

STATS 415: Data Mining and Statistical Learning

Conceptual Summary

University of Michigan at Ann Arbor

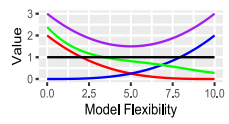
Last Edit Date: 12/14/2023

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. This personal note is adapted (derived) from the slides and codes from *Professor Jackson Loper, Ji Zhu, and Liza Levina*. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

1. Statistical Overview

1. Bias, Variance, Flexibility, Accuracy Trade-off



Bayes Error: The irreducible error, always present in models.
Squared Bias: $(E(\hat{Y}) - Y)^2$. Occurs when an algorithm produces prejudiced results (method's assumptions are not met).

Variance: $E[(\hat{Y} - E(\hat{Y}))^2]$. The measure of spread in data from its mean.

Test & training error: Usually training error decreases as we train models; however, test error may increase at a point, overfitting.

Total error: $E[(Y - \hat{Y})^2] = \text{Bias}^2 + \text{Variance} + \text{Bayes Error}$

II. Regression vs. Classification & Inference vs. Prediction

- The response (Y) of a regression problem is **numeric**, but **classification** problem is **categorical**.
- Inference** is to use models to learn about things that happened in the **past**, but **prediction** is to use models to make predictions in the **future**.

2. KNN (Flexible, Regression or Classification)

$\hat{f}_{KNN}(x_0) = \frac{1}{K} \sum_{i \in N_0} y_i$, where K is the number of neighbors, N_0 is the set of K nearest points in the training data, and y_i is the training response. Example below.

Regression			Classification		
x_1	x_2	y	x_1	x_2	y
1	2	77	1	2	c
4	3	20	4	3	d
1	8	12	1	8	d
2	3	91	2	2	d

Given $x = (6, 4)$, we calculate the Euclidean distance to find its neighbors.

$$d_1 = \sqrt{(6-1)^2 + (4-2)^2} = \sqrt{29} \quad d_2 = \sqrt{(6-4)^2 + (4-3)^2} = \sqrt{5} \\ d_3 = \sqrt{(6-1)^2 + (4-8)^2} = \sqrt{41} \quad d_4 = \sqrt{(6-2)^2 + (4-3)^2} = \sqrt{17}$$

$$\hat{f}_{KNN}(x) = (d_2)/1 \quad \hat{f}_{KNN}(x) = (d) = d, K = 1 \\ \hat{f}_{KNN}(x) = (d_2 + d_4)/2 \quad \hat{f}_{KNN}(x) = (d, d) = d, K = 2 \\ \hat{f}_{KNN}(x) = (d_2 + d_4 + d_1)/3 \quad \hat{f}_{KNN}(x) = (d, d, c) = d, K = 3 \\ \hat{f}_{KNN}(x) = (d_2 + d_4 + d_1 + d_3)/4 \quad \hat{f}_{KNN}(x) = (d, d, c, d) = d, K = 4$$

Advantages: 1) No training period. 2) Easy implementation.

Disadvantages: 1) Performs badly with large data. 2) Performs badly with high dimensional data. 3) Sensitive to noise and missing data. 4) Need scaling on features.
Note: When $K = 1$, the prediction is the same as data. Low bias but high variance.

3. Linear Regression (Inflexible, Regression)

I. Basics

$$Y = X\beta + \varepsilon, \text{ where } Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}, X = \begin{bmatrix} 1 & \dots & x_{1p} \\ \vdots & \ddots & \vdots \\ 1 & \dots & x_{np} \end{bmatrix}_{n \times (p+1)}, \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_p \end{bmatrix}_{(p+1) \times 1}, \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}_{n \times 1}$$

when we have n observations and p input in each observations.

II. Assumption: $\varepsilon \sim N(0, \sigma^2 I)$ independent identically distributed. $Cov(X_i, X_j) = 0$.

III. Some Properties and Definitions

$$\hat{\beta} = (X^T X)^{-1} X^T Y, E(\hat{\beta}) = \beta, Var(\hat{\beta}) = (X^T X)^{-1} \sigma^2, \sigma^2 = \frac{1}{n} (Y - X\hat{\beta})^T (Y - X\hat{\beta})$$

$$\hat{\varepsilon} = Y - X\hat{\beta}, E(\hat{\varepsilon}) = 0, Var(\hat{\varepsilon}) = (I - H) \sigma^2, H = X(X^T X)^{-1} X^T$$

$$SSTO = \sum (Y_i - \bar{Y})^2 = Y^T Y - n\bar{Y}^2, SSE = \sum (Y_i - \hat{Y}_i)^2 = Y^T Y - \hat{\beta}^T X^T Y, SSR = SSTO -$$

$$SSE = \sum (\hat{Y}_i - \bar{Y})^2 = \hat{\beta}^T X^T Y - n\bar{Y}^2$$

$$MSE = \frac{SSE}{df(SSE)} = \frac{SSE}{n-p} \text{ (unbiased estimator of } \sigma^2), MSR = \frac{SSR}{df(SSR)}, MSTO = \frac{SSTO}{df(SSTO)}$$

$$E(SSE) = (n-p) \sigma^2, E(MSE) = \frac{E(SSE)}{n-p} = \sigma^2$$

IV. ANOVA (Analysis of Variance)

	df	Sum Square	Mean Square	F-score
Regression	p - 1	SSR	MSR	F* = MSR/MSE
Error	N - p	SSE	MSE	
Total	N - 1	SSTO	MSTO	

N is the total number of observations, p is the number of predictors.

V. Coefficient of Determination

$$R^2 = \frac{SSR}{SSTO} = 1 - \frac{SSE}{SSTO} > R^2_{adj} = \frac{MSR}{MSTO} = 1 - \frac{MSE}{MSTO}, R^2 \in [0, 1], R^2 = corr(X, Y)^2.$$

Interpretation: We can say R^2 of the variability in Y can be explained by X .

VI. Coefficient Interpretation

Normal: A one unit increase in $[predictor]$ at a given point will lead to a $[coefficient]$ units increase / decrease in $[response]$ on average, given other variables are fixed.

Categorical: A $[entity]$ with $[predictor category]$ will lead to a $[coefficient]$ units increase / decrease in $[response]$ on average relative to a $[entity]$ with $[predictor category]$ (the one not shown in the result), given other variables are fixed.

Interaction (numeric * category): A one unit increase in $[interaction 1 (numeric)]$ for a $[entity]$ with a $[interaction 2 (categorical)]$ results in increase / decrease in $[response]$ that is

University of Michigan – STATS 415 Conceptual Summary

approximately $[coefficient]$ units less than the decrease in $[interaction 1 (numeric)]$ for a $[entity]$ with $[predictor category]$ (the one not shown in the result) on average, given other variables are fixed.

Polynomial: A one unit increase in the square of $[predictor]$ at a given point will lead to a $[coefficient]$ units increase / decrease in $[response]$ on average, given other variables are fixed.

4. Logistic Regression (Inflexible, Classification for 2 Classes)

I. Basics

Same assumptions as linear regression but use sigmoid function $p(X) = \Pr(Y = 1|X) =$

$$\frac{e^{X\beta}}{1 + e^{X\beta}}. Odds = \frac{p(X)}{1-p(X)} = e^{X\beta}. \text{logit} = \log(Odds) = X\beta.$$

II. Errors and Confusion Matrix

True \ Predicted	0	1	
0	True Negative	False Positive	N
1	False Negative	True Positive	P
	N*	P*	Total

Accuracy = (True Positive + True Negative) / Total

Error rate = (False Positive + False Negative) / Total = 1 - Accuracy

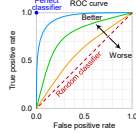
False Positive Rate = False Positive / N **False Negative Rate** = False Negative / P

Sensitivity = 1 - False Negative Rate **Specificity** = 1 - False Positive Rate

III. ROC Curve and AUC

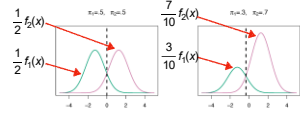
The overall performance of a classifier, summarized over all possible thresholds, is given by the area under the curve (AUC). An ideal ROC curve will hug the top left corner, so the larger area under the ROC curve, the better the classifier. $AUC \in [0.5, 1]$.

Good for uneven classes.



5. Bayesian Approach of Classification (LDA and QDA, Inflexible)

$$\Pr(Y = y|X = x) = \frac{\pi_y f(x|y)}{\sum_y \pi_y f(x|y)}, \text{ where } \pi_y =$$



$\Pr(Y = y)$ is the prior class probability,

$f(x|y)$ is the class conditional probabilities, a probability density function. The numerator $\sum_y \pi_y f(x|y)$ is always added up to 1. For

normal distributions showed in the plot, it will be $p_k(x) = \frac{\pi_k \exp\left(-\frac{1}{2\sigma_k^2} \left(x - \mu_k\right)^T \Sigma_k^{-1} (x - \mu_k)\right)}{\sum_{k=1}^K \pi_k \exp\left(-\frac{1}{2\sigma_k^2} \left(x - \mu_k\right)^T \Sigma_k^{-1} (x - \mu_k)\right)}$ (Normal)

Assumptions: Observation of each class is from a normal distribution (LDA and QDA).

Each of the classes has identical covariance matrices (LDA only). Each of the classes can have different covariance matrices (QDA only).

LDA (special case of QDA)	QDA
$\delta_k(x) = x \frac{\mu_k}{\sigma} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$	$\delta_k(x) = -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log(\pi_k)$

Bayesian Classifier Correction

Idea: Change the prior π to match our population of interest.

Assume X categorical, then the distribution in training population is $\Pr1(Y = y, X = x) = \pi_y f_y(x)$ and in broader population we assert $\Pr2(Y = y, X = x) = \tilde{\pi}_y \tilde{f}_y(x)$. Then we have

$$\frac{\Pr1(Y=y|x)}{\Pr1(Y=no|x)} = \frac{\pi_{yes} f_{yes}(x)}{\pi_{no} f_{no}(x)} = \frac{\pi_{yes} f_{yes}(x)}{\pi_{no} f_{no}(x)} = \frac{\pi_{yes} f_{yes}(x)}{\pi_{no} f_{no}(x)} = \frac{\pi_{yes} (1 - \pi_{yes})}{\pi_{no} (1 - \pi_{yes})}$$

6. Bootstrap (Resampling / Simulation Method, Metric Evaluation)

I. Parametric Bootstrap (Simulation)

Use the observed sample to estimate the population distribution. Then samples can be drawn from the estimated population and the sampling distribution of any type of estimator can itself be estimated.

II. Nonparametric Bootstrap (Resampling)

A sample of sample size as the data is taken from the data **with replacement**. This means we measure 10 samples by replicating some of the samples that we've already seen and omitting others.

Advantage: 1) Don't need distribution. 2) Intuitive feeling. 3) Works well broadly.

Disadvantage: The approximation joint is always be a categorical distribution.

7. Cross Validation (Metric Evaluation, Model Selection)

I. Cross Validation - Train and Test Split

Idea: Given a dataset, randomly select 70% of the data as the training set, 30% as the testing set. We train the model based on the training set and finally evaluate the model with the testing set and get the test error.

II. K-fold Cross Validation

Idea: 1) Shuffle the dataset randomly. 2) Split the dataset into k groups. 3) For each unique group: *i*) take the group and a hold out or test data set; *ii*) take the remaining groups as a training data set; *iii*) fit a model on the training set and evaluate it on the test set; *iv*) retain the evaluation score and discard the model. 4) Summarize the skill of the model by averaging the k evaluation scores of the model.

III. LOOCV (Leave One Out Cross-Validation)

Idea: Special case of k -fold cross validation. It simply uses 1 datapoint as the validation set, the other $N - 1$ datapoints are the training set at each fold. Less bias but higher variance.

8. Regularization (Model Selection, Model Tuning)

I. L2 (Ridge) and L1 (Lasso) Approaches

L2 objective: $\arg\min_{\beta} \left[\sum_{i=1}^n (x_i^T \beta - y_i)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right]$, λ is the penalty term.

L1 objective: $\arg\min_{\beta} \left[\sum_{i=1}^n (x_i^T \beta - y_i)^2 + \lambda \sum_{j=1}^p |\beta_j| \right]$, λ is the penalty term.

L2 & L1 objective: $\arg\min_{\beta} \left[\sum_{i=1}^n (x_i^T \beta - y_i)^2 + \lambda_1 \sum_{j=1}^p \beta_j^2 + \lambda_2 \sum_{j=1}^p |\beta_j| \right]$.

As λ increases, the **training RSS increases steadily**; **test RSS decreases first, then increases**; **variance decreases steadily**; **squared bias increases steadily**; **irreducible error unchanged**.
Note: Ridge penalty is usually **stronger than Lasso**. We can use cross validation to choose a good λ , then having a good model. We can also combine these two penalties. Moreover, these penalties can help us select features while doing regressions.

II. Non penalty Regularization

1. Early stopping of gradient descent

Idea: Before the model starts overfitting, stop training the model. Use cross validation.

2. Data-driven featurization

Idea: Create a new feature based on what we've already had. For example, if we want to predict the MPG from Weight, we can create a new feature called $Weight^2$. Usually decreases bias but increases variance.

1) Forward & Backward Selection

Idea: Forward selection starts empty and adds one feature at a time, while backward selection begins full and removes one feature at a time, both choosing the best fit.

Note: Backward selection requires that the number of observations N is larger than the number of features p because degrees of freedom equal to $N - p > 0$.

2) Principal Component Analysis (PCA)

Idea: 1) Assume the featurization is an orthonormal project of centered data. Refeaturized version of data $Z \in R^M$ is defined by $Z = F(X - \mu)$, where $\mu \in R^p \approx E(X)$ and $F \in R^{M \times p}$ is an orthonormal matrix. That means that $F^T F = I_M$. 2) PCA picks projection, so we don't lose too much information about original features. 3) Given the reconstruction error, PCA picks the directions of maximal variance for the random variable X . **Note:** PCA is a linear autoencoder.

3. Data Augmentation

Idea: Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points. An example is generating different layers from a colored picture. It can have a de-texturized layer, de-colored layer, edge-enhanced layer, and so on. **Note:** With larger dataset, many estimators enjoy lower variance.

9. Splines (Nonlinear, Regression, Classification)

Definition: A spline is a special function defined piecewise by polynomials.

I. Linear Splines

A linear spline with knots at $\xi_k, k = 1, 2, \dots, K$ is a piecewise linear polynomial continuous at each knot. The model is: $y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \dots + \beta_{K+1} b_{K+1}(x_i) + \epsilon_i$, where b_k are basis functions $b_1(x_i) = x_i, b_{k+1} = (x_i - \xi_k)_+$. Note, if $x_i > \xi_k$, then $(x_i - \xi_k)_+ = x_i - \xi_k$, else, $(x_i - \xi_k)_+ = 0$.

II. Cubic Splines

A cubic spline with knots at $\xi_k, k = 1, 2, \dots, K$ is a piecewise cubic polynomial with continuous derivatives up to order 2 at each knot. The model is: $y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \dots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i$, where $b_1(x_i) = x_i, b_2(x_i) = x_i^2, b_3(x_i) = x_i^3, b_{k+3} = (x_i - \xi_k)^3$.

III. Natural Cubic Splines

A natural cubic spline extrapolates linearly beyond the boundary knots. This adds 4 = 2 * 2 extra constraints and allows us to put more internal knots for the same degrees of freedom as a regular cubic spline. It is less variable beyond the range of the data.

IV. Smoothing Splines

Consider fitting a smooth function $g(x)$ to some data: $\arg\min_{g \in \mathcal{G}} \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$. The first term is residual standard error, and the second term is the roughness penalty and controls how wiggly $g(x)$ is. It is regulated by $\lambda \geq 0$, the tuning parameter. Note: the smaller λ the more wiggly, the larger λ the more linear.

V. Example

Given $f(x) = (x - 3)^3$ if $x > 3$, $f(x) = (x - 3)^2$ if $x < 3$. There is only 1 knot $x = 3$. It is not a natural cubic spline because it is **not linear** beyond the boundary knot. It is also not cubic spline because its second derivative is **not continuous**. For $x > 3$, $f'(x) = 3(x - 3)^2$, $f(x)'' = 6(x - 3)$. For $x < 3$, $f'(x) = 2(x - 3)$, $f''(x) = 2$. $f''(2) = -6 \neq 2$.

	Knot replacement method	Spline fitting method
Cubic splines	Hand-pick, percentiles of input data, or spaced regularly throughout range of input data.	Minimize squared error on training data.
Smoothing splines	Place knot at every training point.	Minimize 1) sum of squared error on training data and 2) a penalty on the integrated squared second derivative of the function.

10. Supported Vector Machines (SVM)

SVM can accommodate **nonlinear** relationships. It is used for binary classification, but there are extensions for multi-way classification and regression. **There are two innovations of SVM:** 1) there is a special way designed to control estimator variance in binary classification problems; 2) the kernel trick allows us to use infinite-dimensional featurization.

Terminology and Properties

1. Terminology

Separating hyperplane: a linear boundary between the classes.

Margin: distance from the separating hyperplane to each class (need to maximize).

Support vectors: data points on the margin boundary.

2. Property of Hyperplanes

Hyperplane is defined by $F = \{x: \beta_0 + x^T \beta = 0\}$. β is perpendicular to F . For any point x_0 in the hyperplane, $x_0^T \beta = -\beta_0$. Signed distance from point x to F is $\frac{1}{\|\beta\|} (x^T \beta + \beta_0) =$

$\frac{\beta}{\|\beta\|}, x = x_0\}$, x_0 is any point in the plane. Minimize distance.

3. Overlapping Classes

m : The minimum perpendicular distance from the point to the hyperplane.

ξ_i : The slack variables. B : The tuning parameter.

Objective function: $\frac{\gamma_i}{\|\beta\|} (\beta_0 + x_i^T \beta) \geq m(1 - \xi_i), \xi_i \geq 0$.

$\sum \xi_i \leq B$. **Note: SVM wants small ξ_i , large margin.**

- $\xi_i = 0$ (points outside the margin), correctly classified.

- $0 < \xi_i < 1$ (points inside the margin), but correctly classified.

- $\xi_i > 1$, misclassified.

11. Tree (Flexible)

I. Classification Tree

I. Splitting Rules: Binary splits. The proportion of observations from class k in region R_m is $p_k(m) = \frac{n_k}{n} = \frac{1}{n} \sum_{x \in R_m} I(Y_i = k)$, where R_m is the part of the feature corresponding to node m , containing n_m observations. Observations are classified to the majority class $k(m) = \text{argmax}_k p_k(m)$. It is a **greedy method**, **no global optimization**.

2. Impurity: A parent node P with n_p observations to split into two children nodes L and R

with n_L and n_R observations, we want to maximize $M(s) = Q(P) - \left(\frac{n_L}{n_p} Q(L) + \frac{n_R}{n_p} Q(R) \right)$.

This measures the average increase in “quality” (decrease in impurity) from parent to children.

3. Metrics

1) Misclassification rate: $Q_m = \frac{1}{n} \sum_{x \in R_m} I(Y_i \neq k) = 1 - \max_k p_k(m)$

2) Gini index: $Q_m = \sum_{k=1}^K p_k(m)(1 - p_k(m))$ (more sensitive to node purity)

3) Entropy (deviance): $Q_m = -\sum_{k=1}^K p_k(m) \log p_k(m)$

4. **Tree Size:** The tree size $|T|$ (number of leaves) is a measure of model complexity. Too large size can overfit, too small may underfit.

II. Regression Tree

Idea: Divide the space into boxes by splitting one variable at a time. Within each both R_j , estimate the function by the average of all training point responses in that box $\hat{y}_{R_j}$. Then

minimize for final boxes $R_1, \dots, R_{|T|}$: $\sum_{j=1}^{|T|} \sum_{i: x_i \in R_j} (y_i - \hat{y}_{R_j})^2$. For impurity, suppose we have $R_1(j, s) = \{X|X_j < s\}$, $R_2(j, s) = \{X|X_j \geq s\}$, just split by minimizing $\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1(j, s)})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2(j, s)})^2$ on j and s .

III. Bagging (Bootstrap Aggregating)

Idea: Average a given procedure over many bootstrap samples to **reduce its variance**.

1. Classification tree: Get a version $T^{(b)}$ by training on each of the bootstrap samples $b = 1, \dots, B$. Then bagging predicts $\hat{p}_{\text{bag}}(y|x) = \sum_b T^{(b)}(y|x)/B$.

2. Regression tree: Get a version $T^{(b)}$ by training on each of the bootstrap samples $b = 1, \dots, B$. Then bagging predicts $\hat{f}_{\text{bag}}(x) = \sum_b T^{(b)}(x)/B$.

3. Subbagging: resamples rows without replacement.

Advantage: Reduce variance. **Disadvantage:** Each base estimator T_b has high variance.

IV. Variable Importance

Summarize the importance of each predictor using relative variable importance. **1)** Find all splits using a given variable, across all trees. **2)** Compute the quality of split for each one. **3)** The sum of the quality if the predictor's overall contribution. **4)** Normalize to range [0, 100] %. **5)** The larger the score, the more important overall.

V. Boosting (not parallel, different from bagging)

1) Set $f(x) = 0$ and $\eta_i = y_i$ for all i in the training set. **2)** For $b = 1, 2, \dots, B$, repeat: **2.1)** Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r) . **2.2)** Update $\hat{f}$ by adding in a shrunken version of the new tree $\hat{f}(x) \leftarrow \hat{f}(x) + \eta \hat{f}^b(x)$. **2.3)** Update the residuals $\eta_i \leftarrow r_i - \eta \hat{f}^b(x_i)$. **3)** Output model $\hat{f}(x) = \eta \sum_{b=1}^B \hat{f}^b(x)$.

Note: The algorithm above is for regression tree η_i can be MSE. Classification tree is similar but use a pseudo-residual $\eta_i = \frac{\partial}{\partial \xi_i} \ell(\xi_i, y_i)$ for gradient boosting $\hat{f}(x) \leftarrow \hat{f}(x) +$

$\eta \sum_{b=1}^B \gamma_b I(x \in R_b)$, where $\gamma_b = \text{argmin}_{\gamma} \sum_{i: x_i \in R_b} \ell(\hat{f}(x) + \gamma, y_i)$.

VI. Advantages and Disadvantages of Tree

Advantage: Inexpensive to construct, variable selection, implicit accounts for interactions, easy to interpret when small, can handle missing data.

Disadvantage: Lack of global optimality, instability (error at top can make next wrong), difficulty in modeling additive structures.

12. Neural Network (Flexible)

I. Function Family

Function families in neural networks are high-dimensional: many parameters, many inputs, and many outputs.

Example: $h(x_1, x_2, x_3, x_4; \beta_1, \beta_2) = (x_1 + \beta_1, x_1 \cdot x_2, \frac{x_1}{\beta_2}, x_3 \cdot x_4, x_1 \cdot x_4)$ is a family function has **5 outputs and 4 inputs** x_i , where $i = 1, \dots, 4$; β_j are fixed values defined by ourselves. If we set $\beta_1 = 4, \beta_2 = 0.5$, then $f(5, 4, 3, 2) = (7, 20, 10, 6, 10)$.

An example of 2-d polynomial family: $f(x; \beta_1, \beta_2) = \text{sqrt}(x - \beta_1) \times \beta_2$.

II. Gradient Descent Detail (negative log likelihood - cross entropy)

The negative log likelihood is defined as $NLL = -\sum_{i=1}^n \log P(Y_i|X_i; \beta)$. We want to solve the objective function $\text{argmin}_{\beta} NLL$ but there is no closed form solution. As a result, we use gradient descent. **Note: NLL is used for classification.**

The algorithm works as follows: **1)** Initialize β randomly. **2)** Pick learning rate η . **3)**

Calculate the gradient ∇NLL . **4)** Update $\beta \leftarrow \beta - \eta \nabla NLL$.

Example: Assume $\eta = 1/3, \beta = 5, L(\beta) = (\beta - 2)^2, L'(\beta) = 2(\beta - 2)$.

Round 1: $L'(\beta) = 6 \Rightarrow \beta \leftarrow \beta - \eta L'(\beta) = 5 - \frac{6}{3} = 5 - 2 = 3$

Round 2: $L'(\beta) = 2 \Rightarrow \beta \leftarrow \beta - \eta L'(\beta) = 3 - \frac{2}{3} = 2.33$

Gradient Descent with Momentum

Algorithm: **1)** Initialize β randomly. **2)** Pick learning rate $\eta > 0$ and momentum $\alpha \in [0, 1]$.

Set velocity $V = 0$. **3)** Calculate the gradient ∇NLL . **4)** Update velocity $V \leftarrow \alpha V + \nabla NLL$ **4)**

Update $\beta \leftarrow \beta - \eta V$.

Example: Assume $\eta = 2, \alpha = 0.5, V = 0, \beta = 10, L'(\beta) = 1$.

Round 1: $L(\beta) = 10, L'(\beta) = 1 \Rightarrow V \leftarrow \alpha V + L'(\beta) = 1 \Rightarrow \beta \leftarrow \beta - \eta V = 8$

Round 2: $L(\beta) = 8, L'(\beta) = 1 \Rightarrow V \leftarrow \alpha V + L'(\beta) = 1.5 \Rightarrow \beta \leftarrow \beta - \eta V = 5$

Round 3: $L(\beta) = 5, L'(\beta) = 1 \Rightarrow V \leftarrow \alpha V + L'(\beta) = 1.75 \Rightarrow \beta \leftarrow \beta - \eta V = 1.5$

III. Minibatch Stochastic Gradient Descent (faster than gradient descent)

Idea: In each iteration, we firstly randomly sample a minibatch B consisting of a fixed number of training examples. As the minibatch approach never sees the whole training dataset at once, it is **less liable to overfit** to the whole training dataset.

The objective function is $\beta \leftarrow \beta - \frac{\eta}{|B|} \nabla NLL$, which is similar to gradient descent.

Note: The set B is random and changes from iteration to iteration. Since we pick a minibatch B , we need to normalize by its size $|B|$. η and $|B|$ (the batch size) are hyperparameters, meaning that they are kept fixed during training.

IV. Sequential Networks

A sequential network architecture defines a function family $\{f(x; \beta)\}_{\beta}$ by composing a sequence of simpler function families. $f(x; \beta) = g_3(g_2(g_1(x; \beta); \beta_2); \beta_3)$, each little function $\{g_j(z; \beta_j)\}_{\beta_j}$ is called a layer, each has inputs, parameters, outputs.

V. Layers

1. Dense Layer (Affine Layer): $g_{\text{aff}}(z; w, b) = wz + b$. Input $z \in R^{n_{\text{in}}}$. Output $g(z; w, b) \in R^{n_{\text{out}}}$. Parameters $w \in R^{n_{\text{out}} \times n_{\text{in}}}$ and $b \in R^{n_{\text{out}}}$.

2. ReLU (activation) Layer: $(\text{ReLU}(z))_i = \max(x, z_i)$. Input $z \in R^{n_{\text{in}}}$. Output $\text{ReLU}(z) \in R^{n_{\text{in}}}$. There is no parameter for this layer.

3. Other Activation Layers: $(\text{sigmoid}(z))_i = \frac{1}{1 + \exp(-z_i)}$. $(\text{swish}(z))_i = \frac{z_i}{1 + \exp(-z_i)}$. $(\text{softplus}(z))_i = \log(1 + e^{z_i})$.

4. 1D-CNN Layer: $(g_{\text{conv}}(z; w, b))_{i,j} = b_j + \sum_{k,s} w_{j,k,s} z_{i+s,j+k}$. Input $z \in R^{n_{\text{day}} \times n_{\text{inchannel}}}$.

Output $g(z; w, b) \in R^{(n_{\text{day}} - n_{\text{filterwidth}} + 1) \times n_{\text{outchannel}}}$. Parameters $b \in R^{n_{\text{outchannel}}}$, $w \in R^{n_{\text{outchannel}} \times n_{\text{inchannel}} \times n_{\text{filterwidth}}}$.

5. 2D-CNN Layer: $(g_{\text{conv}}(z; w, b))_{i,j,k} = b_k + \sum_{\ell, r, s} w_{k,\ell,rs} z_{i+s,j+r,\ell}$. Input $z \in R^{n_{\text{width}} \times n_{\text{height}} \times n_{\text{inchannel}}}$. Output $g(z; w, b) \in R^{(n_{\text{width}} - n_{\text{filterwidth}} + 1) \times (n_{\text{height}} - n_{\text{filterheight}} + 1) \times n_{\text{outchannel}}}$. Parameters $b \in R^{n_{\text{outchannel}}}$, $w \in R^{n_{\text{outchannel}} \times n_{\text{inchannel}} \times n_{\text{filterwidth}} \times n_{\text{filterheight}}}$.

6. Max Pooling Layer: $(g_{\text{conv}}(z; w, b))_{i,j} = \max_{s \in \{1, 2, n_{\text{filterwidth}}\}} z_{i+s,j}$. Input $z \in R^{n_{\text{day}} \times n_{\text{inchannel}}}$. Output $g(z; w, b) \in R^{(n_{\text{day}} - n_{\text{filterwidth}} + 1) \times n_{\text{inchannel}}}$. No parameters.

7. Downsampling: Drop half the data after performing CNN and max pooling. Drop $(k - 1)/k$ of the output by only taking one out of every k outputs. Stride convolution.

13. Unsupervised Learning

I. Partitional Clustering

K-means alternates between assigning centroids to points and recalculating centroid $\mu_y = \sum_{i \in C_y} X_i / |C_y|$, where $|C_y|$ is the sample count. It attempts to minimize Within-cluster Sum of Squares $WSS = \sum_{y=1}^K \sum_{i \in C_y} \sum_{j=1}^p (X_{ij} - \mu_{yj})^2$. Between-cluster Sum of Squares $BSS = \sum_{y=1}^K |C_y| \sum_{j=1}^p (\mu_{yj} - \bar{\mu}_j)^2$. **Note: TSS = WSS + BSS.**

Algorithm: **1)** Select K points as the initial centroids. **2)** Repeat **2.1)** form K clusters by assigning all points to the closest centroids **2.2)** recompute the centroid of each cluster until the centroids do not change anymore.

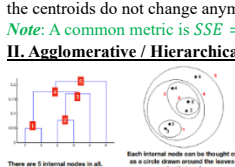
Note: A common metric is $SSE = \sum_{k=1}^K \sum_{x \in C_k} \text{dist}^2(m_k, x)$.

II. Agglomerative / Hierarchical Clustering

Instead of minimizing WSS, it makes **dendrograms**. **1)** Create root node for each data point. **2)** Merge roots until there is only one root left, achieved by creating a new root with two of the existing roots as children. Anatomy of a dendrogram (nodes, children, leaves under a node, height). We can cut a dendrogram at a given height.

1. Minimum Linkage (Single Link): Find the two points that have the **minimum distance**.

The **most similar** points in the different clusters. **Good for non-elliptical shapes.**



2. Maximum Linkage (Complete Link): Find the two points that have the **maximum distance**. The least similar points in the different clusters. **Even sized clusters, single leaves are fused one by one.**

3. Group Average: Find the **average of pairwise distance** (average dissimilarity) between points in the clusters. **Even sized clusters.**

III. Silhouette Plots

Defined by $s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$. It is always between -1 and 1. Big when points are far away from other clusters. Small when points are close to their own cluster.

14. Other Important Stuff for the Exam

I. Methods in Regard to Variance

Lower variance: Moreo ridge or lasso, feature deletion, data augmentation, early stopping, larger slack for SVM (when slack is large, margin also large, less variance).

Higher variance: Deep decision trees, polynomial features.

II. Decision Tree Example

Given $(X_1 = 2, Y = 20)$, $(X_2 = 3, Y = 23)$, $(X_3 = 4, Y = 4)$, and $(X_4 = 10, Y = 3)$ as a decision tree. Suppose we have $X = 11$ and we want to predict the result Y .

At depth 1, the decision tree will split the data based on a single feature to create two regions. Region 1 has $(X_1 = 2, Y = 20)$ and $(X_2 = 3, Y = 23)$, $\bar{Y} = 21.5$. Region 2 has $(X_3 = 4, Y = 4)$ and $(X_4 = 10, Y = 3)$, $\bar{Y} = 3.5$. If $X \leq 3$, region 1. Otherwise, region 2. As here $X = 11$, the prediction is $Y = 3.5$.

At depth 2, we continue split based on depth 1. We keep region 1 as it is, then split region 2.1 as $(X_3 = 4, Y = 4)$ and region 2.2 as $(X_4 = 10, Y = 3)$. If $X \leq 4$, region 2.1. Otherwise, region 2.2. As here $X = 11$, the prediction is $Y = 3$.

III. SVM Example

Suppose we have an SVM, decision boundary is $\{15\}$, orientation is such that points more than 15 are classified as “+1”, and points less than 15 considered as “-1”, margin $M = 3$. We now calculate the slack

for the following points: On the boundary: $(X, Y) = (15, +) \Rightarrow \xi = 1$, $(X, Y) = (15, -) \Rightarrow \xi = 1$; Correct side: $(X, Y) = (18, -) \Rightarrow \xi = \left(-\frac{3}{M}\right) + 1 = 0$; Wrong side $(X, Y) =$

$(18, +) \Rightarrow \xi = \left(\frac{3}{M}\right) + 1 = 2$.

IV. CPU vs GPU

CPU is faster when doing usual math: $f(x) = \cos(\cos(\cos(\cos(x))))$.

GPU is faster when having parallelable task: $g(x) = \cos(x) + \cos(2x) + \dots + \cos(10x)$.

V. Autodifferentiation Example

$f(x; \beta_1, \beta_2) = g_2(g_1(x; \beta_1); \beta_2)$. Calculate the derivative of f with respect to β_1 using: **1)** derivative of g_1 is $h_1(x; \beta_1)$; **2)** derivative of g_2 is $h_2(z; \beta_2)$; **3)** derivative of g_1 w.r.t. β_1 is $r_1(x; \beta_1)$; **4)** derivative of g_2 w.r.t. β_2 is $r_2(z; \beta_2)$. The answer is $h_2(z; \beta_2) \times r_1(x; \beta_1)$.

VI. Neural Network Estimation Examples

Example 1: Consider an affine layer with one input and one output, $g(x, w, b) = wx + b$. Consider the sequential network with two layers, one input, and one unit for each layer, defined programmatically by: **INPUT:** $x \rightarrow z = g(x, 2, -100) \rightarrow z = \text{ReLU}(z) \rightarrow z = g(z, 4, 5) \rightarrow \text{OUTPUT: } z$. What is $f(3)$? **INPUT:** $3 \rightarrow z = g(3, 2, -100) = -94 \rightarrow z = \text{ReLU}(-94) = 0 \rightarrow z = g(0, 4, 5) = 5 \rightarrow \text{OUTPUT: } 5$.

Example 2: Let $w = (1, 1, -1)$ be a filter with width 3, one input channel and one output channel. Let $x = (10, 10, 5, 4, 0)$, what is $g_{\text{conv}}(x; w, 0)$?

1st filter: $10 \cdot 1 + 10 \cdot 1 + (-1) \cdot 5 = 15$ **2nd filter:** $10 \cdot 1 + 5 \cdot 1 + (-1) \cdot 4 = 11$

3rd filter: $5 \cdot 1 + 4 \cdot 1 + (-1) \cdot 0 = 9$ **Output: (15, 11, 9)**

Example 3: Let $x = (5, 6, 5, 7, 9, 10)$. Note that x has 7 numbers in it. Let y be the result of padding x with a zero on the left and a zero on the right and downsampling by a factor of 3. How many numbers are in y ? **Answer:** padding makes x has 9 numbers, after dividing by 3 (downsampling), we have 3 left.

Example 4: Consider a sequential neural network architecture with $p = 4$ input features comprising an affine layer with 2 units, another affine layer with 3 units, and a final output layer with 1 unit. There are $(4 + 1) \times 2 + (2 + 1) \times 3 + (3 + 1) \times 1 = 10 + 9 + 4 = 23$ parameters in total.

Example 5: How many parameters for a one-dimensional convolutional layer with 2 input channels, 3 output channels, and filter width 6? **Answer:** $2 \cdot 3 \cdot 6 = 36$ (39 with bias).

VII. PCA Autoencoder Example

Consider the encoder $g(x_1, x_2, x_3) = (x_1, x_2 - x_3)$ and the decoder $f(z_1, z_2) = (z_1, z_2, z_2)$. What is the squared reconstruction loss for the point $x = (1, 2, 3)$?

$g(1, 2, 3) = (1, -1)$; $f(1, -1) = (1, -1, -1)$; $\|x - f(1, -1)\|^2 = (1 - 1)^2 + (2 + 1)^2 + (3 + 1)^2 = 0 + 9 + 16 = 25$.

VIII. Statements

- **TRUE:** ridge and lasso penalties can be used for densely connected relu-activated sequential network architectures.

- **TRUE:** In an autoencoder, the number of outputs for the decoder is the same as the number of inputs for the encoder. The reverse is also true.

- **FALSE:** The goal of image generation is to learn a regression function: we must estimate the expected image, conditioned upon a particular caption.

- **TRUE:** Large values of the span parameter lead to prediction rules that depend on a large fraction of the training points. (local regression)

- **TRUE:** Suppose there are n predictors, PCA can choose any number of predictors from $\{1, \dots, n-1\}$.