Mathematics of Data Science

Contents

1 Ranking Methods

1.1 Plurality

Most naive way of ranking candidates by taking the number of first place votes.

1.2 Borda Count

Count the number of times a candidate appears in ith position and apply the following set of weights as follows:
Let n be the total number of candidates. (n - 1) points are given to first place, (n - k) to k-th place, and 0 is given to last place. The tallies are summed and the winner is the candidate with the highest sum.

1.3 Massey Ranking

Assume there are n teams. Let r_i denote the scoring power of a team, where $r_i - r_j = y_{ij}$ for the margin of victory of team i over team j. Solve the following system of equations for r:

$$Xr = y$$

Each row of matrix X is zero except 1 at position i and -1 at position j, y is a list of the head-to-head scores. We can solve this using least squares.

$$\min_r \|Xr - y\|_2^2$$

$$X^T Xr = X^T y$$

Denote $M = X^T Xr$, $d = X^T y$

$$Mr = d$$

$$\begin{pmatrix} t_1 & -m_{12} & \dots & -m_{1n} \\ -m_{21} & t_2 & \dots & -m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & -m_{n2} & \dots & t_n \end{pmatrix} * \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

Here, t_i is the total games team i has played, m_{ij} is the number of games team i played against team j, and d_j is the score differential of the team.

It is possible to use this method to rank candidates. In this case, each voters ranking counts as a game (if there are 5 candidates, each candidate plays a total of 4 matches for each voter).

1.4 PageRank Algorithm

Give each page a score x_j that indicates importance. A page votes for another page by having n_j outgoing links, each vote counts as $\frac{x_j}{n_j}$. The problem becomes:

$$x = Ax$$

Where $x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and A is a matrix whose x columns is a vector whose y entry is $\frac{1}{n_j}$ if there is a link from website x to y. If A is not reducible, we can modify the algorithm to be

$$\tilde{A} = \alpha A + \frac{(1-\alpha)}{n} * 1, 0 < \alpha < 1$$

This algorithm can also be used to rank candidates. Matrix A columns become 0 for a win and $\frac{1}{j_i}$, j_1 represents the number of loses. Replace the undefeated candidate vector of 0^T with $(\frac{1}{n}, \ldots, \frac{1}{n})$

2 Linear Regression

2.1 Ordinary Linear Regression

Take m data points $x_1, x_2, \ldots, x_m$, each of which has n features, which gives us x_{ij}. Denote the response variable y_i. Approximate y as a linear function of x.

$$h(x_i) = \sum_{k}^{n} w_k x_{ik} + b = w^T x_i + b$$

To do this we minimize the cost function.

$$\min_{w,b} \|X^T w + b1 - \hat{y}\|_2$$

Here, X is a n x m matrix, whose columns are data points. w is a length n vector of weights. b is a scalar. The optimal solution to this is:

$$\tilde{X}\tilde{X}^T \begin{bmatrix} w \\ b \end{bmatrix} = \tilde{X}\hat{y}, \tilde{X} = \begin{bmatrix} X \\ 1^T \end{bmatrix}$$

This equation is the **least squares problem**.

2.2 Polynomial Regression

Approximate y as a polynomial of degree M.

$$h(x_i) = \sum_{a_1+a_2+\ldots+a_n \leq M} c_{i_2,i_2,\ldots,i_n} x_{i1}^{a_1} x_{i2}^{a_2} \ldots x_{in}^{a_n}$$

There are $r = \binom{M+n}{n}$ monomials. We apply these monomials to data X, with m columns (data points) and r rows.

$$\Phi(X) = \begin{bmatrix} h_1(x_1) & \ldots & h_r(x_1) \\ h_1(x_2) & \ldots & h_r(x_2) \\ \vdots & \vdots & \vdots \\ h_1(x_m) & \ldots & h_r(x_m) \end{bmatrix}^T$$

Then our least squares problem becomes:

$$\min_{w} \|\Phi(X^T)w - \hat{y}\|_2$$

2.3 Ridge Regression

Idea of Ridge Regression is to introduce a little bit of bias to severely decrease the variance of our estimator.
Let $Y \in \mathbb{R}^n$, $X \in \mathbb{R}^{nxd}$, and $y \in \mathbb{R}^d$

Our true model is $Y = B^T X + \epsilon$

We are given n samples: $(x_1, y_1), \ldots, (x_n, y_n)$

Recall our estimation of β using ordinary least squares:

$$\hat{\beta} = \min_{\beta} \|Y - X\beta\|_2^2 \rightarrow \hat{\beta} = (X^T X)^{-1} X^T Y$$

Instead of minimizing over $\mathbb{R}^d$, we can minimize over a constrained set $\beta \in \{\beta : \|\beta\|_2^2 \leq \tau\}$. This can also be written as an unconstrained problem (λ, the tuning parameter, is the dual of τ, also not when $\lambda \rightarrow 0$, $\tau \rightarrow \infty$ we have ordinary least squares):

$$\hat{\beta} = \min_{\beta} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \rightarrow \hat{\beta}(\lambda) = (X^T X + \lambda I)^{-1} X^T Y)$$

As with least squares we seek coefficients that estimate data well, but add the second term (shrinkage penalty), which becomes small when $\beta_1, \ldots, \beta_j$ are close to zero, and has the effect of shrinking these estimates towards zero. λ controls the relative impact of these two terms.

$$[\hat{\beta}_r(\lambda)]_j = \frac{\hat{\beta}_j}{1 + \lambda}$$

We can use the expectation of $\hat{\beta}_r$ to calculate the bias and variance.

$$MSE = \|Bias^2(\hat{\beta}_r(\lambda))\|_2^2 + \sum_{j=1}^{p} Var(\hat{\beta}_r(\lambda)_j) = \frac{p\sigma^2}{(1+\lambda)^2} + \frac{\lambda^2 \|\beta\|_2^2}{(1+\lambda)^2}$$

We need to know σ^2. Hence, use cross-validation to select λ.

2.4 LASSO

Ridge regression has one obvious disadvantage, despite shrinking $\beta_1, \ldots, \beta_p$, the final model includes all p predictors.
LASSO works by replacing ℓ_2 with ℓ_1. We minimize over a constrained set $\beta \in \{\beta : \|\beta\|_1 \leq S\}$

$$\hat{\beta} = \min_{\beta} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1$$

$$(\hat{\beta}_L)_j(\lambda) = sign(\hat{\beta}_j)[|\hat{\beta}_j| - \frac{\lambda}{2}]_+$$

3 Linear Classification Models

3.1 Bayes Classification Rule

The optimal Bayes classifier rule is given by:

$$f^*(X) = \begin{cases} +1 & \text{if } \eta(X) \geq \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$$

Where $\eta(X) = \mathbb{P}(Y = 1 | X = x) = \frac{\mathbb{P}(X=x|Y=1)\mathbb{P}(Y=1)}{\mathbb{P}(X=x)}$

This can also be written as $f^*(X) = sign(log(\frac{\mathbb{P}(Y=1|X)}{\mathbb{P}(Y=-1|X)}))$, which is referred to as the log-odds ratio.

3.2 Discriminative Case

Map $\mathbb{P}(Y = 1|X = x)$ (predict directly using observed values)

3.3 Generative Case

Map $\mathbb{P}(X = x|Y = 1)$ (attempt to model $\mathbb{P}(X = x|Y = 1)$)

3.4 Logistic Regression

Logistic Regression model:

$$\eta(X) = \mathbb{P}(Y = y|X = x) = \frac{1}{1 + e^{-y(x^T\beta)}}$$

Let $p = \mathbb{P}(Y = 1|X)$

$$log(\frac{p}{1-p}) = X^T\beta$$

Given a vector of inputs (X, Y), estimate β

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} log(1 + e^{-y_i(\beta^T x_i)})$$

3.5 Discriminate Analysis

Assume $X|Y = y \sim N(\mu_y, \Sigma_y)$, ($\mu_y, \Sigma_y$ are the mean vector and covariance matrix of $X|Y = y$). Model $\mathbb{P}(X = x|Y = y)$ as:

$$\mathbb{P}(X = x|Y = y) = (2\pi)^{-d/2}|\Sigma_y|^{-\frac{1}{2}} exp(-\frac{1}{2}(x - \mu_y)^T\Sigma_y(x - \mu_y))$$

Thus, the Bayes optimal classifier becomes:

$$f^*(x) = sign\left(log\frac{\Sigma_{-1}}{\Sigma_1} + 2log\frac{\mathbb{P}(Y = 1)}{\mathbb{P}(Y = -1)} - (X - \mu_1)^T\Sigma_1^{-1}(X - \mu_1) + (X - \mu_{-1})^T\Sigma_{-1}^{-1}(X - \mu_{-1})\right)$$

The classifier is linear if $\Sigma_1 = \Sigma_{-1} = \Sigma$

$$f^*(x) = sign\left(2(\mu_1 - \mu_{-1})^T\Sigma^{-1}X + \mu_{-1}^T\Sigma^{-1}\mu_{-1} - \mu_1^T\Sigma^{-1}\mu_1) + 2log\frac{\mathbb{P}(Y = 1)}{\mathbb{P}(Y = -1)}\right)$$

3.6 Support Vector Machines

Separable case:

$$\begin{aligned} \min_{\beta} \quad & \beta^T\beta \\ \text{s.t.} \quad & y_i(\beta^T x_i) \geq 1 \end{aligned}$$

Inseparable case (cannot satisfy $y_i(\beta^T x_i) \geq 1$):

$$\min_{\beta} \quad \lambda\beta^T\beta + \sum_{i=1}^{n} \max(0, 1 - y_i\beta^T x_i), \lambda > 0$$

This can also be represented as a quadratic programming problem. Let $t_i = \max(0, 1 - y_i \beta^T x_i)$, A = m x n with rows as data points, D be an m x m diagonal matrix with $D_{ii} = 1 \; if \; x_i \in P_+, \; else \; -1$, $e = (1, 1, \ldots, 1)^T$)

$$\begin{aligned} \min_{w} \quad & \lambda w^T w + e^T t \\ \text{s.t.} \quad & D(Aw + eb) + t \geq e, \\ & t \geq 0 \end{aligned}$$

4 Kth-nearest neighbor

Given a positive integer K and a test observation x_0, identify the K points in the training data that are closest to x_0, denoted by N_0. Model probability of class j as:

$$\mathbb{P}(Y = j | X = x_o) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

5 Decision Trees

Model data based on decisions. Rectangles can be achieved by making successive binary splits on predictor variables $X_1, \ldots, X_p$. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.

Classification Trees:
Let $(x_i, y_i), i = 1, \ldots, n$ be our training data, where $y_i \in 1, \ldots, K$, $x_i \in \mathbb{R}^p$. We want m rectangles $R_1, \ldots, R_m$. Each R_j has a class label $c_j \in 1, \ldots, K$. Given a new point, the classification is:

$$\hat{f}^{tree}(x) = \sum_{j=1}^{m} c_j I(x \in R_j) = c_j$$

We can also estimate the probability a label k lies in region R_j

$$\hat{p}_k(R_j) = \frac{1}{n_k} \sum_{x_i \in R_j} I(y_i = k)$$

which finds the proportion of points in the region that are of class k.

Regression Trees:
Just as in classification, the regression function takes the form

$$\hat{f}^{tree}(x) = \sum_{j=1}^{m} c_j I(x \in R_j) = c_j$$

However, we now just take c_j to be the average response of points in the region:

$$c_j = \frac{1}{n_j} \sum_{x_i \in \mathbf{R}_j} y_i$$

5.1 CART Algorithm

1. Start by defining regions

   $$R_1 = \{X \in \mathbb{R}^p; X_j \leq s\}, R_2 = \{X \in \mathbb{R}^p; X_j \geq s\}$$

   Choose j, s by minimizing the misclassification error:

   $$\min_{j,s}([1 - \hat{p}_{c_1}(R_1)] + [1 - \hat{p}_{c_2}(R_2)])$$

   Here c_1 is the most common class in R_1 and c_2 is the most common class in R_2.

2. Repeat this for our newly created regions R_1, R_2.

3. We will get a big tree T_0. We prune this tree, we collapse some of the leaves into the parent noes.

   Let $|T|$ denote the number of leaves. Define the cost complexity as:

   $$C_\alpha(T) = \sum_{j=1}^{|T|}[1 - \hat{p}_{c_j}(R_j)] + \alpha|T|$$

   Seek $T \in T_0$ that minimizes $C_\alpha(T)$. Do this by pruning the weakest leaf one at a time.

4. Choose α by using k-fold cross-validation.

For regression trees, use the squared error loss instead of the missclassification error.

5.2 Bagging

Make B decision trees:

$$f^1(x), \ldots, f^B(x)$$

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

Training data: D = $(x_1, y_1), \ldots, (x_n, y_n)$

$f^b(x)$: pick $'n'$ samples from D with replacement

5.3 Random Forests

Same idea as bagging but we attempt to decorrelate the trees. Do so by not considering all the predictors for each split. Typically if there are m predictors we take $p = \sqrt{m}$.

5.4 Adaboost

We have training data $(x_1, y_1), \ldots, (x_n, y_n)$

1. Initialize weights $w_1 = \frac{1}{n}, \ldots, w_n = \frac{1}{n}$
2. for $b = 1 : B$
   (a) fit decision tree $\hat{f}^{(b)}$ with weights $w_1, \ldots, w_n$
   (b) $e_b = \frac{\sum_{i=1}^{n} w_i 1_{\{y_i \neq \hat{f}^{(b)}\}}}{\sum_{i=1}^{n} w_i}$
   (c) $\alpha_b = \frac{1}{2} log\big[\frac{1-e_b}{e_b}\big]$
   (d) Update weights $w_i = w_i exp\big[-\alpha_b y_i \hat{f}^{(b)}(x_i)\big]$ for all i
3. final model $\hat{f}_{boost}(x) = sign\big[\sum_{i=1}^{B} \hat{f}^{(b)}(x)\alpha_b\big]$

6 Splines

6.1 Cubic Spline

A cubic spline with K knots can be modeled as:

$$f(x) = \beta_0 + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots + \beta_{k+3} b_{k+3}(x)$$

6.2 Smoothing Splines

$$\min_g \sum_{i=1}^{n} (y_i - g(x_2))^2 + \lambda \int g''(t)^2 dt$$

Here, $\lambda > 0$ is a *tuning parameter*.

7 Additive Models

Let $X \in \mathbb{R}^d$ and $y \in \mathbb{R}$ Our GAM model is:

$$y = \sum_{j=1}^{d} f_j(x_j)$$

To solve for f_j we have the following problem:

$$\min_{f_1,\ldots,f_d} \mathbb{E}(y - \sum_{j=1}^{d} f_j(x_j))^2$$

8 Kernel Methods

The idea of Kernel Methods is your estimates of β is essentially a linear combination of your training samples.
For OLS:

$$\hat{\beta} = X^T(X(X^TX)^{-1}X^TY) = X^T\alpha$$

Define a Kernel (inner-product kernel): $k(u, v) = u^T v$

Given a new sample $\bar{x}$, the prediction is:

$$\bar{x}^T \sum x_i \alpha_i$$

$$= \alpha_1 k(\bar{x}, x_1) + \ldots + \alpha_n k(\bar{x}, x_n)$$

You can change the linear kernel to other kernels.
Polynomial Kernel:

$$k(u, v) = (1 + u^T v)^p$$

Gaussian Kernel:

$$k(u, v) = exp(-\lambda \|u - v\|_2^2)$$

8.1 Kernel Ridge Regression

$$\alpha = (XX^T + \lambda I)^{-1} Y$$

Our prediction is:

$$\sum \alpha_i k(\bar{x}^T, x_i)$$

8.2 Kernel SVM

Let α be the solution given to us by the SVM optimization problem where $\hat{\beta} = \sum \alpha x$
Given a new test sample, the prediction is:

$$sign(\sum \alpha_i k(\bar{x}^T, x_i))$$

9 Resampling Methods

9.1 K-fold Cross-validation

The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set. In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set — are used to fit the model. This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set.

Solution: We randomly divide the training data into K equal sized parts. We leave out part k, then fit the model on the other k-1 parts. We do this for $k = 1, \ldots, K$,then combine the results.

Denote the K part of the training data as $C_1, \ldots, C_k$, and let each observation have n_k observations. Compute:

$$CV_k = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k$$

Where $MSE_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$. When $K = n$, we have leave-one out cross-validation.

9.2 Bootstrap

What bootstrap accomplishes:

- quantify the uncertainty associated with a given estimator or statistical learning method
- provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.

What bootstrap does:
Boostrap mimics the process of obtaining new data sets from one single dataset, so we can estimate the variability of our estimate without generating additional samples. We instead obtain distinct data sets by repeatedly sampling observations from the original data set with replacement. Each of these 'bootstrap data sets' is created by sampling with replacement, and is the same size as our original dataset
In notation, suppose we repeat this procedure B times, and produce B different bootstrap data sets $Z^{*1}, \ldots, Z^{*B}$, and the corresponding estimates $\hat{\alpha}^{*1}, \ldots, \hat{\alpha}^{*B}$. We can estimate the standard error using the formula:

$$SE_B(\hat{\alpha}) - \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \bar{\hat{\alpha}}^{*})^2}$$

where $\bar{\hat{\alpha}}^{*}$ is the average of $\hat{\alpha}^{*}$

Classic Bootstrap C.I.: $\hat{\alpha} \pm SE * z_{0.025}$

Bootstrap Percentile C.I. (works better for skewed data): Consider the 2.5th and 97.5th percentile of $\hat{\alpha}^{*1}, \ldots, \hat{\alpha}^{*B}$

9.3 Bag of Little Bootstrap

The efficacy of bootstrapping depends on two things: the size of the dataset and the number of resamplings. Both need to be high in order to have accurate bootstrapping results. However, when the size of the dataset becomes too large, in otherwords dealing with "big data", the computational efficiency of performing bootstrap dramatically decreases.

Blb is a technique that combines the map reduce algorithm with bootstrapping to create a robust and computationally efficient resampling method that is scalable for large amounts of data. First, the dataset is divided into n approximatly equal sized subsets. Bootstrapping is then performed on each of these subsets. The results of each subset are then averaged.

10 SVD

Any non-zero real (m×n) matrix A of rank $r > 0$ can be factored as:

$$A = P\Sigma Q^T$$

P (m x r) orthanormal columns
Σ (r x r) diagonal matrix with eigenvalues of A
Q^T (r x n) orthanormal rows

1. Numerical Applications:

   Let $\sigma_1 > \ldots > \sigma_n$. Define the condition number of the matrix as $\frac{\sigma_1}{\sigma_n}$

   An ill-conditioned matrix has a large condition number (Almost Singular, computation of inverse prone to high numeric errors).

2. Image Compression:

   For an image matrix, we only need to keep the large σ

3. Low-rank Approximations:

   We can choose the first few basis vectors to approximate our matrix.

11 Dimension Reduction

11.1 PCA

Data $X_1, \ldots, X_n$ that is p dimensions arranged into a nxp data matrix $\mathbb{X}$. The idea is to change the the axis to $W = (w_1, \ldots, w_p)$, called the principle component loading vectors, to maximize the variance

$$\underbrace{Z}_{(n\times p)} = \underbrace{\mathbb{X}}_{(n\times p)}\underbrace{W}_{(p\times p)},$$

The rows are called the principle component scores. These rows also happen to be uncorrelated. w_i is just the ith eigenvector of the sample covariance matrix ($\mathbb{X}_c^T\mathbb{X}_c$, $\mathbb{X}_c$ is the centered data matrix)

We can do dimension reduction by taking the first $w_1, \ldots, w_k$. Keeping the first k σ will keep $\frac{\sigma_1+\ldots+\sigma_k}{\sigma_1+\ldots+\sigma_p}$ of the variation.

We can also reconstruct the original by finding the approximation of $X_j - \overline{X}_n$ by only using the first k coordinates in the new basis, and then adding back the mean:

$$\hat{X}_j = \overline{X}_n + \sum_{i=1}^{k} Z_{ji}w_i.$$

11.2 CCA

Two sets of variables $X = \begin{pmatrix} X_1 \\ \vdots \\ X_p \end{pmatrix}$ and $Y = \begin{pmatrix} Y_1 \\ \vdots \\ Y_r \end{pmatrix}$.

Like PCA, we can define U, V as linear combinations of X, Y.

$$U_i = a_{i1}X_1 + \ldots + a_{ip}X_p$$

$$V_j = a_{j1}X_1 + \ldots + a_{jq}X_q$$

Let:

$$(U_i, V_i)$$

be the ith canonical pair, which are iteratively found using

$$\begin{aligned} \max \quad & corr(U_i X, V_i Y) \\ \text{s.t.} \quad & corr(U_i X, UX) = 0, \\ & corr(V_i X, VX) = 0 \end{aligned}$$

Similar to PCA, in practice, we just use the sample covariance matrix of X and Y, which we will denote $S_{XX} \in \mathbf{R}^{pxp}, S_{YY} \in \mathbf{R}^{qxq}$ and the cross sample covariance matrix as $S_{XY} \in \mathbf{R}^{pxq}$.
The solution matrices U, V satisfy:

$$S_{XX}^{-1} S_{XY} S_{YY}^{-1} S_{YX} U = UD^2$$

where $D^2 = \text{diag}(\lambda_1^2, \ldots, \lambda_r^2)$, and

$$V = S_{YY}^{-1} S_{YX} U D^{-1}.$$

12 Clustering

12.1 K-Means

Set of points $X = (x_1, \ldots, x_n)$, want to group our data into K clusters $c = c_1, \ldots, c_k$. K-means finds a partition so that the squared error between the mean of a cluster and the points in the cluster is minimized. Let u_k be the mean of cluster c_k. Define the following:

$$J(c_k) = \sum_{x_j \in c_k} \|x_k - u_k\|^2$$

$$J(c) = \sum J(c_k)$$

1. Select initial partition with K-clusters. Do steps 2 and 3 until clusters stabilize
2. Generate new partition by assigning each point to its closest cluster center
3. Compute new cluster centers

12.2 Spectral clustering

Given data points $x_1, \ldots, x_n$ and similarities $w(x_i, x_j)$, construct similarity graph: G(V,E,W)
V: vertices (data points)
E: edge if similarity > 0
W: edge weights (similarities)
Example of similarity - Gaussian similarity function: $W_{ij} = e^{\frac{\|x_i - x_j\|_2^2}{2\sigma^2}}$, σ controls size of neighborhood.

Normalized Laplacian:

$$L' = I - D^{-1} W$$

where D is a diagonal matrix of d_i, where $d_i = \sum_{j:(i,j) \in E} w_{ij}$

Algorithm:

1. Input: Similarity matrix W, k clusters
2. Build similarity graph

3. compute largest k eigenvectors of matrix L'
4. Build matrix $V \in \mathbb{R}^{nxk}$ with eigenvectors as columns
5. interpret rows of V as new data points $Z_i \in \mathbb{R}^k$
6. cluster the points Z_i using k-means in $\mathbb{R}^k$

13 Feed Forward Neural Networks

13.1 Architecture

Let m be the number of neurons in the input layer. Weight from i-th node to j-th node is denoted by w_{ij}. We have the value that enters node i in the first hidden layer is:

$$L(X) = w_{1i}x_1 + w_{2i}x_2 + \ldots + w_{mi}x_m + b$$

At the node there is an activation function, which determines if node is used or not.

$$\phi(L(X))$$

Activation functions

1. Relu: $ReLu(x) = max(0, x)$
2. Logistic Sigmoid: $\phi(x) = \frac{e^x}{e^x+1}$
3. Hyperbolic tangent function: $\phi(x) = tanh(\frac{x}{2}) = \frac{e^x-1}{e^x+1}$

13.2 Learning

Let training data set $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ contain N points with m features. Labels $y_1, \ldots, y_N$.

Take $X_1^{(1)}$. This becomes the first layer $V_0 = (x_1^{(1)}, \ldots, x_m^{(1)})$. K-th layer ouputs $V_k = ReLu(A_kV_{k-1} + b_k)$, A_k is $N_{k-1} \times N_k$.

Compare the final layer to the target value using a loss function. Adjust the weights to minimize the error. This becomes an optimization problem, which we should solve using Stochastic Gradient Descent (See section 4.3). Common loss functions used:

1. Square Loss: $L(w) = \frac{1}{N}\sum_{i=1}^{N}(F(w, x_i) - y_i)^2$
2. Hinge Loss: $L(w) = \frac{1}{N}\sum_{i=1}^{N}\max\{0; 1 - y_iF(w, x_i)\}, y_i = -1, 1$
3. Cross-Entropy Loss: $L(w) = -\frac{1}{N}\sum_{i=1}^{N}[y_i\log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)], y_i = 0, 1, \hat{y}_i = \frac{1}{1+e^{w^t x_i}}$

14 Convolutional Neural Network

14.1 Architecture

Convolution Layer:
The convolution of an image x with a kernel k is:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q}k_{p,q}$$

Let the imput matrix be of size I, P the amount of zero padding, and S the stride (number of pixels moved after each operation). The output size of the convolution layer is

$$O = \frac{I - F + P_{start} + P_{end}}{S} + 1$$

Suppose we have K filters.
Input: I x I x C, Output: O x O x K

Pooling Layer:
Downsizing operation, can take the max or average
Input: I x I x C, Output: O x O x C

Fully Connected Layer:
Operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.

14.2 YOLO

1. Divide input image into a G x G grid
2. For each grid cell, run a CNN that predicts y from the following form:

$$y = [\underbrace{p_c, b_x, b_y, b_h, b_w, c_1, c_2, \ldots, c_p}_{\text{repeated } k \text{ times}}, \ldots]^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

   p_c is the probability of detecting an object, b_x, b_y, b_h, b_w (center, height, width) are the properties of the detected box, $c_1, \ldots, c_p$ is a one-hot representation of which of the p classes were detected, and k is the number of boxes.
3. Run non-max suppression algorithm.

14.3 R-CNN

Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.

15 Recurrent Neural Network

15.1 Architecture

Let $a^{<t>}$ be the activation at time t, $x^{<t>}$ be the input at time t, and $y^{<t>}$ be the output at time t. At each timestep t, the activation $a^{<t>}$ and output $y^{<t>}$ are:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

15.2 Learning

The loss function is defined as: $L(\hat{y}, y) = \sum_{t=1}^{T_y} L(\hat{y}^{<t>}, y^{<t>})$
Backpropagate:

$$\frac{dL}{dW} = \sum_{t=1}^{T_y} \frac{dL}{dW}$$

15.3 Handling Long-term Dependencies

Multiplicative nature of the gradient causes a phenomenon known as vanishing/exploding gradient, as a small gradient will be exponentially smaller and large gradient exponentially bigger.

To handle exponential gradients, gradient clipping (capping the maximum gradient size) is employed.

To handle vanishing gradients, gates (characterized as: $\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$) are employed:

- Update gate Γ_u: How much past should matter now? (GRU/LSTM)
- Relevance gate Γ_r: Drop previous information? (GRU/LSTM)
- Forgot Γ_f: Erase a cell or not? (LSTM)
- Output gate Γ_o: How much to reveal of a cell? (LSTM)

15.4 GRU Architect

Inputs: Previous outputs: $a^{<t-1>}$, $c^{<t-1>}$, new input: $x^{<t>}$
Hidden Unit: $\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * a^{<t-1>}, x^{<t>}] + b_c)$
Outputs: $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$, $a^{<t>} = c^{<t>}$

15.5 LSTM Architect

Inputs: Previous outputs: $a^{<t-1>}$, $c^{<t-1>}$, new input: $x^{<t>}$
Hidden Unit: $\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * a^{<t-1>}, x^{<t>}] + b_c)$
Outputs: $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$, $a^{<t>} = \Gamma_o * c^{<t>}$

16 Optimization

16.1 Convex Optimization Problem

Takes the form:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, i = 1, \ldots, m, \\ & a_i^T x = b_i, i = 1, \ldots, p \end{aligned}$$

$f_0, \ldots, f_m$ are convex functions (constraints create a convex set/$f_i(\lambda x + (1 - \lambda)y) \leq \lambda f_i(x) + (1 - \lambda) f_i(y), \forall x, y \in \mathbb{R}^n, \lambda \in [0, 1]$/any straight line between any two points in the set is contained inside the set.

Lagrangian: $L(x, \lambda, v) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} v_i h_i(x)$

16.2 Gradient Descent

1. start with x_0 inside domain
2. At k-th iteration, if $\|\nabla f(x_k)\| \geq \epsilon$, continue. Else we find our optimum.
3. choose $v_k = -\nabla f(x_k)$
4. Give step size $a_k > 0$/choose $\alpha_k = \min_{\alpha} f(x_k - \alpha \nabla f(x_k))$
5. iterate $x_{k+1} = x_k + \alpha_k v_k$

16.3 Stochastic Gradient Descent

Optimization problem takes the form $\min_{w} L(X) = \frac{1}{N} \sum_{i=1}^{N} f_i(w)$, N is large. Assume it is too costly to compute this using normal gradient descent. Instead, choose B as a subset of the indices of N. Instead of $w_{k+1} = w_k - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(w_k)$, we calculate:

$$w_{k+1} = w_k - \eta_k \frac{1}{|B|} \sum_{i \in B} \nabla f_i(w_k)$$

To ensure convergence of Stochastic Gradient Descent, a decreasing η must be used. A constant η will converge for normal Gradient Descent.

16.4 KKT

KKT Conditions:

- primal constraints:$f_x(x) \leq 0, i - 1, \ldots, m, h_i(x) = 0, i = 1, \ldots, p$
- duel constraints: $\lambda \geq 0$
- complementary slackness: $\lambda_i f_i(x) = 0, i = 1, \ldots, m$
- gradient of Lagrangian wrt x vanishes: $\nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} v_i \nabla h_i(x) = 0$

KKT for convex problems
If $\tilde{x}, \tilde{\lambda}, \tilde{v}$ satisfy the KKT conditions for a convex problem, they are optimal.

KKT for NLP (no convexity assumed)
Can only guarantee local optimum for NLP. The mentioned KKT conditions are now the first order KKT conditions. Also need to check for second-order condition:

$$s^T \nabla^2_{xx} L(\bar{x}, \bar{\lambda}, \bar{v}) \geq 0$$

If $\bar{x}, \bar{\lambda}, \bar{v}$ satisfy both the first and second order conditions, $\bar{X}$ is a local minimum (Strict if 2nd order condition is strict)

16.5 Branch and Bound

This algorithm is used to solve IP problems. If you solve the LP relaxation of a pure IP and obtain a solution in which all variables are integers, then the optimal solution to the LP relaxation is also the optimal solution to the IP.

1. (Fathoming/Bounding)
2. (Branching)

17 Network Flow

17.1 LP form

Nodes: the sources, destinations, and intermediate points. Denote $1 \dots n$.
Arcs: the transportation links connecting nodes are termed arcs. Denote $i - j$ meaning from node i to node j.

Define x_{ij} as number of units moved from node i to j using arc i-j. Our optimization problem becomes:

$$\begin{aligned} \min \quad & z = \sum_{i,j} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} - \sum_k x_{ki} = b, i = 1, \dots, n \ (flow\ balance), \\ & l_{ij} \le x_{ij} \le u_{ij}, \ (flow\ capacities) \end{aligned}$$

17.2 Transportation Problem

Want to move units from sources to destinations with no intermediate.

Let a_i be the number of units available at source i ($i = 1, \dots, m$)
Let b_j be the number of units required at destination j ($j = 1, \dots, n$)
Let c_{ij} Per-unit transportation cost from source i to destination j

Assume total units available = total units required ($\sum a_i = \sum b_i$). Let x_{ij} be the number of units moved from source i to destination j. Problem becomes:

$$\begin{aligned} \min \quad & \sum_i \sum_j c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} = a_i, \\ & \sum_i -x_{ij} = -b_i, \\ & x_{ij} \ge 0 \end{aligned}$$

- c1: units from source i should equal units available at source i
- c2: units shipped to destination j should equal the requirements of destination j

17.3 Assignment Problem

Assign n objects to n places depending on performance of object i to place j denoted c_{ij}. Define $x_i j = 1$ if i is assignment to j, 0 otherwise.

$$\begin{aligned} \min \quad & \sum_i \sum_j c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} = 1, \\ & \sum_i -x_{ij} = 1, \\ & x_{ij} = 0, 1 \end{aligned}$$

- c1: i is assigned to only one j
- c2: j has only one i assigned to it

17.4 Max Flow Problem

Send as much material from node s (source) to not t (sink). Let u_{ij} be the capacity for arc i-j. Let v denote material sent from s to t. Let s_{ij} represent flow from node i to j over arc i-j.

$$\begin{aligned}
\max \quad & v \\
\text{s.t.} \quad & \sum_j x_{ij} - \sum_k x_{ki} = v, i = s, \\
& \sum_j x_{ij} - \sum_k x_{ki} = -v, i = t, \\
& \sum_j x_{ij} - \sum_k x_{ki} = 0, for\ other\ nodes, \\
& 0 \leq x_{ij} \leq u_{ij}
\end{aligned}$$

Alternate formulation: Introduce arc x_{ts}, has unlimited capacity.

$$\begin{aligned}
\max \quad & x_{ts} \\
\text{s.t.} \quad & \sum_j x_{ij} - \sum_j x_{kj} = 0, i = 1, 2, \ldots, n
\end{aligned}$$

17.5 Shortest Path Problem

Given network with distance c_{ij}, find path from source to sink that has the shortest total distance.

$$\begin{aligned}
\min \quad & \sum_j \sum_j c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_j x_{ij} - \sum_k x_{ki} = 1, i = s, \\
& \sum_j x_{ij} - \sum_k x_{ki} = -1, i = t, \\
& \sum_j x_{ij} - \sum_k x_{ki} = 0, for\ other\ nodes, \\
& x_{ij} \geq 0
\end{aligned}$$

18 Integer Programming

18.1 Capitol Budgeting

Given projects $p_1, \ldots, p_n$, over t years, with expenditures e_{ij} (expenditure for ith year jth project), with available funds each year $f_1, \ldots, f_n$ which gives returns $r_1, \ldots, r_n$.

$$x_j = \begin{cases} 1 & \text{if project } j \text{ is selected} \\ 0 & \text{if project } j \text{ is not selected} \end{cases}$$

$$\begin{aligned} \max \quad & z = \sum_n p_j x_j \\ \text{s.t.} \quad & \sum_n x_j e_{ij} = f_i, i = 1, \ldots, t, \\ & x_j = 0, 1 \end{aligned}$$

18.2 Set-Covering Problem

Select minimal amount of points to cover set. Here, $j = 1, \ldots, n$

$$x_j = \begin{cases} 1 & \text{if node j is chosen} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} \min \quad & z = \sum_n x_j \\ \text{s.t.} \quad & x_i + x_k \geq 0 \text{ if there exists arc from i to k}, \\ & x_j = 0, 1 \end{aligned}$$

18.3 Fixes-Charge Problem

Request services from n companies, each who charge a flat fee and a variable fee. Let $x_1, \ldots, x_n$ be the units/time of each company. Define $y_j = 1$ if $x_j > 0$ and 0 if $x_j = 0$, $j = 1, \ldots, n0$. Let M be the total units demanded.

$$\begin{aligned} \min \quad & z = \sum_n x_j y_j \\ \text{s.t.} \quad & \sum_n x_j = M, \\ & x_j \leq M y_j, j = 1, \ldots, n, \\ & x_j \geq 0, y_j = 0, 1 \end{aligned}$$

18.4 Either-Or Constraint

Either $f(x_1, \ldots, x_n) \leq 0$ or $g(x_1, \ldots, x_n) \leq 0$, set constraints as:

$$\begin{gathered} f(x_1, \ldots, x_n) \leq My \\ g(x_1, \ldots, x_n) \leq M(1 - y) \\ y = 0, 1, \ M \ Large \end{gathered}$$

18.5 If-then Constraint

If $f(x_1, \ldots, x_n) > 0$ is satisfied, then constraint $g(x_1, \ldots, x_n) \geq 0$ must be satisfied, while if $f(x_1, \ldots, x_n) > 0$ is not satisfied, then $g(x_1, \ldots, x_n) \geq 0$ may not have to be satisfied. Set constraints as:

$$\begin{gathered} f(x_1, \ldots, x_n) \leq M(1 - y) - \epsilon \\ -g(x_1, \ldots, x_n) \leq My \\ y = 0, 1 \end{gathered}$$

18.6 Traveling Salesperson Problem

A salesperson visits cities $1, 2, \ldots, n$ in some order and ends up at the city where he started. c_{ij} is the cost of going from i to j.

Consider the undirected case $(i, j) = (j, i)$. Let the graph be denoted by $K_n = (V, E)$. Let U denote any proper subset of nodes.

$$\begin{aligned}
\min \quad & z = \sum_{ij} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j:(i,j)\in E} x_{ij} = 2, i = 1, \ldots, n, \\
& \sum_{i\in U j\in \bar{U}} x_{ij} \geq 2, \\
& x_{ij} = 0, 1
\end{aligned}$$

Consider the directed case. $x_{ij} = 1$ if there is a path from i to j, otherwise 0.

$$\begin{aligned}
\min \quad & z = \sum_{ij} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j:j\neq i} x_{ij} = 1, i = 1, \ldots, n, \\
& \sum_{i:i\neq j} x_{ij} = 1, j = 1, \ldots, n, \\
& \sum_{i\in S j\in S i\neq j} x_{ij} \leq |S| - 1 \text{ for all vertex subset S with } 3 \leq |S| \leq n - 3, \\
& x_{ij} = 0, 1
\end{aligned}$$

18.7 Solving IP problems

See **section 16.5: Branch and Bound**