmentation, streaming data, ...)
- Naive solution: Select K, cluster with EM, evaluate the result and iterate

What if we just select a large enough K?



Latent clusters: *For a finite number of drawings N, we do not have to realize all K clusters. All components will be realized with probability 1, but only when $N \to \infty$*

- Select a large K, only realize some of them and get more when needed
- This only solves the problem partially: How large should this K be? Our belief in K could change as we observe more data-points. We still have issues with streaming/growing data.
- **Solution: Select** $k = \infty$

12.1.3 Infinite mixture models (Selecting $K = \infty$)

- With $K = \infty$, we have nonparametric Bayesian methods (nonparametric means infinitely many parameters) → we can keep drawing new parameters
- Cannot draw infinite points from Dir

12.2 Dirichlet Process (DP) and stick breaking

We can **sample from a DP using either Stick-Breaking or the Chinese Restaurant Process**

12.2.1 Dirichlet Proces

$DP(\alpha, H)$ is a distribution over probability distributions on a space Θ.

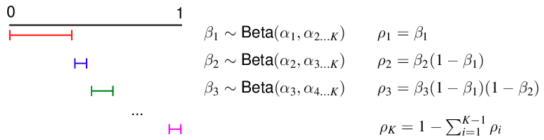
- $\alpha \in \mathbb{R}_{>0}$ is a concentration parameter
- H is the base measure on Θ (the space we want to draw parameters from)
- A Sample $G \sim DP(\alpha, H)$ is a function $G : \Theta \to \mathbb{R}_{\geq 0}$ s.t. $\int_\Theta G(\theta)d\theta = 1$

$DP(\alpha H)$ is characterized by the following property: *For every partition $(T_1, .., T_k)$ of Θ and $G \sim DP(\alpha, H)$ we have (for each tuple drawn from multivariate Dirichlet distribution):*

$$(G(T_1), ..., G(T_K)) \sim Dir(\alpha H(T_1), ..., \alpha H(T_k))$$

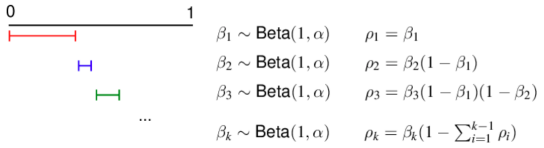
12.2.2 Stick-Breaking Process

Observation: sampling $(\rho_1, ..., \rho_K) \sim Dir(\alpha_1, ..., \alpha_K)$ is equivalent to sampling $\rho_1 \sim Beta(\alpha_1, ..., \alpha_K)$ and $(\rho_2, ..., \rho_K) \sim Dir(\alpha_2, ..., \alpha_K)$



We can keep doing this, but only if $\alpha = (\alpha_1, ..., \alpha_K)$ has finite length K.

Solution: fix α s.t. $\beta_i \sim Beta(1, \alpha) \forall i$



This is called the GEM (Griffiths-Engen-McCloskey) distribution:

$$\rho \sim GEM(\alpha), \rho. = \{p_k\}_{k=1}^{\infty}$$

Stick-Breaking Construction of the Dirichlet Process:
If $\rho \sim GEM(\alpha)$ and $\theta_k \sim H$ for $k = 1, 2, ...$, then

$$G(\theta) = \sum_{k=1}^{\infty} \rho_k \delta_{\theta_k}(\theta)$$

is a sample from $DP(\alpha, H)$.
If we repeatedly sample $\theta^{(1)}, \theta^{(2)}, ...$ from $G \sim DP(\alpha, H)$, then we have $\theta^{(i)} = \theta_{k_i}$ for some k_i.

- Sometimes we get a new value $(k_i \neq k_j \forall i < j)$
- Sometimes we get a repetition$(k_i = k_j$ for some $i < j)$
- Think of $\theta^{(i)}, \theta^{(j)}$ with $k_i = k_j$ as points belonging to the **same cluster**.

12.2.3 Chinese Restaurant Process

Technique to draw samples from a Dirichlet Process:

- Join an existing table with probability $\propto$ the number of people already sitting there
- Start a new table with probability $\propto \alpha$

$$P(\textit{customer } n+1 \textit{ joins table } \tau | \mathcal{P}) = \begin{cases} \frac{|\tau|}{\alpha+n} & \textit{if } \tau \in \mathcal{P} \\ \frac{\alpha}{\alpha+n} & \textit{Otherwise} \end{cases}$$

- α controls the number of new tables (α large: new table likely),
- $|\tau|$ is the number of people on table τ
- $\mathcal{P}$ is a table assignment (partition over the integers)

Expected number of tables (i.e. clusters) created:

$$\mathbb{E}[\textit{created tables}] = \sum_{i=1}^{N} \frac{\alpha}{\alpha + i} \sim O(\alpha \log(N))]$$

"Rich-get-Richer"-effect (preferential attachment): Already popular clusters attract new datapoints

12.2.4 Exchangeability

Definition: Let $(X_1, X_2, ...)$ a sequence of random variables. The sequence is exchangeable when, for every permutation π of $\mathbb{N}$. The random vectors

$$(X_1, X_2, ...) \quad \textit{and} \quad (X_{\pi(1)}, X_{\pi(2)}, ...)$$

have the same distribution.
De Finetti's Theorem: Let $(X_1, X_2, ...)$ be an infinitely exchangeable sequence (one that can be represented by conditionally independent random variables) of random variables. Then $\forall n$

$$p(X_1, ..., X_n) = \int \left(\prod_{i=1}^{n} p(x_i|G) \right) dP(G)$$

for some random variable G
Pólya Urn

- Urn with colored balls
- Draw balls from the urn at random. After drawing a ball, put it back in the urn together with a new ball of the same color

Hoppe Urn: Pólya Urn with special black ball

- Urn with colored balls
- Draw balls from the urn at random. After drawing
  - If non-black ball: put it back in the urn together with a new ball of the same color
  - If black ball: put it back in the urn with a new ball of the same color

Exchangeability Results

- CRP is identical to the hoppe earn process (we just need to add colors to the tables)
- Hoppe Urn and CRP are exchangeable: Can apply **De Finetti's theorem**
- DP is the rancom variable G in De Finetti's theorem for Hoppe urn / CPR
- If the prior of G is the DP, then CRP is how we assign points to clusters when we integrate out G.

12.3 The DP Mixture Model

Θ is a set that parametrizes a set of probability distributions, H fixed base measure on Θ. Example:

- $\Theta = \mathbb{R}$ with $\mu \in \Theta$ corresponding to $\mathcal{N}(\mu, \sigma)$ for some fixed $\sigma > 0$
- $H = \mathcal{N}(\mu_0, \sigma_0)$ for some $\mu_0 \in \mathbb{R}, \sigma_0 \in \mathbb{R}$

Based on that we define the **DP Mixture Model** (a generative model):

- Probabilities of clusters ("mixture weights"): $\rho = (\rho_1, \rho_2, ...) \sim GEM(\alpha)$
- Center of clusters $\mu_k \sim \mathcal{N}(\mu_0, \sigma_0)$
- Assignments to clusters $z_i \sim Categorical(\rho)$
- Coordinates of data points $x_i \sim \mathcal{N}(\mu_{z_i}, \sigma)$

12.4 Gibbs sampling

Useful way of simulating from distributions that are difficult to simulate from directly.
Technique to fit the DPMM (EM considered difficult for nonparametric distributions) by sampling each variable in turn (conditioned on the values of all other variables in the distribution). **Requires exchangeability**.

12.4.1 Fitting

Leverage exchangeability: Any point can be considered "last arrived". Change the assignment of te element without influencing other assignments.

- **Prior**: Probabilities of table assignments w.r.t people seating (cluster size)
- **Posterior**: Probability of the point given the cluster centers

12.4.2 Gibbs Sampling for Fitting

Probability Distribution: Collapsed Gibbs sampling formulation

$$\begin{aligned} & p(z_i = k | \mathbf{z}_{-i}, \mathbf{x}, \alpha, \mu) \\ & \propto p(z_i = k | \mathbf{z}_{-i}, \alpha, \cancel{\mu}) p(\mathbf{x} | z_i = k, \mathbf{z}_{-i}, \cancel{\alpha}, \mu) \\ & \propto p(z_i = k | \mathbf{z}_{-i}, \alpha) p(x_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \mu) p(\mathbf{x}_{-i} | \mathbf{z}_{-i}, \mu) \\ & \propto \underbrace{p(z_i = k | \mathbf{z}_{-i}, \alpha)}_{\textit{Prior}} \underbrace{p(x_i | \mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \mu)}_{\textit{Likelihood}} \end{aligned}$$

where $\mathbf{z}_i$ and $\mathbf{x}_i$ are the assignments and points excluding the considered point i.
Prior:

$$p(z_i = k | \mathbf{z}_{-i}, \alpha) = \begin{cases} \frac{N_{k,-i}}{\alpha+N-1} & \textit{For existing } k \\ \frac{\alpha}{\alpha+N-1} & \textit{otherwise} \end{cases}$$

where $N_{k,-i}$ is the number of elements sitting at table k excluding i ($|\tau \backslash i|$)
Posterior: Observe that if $z_i = k$ we don't need to consider points in $\mathbf{x}$ that are not in k.
Let $\mathbf{x}_{-i,c} = \{x_j : z_j = c, j \neq i\}$ the data assignment to cluster c, then

$$p(x_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \mu) = \begin{cases} p(x_i|\mathbf{x}_{-i,k}, \mu) = \frac{p(x_i, \mathbf{x}_{-i,k}|\mu)}{p(\mathbf{x}_{-i,k}|\mu)} & \textit{for existing } k \\ p(x_i|\mu) & \textit{otherwise} \end{cases}$$

12.4.3 Final Collapsed Gibbs sampler:

$$p(z_i = k|\mathbf{z}_{-i}, \mathbf{x}, \alpha, \mu) = \textit{Prior} \times \textit{Likelihood} = \begin{cases} \frac{N_{k,-i}}{\alpha+N-1} p(x_i|\mathbf{x}_{-i,k}, \mu) & \textit{For existing } k \\ \frac{\alpha}{\alpha+N-1} p(x_i|\mu) & \textit{otherwise} \end{cases}$$

Algorithm 15: Collapsed Gibbs sampler for DP mixtures

for $i = 1$ *to* N *in random order* **do**
  Remove x_i's sufficient statistics from old cluster z_i
  for $k = 1$ *to* K **do**
    Compute $p_k(x_i) = p_k(x_i|\mathbf{x}_{-i,k})$
    Set $N_{k,-i} = |\mathbf{x}{-}i, k|$
    Compute $p(z_i = k|\mathbf{z}_{-i}, \mathbf{x}) = \frac{N_{k,-i}}{\alpha+N-1} p_k(x_i)$
  end
  Compute $p_*(x_i) = p(x_i|\mu)$ *(new cluster)*
  Compute $p(z_i = *|\mathbf{z}_{-i}, \mathbf{x})$
  Normalize $p(z_i|\cdot)$
  Sample $z_i \sim p(z_i|\cdot)$
  Add x_i's sufficient statistics to new cluster z_i
  If any cluster is empty, rmeove it and decrease K
end

12.4.4 Latent Dirichlet Allocation

- Popular nonparametric Bayesian method
- Extension of the model we just defined
- **support multivariate distributions** (e.g. topic modeling on documents, each document belongs to more than one topic; mixture over topics → multivariate)

Given K topics and V words in the vocabulary, for M documents with N words each:

Distribution of topics in doc. d:

$$\theta_d \sim Dir(\alpha)$$

Topic that word w in d belongs to:

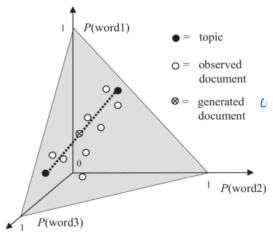
$$z_{d,w} \sim Categorical(\theta_d)$$

Distribution of words in topic k:

$$\varphi_k \sim Dir(\beta)$$

word w in document d:

$$w_{d,w} \sim Categorical(\varphi_{z_{d,w}})$$



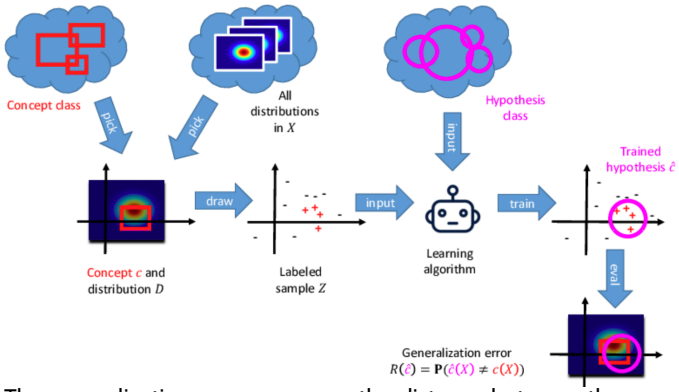
α controls prior weights of topics in documents, β controls prior weights of words in topic

13 Probably Approximately Correct (PAC) Learning

Machines can't compute everything. There are undecidable problems (Turning's halting problem, Post's correspondence problem). Formal logic can't prove everything (Gödel's incompleteness theorem: There are infinitely many truths about arithmetic that cannot be proven formally)

- **Statistical Learning Theory:** Framework for ML aiming at learning **functions from data**
- **PCA learning:** subfield of ML concerned with the question of *what is learnable and how much can we learn something by empirically minimizing a cost function*.

The learning problem:



The generalization error measures the distance between the concept c and the trained hypothesis $\hat{c}$.

- The generalization error is not computable by the learner: $\mathcal{R}(\hat{c}) := \mathbf{P}(\hat{c}(X) \neq c(X))$
- Empirical error is computable by the learner: $\hat{\mathcal{R}}_n(\hat{c}) := \sum_{i \leq n} \mathbf{1}_{\hat{c}(x_i) \neq c(x_i)}$

  It can be shown that $\mathbb{E}_{X, X_1, \ldots, X_N}\left[\hat{\mathcal{R}}_n(\hat{c}(X))\right] = \mathcal{R}(\hat{c})$

13.1 Notions from statistical learning theory

- **Instance space** $\mathcal{X}$ Seit of instances or objects in the learner's world
- **Concept:** subset c of $\mathcal{X}$ (function $c : \mathcal{X} \to \{0, 1\}$ for binary classification).
- **Concept Class:** Set of concepts we wish to learn
- **Hypothesis class:** Other set of concepts that we use to learn a target concept from the concept class.
- No additional prior knowledge on $\mathcal{X}$ is available. This differs from Bayesian approaches.

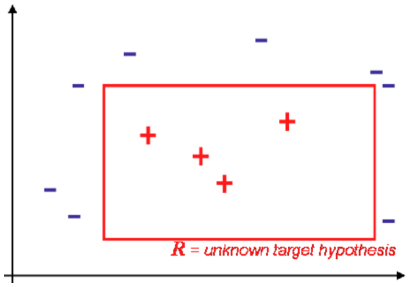
13.2 The PAC Learning Model

- A concept class C is **PAC learnable** from a hypothesis class $\mathcal{H}$ if there is an algorithm that can lern every concept in $\mathcal{C}$.
- If the algorithm runs polynomial time to $1/\epsilon$ and $1/\delta$, we say that $\mathcal{C}$ is **efficiently PAC learnable**. (ϵ error parameter, δ confidence value)

13.3 Rectangle Learning

Axis-Aligned rectangles are PAC learnable.

- $\mathcal{C}$ concept of all axis-aligned rectangles.



- We show $\mathcal{C}$ can be learned from $\mathcal{H} = \mathcal{C}$
- Consider $\mathcal{A}$ that outputs smalles rectangle $\hat{R}$ containing all positively labeled points. **We show that $\mathcal{A}$ can learn any concept $R \in \mathcal{C}$**

How do we prove that $\mathcal{A}$ learns rectangles?

- Define event $\hat{\mathcal{R}}IG$ *($\hat{\mathcal{R}}$ is good enough)*, such that

$$\mathbf{P}(\mathcal{R}(\hat{R}) \leq \epsilon) \geq \mathbf{P}(\hat{\mathcal{R}}IG) \geq 1 - 4\exp\left(-\frac{n\epsilon}{4}\right)$$

- Observe we just need to ensure that $1 - 4\exp\left(-\frac{n\epsilon}{4}\right) \geq 1 - \delta$ or equivalently

$$n \geq \frac{4}{\epsilon} \ln \frac{4}{\delta}$$

- We can ensure this by letting

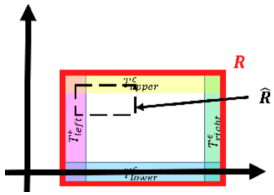
$$n \geq \underbrace{\frac{4}{\epsilon} \times \frac{4}{\delta}}_{poly(1/\epsilon, 1/\delta, 4)} \geq \frac{4}{\epsilon} \ln \frac{4}{\delta}$$

1. Define event $\hat{\mathcal{R}}IG$

Let T^{ϵ}_{upper} be the upper strip sucht that $\mathbf{P}(T^{\epsilon}_{upper}) = \epsilon/4$, ...

$$T^{\epsilon} = \bigcup_i T^{\epsilon}_i$$

$\hat{\mathcal{R}}IG$ is the event in which $\hat{R}$ intersects all 4 strips



2. Prove $\mathbf{P}(\mathcal{R}(\hat{R}) \leq \epsilon) \geq \mathbf{P}(\hat{\mathcal{R}}IG) \geq 1 - 4\exp\left(-\frac{n\epsilon}{4}\right)$

- Learning algorithm: "tightest fitting rectangle"
- Determine

$$\begin{aligned} x_1^{min} &= \min\{x_{1,c} : 1 \leq i \leq n\} \\ x_1^{max} &= \max\{x_{1,c} : 1 \leq i \leq n\} \\ x_2^{min} &= \min\{x_{2,c} : 1 \leq i \leq n\} \\ x_2^{max} &= \max\{x_{2,c} : 1 \leq i \leq n\} \end{aligned}$$

- Select Rectangle $\hat{R} = \square(x_1^{min}, x_1^{max}, x_2^{min}, x_2^{max})$

Error:

$$\begin{aligned} \mathbf{P}(\textit{"error"}) &= \mathbf{P}(R \Delta \hat{R}) \\ &= \mathbf{P}(R - \hat{R}) \cup (\hat{R} - R) = \mathbf{P}(R - \hat{R}) \\ P(R - \hat{R}) &= \mathbf{P}(\textit{top strip} \cup \textit{bottom s.} \cup \textit{left s.} \cup \textit{right s.}) \\ &\leq \mathbf{P}(\textit{top}) + \mathbf{P}(\textit{bottom}) + \mathbf{P}(\textit{left}) + \mathbf{P}(\textit{right}) \end{aligned}$$

Since the probability to be in one of the strips is $\frac{\epsilon}{4}$ for each strip, we get:

$$\begin{aligned} \mathbf{P}(\textit{no sample in top strip}) &= \left(1 - \frac{\epsilon}{4}\right)^n \\ \mathbf{P}(\textit{no sample in any strip}) &\leq 4\left(1 - \frac{\epsilon}{4}\right)^n \\ &\leq 4\exp\left(\frac{n\epsilon}{4}\right) \qquad \leq \delta \end{aligned}$$

Rectangles are PAX learnable, since $n' \geq \frac{4}{\epsilon} \geq \frac{4}{\epsilon} \log \frac{4}{\epsilon}$:

$$\begin{aligned} \mathbf{P}(R \Delta \hat{R} < \epsilon) &\geq 1 - 4\exp\left(\frac{n\epsilon}{4}\right) \\ \mathbf{P}(R \Delta \hat{R} \geq \epsilon) &\leq \underbrace{4}_{\textit{Complexity}} \underbrace{\exp\left(\frac{n\epsilon}{4}\right)}_{\textit{model fit.}} \end{aligned}$$

13.3.1 Error Prob. for realizable finite hypothesis classes $\mathcal{H} = \mathcal{C}$

$\mathcal{C}$ finite, $\mathcal{H} = \mathcal{C}$, consistent hypothesis $\hat{c}\ \forall n < \infty : \hat{\mathcal{R}}_n(\hat{c}) = 0$ for any target concept $c \in \mathcal{C}$, $\hat{c} \in \text{argmin}_{c \in \mathcal{C}} \hat{R}_n(c)$

$$\begin{aligned} \mathbf{P}(R(\hat{c} > \epsilon) &= \mathbf{P}(\max_{c \in \mathcal{X} : \hat{R}_n = 0} R(x) > \epsilon) \\ &\leq \sum_{c \in \mathcal{C}} \mathbf{P}(R(c) > \epsilon \wedge \hat{R}_n(c) = 0) \\ &\leq |\mathcal{C}|(1 - R(c))^n \leq |\mathcal{C}|e^{n\epsilon} \leq \delta \\ &\leq |\mathcal{C}|(1 - \epsilon)^n \leq |\mathcal{C}|e^{n\epsilon} \leq \delta \\ \implies -n\epsilon + \log|\mathcal{C}| &\leq \log\delta \\ \implies n &\geq \frac{1}{\epsilon}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right) \end{aligned}$$

Further:

$$\begin{aligned} &\mathbf{P}(\mathcal{R}(\hat{c}) \leq \epsilon) \geq 1 - \delta \qquad \textit{bounds the success probability } \delta \\ &\mathbf{P}(\mathcal{R}(\hat{c}) > \epsilon) \leq \delta \qquad \textit{bounds the error probability } \epsilon \end{aligned}$$

Proving (efficient) PAC learnability

To prove PAC learnability, we have to show that

$$\mathbf{P}(\mathcal{R}(\hat{c}_n^*) \leq \epsilon) \geq 1 - \delta$$

Efficient PAC learnability:
Show the algorithm runs polynomial in $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$

Example in Exercise 8.3 (Concentric Circles).

13.3.2 The general stochastic setting

In general, an instance's label is not determined by the underlying concepts. We model this by distribution $\mathcal{D}$ on $\mathcal{X} \times \{0, 1\}$. (e.g. two patients with similar features have different reactions on same drug).

- Training dataset: $\mathcal{Z} = \{(x_1, y_1), ..., (x_n, y_n)\}$ from $\mathcal{D}$.

- Goal: Find hypothesis $\hat{c} \in \mathcal{H}$ with small generalization error

$$\mathcal{R}(\hat{c}) = \mathbf{P}_{x,y \sim \mathcal{D}}(\hat{c}(x) \neq y) = \mathbb{E}_{x,y \sim \mathcal{D}}\left(\mathbf{1}_{\hat{c}(x) \neq y}\right)$$

- If the bayes optimimal classifier is not in the hypothesis class $\mathcal{C}$, then it is **impossible to attain** $\forall 0 < \epsilon \leq \frac{1}{2} : \mathcal{R}(\hat{c}) \leq \epsilon$. Instead, we aim to attain the best solution given in the hypothesis class:

$$R(\hat{c}) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) \leq \epsilon$$

13.3.3 The general PAC model

A learning algorithm $\mathcal{A}$ can learn a concept class $\mathcal{C}$ from $\mathcal{H}$ if given sufficiently large sample as input, outputs a hypothesis that generalizes well with high probability

Definition: A learning algorithm $\mathcal{A}$ can learn a concept class $\mathcal{C}$ from $\mathcal{H}$ if there is a polynomial function $poly$, such that

1. For any distribution $\mathcal{D}$ on $\mathcal{X} \times \{0, 1\}$ and
2. for any $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$

if $\mathcal{A}$ receives as input a sample $\mathcal{Z}$ of size $n \geq poly(1/\epsilon, 1/\delta, dim(\mathcal{X}))$, then $\mathcal{A}$ outputs $\hat{c} \in \mathcal{H}$, sucht that

$$\mathbf{P}_{\mathcal{Z} \sim \mathcal{D}^n}\left(\mathcal{R}(\hat{c}) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) \leq \epsilon\right) \geq 1 - \delta$$

Proving PAC learnability in the General stoch. setting

To prove PAC learnability, we have to show that

$$\mathbf{P}(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) \leq \epsilon) \geq 1 - \delta$$

Example (ex. 8.2): Given:
$\mathbf{P}(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) > \epsilon) \leq \exp(-\epsilon n)$

1. Set $\mathbf{P}(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) > \epsilon) \leq \exp(-\epsilon n) \leq \delta$
2. Find n sucht that $\exp(-\epsilon n) \leq \delta$
3. Observe that for this n

$$1 - \mathbf{P}(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) \leq \epsilon) \leq \delta$$

$$\mathbf{P}(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) \leq \epsilon) \leq 1 - \delta$$

Hence $\mathcal{C}$ is PAC learnable from itself.

14 Computational Learning Theory

Attention: There are different notiations used due to different notations in lecture notes and slides.
Minimizing risk is generally unreasonable, because estimating density is more difficult than minimizing the expected risk. It is only plausible if

substantial prior information on $\mathbf{P}(x, y)$ is available and $\mathbf{P}(x, y)$ can be defined up to its parameters. (Vapnik, 1982)

14.1 Empirical Risk Minimization

- Induction principle: Empirical Risk Minimization. Select the classifier $\hat{c}_n^* \in \mathcal{C}$ with the smallest error on the training data $\mathcal{Z} = \{(x_1, y_1), ...(x_n, y_n)\}$

$$\hat{c}_n^* = \underset{c \in \mathcal{C}}{\operatorname{argmin}} \underbrace{\frac{1}{n} \#\{(x_i, y_i) : c(x_j) \neq y_j, 1 \leq i \leq n\}}_{\text{Training error } \hat{\mathcal{R}}_n(c)}$$

- Empirical Classification Error: $\hat{\mathcal{R}}_n(c) = \frac{1}{n} \sum_{j=1}^{n} \mathbb{I}_{\{c(x_j) \neq y_j\}}$
- Expected Classification Error $\mathcal{R}(c) = \mathbf{P}\{c(X) \neq Y\}$ is the quality measure which we care about.
- **Goal:** Derive a distribution independent bound for the prob. of larve deviatons between the expected risk of the ERM classifier and the opimal classifier:
  $\mathbf{P}\{\mathcal{R}(c_n^*) - \inf_{c\mathcal{C}} \mathcal{R}(c) > \epsilon\}$
- Generalization Error: $\mathcal{R}(c_n^*) = \mathbf{P}\{c_n^*(X) \neq Y | \mathcal{Z}\}$
- **Problem:** we cannot measure $\mathcal{R}(c_n^*)$

14.2 VC inequality

How can we bound the difference between expected risk $R(x)$ and empirical risk $\hat{R}_n(x)$? Vapnik-Chervonenkis Inequality.

- $\hat{c} \in \operatorname{argmin}_{c \in \mathcal{C}} \hat{R}_n(c) = \operatorname{argmin}_{c \in \mathcal{C}} |\{\hat{c}(x_i) \neq y_i\}|$ is the minimizer of emirical risk and
- $c^* \operatorname{argmin}_{c \in \mathcal{C}} R(c)$ is the minimizer of the expected risk.

Consider

$$\begin{aligned} R(\hat{c}) - \inf_{c \in \mathcal{C}} R(c) = & \underbrace{R(\hat{c}) - \hat{R}_n(\hat{c})}_{\leq \sup_{c \in \mathcal{C}} |R(c) - \hat{R}_n(c)|} + \underbrace{\hat{R}_n(\hat{c})}_{\leq \hat{R}_n(c^*)} - R(c^*) \\ & \leq \sup_{c \in \mathcal{C}} |R(c) - \hat{R}_n(c)| + \underbrace{\hat{R}_n(c^*) - R(c^*)}_{\leq \sup_{c \in \mathcal{C}} |R(c) - \hat{R}_n(c)|} \\ & \leq 2 \sup_{c \in \mathcal{C}} |R(c) - \hat{R}_n(c)| \end{aligned}$$

The difference between the expected risk of the ERM and smallest expected risk is bounded by twice the worst derivation of expected from empirical risk!

14.2.1 Error Probability Bounds:

$$\begin{aligned} & \mathbf{P}(R(\hat{c}) - \inf_{c \in \mathcal{C}} R(c) \geq \epsilon) \\ & \quad \textit{(VC Inequality)} \leq \mathbf{P}(\sup_{c \in \mathcal{C}} |R(c) - \hat{R}_n(c)|) \\ & \quad \textit{(Union Bound)} \leq \sum_{c \in \mathcal{C}} \mathbf{P}\left(|R(c) - \hat{R}_n(c)| \geq \frac{\epsilon}{2}\right) \\ & \quad = ub(n, \epsilon) = \delta \end{aligned}$$

(*Union Bound:* $\mathbf{P}(e_1 \vee e_2 \vee ... \vee e_k) \leq \sum P(e_j)$)
Solving the last equation we get:

$$\begin{aligned} & ub(n, \epsilon) = \delta \\ & \implies \epsilon = st(n, \delta) \end{aligned}$$

We have $|R(c) - \hat{R}_n(c)| < \frac{\epsilon}{e} = st(n, \delta)$ with high probability $1 - \delta$
We can bound an expected cost by

$$\forall c \in \mathcal{C} : R(c) < \underbrace{\hat{R}_n(c)}_{\text{computable}} + \underbrace{\frac{1}{2} st(n, \delta)}_{\text{overfitting correction}}$$

Risk bounding is a difficult task. In classification, the risk is bounded by $0 \leq R(c) \leq 1$. Therefore we can achieve distribution independent bounds.

14.2.2 Strategies for tight(er) bounds:

- Estimate how different are functions when they are evaluated on finite samples
- Approximate unknown true distribution by empirical distribution

14.2.3 Hoeffding's Inequality

(I think this was not in the lecture)
Lemma 2: *(Markov inequality) Let X be a non-negative random variable. Then*

$$\mathbf{P}\{X \geq \epsilon\} \leq \frac{\mathbb{E}[X]}{\epsilon}$$

Lemma 3 *Let X be a random variable with $\mathbb{E}[X] = 0$ and $a \leq X \leq b$. Then for $s > 0$ it holds that*

$$\mathbb{E}[\exp(sX)] \leq \exp(s^2 (b - a)^2 / 8)$$

Hoeffding's Theorem / Chernoff Bound *Let $X_1, ..., X_n$ be independent bounded random variables such that X_i falls in the interval $[a_i, b_i]$ with probability 1 and let $S_n = \sum_{i=1}^{n} X_i$. Then for any $t > 0$ we have*

$$\mathbf{P}\{S_n - \mathbb{E}[S_n] \geq t\} \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n} (b_i - a_i)^2}\right)$$

$$\mathbf{P}\{S_n - \mathbb{E}[S_n] \leq -t\} \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n} (b_i - a_i)^2}\right)$$