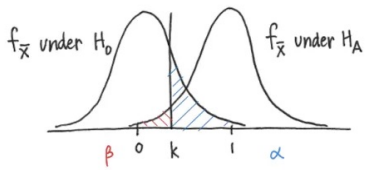
Basics of Statistics

	Definition	Properties
Sample Mean	A random variable. It can be an unbiased estimator of the real (population) mean. $\overline{X_n} = \frac{1}{n}\sum_i^n X_i$	$E[\overline{X_n}] = \mu$ $var(\overline{X_n}) = \frac{\sigma^2}{n} = (SE)^2$, *where* σ^2 *is the variance of* X_i
Central Limit Theorem	Let $X_1, \dots, X_n$ form a random sample of size n from a distribution with finite mean and variance, then for any fixed number z: $\lim_{n\to\infty} \mathbb{P}\left(\frac{\overline{X_n}-\mu}{\frac{\sigma}{\sqrt{n}}} \le z\right) = \Phi(z)$	Interpreting: When n is large, $\overline{X_n}$ is approximately normal: $\overline{X_n} \sim N(\mu, \frac{\sigma^2}{n})$.
Estimation	A guess of an unknow constant/parameter θ. **Estimator**: a random variable. **Estimate**: a number, the realization of the estimator. This course use $\hat{\theta}$ to stand for both.	**Unbiased:** An estimator is unbiased if $E[\hat{\theta}] = \theta$. **Efficient:** An estimator is more efficient if it has a less variance. **Consistent:** An estimator is consistent when it converge to θ in probability as $n \to \infty$. **Robust:** A estimator is robust if it still has a low bias when some assumptions of distribution are wrong.
Sample Variance	$S^2 = \frac{1}{n-1}\sum_i^n (X_i - \overline{X_n})^2$ It's a unbiased estimator of real (population) variance.	
Moment	Population Moments: $E[X], E[X^2], E[X^3], \dots$ Sample Moments: $\frac{1}{n}\sum_i X_i, \frac{1}{n}\sum_i X_i^2, \frac{1}{n}\sum_i X_i^3, \dots$	
? Method of moment		
Maximum Likelihood	This estimator of θ describes a PDFs which is most likely to have produced our data. Let $L(\theta\|X) = \prod f(X_i;\theta)$ be the likelihood function. Then we can find $\hat{\theta}$ by maximize the $L(\theta\|X)$ (derivative=0).	If the maximization is difficult, try: 1. The log of the likelihood function. 2. Any monotone (increasing) transformation of the likelihood function. Properties of maximum likelihood estimator: 1. If there is an efficient estimator, MLE will provide it. 2. Under certain regularity conditions, MLEs will have asymptotically normal distributions.

Standard Error	$SE(\overline{X_n}) = \frac{\sigma}{\sqrt{n}} = \sqrt{var(\overline{X_n})}$ $SE(\overline{X_n}) = \frac{\hat{\sigma}}{\sqrt{n}}$, when σ is unknown, you need to estimate it.	
χ^2 (Chi Square)	If is X_i is normal, *then* $\frac{(n-1)S^2}{\sigma^2} \sim \chi^2_{n-1}$ Where S^2 is the sample variance, σ^2 is the variance of X_i, and $n-1$ is the degrees of freedom	Applications: - You think the true variance is σ^2 , and you want to use your data to make inferences about it .
Student T	If $X \sim N(0,1)$ and $Z \sim \chi^2_n$ and they're independent, then $\frac{X}{\sqrt{Z/n}} \sim t_n$. $\frac{\overline{X_n}-\mu}{S/\sqrt{n}} \sim t_{n-1}$, for X_i *i.i.d.* $N(\mu, \sigma^2)$	The t distribution is similar to the normal but has "fatter tails." It converges to the normal as $n \to \infty$. Applications: - Find CI when is σ unknown. - You think the true mean is 0, and you want to use your data to test whether $\mu = 0$. You don't know the true variance.
F	If $X \sim \chi^2_n$ and $Z \sim \chi^2_m$ and they're independent, then $\frac{\frac{X}{n}}{\frac{Z}{m}} \sim F_{n,m}$	Applications: - Compare the variances in two independent population. - check if 2 random samples are in the same population.
Confidence Interval	A $1-\alpha$ confidence interval is an interval $[A(X_1,\dots,X_n), B(X_2,\dots,X_n)]$ such that: $\mathbb{P}(A(X_1,\dots,X_n) \le \theta \le B(X_2,\dots,X_n)) = 1-\alpha$	Case 1: sampling from normal, variance is known, find CI for the mean $\mathbb{P}\left(\overline{X} + \frac{\sigma}{\sqrt{n}}\Phi^{-1}(\alpha/2) \le \mu \le \overline{X} - \frac{\sigma}{\sqrt{n}}\Phi^{-1}(\alpha/2)\right) = 1-\alpha$ Case 2: sampling from normal, variance is unknown, find CI for the mean $\mathbb{P}\left(\overline{X} + \frac{S}{\sqrt{n}}t^{-1}_{n-1}(\alpha/2) \le \mu \le \overline{X} - \frac{S}{\sqrt{n}}t^{-1}_{n-1}(\alpha/2)\right) = 1-\alpha$
Hypothesis Testing	An **hypothesis** is an assumption about the distribution of a random variable in a population. A **maintained hypothesis** is one that cannot or will not be tested. A **testable hypothesis** is one that can be tested using evidence from a random sample. The **null hypothesis**, H_0, is the one that will be tested. The **alternative hypothesis**, H_A, is a possibility (or series of possibilities) other than the null. A **simple hypothesis** is one characterized by a single point, i.e., $\theta = 0$. A **composite hypothesis** is one characterized by multiple points, i.e., $\theta > 0$. **Critical region** of the test, C or C_X, as the region of the support of the random sample for which we reject the null. Test statistic T is the random vairable you calculated from the sample data, and you use it to calculate the p-value.	Testing steps: 1. You have a hypothesis (null hypothesis) 2. You observed something relevant (you got a sample) 3. You setup a threshold, i.e., 5% (***significant level***) 4. Calculate that (***p-value***) if the hypothesis is true, what is the probability (with the tail) that your sample will show up 5. If the p-value is less than the your threshold (which means it's very unlikely to happen), then you should reject it
Errors	<table><tr><th></th><th>H_0 true</th><th>H_0 false</th></tr><tr><td>accept H_0</td><td>No error</td><td>Type II error</td></tr><tr><td>reject H_0</td><td>Type I error</td><td>No error</td></tr></table> Significance Level = The Probability of Type I Error = α Confidence Level = $1-\alpha$ Operating Characteristic = The Probability of Type II Error = β Power = $1-\beta$	
? Power Calculation		

Randomised Controlled Trials (RCTs)

Some Basic Concepts	**Causal effect** for person i: $Y_i(1) - Y_i(0)$, 1 means being treated; only one of them can be observed. **Treatment effect on the treated:** $E[Y_i(1)\|W_i = 1] - E[Y_i(0)\|W_i = 1]$ **Selection bias:** $E[Y_i(0)\|W_i = 1] - E[Y_i(0)\|W_i = 0]$	Types of RCT: - Completely randomized - Stratified randomization - Pairwise randomization - Clustered randomization
SUTVA	**Stable Unit Treatment Value Assumption** The potential outcomes for any unit do not vary with the treatments assigned to other units; and, for each unit, there are no different forms or versions of each treatment unit leading to different outcomes.	SUTVA wikipedia version: "the [potential outcome] observation on one unit should be unaffected by the particular assignment of treatments to the other units" (Cox 1958, §2.4)
ATE & Estimator	**Average treatment effect:** $E[Y^{obs}\|W_i = 1] - E[Y^{obs}\|W_i = 0]$ Estimator: $\hat{\tau} = \overline{Y_t^{obs}} - \overline{Y_c^{obs}}$ Variance of estimator ($(SE)^2$): $V(\hat{\tau}) = \frac{S_c^2}{N_c} + \frac{S_t^2}{N_t}$ Estimator of variance ($(SE)^2$): $\widehat{V(\hat{\tau})} = \frac{s_c^2}{N_c} + \frac{s_t^2}{N_t}$ $s_c^2 = \frac{1}{N_c - 1}\sum_{i:W_i=0} (Y_i - \overline{Y_c^{obs}})$ $s_t^2 = \frac{1}{N_t - 1}\sum_{i:W_i=0} (Y_i - \overline{Y_t^{obs}})$	
Confidence Interval of ATE Estimator	Test statistic (t statistic): $t = \frac{\hat{\tau}}{\sqrt{\hat{V}}} = \frac{\overline{Y_t^{obs}} - \overline{Y_c^{obs}}}{\sqrt{\hat{V}}}$, which follows a t-distribution. Confidence interval: $CI^{\tau}_{1-\alpha} = \left(\hat{\tau} - t_{crit} * \sqrt{\hat{V}}, \hat{\tau} + t_{crit} * \sqrt{\hat{V}}\right)$	Small samples: - find t_{crit} in t-table with α and $df = N_t + N_c - 1$ Large samples: - Use normal approximation - 1.645 for $\alpha = 0.1$ - 1.96 for $\alpha = 0.05$
Hypothesis Testing	H_0: ate is 0 H_1: ate is not 0 Test statistic (t statistic): $t = \frac{\overline{Y_t^{obs}} - \overline{Y_c^{obs}}}{\sqrt{\hat{V}}}$	Small samples: - t follows a t-distribution - $df = N - 1$ - $p = 2 * (1 - F_T(t))$ Large samples: - Use normal approximation - $p = 2 * (1 - \Phi(t))$

Fisher Exact Test

Hypothesis Testing	$H_0: Y_i(1) = Y_i(0)\ for\ all\ i$ $p = \Pr(\|T^{ave}(W, Y^{obs})\| \ge \|T^{ave}(W^{obs}, Y^{obs})\|)$	Steps: 1. Calculate the estimate of ATE, which is $\hat{\tau} = \overline{Y_t^{obs}} - \overline{Y_c^{obs}}$ 2. Get all possible permutations for treatments and controls 3. Calculate $t^{ave} = \overline{Y_t} - \overline{Y_c}$ in every permutation 4. $p = \frac{count(t^{ave*} > \hat{\tau})}{count(permutations)}$
Simulation	Instead of getting all permutations, we generate some random permutations, then p is: $p = \frac{count(t^{ave*} > \hat{\tau})}{count(random\ \ permutations)}$	

KS Test

Some Important Concepts	**Exploratory data analysis:** In statistics, exploratory data analysis (EDA) is an approach analyzing data sets to summarize their main characteristics, often with visual methods. (from wikipedia) **First order stochastically dominance (FOD) :** Distribution A to have first order stochastically dominance over Distribution B means: the cumulative distribution function (CDF) of Distribution A is always below or equal to the CDF of Distribution B, and must be strictly below at some value correct.	
K-S Test	**Kolmogorov–Smirnov test**: In statistics, the Kolmogorov–Smirnov test (K–S test or KS test) is a nonparametric test of the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one-sample K–S test), or to compare two samples (two-sample K–S test). (from wikipedia)	Applications: - Compare distributions (r.v.'s) - Test if two distributions are the same, e.g., $Y(0)$ and $Y(1)$
Statistics	Hypothesis testing: $H_0: F = G$ (KS statistics tends to 0) $H_a: F > G$ KS statistics: $D^+_{nm} = \max[F_n(x) - G_m(x)]$ KS statistics follows the distribution of: $\left(\frac{nm}{n+m}\right)^{\frac{1}{2}} D_{nm}$	Reject H_0 if $D_{nm} > \frac{c(\alpha)(nm)}{n+m}$. Use `ks.test` in R.

Kernel Regression

Nonparametric Regression	**Nonparametric regression**: Nonparametric regression is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data. (from wikipedia)	
Kernel Regression	**Kernel regression**: Kernel regression is a non-parametric technique in statistics to estimate the conditional expectation of a random variable. The objective is to find a non-linear relation between a pair of random variables X and Y. $E[Y\|X] = g(X) \to y = g(x) + \epsilon$ $E[Y\|X = x] = \frac{1}{f(x)}\int y f(x,y) dy$ Kernel estimator replace $f(x,y)$ and $f(x)$ by their empirical estimates: $\hat{g}(x) = \frac{1}{\hat{f}(x)}\int y \hat{f}(x,y) dy = \frac{\sum_{i=1}^n y_i K\left(\frac{x - x_i}{h}\right)}{\sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)}$ Where h is the bandwidth (a positive number), the kernel estimate of the density of x; And $K()$ is your choice of kernel function, a density.	Choice of bandwidth: - A large bandwidth will lead to more bias (and less variance). - A small bandwidth will lead to more variance (and less bias) Choice of kernel: ❶ Histogram: $K(u) = 1/2$ if $\|u\| \le 1$, $K(u) = 0$ otherwise. ❷ Epanechnikov $K(u) = \frac{3}{4}(1-u^2)$ if $\|u\| \le 1$ $K(u) = 0$ otherwise ❸ Quartic $K(u) = (\frac{3}{4}(1-u^2))^2$ if $(u \le 1)$, $K(u) = 0$ otherwise Optimize your choice: using cross-validation.
Cross Validation	**Cross Validation** Cross-validation, sometimes called rotation estimation, or out-of-sample testing is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.	

Linear Regression

Linear model

$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i, i = 1, \dots, n$

- Y: dependent variable
- X: regressor/explanatory variable
- β_0, β_1: regression coefficients
- ϵ: error

Linear regression

The way we estimate the parameters (β_0 and β_1).

Assumptions

1. X_i, ϵ_i uncorrelated
2. Identification: $\frac{1}{n}\sum_i (X_i - \overline{X})^2 > 0$
3. Zero mean: $E[\epsilon_i] = 0$
4. Homoskedasticity: $var(\epsilon_i) = \sigma^2$ for all i
5. No serial correlation: $E[\epsilon_i \epsilon_j] = 0, if\ i \neq j$

For:

- Prediction
- Determining causality
- understanding the world better

Multivariate Linear Model

Least Squares Estimator

$$\min_{\beta} \sum_i (Y_i - \beta_0 - \beta_1 X_i)^2$$

$$\widehat{\beta_1} = \frac{\frac{1}{n}\sum_i (X_i - \overline{X})(Y_i - \overline{Y})}{\frac{1}{n}\sum_i (X_i - \overline{X})^2}$$

$$\widehat{\beta_0} = \overline{Y} - \widehat{\beta_1}\overline{X}$$

Let $\overline{X} = \frac{1}{n}\sum X_i$ and $\hat{\sigma}_x^2 = \frac{1}{n}\sum (X_i - \overline{X})^2$.

	mean	Variance	covariance
$\hat{\beta}_0$	β_0	$\sigma^2 \overline{X}^2 / n\hat{\sigma}_x^2 + \sigma^2/n$	$-\sigma^2 \overline{X} / n\hat{\sigma}_x^2$
$\hat{\beta}_1$	β_1	$\sigma^2 / n\hat{\sigma}_x^2$	

Estimation of error variance σ^2:

$$\widehat{\sigma^2} = \frac{1}{n-2}\sum_i \epsilon_i^2$$

Properties of estimator (in model):

- Least variance (most efficient)
- Unbiased
- Normal errors mean normal estimates

Some comparative statics:

---A larger σ^2 means larger $Var(\hat{\beta})$

---A larger $\hat{\sigma}_x^2$ means smaller $Var(\hat{\beta})$

---A larger n means smaller $Var(\hat{\beta})$

---If $\overline{X} > 0$, $Cov(\beta_0, \beta_1) < 0$

If we use the stronger assumption that the errors are i.i.d. $N(0, \sigma^2)$, we obtain the result that β_0 and β_1 will also have normal distributions.

Multivariate Linear Model	$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_k X_{ki} + \epsilon_i, i = 1, \dots, n$ Or $X\beta + \epsilon = Y$ $Y = (Y_1, \dots, Y_n)^T$, a $n \times 1$ column vector $\epsilon = (\epsilon_1, \dots, \epsilon_n)^T$, a $n \times 1$ column vector $X = \begin{bmatrix} X_{01} & \dots & X_{k1} \\ X_{02} & \dots & X_{k2} \\ & \vdots & \end{bmatrix}$, a $n \times (k+1)$ matrix ($X_{0i} = 1$, for β_0) Assumptions: 1) $n > k + 1$ 2) $\epsilon \sim N(0, \sigma^2 I_n)$, *where I_n is a $n \times n$ identity matrix*
Estimator	$\hat{\beta} = (X^T X)^{-1} X^T Y$ $T = \frac{\hat{\beta}_i - \mu}{SE(\hat{\beta}_i)}$ follows a t-distribution. Hypothesis testing H_0: $\beta_i = c$ $T = \frac{\hat{\beta}_i - c}{SE(\hat{\beta}_i)}$, *where* $SE(\hat{\beta}_i) = (\sigma^2 (X^T X)^{-1})_{ii}^{1/2}$ R is a $1 \times (k+1)$ matrix: H_0: $R\beta = c$ $T = \frac{R\hat{\beta} - c}{SE(R\hat{\beta})}$, *where* $SE(R\hat{\beta}) = (\sigma^2 R(X^T X)^{-1} R^T)^{1/2}$
Testing Restrictions	R is a r*(k+1) matrix of restrictions. H_0: $R\beta = c$ Testing steps: 1. Estimate the unrestricted model. 2. Impose the restrictions of the null and estimate that model. 3. Compare the goodness-of-fit of the models. If the restrictions don't really affect the fit of the model much, then the null is probably true or close to true. $T = \frac{\frac{SSR_R - SSR_U}{r}}{\frac{SSR_U}{n-(k+1)}}$ r is the restriction number (the number of independent variables are dropped) n is the observations number, $k+1$ is the variables number $T \sim F_{r, n-(k+1)}$

Interpreting assumptions:

- need to have more observations than regressors
- can't have any regressors that do not have positive sample variation
- can't have any regressors that are linear functions of one or more other regressors

$E[\hat{\beta}] = \beta$

$cov(\hat{\beta}) = \sigma^2 (X^T X)^{-1}$

$\hat{\sigma}^2 = \frac{\hat{\epsilon}^T \epsilon}{n-k}$

For the hypothesis H_0: $\beta_i = c$ versus H_0: $\beta_i \neq c$, the F-test is equivalent to the t-test ($\sqrt{F}$=T).

Restriction examples:

If, for instance, R = [0 1 0 . . 0] and c = [0] that corresponds to H_0: $\beta_1 = 0$.

If, for instance, $R = \begin{bmatrix} 0 & 1 & 0 & . & . & 0 \\ 0 & 0 & 1 & . & . & 0 \\ & & & . & . & \\ 0 & 0 & 0 & . & . & 1 \end{bmatrix}$ and $c = \begin{bmatrix} 0 \\ 0 \\ \\ . \\ . \\ 0 \end{bmatrix}$

that corresponds to H_0: $\beta_1 = \beta_2 = \dots = \beta_k = 0$.

If, for instance, $R = \begin{bmatrix} 0 & 1 & -1 & . & . & 0 \\ 0 & 0 & 0 & 1 & . & 0 \\ 0 & 0 & 0 & . & . & 1 \end{bmatrix}$ and $c = \begin{bmatrix} 0 \\ 5 \\ -2 \end{bmatrix}$

that corresponds to H_0: $\beta_1 = \beta_2$, $\beta_3 = 5$, and $\beta_k = -2$.

R functions - part 1

name	description
dname( )	density or probability function
pname( )	cumulative density function
qname( )	quantile function
Rname( )	random deviates

Dummy Variables/DID

Dummy Variables	**Dummy Variable** Dummy variable (indicator variable) is one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. (from wikipedia) Categorical Variables: • Use multiple dummy variables to build categories • "−1" to avoid colinear problem	Example: $Y_i = \alpha + \beta D_i + \epsilon_i$ D_i takes the value 1 if the observation is in group A, and 0 if in group B. $\hat{\beta} = \overline{Y_A} - \overline{Y_B}$ You can always estimate the difference between the treatment and control group for an RCT using an OLS regression framework. The standard errors will be slightly different from the Neyman standard errors, but it won't matter that much if the samples are large enough, and similar in treatment and control groups.
Difference in differences	$Y_i = \alpha + \beta D_i + \gamma M_i + \delta M_i * D_i + \epsilon_i$ α: mean for women in the control group β: treatment main effect γ: gender main effect δ: difference between the treatment effect for males and for female, interaction effect DID table: $D=0$ | $D=1$ $M=0$ | α | $\alpha+\beta$ $M=1$ | $\alpha+\gamma$ | $\alpha+\beta+\gamma+\delta$ $\delta = ((\alpha+\beta+\gamma+\delta) - (\alpha+\gamma)) - ((\alpha+\beta) - \alpha)$	A variation example: $$S_{ijk} = c_1 + \alpha_{1j} + \beta_{1k} + (P_j * T_i)\gamma_1 + \epsilon_{ijk}\ , \qquad (1)$$ where • S_{ijk} is the education of individual i born in region j in year k, • T_i is a dummy indicating whether the individual belongs to the "young" cohort in the subsample, • P_j denotes the intensity of the program in the region of birth (number of school built) • c_1 is a constant, • β_{1k} is a set of cohort-of-birth fixed effects [in practice, a series of dummies=1 for each year of birth, omit 1] • α_{1j} is a set of district-of-birth fixed effects [in practice, a series of dummies=1 for each district of birth, omit 1]

Non-Linear Models

Transformations	$Y_i = A X_{1i}^{\beta_1} X_{2i}^{\beta_2} e^{\epsilon_i} \Rightarrow \log(Y_i) = \beta_0 + \beta_1 \log X_{1i} + \beta_2 \log X_{2i} + \epsilon$ Education formulation: $\log Y_i = \beta_0 + \beta_1 S_i + \epsilon_i$ Box Cox Transformation: $Y_i = \frac{1}{\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i} \Rightarrow \frac{1}{Y_i} = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i$ Discrete choice model: $P_i = \frac{e^{\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i}}{1 + e^{\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i}} \Rightarrow Y_i = \log \frac{P_i}{1 - p_i} = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i$
Polynomial Models	$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{1i}^2 + \cdots + \beta_k X_{1i}^k + \epsilon_i$
Using Dummies for Approximation	• Partition the range of X is interval. $X_0, \dots X_J$ • Define the dummies as: $D_{1i} = I_{[X_0 \leq X_{1i} < X_1]}$ $D_{2i} = I_{[X_1 \leq X_{1i} < X_2]}$ $\vdots$ $D_{ji} = I_{[X_{j-1} \leq X_{1i} < X_j]}$ • you can run regression: $Y_i = \beta_1 D_{1i} + \beta_2 D_{2i} + \cdots + \beta_J D_{ji} + \epsilon_i$

Keys:

- You can chose series expansion, or orthogonal polynomials
- If you assume the model is known, this is OLS
- If you assume the model is unknown, this is a non-parametric method: series regression

- Define Piece wise linear variables as:
  $S_{1i} = I_{[X_0 \leq X_{1i} < X_1]}(X_{1i} - X_1)$ $S_{2i} = I_{[X_1 \leq X_{1i} < X_2]}(X_{1i} - X_2)$
- Run regression
  $$Y_i = \beta_1 X_{1i} + \beta_2 S_{1i} + \cdots + \beta_J S_{j-1i} + \epsilon_i$$

Regression Discontinuity Design

Locally Linear Regression	Like a kernel, but using a linear regression in each little interval instead.
Regression Discontinuity Design	A regression discontinuity design (RDD) is a quasi-experimental pretest-posttest design that elicits the causal effects of interventions by assigning a cutoff or threshold above or below which an intervention is assigned. (from wikipedia) **Running Variable** RD is appropriate in any circumstance where some treatment shift discontinuously with a variable a, called the running variable.

Locally linear estimation would have no bias if the true model were linear.
Locally linear estimation removes a bias term from the kernel estimator, that makes it have better behavior near the boundary of the x's and smaller MSE everywhere.

"You cannot do a regression discontinuity analysis without a graph."

An Example

Simplest model: $Y_i = \beta_0 + \beta_1 D_{ai} + \beta_2 a_i + \epsilon_i$, (same slopes for two parts)

Improvement:

- Add polynomials
- Fit a polynomial on each side:
  $Y_i = \beta_0 + \beta_1 D_{ai} + \beta_2 (a_i - a_0) + \beta_3 (a_i - a_0) * D_{ai} + \cdots + \epsilon_i$
- Narrow the bandwidth

Omitted Variable Bias

Omitted Variable Bias	**OVB** In statistics, omitted-variable bias (OVB) occurs when a statistical model leaves out one or more relevant variables. The bias results in the model attributing the effect of the missing variables to the estimated effects of the included variables. (from wikipedia) Correct model: $Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i$ Estimated model: $Y_i = \alpha_0 + \alpha_1 X_{1i} + w_i$ **Ancillary regression**: $X_{2i} = \delta_0 + \delta_1 X_{1i} + \xi_i$ **OVB**: $OVB = \widehat{\alpha_1} - \beta_1 = \delta_1 \beta_2$

Omitted variable bias depends on:

1. How important is X_2 in the original model
2. How correlated it is with X_1

Proof:

$$\widehat{\alpha_1} = \frac{Cov(Y_i, X_{1i})}{V(X_{1i})}$$

substituting for Y_i we get:

$$\frac{Cov(\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \epsilon_i, X_{1i})}{V(X_{1i})}$$

$$= \frac{\beta_1 V(X_{1i}) + \beta_2 Cov(X_{2i}, X_{1i}) + Cov(\epsilon_i, X_{1i})}{V(X_{1i})}$$

$$= \beta_1 + \delta_1 \beta_2$$

Advanced techniques a reasonable causal model:

- Matching methods
- Propensity score matching
- Machine Learning techniques

R functions - part 2

distribution	R name	distribution	R name
Beta	beta	Lognormal	lnorm
Binomial	binom	Negative Binomial	nbinom
Cauchy	cauchy	Normal	norm
Chisquare	chisq	Poisson	pois
Exponential	exp	Student t	t
F	f	Uniform	unif
Gamma	gamma	Tukey	tukey
Geometric	geom	Weibull	weib
Hypergeometric	hyper	Wilcoxon	wilcox
Logistic	logis		

6.86x Machine Learning with Python

This is a cheat sheet for machine learning based on the online course given by Prof. Tommi Jaakkola and Prof. Regina Barzilay. Compiled by Janus B. Advincula.

Last Updated January 17, 2020

Linear Classifiers

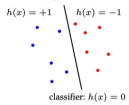
Introduction to Machine Learning

What is machine learning? Machine learning as a discipline aims to design, understand and apply computer programs that learn from experience (i.e., data) for the purpose of modeling, prediction or control.

Types of Machine Learning

- **Supervised learning:** prediction based on examples of correct behavior
- **Unsupervised learning:** no explicit target, only data, goal is to model/discover
- **Semi-supervised learning:** supplement limited annotations with unsupervised learning
- **Active learning:** learn to query the examples actually needed for learning
- **Transfer learning:** how to apply what you have learned from A to B
- **Reinforcement learning:** learning to act, not just predict; goal is to optimize the consequences of actions

Linear Classifier and Perceptron



Key Concepts

- **feature vectors, labels:**

$$x \in \mathbb{R}^d, \quad y \in \{-1, +1\}$$

- **training set:**

$$S_n = \left\{ \left(x^{(i)}, y^{(i)} \right), i = 1, \ldots, n \right\}$$

- **classifier:**

$$h : \mathbb{R}^d \to \{-1, +1\}$$
$$\chi^+ = \left\{ x \in \mathbb{R}^d : h(x) = +1 \right\}$$
$$\chi^- = \left\{ x \in \mathbb{R}^d : h(x) = -1 \right\}$$

- **training error:**

$$\mathcal{E}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \left[\left[h\left(x^{(i)} \right) \neq y^{(i)} \right] \right]$$

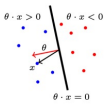
$$\left[\left[h\left(x^{(i)} \right) \neq y^{(i)} \right] \right] = \begin{cases} 1 & \text{if error} \\ 0 & \text{otherwise} \end{cases}$$

- **test error:** $\mathcal{E}(h)$

- **set of classifiers:** $h \in \mathcal{H}$

Linear Classifiers through the Origin We consider functions of the form

$$h(x; \theta) = \text{sign}\left(\theta_1 x_1 + \cdots + \theta_d x_d \right) = \text{sign}\left(\theta \cdot x \right).$$



Linear Classifiers with Offset We can consider functions of the form

$$h(x; \theta) = \text{sign}\left(\theta \cdot x + \theta_0 \right)$$

where θ_0 is the offset parameter.

Linear Separation Training examples S_n are *linearly separable* if there exists a parameter vector $\widehat{\theta}$ and offset parameter $\widehat{\theta}_0$ such that $y^{(i)} \left(\widehat{\theta} \cdot x^{(i)} + \widehat{\theta}_0 \right) > 0$ for all $i = 1, \ldots, n$.

Training Error The training error for a linear classifier is

$$\mathcal{E}_n(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left[\left[y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0 \right) \leq 0 \right] \right].$$

Perceptron Algorithm

procedure PERCEPTRON$\left(\{ (x^{(i)}, y^{(i)}), i = 1, \ldots, n \}, T \right)$
  $\theta = 0$ (vector)
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      if $y^{(i)} \left(\theta \cdot x^{(i)} \right) \leq 0$ **then**
        $\theta = \theta + y^{(i)} x^{(i)}$
  return θ

Perceptron Algorithm (with offset)

procedure PERCEPTRON$\left(\{ (x^{(i)}, y^{(i)}), i = 1, \ldots, n \}, T \right)$
  $\theta = 0$ (vector)
  $\theta_0 = 0$
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      if $y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0 \right) \leq 0$ **then**
        $\theta = \theta + y^{(i)} x^{(i)}$
        $\theta_0 = \theta_0 + y^{(i)}$
  return θ, θ_0

Convergence Assumptions:

- There exists θ^* such that $\frac{y^{(i)} \left(\theta^* \cdot x^{(i)} \right)}{\left\| x^{(i)} \right\|} \geq \gamma$ for all $i = 1, \ldots, n$ for some $\gamma > 0$.
- All examples are bounded $\left\| x^{(i)} \right\| \leq R, i = 1, \ldots, n$.

Then the number k of updates made by the perceptron algorithm is bounded by $\frac{R^2}{\gamma^2}$.

Hinge Loss, Margin Boundaries and Regularization

Distance from a Line to a Point The perpendicular distance from a line with equation $\theta \cdot x + \theta_0 = 0$ to a point with coordinates x_0 is

$$d = \frac{|\theta \cdot x_0 + \theta_0|}{\|\theta\|}$$

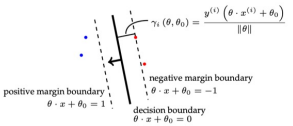
Decision Boundary The decision boundary is the set of points x which satisfy

$$\theta \cdot x + \theta_0 = 0.$$

Margin Boundary The margin boundary is the set of points x which satisfy

$$\theta \cdot x + \theta_0 = \pm 1.$$

$$\gamma_i(\theta, \theta_0) = \frac{y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0 \right)}{\|\theta\|}$$



Hinge Loss

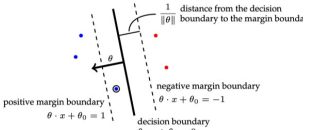
$$\text{Loss}_h(z) = \begin{cases} 0 & \text{if } z \geq 1 \\ 1 - z & \text{if } z < 1 \end{cases}$$

with $z = y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0 \right)$.

Regularization Maximize margin

$$\max \frac{1}{\|\theta\|} \quad \Rightarrow \quad \min \frac{1}{2} \|\theta\|^2$$

Linear Classification and Generalization



Objective function

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left[\text{Loss}_h \left(y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0 \right) \right) + \frac{\lambda}{2} \|\theta\|^2 \right]$$

λ is the regularization factor.

Stochastic Gradient Descent Select $i \in \{1, \ldots, n\}$ at random

$$\theta \quad \leftarrow \quad \theta - \eta_t \nabla_\theta \left[\text{Loss}_h \left(\theta \cdot x^{(i)} + \theta_0 \right) + \frac{\lambda}{2} \|\theta\|^2 \right]$$

η_t is the learning rate which can vary at every iteration.

Support Vector Machine

- Support Vector Machine finds the maximum margin linear separator by solving the quadratic program that corresponds to $J(\theta, \theta_0)$
- In the realizable case, if we disallow any margin violations, the quadratic program we have to solve is:

  Find θ, θ_0 that minimize $\frac{1}{2} \|\theta\|^2$ subject to

$$y^{(i)} \left(\theta \cdot x^{(i)} + \theta_0 \right) \geq 1, \quad i = 1, \ldots, n$$

Nonlinear Classification, Linear Regression, Collaborative Filtering

Linear Regression

Empirical Risk

$$R_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \left(y^{(i)} - \theta \cdot x^{(i)} \right)^2 \qquad \text{squared error}$$

Gradient-based Approach We can use stochastic gradient descent to find the minima of the empirical risk.

Algorithm Initialize $\theta = 0$.
Randomly pick $i = \{1, \ldots, n\}$.
$\theta = \theta + \eta \left(y^{(i)} - \theta \cdot x^{(i)} \right) x^{(i)}$.

η is the learning rate.

Closed Form Solution Let

$$A = \frac{1}{n} \sum_{i=1}^{n} x^{(i)} \left(x^{(i)} \right)^\intercal \quad \text{and} \quad B = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} x^{(i)}.$$

Then,

$$\widehat{\theta} = A^{-1} B.$$

In matrix notation, this is

$$\widehat{\theta} = \left(\mathrm{X}^\intercal \mathrm{X} \right)^{-1} \mathrm{X}^\intercal \mathrm{Y}.$$

Generalization and Regularization

Ridge Regression: The loss function is

$$J_{\lambda,n} = \frac{\lambda}{2} \|\theta\|^2 + R_n(\theta)$$

where λ is the regularization factor. We can find its minima using gradient-based approach.

Algorithm Initialize $\theta = 0$.
Randomly pick $i = \{1, \ldots, n\}$.
$\theta = (1 - \eta\lambda)\theta + \eta \left(y^{(i)} - \theta \cdot x^{(i)} \right) x^{(i)}$.

Nonlinear Classification

Feature Transformation

$$x \mapsto \phi(x)$$
$$\theta \cdot x \to \theta' \cdot \phi(x)$$

Non-linear Classification

$$h(x; \theta, \theta_0) = \text{sign}\left(\theta \cdot \phi(x) + \theta_0 \right)$$

Kernel Function A kernel function is simply an inner product between two feature vectors. Using kernels is advantageous when the inner products are faster to evaluate than using explicit vectors (e.g., when the vectors would be infinite dimensional).

$$K\left(x, x' \right) = \phi(x) \cdot \phi(x')$$

Perceptron

$\theta = 0$
for $i = 1, \ldots, n$ **do**
  if $y^{(i)} \theta \cdot \phi\left(x^{(i)} \right) \leq 0$ **then**
    $\theta \leftarrow \theta + y^{(i)} \phi\left(x^{(i)} \right)$

This algorithm gives

$$\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} \phi\left(x^{(j)} \right)$$

where α_j is the number of mistakes. For the offset parameter, we get

$$\theta_0 = \sum_{j=1}^{n} \alpha_j y^{(j)}.$$

Kernel Perceptron Algorithm We can reformulate the perceptron algorithm so that we initialize and update a_j's, instead of θ.

$$\theta \cdot \phi\left(x^{(i)} \right) = \sum_{j=1}^{n} \alpha_j y^{(j)} \underbrace{\phi\left(x^{(j)} \right) \cdot \phi\left(x^{(i)} \right)}_{K\left(x^{(j)}, x^{(i)} \right)}$$

procedure KERNEL PERCEPTRON$\left(\{ (x^{(i)}, y^{(i)}), i = 1, \ldots, n \}, T \right)$
  Initialize $\alpha_1, \ldots, \alpha_n$ to some values
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      if $y^{(i)} \sum_{j=1}^{n} \alpha_j y^{(j)} K\left(x^{(j)}, x^{(i)} \right) \leq 0$ **then**
        $\alpha_j = \alpha_j + 1$

The initilization $\theta = 0$ is equivalent to $\alpha_1 = \cdots = \alpha_n = 0$.

Composition rules:

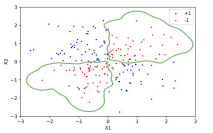
1. $K(x, x') = 1$ is a kernel function.
2. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $K(x, x')$ is a kernel. Then so is $\widetilde{K}(x, x') = f(x) K(x, x') f(x')$
3. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel.
4. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then $K(x, x') = K_1(x, x') K_2(x, x')$ is a kernel.

Decision Boundary The decision boundary satisfies

$$\sum_{j=1}^{n} \alpha_j y^{(j)} K\left(x^{(j)}, x \right) = 0.$$

Radial Basis Kernel

$$K(x, x') = \exp\left(-\frac{1}{2} \left\| x - x' \right\|^2 \right)$$



Other non-linear classifiers

- We can get non-linear classifiers or regression methods by simply mapping examples into feature vectors non-linearly, and applying a linear method on the resulting vectors.
- These feature vectors can be high dimensional.
- We can turn the linear methods into kernel methods by casting the computations in terms of inner products.

Recommender Systems

Problem Description We are given a matrix where each row corresponds to a user's rating of movies, for example, and each column corresponds to the user ratings for a particular movie. It can also be product ratings, etc. This matrix will be very sparse. The goal is to predict user ratings for those movies that are yet to be rated.

$$\begin{array}{c} \\ n \text{ users} \end{array} \begin{array}{c} m \text{ movies} \\ \begin{pmatrix} & \\ & Y_{ai} \\ & \end{pmatrix} \end{array}$$

***K*-Nearest Neighbor Method** The K-Nearest Neighbor method makes use of ratings by K other *similar* users when predicting Y_{ai}. Let $\text{KNN}(a)$ be the set of K users *similar* to user a, and let $\text{sim}(a, b)$ be a **similarity measure** between users a and $b \in \text{KNN}(a)$. The KNN method predicts a rating Y_{ai} to be

$$\widehat{Y}_{ai} = \frac{\sum_{b \in \text{KNN}(a)} \text{sim}(a, b) Y_{bi}}{\sum_{b \in \text{KNN}(a)} \text{sim}(a, b)}$$

The similarity measure $\text{sim}(a, b)$ could be any distance function between the feature vectors x_a and x_b.

- Euclidean distance: $\|x_a - x_b\|$
- Cosine similarity: $\cos\theta = \frac{x_a \cdot x_b}{\|x_a\| \|x_b\|}$

Collaborative Filtering Our goal is to come up with a matrix X that has no blank entries and whose $(a, i)^{\text{th}}$ entry X_{ai} is the prediction of user a's rating to movie i. Let D be the set of all (a, i)'s for which a user rating Y_{ai} exists. A naive approach is to minimize the objective function

$$J(X) = \sum_{(a,i) \in D} \frac{1}{2} \left(Y_{ai} - X_{ai} \right)^2 + \frac{\lambda}{2} \sum_{(a,i)} X_{ai}^2.$$

The results are

$$\widehat{X}_{ai} = \frac{Y_{ai}}{1 + \lambda} \quad \text{for } (a, i) \in D$$
$$\widehat{X}_{ai} = 0 \quad \text{for } (a, i) \notin D.$$

The problem with this approach is that there is no connection between the entries of X. We can impose additional constraint on X:

$$X = UV^\intercal$$

for some $n \times d$ matrix U and $d \times m$ matrix $V^\intercal$, where d is the *rank* of the matrix X.

Alternating Minimization Assume that U and V are rank k matrices. Then, we can write the objective function as

$$J(X) = \sum_{(a,i) \in D} \frac{1}{2} \left(Y_{ai} - \left[UV^\intercal \right]_{ai} \right)^2 + \frac{\lambda}{2} \left(\sum_{a,k} U_{ak}^2 + \sum_{i,k} V_{ik}^2 \right).$$

To find the solution, we fix (initialize) U (or V) and minimize the objective with respect to V (or U). We plug-in the result back to the objective and minimize it with respect to U (or V). We repeat this alternating process until there is no change in the objective function.

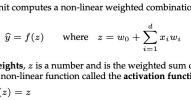
Example Consider the case $k = 1$. Then, $U_{a1} = u_a$ and $V_{i1} = v_i$. If we initialize u_a to some values, then we have to optimize the function

$$\sum_{(a,i) \in D} \frac{1}{2} \left(Y_{ai} - u_a v_i \right)^2 + \frac{\lambda}{2} \sum_i v_i^2.$$

Neural Networks

Introduction to Feedforward Neural Networks

A Unit in a Neural Network A **neural network unit** is a primitive neural network that consists of only the *input layer*, and an output layer with only one output.



A neural network unit computes a non-linear weighted combination of its input:

$$\widehat{y} = f(z) \qquad \text{where} \quad z = w_0 + \sum_{i=1}^{d} x_i w_i$$

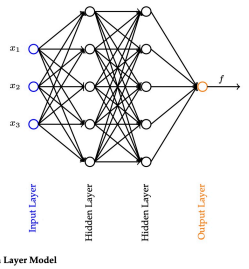
where w_i are the **weights**, z is a number and is the weighted sum of the inputs x_i, and f is generally a non-linear function called the **activation function**.

Linear Function $f(z) = z$

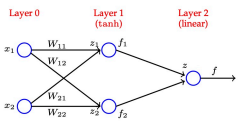
Rectified Linear Unit (ReLU) $f(z) = \max\{0, z\}$

Hyperbolic Tangent Function $\tanh(z) = \frac{e^z + e^{-z}}{e^z + e^{-z}} = 1 - \frac{2}{e^{2z} + 1}$

Deep Neural Networks A **deep (feedforward) neural network** refers to a neural network that contains not only the input and output layers, but also hidden layers in between. Below is a deep feedforward neural network of 2 hidden layers, with each hidden layer consisting of 5 units:

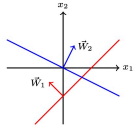


One Hidden Layer Model



$$z_1 = \sum_{j=1}^{2} x_j W_{j1} + W_{01} \qquad z_2 = \sum_{j=1}^{2} x_j W_{j2} + W_{02}$$
$$f_1 = f(z_1) = \tanh(z_1) \qquad f_2 = f(z_2) = \tanh(z_2)$$
$$z = f_1 w_1' + f_2 w_2' \qquad f = f(z) = z$$

Neural Signal Transformation We can visualize what the hidden layer is doing similarly to a linear classifier.



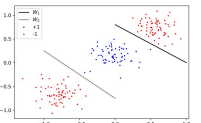
In the figure,

$$\widehat{W}_1 = \begin{pmatrix} W_{11} \\ W_{21} \end{pmatrix} \quad \text{and} \quad \widehat{W}_2 = \begin{pmatrix} W_{12} \\ W_{22} \end{pmatrix}.$$

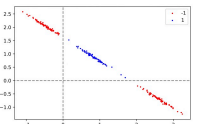
They map the input onto the f_1-f_2 axes.

Hidden Layer Representation

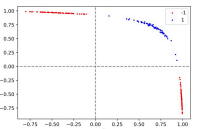
- Hidden Layer Units



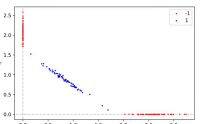
- Linear Activation



- $tanh$ Activation

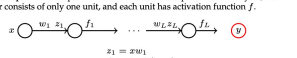


- ReLU Activation



Feedforward Neural Networks, Back Propagation, and Stochastic Gradient Descent (SGD)

Simple Example This simple neural network is made up of L hidden layers, but each layer consists of only one unit, and each unit has activation function f.



$$z_1 = x w_1$$
$$f_1 = \tanh(x w_1)$$
$$\vdots$$
$$f_L = \tanh(f_{L-1} w_L)$$
$$\mathcal{L}(y, f_L) = \text{Loss}\left(y, f_L \right) = \tfrac{1}{2} (y - f_L)^2$$

For $i = 2, \ldots, L$: $z_i = f_{i-1} w_i$ where $f_{i-1} = f(z_{i-1})$. Also, y is the true value and f_L is the output of the neural network.

Gradient Descent The gradient descent update rule for the parameter w_i is

$$w_i \leftarrow w_i - \eta \cdot \nabla_{w_i} \mathcal{L}(y, f_L)$$

where η is the learning rate. For instance, we have

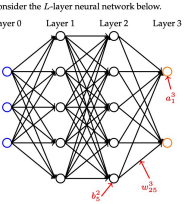
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial f_1}{\partial w_1} \frac{\partial \mathcal{L}}{\partial f_1}$$
$$\frac{\partial f_1}{\partial w_1} = \left[1 - \tanh^2(x w_1) \right] x = \left(1 - f_1^2 \right) x$$
$$\frac{\partial \mathcal{L}}{\partial f_1} = \frac{\partial \mathcal{L}}{\partial f_2} \frac{\partial f_2}{\partial f_1} = \frac{\partial \mathcal{L}}{\partial f_2} \left(1 - f_2^2 \right) w_2.$$

Thus, when we back-propagate, we get

$$\frac{\partial \mathcal{L}}{\partial w_1} = x \left(1 - f_1^2 \right) \cdots \left(1 - f_L^2 \right) w_2 \cdots w_L \cdot 2 \left(f_L - y \right).$$

Note that the above derivation applies to $\tanh$ activation.

Backpropagation Consider the L-layer neural network below.



Summary

- Units in neural networks are linear classifiers, just with different output non-linearity.
- The units in feedforward neural networks are arranged in layers.
- By learning the parameters associated with the hidden layer units, we learn how to represent examples (as hidden layer activations).
- The representations in neural networks are learned directly to facilitate the end-to-end task.
- A simple classifier (output unit) suffices to solve complex classification tasks if it operates on the hidden layer representations.

We have the following notations:

- b_j^ℓ is the bias of the j^{th} neuron in the ℓ^{th} layer.
- a_j^ℓ is the activation of the j^{th} neuron in the ℓ^{th} layer.
- w_{jk}^ℓ is the weight for the connection from the k^{th} neuron in the $(\ell - 1)^{\text{th}}$ layer to the j^{th} neuron in the ℓ^{th} layer.

If the activation function is f and the loss function we are minimizing is C, then the equations describing the network are:

$$a_j^\ell = f\left(\sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell \right)$$
$$\text{Loss} = C\left(a^L \right)$$

Let the weighted inputs to the d neurons in layer ℓ be defined as

$$z^\ell = w^\ell a^{\ell-1} + b^\ell, \quad \text{where } z^\ell \in \mathbb{R}^d.$$

Then, the activation of layer ℓ is also written as $a^\ell = f(z^\ell)$. Also, let $\delta_j^\ell \equiv \frac{\partial C}{\partial z_j^\ell}$ denote the *error* of neuron j in layer ℓ. Then, $\delta^\ell \in \mathbb{R}^d$ denotes the full vector of errors associated with layer ℓ.

Equations of Backpropagation

$$\delta^L = \nabla_a C \odot f'\left(z^L \right)$$
$$\delta^\ell = \left[\left(w^{\ell+1} \right)^\intercal \delta^{\ell+1} \right] \odot f'\left(z^\ell \right)$$
$$\frac{\partial C}{\partial b_j^\ell} = \delta_j^\ell$$
$$\frac{\partial C}{\partial w_{jk}^\ell} = a_k^{\ell-1} \delta_j^\ell$$

The symbol $\odot$ represents the Hadamard product.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \odot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae & bf \\ cg & dh \end{pmatrix}.$$

Recurrent Neural Networks

Temporal/Sequence Problems

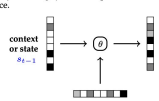
- Sequence prediction problems can be recast in a form amenable to feedforward neural networks.
- We have to engineer how *history* is mapped to a vector (representation). This vector is then fed into, e.g., a neural network.
- We would like to learn how to encode the *history* into a vector.

Key Concepts

- **Encoding** – e.g., mapping a sequence to a vector
- **Decoding** – e.g., mapping a vector to, e.g., a sequence

Example: Encoding Sentences

- Introduce adjustable *lego pieces* and optimize them for end-to-end performance.



$$s_t = \tanh\left(W^{s,s} s_{t-1} + W^{s,x} x_t \right)$$

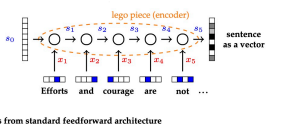
- Let's say we want to encode the incomplete sentence **Efforts and courage are not**. First, we have to represent the first word as a vector (say, a one-hot vector). This will be x_1. Then,

$$s_1 = \tanh\left(W^{s,x} x_1 \right).$$

The second word will be x_2, and we compute for s_2.

$$s_2 = \tanh\left(W^{s,s} s_1 + W^{s,x} x_t \right).$$

We continue this process until we've encoded all the words in the sentence. We can visualize this as follows:



Differences from standard feedforward architecture

- Input is received at each layer (per word), not just at the beginning as in a typical feedforward network.
- The number of layers varies and depends on the length of the sentence.
- Parameters of each layer (representing an application of an RNN) are shared (same RNN at each step).

Basic RNN

$$s_t = \tanh\left(W^{s,s} s_{t-1} + W^{s,x} x_t \right)$$

Simple Gated RNN

$$g_t = \text{sigmoid}\left(W^{g,s} s_{t-1} + W^{g,x} x_t \right)$$
$$s_t = (1 - g_t) \odot s_{t-1} + g_t \odot \tanh\left(W^{s,s} s_{t-1} + W^{s,x} x_t \right)$$

Long Short-Term Memory (LSTM)

$$f_t = \text{sigmoid}\left(W^{f,h} h_{t-1} + W^{f,x} x_t \right) \quad \text{forget gate}$$
$$i_t = \text{sigmoid}\left(W^{i,h} h_{t-1} + W^{i,x} x_t \right) \quad \text{input gate}$$
$$o_t = \text{sigmoid}\left(W^{o,h} h_{t-1} + W^{o,x} x_t \right) \quad \text{output gate}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh\left(W^{c,h} h_{t-1} + W^{c,x} x_t \right) \quad \text{memory cell}$$
$$h_t = o_t \odot \tanh(c_t) \quad \text{visible state}$$

Markov Language Models Let $w \in V$ denote the set of possible words/symbols that includes

- an **UNK** symbol for any unknown word (out of vocabulary)
- ⟨**bog**⟩ symbol for specifying the start of a sentence
- ⟨**end**⟩ symbol for specifying the end of the sentence

First-order Markov Model In a first-order Markov model (**bigram model**), the next symbol only depends on the previous one. Each symbol (except ⟨beg⟩) in the sequence is predicted using the same condition probability table until an ⟨end⟩ symbol is seen. The probability associated to the sentence is

$$\prod_{i=1} \mathbb{P}\left(w_i|w_{i-1}\right).$$

Maximum Likelihood Estimation The goal is to maximize the probability that the model can generate all the observed sentences (corpus S)

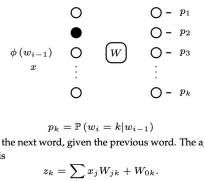
$$s \in S, s = \left\{w_1^s, w_2^s, \ldots, w_{|s|}^s\right\}$$

$$\ell = \log\left\{\prod_{s\in S}\left[\prod_{i=1}^{|s|}\mathbb{P}\left(w_i^s|w_{i-1}^s\right)\right]\right\}$$

The maximum likelihood estimate is obtained as normalized counts of successive word occurrences (matching statistics)

$$\widehat{\mathbb{P}}\left(w'|w\right) = \frac{\text{count}\left(w', w\right)}{\sum_{\widehat{w}}\text{count}\left(w, \widehat{w}\right)}$$

Feature-based Markov Model We can also represent the Markov model as a feedforward neural network (very extendable). We define a one-hot vector, $\phi\left(w_{i-1}\right)$, corresponding to the previous word. This will be an input to the feedforward neural network.



In the figure,

$$p_k = \mathbb{P}\left(w_i = k|w_{i-1}\right)$$

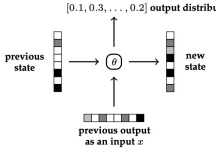
is the probability of the next word, given the previous word. The aggregate input to the k^{th} output unit is

$$z_k = \sum_k x_j W_{jk} + W_{0k}.$$

These input values are not probabilities. A typical transformation is the **softmax transformation**:

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}.$$

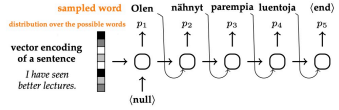
RNNs for Sequences Our RNN now also produces an output (e.g., a word) as well as update its state



$$s_t = \tanh\left(W^{s,s}s_{t-1} + W^{s,x}x_t\right) \quad \text{state}$$

$$p_t = \text{softmax}\left(W^o s_t\right) \quad \text{output distribution}$$

Decoding



Convolutional Neural Networks

Problem Image classification

- The presence of objects may vary in location across different images.

Patch classifier/filter

The patch classifier goes through the entire image. We can think of the weights as the image that the unit prefers to see.

Convolution The convolution is an operation between two functions f and g:

$$(f * g)(t) \equiv \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau.$$

Intuitively, convolution *blends* the two functions f and g by expressing the amount of overlap of one function as it is shifted over another function.

Discrete Convolution For discrete functions, we can define the convolution as

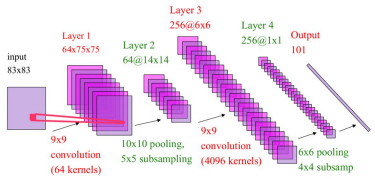
$$(f * g)[n] \equiv \sum_{m=-\infty}^{m=+\infty} f[m]g[n-m].$$

Pooling We wish to know whether a feature was there but not exactly where it was.

Pooling (Max) Pooling region and *stride* may vary.

- Pooling induces translation invariance at the cost of spatial resolution.
- Stride reduces the size of the resulting feature map.

Example of CNN From LeCun (2013)



Unsupervised Learning

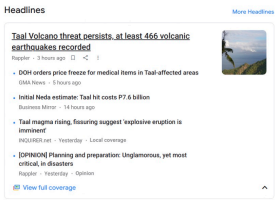
Clustering

Training set We are provided a training set but with no labels

$$S_n = \left\{x^{(i)}\middle| i = 1, \ldots, n\right\}$$

and the goal is to find structure in the data.

Example: Google News



Example: Image Quantization

Partition A partition of a set is a grouping of the set's elements into non-empty subsets, in such a way that **every** element is included in one and only one of the subsets. In other words, $C_1, \ldots, C_K$ is a partition of $\{1, \ldots, n\}$ if and only if

$$C_1 \cup \cdots \cup C_K = \{1, \ldots, n\} \quad \text{and}$$

$$C_i \cap C_j = \emptyset \quad \text{for any} \quad i \neq j \quad \text{in} \quad \{1, \ldots, K\}.$$

Clustering: Input

- Set of feature vectors $S_n = \left\{x^{(i)}\middle| i = 1, \ldots, n\right\}$
- The number of clusters K

Clustering: Output

- A partition of indices $\{1, \ldots, n\}$ into K sets, $C_1, \ldots, C_K$
- *Representatives* in each of the K partition sets, given as $z_1, \ldots, z_K$

Cost We can calculate the total cost by summing the cost of each cluster:

$$\text{Cost}\left(C_1, \ldots, C_K\right) = \sum_{j=1}^{K}\text{Cost}\left(C_j\right)$$

Similarity Measure We use the Euclidean distance between the elements of a cluster and its representative to calculate the cost for each cluster. Then, the total cost is

$$\text{Cost}\left(C_1, \ldots, C_K, z_1, \ldots, z_K\right) = \sum_{j=1}^{K}\sum_{i\in C_j}\left\|x^{(i)} - z_j\right\|^2.$$

***K*-Means Algorithm**

1. Randomly select $z_1, \ldots, z_K$.
2. Iterate:
   (a) Given $z_1, \ldots, z_K$, assign each data point $x^{(i)}$ to the closest z_j so that

$$\text{Cost}\left(z_1, \ldots, z_K\right) = \sum_{i=1}^{n}\min_{j=1,\ldots,K}\left\|x^{(i)} - z_j\right\|^2.$$

   (b) Given $C_1, \ldots, C_K$, find the best representatives $z_1, \ldots, z_K$, i.e., find $z_1, \ldots, z_K$ such that

$$z_j = \underset{z}{\text{argmin}}\sum_{i\in C_j}\left\|x^{(i)} - z\right\|^2 = \frac{1}{|C_j|}\sum_{i\in C_j}x^{(i)}.$$

***K*-Medoids Algorithm** The K-Medoids algorithm is a variation of the K-Means algorithm that addresses some of the K-Means algorithm's limitations.

1. Randomly select $\{z_1, \ldots, z_K\} \subseteq \{x_1, \ldots, x_n\}$.
2. Iterate:
   (a) Given $z_1, \ldots, z_K$, assign each $x^{(i)}$ to the closest z_j so that

$$\text{Cost}\left(z_1, \ldots, z_K\right) = \sum_{i=1}^{n}\min_{j=1,\ldots,K}\text{dist}\left(x^{(i)}, z_j\right)$$

   (b) Given $C_j \in \{C_1, \ldots, C_K\}$, find the best representative $z_j \in \{x_1, \ldots, x_n\}$ such that

$$\sum_{x^{(i)}\in C_j}\text{dist}\left(x^{(i)}, z_j\right)$$

   is minimal.

Generative Models

Generative vs. Discriminative Models *Generative models* work by explicitly modeling the probability distribution of each of the individual classes in the training data. *Discriminative models* learn explicit decision boundary between classes.

Simple Multinomial Generative Model Consider a multinomial model M to generate text documents. Assume that M has a fixed vocabulary W and we generate a document by sampling one word at a time from this vocabulary. Furthermore, all the words that are generated by M are independent of each other. We denote the probability that M generates certain word $w \in W$ is

$$\mathbb{P}\left(w|\theta\right) = \theta_w, \quad \theta_w \geq 0, \sum_{w\in W}\theta_w = 1.$$

Then, the probability of generating the document D is

$$\mathbb{P}\left(D|\theta\right) = \prod_{i=1}^{n}\theta_{w_i} = \prod_{w\in W}\theta_w^{\text{count}(w)}.$$

Maximum Likelihood Estimate The log-likelihood for the model is

$$\ell = \log\mathbb{P}(D|\theta) = \sum_{w\in W}\text{count}(w)\log\theta_w$$

and the maximum likelihood estimate is

$$\widehat{\theta}_w = \frac{\text{count}(w)}{\sum_{w'\in W}\text{count}(w')}.$$

Prediction Consider using a multinomial generative model M for the task of binary classification consisting of two classes: $+$ (positive class) and $-$ (negative class).

- θ^+: parameter for the positive class
- θ^-: parameter for the negative class

Suppose that we classify a new document D to belong to the positive class if and only if

$$\log\frac{\mathbb{P}(D|\theta^+)}{\mathbb{P}(D|\theta^-)} \geq 0.$$

The generative classifier is equivalent to a linear classifier:

$$\log\frac{\mathbb{P}(D|\theta^+)}{\mathbb{P}(D|\theta^-)} = \sum_{w\in W}\text{count}(w)\log\frac{\theta_w^+}{\theta_w^-} = \sum_{w\in W}\text{count}(w)\,\theta_w'.$$

Prior, Posterior and Likelihood In the above discussion, there is an assumption that the likelihood of being in one of the classes is the same. However, we may have some prior knowledge and we want to incorporate it into our model. The posterior distribution for the positive class is then

$$\mathbb{P}\left(y = +|D\right) = \frac{\mathbb{P}(D|\theta^+)\,\mathbb{P}(y=+)}{\mathbb{P}(D)}.$$

The generative classifier becomes

$$\log\frac{\mathbb{P}\left(y=+|D\right)}{\mathbb{P}\left(y=-|D\right)} = \sum_{w\in W}\text{count}(w)\,\theta_w' + \theta_0'$$

where $\theta_w' = \log\frac{\theta_w^+}{\theta_w^-}$ and $\theta_0' = \log\frac{\mathbb{P}(y=+)}{\mathbb{P}(y=-)}$.

Gaussian Generative Models The likelihood of $\mathbf{x} \in \mathbb{R}^d$ being generated by a Gaussian with mean μ and standard deviation σ is

$$f_X\left(\mathbf{x}|\mu, \sigma^2\right) = \frac{1}{\left(2\pi\sigma^2\right)^{d/2}}\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mu\|^2\right).$$

MLE for the Mean

$$\widehat{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}$$

MLE for the Variance

$$\widehat{\sigma}^2 = \frac{1}{nd}\sum_{i=1}^{n}\left\|\mathbf{x}^{(i)} - \mu\right\|^2$$

Mixture Models; EM Algorithm

Gaussian Mixture Models Instead of just a single Gaussian, we have a mixture of Gaussian components. Assume that there are K Gaussians with known means and variances. Assume also that the mixture weights $p_1, \ldots, p_K$ are known. The likelihood for an observation $\mathbf{x}$ obtained from the model is

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{K}p_j\mathcal{N}\left(\mathbf{x}; \mu^{(j)}, \sigma_j^2\mathbf{I}\right).$$

For the training set

$$S_n = \left\{\mathbf{x}^{(i)}, i = 1, \ldots, n\right\},$$

the likelihood is

$$\mathbb{P}\left(S_n|\theta\right) = \prod_{i=1}^{n}\sum_{j=1}^{K}p_j\mathcal{N}\left(\mathbf{x}^{(i)}; \mu^{(j)}, \sigma_j^2\mathbf{I}\right).$$

Observed Case Consider the case of hard clustering, i.e., a point either belongs to a cluster or not. Let

$$\delta(j|i) = \begin{cases}1, & \mathbf{x}^{(i)} \text{ is assigned to } j\\ 0, & \text{otherwise.}\end{cases}$$

Also, let $\widehat{n}_j = \sum_{i=1}^{n}\delta(j|i)$ denote the number of points belonging to cluster j.

Maximizing the likelihood gives

$$\widehat{p}_j = \frac{\widehat{n}_j}{n}$$

$$\widehat{\mu}^{(j)} = \frac{1}{\widehat{n}_j}\sum_{i=1}^{n}\delta(j|i)\,\mathbf{x}^{(i)}$$

$$\widehat{\sigma}_j^2 = \frac{1}{\widehat{n}_j d}\sum_{i=1}^{n}\delta(j|i)\left\|\mathbf{x}^{(i)} - \mu^{(j)}\right\|^2.$$

The EM Algorithm Instead of hard clustering, the data can actually be generated from different clusters with different probabilities. We have soft clustering. We can maximize the likelihood through the EM algorithm.

Randomly initialize θ: $\mu^{(1)}, \ldots, \mu^{(K)}, \sigma_1^2, \ldots, \sigma_K^2, p_1, \ldots, p_K$.

1. **E-step:**

$$p(j|i) = \frac{p_j\mathcal{N}\left(\mathbf{x}^{(i)}; \mu^{(j)}, \sigma_j^2\mathbf{I}\right)}{p\left(\mathbf{x}|\theta\right)}$$

   where $p(\mathbf{x}|\theta) = \sum_{j=1}^{K}p_j\mathcal{N}\left(\mathbf{x}^{(i)}; \mu^{(j)}, \sigma_j^2\mathbf{I}\right)$

2. **M-step:**

$$\widehat{n}_j = \sum_{i=1}^{n}p(j|i)$$

$$\widehat{p}_j = \frac{\widehat{n}_j}{n}$$

$$\widehat{\mu}^{(j)} = \frac{1}{\widehat{n}_j}\sum_{i=1}^{n}p(j|i)\,\mathbf{x}^{(i)}$$

$$\widehat{\sigma}_j^2 = \frac{1}{\widehat{n}_j d}\sum_{i=1}^{n}p(j|i)\left\|\mathbf{x}^{(i)} - \mu^{(j)}\right\|^2.$$

Reinforcement Learning

Objectives of RL The goal of RL is to learn a good policy with no or limited supervision.

Markov Decision Processes

Definition A **Markov decision process (MDP)** is defined by

- a set of **states** $s \in S$ (may be observed or unobserved);
- a set of **actions** $a \in A$;
- action-dependent **transition probabilities** $T(s, a, s') = \mathbb{P}(s'|s, a)$ so that, for each state s and action a,

$$\sum_{s'\in S}T(s, a, s') = 1$$

.

- **reward functions** $R(s, a, s')$, representing the reward for starting in state s, taking action a and ending up in state s' after one step. (The reward function may also depend only on s, or only s and s.)

Property MDPs satisfy the **Markov property** in that the transition probabilities and rewards depend only on the current state and action, and remain unchanged regardless of the history (i.e., past states and actions) that leads to the current state.

Utility Function The main problem for MDPs is to optimize the agent's behavior. We first need to specify the criterion that we are trying to maximize in terms of accumulated rewards. We define a **utility function** and maximize its expectation.

- **Finite horizon based utility:** The utility function is the sum of rewards after acting for a fixed number n of steps. When the rewards depend only on the states, the utility function is

$$U[s_0, s_1, \ldots, s_n] = \sum_{i=0}^{n}R(s_i.)$$

- **(Infinite horizon) discounted reward based utility:** In this setting, the reward one step into the future is discounted by a factor γ, the reward two steps ahead by γ^2, and so on. The goal is to continue acting (without an end) while maximizing the expected discounted reward. The discounting allows us to focus on near term rewards, and control this focus by changing γ. If the rewards depend only on the states, the utility function is

$$U[s_0, s_1, \ldots] = \sum_{k=0}^{\infty}\gamma^k R(s_k).$$

Optimal Policy A **policy** is a function $\pi : S \to A$ that assigns an action $\pi(s)$ to any state s. Given an MDP and a utility function $U[s_0, s_1, \ldots, s_n]$, our goal is to find an **optimal policy** function that maximizes the expectation of the utility. We denote the optimal policy by π^*.

Bellman Equations

Value Function Denote by $Q^*(s, a)$ the expected reward starting at s, taking action a and acting optimally. The *value function* $V^*(s)$ is the expected reward starting at state s and acting optimally.

The Bellman Equations These equations connect the notion of the value of a state and the value of policy.

$$V^*(s) = \max_a Q^*(s, a) = Q^*(s, \pi^*(s))$$

$$Q^*(s, a) = \sum_{s'}T(s, a, s')\left[R(s, a, s') + \gamma V^*(s')\right]$$

We can define the $V^*(s)$ recursively to get

$$V^*(s) = \max_a\left\{\sum_{s'}T(s, a, s')\left[R(s, a, s') + \gamma V^*(s')\right]\right\}$$

Value Iteration Algorithm

Definition *Value iteration* is an iterative algorithm that computes the values of states indexed by k. Let $V_k^*(s)$ be the expected reward from state s after k steps:

$$V_k^*(s) \to V^*(s) \quad \text{as} \quad k \to \infty.$$

1. Initialization: $V_0^*(s) = 0$
2. Iterate until $V_k^*(s) \simeq V_{k+1}^*(s)\ \forall s$

$$V_{k+1}^*(s) \leftarrow \max_a\left\{\sum_{s'}T(s, a, s')\left[R(s, a, s') + \gamma V_k^*(s')\right]\right\}$$

3. Compute $Q^*(s, a)$ and $\pi^*(s, a) = \underset{a}{\text{argmax}}\,Q^*(s, a)$.

Convergence This algorithm will converge as long as $\gamma < 1$.

Q-Value Iteration

Definition We can directly operate at the level of Q-values. Q-value iteration is a reformulation of value iteration algorithm.

Update Rule

$$Q_{k+1}^*(s, a) \leftarrow \sum_{s'}T(s, a, s')\left[R(s, a, s') + \gamma\max_{a'}Q_k^*(s', a')\right]$$

Reinforcement Learning

MDP vs. RL In MDPs, we are given 4 quantities $\langle S, A, T, R\rangle$. In reinforcement learning, we are given only the states and actions $\langle S, A\rangle$. In the real world, transitions and rewards might not be directly available and they need to be estimated.

Estimation Consider a random variable X. The goal is to estimate

$$\mathbb{E}\left[f(X)\right] = \sum_x p(x)f(x).$$

We have access to K samples: $x_i, i = 1, \ldots, K$.

Model-based Learning

$$\widehat{p}(x_i) = \frac{1}{K}\text{count}(x_i)$$

$$\mathbb{E}\left[f(X)\right] \approx \sum_{i=1}^{K}\widehat{p}(x_i)f(x_i)$$

Model-free Learning

$$\mathbb{E}\left[f(X)\right] \approx \frac{1}{K}\sum_{i=1}^{K}f(x_i)$$

Q-Value Iteration for RL

1. Initialization: $Q(s, a) = 0\ \forall s, a$
2. Iterate until convergence:
   (a) Collect sample: $s, a, s', R(s, a, s')$
   (b) Update:

$$Q_{i+1}(s, a) \leftarrow \alpha\left[R(s, a, s') + \gamma\max_{a'}Q_i(s', a')\right] + (1-\alpha)\,Q_i(s, a)$$

$$= Q_i(s, a) + \alpha\left[R(s, a, s') + \gamma\max_{a'}Q_i(s', a') - Q_i(s, a)\right]$$

Recommended Resources

- Introduction to Machine Learning with Python (Müller and Guido)
- Machine Learning with Python – From Linear Models to Deep Learning [Lecture Slides] (`http://www.edx.org`)
- LaTeX File (`github.com/mynameisjanus/686xMachineLearning`)

Please share this cheatsheet with friends!