Chapter 1: Introduction

Dimensionality of Data:
- Low-dimensional data/models: $p \ll n$
- High-dimensional data/models: $p \gg n$

Curse of Dimensionality:
- When the dimensions is too large, p is too large, there are very few points that we can use to predict in each dimension. Points tend to be further apart
- Local methods does not work well in high dimensional data ($p = \sim 10$ it will perform okay)

Parametric Methods:
- f is dependant on a fixed number of unknown scalar parameters
- **Advantages**: Simple and interpretable
- **Disadvantage**: Might not contain the true model

Non parametric methods:
- No explicit assumptions on the function form of f.
- **Advantage**: Likely to contain the true model
- **Disadvantage**: Prone to overfitting because the model is too flexible

Overfitting:
- When the model is too flexible and it tries to fit all the training data. This causes the model to not be able to capture variation in testing data.

Prediction Accuracy Measurements:

ϵ is the irreducible error
- Sources of the irreducible error:
  - Unmeasured variables
  - Unmeasurable variation

Mean Squared Error (MSE):
- For a prediction of $\hat{Y}$ on Y
- Measure to maximise accuracy/ minimise error

$$MSE(\hat{Y}) = E(Y - \hat{Y})^2$$
$$= E\left(f(x) - \hat{f}(x)\right)^2 + Var(\epsilon)$$

$E\left(f(x) - \hat{f}(x)\right)^2$ – Reducible Error.
$Var(\epsilon)$ – Irreducible Error

Training MSE:
- Used to estimate the theoretical MSE and try to reduce it
- Measures how well $\hat{f}$ fits our training data
- We can think of this as the average difference between the predicted points and the actual points for the training data.
- Tends to underestimate the theoretical MSE as it tries to overfit the training data and doesn't account for deviations
- **Lower is better**

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

Where $\hat{f}$ is estimated from the training data $(x_1, y_1), \cdots, (x_n, y_n)$

Test MSE:
- Used to estimate the theoretical MSE and try to reduce it
- Measures how well $\hat{f}$ fits our testing data (unseen data) which is what we desire
- We can think of this as the average difference between the predicted points and the actual points for the training data.
- **Lower is better**

$$Test\ MSE = \frac{1}{m}\sum_{i=1}^{m}\{\tilde{y}_i - \hat{f}(\tilde{x}_i)\}^2$$

Where $\hat{f}$ is estimated from the testing data $(\tilde{x}_1, \tilde{y}_1), \cdots, (\tilde{x}_m, \tilde{y}_m)$

Bias-Variance Trade-off

Consider the Expected Test MSE (Average test MSE that we would get if we repeatedly estimated f using i.i.d. training datasets)

$$E\{y_0 - \hat{f}(x_0)\}^2 = Var\left(\hat{f}(x_0)\right) + \left\{Bias\left(\hat{f}(x_0)\right)\right\}^2 + Var(\epsilon)$$

- Our goal would be to reduce both bias and variance so that the overall expected test MSE is lowered

Bias: $Bias\left(\hat{f}(x_0)\right) = E\{\hat{f}(x_0)\} - f(x_0)$
- When a model is more flexible, there will be lower bias since it is more possible for the model to contain the true model. Converse will then be true for a model that is less flexible

Variance: $Var\left(\hat{f}(x_0)\right)$
- When a model is more flexible, there will be higher variance because when the model is more flexible, it will tend to overfit. With new data coming in, it will cause greater variation in the data.

Choice of Model:
- Choose a model that provides a good balance between bias and variance so that the test MSE is lowered and our ability to predict unseen data is better (more accurate).

Classification
- When the response Y is qualitative

Assessing the classifier to classify the class labels:

Training Error Rate:
- Just check how many of the values are incorrectly labelled using an indicator function. We will then take the average to see on average how many data points are labelled correctly
- Works on the training data
- Note that this training error is not a good estimator of the theoretical error rate
- **Lower is better**

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

Where $\hat{y}_i$ is the predicted class label, and I is the indicator function

Test Error Rate:
- Checks how many of the unseen data we manage to predict correctly
- Bias-variance trade-off works in classification problems as well, therefore we will want to check if there is overfitting
- **Lower is better**

$$\frac{1}{m}\sum_{i=1}^{m} I(\tilde{y}_i \neq \hat{y}_i)$$

Computed on independent test data $(\tilde{x}_1, \tilde{y}_1), \cdots, (\tilde{x}_m, \tilde{y}_m)$

Clustering:

- Unsupervised learning where we try to find subgroups according to similarities among observations

Let $C_1, \cdots, C_k$ denote sets containing the indices of the observations in each cluster
- Clusters should not overlap, therefore the groups are disjoint and should cover all points

$$C_1 \cup \cdots \cup C_K = \{1, \cdots, n\}$$
$$C_j \cap C_k = \emptyset \ \ for\ all\ j \neq k$$

K-means clustering:
- Tries to minimise the within-cluster variation which is to minimise the distance between points in the same cluster
- Goal is to find the clusters that reduces the distance between points in the same cluster.

Suppose that $W(C_k)$ is the within-cluster variation in the cluster C_K. We will want to solve the following problem

$$\underset{C_1,\ldots,C_K}{\arg\min} \sum_{k=1}^{K} W(C_k)$$
$$\underset{C_1,\ldots,C_K}{\arg\min} \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \epsilon C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

We can define $W(C_k)$ to be the following which is like the average distance between all points that we have in the cluster and normalised by the number of observations in the cluster

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \epsilon C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

Chapter 2: Simple Linear Regression

Subset Selection:

Reasons for doing it

1. **Prediction accuracy**

- The full model containing all predictors has low bias, but high variance. Because when we have more predictors, the model is more flexible and we can fit the training data better. (Under Bias-Variance Trade-off)
- When $p > n$, (Parameters more than number of observations), there is no longer a unique least squares coefficient estimate

2. **Interpretability**

- In high-dimensional datasets, some of the variables may not be associated with the response, therefore removing the irrelevant variables helps us to understand the relationship better

Best Subset Selection:

Goal: Among p variables, we want to find $k \leq p$ variables that provide the best fit to our data

Variable Selection Measure for fit: Smallest RSS

- The one with the smallest RSS is the one that we want

Model Selection: CV-Prediction Error, C_p (AIC), BIC, or adjusted R^2

Algorithm: Fit all of the $\binom{p}{k}$ models containing exactly k variables, select the one with the smallest RSS. Total of 2^p models so it is actually very costly

Algorithm *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Note that for each k there is a best model with the lowest RSS but to do model selection, we cannot make use of RSS.

- By using RSS to select k, we will always get $k = p$ since if we increase k, it gives us more flexibility and a better fit for the whole data. Therefore, it will just reduce the training RSS which is not good since we are not accounting for the variability that comes with a model that is more flexible

---

Forward Stepwise Selection:

- Start with the simplest model which contains only the intercept
- Add into the model one predictor at a time that best improves the fit each time
  - The model that reduces the RSS the most so after we add in the predictor, the RSS of the model should be the smallest

Variable Selection Measure for fit: Smallest RSS

- The one with the smallest RSS is the one that we want

Model Selection: CV-Prediction Error, C_p (AIC), BIC, or adjusted R^2

Algorithm:

Algorithm *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Analysis:

- Start with centered predictors and the model with only intercept. Since there is only the intercept, all predictors will just be the mean
  - $\hat{y}_i = \bar{y},\ for\ i = 1, \cdots, n$
- At each step, we compute the correlation between each predictor and the residuals

$$\hat{c}_j = x_j^T (y - \hat{y})$$

  - How large the correlation tells us how strong a predictor is associated with the residuals
  - If the association is strong, adding the predictor to the model will reduce the RSS
  - Choosing the variable with the smallest RSS is the same as choosing the variable that is the most strongly associated with RSS because it can help us to explain the ϵ the most and including it will cause the RSS to reduce the most
- For all the predictors, we update $\hat{y}$ by:

$$\hat{j} = \arg\max_j |\hat{c}_j| \ \ and\ \ \hat{y} \leftarrow \hat{y} + \hat{c}_j \cdot x_j$$

Where x_j is the jth column of the design matrix

Backward Stepwise Selection

- Start with the full model which contains all predictors (Opposite from Forward Stepwise Selection)
- Delete from the model, the predictor that has the least impact on the fit
  - After deleting the predictor, the RSS of the model should still be the lowest. Because as compared to other models, this predictor does not affect the fit as much
- Only need to fit a total of $1 + \frac{p(p+1)}{2}$ models

Variable Selection Measure for fit: Smallest RSS

- The one with the smallest RSS is the one that we want

Model Selection: CV-Prediction Error, C_p (AIC), BIC, or adjusted R^2

Algorithm *Backward stepwise selection*

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Forward Stagewise Regression

- Similar to that of forward stepwise regression just that we are only adding amounts of the variable that is necessary

Goal: Only add a necessary amount of the predictor into the model

Variable Selection Measure for fit: Smallest RSS

- The one with the smallest RSS is the one that we want

Model Selection: CV-Prediction Error, C_p (AIC), BIC, or adjusted R^2

Algorithm: Similar to Forward Stepwise Regression just that when we add in a variable, we add a portion of it only

- Start with centered predictors and the model with only intercept. Since there is only the intercept, all predictors will just be the mean
  - $\hat{y}_i = \bar{y},\ for\ i = 1, \cdots, n$
- At each step, we compute the correlation between each predictor and the residuals

$$\hat{c}_j = x_j^T (y - \hat{y})$$

  - How large the correlation tells us how strong a predictor is associated with the residuals
  - If the association is strong, adding the predictor to the model will reduce the RSS
  - Choosing the variable with the smallest RSS is the same as choosing the variable that is the most strongly

associated with RSS because it can help us to explain the ϵ the most and including it will cause the RSS to reduce the most

- For all the predictors, we update $\hat{y}$ by:

$$\hat{j} = \arg\max_j |\hat{c}_j| \quad and \quad \hat{y} \leftarrow \hat{y} + \delta \cdot sign(\hat{c}_j) \cdot x_j$$

Where x_j is the jth column of the design matrix

Note for Forward Stagewise Regression

- Note that if we set $\delta = |\hat{c}_j|$ then it will lead to forward stepwise selection
- If δ is small, we need many steps (much more than p) to reach the full model
- The same variable can be selected in different steps. If a variable is re-picked in a step, then we do not have a "new" variable added but we still add a portion of the stated predictor inside.
- Only we add in everything then it will be done.
- Compared to forward stepwise selection, a variable can only be selected once and we will always have a new variable added at each step.

Model Selection Criteria:
R^2 and RSS:

$$R^2 = 1 - \frac{RSS}{TSS}$$

- **Usage**: Can be used when the models have the same number of predictors
- Cannot be used when the models have different number of predictors because it will prefer more flexible models. More flexible models → fit data better → reduces the RSS and therefore it does not take into account the variability of larger models as well
- R^2 always increases after adding a variable and RSS always decrease

---

Adjusted R^2

- Idea is to reduce the decrease in RSS caused by an increase in the number of predictors so that there is still a balance
- **Choice of model**: The one with the highest value

$$Adjusted\ R^2 = 1 - \frac{\frac{RSS}{n-k-1}}{\frac{TSS}{n-1}}$$

- Note that we normalise RSS with $(n-k-1)$ and TSS with $(n-1)$
- With the normalising term → When k increases, RSS decreases but n-k-1 is smaller and therefore, it cancels out the effect of the decrease in RSS by an increase in the number of predictors
- **Contrast**: Adjusted R^2 may decrease if the reduction in RSS is relatively small after adding a variable. Therefore, if we add in too many predictors, adjusted R^2 may decrease instead

---

Mallow's C_p

- Idea is also to counter the effect of the increase in predictors with another term that tries to counter the decrease in RSS
- **Choice of model**: The one with the lowest value

$$C_p = \frac{1}{n}(RSS + 2k\hat{\sigma}^2) = \frac{RSS}{n} + \frac{2k\hat{\sigma}^2}{n}$$

Competing terms:

- Lack-of-fit term: $\frac{RSS}{n}$, decreases with k
- Model complexity term: $\frac{2k\hat{\sigma}^2}{n}$, increases with k
  - Helps to penalise larger models

AIC for linear models

- Identical to Mallow's C_p for linear models. However, for other models, they are different so do note
- Same idea of penalising larger models
- **Choice of model**: The one with the lowest value

$$AIC = \frac{1}{n}(RSS + 2k\hat{\sigma}^2)$$

BIC for linear models

- This also penalises the large values of k where it decreases with RSS and helps to negate the effect of k on RSS.

$$BIC = \frac{1}{n}(RSS + \log(n)\, k\hat{\sigma}^2)$$

Note:

- BIC places a heavier penalty than AIC when $n \geq 7$
  - $\log n > 2$ when $n \geq 7$
- BIC normally selects a **smaller model** as compared to AIC due to the additional penalty term

---

Model Selection via Validation:

Jargons:
Training Data: Data used for estimating model parameters
Testing Data: Data used for assessing model prediction accuracy
Validation Data: Hold-out data for selecting models
Training MSE:

$$Training\ MSE = \frac{Training\ RSS}{n}$$

Test MSE:

$$Test\ MSE = \frac{Test\ RSS}{sample\ size\ of\ test\ data}$$

Validation MSE:

$$Validation\ MSE = \frac{Validation\ RSS}{sample\ size\ of\ validation\ data}$$

- Idea is to hold out a portion of the dataset so that we can use it to check on the validity of our dataset

Advantage: Easy to implement
Disadvantage: Lesser data for training since we are holding out a portion of the data that we could have potentially used for training. Therefore, it overestimates the test MSE.

K-Fold CV:

- Choose $K \leq n$ which selects the number of folds that we want to have (disjoint groups that we want our data into)
- Randomly divide our data into the disjoint groups of roughly equal size
- When K is small, there is larger estimation bias but smaller variance. Because we have lesser data during each training and therefore, there will be more bias but lesser variance since it will not overfit as much.

For each $r = 1, \cdots, K$

- Hold out the rth fold, merge the rest of the other K-1 folds to form a training dataset set $\mathcal{D}_{-r}$
- Estimate the model parameters on the training dataset D_{-r} for each model
- Calculate the validation MSE MSE_r of each model using the rth fold of the data that was held out

CV MSE:

- Idea is that we want to find the model with the smallest average mean squared error over the validation datasets. Each MSE_r is the validation MSE on a hold out set and we are computing the average MSE

$$CV = \frac{1}{K}\sum_{r=1}^{K} MSE_r$$

Criteria: Select the model with the smallest CV MSE

Leave-one-out-cross-validation (LOOCV): Special case of K-fold CV with $K = n$

- Because we are using more training data per iteration, it leads to a better fit of the training dataset. Therefore, it gives an unbiased estimate of the test MSE since it is more flexible but it has larger variance due to the model flexibility

Logistic Regression

Logistic Function:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

Always between 0 and 1

Can be used to model probability because it is between 0 and 1. However for a Linear Model, the values will be between $-\infty$ and ∞ and therefore, is not good to model such probabilities

Odds:

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$$

Any value between 0 and ∞

Log Odds / Logit

$$\log\frac{p(X)}{1-p(X)} = \beta_0 + \beta_1 X$$

Any value between $-\infty$ and ∞

- **Interpretation**: Increasing X by 1 unit changes the log odds by β_1
- Note that if we have β_p in multiple logistic regression: **When other covariates are fixed**, increasing X_p by 1 unit changes the log odds by β_p.

Logistic Classifier:

We need to first have the Estimated Probability of the predictor variables

$$\hat{p}(X_i) = P(Y_i = 1 | X_{i1}, \cdots, X_{ip}) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X_{i1}+\cdots+\hat{\beta}_p X_{ip}}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X_{i1}+\cdots+\hat{\beta}_p X_{ip}}}$$

We will then use the estimated probability to classify the point X_i.

- For a threshold value τ, the logistic classifier

$$C(X_i) = \begin{cases} 1, & if\ \hat{p}(X_i) > \tau \\ 0, & otherwise \end{cases}$$

Classification based on Linear Component:

For any $\tau > 0$, we can classify $X = x$ to $Y = 1$ if:

$$\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p > \log\frac{\tau}{1-\tau}$$

- This is because P(X) >τ is equivalent to $\log\frac{p(X)}{1-p(X)} \geq \log\frac{\tau}{1-\tau}$ and we know that $\log\frac{p(X)}{1-p(X)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$

Note that if $\tau = 0.5$, we can classify $X = x$ to $Y = 1$ if:

$$\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p > 0$$

MLE in Logistic Regression:

MLE in Logistic Regression

- Recall the logistic regression model with a predictor

$$p(X_i) = \frac{e^{\beta_0+\beta_1 X_i}}{1+e^{\beta_0+\beta_1 X_i}}$$

- $Y_i \sim \text{Bernoulli}(p(X_i))$ and has the probability mass function

$$f(Y_i; X_i, \beta_0, \beta_1) = \begin{cases} p(X_i) & \text{if } Y_i = 1 \\ 1-p(X_i) & \text{if } Y_i = 0 \end{cases}$$

we can use multinomial to model multiple classes

- Equivalently,

$$f(Y_i; X_i, \beta_0, \beta_1) = p(X_i)^{Y_i}\{1-p(X_i)\}^{(1-Y_i)}$$

simpler formula way

equivalent way of writing Bernoulli

this will give us either case if $Y_i=1$ or $Y_i=0$

Linear Discriminant Analysis:

Discriminant Function:

$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

LDA Classifier: We will classify $X = x$ to the kth class if k maximises the value of $\delta_k(x)$ given the value of x

Note that there is an assumption that each $f_k(x)$ is Gaussian

- Assume each $f_k(x)$ to be Gaussian, i.e.,

we need to estimate pdfs of $f_k(x)$

Assuming single predictor and is continuous & hence follows Gaussian

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k}\exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

- σ_k and μ_k, for $k = 1, \ldots, K$, are unknown parameters
- Assumption: $\sigma_1 = \cdots = \sigma_K$; denote this common variance by σ

Decision Boundary:

Bayes Classifier: Classify an observation x to the class for which $p_k(x)$ is largest, it has the lowest possible error rate out of all classifiers

Decision Boundary: $\delta_1(x) = \delta_2(x) = \cdots = \delta_n(x)$

Empirical LDA Classifier:

If we have data (X_i, Y_i) for $i = 1, \cdots, n$. Where Y_i are the class labels

Mean of group k, μ_k : can be estimated by taking the average X values for those with the label k

$$\hat{\mu}_k = \frac{1}{n_k}\sum_{i:Y_i=k} X_i$$

Common σ^2: can be estimated by taking the average variance across the various classes

$$\hat{\sigma}^2 = \frac{1}{n-K}\sum_{i=1}^{K}\sum_{i:Y_i=k}(X_i - \hat{\mu}_k)^2$$

- Note that we divide by $n-K$ because of the degrees of freedom

Prior probability $\pi_k = \Pr(Y = k)$: can be estimated by just taking the proportion of those in group k.

$$\hat{\pi}_k = \frac{n_k}{n}$$

Linear Discriminant Function: Just need to make use of the previous estimated values. We can make use of this as an LDA classifier whereby for any input x, we will classify it to the kth class if k maximises the value of δ_k

$$\hat{\delta}_k(x) = x\frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

Estimating $\hat{p}(x)$ and classifying x by $\hat{p}(x)$ instead:

Answer. Recall the posterior probability

$$p_k(\boldsymbol{x}) = \frac{\pi_k f_k(\boldsymbol{x})}{\sum_{j=1}^{K}\pi_j f_j(\boldsymbol{x})}$$

classifying $X = \boldsymbol{x}$ to the kth class if k maximizes $p_k(\boldsymbol{x})$.

In LDA, f_k is modeled by a Gaussian distribution and estimated by $\hat{f}$ based on $\hat{\mu}_k$ and $\hat{\sigma}$. In addition, π_k is estimated by $\hat{\pi}_k = n_k/n$, where n_k is the number of observations in Class k. Assumption that $\hat{\sigma}$ is the same across all classes — if it is not the same we will get a quadratic model instead

Therefore, we can obtain an estimate $\hat{p}_k(\boldsymbol{x})$ of $p_k(\boldsymbol{x})$, and classify $\boldsymbol{x}$ to Class k if k maximizes $\hat{p}_k(\boldsymbol{x})$.

LDA doesn't perform well in high dimensions:

Difficult to estimate the covariance matrix Σ when p is large relative to n. Also in high dimensions, approximating f_k by the multivariate Gaussian distribution may not be adequate. (May not be able to assume Gaussian)

Classification Quality

Sensitivity:

The percentage of actual positives that we correctly identify

- Take all the true positives divided by the total number of actual positives

$$\frac{TP}{TP + FN}$$

To increase the sensitivity: We can set a **lower threshold** for a value to be classified as a positive value. If we take it to the extreme, we will classify all points as **positive** since the threshold is so low that no points become negative and just becomes **positive**. This will cause the sensitivity to become 1 because we will be able to correctly identify all actual positives.

Type 2 Error:

False Negative Rate, the percentage of actual positives that we incorrectly identify as negatives.

The Null Hypothesis Doesn't hold, and we didn't reject it

$$\frac{FN}{TP + FN}$$

$$Type\ 2\ Error = \ 1 - Sensitivity$$

- Note that when we want to reduce the Type 2 error, we can try to increase the sensitivity because we will detect actual positives better and that reduces the Type 2 error.
- Look at this if incorrectly identifying positives as negatives is bad.

Specificity:

The percentage of actual negatives that we correctly identify

- Take all the true negatives divided by the total number of actual negatives

$$\frac{TN}{TN + FP}$$

To increase the specificity: We can set a **raise threshold** for a value to be classified as a positive value. If we take it to the extreme, we will classify all points as **negatives** since the threshold is so high that no points become positive and just becomes **negative**. This will cause the specificity to become 1 because we will be able to correctly identify all actual negatives.

Type 1 Error:

False Positive Rate, the percentage of actual negatives that we incorrectly identify as positives.

The Null Hypothesis Holds but we rejected it

$$\frac{FP}{TN + FP}$$

$$Type\ 1\ Error = \ 1 - Specificity$$

- Note that when we want to reduce the Type 1 error, we can try to increase the specificity because we will detect actual negatives better and that reduces the Type 1 error.
- Look at this if incorrectly identifying negatives as positives is bad.

Trade-off between Sensitivity and Specificity:

- Depends on the usage of the application. Whether Sensitivity or Specificity is more important.

  Threshold ↑ → Sensitivity ↓ & Sensitivity ↑

  Threshold ↓ → Sensitivity ↑ & Sensitivity ↓

When we want to increase Sensitivity: We lower the threshold causing more points to become positive. But it will cause more of the actual negatives to become positive also and raises False Positive (FP) which lowers specificity

When we want to increase Specificity: We raise the threshold causing more points to become negative. But it will cause more of the actual positives to become negative also and raises False Negative (FN) which lowers sensitivity

Note that the overall error rate does not always decrease as the threshold increase, there needs to be a balance between sensitivity and specificity

Decision Trees

Regression Trees:

Building the Tree: Suppose we have q non-overlapping regions we want to create
Find the boxes $R_1, \cdots, R_q$ that minimizes the RSS

$$\sum_{j=1}^{q} \sum_{i:x_i \in R_j} \left(Y_i - \bar{Y}_{R_j}\right)^2$$

Where

$$\bar{Y}_{R_j} = \frac{1}{n_j} \sum_{i:x_i \in R_j} Y_i$$

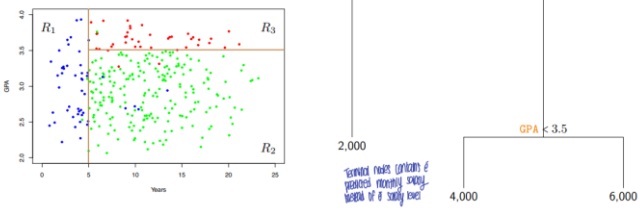
Essentially, find the boxes where the distance between each point and its mean is the smallest

- Note that the prediction if it falls into one region will be the mean of the points in that region.
- If $\tau = 1$, each terminal node contains only 1 observation, so the training RSS is 0

Predictor Space and Corresponding Tree:

- Suppose Salary is a continuous response, i.e., the monthly salary
- A regression tree: Predict

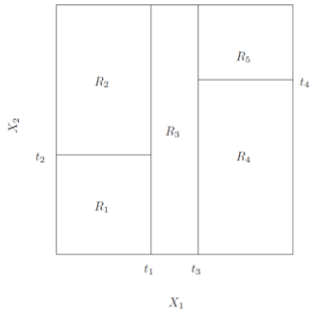
$$\text{Salary} = \begin{cases} 2,000 & \text{if Years} < 5 \\ 4,000 & \text{if Years} \geq 5 \text{ and GPA} < 3.5 \\ 6,000 & \text{otherwise} \end{cases}$$



- Note that if the condition is satisfied, it will go to the left, or else it will go to the right

Decision Boundaries:

- Note that since we are doing a recursive binary split, we should always have rectangles as the decision boundaries, we should not see any regions with overlapping rectangles.



Relation Between Number of Terminal Nodes and the Number of Regions:

- Number of terminal nodes of decision tree is equal to the number of regions it induces in the predictor space

Bias-Variance Tradeoff for Number of Regions:

- $q \uparrow \rightarrow$ Model Flexibility increases
- $q \uparrow \rightarrow$ Bias decreases
- $q \uparrow \rightarrow$ Variance Increases
- $q \uparrow \rightarrow$ Test MSE decreases first then increases to form a U-Shape

Choosing q via CV:

- Randomly divide the training data into K folds.
- For each fold, $k = 1, \ldots, K$, for each $q = 1, 2, \ldots$: Use the observations, except those in the kth fold,, to build a decision tree with exactly q terminal nodes, and then calculate its MSE $E_k(q)$ on the data from the kth fold.
- The CV error of q is $CV(q) = \sum_{k=1}^{K} E_k(q)$
- Choose $\hat{q}$ that minimizes $CV(q)$
- Use all training data to build a decision tree with $\hat{q}$ terminal nodes

Classification Trees:

Gini Index:

- Measures total variance across the K classes in the terminal node j
- **Smaller the value → Better** (This means that the node is pure and that it is primarily from a single class)
- Takes on small values if all of the $\hat{p}_{jk}$ are close to 0 or 1. Meaning that it is predominantly one of the values.

$$G_j = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

p_{jk} is the probability of having class k at the jth terminal node

$$G_j \in [0, 0.5]$$

Total Gini Index of a tree with q terminal nodes:

- Just need to sum up all the values

$$G = \sum_{j=1}^{q} G_j$$

Deviance (Cross-Entropy):

- Same idea as Gini Index just that we are using log function instead.

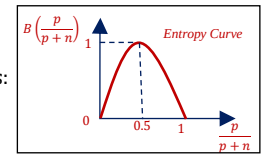
$$D_j = -\sum_{k=1}^{K} \hat{p}_{jk} \log \hat{p}_{jk}$$

Assume that $\hat{p}_{jk} \log \hat{p}_{jk} = 0$ if $\hat{p}_{jk} = 0$

$$D_j \in [0, 1]$$

Total Deviance of a tree with q terminal nodes:

$$D = \sum_{j=1}^{q} D_j$$



Constructing a Classification Decision Tree:

1. Calculate the root node **Gini Index / Deviance**. (i.e. try to see how pure the classification is)
2. For the remaining set of attributes $i = 1, \cdots, n$. Suppose that we split it on attribute i and we split on all the possible values that the attribute can take.
   a. Calculate the total Gini Index/Deviance across all the nodes after the split.
   b. Calculate the Information Gain which is taking the Initial Gini Index / Deviance - Total Gini Index / Deviance after the split.
   c. Choose the attribute that gives us the largest Information Gain. This shows that the purity after the split is higher.
3. Continue doing until a certain threshold or if the node is already pure or until we have no more attributes to split on. Note that we consider the node that we split on as the "root node" and we will calculate the information gain from there.

Prediction:

For each of the tree, just need to follow down the path and choose the label corresponding to the attributes that it has

Binary Classification:

- Idea can be further expanded into multiclass labels

Information Gain Formula:

$$Gain(C, A) = B\left(\frac{p}{p+n}\right) - H(C|A)$$

Entropy at the Root Node

$$H(C) = B\left(\frac{p}{p+n}\right) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right)$$

Entropy after the splitting on the attribute A

$$H(C|A) = \sum_{i=1}^{d} \frac{(p_i + n_i)}{p+n} B\left(\frac{p_i}{p_i + n_i}\right)$$

Entropy $H(C)$ of this node before any split (Prior Entropy)

Expected remaining entropy after testing A (Posterior Entropy)

Entropy Curve:

Maximum Entropy, $H(C) = 1$ (**Maximum Uncertainty**)
Minimum Entropy, $H(C) = 0$ (**Minimum Uncertainty**)

- 0 when $p/(p+n)$ is 0 or 1
- 1 when $p/(p+n)$ is 0.5
- Anything in between we will have some uncertainty

Entropy tells us **how equally the values are spread**, the closer to 1/2, the more uncertain we are since it means that the values are spread equally and we won't know which is the majority and is the correct label

Information Gain

Suppose we divide the training set E into different subsets which corresponds to each of the attribute values of A. Each subset E_i has $\boldsymbol{p_i}$ **+ve and** $\boldsymbol{n_i}$ **-ve** examples

$$H(C|A) = \sum_{i=1}^{d} \frac{p_i + n_i}{p + n} B\left(\frac{p_i}{p_i + n_i}\right)$$

Information Gain of target concept C from the attribute test on A

$$Gain(C, A) = B\left(\frac{p}{p+n}\right) - H(C|A)$$

Proportion that E_i takes out of all examples — Entropy of the subset E_i

Choose the attribute A with the **largest gain** (Largest Value)

Choice of Tree-Based Methods vs Linear Classifiers:

- **Linear Methods**: Better when the underlying relationship is linear.
- **Tree Based Methods**: Better if we have non-linear underlying relationship. It is not able to capture linear relationships properly

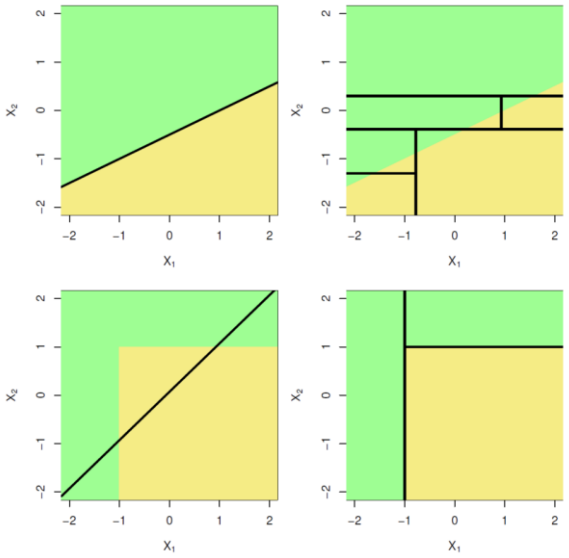


Figure 1 Left: Linear, Right: Tree. Top: Linear Relation, Bottom: Non-Linear Relation

Bias-Variance Tradeoff:

- **Tree size**: The number of terminal nodes (or leaves)
- Larger Tree, Larger Model Flexibility → Larger Variance and Smaller Bias
- If τ is small, i.e. we only allow no more than τ number of observations in each region, we will likely overfit the data since the model is very flexible.

Pruning:

- Helps to tackle the issue of overfitting
- Similar idea to that of Best Subset Selection

Strategy 1: Start with empty tree, grow the tree only so long as the decrease in RSS/Deviance/Gini index due to each split exceeds some (high) threshold.

- Note that this will result in shorter trees. Because there could be splits that may not seem necessary in the first place but turns out to have a larger decrease in RSS/Deviance/Gini Index at the end.

Strategy 2: Grow a full or very large tree T_0, with q terminal nodes, partition the predictor space into q disjoint regions. Add a penalty term to the Measure that we use.
Find a subtree of T_0 that minimizes the measures for either Regression/Classification Trees.
Note that we can select α using cross validation

Intuition: For each subtree $T \subset T_0$, we will add an additional penalty term to control the model size. $\alpha \geq 0$ helps us to add a penalty that penalizes having very large trees. But there is a competing term of RSS/Deviance/Gini Index at the first place as for larger trees, we have smaller RSS/Deviance/Gini Index but they are working in different directions. α can be used to control the Bias Variance Tradeoff

For Regression Tree:

Penalized RSS

$$\sum_{j=1}^{|T|} \sum_{i:x_i \in R_j} \left(Y_i - \bar{Y}_{R_j}\right)^2 + \alpha|T|$$

Note that $|T|$ is the size of the tree (i.e. the number of terminal nodes)

For Classification Tree:

Penalized Gini Index:

$$G = \sum_{j=1}^{|T|} G_j + \alpha|T|$$

G_j – is the gini index at the jth terminal node

Penalized Deviance:

$$D = \sum_{j=1}^{|T|} D_j + \alpha|T|$$

D_j – is the deviance at the jth terminal node

Algorithm:

GROW$(\mathscr{D}, \tau)$ Procedure: Construct a large tree based on data $\mathscr{D}$

- Use recursive binary splitting to grow a large tree T_0 on the data $\mathscr{D}$, stopping only when each terminal node contains no more than τ observations

PREDICT$(\boldsymbol{x}, T)$ Procedure: Given a tree T, predict the response of $\boldsymbol{x}$

- Locate the terminal node R_j such that $\boldsymbol{x} \in R_j$
- Output
  - the mean response of the observations in R_j
  - the majority class label in R_j

  as the predicted response for $\boldsymbol{x}$

PRUNE(T_0, α): Prune a tree T_0 with the hyperparameter α

- For the regression regression tree, find the subtree $T \subset T_0$ that minimizes the penalized RSS

$$\sum_{j=1}^{|T|} \sum_{i:x_i \in R_j} (Y_i - \bar{Y}_{R_j})^2 + \alpha|T|$$

- For the classification regression tree, find the subtree $T \subset T_0$ that minimizes the penalized Gini index or deviance

$$\sum_{j=1}^{|T|} G_j + \alpha|T| \quad \text{or} \quad \sum_{j=1}^{|T|} D_j + \alpha|T|$$

For each candidate value of α, for each $k = 1, \ldots, K$:

1. Hold the kth fold of data, and merge the other folds into a single training data $\mathscr{D}_{-k}$
2. Construct $T_0 = \text{GROW}(\mathscr{D}_{-k}, \alpha)$
3. Prune the tree to obtain $T = \text{PRUNE}(T_0, \alpha)$
4. For each observation (x_i, Y_i) in the kth fold of data, use the tree T from Step 3 to predict the response $\hat{Y}_i = T(\boldsymbol{x}_i) = \text{PREDICT}(\boldsymbol{x}_i, T)$
5. Calculate the cross-validation MSE

$$E_k(\alpha) = \sum_{(\boldsymbol{x}_i, Y_i) \text{ in the } k\text{th fold}} \{T(\boldsymbol{x}_i) - Y_i\}^2$$

or the cross-validation classification error

$$E_k(\alpha) = \sum_{(\boldsymbol{x}_i, Y_i) \text{ in the } k\text{th fold}} \mathbb{I}\{T(\boldsymbol{x}_i) \neq Y_i\}$$

Finally, the cross-validation error of α is $CV(\alpha) = \frac{1}{n} \sum_{k=1}^{K} E_k(\alpha)$

Bagging:

- Bootstrap Aggregation (Bagging): Generate B bootstrapped datasets via independent resampling with replacement, for each of the dataset we build a (bootstrap) decision tree T_b^*
- Given an observation x^*:

Pruning for Bagging:

- No point because Pruning reduces estimation variance but increases Bias. For Bagging, we already are reducing estimation variance and keep bias constant, we also want each bootstrap tree to have small bias.

Regression Tree: Predict Using

$$T_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b^*(x)$$

Classification Tree: Using Majority vote

$$T_{bag}(x) = most\ occurring\ label\ among\ T_1^*(x), \cdots, T_B^*(x)$$

- Note that if it is probability, we will need to look at what is the probability of it first for each tree. P(Red|X), we will predict it as Red if P(Red|X) ≥ 0.5 or else we will predict as the other label assuming there are only 2 labels.

Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of $P(\text{Class is Red} \mid X)$:

$$0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, \text{ and } 0.75.$$

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

Answer. For the majority vote, there are 6 votes for Red, so that the predicted class label is Red. For the second approach, the average probability is

Assuming threshold t = 0.5

$$\frac{0.1 + 0.15 + 0.2 + 0.2 + 0.55 + 0.6 + 0.6 + 0.65 + 0.7 + 0.75}{10} = 0.45$$

so that the predicted class label is Green.

Bias-Variance Tradeoff:

Note that T_b^* is the aggregated tree and T^* is a single bootstrapped tree

Bias: $E\left(T_b^*(x)\right) = E\left(T^*(x)\right)$

- The aggregated tree has the same bias as a single tree. Because they are all from the same distribution

Variance

$$Var\left(T_{bag}(x)\right) \leq Var\left(T^*(x)\right)$$

- The variance of the aggregated tree is much lower as compared to the original

Note that for Bagging, there is no Bias-Variance Tradeoff since the Bias always stays the same for any tree size. But if we increase the tree size, the variance of the aggregated tree will decrease.

Therefore, for Bagging, so long as we decrease the number of trees, the test MSE should be lower. However, note that with a larger tree, it will take up more computational power and we cannot decrease beyond the bias of a single tree.

Choice of B:

Choose a large value of B, as long as computation power can sustain

Out of Bag Error Estimation:

- Note that on average, each bagged tree uses $\frac{2}{3}$ of the original observations. Therefore, for each bagged tree, it should not have seen about $\frac{1}{3}$ of the original data

Suppose we let $\mathcal{T}_i = \{T_b^*$ that do not make use of the ith observation, $(x_i, y_i)\}$

OOB Estimate $\hat{Y}_{i,OOB}$ for x_i

- Bagging of trees in $\mathcal{T}_i$ (What we predict for the ith data point with those trees that have not seen that observation).

For Regression Trees:

$$\frac{1}{|\mathcal{T}_i|}\sum_{T \in \mathcal{T}_i} T(x_i)$$

$|\mathcal{T}_i|$ – Number of trees in $\mathcal{T}_i$

For Classification Trees:

Majority vote by trees in $\mathcal{T}_i$

OOB Error:

- The error that we get for prediction using those in $\mathcal{T}_i$ vs the actual response for the ith observation

For Regression Trees:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_{i,OOB} - Y_i\right)^2$$

- Note that this is just the average error between the predictions of observation i and the actual response for observation i

For Classification Trees:

$$\frac{1}{n}\sum_{i=1}^{n} I\left(\hat{Y}_{i,OOB} \neq Y_i\right)$$

- Note that this is an indicator function whereby we look at the average number of misclassification. We are checking if the label is the same as the actual response. This is essentially misclassification rate.

Random Forests:

(Example on Slide 52 of Lecture 9)

Similar idea to that of Bagging, however when we are building the trees that will be used for the aggregation at the end, every time there is a split in the tree being considered, we will choose m random predictors as the split candidates from the full set of p predictors. The choice of predictor to split on that node will be only those m predictors.

- Helps to reduce the correlation between trees. If the trees are highly correlated, reduction in estimation variance is limited
- For each selection of m, it is independent of one another.
- Note that if $m = p$, then it is just bagging
- Note that it is possible that for m to contain variables that were already selected in a higher node
- Typical choice of $m = \sqrt{p}$

Usage: When there is one very strong predictor, for Bagging, it will most likely be the root node every time and therefore, causing most of the bagged trees to look quite similar, which also means that they are highly correlated. When they are highly correlated, averaging them does not cause a large reduction in variance.

- By randomly selecting the variables for each split, we will be able to decorrelate the trees and aid us in reduction of the variance.

- Why averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities?

Answer. Suppose $\bar{X} = \frac{1}{B}\sum_{b=1}^{B} X_b$. WLOG, assume $\mathbb{E}X_b = 0$ and $\mathrm{Var}(X_b) = \sigma^2$. Then

$$\mathrm{Var}(\bar{X}) = \mathbb{E}\left(\frac{1}{B}\sum_{b=1}^{B} X_b\right)^2 = \frac{1}{B^2}\sum_{b=1}^{B}\mathbb{E}X_b^2 + \frac{1}{B^2}\sum_{a\neq b}\mathrm{Cov}(X_a, X_b)$$

$$= \frac{\sigma^2}{B} + \frac{1}{B^2}\sum_{a\neq b}\mathrm{Cov}(X_a, X_b)$$

In practice, $\sum_{a\neq b}\mathrm{Cov}(X_a, X_b)$ is often positive.

Variable Importance (For Bagging and Random Forests):

(Example on Slide 56 of Lecture 9)

Intuition: A variable is more important if it is used to split nodes more frequently and it leads to overall larger reduction in RSS.

To compute:

For Regression:

Record the total amount of RSS decreased due to splits over the given predictor, averaging the total amount decreased over all B trees. A large value indicates an important predictor

For Classification:

Records the total amount that the Gini Index or Deviance decreased due to splits over the given predictor, averaging the total amount decreased over all B trees. A large value indicates an important predictor

Under R Importance Function:

```
##                %IncMSE IncNodePurity
## CompPrice    11.0245949     147.99672
## Income        2.9562777     116.03560
## Advertising   8.7038546     127.09996
## Population    0.5622538     116.82366
## Price        41.3450591     429.63083
## ShelveLoc    38.7358383     367.43270
## Age          10.0975901     159.43560
## Education     1.7832535      74.28258
## Urban        -0.9239679      14.51735
## US            1.9238949      19.85139
```

- Choose the one with the highest decrease in MSE which corresponds to the highest (%IncMSE) value

For Boosting Importance:
Under the summary() function, we can get the relative influence

```
##                  var    rel.inf
## CAtBat       CAtBat 25.2133452
## CRBI           CRBI 12.5959336
## PutOuts     PutOuts  7.4753238
## Walks         Walks  6.3196327
## CWalks       CWalks  6.2493895
## Years         Years  5.4956576
## RBI             RBI  5.3484365
## Assists     Assists  5.0667371
## AtBat         AtBat  3.9968168
## HmRun         HmRun  3.8327462
## Hits           Hits  3.8173851
## CHmRun       CHmRun  3.6071828
## Runs           Runs  3.4223196
## CHits         CHits  2.2946149
## CRuns         CRuns  2.1013376
## Errors       Errors  1.9727692
## Division   Division  0.5258845
## NewLeague NewLeague  0.4458820
## League       League  0.2186053
```

- Choose the one with the highest (rel. inf) value

Boosting:

Idea: Fit a sequence of small trees, each built upon the residuals of the previous trees. This approach slowly improves the estimation/prediction in areas where it does not perform well previously.

Tuning Parameters:

- B – Number of trees. Boosting can overfit if B is too large. Can use CV to select B
- λ – Shrinkage parameter (small positive number). Controls the learning rate. Very small λ will need high B to achieve good performance since we only add a small amount of information from each tree to the final prediction
  - Typical value – 0.01 and 0.001.
- d – Number of splits in each tree. Controls the complexity of the boosted model. It is the also the interaction depth whereby it controls the interaction, if we have d splits, we can only have at most d variables
  - Typical value is $d = 1$
  - If we have $d = 1$, we will only have 1 split with one root node and 2 terminal nodes
  - If we have $d = 2$, we will only have 2 splits with 1 root node and 3 terminal nodes

Algorithm:

- Fix hyperparameters d and $\lambda \in (0, 1)$
- Set $T(\boldsymbol{x}) = 0$ and $r_i = Y_i$ for all i in the training set
- For $b = 1, 2, \ldots, B$, repeat
  - Fit a tree T_b with d splits to the training data $(\boldsymbol{x}_1, r_1), \ldots, (\boldsymbol{x}_n, r_n)$
  - Update T by adding in a shrunken version of the new tree

$$T(\boldsymbol{x}) \leftarrow T(\boldsymbol{x}) + \lambda T_b(\boldsymbol{x})$$

  - Update the residuals

$$r_i \leftarrow r_i - \lambda T_b(\boldsymbol{x}_i)$$

- Output the boosted model

$$T(\boldsymbol{x}) = \sum_{b=1}^{B} \lambda T_b(\boldsymbol{x})$$

- Trees are built sequentially in boosting, but independently/parallelly in bagging and random forests

- Note that for Boosting, we do not need to divide by B because we just want are updating the model by $T(x) \leftarrow T(x) + \lambda T_b(x)$

Rationale for using small trees in Boosting:
Each tree just needs a small improvement in areas where the previous tree did not do well, so it is not necessary to use a big tree to fit the data too "hard", which actually counters the idea behind boosting.

Bias-Variance Tradeoff:
When $\lambda \uparrow \rightarrow$ Model Bias ↑ and Estimation Variance ↓

- Therefore, if the model bias increase too much, the overall Test MSE will experience a U-Shape of decreasing first then increasing.
- Note that λ is not a penalty term, whereas it is about how much of the tree will add each time.

When $B \uparrow \rightarrow$ Model Bias ↑ and Estimation Variance ↓

- Same as λ

When $d \uparrow \rightarrow$ Model Bias ↑ and Estimation Variance ↓

- Same as λ

Neural Networks:

Activation Functions:

Sigmoid Activation Function:

$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

- Smooth Function
- Firing when $g(z)$ is close to 1 (i.e. input is large)
- Silent when $g(z)$ is close to 0 (i.e. input is small)

ReLU (Rectified Linear Unit) Activation Function):

$$g(z) = \begin{cases} 0 \ \ if\ z < 0 \\ z \ \ otherwise \end{cases}$$

- Non smooth function since it is not differentiable at $x = 0$
- Firing when $g(z)$ is nonzero (i.e. input is non negative)
- Silent when $g(z)$ is zero (i.e. input is negative)

Softmax Activation Function:

- Mostly used for the final output layer or when we have classification

Z_m : Linear combination of inputs from the 2nd hidden Layer (Assuming that we have 2 hidden layers) which is the layer right before the output layer

$$Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} B_{m\ell} A_\ell^{(2)}, \qquad for\ m = 0, \cdots, 9$$

Note that the below activation function is between 0 and 1, the sum of all units in the output layer equals to 1.
The below activation function is checking the probability of the classification being Y_m given that we have the input of X. Note that $Y_m = 1$ denotes that the classification is m

$$f_m(x) = \mathrm{P}(\mathrm{Y_m} = 1|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_m}}$$

Fit the model using the objective function of:
(Cross Entropy)

$$-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im} \log\big(f_m(x_i)\big)$$

$(y_{i0}, \cdots, y_{i9})$ is the one hot encoding for the response of the ith observation

Feed Forward Neural Network:

General Idea:

- Make use of the previous layer as input together with an intercept term and we take a linear combination of them with weights w_i that we want to optimise
- The linear combination will be placed into an activation function and it will denote whether the hidden layer units / output layer unit is activated or not.
- This will keep propagating forwards and the values of the activated units will be passed as inputs to the hidden layer / output layer at the back

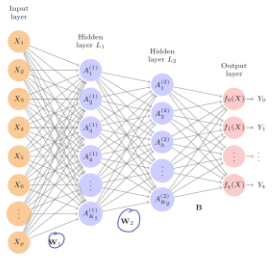
Hidden Layer

- The first hidden layer: For $k = 1, \ldots, K_1$,

$$A_k^{(1)} = g\Big(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j\Big)$$

- The second hidden layer: For $\ell = 1, \ldots, K_2$,

$$A_\ell^{(2)} = g\Big(w_{\ell 0}^{(2)} + \sum_{j=1}^{p} w_{\ell k}^{(2)} A_k^{(1)}\Big)$$



Output Layer:
Number of Units when there are M classes

- When using neural networks for classification, how many units in the output layer if there are M classes?

Answer. The output layer should contain M units. In neural networks, for classification, we encode the class labels by the one-hot encoding, which has M coordinates when there are M classes.

Because one unit will correspond to 1 class

M dimensional vector which only contains a value of 1 that corresponds to the class ith

Total Number of Parameters:

p – Number of input parameters (input units)
K_1 – Number of units in the 1st Hidden Layer
K_2 – Number of units in the 2nd Hidden Layer
O – Number of outputs units

Total:

$$(p+1)K_1 + (K_1+1)K_2 + (K_2+1)O$$

Note that the +1 is because of the Bias term, if we have more layers, we can just build this up recursively using the same idea.

Perceptron Training Rule

Possibilities for weight update:
Suppose $x_i > 0$

(a) **Positive but predicted negative**: $t > 0, o < 0 \rightarrow$ Misclassification occurs → Need to increase the $\underline{w} \cdot \underline{x}$ such that $\underline{w} \cdot \underline{x} > 0 \rightarrow o > 0$
$t - o = 2 \rightarrow \Delta w_i > 0 \rightarrow w_i \uparrow \rightarrow w_i x_i \uparrow \rightarrow \underline{w} \cdot \underline{x} \uparrow$

(b) **Negative but predicted positive**: $t < 0, o > 0 \rightarrow$ Misclassification occurs → Need to decrease the $\underline{w} \cdot \underline{x}$ such that $\underline{w} \cdot \underline{x} < 0 \rightarrow o < 0$
$t - o = -2 \rightarrow \Delta w_i < 0 \rightarrow w_i \downarrow \rightarrow w_i x_i \downarrow \rightarrow \underline{w} \cdot \underline{x} \downarrow$

Note that $o(\underline{x}) = \begin{cases} 1, & \underline{w} \cdot \underline{x} > 0 \\ -1, & \underline{w} \cdot \underline{x} \le 0 \end{cases}$ therefore when we misclassifes as positive, if we put $-o$, it will be the **same sign as *t*** and it will cause w_i to go in the direction of t

Gradient Descent

Simple Linear Unit:

$$o = \underline{w} \cdot \underline{x}$$

- Note that there is no activation function for this as compared to the perceptron unit

Loss Function (Sum of Squared Errors):

$$L_D(\underline{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

D – Set of **training examples**, t_d – **Target Output**, o_d – Output of **linear unit** d – Training example, $\underline{w}$ – **Hypothesis of the weights**

- Note that if we want to minimize the loss function (minimise the number of misclassifications), the constants does not matter
- $L_D = 0$ (**Fully agree**, linearly separable)
- $L_D > 0$ (**Disagrees to some extent**)

Gradient

$$\nabla L_D(\underline{w}) = \left[\frac{\partial L_D}{\partial w_0}, \frac{\partial L_D}{\partial w_1}, \cdots, \frac{\partial L_D}{\partial w_n}\right]$$

- Take the derivative of the loss function w.r.t each of the weights at each point
- Tells the **direction of the steepest increase**

Training Rule

	Weight Update	Weight Change
Vector Form	$\underline{w} \leftarrow \underline{w} + \Delta \underline{w}$,	$\Delta \underline{w} = -\eta \nabla L_D(\underline{w})$
Component Form	$w_i \leftarrow w_i + \Delta w_i$	$\Delta w_i = -\eta \nabla L_D \frac{\partial L_D}{\partial w_0}$

- Note that there is a **-ve** sign for the weight change because we want to find the direction for the **steepest descent**
- η – Learning Rate cannot be **too small** if not the **convergence rate will be too slow** but if it is **too large**, it **may not converge**

Stochastic (Incremental) Gradient Descent

Batch Gradient Descent (GD)	Stochastic Gradient Descent (SGD)
1. Compute true gradient $\nabla L_D(\underline{w})$ 2. $\underline{w} \leftarrow \underline{w} - \eta \nabla L_D(\underline{w})$ where $L_D(\underline{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$	1. Compute stochastic gradient $L_d(\underline{w})$ 2. $\underline{w} \leftarrow \underline{w} - \eta \nabla L_d(\underline{w})$ where $L_d(\underline{w}) = \frac{1}{2}(t_d - o_d)^2$

Computes over the **entire training examples**	**Sample** from the set of training examples with replication. (There is some level of randomness here), each example has a probability of $\frac{1}{\lvert D \rvert}$

SGD approximation of GD:

$$E[\nabla L_d(\underline{w})] = \nabla L_D(\underline{w})$$

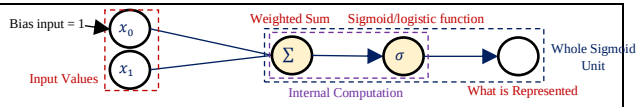
- It is an **unbiased estimator** for the true gradient, but we need **to remove the ½**
- SGD can approximate batch GD arbitrarily closely if learning rate η is sufficiently small (Sufficient condition for convergence)

General Case (Data mini-batch):

$$\min_{\underline{w}} E_{\underline{x}}[L(\underline{x}, \underline{w})]$$
$$\underline{w} \leftarrow \underline{w} - \eta \nabla L(\underline{x}, \underline{w})$$

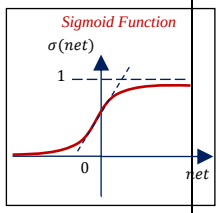
Objective function (differentiable wrt model parameters $\underline{w}$) can be decomposed into a sum of terms, each depending on a subset of training examples

Sigmoid Units



Weighted Sum:

$$net = \sum_{i=0}^{n} w_i x_i$$



Sigmoid/Logistic Function (Output):

$$o = \sigma(net) = \frac{1}{1 + e^{-net}}$$

- Monotonically increasing
- $\sigma(net)$ value is between **0 and 1**
- $net \geq 0 \rightarrow \sigma(net) > \frac{1}{2}$ & $net < 0 \rightarrow \sigma(net) < \frac{1}{2}$
- When net is near 0, it will be a **linear behaviour**

Useful Property:

$$\frac{d\sigma(net)}{dnet} = \sigma(net)(1 - \sigma(net))$$

Issue with linear units:

- Only able to represent a linear function rather than a non-linear function
- The step function is not differentiable

Pros of using sigmoid units:

- Output is non-linear and is a differentiable function so it can be used to represent non-liner functions and we can do gradient descent on it

Convolutional Neural Networks (CNN):

High Level Idea:

1. Build up an image in a hierarchical fashion
2. Identify low level features first in the input image
3. Combine low-level features to form higher-level features (parts of the ears, eyes)

Convolution Layers

- Search for instances of small patterns in the image
- `dot-product`:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \odot \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} = a\alpha + b\beta + c\gamma + d\delta$$

- `convolve` an image with a filter

$$\text{Input Image} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{pmatrix} \quad \text{Convolution Filter} = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

$$\text{Convolved Image} = \begin{pmatrix} a\alpha + b\beta + c\gamma + d\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{pmatrix}$$

- Note that the idea is just to keep sliding the filter from the start and going to the next row when we have completed the current row. Note that it will end once we have covered till the last part
- If the subimage of the input image is similar to the filter, the score through the dot product will be higher and it indicates that a particular feature is present
- Highlights regions that resembles the convolution filter
- Filters are learned during training

Image Size:

Resultant Size of a $n \times n$ image being convolved by a $p \times p$ filter
Result: $(n - p) + 1 \times (n - p) + 1$

Resultant Size of 5×4 being convolved by a 2×2 filter
Result: $(5 - 2) + 1 \times (4 - 2) + 1 = 4 \times 3$ Image

Number of Parameters for Input:

Suppose that we have a $n \times n$ image, with 3 colour channels
No. of input units: $n \times n \times 3$

Pooling Layers:

- Downsample to select a prominent subset

$$\text{Max pool}: \begin{pmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 5 \\ 2 & 4 \end{pmatrix}$$

- Each non-overlapping (e.g., 2×2) block is replaced by its maximum
- Pooling sharpens the feature identification and allows for location invariance
- Reduce the dimension (e.g., by a factor of 4)

Depends on the Type of Pooling that we have, the choice of value that we get at each block is different

1. **Max Pooling**: Take the maximum value
2. **Min Pooling**: Take the minimum value

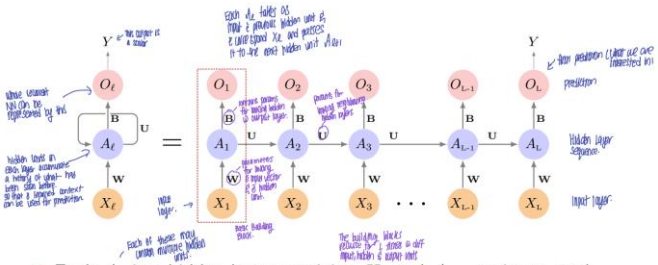
Image Size:

Resultant Size of a $n \times n$ image having 2 x 2 pooling
Result: $\frac{n}{2} \times \frac{n}{2}$

Recurrent Neural Networks (RNN):

- Helps to account for sequential nature of data

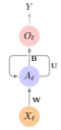
RNN Structure:



- Each A_ℓ is a hidden layer, receiving X_ℓ and $A_{\ell-1}$ as input, and producing an output O_ℓ
- The same weights $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{B}$ are used at each step in the sequence — hence the term recurrent
- The A_ℓ sequence represents an evolving model for the response that is updated as each element X_ℓ is processed
  - Suppose $X_\ell = (X_{\ell 1}, \ldots, X_{\ell p})$, i.e., p-dimensional predictors
  - Suppose $A_\ell = (A_{\ell 1}, \ldots, A_{\ell K})$, i.e., K activations/components

$$A_{\ell k} = g\Big(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{\ell j} + \sum_{s=1}^{K} u_{ks} A_{\ell-1,s}\Big)$$

$$O_\ell = \beta_0 + \sum_{k=1}^{K} \beta_k A_{\ell k}$$



For each of the A_ℓ, we will make use of previous units as inputs, as well as the new X_l as input.

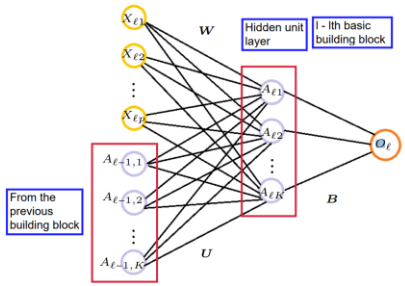
Note that the weights that we use for each of A_ℓ into the output layer is the same as well, β_k since it is a recurrent network, the only different is the number of inputs that is different.

Note that the weights that we use for the previous A_ℓ, u_k is the same for all previous inputs and w_k is also the same throughout. We do not change the weights for different A_ℓ, it is only the number of inputs that is different.

Expanding the Basic Building Block:

- In the RNN, suppose the output is a scalar, each element in the input sequence is a p-dimensional vector, and each hidden layer has K units. Expand each basic building block by a single-layer neural network.

Answer.



Training:

Minimise $\sum_{i=1}^{n}(y_i - o_{iL})^2$ whereby we are only concerned with the final prediction at the last unit. Note that the below formula is a recursive formula where A_{L-1} are recursively built from the first formula and when we try to minimise, it will open up to have all the data.

- Data pairs $(\boldsymbol{x}_i, y_i)$ with
  - $\boldsymbol{x}_i = (\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{iL})$ and
  - each $\boldsymbol{x}_{i\ell} = (x_{i\ell 1}, \ldots, x_{i\ell p}) \in \mathbb{R}^p$
- Training: Minimize $\sum_{i=1}^{n}(y_i - o_{iL})^2$ when only the prediction O_L at the last unit is concerned

$$\sum_{i=1}^{n}(y_i - o_{iL})^2 = \sum_{i=1}^{n}\left\{y_i - \beta_0 - \sum_{k=1}^{K}\beta_k g\Big(w_{k0} + \sum_{j=1}^{p} w_{kj}\boxed{x_{iLj}} + \sum_{s=1}^{K} u_{ks}\boxed{a_{i,L-1,s}}\Big)\right\}^2$$

where $a_{i,L-1,s}$ is the value of $A_{L-1,s}$ when the input is the sequence $\boldsymbol{x}_i$

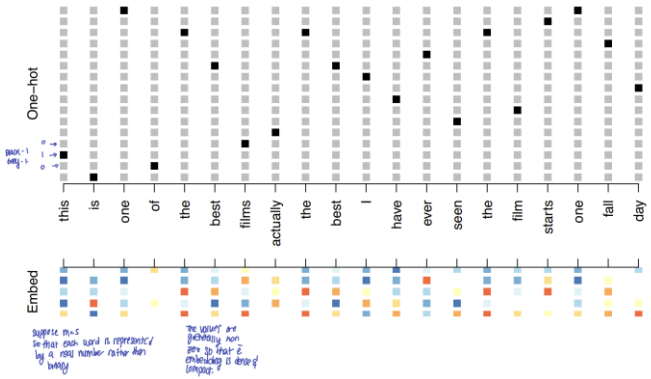
Feature Engineering:

For the encoding of words, instead of doing the one-hot encoding, we can do Word Embedding.

Assuming that we have D words

One-hot Encoding: It is just a binary vector of dimension D, we will have a D dimensional vector to indicate the existence of a certain word. Usually very sparse since most entries are 0

Word Embedding: Embed the words using a $m \times D$ matrix whereby $m \ll D$ and can take on more values instead of just binary (0, 1)



We see that for each word, instead of having D dimensions, it will be $m = 5$ instead for this case and to store all our words it will be much simpler and efficient.

Gradient Descent:

Issues:

1. Starting values are important and having certain starting values may result in suboptimal results or the algorithm not working at all
   a. Partially addressed by repeating with multiple starting points.

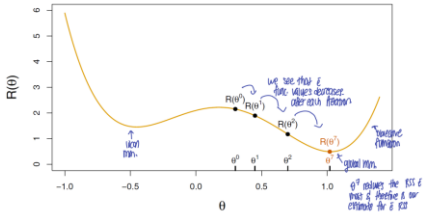
- Starting values:
  - exact zero starting value θ^0 leads to zero derivatives, and the algorithm never moves
  - large starting value θ^0 often leads to poor solutions
  - usually starting value θ^0 is chosen to be a random value near zero
- Multiple minima: partially addressed by trying multiple starting values

Algorithm:

Gradient Descent

Let $R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(\boldsymbol{x}_i))^2$ with θ being a vector of all parameters

1. Start with an initial value θ^0 of θ, and set $t = 0$
2. Iterate until the objective function $R(\theta)$ fails to decrease
   - Find a vector δ that reflects a small change in θ, such that $\theta^{t+1} = \theta^t + \delta$ reduces the objective function, i.e., $R(\theta^t + \delta) = R(\theta^{t+1}) < R(\theta^t)$
   - Set $t \leftarrow t + 1$



Backpropagation

- Let

$$R_i(\theta) = \frac{1}{2}(y_i - f_\theta(\boldsymbol{x}_i))^2 = \frac{1}{2}\left\{y_i - \beta_0 - \sum_{k=1}^{K}\beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\right)\right\}^2$$

- So that $R(\theta) = \sum_{i=1}^{n} R_i(\theta)$ and $\nabla R(\theta^t) = \sum_{i=1}^{n} \nabla R_i(\theta^t)$
- Let $z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}$
- Backpropagation: For $k \geq 1$ and $j \geq 1$

$$\begin{aligned}\frac{\partial R_i(\theta)}{\partial \beta_k} &= \frac{\partial R_i(\theta)}{\partial f_\theta(\boldsymbol{x}_i)} \cdot \frac{\partial f_\theta(\boldsymbol{x}_i)}{\partial \beta_k} \\ &= -(y_i - f_\theta(\boldsymbol{x}_i)) \cdot g(z_{ik}) \\ \frac{\partial R_i(\theta)}{\partial w_{kj}} &= \frac{\partial R_i(\theta)}{\partial f_\theta(\boldsymbol{x}_i)} \cdot \frac{\partial f_\theta(\boldsymbol{x}_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}} \\ &= -(y_i - f_\theta(\boldsymbol{x}_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}\end{aligned}$$

- In summary

$$\theta^{t+1} = \theta^t - \rho\nabla R(\theta^t)$$

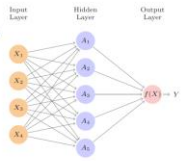
and

$$\begin{aligned}\frac{\partial R_i(\theta)}{\partial \beta_k} &= -(y_i - f_\theta(\boldsymbol{x}_i)) \cdot g(z_{ik}) \\ \frac{\partial R_i(\theta)}{\partial w_{kj}} &= -(y_i - f_\theta(\boldsymbol{x}_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}\end{aligned}$$

- Forward pass: Given θ^t, calculate $f_{\theta^t}(\boldsymbol{x}_i)$ and z_{ik}^t, as well as $g(z_{ik}^t)$ and $g'(z_{ik}^t)$
- Backward pass: Update parameters by

$$\beta_k^{t+1} = \beta_k^t + \rho \cdot \sum_{i=1}^{n}(y_i - f_{\theta^t}(\boldsymbol{x}_i)) \cdot g(z_{ik}^t)$$

$$w_{kj}^{t+1} = w_{kj}^t + \rho \cdot \sum_{i=1}^{n}(y_i - f_{\theta^t}(\boldsymbol{x}_i)) \cdot \beta_k^t \cdot g'(z_{ik}^t) \cdot x_{ij}$$



- Find the backpropagation formula for β_0 and w_{k0}.

Answer.

- Recall $R(\theta) = \sum_{i=1}^{n} R_i(\theta)$ and $\nabla R(\theta^t) = \sum_{i=1}^{n} \nabla R_i(\theta^t)$, with

$$R_i(\theta) = \frac{1}{2}(y_i - f_\theta(\boldsymbol{x}_i))^2 = \frac{1}{2}\left\{y_i - \beta_0 - \sum_{k=1}^{K}\beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\right)\right\}^2$$

- Let $z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}$
- Backpropagation formulas for β_0 and w_{k0}:

$$\begin{aligned}\frac{\partial R_i(\theta)}{\partial \beta_0} &= \frac{\partial R_i(\theta)}{\partial f_\theta(\boldsymbol{x}_i)} \cdot \frac{\partial f_\theta(\boldsymbol{x}_i)}{\partial \beta_0} \\ &= -(y_i - f_\theta(\boldsymbol{x}_i)) \times 1 \\ \frac{\partial R_i(\theta)}{\partial w_{k0}} &= \frac{\partial R_i(\theta)}{\partial f_\theta(\boldsymbol{x}_i)} \cdot \frac{\partial f_\theta(\boldsymbol{x}_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{k0}} \\ &= -(y_i - f_\theta(\boldsymbol{x}_i)) \cdot \beta_k \cdot g'(z_{ik}) \times 1\end{aligned}$$

Regularization:

- Helps to prevent overfitting to the training data

Penalty:

Regularization via Penalty

- Recall the one-hot encoding: $y_i = (y_{i0}, \ldots, y_{i9})$ with

$$y_{im} = \begin{cases} 1 & \text{if the } i\text{th observation is the digit } m \\ 0 & \text{otherwise} \end{cases}$$

- Let $\theta = (\theta_1, \ldots, \theta_d)$ be the vector of the d parameters
- Ridge-type penalty

$$R(\theta;\lambda) = -\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log(f_m(\boldsymbol{x}_i)) + \lambda\sum_{j=1}^{d}\theta_j^2$$

- Lasso-type penalty

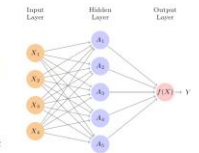
$$R(\theta;\lambda) = -\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log(f_m(\boldsymbol{x}_i)) + \lambda\sum_{j=1}^{d}|\theta_j|$$

Stochastic Gradient Descent:

- Update only using a few observations. Note that the sampling is done on observations that are not yet sampled.
- Forward pass: Given θ^t, calculate $f_{\theta^t}(\boldsymbol{x}_i)$
- Backward pass: Update parameters by

$$\beta_k^{t+1} = \beta_k^t + \rho \cdot \sum_{i=1}^{n}(y_i - f_{\theta^t}(\boldsymbol{x}_i)) \cdot g(z_{ik}^t)$$

$$w_{kj}^{t+1} = w_{kj}^t + \rho \cdot \sum_{i=1}^{n}(y_i - f_{\theta^t}(\boldsymbol{x}_i)) \cdot \beta_k^t \cdot g'(z_{ik}^t) \cdot x_{ij}$$



- SGD: For each t, in the backward pass, instead of $\sum_{i=1}^{n}$ over all n observations, only sum over a random minibatch (a small fraction)
- Example: Suppose $n = 10$ and the minibatch size is 3. At the step $t+1$, we sample 3 observations. Suppose they are indexed by $1, 4, 9$. Then SGD updates

$$\beta_k^{t+1} = \beta_k^t + \rho \cdot \sum_{i\in\{1,4,9\}} (y_i - f_{\theta^t}(\boldsymbol{x}_i)) \cdot g(z_{ik}^t)$$

$$w_{kj}^{t+1} = w_{kj}^t + \rho \cdot \sum_{i\in\{1,4,9\}} (y_i - f_{\theta^t}(\boldsymbol{x}_i)) \cdot \beta_k^t \cdot g'(z_{ik}^t) \cdot x_{ij}$$

Questions about SGD:

- In SGD, does the minibatch remain the same for different t? **Answer.** No, for different t, we have different minibatch. In addition, in practice, a minibatch is often drawn without replacement.
- If the minibatch size is 10 and the sample size is $n = 100$, then how many minibatch gradient updates per epoch? **Answer.** It is $100/10 = 10$.
- **Epoch** – The number of times an equivalent of the full training set has been processed. (If n=48000, then each epoch contains $48000/128 \approx 375$ minibatch gradient updates)

Early Stopping: So that we don't fit to the data too much

Dropout:

At each of the SGD Update, we randomly remove a fraction ϕ of (both input and hidden) units.

The remaining weights of the surviving units are scaled up by a factor of $\frac{1}{1-\phi}$

- At each SGD update, randomly remove a fraction ϕ of (both input and hidden) units
- The weights of the surviving units are scaled up by a factor of $1/(1-\phi)$
- In practice dropout is achieved by setting the activations for the "dropped out" units to zero, while keeping the architecture intact.

Sure Independence Screening:

Motivation: For $p \gg n$, it maybe hard to identify relevant predictors for LASSO and LARS. Idea is to screen out irrelevant predictors then perform variable selection.

Steps:

1. Standardization and then computing r_j

- Standardization:
  - Center the response by redefining $y_i := y_i - \bar{y}$, where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
  - Standardize predictors $x_{ij} := (x_{ij} - \bar{x}_j)/\sigma_j$ with

Standardising the response first and also the predictors

After standardisation, all the predictors will be in the same scale

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \quad \text{and} \quad \sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$$

- Let $r_j = n^{-1}\sum_{i=1}^{n} x_{ij} y_i$

Correlation between each response and the predictor

This formula of the correlation is because of the standardisation

- Interpretation of each r_j: It is the correlation between the jth predictor and the response
- Intuitively, if the jth predictor X_j is irrelevant to the response, then r_j should be small in magnitude

2. Choose q to be the number of predictors to be kept after screening
   a. Keep the predictors whose $|r_j|$ is the first q largest of $|r_1|, \cdots, |r_p|$. Note that this is the largest absolute correlation
   b. We usually choose $q < n$ and $q = p$ if $p < n$.
   c. Or we can choose $q = \left\lfloor \frac{n}{\log n} \right\rfloor$

Combination with LASSO and LARS:

SIS-Lasso:

1. Perform SIS to screen out the predictors
2. Perform LASSO on the surviving predictors

SIS-LARS:

1. Perform SIS to screen out the predictors
2. Perform LARS on the surviving predictors

Intuition:

Both steps can reduce the predictor dimension, with different emphasis
SIS: Screen out most irrelevant predictors, while trying to keep all relevant predictors. q is often chosen to be a not that small number such as (n/logn). Note that the surviving predictors in the first step may still contain many irrelevant predictors.

LASSO: After the number of predictors are substantially reduced, LASSO can have a reasonable performance in the second step

If there are 1million predictors and 100 observations, direct application of lasso in this case is often challenging. If we use SIS, we can bring the dimensions down to 20. In the second step, we will be having 20 predictors and we can therefore, expect a reasonable performance of LASSO in this case

Iterative SIS

(*Example on Page 7 of Lecture Slide 11*)

- The SIS tends to miss some important predictors
- A remedy: Iterate SIS to keep more predictors
- Center the response and standardize the predictors

Same standardisation and centering of the data

- Step 0: Apply SIS on the data to screen out most predictors
- Let $\mathscr{A}$ be the set of surviving predictors
- Iterate the following:
  - Step 1: Apply variable selection (e.g., lasso) to select some predictors from $\mathscr{A}$, and update $\mathscr{A}$ to be the selected predictors
  - Step 2: Fit a linear model with y_i and the predictors in $\mathscr{A}$, and compute the residuals R_i
  - Step 3: Apply SIS on the data with R_i as response values and with the predictors NOT in $\mathscr{A}$. This results in some surviving predictors not in $\mathscr{A}$. Add these predictors into $\mathscr{A}$, if the number of predictors $\mathscr{A}$ is less than q

We see that this is like regressing the residuals with the remaining predictors and see which only explains the residuals the best

Repeat until A has q predictors

- Note that for this we will keep iterating until we get q predictors in the set A.

Hidden Markov Models

Markov Assumption / Property:

- Conditional on Q_i, Q_{i+1} is independent of $Q_1, \cdots, Q_{i-1}$
- The current state is only dependent on the state right before it

$$P(Q_i = t_{i+1} | Q_1 = t_1, c \ldots, Q_i = t_i) = P(Q_{i+1} = t_{i+1} | Q_i = t_i)$$

Time-Homogeneous Markov Chain:

Conditional Probability of going from s to t is the same no matter what the time is.

Distribution of Time-Homogeneous Markov Chain is completely determined by:

1. Initial Distribution
2. Transition Probability

For any $q_1, \ldots, q_m \in S$, write $P(q_1, \ldots, q_m)$ for $P(Q_1 = q_1, \ldots, Q_m = q_m)$. Then

$$\begin{aligned}
P(q_1, \ldots, q_m) &= P(q_m \mid q_1, \ldots, q_{m-1}) P(q_1, \ldots, q_{m-1}) \\
&= P(q_m \mid q_{m-1}) P(q_1, \ldots, q_{m-1}) \\
&= P(q_m \mid q_{m-1}) P(q_{m-1} \mid q_1, \ldots, q_{m-2}) P(q_1, \ldots, q_{m-2}) \\
&= P(q_m \mid q_{m-1}) P(q_{m-1} \mid q_{m-2}) P(q_1, \ldots, q_{m-2}) \\
&= \cdots \cdots \\
&= P(q_m \mid q_{m-1}) P(q_{m-1} \mid q_{m-2}) \cdots P(q_2 \mid q_1) P(q_1)
\end{aligned}$$

In the above, each $P(q_j \mid q_{j-1})$ is given by a transition probability, and $P(q_1)$ is given by the initial distribution.

Terminologies

Symbol Space, T

- Symbols that we can take

Sequence of Observations of the Symbols, $X = (X_1, \cdots, X_n)$

- What we actually observe for the observations

State Space, S

- States that could occur

Sequence of Hidden Random Variables, $Q = (Q_1, \cdots, Q_n)$

Initial probability distribution, π

$$\pi = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_p \end{pmatrix}$$

Transition Matrix, A

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix}$$

- Every row is the starting position and the columns are the position that we are gong to next

Emission Probability Matrix, B

- Specifies the probability that the symbol t is emitted when the state is s

$$B_1 = \begin{bmatrix} P(t_1|s_1) \\ P(t_2|s_1) \\ P(t_3|s_1) \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix}$$

Note that this Emission Probability Matrix is associated with the state s_1

First Order Hidden Markov Model (HMM):

- **Markov Assumption**: $Q_1, \cdots, Q_n$ form a time-homogeneous Markov Chain
- **Output Independence**: The distribution of X_i depends only on the hidden variable Q_i

$$P(X_i | Q_1, \cdots, Q_i, \cdots, Q_n, X_1, \cdots, X_{i-1}, X_{i+1}, \cdots, X_n) = P(X_i | Q_i)$$

- Completely determined by the initial distribution, the transition probability matrix and emission probability matrix

Answer. It is equivalent to showing that the joint probability $P(\boldsymbol{x}, \boldsymbol{q})$ can be calculated from π, A and B.

$$\begin{aligned}
P(\boldsymbol{x}, \boldsymbol{q}) &= P(\boldsymbol{x} \mid \boldsymbol{q}) P(\boldsymbol{q}) \\
&= \{P(x_1 \mid q_1) \times \cdots \times P(x_m \mid q_m)\} \times \\
&\quad \{P(q_m \mid q_{m-1}) \times P(q_2 \mid q_1) P(q_1)\}.
\end{aligned}$$

The probability, $P(x_j \mid q_j)$, is computed from B. The probability, $P(q_j \mid q_{j-1})$, is computed from A. The probability $P(q_1)$ is computed from π.

Computing P(X|Q):

Suppose that we have a sequence of observations $X = (X_1, \cdots, X_n)$, and a corresponding sequence of hidden states $Q = (Q_1, \cdots, Q_n)$

$$\begin{aligned}
P(X|Q) &= P(X_1, \cdots, X_n | Q_1, \cdots, Q_n) \\
&= P(X_1|Q_1) \times \cdots \times P(X_m|Q_m)
\end{aligned}$$

- Note that the above result is by doing a decomposition on the joint probability and also making use of the Output independence. (Example on Slide 36 of Lecture 11)

Computing P(Q)

Suppose that we have a sequence of observations $X = (X_1, \cdots, X_n)$, and a corresponding sequence of hidden states $Q = (Q_1, \cdots, Q_n)$

$$\begin{aligned}
P(Q) &= P(Q_1, \cdots, Q_n) \\
&= P(Q_n|Q_{n-1}) \times \cdots \times P(Q_2|Q_1) \times P(Q_1)
\end{aligned}$$

- Note that the above result follows from the Decomposition on the Joint Distribution and the Markov Assumption.

Computing P(X):

Suppose that we have a sequence of observations $X = (X_1, \cdots, X_n)$, and a corresponding sequence of hidden states $Q = (Q_1, \cdots, Q_n)$

$$P(X) = \sum_Q P(X|Q)P(Q)$$

- Note that this will just make use of the P(X|Q) and P(Q) where we will sum over all possible states that occurred at $t = 1, \cdots, n$

- In the ice cream example, compute the probability $P(HHC)$.

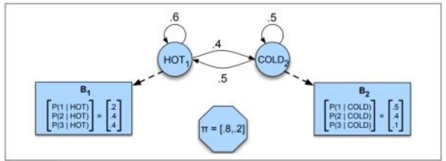


Figure A.2 A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

Answer. By the Markov property,

$$\begin{aligned}
P(HHC) &= P(C \mid HH) P(HH) = P(C \mid H) P(H \mid H) P(H) \\
&= 0.4 \times 0.6 \times 0.8 = 0.192.
\end{aligned}$$

Estimation of the parameters when we have a set of data:
(Example is on Page 44 of Lecture 11)
Suppose that the hidden sequence $q_1, \cdots, q_n$ are missing data. If we have the complete data of $(x_1, q_1), \cdots, (x_n, q_n)$

Initial Distribution:

$$\pi = (\pi_1, \cdots, \pi_p)$$
$$\hat{\pi}_j = \frac{n_j}{n}$$

n – Total number of observations
n_j – Total number of observations with state s_j at the start

- Estimated by the fraction of sequences starting with state s_j among all hidden sequences

Transition Probability:

$$A = (a_{jk})_{j,k=1}^{p}$$

$$\hat{a}_{j,k} = P(q_{t+1} = s_k | q_t = s_j)$$
$$= \frac{n_{jk}}{n_j}$$

n_{jk} – Those pair of states whereby for time $t-1$ it was at s_j and at time t it was at s_k
n_j – The number of observations that were in state s_j at the time $t-1$

- Estimated by the fraction of adjacent pairs of hidden states such that the first hidden state is s_j and the second hidden state is s_k among all states that have state s_j at the current iteration
- Intuitively, it is just out of all those that were at s_j, how many went on to s_k

Emission Probability:

$$b_{jk} = P(s_j \to t_k)$$

$$\hat{b}_{jk} = \frac{n_{jk}}{n_j}$$

n_{jk} – Those pair of states whereby for time t, the state is s_j and the symbol is t_k
n_j – The number of observations that were in state s_j at the time t

- Fraction of instances emitting the symbol t_k when the state is s_j among all the instances with state s_j
- Intuitively, it is just out of all those that were at s_j, how many have the symbol, t_k

EM Algorithm:

For Gaussian Models:

Probability Density:

$$f(x) = (1-r)\phi(x;\mu_0,\sigma^2) + r\phi(x;\mu_1,\sigma^2)$$

$\phi(x;\mu,\sigma^2)$ – Probability Density of the Gaussian Distribution with mean μ_i and variance sigma

- Note that this is a mixture density and the relative weights are the $(1-r)$ and r
- Note that the 2 Gaussians have the same variance, σ^2 but different means

Expectation Part:

Note that k is the number of iteration and i is for the ith iteration. Note that the $\hat{Z}_{i,k}$ we can think of it as the probability of X_i being in the μ_1 group. For the probability of being in the μ_2 group, it will just be $1 - \hat{Z}_{ik}$

Computing the $\hat{Z}_{ik}$

$$\hat{Z}_{ik} = P(Z_i = 1|X_i) = \frac{P(X_i|Z_i=1)P(Z_i=1)}{P(X_i)}$$

$$= \frac{r_{k-1}\phi\left(X_i;\mu_{1,k-1}\sigma^2_{k-1}\right)}{r_{k-1}\phi\left(X_i;\mu_{1,k-1}\sigma^2_{k-1}\right) + (1-r_{k-1})\phi\left(X_i;\mu_{0,k-1}\sigma^2_{k-1}\right)}$$

Maximization Part:

Update the parameters as follows:

$$\mu_{1k} = \frac{\sum_{i=1}^n \hat{Z}_{ik}X_i}{\sum_{i=1}^n \hat{Z}_{ik}}$$

Intuition: This is a weighted average of the X_i values to determine the mean of group 1. Those with higher weights will affect the mean more, those with lesser weights will not affect as much.

$$\mu_{0k} = \frac{\sum_{i=1}^n (1-\hat{Z}_{ik})X_i}{n - \sum_{i=1}^n \hat{Z}_{ik}}$$

Intuition: This is a weighted average of the X_i values to determine the mean of group 0. Those with higher weights will affect the mean more, those with lesser weights will not affect as much. Note that we are using $1 - \hat{Z}_{ik}$ since that is the probability of being in group 0 and we will sum over those only.

$$\sigma_k^2 = \frac{1}{n}\sum_{i=1}^n \hat{Z}_{ik}(X_i - \mu_{1k})^2 + \frac{1}{n}\sum_{i=1}^n (1-\hat{Z}_{ik})(X_i - \mu_{0k})^2$$

Intuition: This is just like an average of the variance but with the relative weightage from the respective distributions

$$r_k = \frac{1}{n}\sum_{i=1}^n \hat{Z}_{ik}$$

Intuition: This is the mixture weight for group 1. We can basically just take the average of the probabilities for them being in group 1. Note that to find the mixture weight for the other Gaussian, it will just be $1 - r_k$.

K-Means Clustering:

Within-Cluster-Variation

$$WCV(C_k) = \frac{1}{n_k} \sum_{i_1, i_2 \in C_k} \sum_{j=1}^{p} (x_{i1,j} - x_{i_2,j})^2$$

n_k – Number of elements in C_k

- Idea is that the distance between points in the same cluster should be small.
- The larger the variation of a cluster, it indicates a larger difference on average among all observations within the cluster.
- We are summing over all pairs of points in the cluster and finding the average distance between them to be the Within-Cluster-Variation
- *Note that if we have only 1 point in every cluster, the variation will be 0*

Algorithm:

Data: n observations, $x_1, \cdots, x_n \in \mathbb{R}^p$, where p is the number of predictors

1. (Initialization) Randomly assign a number, from 1 to K, to each of the observations to denote which cluster they will belong to
2. Iterate until the cluster assignments stops changing
   a. For each of the K clusters, compute the centroid (*Just need to take the average of all the points in the current centroid*)

$$\mu_k = \frac{1}{n_k} \sum_{i \in C_k} x_i$$

   b. Assign each observation to the cluster whose centroid it is the closest to, i.e. place each index i into C_k such that k is the minimizer of

$$\min_k \sqrt{\sum_{j=1}^{p} (x_{ij} - \mu_{kj})^2}$$

Note that j here is the dimensions

Hierarchical Clustering:

Dissimilarity Measures:

Comparison between Points:

Euclidean Distance

$$D(x_i, x_l) = \sqrt{\sum_{j=1}^{p} (x_{ij} - x_{lj})^2}$$

Note that x_i and x_l are p dimensional vectors

Comparison between Clusters:

- We will look at the linkages, but note that we will still want to have the minimal linkage no matter which one we choose as the dissimilarity measure

Complete Linkage:

- Maximum distance between all pairwise distance of observations in C_k vs observations in C_q. (Take all pairs of (x_i from C_k, x_l from C_q))

$$Complete - Linkage(C_k, C_q) = \max_{i \in C_k, l \in C_q} D(x_i, x_l)$$

Gives more balanced Dendrograms

Single Linkage:

- Minimum distance between all pairwise distance of observations in C_k vs observations in C_q. (Take all pairs of (x_i from C_k, x_l from C_q))

$$Single - Linkage(C_k, C_q) = \min_{i \in C_k, l \in C_q} D(x_i, x_l)$$

Gives more unbalanced Dendrograms

Average Linkage:

- Kind of like the average distance between all pairwise distance of observations in C_k vs observations in C_q. (Take all pairs of (x_i from C_k, x_l from C_q))

$$Average - Linkage(C_k, C_q) = \frac{1}{n_k n_q} \sum_{i \in C_k, l \in C_q} D(x_i, x_l)$$

Gives more balanced Dendrograms

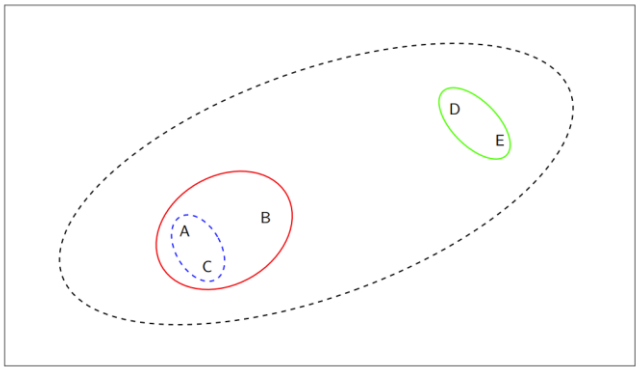
Centroid Linkage:

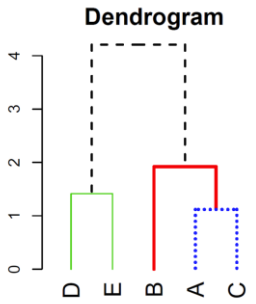
- Distance between the centroids

$$Centroid - Linkage(C_k, C_q) = D(\mu_k, \mu_q)$$
$$\mu_k = \frac{1}{n_k} \sum_{i \in C_k} x_i$$

Where μ_k is the centroid of Cluster C_k and μ_q is the centroid of Cluster C_q

Reading a Dendrogram





Suppose that we have the above dendrogram

- Note that we will start off as having no clusters, therefore, it is 5 groups
- If clusters were merged at a lower point, they are more similar to each other
- The vertical axis is the dissimilarity measure value that we have chosen.
- Note that the similarity is not based on the proximity of points in the horizontal axis.
- Terminal Nodes are Single Observations

Find out the dissimilarity measure between B, (A, C): Just need to read off the vertical axis value where they merge (This case it is 2)

Find out the Clusters when $k = 3$: Cut across horizontally whereby there will only be 3 vertical lines at that point. For instance, we cut at 1.5, then we look downwards each of the lines and that will be the various clusters. (In this case, we have $(D, E), (B), (A, C)$)

Hypothesis Testing:

Definition of Null and Alternative Hypothesis:

Null Hypothesis: Default state of belief about the world
Alternative Hypothesis: Represents something different and unexpected

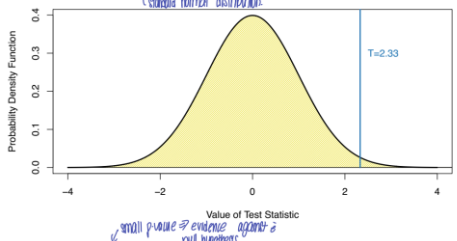
T-Statistics and p-value:

- The test statistic summarizes the extent to which our data are consistent with H_0.
- Let $\hat{\mu}_t$ / $\hat{\mu}_c$ respectively denote the average blood pressure for the n_t / n_c mice in the treatment and control groups.
- To test $H_0 : \mu_t = \mu_c$, we use a two-sample t-statistic

$$T = \frac{\hat{\mu}_t - \hat{\mu}_c}{s\sqrt{\frac{1}{n_t} + \frac{1}{n_c}}}$$

- If T is large, we can expected observed data to be inconsistent with the null hypothesis.

- The p-value is the probability of observing a test statistic at least as extreme as the observed statistic, *under the assumption that H_0 is true*.
- A small p-value provides evidence *against* H_0.
- Suppose we compute $T = 2.33$ for our test of $H_0 : \mu_t = \mu_c$.
- Under H_0, $T \sim N(0, 1)$ for a two-sample t-statistic.



- The p-value is 0.02 because, if H_0 is true, we would only see $|T|$ this large 2% of the time.
- P-value is checking that if the null hypothesis is correct, what is the probability of observing a T-Statistics as extreme as that

If we were asked to compute the p-value:

- Assume that the null hypothesis is true, think of all possible scenarios where it could be the same or more extreme
  - If we reject the null hypothesis that the coin is fair when the coin come up all tails. Suppose, among 10 flips, 7 are tails. What is the p-value? **Answer.** The events that are at least as extreme as the observed one (7 tails out of 10 flips) are
    - 7 tails out of 10 flips, with probability $\binom{10}{7}P(T^7)P(H^3) = 120/1024$
    - 8 tails out of 10 flips, with probability $\binom{10}{8}P(T^8)P(H^2) = 45/1024$
    - 9 tails out of 10 flips, with probability $\binom{10}{9}P(T^9)P(H) = 10/1024$
    - 10 tails out of 10 flips, with probability $1/1024$.

    Therefore, the p-value is

$$\frac{120 + 45 + 10 + 1}{1024} = \frac{176}{1024}.$$

Decision to Reject Null Hypothesis:

- Small-value of p-value indicates that there is evidence against the H_0 since the large value of T is unlikely to occur under the null hypothesis

		Truth	
		H_0	H_a
Decision	Reject H_0	Type I Error	Correct
	Do Not Reject H_0	Correct	Type II Error

Truth: H_0, Decision: Reject H_0

- Type 1 Error, because even though H_0 is correct, we are rejecting it because we thought that it is too extreme
- We normally prioritise this instead of Type II error, because this is saying that we have a new discovery when there actually isn't.
- Note that if we reject H_0 when the p-value $\leq \alpha$ then the Type I error rate is at most α because we are rejecting with at most α probability that it is actually correct.

Truth: H_a, Decision: Reject H_0

- Correct, because since if H_a is correct, then it means that we should actually reject the H_0 since we are actually having a new discovery.

Truth: H_0, Decision: Do not Reject H_0

- Correct, because since H_0 is correct, it means that we should not have evidence against it. Therefore, we shouldn't reject it.

Truth: H_a, Decision: Do Not Reject H_0

- Type II Error, because even though H_a is correct, we are not rejecting the null and determining that we have a new discovery.

Multiple Testing:

If we want to test m hypothesis, $H_{01}, \cdots, H_{0m}$

As compared to testing the hypothesis 1 time, if we were to test multiple hypothesis at a time, we are almost certain to get one very small p-value by chance. (Therefore, we could potentially falsely reject one null hypothesis when it is actually true and committing Type I Error)

- If we reject any null hypothesis that is under 0.01, we can expect to reject $0.01 \times m$ null hypotheses
- If $m = 10000$, we will reject 100 null hypotheses by chance (False Positives)

Family-Wise Error Rate (FWER):

Probability of making at least 1 Type I error when conducting m hypothesis tests. Focuses on falsely rejecting any null hypothesis by any pure coincidence

$$FWER = P(V \geq 1)$$

Where V is the number of Type I errors committed

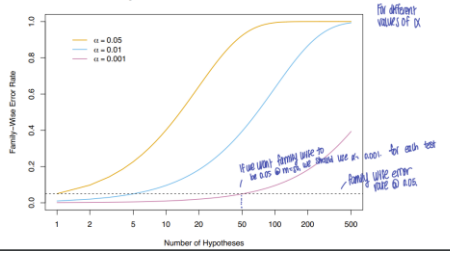
	H_0 is True	H_0 is False	Total
Reject H_0	V	S	R
Do Not Reject H_0	U	W	$m - R$
Total	m_0	$m - m_0$	m

Formula:

$$\begin{aligned} \text{FWER} &= 1 - \Pr(\text{do not falsely reject any null hypotheses}) \\ &= 1 - \Pr\left(\bigcap_{j=1}^{m} \{\text{do not falsely reject } H_{0j}\}\right). \end{aligned}$$

If the tests are independent and all H_{0j} are true then

$$\text{FWER} = 1 - \prod_{j=1}^{m}(1-\alpha) = 1 - (1-\alpha)^m.$$



If the tests are highly positively correlated, testing m hypotheses is like testing 1 hypothesis when FWER is concerned.

Answer. Imagine the situation that $A_1 = A_2$. Then

$$\begin{aligned} FWER &= \Pr(V \geq 1) \\ &= 1 - \Pr(A_1 \cap A_2) \\ &= 1 - \Pr(A_1) \\ &= \alpha. \end{aligned}$$

When $A_1 \approx A_2$, then FWER is approximately α, which is relatively smaller than the FWER in (b). In other words, when the tests are highly positively correlated, testing m hypotheses is like testing one hypothesis, when FWER is concerned.

If they are negatively correlated, the FWER will be substantially inflated if we do not control it properly.

(d) Suppose again that $m = 2$, but that now the p-values for the two tests are negatively correlated, so that if one is large then the other will tend to be small. How does the family-wise error rate associated with these $m = 2$ tests qualitatively compare to the answer in (b) with $m = 2$? Hint: First, suppose that whenever one p-value is less than α, then the other will be greater than α. In other words, we can never reject both null hypotheses.

Answer. Imagine that $A_1 \cap A_2 = \emptyset$. Then

$$\begin{aligned} FWER &= \Pr(V \geq 1) \\ &= 1 - \Pr(A_1 \cap A_2) \\ &= 1 - \Pr(\emptyset) \\ &= 1. \end{aligned}$$

In other words, if tests a negatively correlated, FWER will be substantially inflated if not controlled properly.

Bonferroni Correction:

Control the FWER at level α, therefore, we reject any null hypothesis with p-value below $\frac{\alpha}{m}$.

Intuition: Since we are only rejecting each of the null hypothesis at a level of $\frac{\alpha}{m}$, it must mean that we at most reject those values at an error rate of $\frac{\alpha}{m}$ and the sum of m null hypotheses will have at most α level of error rate

$$\begin{aligned} \text{FWER} &= \Pr(\text{falsely reject at least one null hypothesis}) \\ &= \Pr(\cup_{j=1}^{m} A_j) \\ &\leq \sum_{j=1}^{m} \Pr(A_j) \end{aligned}$$

where A_j is the event that we falsely reject the jth null hypothesis.

- If we only reject hypotheses when the p-value is less than α/m, then

$$\text{FWER} \leq \sum_{j=1}^{m} \Pr(A_j) \leq \sum_{j=1}^{m} \frac{\alpha}{m} = m \times \frac{\alpha}{m} = \alpha,$$

because $\Pr(A_j) \leq \alpha/m$.

- This is the *Bonferroni Correction*: to control FWER at level α, reject any null hypothesis with p-value below α/m.

Basic Idea: If we want to control at α, just divide by m and compare with each of the hypotheses' p-values. If $p - value < \frac{\alpha}{m}$ then we reject it. Or the same can be done by multiplying the p-value by m and then checking if it is lesser than α

False Discovery Rate:

- Helps to control the fraction of candidates in the smaller set that are really false rejections

	H_0 is True	H_0 is False	Total
Reject H_0	V	S	R
Do Not Reject H_0	U	W	$m - R$
Total	m_0	$m - m_0$	m

It is another counter whereby when m is large, the FWER will be too conservative, and it will rarely reject. Because we will have $\frac{\alpha}{m}$ and when m is large, this value is very small.

$$FDR = E\left(\frac{V}{R}\right) = E\left(\frac{\#\ False\ Rejection}{\#\ Total\ Rejections}\right)$$

V – Number of True H_0 that we falsely reject (False Discoveries)
R – Total Number of Hypothesis that we reject regardless of whether it is actually True/False.

Benjamini-Hochberg Procedure to Control FDR:

(*Example Under Slide 46 of Lecture 12*)

1. Specify q, the level at which to control the FDR.
2. Compute p-values $p_1, \ldots, p_m$ for the null hypotheses $H_{01}, \ldots, H_{0m}$.
3. Order the p-values so that $p_{(1)} \leq p_{(2)} \leq \ldots \leq p_{(m)}$.
4. Define $L = \max\{j : p_{(j)} < qj/m\}$.
5. Reject all null hypotheses H_{0j} for which $p_j \leq p_{(L)}$.

Then, FDR $\leq q$.

Note that for step 2, we will multiply j to the numerator as well indicating the index of the p-value.

We will reject all the hypothesis where they have $p_{(j)} < \frac{qj}{m}$. Noting that they need to be ordered based on their p-values.

Cherry-Picking points:

- Note that we should not do cherry picking, because we will not be adjusting correctly for m of the Bonferroni / BH

(f) Explain why the analysis in (e) is misleading. Hint: The standard approaches for controlling the FWER and FDR assume that all tested null hypotheses are adjusted for multiplicity, and that no "cherry-picking" of the smallest p-values has occurred. What goes wrong if we cherry-pick?

Answer. For FWER with Bonferroni correction, we adjust the p-values by the number m of tests. By cherry-picking the k best managers, the tests with small p-values are adjusted by $k = 10 \ll m$. This will then lead to higher chance of (falsely) rejecting some null hypotheses. Similar arguments apply to FDR.