Introduction to Machine Learning, John Lafferty

Made by: Nokkvi Dan Ellidason, MS in Statistics and Data Science

18 December 2021

Contents

1 Introduction **4**

2 Regression **4**
2.1 Regression Theory: . . . 5
2.2 Measuring performance . . . 5
2.3 Linear Regression - why start here? . . . 5
2.4 Sums of squares and R^2 . . . 6
2.5 Multiple linear regression . . . 6

3 Classification **6**
3.1 Important concepts before blast off . . . 6
3.2 Optimal classification rule: (Important) . . . 6
3.3 Two flavours of classifiers . . . 7
3.4 Logistic Regression . . . 7
3.4.1 Discriminative Model - Logistic Regression . . . 7
3.4.2 Fitting a logistic regression . . . 8
3.4.3 Extension to more than 2 classes . . . 8
3.5 Gaussian Discriminant Analysis: . . . 8
3.5.1 Quadrarric Discriminant Analysis: . . . 8
3.5.2 Linear Discriminant Analysis: . . . 9
3.5.3 Separable classes . . . 9
3.5.4 Loss function for 3 classses . . . 10
3.6 Regularization . . . 10

4 Stochastic Gradient Descent **10**
4.1 SDG for general loss . . . 11
4.2 Batch vs. SGD . . . 11
4.3 SGD for logistic regression . . . 12
4.4 Choice of learning rate . . . 12
4.5 Regularization for SGD . . . 12

5 Bias-Variance Tradeoff and Cross Validation **13**
5.1 Example: Regularization . . . 13
5.2 Model Selection . . . 14
5.3 Cross-Validation . . . 14
5.3.1 Leave-One-Out Cross-Validation . . . 15
5.3.2 k-fold Cross-Validation . . . 15
5.3.3 A slick shortcut for the LOOCV . . . 15
5.3.4 Model selection . . . 15

6 Trees **15**
6.1 Tree building and Predicting . . . 16

6.2 Tree growth . . . 16
6.2.1 Regression: . . . 16
6.2.2 Classification: . . . 17
6.3 Model Selection, Bias vs. Variance: . . . 18
6.3.1 Cost-Complexity pruning: . . . 18
6.3.2 Cross-validation pruning: . . . 18
6.4 Issues with trees . . . 18

7 Ensemble methods of Trees **18**
7.1 Bagging - reduce variance by averaging over many trees . . . 18
7.2 Random Forests - decorrelate by random sampling of predictors . . . 19
7.2.1 OOB prediction . . . 19

8 Principal Component Analysis (PCA) **19**
8.0.1 PCA Intro, not a generative model . . . 20
8.0.2 Using PCA for classification or regression . . . 20

9 Language Models **20**
9.1 Reducing & Estimating Parameters . . . 21
9.1.1 Reducing . . . 21
9.1.2 Estimating . . . 21
9.1.3 Interpolation . . . 21
9.2 Estimating Language Models Predictive Power - Perplexity . . . 22
9.3 PCA - Pointwise mutual information (PMI) . . . 22

10 Word Embeddings **22**
10.1 Word2Vec - Constructing embeddings . . . 23
10.1.1 SGD for Word2Vec . . . 23
10.2 GloVe - Constructing embeddings . . . 23
10.2.1 SGD for GloVe . . . 24
10.3 t-SNE - Visualizing Embeddings . . . 24
10.3.1 Idea of t-SNE . . . 24

11 Mixtures **24**
11.1 Bayesian Inference . . . 25
11.1.1 Bayes Theorem . . . 25
11.1.2 Dirichlect . . . 26
11.2 Hierarchical models . . . 26
11.2.1 Binomial . . . 26
11.2.2 Gaussian . . . 27
11.2.3 Shrinkage . . . 27

12 Bayes and Topic Models **28**
12.1 Introduction . . . 28
12.2 Constructing TM . . . 28
12.2.1 Generative model for Latent Dirichlet allocation (LDA) . . . 29
12.2.2 LDA as a graphical model . . . 29
12.2.3 Posterior Distribution . . . 29
12.2.4 Shortcomings for LDA . . . 30
12.2.5 Markov Chain Monte Carlo . . . 30
12.2.6 Correlated topic model . . . 30
12.3 Modeling Evolution of Topics . . . 30
12.3.1 Time series topic model . . . 30

13 Neural Networks **31**
13.1 Logistic Regression . . . 31
13.2 Word embeddings . . . 31
13.3 Starting with regression: . . . 31

13.3.1 Nonlinearities . . . 31
13.4 Training . . . 32
13.4.1 Example: . . . 33
13.5 Classification . . . 33

14 Autoencoders **34**
14.1 Important aspects . . . 34
14.2 Simple autoencoder example: Single hidden layer . . . 34
14.3 Simple example: Code of dimension K . . . 35
14.4 Adam optimizer . . . 35
14.5 Variants . . . 35

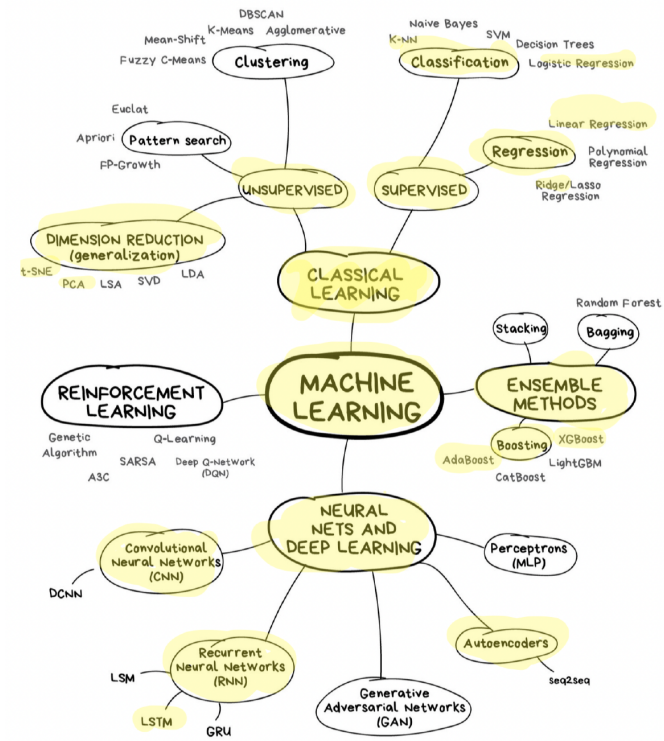
15 Reinforcement Learning **35**
15.1 Formalization . . . 35
15.2 Charectistics . . . 35
15.3 Q learning . . . 36
15.4 Principles of RL . . . 36
15.5 Example: Tic-Tac-Toe . . . 37
15.5.1 Learning strategy . . . 37
15.6 More on Bellman Equation . . . 37
15.6.1 Deterministic case . . . 37
15.6.2 Random environments . . . 38
15.7 Deep Reinforcement Learning . . . 38
15.7.1 Motivation . . . 38
15.8 Some Cool DeepMind RL - VideoGames . . . 39
15.9 Connection to Neuroscience . . . 39
15.10 Summary of RL . . . 40

16 Societal Issues for Machine Learning **40**
16.1 Example of representation learning: Word embeddings . . . 40
16.2 ML at large internet company . . . 40

1 Introduction

These notes were made by Nokkvi Dan Ellidason, a MS student in Statistics and Data Science at Yale University while attending the course Fall 2021. These are Nokkvi's notes made from John Lafferty's slides with an addition of other helpful sources.

Visual Overview of the notes:



What exactly is Machine Learning?

Machine Learning is closely aligned with Statistics, but with a focus on computation, scalability, prediction, representation, and complex problems. Machine learning is for example a leading force in speech recognition, object recognition and scene classification.

AI vs. ML?

Machine learning focuses on making predictions and inferences from data. AI combines machine learning components into a larger system that includes a decision making component. Hence, AI system exhibits a behavior, resulting from the collective decisions that are made.

Supervised Learning vs Unsupervised Learning?

The main difference is that **supervised learning uses labeled data to help predict outcomes**, while the unsupervised learning does not.

Supervised Learning: Given a set (x,y), the algorithm learns to predict y using x.

Unsupervised learning: Given a set of x, the algorithm learns underlying structure or relationships of x.

2 Regression

Regression is a statistical method that attempts to determine the strength and character of the relationship between one dependent variable, Y, and a series of independent variables, $X = x_i$.

What is the difference between Regression and Classification?

- **Regression:** Model income based on other characteristics. $Y = f(X) + \epsilon$

- **Classification:** Model whether someone will earn above the median income based on other characteristics.

What is the difference between prediction and inference?

- **Prediction:** Accurately predict Y for new observations
- **Inference:** Explain the underlying relationship between Y and X.

2.1 Regression Theory:

We model the fixed unknown function to get estimates of Y (noted as $\hat{Y}$) which we do by deriving $\hat{Y} = \hat{f}(X)$. There are many ways to approach regression, we will look at these two:

- **Linear Regression:** Fitting a straight line through the data
- **K-Nearest neighbors regression:** Average together the y_i for x_i close to x.

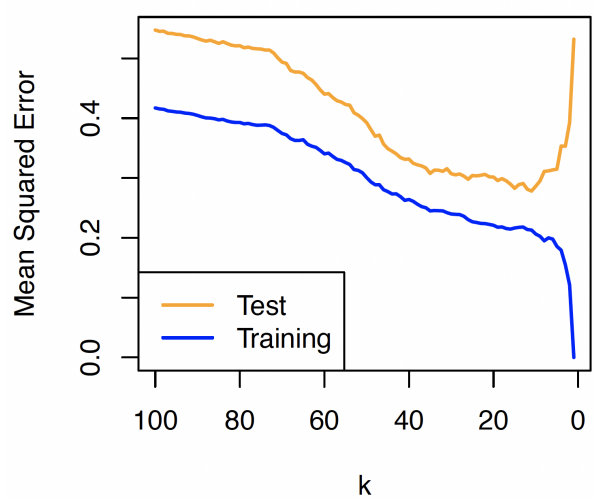
2.2 Measuring performance

The best way to measure performance in regressions is the use of the mean squared error (MSE). MSE tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them. It's called the mean squared error as you're finding the average of a set of errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

If some method, say k-nearest neighbors model w. k = 5 achieves lowest error... ***is it the best one?***

Well it depends, remember we are interested in the test MSE (instead of train MSE). We say a method is overfitting the data if it has small training MSE but large test MSE. It is easy to overfit with very flexible models and algorithms, need to be cautious of that.



2.3 Linear Regression - why start here?

Linear regression is the foundation for more sophisticated topics like regularization and neural networks. Many advanced machine learning methods are generalizations or extensions of linear regression. Linear regression attempts to model the relationship between two variables by fitting a linear equation (a straight line) to the observed data.

- **Fitted values:** For any coefficients $\hat{\beta}_0, \hat{\beta}_1$ we predict fitted values $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$
- **Residual:** $e_i = y_i - \hat{y}_i$ is the difference between the i-th observed value and its fitted value.

The **least squares** approach selects coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the *residual sum of squares* (RSS):

$$RSS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
$$\Rightarrow RSS(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n} (y_1 - \beta_0 - \beta_1 x_1)^2 + ... + (y_n - \beta_0 - \beta_1 x_n)^2$$

Least squares coefficients correspond to minimum of quadratic surface (orthogonal surface).

2.4 Sums of squares and R^2

$$\sum (y_i - \bar{y})^2 = \sum (\hat{y}_i - \bar{y})^2 + \sum (y_i - \hat{y}_i)^2$$
$$\text{Total Error} = \text{Explained Error} + \text{Residuals}$$

For least squares linear regression we get R^2 as the proportion of variability in y explained by the model: $R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$. Remember that R^2 is between 0 and 1 and doesn't depend on scale of Y. R^2 is thus a scale-invariant accuracy measure; proportion of variance in Y explained by the model.

2.5 Multiple linear regression

The regression takes the form:

Vector notation:

$$y = \beta_0 + \beta_1 x_i + ... + \beta_p x_p + \epsilon, \text{ where } \epsilon \sim N(0, \sigma^2)$$

Matrix notation:

$$y = X\beta + \epsilon, \text{ where the intercept } \beta_0 \text{ corresponds to a column of all 1s}$$

We estimate multiple linear regression by solving linear system. Recall that $\hat{\beta} = \arg\, min_\beta RSS(\beta)$. Compute derives of $RSS(\beta)$ with respect to β_i and set equal to 0.

The β that minimizes $RSS(\beta)$ satisfies the normal equations: $X^T X\beta = X^T y$ and if the matrix $X^T X$ is invertible, we solve to get: $\hat{\beta} = (X^T X)^{-1} X^T y$

The coefficients are correlations between the variables x_j and the data $y\ldots$ after the variables are "whitened" to become uncorrelated (centered).

3 Classification

3.1 Important concepts before blast off

- Binary classifier h: function from $\chi \to \{0, 1\}$
  - Is linear if there exists a function $H(x) = \beta + \beta^T x \Rightarrow h(x) = \begin{cases} 1 & \text{if } H > 0 \\ 0 & \text{if otherwise} \end{cases}$
- $H(x)$ is also called a **linear discriminant function**.
  - Decision boundary: $\{x \in \mathbb{R}^d : H(x) = 0\}$

Classification risk (error rate of h): $R(h) = P(Y! = h(X))$

Empirical classification error (training error): $\hat{R}(h) = \frac{1}{n} \sum_i^n I(h(x_i)! = y_i)$

3.2 Optimal classification rule: (Important)

Theorem: The rule h that minimizes R(h) is the **Bayes Rule**:

$$h^*(x) = \begin{cases} 1 \text{ if } m(x) > 0.5, \\ 0 \text{ otherwise} \end{cases}$$

Where $m(x)$ is the regression function:

$$\begin{aligned} m(x) = E(Y|X=x) &= P(Y=1|X=x) \\ &= \frac{\pi_1 p_1(x)}{\pi_1 p_1(x) + (1-\pi_1)p_0(x)} \end{aligned}$$

Remember $p_1 = P(Y=1)$, so $m(x) > 0.5 = \frac{p_1(x)}{p_0(x)} > \frac{1-\pi_1}{\pi_1}$

Thus the Bayes Rule can be rewritten as:

$$h^*(x) = \begin{cases} 1 \text{ if } \frac{p_1(x)}{p_0(x)} > \frac{1-\pi_1}{\pi_1}, \\ 0 \text{ otherwise} \end{cases}$$

- The risk $R^* = R(h^*)$ of the Bayes rule is called the **Bayes risk**.
- The set $\{x \in \chi : m(x) = 0.5\}$ is the **Bayes decision boundary**.

3.3 Two flavours of classifiers

- **Generative models** model both the input X and the output Y:
  - We build a model of the inputs x and the outputs y. In the generative case we typically estimate the joint distribution by maximizing the **joint likelihood**: p(x,y)
- **Discriminative models** model only the output Y given X:
  - In discriminative case we are on concerned about the conditional distribution of the output given the input. We typically estimate by maximizing the **conditional likelihood**

Which one is better?

The form of Bayes classification rule suggests we should use a **generative model**: $m_\theta(x) = P(Y=1|X=x) = \frac{\pi_1 p_{\theta_1,1}(x)}{(1-\pi_1)p_{\theta_0,0}(x) + pi_1 p_{\theta_1,1}(x)}$ Given an estimator $(\hat{\theta_n}, \hat{\pi_1})$ define the classification rule: $\hat{h}(x) = 1(m_{\hat{\theta}_n}(x) > 0.5)$ Where 1 is the indicator function which is 1 if its argument is true and zero otherwise.

3.4 Logistic Regression

Now logistic regression is linear regression model of the log odds: $logit(\hat{p}(x)) = \hat{\beta}_0 + \hat{\beta}_1 x$ where p is probability and $logit(p) = log\left(\frac{p}{1-p}\right)$ is the (natural) log odds!

It can easily be transformed back to probability space: $\hat{p}(x) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x}}{1+e^{\hat{\beta}_0 + \hat{\beta}_1 x}} = logistic(x^T\hat{\beta}) = softmax(x^T\hat{\beta})$

3.4.1 Discriminative Model - Logistic Regression

Logistic regression is a discriminative model, because we don't have a model for the inputs X. We only model the conditional probability $p(Y|X)$. Logistic regression is the discriminative version of linear discriminative analysis (the latter is a generative model).

Fitting a logistic regression model:

- We maximize conditional likelihood. There is no closed form.
- Need to iterate
- Standard Approach is equivalent to Newton's algorithm:
  1. Make quadratic approximation
  2. Do a weighted least squares regression
  3. Repeat

Conditional probabilities logistic regression:

$$\begin{aligned} P(Y_i = 1|X = x_i) &= p(x_i) \\ P(Y_i = 0|X = x_i) &= 1 - p(x_i) \end{aligned}$$

The logit **transform** is $logit(p) = log(\frac{p}{1-p})$ and it is monotone and maps $[0,1] \to (-\infty, \infty)$

$$\text{Linear regression has linear Decision boundary if: } \begin{cases} \hat{\beta}_0 + \hat{\beta}_1 x = 0, \\ \frac{\hat{p}}{1-\hat{p}} = 1 \Longleftrightarrow \hat{p} = 0.5 \end{cases}$$

If our goal is to minimize the overall training error rate then we use this rule:

$$\hat{y} = \begin{cases} 1 \text{ if } \hat{p} \geq 0.5, \\ 0 \text{ if } \hat{p} < 0.5 \end{cases}$$

Hence, the decision boundary is given by $\{x : x^T \hat{\beta} = 0\}$.

3.4.2 Fitting a logistic regression

- Likelihood of a single observation (x_i, y_i):

$$L_i(\beta) = p_i^{y_i} \cdot (1-p_i)^{1-y_i} = \left(\frac{e^{x_i^T \beta}}{1+e^{x_i^T \beta}}\right)^{y_i} \cdot \left(1 - \frac{e^{x_i^T \beta}}{1+e^{x_i^T \beta}}\right)^{1-y_i}$$

- Log-likelihood of a single observation:

$$log(L_i(\beta)) = \ell(\beta) = y_i \cdot log\left(\frac{e^{x_i^T \beta}}{1+e^{x_i^T \beta}}\right) + (1-y_i) \cdot log\left(1 - \frac{e^{x_i^T \beta}}{1+e^{x_i^T \beta}}\right)$$

- Aggregate log-likelihood:

$$\ell(\beta) = \sum \left(y_i x_i^T \beta - log(1 + e^{x_i^T \beta})\right)$$

3.4.3 Extension to more than 2 classes

Multinomial logistic regression extends the logistic regression model to K $\geq$ 2 classes

$$log\left(\frac{P(Y=k|X=x)}{P(Y=0|X=x)}\right) = x^T \beta_k, \text{ k = 1,2...,K-1}$$
$$P(Y=k|X=x) = \frac{exp(x^T \beta_k)}{1 + \sum_{I=1}^{K-1} exp(x^T \beta_I)}, \text{ k = 1,2...,K-1}$$
$$P(Y=\cdot|X=x) = softmax(1, exp(x^T \beta_1), ..., exp(x^T \beta_{K-1}))$$

3.5 Gaussian Discriminant Analysis:

GDA, is a method for data classification, generative model, commonly used when data can be approximated with a Normal distribution. We model the inputs x using Normal distribution (Gaussian)

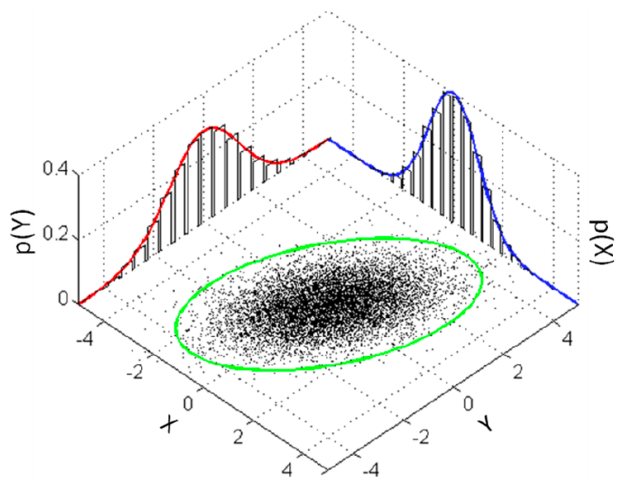
We have two flavors of this ice cream: Linear and Quadratic

3.5.1 Quadrarric Discriminant Analysis:

In the binary (two-class) case, we have two Gaussians:

- $X|y = 1 \sim N(\mu_1, \sum_1)$
- $X|y = 0 \sim N(\mu_0, \sum_0)$

The decision boundary is a quadratic surface (algebra!) To estimate this we just separate the training data according to the two labels and estimate two separate Normal distributions... Easy-Peasy!

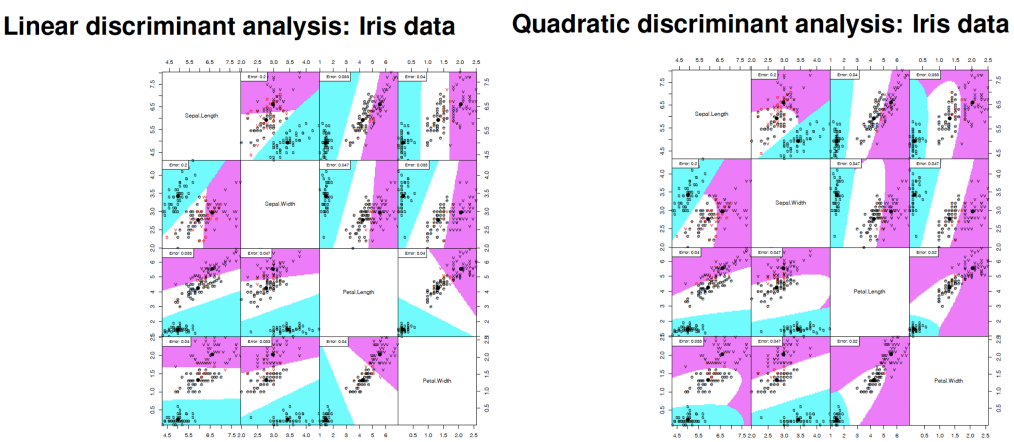


3.5.2 Linear Discriminant Analysis:

In the binary (two-class) case, we have two Gaussians:

- $X|y = 1 \sim N(\mu_1, \sum)$
- $X|y = 0 \sim N(\mu_0, \sum)$

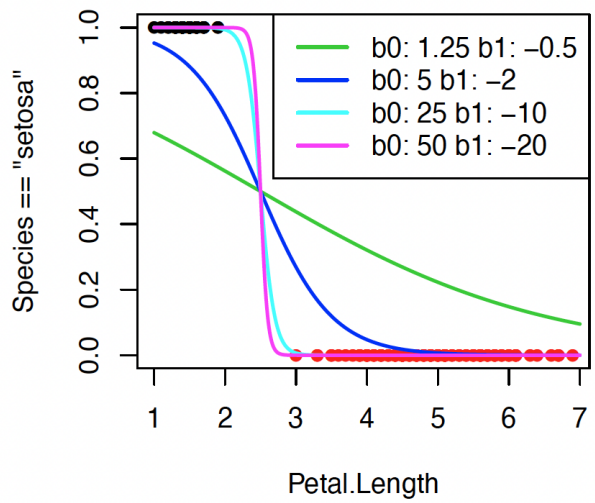
But now we use the same covariance matrix for each. The decision boundary is now linear.



3.5.3 Separable classes

An important consideration related to the likelihood is the separable classes. When we write $x^T\beta$ it usually implies that we have included an intercept: $x^T\beta = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$ and recall that logistic regression fits a linear decision boundary $\{x : x^T\hat{\beta} = 0\}$.

Question: What happens when the two classes are separable, i.e. a hyperplane can perfectly separate out the two classes?



It appears that the predictor is not informative, but it truly is! Theoretically we obtain a perfect fit on the training data. Overfitting is possible and regularization can help.

3.5.4 Loss function for 3 classses

It's a method of evaluating how well specific algorithm models the given data. If predictions deviates too much from actual results, loss function would cough up a very large number. We want to maximize the likelihood of the data, which is equivalent to minimizing the log-likelihood:

$$-\sum_{i=1}^{n} logP(Y = y_i|X = x_i) = \sum_{i=1}^{n} \left\{ log\left(1 + e^{\beta_1^T x_i} + e^{\beta_2^T x_i}\right) - \beta_{y_i}^T x_i \right\}$$

3.6 Regularization

This is a form of regression, that constrains/regularizes/shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.

In the simplest setting, we know that the least squares estimator is $\hat{\beta} = argmin_\beta (y - \beta)^2$

Solution: $\hat{\beta} = y$

Now we **penalize** β from getting to large: $\hat{\beta} = argmin_\beta (y - \beta)^2 + \lambda\beta^2$

Solution: $\hat{\beta} = \frac{y}{1+\lambda}$, as λ gets larger, $\hat{\beta}$ shrinks to zero.

4 Stochastic Gradient Descent

Stochastic gradient descent refers to calculating the derivative from each training data instance and calculating the update immediately. Stochastic gradient descent is a simple algorithm that can be applied to large classification and regression problems where we have a really big dataset continually arriving. In essence we read in the data (few at a time) and update the model for each sample. This is the "go to" algorithm for fitting large or complex machine learning models. Choosing the learning rate is a little tricky

The idea behind it is as follows:

- For each parameter β_j, see what happens to the loss if that parameter is increased a little bit.
- If the loss goes down, then increase β_j proportionately [and vise versa]
- Do this simultaneously for all of the parameters
- Rinse and repeat

Performing SGD we initialize all parameters to zero: $\beta_j = 0, j = 1...p$. Read through the data one record at a time and update the model (for example linear regression):

1. Read data item x
2. make a prediction $\hat{y}(x)$

- Linear regression: $\hat{y}(x) = \sum_{j=1}^{p} \beta_j x_j$

3. Observe the true response y
4. Update the parameters β so $\hat{y}$ is closer to y

The process of changing β_j by a little bit: $\beta_j \to \beta_j + \epsilon$

But what happens to the squared error?

$$\begin{aligned}(y - \hat{y})^2 &\to (y - \hat{y} - \epsilon x_j)^2 \\ &\approx (y - \hat{y})^2 + (-2(y - \hat{y})x_j)\end{aligned}$$

Note: $(-2(y - \hat{y})x_j)$ is derivative of loss function

Use adjustment:

$$\beta_j \to \beta_j - \eta \cdot (-2(y - \hat{y})x_j)$$

Squared error then decreases as: $(y - \hat{y})^2 \approx (y - \hat{y})^2 - 4\eta(y - \hat{y})^2 x_j^2$

4.1 SDG for general loss

Suppose $L(y, \beta^T x)$ is the loss for an input (x,y), e.g. $(y - \beta^T x)^2$

SGD update:

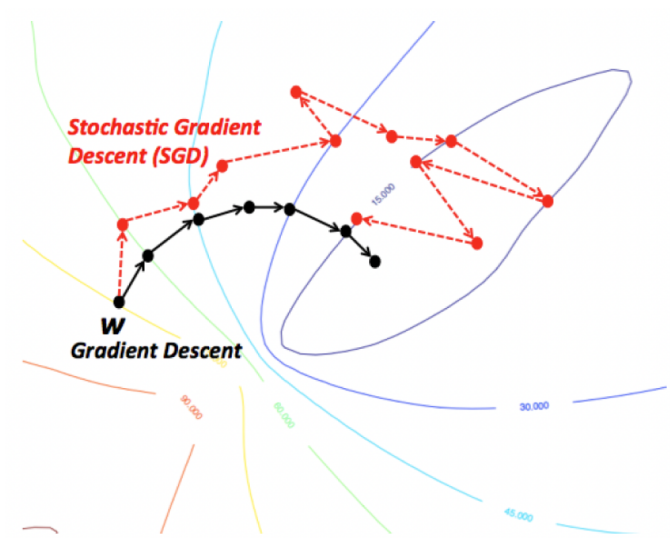
$$\beta_j \longleftarrow \beta_j - \eta \frac{\partial L(y, \beta^T x)}{\partial \beta_j}$$

$$\boldsymbol{\beta} \longleftarrow \boldsymbol{\beta} - \eta \nabla_\beta L(y, \boldsymbol{\beta}^T x) \quad \text{(vector notation)}$$

Batch gradient descent uses the entire training set in each step of gradient descent. Stochastic gradient descent computes a quick approximation to these gradient, using only a single or a small mini batch of data points.

4.2 Batch vs. SGD

- The average derivative over a mini-batch can be thought of as a noisy version of the average derivative over the entire data set.
  - Which can in turn be thought of as a sample estimate of population
- The SGD is computed more cheaply and updating the parameters makes progress more quickly.



https://wikidocs.net/3413

4.3 SGD for logistic regression

SGD Update:

$$\beta_j \leftarrow \beta_j + \eta(y - p(x))x_j$$
$$\beta_j x_j \leftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$
$$p(x) = \frac{1}{1 + exp(-\beta^T x)}$$

Case checking:

- Suppose y = 1 and probability p(x) is high? small change
- Suppose y = 1 and probability p(x) is small? big change upwards
- Suppose y = 0 and probability p(x) is small? small change
- Suppose y = 0 and probability p(x) is high? big change downwards

4.4 Choice of learning rate

A conservative choice of learning rate is: $\eta_t = \frac{1}{t}$ while a more aggressive choice is $\eta_t = \frac{1}{\sqrt{(t)}}$. In practice we try learning rates $\frac{C}{\sqrt{(t)}}$ for different choices of C, and monitor the error:

$$\frac{1}{T}\sum_{t=1}^{T}(Y_t = \hat{Y}_t)^2$$

4.5 Regularization for SGD

A ridge penalty $\frac{1}{2}\lambda\sum_{j=1}^{d}\beta_j^2$ is easily handled. Gradient changes by an additive term $2\lambda\beta_j$. Update becomes:

$$\begin{aligned}\beta_j &\leftarrow \beta_j + \eta(y - p(x))x_j - \lambda\beta_j \\ &= (1 - \eta\lambda)\beta_j x_j + \eta(y - p(x))x_j\end{aligned}$$

Notice that this does the right thing whether β_j wants to be large positive or negative. The penalty thus shrinks β_j towards zero.

5 Bias-Variance Tradeoff and Cross Validation

1. Bias: $\theta - E\hat{\theta}$
2. Variance: $E(\hat{\theta} - E\hat{\theta})^2$

- $\hat{\theta}$ is an estimate from a sample
- E is the expectation/average with respect to the sample
- So $E\hat{\theta}$ is the average estimate
- We can only directly compute $\hat{\theta}$ for the sample we have, so it is a random variable
- **REMBEMBER we don't know θ**

Bias and variance are two sides of the same coin: as the squared bias goes up, the variance goes down (and vise versa)

Risk is essentially expected error and we want to minimize that.

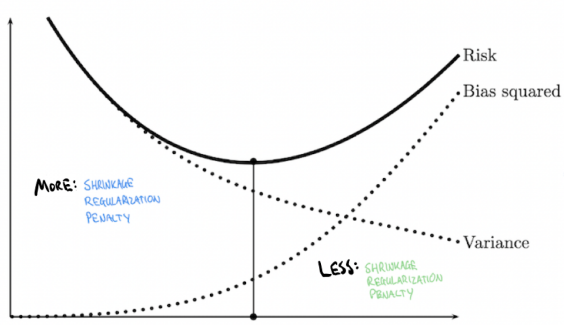
$$Risk = Bias^2 + Variance$$
$$E[\theta - \hat{\theta}]^2 = Bias(\hat{\theta})^2 + Variance(\theta)$$
$$E[\theta - \hat{\theta}]^2 = (\theta - E\hat{\theta})^2 + E(\hat{\theta} - E\hat{\theta})^2$$

Bias-variance tradeoff

Proof:

$$\begin{aligned}
\mathbb{E}(\theta - \widehat{\theta})^2 &= \mathbb{E}(\theta - \mathbb{E}\widehat{\theta} + \mathbb{E}\widehat{\theta} - \widehat{\theta})^2 \\
&= \mathbb{E}(\theta - \mathbb{E}\widehat{\theta})^2 - 2\mathbb{E}\left\{(\theta - \mathbb{E}\widehat{\theta})(\widehat{\theta} - \mathbb{E}\widehat{\theta})\right\} + \mathbb{E}(\widehat{\theta} - \mathbb{E}\widehat{\theta})^2 \\
&= \mathbb{E}(\theta - \mathbb{E}\widehat{\theta})^2 - 2(\theta - \mathbb{E}\widehat{\theta})\mathbb{E}(\widehat{\theta} - \mathbb{E}\widehat{\theta}) + \mathbb{E}(\widehat{\theta} - \mathbb{E}\widehat{\theta})^2 \\
&= \mathbb{E}(\theta - \mathbb{E}\widehat{\theta})^2 + \mathbb{E}(\widehat{\theta} - \mathbb{E}\widehat{\theta})^2 \\
&= \text{Bias}(\widehat{\theta})^2 + \text{Variance}(\widehat{\theta})
\end{aligned}$$

Bias-variance tradeoff



As model complexity increases, squared bias decreases while variance increases

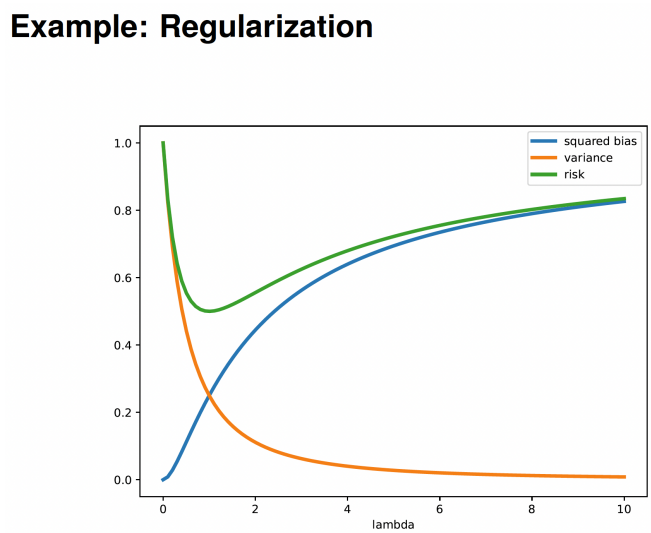
5.1 Example: Regularization

Suppose that we have Y which is a sample and $E(Y) = \theta^*$, where θ^* can be any estimate like the mean of Y. and we estimate the regularized θ^* with shrinkage estimator s.t. $\hat{\theta} = argmin_\theta (Y - \theta)^2 + \lambda\theta^2$

To minimize this we take the derivative and get $\hat{\theta} = \frac{Y}{1+\lambda}$. What are the squared bias and variance? Use the Linearity of expectations to derive this $[E[X + Z] = E[X] + E[Z], E[cX] = cE[X]]$

$$Bias^2 = (\theta^*)^2 \left(\frac{\lambda}{1+\lambda}\right)^2$$
$$Variance = \left(\frac{\lambda}{1+\lambda}\right)^2 \cdot Variance(Y)$$

Hence we get a model that looks something like this where the far right side is the most complex model:



Mean squared error splits into squared bias plus variance.

5.2 Model Selection

For purpose of prediction, **minimizing test error** is priority. Recall our two error metrics for evaluating predictions $\hat{f}(x_i)$:

- Regression: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$

Bias Variance trade-off for regression: $E[(Y - \hat{f}(x))^2] = Var(\hat{f}(x)) + [Bias(\hat{f}(x))]^2 + \sigma^2$

- Variance is the amount of variability in our predictor with different training set. *This increases with increasing model flexibility*
- Bias is the systematic error introduced by model approximation. *This decreases with increasing model flexibility*
- σ^2 is irreducible error, inherent in the error term ϵ. *Cannot get rid of this!*

Need to balance bias and variance for optimal regression.

- Classification: $Err = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{\hat{f}(x_i) \neq y_i\}$

For classification we replace mean squared error by probability of making a mistake. There is no direct decomposition of misclassification error into (squared) bias and variance, but the situation is the same. We can break down the error into **approximation error** (like squared bias) and **estimation error** (like variance). Approximation error results from using a classifier that is too simple and estimation error results from training the classifier on too little data.

5.3 Cross-Validation

Cross-Validation is an intuitive, widely-applicable approach for model assessment and model selection. Cross validation is a practical way of estimating the variability of test error and is used for model selection.

In some sense it is the only way to manage bias-variance trade-off [controversial statment]

Process:

1. Divide dataset randomly into training and a validation set (75% vs. 25%)
2. Fit the model on the training set.
3. Use validation set to obtain estimated test error.
4. Repeat

Validation sets have highly variable validation error and only use a fraction of the training set. The problem with this is that we could use more data.

How do we use more data to train with?

5.3.1 Leave-One-Out Cross-Validation

Leave-one-out CV is the most important version of CV. Has a cool shortcut formula.

- Use a tiny validation set (x_1, y_1)
- Train the rest $\{(x_2, x_2), ..., (x_n, y_n\}$

We are only evaluating the error using a single observation but we can iterate through the dataset, each time using a different (x_i, y_i) as the validation set and obtaining an error MSE_i.

LOOCV estimate of test error is given by: $CV_{(n)} = \frac{1}{n}\sum_i MSE_i$. A single number, no randomness. This is a computationally heavy approach, but very useful for certain models.

5.3.2 k-fold Cross-Validation

This is a potentially faster approach"

- Randomly divide the dataset into k folds.
- For $b = 1, ...k$
  - Use b-th fold (batch) as validation set.
  - Use everything else as training set.
  - Compute validation error on b-th fold

k-fold estimated test error is given by: $CV_{(k)} = \sum_b \frac{n_b}{n} MSE_b$. Where n_b is the total # observations in the b-th fold. and n is the total # observations in the entire dataset.

5.3.3 A slick shortcut for the LOOCV

Suppose the fitted values can be written as $\hat{Y} = HY$ which is a linear smoother where H is a $n \times n$ matrix. This is the case for least squares and regularized regrssion. Then the LOOCV is:

$$
\begin{aligned}
R_{LOOCV} &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{Y}_i}{1 - H_{ii}}\right)^2, \text{ where } H_{ii} \text{ is the ith diagonal entry} \\
H &= X(X^TX)^{-1}X^T \\
&or \\
H &= X(X^TX + \lambda I)^{-1}X^T
\end{aligned}
$$

So no need to fit n regression.

5.3.4 Model selection

So far we have used it to estimate the test error. It's also useful for model selection.

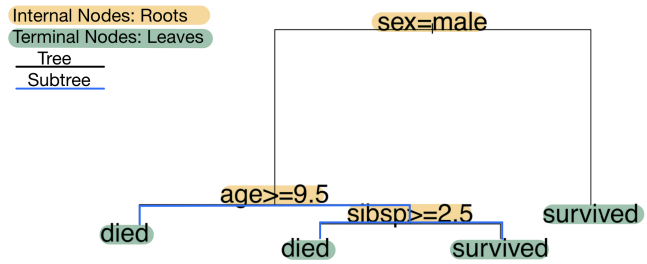
Suppose we are interested in 3 models. We can use leave-one-out cross-validation to estimate the test error for each of these models, and select the model with the lowest test error.

6 Trees

Trees provide ways of modeling nonlinear relationships by carving out rectangular regions in the feature space. Trees give interpretable, nonlinear prediction rules.

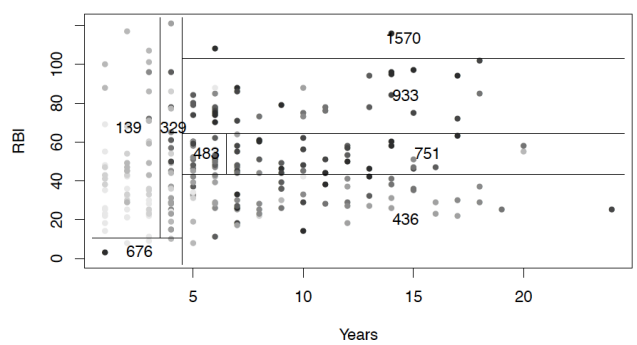
- Response variables can be categorical or quantitative
  - Likewise with the predictors
- Yields a set of interpretable decision rules
- Predictive ability is mediocre, but can be improved by combining multiple trees (resampling, ensemble methods)

Internal nodes are points where the predictor space is split. The internal node at the top of a tree is the root of the tree. Terminal nodes (or leaves) are the ends of the tree where no further splitting occurs. These J regions are denoted as $R_1, ..., R_J$



6.1 Tree building and Predicting

Fitting a tree boils down to identifying the appropriate set of regions $R_1, ..., R_J$ that "best" describes the relationship between X and Y. Tree building takes a greedy approach, i.e. Trees are grown greedily to be full, then pruned back. Trees are essentially the same as k-nearest neighbors method but instead of taking neighbors we split the data up by rectengulars. Prediction accuracy of Trees are not optimal.



We want to choose $R_1, ..., R_J$ to minimize error, (Remember $\bar{y}_{R_j}$ is sample mean in region). This method of minimizing error is very computationally heavy:

$$RSS = \sum_{j=i}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2$$

For predicting via trees we trace each test observation into a leaf R_j based on the sequence of conditions. Predict $\hat{y}_{R_j}$ (it is a function of all training observations i in R_j) for all observations in R_j.

- Regression: $\hat{y}_{R_j} = \bar{y}_{i:i \in R_j}$
- Classification: $\hat{y}_{R_j} =$ most freq occuring class y_i for $i \in R_j$

6.2 Tree growth

6.2.1 Regression:

1. Cycle trough predictors X_k for $k = 1, ..., p$. For each X_k:

- Quantitative X_k: Consider cut points s ($unique(X_k)$) that divide up the region into two parts:
  - $R_1(k, s) = \{i | X_{i,k} < s\}$
  - $R_2(k, s) = \{i | X_{i,k} \geq s\}$

- Evaluate: $Q_k(s) = \sum_{i:i\in R_1(k,s)}(y_i - \bar{y}_{R_1})^2 + \sum_{i:i\in R_2(k,s)}(y_i - \bar{y}_{R_2})^2$. We compute this for all distinct values of y_i.
- Find the value of s that minimizes $Q_k(s)$, call this s_k

2. Find the predictor X_k with the minimum $Q_1(s_1), ..., Q_p(s_p)$. Make the first binary partition along predictor X_k at cut point s_k. Then we repeat this process recursively in all regions.

6.2.2 Classification:

If a predictor X_k is categorical, then cut points are modified. For example, if X_k has unique values in $\{A, B, C, D\}$ then the possible splits are: $\{A\}$ vs. $\{B, C, D\}$, $\{A, C\}$ vs. $\{B, D\}$ and so on. Every possible partition of the set unique values into 2 subsets in considered and again we identify the split producing the lowest resulting RSS (Q_k). This has computational issues as for n categories we have to evaluate 2^n "questions".

$\hat{y_i}$ for all $i \in R_j$ is most commonly class of training obs. in R_j s.t. $\hat{y}_{R_j} = argmax_k \hat{p}_{j,k}$, where $\hat{p}_{j,k}$ is the proportion of training obs in the R_j. We no longer want to minimize RSS, but instead minimize the classification error rate or the Gini Index (which encourages higher node purity):

$$E = \sum_{j=1}^{J} |R_j|(1 - max_k(\hat{p_{j,k}}))$$

$$G = \sum_{j=1}^{J} |R_j| \sum_{k=1}^{K} \hat{p_{j,k}}(1 - \hat{p_{j,k}})$$

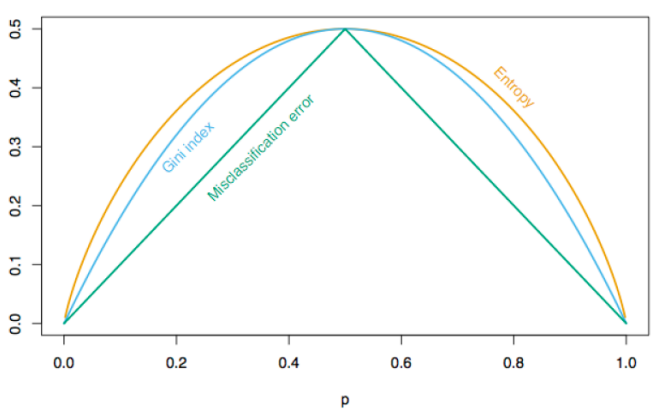
What exactly do we mean by the impurity measure?

We define node proportion of class k as such:

$$\hat{p_{m,k}} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$
$$k(m) = argmax_k(\hat{p_{m,k}})$$

Where we extract:

- Misclassification error: $1 - \hat{p_{m,k}}(m)$
- Gini Index: $\sum_{k=1}^{K} \hat{p_{m,k}}(1 - \hat{p_{m,k}})$
- Entropy: $-\sum_{k=1}^{K} \hat{p_{m,k}}[log(\hat{p_{m,k}})]$



6.3 Model Selection, Bias vs. Variance:

Shallow trees have high bias, low variance while deep trees have low bias, high variance. As the trees grow deeper, the bias decreases but the variance increases... *How do we choose the right size of tree?*

We can introduce a stopping criterion, stop when x, but these result in tuning parameters that are hard to deal with. We essentially pick couple of values for each criteron and cross validate to pick the optimal one.

- Number of obs. in a node has reached a minimum
- Depth of tree has reached a maximum
- Grow until no further splits can reduce RSS by some amount

Another way to get around the overfitting problem is to grow a large tree and then prune it back. Typically, pruning involves looking at subtrees of fully-grown tree, and comparing how well the subtree performs. We prune with the use of cross validation or cost-complexity pruning.

6.3.1 Cost-Complexity pruning:

It is possible to efficiently identify a sequence of nested subtrees that are optimal for sequence of increasing tuning parameter α that controls for the complexity of the model. We thus favour simple trees:

$$C(T) = \sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

$\alpha = 0$ implies the full tree and Larger α implies higher penalty for complexity of model. T is the number of leaves.

6.3.2 Cross-validation pruning:

1. Grow a big tree on training set
2. Obtain a nested set of subtrees $T_I \subset ...T_1 \subset T_0$ corresponding to a sequence of α values.
3. Use K-fold cross-validation to identify the subtree/α that does the best.

6.4 Issues with trees

- Instability. Trees can have high variance. As data change, tree topology can change dramatically, making interpretation difficult
- Lack of smoothness. The splits lead to a "jagged" decision boundary. More of a problem for regression than classification
- Difficulty capturing additive structure, where the regression function is a sum of terms

7 Ensemble methods of Trees

Ensemble methods pool together multiple models to arrive at more reliable predictions.

7.1 Bagging - reduce variance by averaging over many trees

Bagging is essentially regression trees, If we had multiple training sets, could grow multiple trees, then take an average (or majority vote in classification).

Create B bootstrap samples, grow tree (without pruning) using each $\hat{f}^{*1}, ..., \hat{f}^{*B}$. For prediction at x, we take an average:

$$\hat{f_{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

Out-of-bag error estimation is the fact that we can show that each bagged tree uses about $\frac{2}{3}$ of all observations (with repeats). The remaining data, we call, out of bag (OOB) observations can be put to good use.

	Bagging iteration					OOB Est.
Obs	1	2	3	...	B	
1	OOB	train	train	...	train	$\widehat{y}_1$
2	train	OOB	train	...	train	$\widehat{y}_2$
3	train	train	OOB	...	train	$\widehat{y}_3$
4	OOB	train	train	...	OOB	$\widehat{y}_4$
...	...	...	...	...	...	...
n	train	train	OOB	...	train	$\widehat{y}_n$

For each bagged tree, we can make predictions for the OOB observations. At the end we aggregate over all predictions for the i'th obs. to arrive at OOB prediction $\hat{y}_i$. We can compute the prediction error based on these OOB predictions $\hat{y}_i$.

While bagging improves upon the predictive ability of trees, it kills of the interpretability of the model. A good tool for interpreting a bagged tree model (and other tree based ensemble methods like forests) is the variable importance measure. Variable importance can be measured by the amount that RSS (or Gini Index) is reduced due to splits over given predictor averaged over all B trees.

7.2 Random Forests - decorrelate by random sampling of predictors

Similar to bagging, but takes the averaging idea even further by averaging uncorrelated things, which decreases the error. We still use the bootstrap samples, but only use random choses m out of p predictors at each split. Both random forests and bagging grow many trees and average their predictions. Trees are grown deep, to have low bias but high variance. To decorrelate the predictors each tree is grown on a bootstrap sample of the data and/or grown with random subsets of the predictors at each split. Tree growing can be done in parallel.

1. For b=1 to B:

- Draw a bootstrap sample Z^* of size n from the training data
- Grow a random forrest tree T_b to the bootstrapped data, recursively repeating following steps, until minimum node size reached:
  - Select m variables at random from the p variables
  - Pick the best cariable/split-point among the m
  - Split the node into two children nodes

2. Output the ensemble for trees $\{T_b\}_{b=1}^B$

To make predictions at a new point x:

- Regression: Average $\hat{f_{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$
- Classification: Majority vote of the individual trees

Recommended parameters:

- Regression has default value of $m = \frac{p}{3}$ and minimum node size is 5.
- Classification has default value of $m = \sqrt{p}$ and minimum node size is 1.

7.2.1 OOB prediction

For each observation $z_i = (x_i, y_i)$ construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear. Thus, cross-validation can be performed "along the way".

8 Principal Component Analysis (PCA)

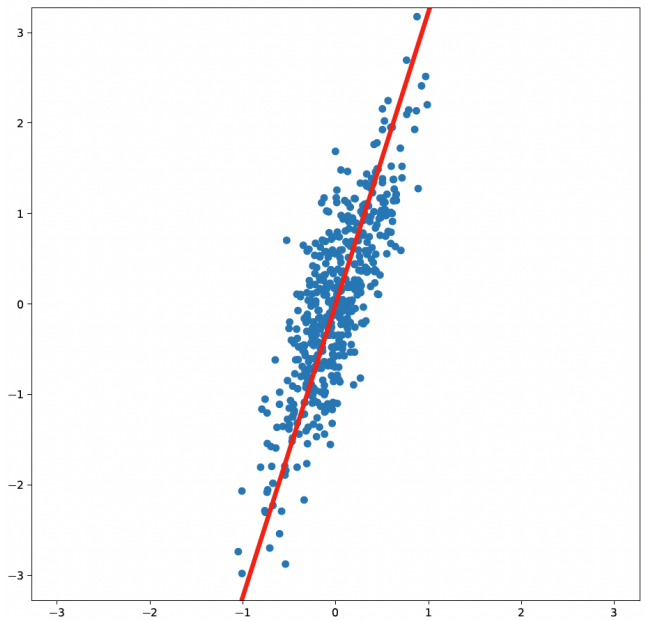
For PCA's we are not trying to predict anything, so what else might we hope to do?

- Are there interesting ways to visualize the data?
- Are there natural subgroups of the data?
- What are the important types of variation in the data?

8.0.1 PCA Intro, not a generative model

PCA is an unsupervised method that finds the directions of greatest variation in the data. The directions are called the principal vectors; weigthtings on the vectors are called the principal components. The first few vectors may be interpretable. Orthogonality makes interpretation difficult for the higher components which can be used for visuallization or dimensionality reduction.

PCA finds the directions of the greatest variability in the data.



1. Center the data: $x_i \to x_i - \frac{1}{n}\sum_{j-1}^{n} x_j = x_i - \bar{x}$
2. Compute the $d \times d$ sample covariance $S = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$ and note that $\frac{1}{n}\sum_{i=1}^{n}(x_{i,j} - \bar{x})^2$ is the sample variance of the jth coordinate of data.
3. Find the first k eigenvectors of S, $\phi_i, ..., \phi_k \in \mathbb{R}^d$, $S\phi_j = \lambda\phi_j$
4. Project the data onto those k vectors: $x_i \to \bar{x} + (\phi_1^T x_i)\phi_1 + ... + (\phi_k^T x_k)\phi_k$

In Python we can compute everything directly except for the eigenvectors.

8.0.2 Using PCA for classification or regression

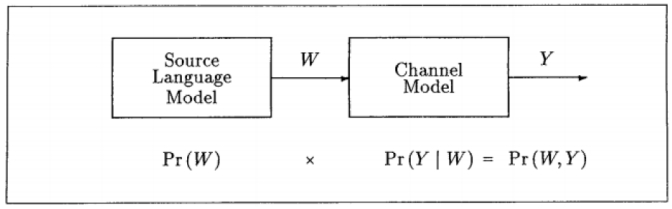
This utilizes PCA as a combination of supervised learning and unsupervised learning. Given data $\{x\}$ extract principal vectors and componenets. Map each data point x_i to its principal components $z_i == (x_i^T\phi_1, ..., x_k^T\phi_k)$ for labeled data $\{(x_i, y_i)\}$. Now train a supervised learning algorithm using the transformed data $\{(z_i, y_i)\}$

9 Language Models

In simple terms, language models are used to predict or generate the next word in sentences. A language model is a way of assigning a probability to any sequence of words (or string of text) $p(w_1, ..., w_n)$, by the basic rules of conditional probability.

$$p(w_1, ..., w_n) = p(w_1)p(w_2|w_1)...p(w_n|w_1, ..., w_{n-1})$$
$$p(\text{Sid is King}) = p(\text{Sid}) \times p(\text{is} \mid \text{Sid}) \times p(\text{King} \mid \text{Sid is})$$

Words are generated one-by-one (text generation). A word is chosen by sampling from a probability distribution and then treated as if it were "real" (think dreaming). The result is purely synthetic text. Speech Recognition, Machine translation, email completion, image captioning are all examples of this framework.



Here above we can think of an example with speech recognition. Think of w as words, Y as acustics of the pronunciation of words. then we have:

$$\text{argmax}_w P(W|Y) = \text{argmax}_w \frac{P(Y|W)P(W)}{P(Y)} = \text{argmax}_w P(W)P(Y|W)$$

The number of histories grows as V^{n-1}. Number of parameters in model grows as V^n, where V is number of words in vocabulary.

9.1 Reducing & Estimating Parameters

9.1.1 Reducing

The best way of reducing parameters is to group by histories. Let $g(w_1, ..., w_n)$ be the group assigned to the history, Then our model becomes:

$$p(w_{n+1}|w_1, ..., w_n) = p(w_{n+1}|g(w_1, ..., w_n))$$

Number of parameters are $O(V \cdot \text{number of groups})\ldots$ What are some example groupings?

- Unigrams: $g(w_1, ..., w_n) = \emptyset \rightarrow$ No params: $O(V)$
- Bigrams: $g(w_1, ..., w_n) = w_n \rightarrow$ No params: $O(V^2)$
- Trigrams: $g(w_1, ..., w_n) = (w_{n-1}, w_n) \rightarrow$ No params: $O(V^3)$

9.1.2 Estimating

The maximum likelihood estimate of a trigram model:

$$\hat{p}(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

There are some problems with this model. "Can have zeroes"

Suppose we have a corpus of 500 million words. I count trigrams and find that 50 million of them are unique. When I see a new trigram, 10% of the time it won't have been seen before (note: this is a typical proportion). This means that the MLE is zero, and the probability that my model predicts the next word will be zero. The MLE is supported on the observed data. We need to spread out the probability over unseen events. Thus it is important to introduce a **shrinkage/smoothing** to this model.

9.1.3 Interpolation

A type of mixture model:

$$p(w_{n+1}|w_1, ..., w_n) = \lambda_3 \hat{p}(w_3|w_1, w_2) + \lambda_2 \hat{p}(w_3|w_2) + \lambda_1 \hat{p}(w_3)$$

where $\sum \lambda_i = 1$

9.2 Estimating Language Models Predictive Power - Perplexity

Recall that the geometric mean of $x_1, ..., x_n$ is $\sqrt[n]{x_1 \cdots x_n} = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$

We use this to estimate the perplexity of a language model:

$$\text{Perplexity}(\theta) = \left(\prod_{i=1}^{n} p_\theta(w_i | w_{1:i-1})\right)^{-\frac{1}{n}}$$

where $w_1, w_2, ..., w_n$ is a large chunk of text that wasn't used to train the language model. If the perplexity is 100, the model predicts, on average, as if there were 100 equally likely words to follow. This is the (geometric) average "branching factor" for the model on real text. Lower perplexity thus means that the model is better.

9.3 PCA - Pointwise mutual information (PMI)

We can cluster words based on "pointwise mutual information" (PMI). Form a $V \times V$ matrix of PMI values:

$$log\left(\frac{p_{near}(w_1, w_2)}{p(w_1)p(w_2)}\right)$$

Now we compute top k eigenvectors $\phi_1, ..., \phi_k$ and for each word w, we define embedding as: $\phi(w) = (\phi_{1w}, \phi_{2w}, ..., \phi_{kw})^T$

How likely are specific words/clusters to co-occur together within some window, compared to if the were independent. Surprising amount of information captured by purely local cooccurrence counts.

we our us ourselves ours
question questions asking answer answers answering
performance performed perform performs performing
tie jacket suit
write writes writing written wrote pen
morning noon evening night nights midnight bed
attorney counsel trial court judge
problems problem solution solve analyzed solved solving
letter addressed enclosed letters correspondence
large size small larger smaller
operations operations operating operate operated
school classroom teaching grade math
street block avenue corner blocks
table tables dining chairs plate
published publication author publish writer titled
wall ceiling walls enclosure roof
sell buy selling buying sold

Shortcomings of word clusters is the fact that the clusters are still categorical. We thus can't use vector space operation. It is also "one hot" representation wasteful. These problems are however addressed with distributed representations.

Key intuition

- Words that have similar neighbors will be similar
- Note, Self-Refferal notion of similarity!
- This will be an intuition behind "word embeddings"

10 Word Embeddings

Word embeddings form a language model but replace classes by vectors, one for each word. In the case of word embedding we use PMI-like scores to fit the vectors and optimize the embedding vectors using SGD. The PMI scores use a "Two words are similar if they appear with similar words" approach. This method can be applied whenever

have cooccurrence data. Thus word embeddings are vector representations of words, learned from cooccurrence statistics. The models can be viewed in terms of logistic regression and class-based bigram models. Various heuristics have been introduced to get scalability which is vital as embeddings improve with more data.

Interesting: https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html

10.1 Word2Vec - Constructing embeddings

Language model is:

$$p(w_2|w_1) = \frac{exp(\phi(w_2)^T\phi(w_1))}{\sum_w exp(\phi(w_2)^T\phi(w_1))}$$

This is trained on pairs of words that co-occur in a window. Carry out stochastic gradient descent over the embedding vectors $\phi \in R^d$ (where $d \approx 50 - 500$ is chosen by trial and error). This is what Mikolov did at Google with a couple of heuristics:

Heuristics used:

- Skip-gram: predict surrounding words from current word (like PMI)
  - An issue with this is that it "over generates" the data. With text the lazy brown fox jumped we will have $p(brown|lazy)$ and $p(brown|fox)$
- Second is computational. The bottleneck is computing the denominator in the logistic (softmax) probability.
  - Use "negative sampling":
  - Approximation $\sum_w exp(\phi(w_2)^T\phi(w_1)) \approx exp(\phi(w_2)^T\phi(w_1)) + \sum_{\text{random } w} exp(\phi(w_2)^T\phi(w_1))$

These heuristics enable training on very large text collections. Leads to vector representations of words with interesting properties. For example *analogies*:

`king` is to `man` as ? is to `woman`

`Paris` is to `France` as ? is to `Germany`

$$\phi(\texttt{king}) - \phi(\texttt{man}) \overset{?}{\approx} \phi(\texttt{queen}) - \phi(\texttt{woman})$$

$$\widehat{w} = \arg\min_w \|\phi(\texttt{king}) - \phi(\texttt{man}) + \phi(\texttt{woman}) - \phi(w)\|^2$$

Does $\widehat{w} = \texttt{queen}$?

10.1.1 SGD for Word2Vec

Start with a word pair (w_1, w_2) and define loss as $\mathcal{L} = -\log\, p(w_2, w_1)$. Now the gradient update for $\phi(w_2)$ is:

$$\begin{aligned}\phi(w_2) \leftarrow \phi(w_2) - \eta\frac{\partial\mathcal{L}}{\partial\phi(w_2)} \\ = \phi(w_2) + \eta\,(1 - p(w_2 \mid w_1))\,\phi(w_1) \\ = \phi(w_2) + \eta \cdot \textit{weight} \cdot \phi(w_1)\end{aligned}$$

Vector $\phi(w_2)$ is updated to be more similar to $\phi(w_1)$

10.2 GloVe - Constructing embeddings

Shortly after, Stanford group introduced a computational expedient (with attempt to give "principled" motivation)"

$$O(\phi) = \sum_{w_1,w_2} f(c_{w_1,w_2})\Big(\phi(w_1)^T\phi(w_2) - log(c_{w_1,w_2})\Big)^2$$

Where $c_{w,w'}$ are cooccurence counts in a window (PMI),

This type of regression estimator can interpret/relate to other objectives. Main advantage is that SGD can be carried out much more efficiently. The parameters are the embedding vectors $\{\phi(w)\}$. Unlike least-squares regression, this is a nonconvex minimization problem (wiggly, not bowl-shaped). Trained using stochastic gradient descent

The heuristic weight function $f(x) = \left(\frac{x}{x_{max}}\right)^{\alpha}$, wheere $\alpha = \frac{3}{4}$ set empirically. So $10^4 \to 10^3$. Each order of magnitude down gets "boosted" by $\frac{1}{4}$-magnitude.

10.2.1 SGD for GloVe

- Sample a word pair (w_1, w_2)
- Loss is $\mathcal{L} = \left(\phi^T(w_1)\phi(w_2) - \log c_{w_1,w_2}\right)^2$
- Gradient update for $\phi(w_2)$ is

$$\begin{aligned}\phi(w_2) &\leftarrow \phi(w_2) - \eta\frac{\partial\mathcal{L}}{\partial\phi(w_2)} \\ &= \phi(w_2) + 2\eta\left(\log c_{w_1,w_2} - \phi(w_1)^T\phi(w_1)\right)\phi(w_1) \\ &= \phi(w_2) + \eta \cdot \textit{weight} \cdot \phi(w_1)\end{aligned}$$

Vector $\phi(w_2)$ is updated to be more similar to $\phi(w_1)$

10.3 t-SNE - Visualizing Embeddings

How can we visualize the embeddings? We're in a very high dimensional space. We possibly could use PCA—this will tend to distort more. There are many visualization techniques exist. A currently popular one is t-SNE: "Student-t Stochastic Neighborhood Embedding".

10.3.1 Idea of t-SNE

Here's the idea behind t-SNE (tee-snee) which is btw just a visualization technique, to give intuition for the high dimensional embedding:

- Form a language model using the embeddings
- Scale and symmetrize, giving a matrix $P = [P_{ij}]$
- Represent word i by $y_i \in R^2$. Use a heavy-tailed distribution (Student-t)
- Select y_i using stochastic gradient descent

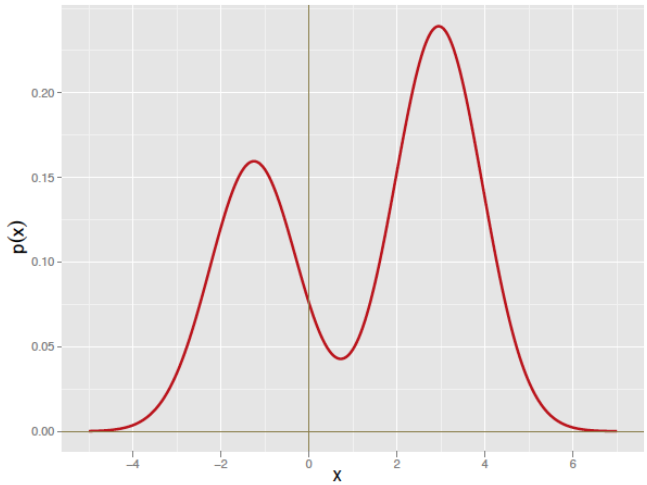
11 Mixtures

Where are we going?

So far we have mostly be dealing with surface representations and explicit features like the listing information for CA housing or Political blogs (word counts).

Now we move on to "Hidden" representations and latent variables. Mixtures are latent variable models. The mixing weight encodes a hidden variable. The computing of mixtures uses basic probabilistic reasoning where the Bayesian approach is the most popular in ML. In a Bayesian approach, the parameters are random, and the data are fixed. Bayesian hierarchical models tie together data using latent variables

An example of a mixture can be the gaussian mixture: $p(x) = \frac{2}{5}\phi(x; -1.25, 1) + \frac{3}{5}\phi(x; 2.95, 1)$



The simplest, most common kind of latent variable model:

$$p(x) = \theta f(x) + (1-\theta)g(x)$$

Hidden variable representation: Define $Z \sim \text{Bernoulli}(\theta)$ and

$$p(x) = \sum_{z=0,1} p(x|z) \cdot p(z)$$

with $p(x|1) = f(x), p(x|0) = g(x), p(z) = \theta^z(1-\theta)^{1-z}$.

11.1 Bayesian Inference

The parameter θ of a model is viewed as a random variable. Inference usually carries out as follows:

- Choose a generative model $p(x|\theta)$ for the data.
- Choose a prior distribution $\pi(\theta)$ that expresses beliefs about the parameter before seeing any data.
- After observing data $D_n = \{x_1, ..., x_n\}$, update beliefs and calculate the posterior distribution $p(\theta|D_n)$.

11.1.1 Bayes Theorem

A simple consequence of condition probability: $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$

The posterior distribution can be written as

$$p(\theta \mid x_1, \ldots, x_n) = \frac{p(x_1, \ldots, x_n \mid \theta)\pi(\theta)}{p(x_1, \ldots, x_n)} = \frac{\mathcal{L}_n(\theta)\pi(\theta)}{c_n} \propto \mathcal{L}_n(\theta)\pi(\theta)$$

where $\mathcal{L}_n(\theta)$ is the *likelihood function* and

$$c_n = p(x_1, \ldots, x_n) = \int p(x_1, \ldots, x_n \mid \theta)\pi(\theta)d\theta = \int \mathcal{L}_n(\theta)\pi(\theta)d\theta$$

is the normalizing constant, which is also called *evidence*.

Important Example

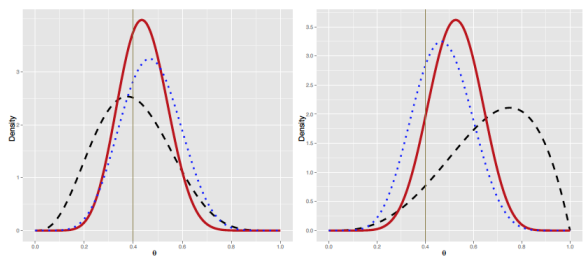
Take model $X \sim \text{Bernoulli}(\theta)$ for coinflip for example:

- Let data be $D_n = \{x_1, ..., x_n\}$
- Let $s = \sum_{i=1}^n x_i$ be the number of heads

- Natural prior: Beta(α, β): $\pi_{\alpha,\beta}(\theta) = \theta^{\alpha-1}(1-\theta)^{\beta-1}$
- The scaling constant: $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$ where $\Gamma(z) = \int_0^\infty x^{z-1}e^{-1}dx$
- Posterior distribution $\theta|D_n$ is another beta distribution:
  - $\tilde{\alpha} = \alpha + \text{no. heads } = \beta + s$
  - $\tilde{\beta} = \beta + \text{no. tails } = \beta + n - s$

Example

$n = 15$ points sampled as $X \sim$ Bernoulli$(\theta = 0.4)$, with $s = 7$ heads.



Prior distribution (black-dashed), likelihood function (blue-dotted), posterior distribution (red-solid).

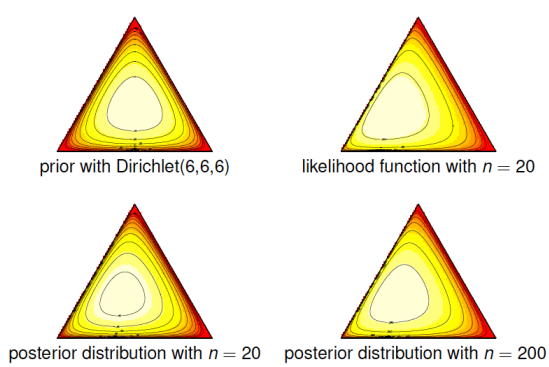
11.1.2 Dirichlect

Multinomial model with Dirichlet prior is generalization of the Bernoulli/Beta model:

$$\text{Dirichlet}_\alpha(\theta) = \theta_1^{\alpha_1-1}\theta_2^{\alpha_2-1}...\theta_n^{\alpha_n-1}$$

Where $\alpha = (\alpha_1, ..., \alpha_K) \in R_+^K$ is a non-negative vector.

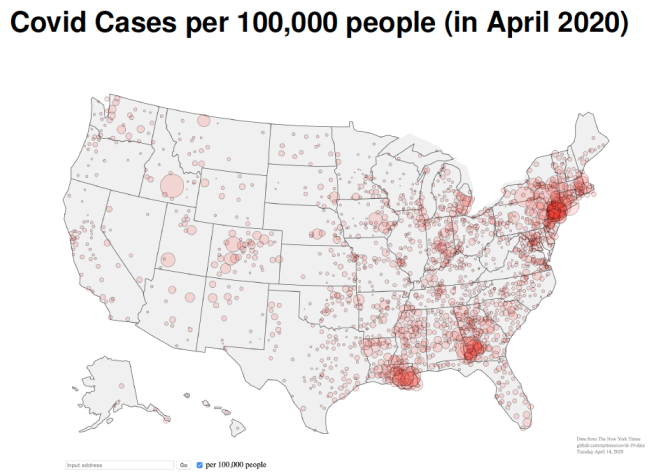
Example



11.2 Hierarchical models

11.2.1 Binomial

Let us think about the covid cases



We are interested in modeling the infection rates across counties. In a Bayesian hierarchical model, we tie together our inferences across regions. In each county c we test n_c people (at random) and observe how many people Y_c have Covid-19.

$Y_c \sim \text{Binomial}(n_c, \theta_c)$. This is equivalent to n_c coin flips (Bernoulli) each with probability of heads θ_c.

For each county $c := \begin{cases} \theta_x \sim F \\ Y_c|\theta_c \sim \text{Binomial}(n_x, \theta_c) \end{cases}$. This ties together the parameters and makes better use of the data.

11.2.2 Gaussian

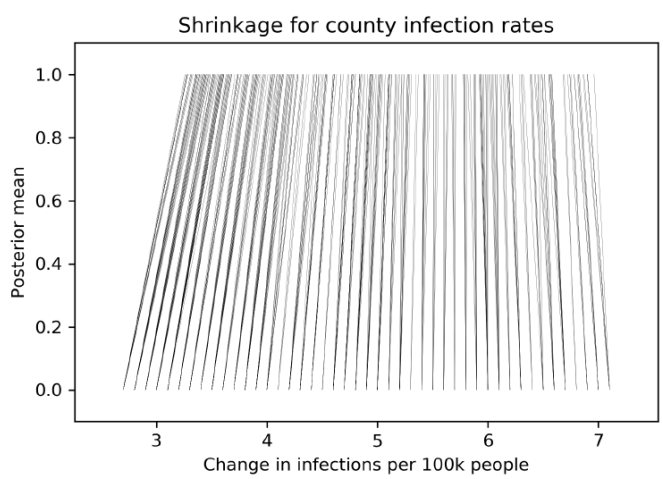
A simple transformation of $\psi_c = log(\frac{\theta_c}{1-\theta_c})$ makes the use of Gaussion more appropriate to use approximation.

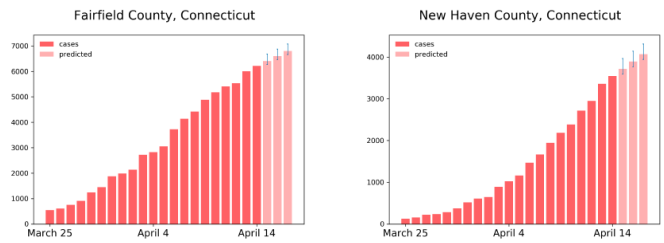
$$\psi_c \sim N(\mu, \tau^2)$$
$$Z|\psi_c \sim N(\psi_c, \sigma_c^2)$$

- Posterior distribution may not be written down explicitly
- But we can sample from it
- This approximates the posterior as a mixture of Gaussians

11.2.3 Shrinkage

Note: This is a different shrinkage from regularization shrinkage. Now we ty the parameters together resulting in the small probabilities being increased and the larger probabilities being decreased. This is called "Shrinkage." The posterior estimates are closer together than the raw frequencies.





Interesting another example is the election forecasting: https://projects.economist.com/us-2020-forecast/president/how-this-works

See notes on Bayesian inference on iML site (bayes-notes.pdf)

- These are more advanced, from S&DS 365.
- Not necessary to understand everything—but you should be able to do some of the basic "coin flipping" calculations
- We'll build on this when discussing topic models

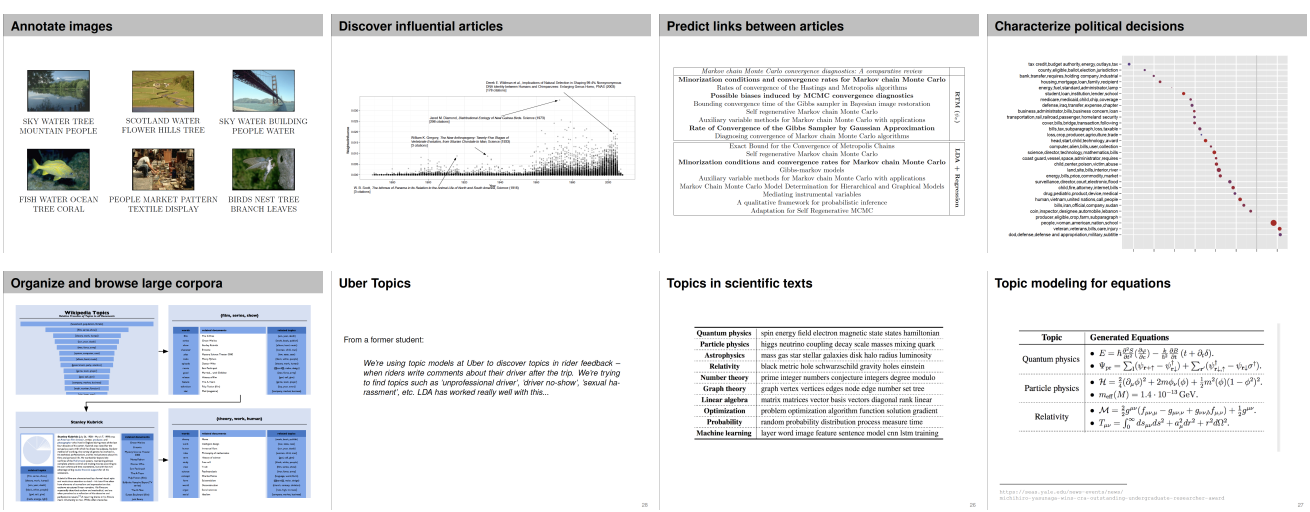
12 Bayes and Topic Models

Now we move onto high level intro to topic models where we utilize latent variables, mixtures. A good read for a better grasp of the idea behind topic models: http://www.cs.columbia.edu/~blei/topicmodeling.html.

12.1 Introduction

Topic models provides methods for automatically organizing, understanding, searching and summarizing large electronic archives. Topic models automatically extract "semantic themes" from large document collections. Based on latent variables, mixtures, and Bayesian inference. Can be useful for a wide variety of data for example:

1. Discover the hidden themes that pervade the collection
2. Annotate the documents according to those themes
3. Use annotations to organize, summarize and search texts



12.2 Constructing TM

1. Data are assumed to be observed from a generative probabilistic process that includes hidden variables.

- In text, the hidden variables are the thematic structure.

2. Infer the hidden structure using posterior inference

- What are the topics that describe this collection?

3. Situate new data into the estimated model.

- How does a new document fit into the topic structure?

Like mentioned above Latent Dirichlet allocation (LDA) has worked historically pretty well for topic models.

12.2.1 Generative model for Latent Dirichlet allocation (LDA)

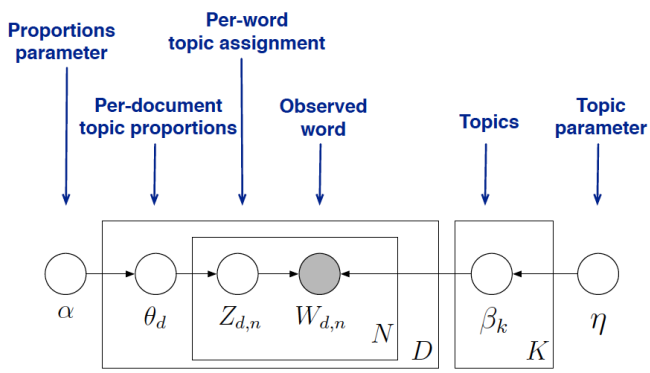
Simple intuition is that documents exhibit multiple topics (computer topics, sports topics, health topics), so each document is a mixture of collections of topics. Each topic is a distribution over words and each word is drawn from one of these topics.

We can think of this as tiny document has a the sentence: *Sid is an athletic and sexy kingpin*

This document presumably has 3 topics: sex (Sid, sexy, kingpin), sports (athletic) and small words (is, an, and).

12.2.2 LDA as a graphical model

Encodes our assumptions about the data and connects to algorithms for computing with data. The nodes are random variables, edges indicate dependence. Shaded nodes are observed and the plates indicate replicated variable.



The equation is:

$$\prod_{i=1}^{K} p(\beta_i|\eta) \prod_{d=1}^{D} p(\theta_d|\alpha) \left(\prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n}) \right)$$

This joint distribution defines a posterior distribution. Intuition into the equation, from a collection of documents in, we infer:

- Per-word topic assignment: $z_{d,n}$
- Per-document topic proportions: θ_d
- Per-corpus topic distribution: β_d

Then use the posterior expectations to perform the task at hand, e.g. information retrieval, document similarity, exploration...

12.2.3 Posterior Distribution

In reality we as humans only observe the documents, the other structure are hidden variables. The goal of the topic model is to infer the hidden variables i.e. compute there distribution conditioned on the document $p(\text{topics, proportions, assignments}|\text{ documents})$. Estimating Posterior distribution is done with Gibbs sampling.

There is a large literature on approximating the posterior. We will focus on *Gibbs sampling* and *mean-field variational methods* (batch and online)

12.2.4 Shortcomings for LDA

Dirichlet for topic proportions: $p(\theta|\alpha) \propto \prod_{j=1}^{k} \theta_j^{\alpha_j - 1}$

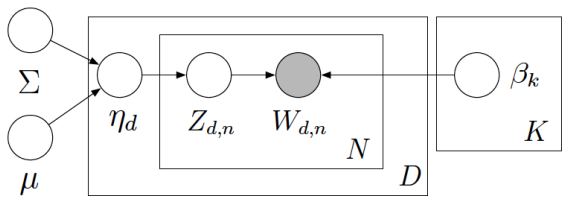
Near independence of components makes it an unrealistic. Take for example an article about fossil fuels. It is way more likely to be also about geology than about genetics.

12.2.5 Markov Chain Monte Carlo

- We construct a Markov chain on the hidden variables, whose limiting distribution is the posterior.
- We collect independent samples from that distribution; approximate the posterior with them
- In Gibbs sampling the chain is defined by the conditional distribution of each hidden variable given observations and the current setting of the other hidden variables.

12.2.6 Correlated topic model

Draw topic proportions from a logistic normal. This is extremely useful for providing a map of topics and how they are related and thus we get better predictions via correlated topics. This comes though on the cost of conjugacy; Posterior over θ does not have same form.



12.3 Modeling Evolution of Topics

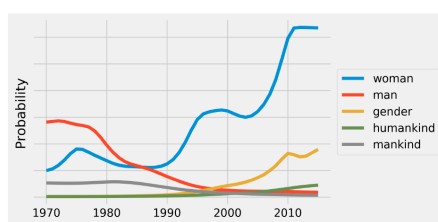
In LDA the document order doesn't matter. The topics should evolve over time. Many document collections have such dynamics such as emails and news articles to name a few.

12.3.1 Time series topic model

Allow topics to evolve between time slices. For each document in current time slice we select a distribution over topics and generate the words from the resulting topic mixture.

Very Interesting: https://towardsdatascience.com/exploring-the-un-general-debates-with-dynamic-topic-models-72dc0e307696

	1970	1980	1990	2000	2010
0	right	right	right	right	right
1	human	human	human	human	human
2	people	people	freedom	law	law
3	international	freedom	people	democracy	woman
4	principle	international	democracy	respect	freedom
5	justice	political	respect	international	respect
6	freedom	principle	law	people	people
7	law	respect	international	freedom	democracy
8	state	justice	principle	principle	rule
9	must	social	state	must	international



More Tutorials:

- https://medium.com/@lettier/how-does-lda-work-ill-explain-using-emoji-108abf40fa7d
- https://towardsdatascience.com/latent-dirichlet-allocation-intuition-math-implementation-and-visualisation-63ccb616e094

13 Neural Networks

For complex data we may want to learn features of the inputs. That kind of representation can be part of a logistic regression model. neural Networks connect embeddings and logistics regression. It can be looked at as an alternative view of linear models, adding nonlinearities (activation functions). Features that are linear transforms followed by a nonlinearity form are the building blocks of (artificial) neural networks. The name "neural network" comes from a crude analogy with neurons in biological brains. They are trained using stochastic gradient descent and implemented using backpropagation. Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. With this knowledge you can automate a neural network to train complex networks (WITH NO MATH AT ALL)!

This is a cool way to play around with neural networks: https://playground.tensorflow.org/

13.1 Logistic Regression

Recall : $log\left(\frac{P(y=1|x)}{P(y=0|x)}\right) = \beta^T x + \beta_0$

Equivalently : $P(Y=1|x) \propto e^{\beta^T x + \beta_0}$

Multi class : $P(Y=k|x) \propto e^{\beta_k^T x + \beta_{k,0}} = \text{ Softmax}\left(\left\{\beta_k^T + \beta_{k,0}\right\}\right)$ where the latter is ML terminology. Remember $k = 0, 1, ..., k-1$

Now let's think about x. What if x is an image, represented as pixels? It might be hard to get an accurate classifier. We want to learn feature representation $\phi(x)$. Now the model becomes as follows, and the parameters β and ϕ need to be trained/learned:

$$P(Y=k|x) \propto e^{\beta_k^T \phi(x) + \beta_{k,0}} \text{ where } k = 0, 1, ..., k-1$$

13.2 Word embeddings

Applying the model above, we could have a bigram model. The model then becomes: $P(w_1|w_2) \propto e^{\beta_{w_2}^T \phi(w_1)}$ where $\phi(w)$ is a learned feature vector or embedding vector for each word. The parameters $\beta_w = \beta(w)$ are also embeddings. By making them the same as ϕ we get the word2vec model.

13.3 Starting with regression:

For linear regression, our loss function for an example (x,y) is: $\mathcal{L} = \frac{1}{2}(y - \beta^T x - \beta_0)^2 = \frac{1}{2}(y-f)^2$ where $f = \beta^T x + \beta_0$. Now if we add a layer we let $f = \beta^T h + \beta_0$ where $h = Wc + b$. But this is just a linear model $f = \tilde{\beta}^T x + \tilde{\beta}_0$. Resulting in the fact that we get a reparameterization of a linear model; nothing new. Hence, we need to add *nonlinearities* for us to benefit from linear regression.

13.3.1 Nonlinearities

Add nonlinearity: $h = \phi(Wx + b)$ applied component-wise. For regression the last layer is just linear: $f = \beta^T + \beta_0$. Commonly used nonlinearities:

$$\phi(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$
$$\phi(u) = \text{sigmoid}(u) = \frac{e^u}{1 - e^u}$$
$$\phi(u) = \text{relu}(u) = \max(u, 0)$$

So a neural network is nothing more than a parametric regression model with a restricted type of nonlinearity. So why are they called neural networks?

13.4 Training

We train the parameters with stochastic gradient descent. To calculate derivatives we just use the chain rule, working our way backwards from the last layer to the first.

First layer

$$\mathcal{L} = \frac{1}{2}(y - \beta^T x - \beta_0)^2 = \frac{1}{2}(y - f)^2$$

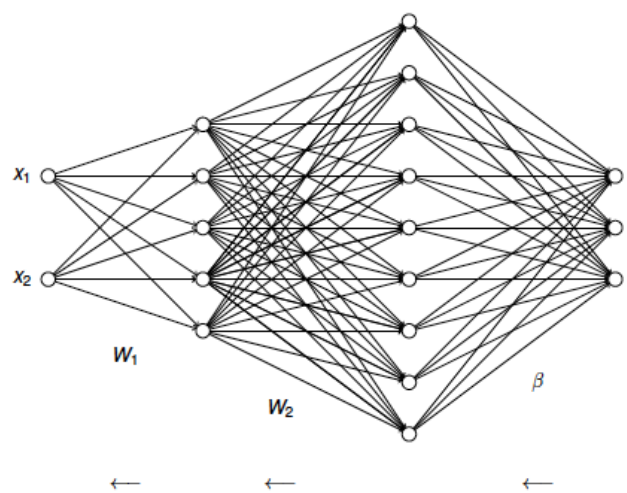
First layer derivative f

$$\frac{\partial \mathcal{L}}{\partial f} = -(y - f)$$

Second layer derivative β

$$\frac{\partial \mathcal{L}}{\partial \beta} = \frac{\partial \mathcal{L}}{\partial f}\frac{\partial \mathcal{L}}{\partial \beta} = -(y - f)\frac{\partial \mathcal{L}}{\partial \beta} = -(y - f)h$$

Intuitive explanation



Start at the last layer and send error information back to previous layers. The loss function is: $\mathcal{L} = \frac{1}{2}(y - f)^2$. The change in loss due to making a small change in output f is $\frac{\partial \mathcal{L}}{\partial f} = -(y - f)$. We now send this backward through the network. Now suppose $f = ab$:

First for b:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a} &= \frac{\partial \mathcal{L}}{\partial f}\frac{\partial f}{\partial a} \\ &= \frac{\partial \mathcal{L}}{\partial f} b \\ &= -(y - f)b\end{aligned}$$

Now for a:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial f}{\partial b}\frac{\partial \mathcal{L}}{\partial f} \\ &= a\frac{\partial \mathcal{L}}{\partial f} \\ &= a(f - y)\end{aligned}$$

The fancy version for this is used with matrix notation. If $A = BC$:

$$\frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial A} C^T$$
$$\frac{\partial \mathcal{L}}{\partial C} = B^T \frac{\partial \mathcal{L}}{\partial A}$$

13.4.1 Example:

One layer:

So if $f = Wx + b$ then:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial f} x^T \\ &= (f - y)x^T\end{aligned}$$

Two layers:

So if $f = W_2 h + b_2$ and $h = W_1 x + b_1$ then:

Start with last layer:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_2} &= \frac{\partial \mathcal{L}}{\partial f} h^T = (f - y)h^T \\ \frac{\partial \mathcal{L}}{\partial h} &= W_2^T \frac{\partial \mathcal{L}}{\partial f} = W_2^T (f - y)\end{aligned}$$

Now send this backpropagate to first layer:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_1} &= \frac{\partial \mathcal{L}}{\partial h} x^T = (f - y)h^T \\ &= W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T \\ &= W_2^T (f - y)x^T\end{aligned}$$

Remember, this **just gives a linear model!** so we **NEED TO ADD A NONLINEARITY**:

- $h = \varphi(W_1 x + b_1)$
- $f = W_1 h + b_2$
- These functions can be looked at as $A = BC$ showed above, be careful b_i is not the last matrix. In these examples it is x and h.

If $\varphi(u) = \text{ReLU}(u) = max(u, 0)$ we know the derivative of the ReLU is $\mathbf{1}(h > 0)$ then this just becomes:

Picking up where we left off:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_1} &= \mathbf{1}(h > 0) \frac{\partial \mathcal{L}}{\partial h} x^T \\ &= \mathbf{1}(h > 0) W_2^T \frac{\partial \mathcal{L}}{\partial h} x^T \\ &= \mathbf{1}(h > 0) W_2^T (f - y)x^T\end{aligned}$$

If we use another nonlinearity just substitue the derivative of that nonlinearity with the derivative of the ReLU.

13.5 Classification

For classification:

Use softmax to compute probabilities:

$$(p_1, p_2, p_3) = \frac{1}{e^{f_1}, e^{f_2}, e^{f_3}} \left(e^{f_1}, e^{f_2}, e^{f_3} \right)$$

The loss function is:

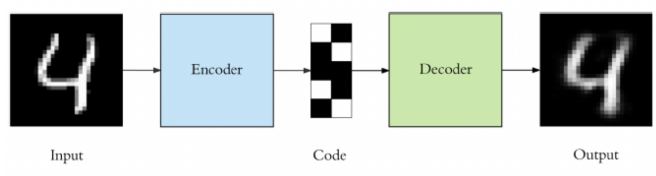
$$\mathcal{L} = -\log P(y|x) = log\left(e^{f_1}, e^{f_2}, e^{f_3} \right) - f_y$$

So we have:

$$\frac{\partial \mathcal{L}}{\partial f_k} = p_k - \mathbf{1}(y = k)$$

14 Autoencoders

Unsupervised learning methods. Squeeze high dimensional data through a "bottleneck" of lower dimension which we train to minimize reconstruction error. Autoencoders essentially compress the input and then reconstruct it where the bottleneck forces extraction of useful features. Unlike other methods we know that autoencoders will overfit and "memorize" the data. Overfitting mitigated by denoising autoencoders.



14.1 Important aspects

- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

14.2 Simple autoencoder example: Single hidden layer

- Encoder network of the form: $h = \text{ReLU}(Wx + b)$
  - where $W \in \mathbb{R}^{H \times D}$ and $b \in \mathbb{R}^H$
- Decoder Network is then: $\hat{x} = \text{ReLU}(\tilde{W}h + \tilde{b})$
  - where $\tilde{W} \in \mathbb{R}^{H \times D}$ and $\tilde{b} \in \mathbb{R}^H$
- Objective function: $\mathbb{L} = \frac{1}{n} \sum_{i=1}^{n} ||x_i - \text{ReLU}(\tilde{W}\ \text{ReLU}(Wx_i + b) + \tilde{b})||^2$

Remember that $\text{ReLU}(x) = max(0, x)$, applied component wise.

14.3 Simple example: Code of dimension K

Encoder network of the form:

$$h = \text{ReLU}(W_1 x + b_1)$$
$$c = \text{ReLU}(W_2 x + b_2)$$

where $W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{K \times H}, b_1 \in \mathbb{R}^H, b_2 \in \mathbb{R}^K$

Decoder Network is then:

$$\hat{h} = \text{ReLU}(\tilde{W}_1 c + \tilde{b}_1)$$
$$\hat{x} = \text{Sigmoid}(\tilde{W}_2 \hat{h} + b_2)$$

where $\tilde{W}_1 \in \mathbb{R}^{H \times K}, \tilde{W}_2 \in \mathbb{R}^{D \times H}, \tilde{b}_1 \in \mathbb{R}^H, \tilde{b}_2 \in \mathbb{R}^D$

Objective function - binary cross entropy:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (x_i \log \hat{x}_i + (1 - x_i) \log (1 - \hat{x}_i)$$

14.4 Adam optimizer

Variant of stochastic gradient descent where separate learning rate (step size) is maintained for each network weight (parameter). Each step size adapted as learning progresses based on moments of the derivatives.

14.5 Variants

- **Denoising Autoencoder:** We can denoise auto encoders by feeding it noisy data and train to match denoised data.
- **Sparse Autoencoder:** Add penalty to encourage sparsity which forces autoencoder to discover interesting structure in data

15 Reinforcement Learning

Reinforcement learning is a framework for sequential decision making to achive a long-term goal. Take for example an agent that interacts with an environment. The actions the agent takes change the state of the environment. The agent receives rewards for each action, and seeks to maximize the total cumulative reward. Other motivating examples include preparing a meal, robot vacuum cleaning the house, learning to walk or ride a bike. The value of a state is the total reward the agent can expect to accumulate in the future, starting from that state.

15.1 Formalization

- A environment is in state s a given time
- A agent takes action a
- The environment transitions to state $s' = \text{next}(s, a$
- The agent receives reward $r = \text{reward}(s, a)$

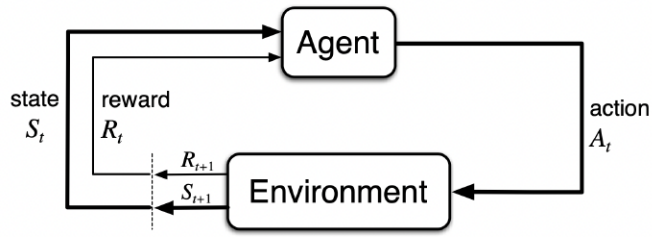
This is said to be a Markov decision process because the next state only depends on the current state and the action selected. It's a "decision process" because the agen is making choices of actions in a sequential manner.

15.2 Charectistics

The characteristics of Reinforcement learning are the following:

- RL is inherently sequential
- In between supervised and unsupervised learning
- Agent can't act too greedily; needs to be strategic

The aim of RL is to learn to make optimal decisions from experience



15.3 Q learning

Q-learning is an iterative algorithm that maximizes the "quality" of each state-action pair. Q-learning is thus an example of temporal difference (TD) learning. In further detail, it is an "off-policy" approach that is practical if the space actions is small. Value iteration is analogous approach for learning the value function for a given policy π, a (possibly random) coice of action for each state.

- Maintains a quality variable $Q(s, a)$ for taking action a in state s.
- A measure of the cumulative rewards obtained by the algorithm when it takes action a in state s.
- Quality should not be assessed purely based on the reward the action has in the current time step.
- Need to take into account the future rewards.

Normally written:

$$Q(s,a) \leftarrow (a-\alpha)Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a')\Big)$$

Written as gradient ascent algorithm w. step size α :

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a), a') - Q(s,a)\Big)$$

When the action a is taken in state s, reward(s, a) is given. Then the algorithm moves to a new state next(s, a). The cumulative future reward of this action is $\max_{a'} Q(\text{next}(s, a), a')$. Future rewards discounted by factor $\gamma < 1$. Hence, it trades of short-term rewards against the long term rewards.

15.4 Principles of RL

There are 5 main principles, Policy, reward signal, value function, model and Bellman equation. Let us define them:

- **Policy**: A mapping from states to actions. An algorithm/rule to make decisions at each time step, designed to maximize the long term reward.
- **Reward signal**: The sequence of rewards received at each time step. An abstraction of "pleasure" (positive reward) and "pain" (negative reward) in animal behavior.
- **Value function**: A mapping from states to total reward. The total reward the agent can expect to accumulate in the future, starting from that state. Rewards are short term. Values are predictions of future rewards.
- **Model**: Used for planning to mimic the behavior of the environment, to predict rewards and next states. A model-free approach directly estimates a value function, without modeling the environment.

Now let's take a look at the Bellman Equation in more detail.

- **Bellman Equation**: The optimal value function is the largest expected discounted long term reward starting from that state. The Bellman equations are optimality conditions that characterize the value function

The optimality condition for the value function v_* is

$$\mathrm{v}_*(s) = \max_a \left\{ \mathrm{reward}(s,a) + \gamma \mathrm{v}_*(\mathrm{next}(s,a)) \right\}$$

The optimality condition for the Q-function is

$$Q_*(s,a) = \mathrm{reward}(s,a) + \gamma \max_{a'} Q_*(\mathrm{next}(s,a), a')$$

$$\Rightarrow \text{and then } \mathrm{v}_*(s) = \max_{a'} Q_*(s,a')$$

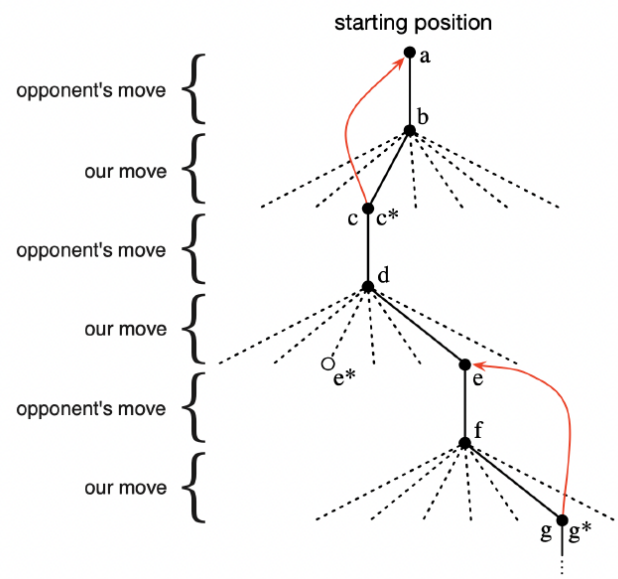
15.5 Example: Tic-Tac-Toe

Tic-Tac-Toe is a fantastic example that can be approached using Q-learning or "value iteration". Unlike classical game (minimax) search, does not assume opponent is playing optimally. Also does not require specification of the opponent. Can be learned using "self-play"

15.5.1 Learning strategy

Set up a table of numbers, one for each possible state. Each number is latest estimate of the probability of winning from that state. The table is the value function. Playing as X, for all states with three Xs in a row, the value is one (similarly 0 for three Os in a row). Initialize all other values to $\frac{1}{2}$.

Now we play many games agains the opponent. Most of the time we move greedily, selecting the move that leads to the state with gratest value. Occasionally, select randomly from other moves (Exploratory Moves). For greedy moves adjust the state's value a little bit toward the value of the next state $V(S_t) \leftarrow V(S_t) + \alpha(V(S_{t+1}) - V(S_t))$



15.6 More on Bellman Equation

15.6.1 Deterministic case

If we know Q_*, we know v_*

$$\begin{aligned} v_*(s) &= \max_a Q_*(s,a) \\ &= \max_a \left\{ \mathrm{reward}(s,a) + \gamma \max_{a'} Q_*(\mathrm{next}(s,a), a') \right\} \\ &= \max_a \left\{ \mathrm{reward}(s,a) + \gamma \max_{a'} v_*(\mathrm{next}(s,a)) \right\} \end{aligned}$$

which is the Bellman equation

15.6.2 Random environments

The optimality condition for the value function v_*, is:

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\{r + \gamma \mathrm{v}_*(s')\}$$
$$= \max_a \mathbb{E}\Big[R_{t+1} + \gamma \mathrm{v}_*(S_{t+1})|S_t = s, A_t = a\Big]$$

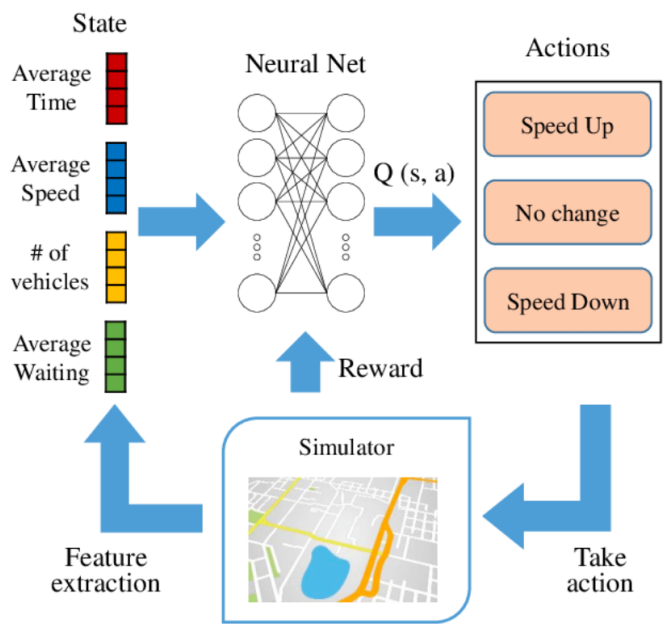
The optimality condition for the Q function is:

$$Q_*(s,a) = \sum_{s',r} p(s',r|s,a)\{r + \gamma \max_{a'} \mathrm{Q}_*(s',a')\}$$
$$= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} \mathrm{Q}_*(S_{t+1},a')|S_t = s, A_t = a\Big]$$

15.7 Deep Reinforcement Learning

15.7.1 Motivation

Direct implementation of Q-learning only possible for small state and action spaces. For large state spaces we need to map states to "features". Deep RL uses a multilayer neural network to learn these features and the Q-function.



Starting Point: Bellman equation of course!

$$Q(s,a;\theta) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} \mathrm{Q}_*(S_{t+1},a';\theta)|S_t = s, A_t = a\Big]$$

The parameters θ are weights in a neural network. The state S_{t+1} is the input to the network and each possible action a is assigned a value by the network.

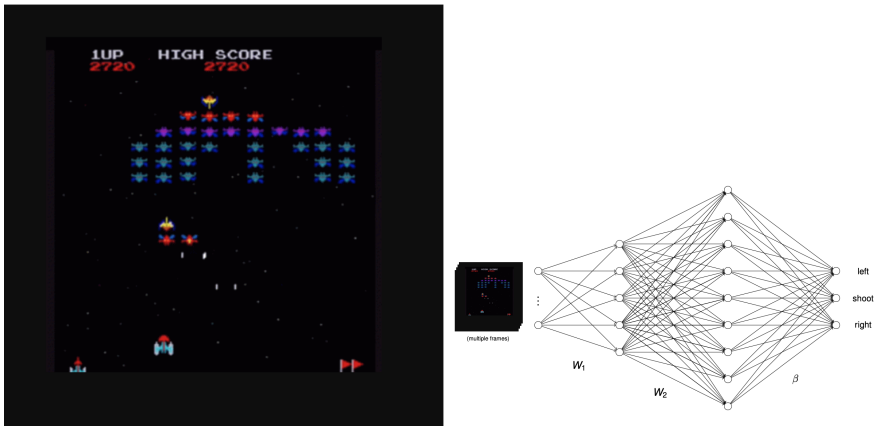
How do we solve this implicit equation for the network parameters?

- Let y_t be a sample from this conditional distribution: $y_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1},a';\theta)$
- Adjust the parameters θ to make the squared error small (SGD): $(y_t - Q(s,a;\theta))^2$
  - by carrying out SGD using backpropagation: $\theta \leftarrow \theta + \eta(y_t - Q(s,a;\theta))\triangledown_\theta Q(s,a;\theta)$

When does learning take place?

Recall from Bellman equation that y_t is an expectation. Thus learning takes place when expectations are violated. The receipt of the reward itself does not cause changes.

15.8 Some Cool DeepMind RL - VideoGames

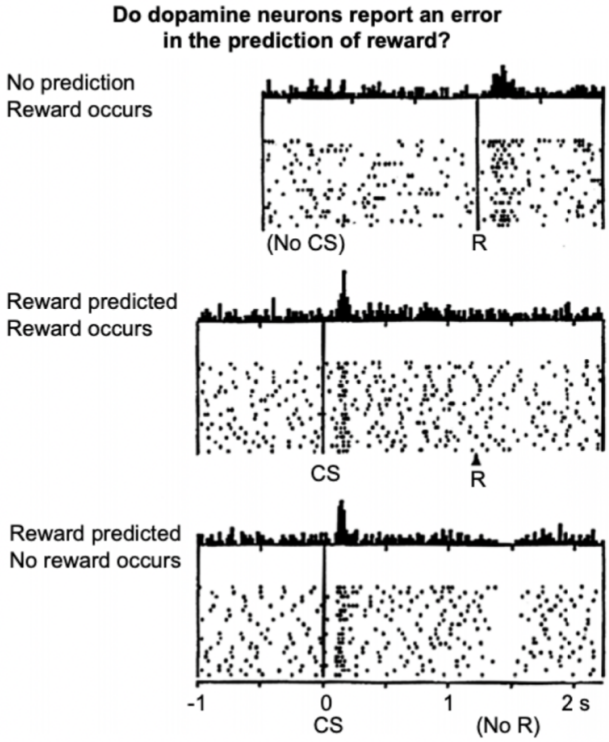


- Images cropped to 84×84 pixels; 128 color palette; input sequence of 4 frames; reward is score
- 3-layer convolutional neural network, ReLU nonlinearity, final layer fully connected, 256 neurons
- Q-learning carried out over minibatches of playing sequences that are "remembered and replayed"
- Identical architecture used for seven games (Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, Space Invaders)
- Each game trained for 100 epochs (50,000 minibatch weight updates / epoch), about 50 hours
- Surpasses human expert on three of seven games

This is cool: https://deepmind.com/blog/article/deep-reinforcement-learning

Also John Lafferty highly recommends the book *Models of the Mind: How Physics, Engineering and Mathematics Have Shaped Our Understanding of the Brain*

15.9 Connection to Neuroscience



- Dopamine neurons fire when there is an error — expectations are violated
- Initially they fire when a reward is received when none was expected

- When a light is flashed before the reward (conditioning stimulus CS), dopamine firing moves to the stimulus, as the animal learns the association
- When the reward is removed, an error in expectation results; dopamine is again released

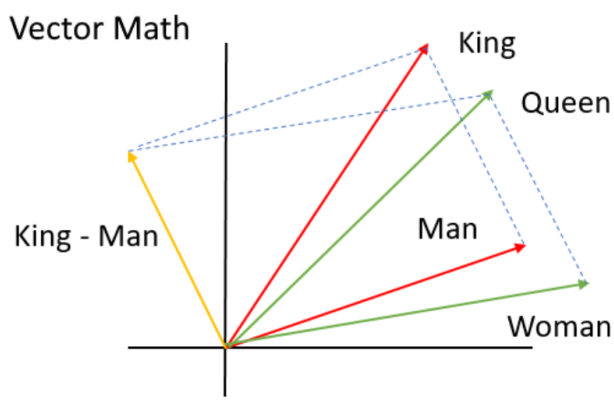
15.10 Summary of RL

- For large state spaces, can't estimate Q-function directly
- Deep Q-learning: Using a (deep) neural network with state as input, value assigned to each possible action
- Success at games (Backgammon, Atari, Go)
- Close connection to neuroscience of behavior and reward

16 Societal Issues for Machine Learning

16.1 Example of representation learning: Word embeddings

Each word in vocab is mapped to 100 or 500 dimensional vector. Based solely on co-occurrence statistics in corpus of text. Which lets us calculate analogies.



However, embedding encode some societal bias:

ϕ(`smart`) - ϕ(`girl`) + ϕ(`boy`):

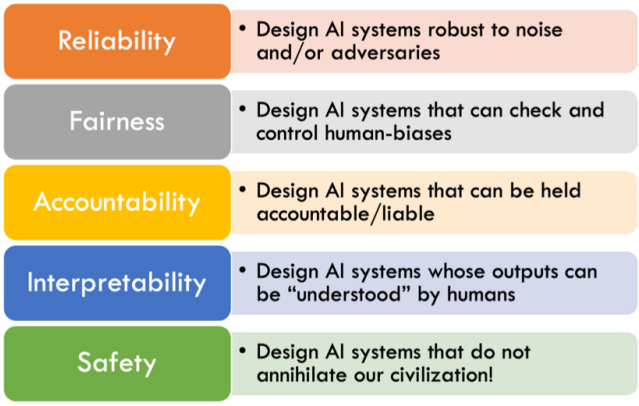
```
wise
better
guy
kind
good
kid
```

ϕ(`smart`) - ϕ(`boy`) + ϕ(`girl`):

```
sexy
pretty
incredibly
cute
exciting
funny
```

16.2 ML at large internet company

Typical project lifetime: 6 months to 1 year. Ads projects involve thousands of software engineers. Often adding new "feature" to existing black box model. No single person understands entire model. With this follows little to

non interpretability. On top of that there are always security issues with data.

We thus need to incorporate important directions which requires principled, interdisciplinary approach: