

Beat the Shit Out of Data Science Interviews

Machine Learning

Qiu-Yue Zhong, ScD

Last Updated in Jan 2019

Contents

1	ML strategy	1
1.1	Why ML strategy	1
1.2	Error Analysis	2
1.2.1	Cleaning up incorrectly labeled data being assigned by human	3
1.2.2	Training and testing on different distributions	3
1.2.3	Bias and Variance with mismatched data distributions	4
1.3	Choose the Optimal Models	7
1.3.1	Regression	7
1.3.1.1	Total Sum of Squares (TSS)	7
1.3.1.2	Explained Sum of Squares	7
1.3.1.3	Residual Sum of Squares (RSS)	7
1.3.1.4	Mean Absolute Error (MAE)	7
1.3.1.5	Mean Squared Error (MSE)	7
1.3.1.6	Root Mean Square Error (RMSE)	8
1.3.1.7	R-Square	8
1.3.1.8	Adjusted R-Square	8
1.3.1.9	Cp	9
1.3.1.10	Akaike Information Criterion (AIC)	9
1.3.1.11	Bayesian Information Criterion (BIC)	9
1.3.1.12	Cross-Validation	9
1.3.1.13	Bootstrap to Estimate Prediction Error	11
1.3.2	Classification	13
1.3.2.1	AUC or c-statistic	13
1.3.2.2	Confusion Matrix	13
1.3.2.3	Accuracy	13
1.3.2.4	Sensitivity/Recall	13
1.3.2.5	Specificity	14
1.3.2.6	Youden's J/Informedness	14
1.3.2.7	Precision/PPV	14
1.3.2.8	NPV	14
1.3.2.9	Markedness/Delta p	14
1.3.2.10	F1	14
1.3.2.11	F-beta	14

1.3.2.12	Matthew's Correlation Coefficient(MCC)	15
1.4	Loss function	17
1.4.1	Regression	17
1.4.1.1	MAE/Mean Bias Error (MBE)/Laplace L1 Loss	17
1.4.1.2	MSE/Squared Loss/Quadratic Loss/Euclidean Loss/Gaussian L2 Loss	18
1.4.1.3	Huber Loss/Smooth Mean Absolute Error	18
1.4.1.4	Quantile Loss Function	19
1.4.2	Classification	19
1.4.2.1	Hinge Loss	19
1.4.2.2	Cross Entropy Loss/Negative Log Loss/Binomial Log-likelihood Loss	20
1.4.2.3	Exponential Loss	21
1.5	Bias-variance Trade-off	23
1.5.1	Avoid Over-fitting	23
1.5.2	Avoid Under-fitting	24
1.6	Feature Standardization, Normalization, and Scaling	26
1.7	Generative Model and Discriminative Model	29
1.8	Missing Value	31
1.8.1	Missing Completely at Random (MCAR)	31
1.8.1.1	Definition	31
1.8.1.2	How to proceed	31
1.8.2	Missing at Random (MAR)	33
1.8.2.1	Definition	33
1.9	Imbalanced Datasets	35
1.9.1	Imbalanced Datasets	35
1.9.2	Performance Measure	35
1.9.3	Re-sampling	35
1.9.3.1	Random Over-sampling/Sampling with Replacement	35
1.9.3.2	Random Under-sampling	35
1.9.3.3	Informed Under-sampling	36
1.9.4	Generate Synthetic Samples	37
1.9.4.1	Synthetic Minority Over-sampling Technique (SMOTE)	37
1.9.5	Cost-Sensitive Learning	38
1.9.5.1	Cost-Sensitive Dataspace Weighting	38
1.10	Model Interpretability	40
1.10.1	Global Feature Importance	40
1.10.1.1	Permutation Importance/Mean Decrease Accuracy (MDA)	40

1.10.1.2	Mean Decrease in Impurity (MDI)	42
1.10.1.3	Partial Dependence Plots	43
1.10.2	Global Surrogate	45
1.10.3	Local Surrogate	47
1.10.3.1	Local Interpretable Model-agnostic Explanations (LIME)	47
1.10.4	Shapley Values	50
1.10.4.1	Summary	50
1.10.4.2	Shapley Value in Game Theory	50
1.10.4.3	Shapley Value in ML Models	51
1.10.4.4	SHAP: SHapley Additive explanations	55
1.11	Bootstrap	57
1.11.1	Bootstrap for Confidence Intervals	59
1.11.1.1	Normal-Theory CIs:	59
1.11.1.2	t-Confidence Interval with Bootstrap Standard Error	60
1.11.1.3	Bootstrap Confidence Intervals via Percentiles	60
1.11.1.4	Bootstrap t-Table Confidence Intervals	61
1.11.1.5	The bias-corrected and Accelerated (BCa) Bootstrap Interval	61
1.11.2	Bootstrap Hypothesis Testing	63
1.11.3	Bootstrapping Regression Models	64
1.11.3.1	Random	64
1.11.3.2	Fixed	64
1.12	Pro and Cons of Algorithms	66
2	Supervised Learning	67
2.1	Linear regression	67
2.1.1	Derivation	67
2.1.2	Assumptions	70
2.1.3	Collinearity	71
2.1.4	Best Linear Unbiased Estimator (BLUE)	72
2.1.5	Pros	72
2.1.6	Cons	72
2.2	Logistic regression	74
2.2.1	Logit and Logistic	74
2.2.2	Maximum Likelihood Estimation	75
2.2.3	Least Squared Error Estimation	75
2.2.4	Newton's Method	76

2.2.5	Pros	76
2.2.6	Cons	76
2.3	GLM	77
2.3.1	Three Components	77
2.3.2	Assumptions	78
2.3.3	Pros	78
2.3.4	Cons	78
2.3.5	GLM vs. Transformations of Response Variable in Linear Regression	78
2.3.6	Generalized Linear Models for Counts	80
2.3.7	Models for Overdispersed Data	81
2.3.7.1	Overdispersion	81
2.3.7.2	Deviance or Likelihood Ratio Statistic	81
2.3.7.3	Alternative Models	81
2.3.8	Loglinear Models for Contingency Tables	83
2.3.9	Common distributions with typical uses and canonical link functions for GLM	84
2.3.10	Logit vs. Probit	85
2.4	GAM	87
2.5	Linear Discriminant Analysis (LDA)	89
2.5.1	Linear Methods for Classification	89
2.5.2	Linear Discriminant Analysis	90
2.5.2.1	Linear Discriminant Analysis as a restricted Gaussian classifier	90
2.5.2.2	Derivations	90
2.5.2.3	Time complexity	91
2.5.2.4	Pros	91
2.5.2.5	Cons	91
2.5.3	LDA vs. PCA	93
2.5.4	LDA vs. LR	94
2.5.5	Quadratic Discriminant functions (QDA)	95
2.6	Naïve Bayes	97
2.6.1	Bayes Rule	97
2.6.2	Conditional Independence Assumption	97
2.6.3	Derivation	97
2.6.4	Properties	98
2.7	SVM	100
2.7.1	Maximal Margin Classifier	100
2.7.1.1	Hyperplane	100

2.7.1.2	Margin	100
2.7.1.3	Derivation	101
2.7.1.4	Loss Function View	101
2.7.1.5	Support Vectors	102
2.7.1.6	Cons	102
2.7.2	Support Vector Classifier/Soft Margin Classifier	103
2.7.2.1	Why SVC	103
2.7.2.2	Soft Margin SVM: Geometric View	104
2.7.2.3	Tuning parameter C	104
2.7.2.4	Soft Margin SVM: Loss Function View	105
2.7.3	Support Vector Machine	106
2.7.3.1	Kernel	106
2.7.3.2	Why using kernel rather than enlarging feature space using functions of the original features	106
2.7.3.3	Pros	107
2.7.3.4	Cons	107
2.8	kNN	109
2.8.1	Summary	109
2.8.2	Pros	109
2.8.3	Cons	110
2.9	Classification and Regression Tree (CART)	111
2.9.1	Pros	111
2.9.2	Cons	111
2.9.3	Trees vs. Linear Models	111
2.9.4	CART: Regression Trees	113
2.9.4.1	Greedy Algorithm: Recursive Binary Splitting	113
2.9.4.2	Tree Size: Cost-complexity Pruning	114
2.9.5	CART: Classification Trees	116
2.9.5.1	Measures of Node Impurity	116
2.9.5.2	Loss Matrix	117
2.9.5.3	Missing Predictor Values	117
2.10	MARS: Multivariate Adaptive Regression Splines	119
3	Ensemble	120
3.1	Bagging /Bootstrap Aggregation	121
3.2	Random Forest	122

3.2.1	Variance Reduction	122
3.2.2	Algorithm	122
3.2.3	OOB (Out-of-Bag) Estimate of Error Rate	123
3.2.3.1	Steps	123
3.2.3.2	The Forest Error Rate Depends on Two Things	123
3.2.4	Variable Importance Measures	124
3.2.4.1	Method 1	124
3.2.4.2	Method 2: Measure the Prediction Strength of Each Variable?	125
3.2.5	Proximities?	125
3.2.6	Parameter Tuning	125
3.2.7	Pros	126
3.2.8	Cons	126
3.3	Boosting	127
3.3.1	Three Tuning Parameters	127
3.4	AdaBoost	129
3.4.1	Summary	129
3.4.2	Algorithm	130
3.4.3	Why Exponential Loss	130
3.5	Boosting Trees	131
3.5.1	A Single Tree as Base Learner	131
3.5.1.1	Why is tree an excellent base learner for boosting	131
3.5.2	Boosting Trees	131
3.5.3	Optimization via Gradient Descent	132
3.5.4	Algorithms	132
3.5.5	Parameters Tuning	132
3.5.6	Interpretations	134
3.5.6.1	Relative Importance of Predictor Variables	134
3.6	Gradient Boosting Machines: GBM	135
3.6.1	Summary	135
3.6.2	Algorithm	135
3.6.3	Pros	136
3.6.4	Cons	136
3.7	RF vs. GBM	138
3.8	AdaBoost vs. GBM	139
3.9	XGBoost: eXtreme Gradient Boosting	140
3.9.1	Pros	140

3.9.2	Parameters Tuning	141
3.10	LightGBM	143
3.10.1	Gradient-based One-Side Sampling (GOSS)	143
3.10.2	Exclusive Feature Bundling	143
3.10.3	Optimization in Accuracy: Leaf-wise (Best-first) Tree Growth	144
4	Subset Selection and Shrinkage Methods	146
4.1	Subset selection	146
4.1.1	Best subset selection	146
4.1.2	Stepwise selection	147
4.1.2.1	Forward	147
4.1.2.2	Backwards	147
4.1.2.3	Hybrid Approaches	147
4.2	Regularization	148
4.2.1	RIDGE	148
4.2.1.1	Summary	148
4.2.1.2	lambda	148
4.2.1.3	Applications of RIDGE	149
4.2.1.4	Scaling	149
4.2.2	LASSO	150
4.2.2.1	L1 sparsity	150
4.2.2.2	Computation of the Lasso solutions	150
4.2.2.3	Oracle property?Unbiased?	151
4.2.2.4	Pros	151
4.2.2.5	Cons	152
4.2.2.6	LASSO, RIDGE and Best Selection: Bayes Estimates with Different Priors	152
4.2.3	Least Angle Regression	153
4.2.4	Principal Components Regression	154
4.2.5	Partial Least Squares	155
5	Unsupervised Learning	157
5.1	Dimension Reduction	157
5.1.1	PCA	157
5.1.1.1	First Principal Component	157
5.1.1.2	Second principal component	157
5.1.2	ICA	159
5.1.3	Singular Valued Decomposition	160

5.2	Clustering Algorithms	161
5.2.1	Proximity Matrices	161
5.2.2	k-Means Clustering	163
5.2.2.1	Algorithm	163
5.2.2.2	Properties	163
5.2.2.3	How to choose number of clusters	164
5.2.2.4	Choosing the initial centroids	165
5.2.2.5	Space complexity	166
5.2.2.6	Time complexity	166
5.2.2.7	Why Euclidean not Manhattan distance	166
5.2.2.8	Pros	166
5.2.2.9	Cons	167
5.2.2.10	k-means: a Variant of EM algorithm	167
5.2.3	k-medoids Clustering	169
5.2.4	Agglomerative Hierarchical Clustering	170
5.2.4.1	Algorithm	170
5.2.4.2	Properties	170
5.2.4.3	Dendrogram	170
5.2.4.4	Dissimilarity	171
5.2.4.5	Time complexity	172
5.2.4.6	Space complexity	172
5.2.4.7	Pros	172
5.2.4.8	Cons	172
5.2.5	DBSCAN	174
5.2.6	Expectation-Maximization Algorithm	175
5.2.6.1	Algorithm	175
5.2.6.2	Derivation	176
5.2.6.3	Pros	177
5.2.6.4	Cons	177
5.3	t-Distributed Stochastic Neighbor Embedding	179
5.3.1	Algorithm	179

1 ML strategy

1.1 Why ML strategy

You have a lot of ideas for how to improve the accuracy of your deep learning system:

- Collect more data
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try different optimization algorithm
- Add L_2 regularization

ML strategy will give you some strategies to help analyze your problem to go in a direction that will help you get better results.

For a supervised learning system to do well, you usually need to tune the knobs of your system to make sure that four things hold true - chain of assumptions in machine learning:

1. You'll have to fit training set well on cost function (near human level performance if possible).
 - If it's not achieved you could try bigger network, another optimization algorithm
2. Fit dev set well on cost function.
 - If its not achieved you could try regularization, bigger training set
3. Fit test set well on cost function.
 - If its not achieved you could try bigger dev set
4. Performs well in real world.
 - If its not achieved you could try change dev set, change cost function

Single number evaluation metric

Its better and faster to set a single number evaluation metric for your project before you start it.

Satisfying and Optimizing metric

It's hard sometimes to get a single number evaluation metric. So we can solve that by choosing a single optimizing metric and decide that other metrics are satisfying.

Maximize one optimizing metric
subject to $N - 1$ satisfying metrics

1.2 Error Analysis

- **Bayes optimal error:** An error level you won't surpass.
 - You can't do better than Bayes error unless you are over-fitting.
- **Human level error**
 - After an algorithm reaches the human level performance the progress and accuracy slow down
 - There isn't much error range between human-level error and Bayes optimal error
 - The human-level error as a proxy (estimate) for Bayes optimal error. Bayes optimal error is always less (better), but human-level in most cases is not far from it
 - Improving deep learning algorithms is harder once you reach a human-level performance.
- **Training error**
- **Dev error**
- **Avoidable bias:** Training error - Human (Bayes) error
 - If avoidable bias difference is the bigger, then it's **bias** problem and you should use a strategy for bias resolving.
 - * Train bigger model.
 - * Train longer/better optimization algorithm (like Momentum, RMSprop, Adam).
 - * Find better architecture/hyperparameters search.
- **Variance:** Dev error - Training error
 - If variance difference is bigger, then you should use a strategy for variance resolving.
 - * Get more training data.
 - * Regularization (L_2 , Dropout, data augmentation).
 - * Find better NN architecture/hyper-parameters search.

Error analysis:

Process of manually examining mistakes that your algorithm is making. It can give you insights into what

to do next. Error analysis helps you to analyze the error before taking an action that could take lot of time with no need.

1.2.1 Cleaning up incorrectly labeled data being assigned by human

Training set:

DL algorithms are quite robust to *random errors* in the training set but less robust to *systematic errors*. But it's OK to go and fix these labels if you can.

Dev/test sets:

Consider these guidelines while correcting the dev/test mislabeled examples:

- Apply the same process to your **dev** and **test** sets to make sure they continue to come from the same distribution.
- Consider examining examples your algorithm got right as well as ones it got wrong.
- It's very important to have dev and test sets to come from the same distribution. But it could be OK for a training set to come from slightly other distribution.

(training set has much larger sample size)

1.2.2 Training and testing on different distributions

Strategies to follow up when training set distribution differs from dev/test sets distribution:

- Option one (not recommended): shuffle all the data together and extract randomly training and dev/test sets.
 - Advantages: all the sets now come from the same distribution.
 - Disadvantages: the new dev/test sets now do not come from the distribution you care about.
- Option two: take some of the dev/test set examples and add them to the training set.
 - Advantages: the distribution you care about is your target now.
 - Disadvantage: the distributions in training and dev/test sets are now different. But you will get a better performance over a long time.

Addressing data mismatch:

There aren't completely systematic solutions to this, but there some things you could try.

- Carry out manual error analysis to understand the difference between training and dev/test sets

- Collect more data similar to dev/test sets
- Make training data more similar to dev/test sets;
 - Artificial data synthesis
 - * Combine some of your training data with something that can convert it to the dev/test set distribution.
 - You do not lots of audios recorded inside the car with background noise of a car (dev/test). You have recorded a large amount of clean audio without car background noise (training). Combine normal audio with car noise to get audio with car noise example.
 - * Be cautious and bear in mind whether or not you might be accidentally simulating data only from a tiny subset of the space of all possible examples because your NN might overfit these generated data (like particular car noise or a particular design of 3D graphics cars).

1.2.3 Bias and Variance with mismatched data distributions

- Example 1:
 - Human error: 0%
 - Train error: 1%
 - Dev error: 10%

In this example, you'll think that this is a variance problem, but because the distributions aren't the same you can't tell for sure. Because it could be that train set was easy to train on, but the dev set was more difficult.

To solve this issue we create a new set called *train-dev set* as a random subset of the training set (so it has the same distribution)

Then you get:

- Human error: 0%
- Train error: 1%
- Train-dev error: 9%
- Dev error: 10%

Now we are sure that this is a *high variance* problem.

- Example 2:

- Human error: 0%
- Train error: 1%
- Train-dev error: 1.5%
- Dev error: 10%

In this case we have something called *Data mismatch* problem.

Summary:

1. Human-level error (proxy for Bayes error)

2. Train error

- Calculate

$$\text{avoidable bias} = \text{training error} - \text{human level error}$$

- If the difference is big then its avoidable bias problem then you should use a strategy for high bias.

3. Train-dev error

- Calculate

$$\text{variance} = \text{training-dev error} - \text{training error}$$

- If the difference is big then its high variance problem then you should use a strategy for solving it.

4. Dev error

- Calculate

$$\text{data mismatch} = \text{dev error} - \text{training-dev error}$$

- If difference is much bigger than train-dev error its Data mismatch problem. Unfortunately, there aren't many systematic ways to deal with data mismatch

5. Test error

- Calculate

$$\text{degree of overfitting to dev set} = \text{test error} - \text{dev error}$$

- Is the difference is big (positive) then maybe you need to find a bigger dev set (dev set and test set come from the same distribution, so the only way for there to be a huge gap here, for it to do much better on the dev set than the test set, is if you somehow managed to overfit the dev set).

References

<https://www.coursera.org/learn/machine-learning-projects?specialization=deep-learning>

1.3 Choose the Optimal Models

1.3.1 Regression

1.3.1.1 Total Sum of Squares (TSS)

$$SS_{\text{tot}} = \sum_{i=1}^m (y_i - \bar{y})^2$$

1.3.1.2 Explained Sum of Squares

$$SS_{\text{reg}} = \sum_{i=1}^m (\hat{y}_i - \bar{y})^2$$

1.3.1.3 Residual Sum of Squares (RSS)

$$SS_{\text{res}} = \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

1.3.1.4 Mean Absolute Error (MAE)

- $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$
- L_1
- Pros:
 - Conceptually simpler and more interpretable than RMSE
 - Each error contributes to MAE in proportion to the absolute value of the error
- Cons:
 - Can not tell the direction of errors (??? RMSE can't tell too)

1.3.1.5 Mean Squared Error (MSE)

- $\frac{RSS}{n} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
 - RSS: residual sum of squares
 - n : degrees of freedom

-

$$\begin{aligned}\text{MSE}(x_0) &= \text{E}_{\mathcal{T}} [f(x_0) - \hat{y}_0]^2 \\ &= \text{E}_{\mathcal{T}} [\hat{y}_0 - \text{E}_{\mathcal{T}}(\hat{y}_0)]^2 + [\text{E}_{\mathcal{T}}(\hat{y}_0) - f(x_0)]^2 \\ &= \text{Var}_{\mathcal{T}}(\hat{y}_0) + \text{Bias}^2(\hat{y}_0)\end{aligned}$$

- L_2

- Training set MSE is generally an **underestimate** of the test MSE

- Training set RSS cannot be used to select from among a set of models with different numbers of variables.

- Pros:

- Increase the importance of larger errors

- Cons:

- Hard to interpret (compare to RMSE and MAE)

1.3.1.6 Root Mean Square Error (RMSE)

- $\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$

- The square root of the average squared distance between the actual score and the predicted score

- The RMSE is thus the distance, on average, of a data point from the fitted line, measured along a **vertical line**

- Has the same units as the quantity plotted on the vertical axis (more interpretable than MSE)

1.3.1.7 R-Square

- $1 - \frac{RSS}{TSS}$

- Larger, better

- Training set R^2 cannot be used to select from among a set of models with different numbers of variables

1.3.1.8 Adjusted R-Square

- $1 - \frac{RSS/(n-p-1)}{TSS/(n-1)}$

- Larger, better

- A large value of adjusted R^2 indicates a model with a small test error

- Model with the largest adjusted R^2 will have only correct variables and no noise variables.
- Unlike the R^2 statistic, the adjusted R^2 statistic pays a price for the inclusion of unnecessary variables in the model

1.3.1.9 Cp

- $C_P = \frac{1}{n}(RSS + 2p\hat{\sigma}^2)$
 - p : number of predictors in the model
 - $\hat{\sigma}^2$: an estimate of the variance of the error ϵ associated with each response measurement using the full model containing all predictors
- Lower, better
- If $\hat{\sigma}^2$ is an unbiased estimate of σ^2 , then C_P is an unbiased estimate of test MSE
- Add a penalty of $2p\hat{\sigma}^2$ to the training RSS to adjust for the fact that the training error tends to underestimate the test error
- Penalty increases as the number of predictors in the model increases: adjust for the corresponding decrease in training RSS

1.3.1.10 Akaike Information Criterion (AIC)

- Lower, better

$$AIC = n + n \log 2\pi + n \log(RSS/n) + 2(p + 1)$$

1.3.1.11 Bayesian Information Criterion (BIC)

- Lower, better

$$BIC = n + n \log 2\pi + n \log(RSS/n) + (\log n)(p + 1)$$

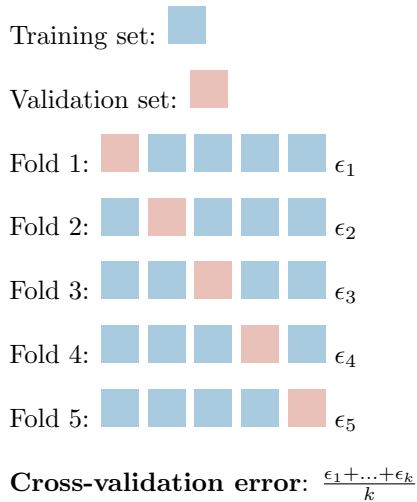
1.3.1.12 Cross-Validation

- Directly estimates the **expected extra-sample error**
 - $\text{Err} = E[L(Y, f(X))]$
 - Average generalization error when the method $\hat{f}(X)$ is applied to an **independent test sample** from the joint distribution of X and Y

- Cross-validation only estimates effectively the **average error**
- **One-standard error rule:** choose the most parsimonious model whose error is no more than one standard error above the error of the best model
- With a multistep modeling procedure, cross-validation must be applied to the **entire sequence of modeling steps**. In particular, samples must be left out before any selection or filtering steps are applied.

1. k -fold cross-validation

- With $K = 5$ say, cross-validation has **lower variance**
- **Bias** could be a problem, depending on how the performance of the learning method varies with the size of the training set (**training-set-size bias**)
 - If the learning curve has a considerable slope at the given training set size, five- or tenfold cross-validation will **overestimate the true prediction error**



2. Leave-one-out Validation

- when $k = N$
 - Train the model on every point except for one point
 - Compute the test error on the held out point
- Approximately **unbiased** for the true (expected) prediction error
- **High variance** because the N training sets are so similar to one another
- Considerable **computational burden**: requiring N applications of the learning method

1.3.1.13 Bootstrap to Estimate Prediction Error

- **Simple Bootstrap Cross-Validation:**

- Fit the model on a set of bootstrap samples (as the training set) and then keep track of how well it predicts the original training set (as the validation set)

$$- \widehat{\text{Err}}_{\text{boot}} = \frac{1}{B} \frac{1}{N} \sum_{b=1}^B \sum_{i=1}^N L(y_i, \hat{f}^{*b}(x_i))$$

- Problems:

- * Overlap between training and validation set: about 0.632 of the original data points appear in each bootstrap sample

$$\Pr\{\text{observation } i \in \text{bootstrap sample } b\} = 1 - \left(1 - \frac{1}{N}\right)^N$$

$$\begin{aligned} * \quad & \approx 1 - e^{-1} \\ & = 0.632 \end{aligned}$$

- * $\widehat{\text{Err}}_{\text{boot}}$ seriously **underestimates** (only 0.632 of the true prediction error, **overfitting**) the true prediction error

- **Leave-one-out bootstrap**

- For each observation, only keep track of predictions from bootstrap samples **not containing that observation**

$$- \widehat{\text{Err}}^{(1)} = \frac{1}{N} \sum_{i=1}^N \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} L(y_i, \hat{f}^{*b}(x_i))$$

- * C^{-i} is the set of indices of the bootstrap samples b that do not contain observation i

- * $|C^{-i}|$ is the number of such samples

- Problems:

- * Solves the overfitting problem suffered by $\widehat{\text{Err}}_{\text{boot}}$, but has the **training-set-size bias** like the cross-validation

- * The average number of distinct observations in each bootstrap sample is about $0.632 \times N \implies$ Roughly behave like that of twofold cross-validation \implies **Overestimate (bias upward)** the true error, if the learning curve has considerable slope at sample size $N/2$

- **0.632 estimator**

$$- \widehat{\text{Err}}^{(0.632)} = 0.368 \cdot \overline{\text{err}} + 0.632 \cdot \widehat{\text{Err}}^{(1)}$$

- Work well in light fitting situations, but can break down in overfit ones.

- **0.632+ estimator**

- Take into account the amount of overfitting using the no-information error rate and relative overfitting rate \hat{R}

–

$$\widehat{\text{Err}}^{(0.632+)} = (1 - \hat{w}) \cdot \overline{\text{err}} + \hat{w} \cdot \widehat{\text{Err}}^{(1)}$$

$$\text{with } \hat{w} = \frac{0.632}{1 - 0.368\hat{R}}$$

1.3.2 Classification

1.3.2.1 AUC or c-statistic

- The Receiver Operating Characteristic (ROC) curve is a standard technique for summarizing classifier performance over a range of tradeoffs between true positive and false positive error rates.
 - For each candidate threshold, the resulting true-positive rate (i.e., the sensitivity, Y -axis) and the false-positive rate ($1 - \text{specificity}$, X -axis) are plotted against each other
- The optimal model should be shifted towards the **upper left corner** of the plot
- Independent of the decision criterion selected and prior probabilities

1.3.2.2 Confusion Matrix

Table 1: Confusion Matrix

	Actual: positive	Actual: negative
Prediction: positive	True Positive	False Positive, Type I error
Prediction: negative	False Negative, Type II error	True Negative

1.3.2.3 Accuracy

- The fraction of predictions that are true
- Easy to interpret
- A good measure when the target variable classes in the data are nearly balanced
- High accuracy does not necessarily characterize a good classifier

$$\frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{False Negative} + \text{True Negative}}$$

1.3.2.4 Sensitivity/Recall

- Proportion of known positives that are predicted correctly

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

1.3.2.5 Specificity

- The fraction of actual negatives that are correctly predicted

$$\frac{\text{blue square}}{\text{blue square} + \text{light blue square}}$$

1.3.2.6 Youden's J/Informedness

$$J = \Delta p' = \text{sensitivity} + \text{specificity} - 1$$

1.3.2.7 Precision/PPV

- The proportion of predicted positives that are correct
- Sensitive to data distributions

$$\frac{\text{yellow square}}{\text{yellow square} + \text{light blue square}}$$

1.3.2.8 NPV

$$\frac{\text{blue square}}{\text{blue square} + \text{orange square}}$$

1.3.2.9 Markedness/Delta p

$$MK = \Delta p = PPV + NPV - 1$$

1.3.2.10 F1

- Harmonic mean of the PPV and sensitivity

$$F_1 = 2 \times \frac{PPV * Sensitivity}{PPV + Sensitivity}$$

1.3.2.11 F-beta

- As β decreases, precision is given greater weight
- With $\beta = 1$, we have the commonly used F_1 score

$$F_\beta = \frac{(1 + \beta^2)(\text{Precision} \times \text{Recall})}{\beta^2 \times \text{Precision} + \text{Recall}}$$

1.3.2.12 Matthew's Correlation Coefficient(MCC)

- -1: when the classification is always wrong
- 0: when it is no better than random
- 1: when it is always correct

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

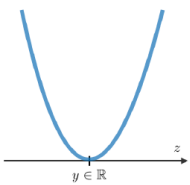
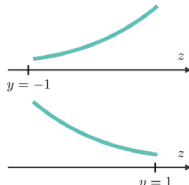
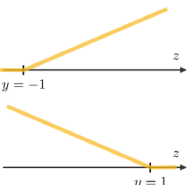
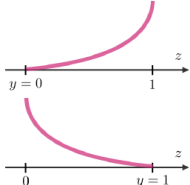
References

Lever, J., Krzywinski, M. and Altman, N., 2016. Classification evaluation. *Nature methods*, 13, p.603.

1.4 Loss function

A loss function is a function $L/\Psi : (z, y) \in \mathbb{R} \times Y \mapsto L(z, y) \in \mathbb{R}$ that takes as inputs the predicted value $z/f(x)$ corresponding to the real data value y and outputs how different they are.

$$\hat{f}(x) = \underset{f(x)}{\operatorname{argmin}} \Psi(y, f(x))$$

Least squared	Logistic	Hinge	Cross-entropy
$\frac{1}{2}(y - z)^2$	$\log(1 + \exp(-yz))$	$\max(0, 1 - yz)$	$-[y \log(z) + (1 - y) \log(1 - z)]$
			
Linear regression	Logistic regression	SVM	Neural Network

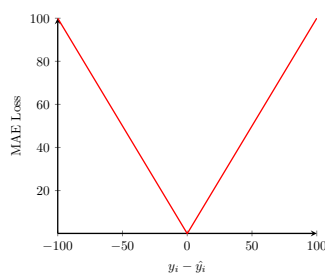
1.4.1 Regression

Continuous $y, y \in \mathbb{R}$

1.4.1.1 MAE/Mean Bias Error (MBE)/Laplace L1 Loss

$$\Psi(y, f) = |y - f|$$

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$



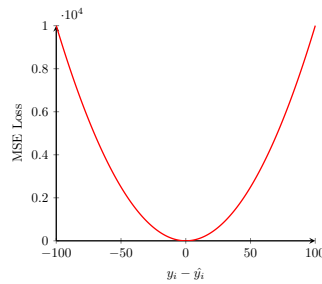
- More robust to outliers
 - Useful when the training data is corrupted with outliers (training data has unrealistically huge negative/positive values)

- Its derivatives are not continuous (not differentiable), making it inefficient to find the solution.
 - The gradient is the same throughout, which means the gradient will be large even for small loss values.

1.4.1.2 MSE/Squared Loss/Quadratic Loss/Euclidean Loss/Gaussian L2 Loss

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

$$\Psi(y, f) = \frac{1}{2}(y - f)^2$$



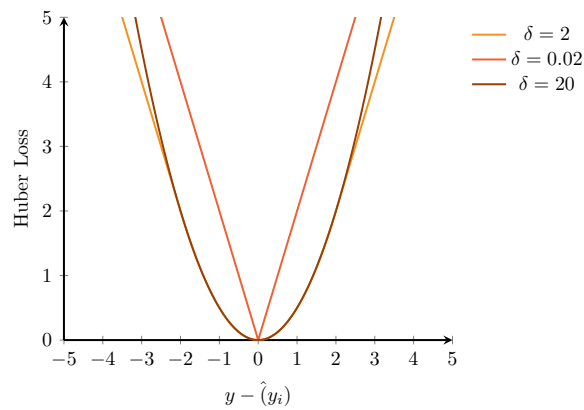
- Used in **linear regression**
- Sensitive to outliers
 - Far less robust
 - Performance severely degrades for long-tailed error distributions and especially for grossly mis-measured y -values (outliers)
- Easier to solve than MAE
 - Gives a more stable and closed form solution (by setting its derivative to 0)
 - The gradient of MSE loss is high for larger loss values and decreases as loss approaches 0.
- Why not use in classification
 - Not a monotone decreasing function of increasing margin $yf(x)$

1.4.1.3 Huber Loss/Smooth Mean Absolute Error

$$\Psi(y, f)_{\text{Huber}, \delta} = \begin{cases} \frac{1}{2}(y - f)^2, & \text{if } |y - f| \leq \delta \\ \delta|y - f| - \frac{\delta^2}{2}, & \text{if } |y - f| > \delta \end{cases}$$

$$l_{\delta}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

- Less sensitive to outliers than the squared error loss
- Differentiable at 0
- Hyperparameter δ :
 - Determines what to consider as an outlier
 - Specify the robustification effect of the loss-function
 - Residuals larger than δ are minimized with L_1 (which is less sensitive to large outliers)
 - Residuals smaller than δ are minimized “appropriately” with L_2



1.4.1.4 Quantile Loss Function

$$\Psi(y, f)_{\text{Quantile}, \alpha} = \begin{cases} (1 - \alpha)|y - f|, & \text{if } y - f \leq 0 \\ \alpha|y - f|, & \text{if } y - f > 0 \end{cases}$$

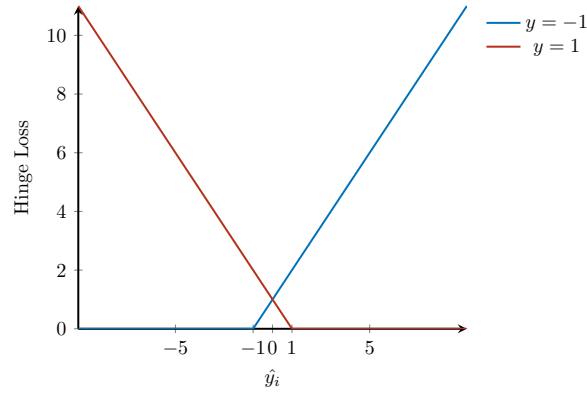
- α specified
- Distribution free and in general proves to provide good robustness to outliers

1.4.2 Classification

1.4.2.1 Hinge Loss

$$\max(0, 1 - y_i * \hat{y}_i), y_i \in \{1, -1\}$$

- Used in **SVM**
- Convex but not differentiable, has a subgradient



1.4.2.2 Cross Entropy Loss/Negative Log Loss/Binomial Log-likelihood Loss

$$l(Y, p(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x))$$

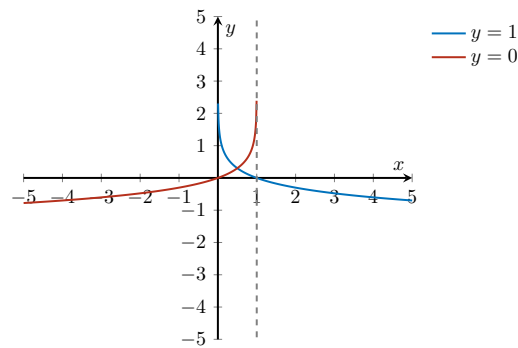
- Deviance:

$$-l(Y, f(x)) = \log(1 + e^{-2Y f(x)})$$

$$Y' \in \{1, 0\}, Y \in \{1, -1\}$$

- Convex
- Monotone continuous approximations to misclassification loss
- The penalty associated with binomial deviance increases linearly for large increasingly negative margin
 - More robust in noisy settings where the Bayes error rate is not close to zero, and especially in situations where there is **misspecification of the class labels** in the training data
- Used for **logistic regression**

$$-\sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)), y_i \in \{1, 0\}$$



- Used for softmax ($f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$) with $K > 2$ class

$$-\sum_{i=1}^N \sum_{j=1}^K y_{ij} \log(\hat{y}_{ij})$$

- K -class multinomial deviance loss function:

$$\begin{aligned} L(y, p(x)) &= -\sum_{k=1}^K I(y = \mathcal{G}_k) \log p_k(x) \\ &= -\sum_{k=1}^K I(y = \mathcal{G}_k) f_k(x) + \log \left(\sum_{\ell=1}^K e^{f_{\ell}(x)} \right) \end{aligned}$$

1.4.2.3 Exponential Loss

$$\Psi(y, f) = \exp(-yf)$$

- Used in [Adaboost](#)
- Monotone continuous approximations to misclassification loss
- The penalty associated with exponential criterion increases the influence of such observations exponentially

References

<http://rohanvarma.me/Loss-Functions/>

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

<http://cs231n.github.io/linear-classify/#svm>

<https://github.com/afshinea/stanford-cs-229-machine-learning/blob/master/en/cheatsheet-supervised-learning.pdf>

https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Hinge_loss.html

<https://peterroelants.github.io/posts/cross-entropy-softmax/>

<https://github.com/afshinea/stanford-cs-229-machine-learning/blob/master/en/cheatsheet-supervised-learning.pdf>

1.5 Bias-variance Trade-off

Models with a lower bias in parameter estimation have a higher variance of the parameter estimates across samples, and vice versa.

$$\text{Err}(x) = \text{Bias}^2 + \text{Variance} + \text{IrreducibleError}$$

- **Bias:** the difference between the prediction and the true value
 - High bias \implies Under-fitting
- **Variance:** how much the predictions for a given point vary between different realizations of the model
 - High variance \implies Over-fitting
- **Irreducible Error:** noise term in the true relationship that cannot fundamentally be reduced by any model

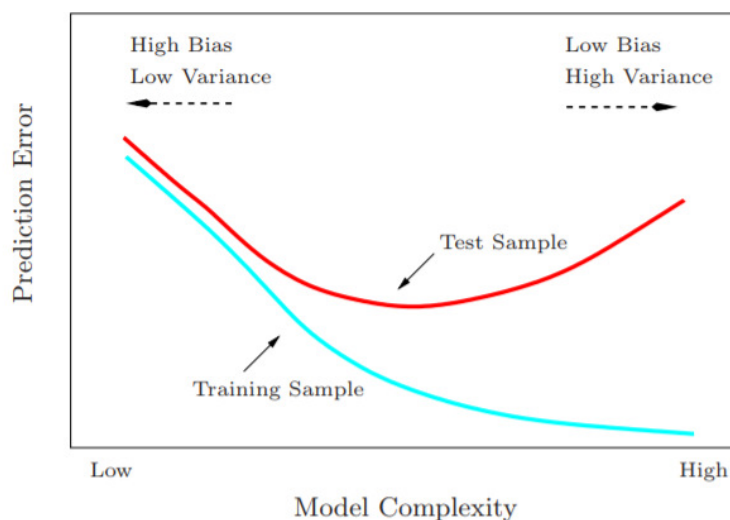


FIGURE 2.11. Test and training error as a function of model complexity.

1.5.1 Avoid Over-fitting

- Simpler model/Remove features
- Cross-validation
- Train with more data
- Regularization
- Ensemble learning

- Early stopping
- For Deep Learning: Dropout and Dropconnect

1.5.2 Avoid Under-fitting

- More data
- Add new domain-specific features
- Change the types of feature processing used
- Decrease the amount of regularization used

References

<http://scott.fortmann-roe.com/docs/BiasVariance.html>

1.6 Feature Standardization, Normalization, and Scaling

- Data **normalization** is the process of rescaling one or more attributes to the **range of 0 to 1**.
 - Good to use when you do **not** know the distribution of your data or when you know the distribution is not Gaussian
- Data **standardization** is the process of rescaling one or more attributes so that they have **a mean value of 0 and a standard deviation of 1**
 - Assumes that data has a Gaussian distribution
 - More effective if distribution is Gaussian

Require scaling:

- Distance based algorithms
 - k-means
 - k-NN
- Algorithms using gradient descent/ascent-based optimization
 - Linear/non-linear regressions (when regularized), logistic regression (when regularized), Neural Networks, and SVM
 - Help in faster convergence/find support vector

Require normalization:

- Algorithms used for matrix factorization, decomposition or dimensionality reduction
 - PCA, SVD, Factorization Machines

Not require normalization/scaling

- Algorithms rely on rules
 - CART, Random Forests, Gradient Boosted Decision Trees
- Algorithms rely on distributions of the variables
 - Naive Bayes

Methods

- Standardization:

$$x' = \frac{x - \bar{x}}{\sigma}$$

- This redistributes the features with their mean $\mu = 0$ and standard deviation $\sigma = 1$

- Min-Max Scaling:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- This scaling brings the value between 0 and 1.

- Unit Vector:

$$x' = \frac{x}{\|x\|}$$

- Scaling is done considering the whole feature vectored to be of unit length.
 - This scaling brings the value between 0 and 1.

- Mean Normalization:

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

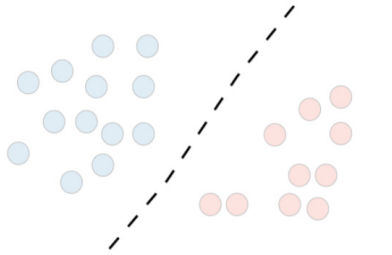
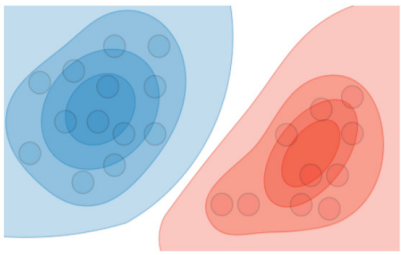
- This distribution will have values between -1 and 1 with $\mu = 0$.

References

http://sebastianraschka.com/Articles/2014_about_feature_scaling.html

<https://www.dataschool.io/comparing-supervised-learning-algorithms/>

1.7 Generative Model and Discriminative Model

	Discriminative model	Generative model
Goal	Directly estimate $P(y x)$	Estimate $P(x y)$ to deduce $P(y x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

- **Discriminative classifiers** directly predict the class labels **without explicitly describing the distribution** of every class label.

$$P(y|x)$$

- Decision trees, rule-based classifier, kNN, artificial neural networks, and SVM
 - Solve a simpler problem than generative models since they do not have the onus of deriving insights about the generative mechanism of data instances
 - Sometimes preferred over generative models, especially when it is not crucial to obtain information about the properties of every class
- Classifiers that learn a generative model of every class in the process of predicting class labels are known as **generative classifiers**

$$P(y, x)$$

- Naïve Bayes classifier and Bayesian networks

References

<https://github.com/afshinea/stanford-cs-229-machine-learning/blob/master/en/super-cheatsheet-machine-learning.pdf>

1.8 Missing Value

1.8.1 Missing Completely at Random (MCAR)

1.8.1.1 Definition

If the missing distribution of R doesn't depend on the observed or missing data

- MCAR is a stronger assumption than MAR: most imputation methods rely on MCAR for their validity.

1.8.1.2 How to proceed

- Discard observations with any missing values
 - Can be used if the relative amount of missing data is small, but otherwise should be avoided
- Rely on the learning algorithm to deal with missing values in its training phase
 - CART is one learning algorithm that deals effectively with missing values, through **surrogate splits**
 - MARS and PRIM use similar approaches
 - In GAM, all observations missing for a given input feature are omitted when the partial residuals are smoothed against that feature in the backfitting algorithm, and their fitted values are set to zero. Since the fitted curves have mean zero (when the model includes an intercept), this amounts to assigning the average fitted value to the missing observations
- Impute all missing values before training
 - Impute the missing value with the **mean or median** of the nonmissing values for that feature
 - If the features have at least some moderate degree of dependence
 - * Estimate a predictive model for each feature given the other features and then imputing each missing value by its prediction from the model
 - * Choose the learning method for imputation of the features
 - A flexible, adaptive method will often be preferred
 - If there are many missing feature values in the training set, the learning method must itself be able to deal with missing feature values
 - CART is an ideal choice
 - Cons

- * Imputation will itself introduce additional uncertainty into estimates and predictions from the response model.
- * Measure this additional uncertainty:
 - Multiple imputations and hence creating many different training sets The predictive model for \mathbf{y} can be fit to each training set, and the variation across training sets can be assessed

1.8.2 Missing at Random (MAR)

1.8.2.1 Definition

If the mechanism resulting in its omission is independent of its (unobserved) value

References:

<https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4>

1.9 Imbalanced Datasets

1.9.1 Imbalanced Datasets

Imbalanced data typically refers to classification tasks where the classes are not represented equally.

1.9.2 Performance Measure

- Not appropriate
 - Accuracy
- Appropriate
 - F_1

1.9.3 Re-sampling

- Either increasing the samples of the minority class or decreasing the samples of the majority class
- Obtain a fair balance in the number of instances for both the classes

1.9.3.1 Random Over-sampling/Sampling with Replacement

- Replicate some points from the minority class in order to increase its cardinality
- Pros
 - No information loss
- Cons
 - Increases the likelihood of **overfitting** since it replicates the minority class events

1.9.3.2 Random Under-sampling

- Sampling from the majority class in order to keep only a part of these points
- Pros
 - Improve the runtime of the model and solve the memory problems by reducing the number of training data samples when the training data set is enormous
- Cons

- Discard useful information about the data itself which could be necessary for building rule-based classifiers such as Random Forests
- Removing examples from the majority class may cause the classifier to miss important concepts pertaining to the majority class
- A biased sample: not an accurate representation of the population in that case
- Perform poorly on real unseen data

1.9.3.3 Informed Under-sampling

- To overcome the deficiency of information loss introduced in the traditional random undersampling method
- EasyEnsemble
 - **Unsupervised strategy** to explore the **majority class** using **independent random sampling with replacement**
 - An **ensemble** learning system by independently sampling several subsets from the majority class \mathcal{N} and developing multiple classifiers based on the combination of each subset with the minority class data \mathcal{P}
 - Steps:
 - * Independently sample several subsets $\mathcal{N}_1, \mathcal{N}_2 \dots, \mathcal{N}_T$ from \mathcal{N} with the **same sample size** of the minority class $\mathcal{P} \implies$ balanced class ($|\mathcal{N}_i| = |\mathcal{P}|$)
 - * For each subset $\mathcal{N}_i (1 \leq i \leq T)$, a classifier H_i is trained using \mathcal{N}_i and \mathcal{P}
 - * All generated classifiers are then combined for the final decision
- BalanceCascade
 - A **supervised learning** approach that develops an **ensemble** of classifiers to systematically select which **majority class examples to undersample**
 - **Sequential dependency between classifiers**: to reduce the redundant information in the major class
 - Steps
 - * After H_1 is trained, if an example $x^* \in \mathcal{N}$ is classified correctly as major class by $H_1 \implies x^*$ is somewhat redundant in \mathcal{N} given that already have H_1
 - * Remove part of the correctly classified **major class examples** from \mathcal{N}

- * \mathcal{N} is shrunk after every node H_i is trained, and every H_i deals with a balanced sub-problem ($|\mathcal{N}_i| = |\mathcal{P}|$)
- * $H(x)$ predicts positive if and only if every $H_i(x)(i = 1, 2, \dots, T)$ predicts positive
- K-nearest neighbor (KNN) classifier
 - NearMiss-1 ????
 - * Select those majority examples whose average distance to the three closest minority class examples is the smallest
 - NearMiss-3????
 - * Select a given number of the closest majority examples for each minority example to guarantee that every minority example is surrounded by some majority examples

1.9.4 Generate Synthetic Samples

1.9.4.1 Synthetic Minority Over-sampling Technique (SMOTE)

- Form new minority class examples by interpolating between several minority class examples that lie together
- The **minority class** is **over-sampled** by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors
- For a given observation x_i , a new (synthetic) observation is generated by interpolating between one of the k nearest neighbors, x_{zi}

$$x_{new} = x_i + \lambda (x_{zi} - x_i)$$

- $\lambda \in [0, 1]$
- Pros
 - Alleviates overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances
 - No loss of information
 - Simple to implement and interpret
- Cons
 - Not very practical for high dimensional data

- Not take into consideration neighboring examples can be from other classes. This can increase the overlapping of classes and can introduce additional noise.

1.9.5 Cost-Sensitive Learning

- Use different cost matrices that describe the costs for misclassifying any particular data example
- Cost matrix
 - A numerical representation of the penalty of classifying examples from one class to another
 - $C(Min, Maj)$: the cost of misclassifying a majority class example as a minority class example
 - $C(Maj, Min)$: the cost of misclassifying a minority class example as a majority class example
 - Cost of misclassifying minority examples is higher than the contrary case: $C(Maj, Min) > C(Min, Maj)$

- To develop a hypothesis that minimizes the **overall cost** on the training data set

$$\sum_j P(j|x)C(i, j, x)$$

- Each example, x , can be associated with a cost $C(i, j, x)$, which defines the cost of predicting class i for x when the true class is j
- Bayes conditional risk: conditional probabilities of class j given feature vector or example x
- Cons
 - Not always have a cost attached to making an error
 - The costs can be different for every example and not only for every type of error

1.9.5.1 Cost-Sensitive Dataspace Weighting

References

<https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28>

<https://www.datacamp.com/community/tutorials/diving-deep-imbalanced-data>

He, H. & Garcia, E.A., 2009. Learning from Imbalanced Data. *IEEE transactions on knowledge and data engineering*, 21(9), pp.1263–1284.

Chawla, N.V. 2010. Data Mining for Imbalanced Datasets: An Overview. *Data Mining and Knowledge Discovery Handbook*. O. Maimon and L. Rokach, eds. Springer US. 875–886.

Liu, X.-Y., Wu, J. and Zhou, Z.-H. 2009. Exploratory Under-Sampling for Class-Imbalance Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*. 39, 2 (2009), 539–550.

1.10 Model Interpretability

1.10.1 Global Feature Importance

1.10.1.1 Permutation Importance/Mean Decrease Accuracy (MDA)

- Question: what variables most affect predictions
- Measure the **importance of a feature** by calculating the increase in the model's prediction error after permuting the feature/how random re-shuffling (thus preserving the distribution of the variable) of each predictor influences model performance
 - Replace a feature with random noise (e.g., shuffle values) in the test/train set, and compute score
 - A feature is important if shuffling its values increases the model error
 - Calculated after a model has been fitted
 - **Global** method: considers all instances and gives a statement about the global relationship of a feature with the predicted outcome
- Pros
 - Fast to calculate
 - * Most suitable for computing feature importance when the number of features is not huge; it can be resource-intensive otherwise
 - Nice interpretation
 - * Feature importance is the increase in model error when the feature's information is destroyed
 - Feature importance measurements are **comparable** across different problems
 - The importance measure automatically takes into account all **interactions** with other features
 - * By permuting the feature \longrightarrow destroy the interaction effects with other features
 - * Permutation feature importance takes into account both the main feature effect and the interaction effects on model performance
 - Does not require retraining the model
 - Consistent
 - * Consistency states that changing a model so a feature has a larger impact on the model will never decrease the attribution assigned to that feature.

- Cons
 - Unclear whether should use training or test data to compute the feature importance
 - * In random forest, MDA measures the accuracy reduction in **out-of-bag samples** when the values of the feature are randomly permuted
 - Need the true outcome
 - Depend on shuffling the feature
 - * Adds randomness to the measurement
 - * When the permutation is repeated, the results might vary greatly
 - * Repeating the permutation and averaging the importance measures over repetitions stabilizes the measure, but increases the time of computation
 - The importance of the interaction between two features is included in the importance measurements of both features
 - * Feature importances do not add up to the total drop in performance, but the sum is larger
 - * Only if there is no interaction between the features, as in a linear model, the importances add up approximately.
 - If features are correlated, the permutation feature importance can be biased by unrealistic data instances

1.10.1.2 Mean Decrease in Impurity (MDI)

- MDI calculates each feature importance as
 - the times a feature is used to **split a node**, weighted by the number of samples it splits
 - total **decrease in node impurity** (Gini index, information gain, entropy, or variance) at all tree nodes where the feature appears (in `scikit-learn`)
- Global method
- Pros
 - Fast to compute
- Cons
 - Inconsistent
 - * When a model is changed such that a feature has a higher impact on the model's output, current methods can actually lower the importance of that feature
 - * A feature with a large attribution value might be less important than another feature with a smaller attribution
 - Inconsistency prevents the meaningful comparison of attribution values across features

1.10.1.3 Partial Dependence Plots

- Question: how a feature affects predictions
- The partial dependence plot shows the **marginal effect** of each features (one or two) on the predicted outcome of a model when the **rest of the predictors are held constant**
 - Calculated after a model has been fit
- Global method
- **Partial dependence function** of $f(X)$ on X_S

$$f_S(X_S) = E_{X_C} f(X_S, X_C)$$

- Input predictor variables: $X^T = (X_1, X_2, \dots, X_p)$
- Subvector X_S
 - * $S \subset \{1, 2, \dots, p\}$
 - * Features for which the partial dependence function should be plotted
 - * For which we want to know the effect on the prediction
 - * Size of the subsets X_S must be small ($l \approx 1, 2, 3$)
 - * Subsets whose effect on $f(X)$ is approximately additive or multiplicative will be most revealing
- \mathcal{C} : the complement set
- $S \cup \mathcal{C} = \{1, 2, \dots, p\}$
- $f(X)$: a general function will in principle depend on all of the input variables: $f(X) = f(X_S, X_C)$
- Represent the **average marginal effect** of X_S on $f(X)$ after **accounting for the (average) effects of the other variables** X_C on $f(X)$
 - By marginalizing the machine learning model output over the distribution of the features in set \mathcal{C} , get a function that depends only on features in S
- Assumptions
 - X_S are not correlated with the X_C
 - Useful description of the effect of the chosen subset on $f(X)$ when X_S do **not have strong interactions** with X_C
- Pros
 - Intuitive

- Easy to implement
- Cons
 - Realistic maximum number of features in a partial dependence function is two
 - Some PD plots do not show the feature distribution
 - * Might overinterpret regions with almost no data
 - * Show a rug (indicators for data points on the x-axis) or a histogram
 - Assumption of independence
 - * When the features are correlated, create new data points in areas of the feature distribution where the actual probability is very low
 - * Alternative: Accumulated Local Effect (ALE) plots or short ALE plots that work with the **conditional** instead of the marginal distribution
 - Heterogeneous effects might be hidden
 - * Only show the average marginal effects
 - * Suppose that for a feature half of the data points have a positive association with the prediction and the other half has a negative association, the PD curve could be a horizontal line

1.10.2 Global Surrogate

- An interpretable surrogate model is trained to approximate the predictions of a black box model as accurately as possible

- Steps

1. Select a dataset X

- This can be the same dataset that was used for training the black box model or a new dataset from the same distribution

2. For the selected dataset X , get the predictions of the black box model

3. Select an interpretable model type (linear model, decision tree, ...)

4. Train the interpretable model on the dataset X and **the predictions of the black box model**

5. Measure how well the surrogate model replicates the predictions of the black box model

-

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^n \left(\hat{y}_*^{(i)} - \hat{y}^{(i)} \right)^2}{\sum_{i=1}^n \left(\hat{y}^{(i)} - \bar{\hat{y}} \right)^2}$$

- $\hat{y}_*^{(i)}$ is the prediction for the i -th instance of the surrogate model

- $\hat{y}^{(i)}$ the prediction of the black box model and

- $\bar{\hat{y}}$ the mean of the black box model predictions

- The percentage of variance that is captured by the surrogate model

- * If R^2 is close to 1 (low SSE), then the interpretable model approximates the behavior of the black box model very well

6. Interpret the surrogate model

- Pro

- Flexible

- * Any interpretable model can be used

- Intuitive and straightforward

- Con

- Draw conclusions about the model and not about the data, since the surrogate model never sees the real outcome

- Not clear what the best cut-off for R-squared is in order to be confident that the surrogate model is close enough to the black box model
- It could happen that the interpretable model is very close for one subset of the dataset, but widely divergent for another subset
 - * The interpretation for the simple model would not be equally good for all data points

1.10.3 Local Surrogate

- Interpretable models that are used to explain **individual predictions** of black box machine learning models

1.10.3.1 Local Interpretable Model-agnostic Explanations (LIME)

Summary

Version 1:

An explanation is obtained by locally approximating the selected model with an interpretable one (such as linear models with regularisation or decision trees). The interpretable models are trained on small perturbations (adding noise) of the original observation (row in case of tabular data), thus they only provide a good local approximation.

Version 2:

LIME tests what happens to the predictions when you give variations of your data into the selected model. LIME generates a new dataset consisting of **permuted samples** and the corresponding predictions of the black box model. On this new dataset LIME then trains an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest. The learned model should be a good approximation of the selected model predictions **locally (local fidelity)**, but it does not have to be a good global approximation.

- Train local surrogate models to explain individual predictions
- Goal: to understand why the machine learning model made a certain prediction

Explanation model for instance x

$$\text{explanation}(x) = \arg \min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

- $\mathcal{L}(f, g, \pi_x)$
 - How unfaithful g is in approximating f in the locality defined by π_x
 - In order to ensure both interpretability and local fidelity, minimize $\mathcal{L}(f, g, \pi_x)$ while having $\Omega(g)$ be low enough to be interpretable by humans
 - Approximate $\mathcal{L}(f, g, \pi_x)$ by drawing samples, weighted by π_x
 - LIME only optimizes this part
- G

- A class of potentially interpretable models
- $g \in G$
- Model complexity $\Omega(g)$
 - User has to define the number of features in the interpretable model
 - E.g., number of features (K)
- π_x
 - Proximity measure between an instance z to x , so as to define **locality** around x
 - * z : a perturbed sample
 - Exponential smoothing kernel

Pros

- **Model-agnostic:** the same local, interpretable model works for any black box model
- Can estimate the faithfulness of the explanation on \mathcal{Z} of perturbed samples with the associated labels
 - The fidelity measure (how well the interpretable model approximates the black box predictions) gives us a good idea of how reliable the interpretable model is in explaining the black box predictions in the neighborhood of the data instance of interest
 - This estimate of faithfulness can also be used for selecting an appropriate family of explanations from a set of multiple interpretable model classes, thus adapting to the given dataset and the classifier
- The explanations created with local surrogate models can use other features than the original black box model
- **Interpretable Representation**
 - LIME incorporates interpretability both in the optimization and in the notion of interpretable representation, such that domain and task specific interpretability criteria can be accommodated.
- Work for tabular data, text and images

Cons

- The correct definition of the neighborhood (π_x) is an unsolved problem when using LIME with tabular data
- The complexity of the explanation model $\Omega(G)$ has to be defined in advance
- Data points are sampled from a Gaussian distribution, ignoring the **correlation between features**

- Lead to unlikely data points which can then be used to learn local explanation models
- Instability of the explanations
- While the underlying model can be treated as a black-box, certain interpretable representations will not be powerful enough to explain certain behaviors
- Our choice of G (sparse linear models) means that if the underlying model is highly non-linear even in the locality of the prediction, there may not be a faithful explanation

1.10.4 Shapley Values

1.10.4.1 Summary

A method for assigning payouts to players depending on their contribution to the total payout

1.10.4.2 Shapley Value in Game Theory

- Coalitional game
 - A set N (of n players)
 - A **characteristic function** v that maps subsets of players to the real numbers: $v : 2^N \rightarrow \mathbb{R}$
 - * $v(\emptyset) = 0$, where \emptyset denotes the empty set
 - * if S is a coalition of players, then $v(S)$, called the worth of coalition S , describes the total expected sum of payoffs the members of S can obtain by cooperation
- Shapley Value
 - One way to distribute the total gains to the players, assuming that they all collaborate
 - The amount that player i gets given in a coalitional game (v, N)

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(N - |S| - 1)!}{N!} (v(S \cup \{i\}) - v(S))$$

- * N is the total number of players and the sum extends over all subsets S of N not containing player i

$$\varphi_i(v) = \frac{1}{\text{number of players}} \sum_{\text{coalitions excluding } i} \frac{\text{marginal contribution of } i \text{ to coalition}}{\text{number of coalitions excluding } i \text{ of this size}}$$

- Imagine the coalition being formed one actor at a time, with each actor demanding their contribution $V(S \cup \{i\}) - V(S)$ as a fair compensation, and then for each actor take the average of this contribution over the possible different permutations in which the coalition can be formed
- It is a **fair distribution** in the sense that it is the only distribution with certain desirable properties

Properties

1. Efficiency

- The sum of the Shapley values of all agents equals the value of the grand coalition, so that all the

gain is distributed among the agents

$$\sum_{i \in N} \varphi_i(v) = v(N)$$

2. Symmetry

- If i and j are two actors who are equivalent in the sense that $v(S \cup \{i\}) = v(S \cup \{j\})$ for every subset S of N which contains neither i nor j , then $\varphi_i(v) = \varphi_j(v)$

3. Linearity

- If two coalition games described by gain functions v and w are combined, then the distributed gains should correspond to the gains derived from v and the gains derived from w for every i in N

$$\varphi_i(v + w) = \varphi_i(v) + \varphi_i(w)$$

- For any real number α

$$\varphi_i(\alpha v) = \alpha \varphi_i(v)$$

4. Null player/Dummy?

- The Shapley value $\varphi_i(v)$ of a null i in a game v is zero
- A player i is null in v if $v(S \cup \{i\}) = v(S)$ for all coalitions S that do not contain i

1.10.4.3 Shapley Value in ML Models

The Shapley value of a feature value is its contribution to the payout, weighted and summed over all possible feature value combinations

$$\varphi_j(val) = \sum_{S \subseteq \{x_1, \dots, x_p\} \setminus \{x_p\}} \frac{|S|!(p - |S| - 1)!}{p!} (val(S \cup \{x_j\}) - val(S))$$

- S is a subset of the features used in the model
- x is the vector of feature values of the instance to be explained
- p : the number of features
- $val_x(S)$ is the prediction for feature values in set S that are marginalized over features that are not included in set S

$$val_x(S) = \int \hat{f}(x_1, \dots, x_p) d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X))$$

Interpretation

For feature value j :

- The Shapley value of the j -th feature contributed φ_j to the prediction of this particular instance compared to the **average prediction** for the dataset
- Contribution of a feature value to the difference between the actual prediction and the mean prediction, given the current set of feature values
- The feature values enter a room in random order. All feature values in the room participate in the game (= contribute to the prediction). The Shapley value of a feature value j is **the average change in the prediction that the coalition already in the room receives when the feature value joins them**.

Properties

1. Efficiency

- The feature contributions must add up to the difference of prediction for x and the average

$$\sum_{j=1}^p \varphi_j = \hat{f}(x) - E_X(\hat{f}(X))$$

2. Symmetry

- The contributions of two feature values j and k should be the same if they contribute equally to all possible coalitions

If

$$\text{val}(S \cup \{x_j\}) = \text{val}(S \cup \{x_k\})$$

for all

$$S \subseteq \{x_1, \dots, x_p\} \setminus \{x_j, x_k\}$$

then

$$\varphi_j = \varphi_k$$

3. Linearity/Additivity

- Suppose you trained a random forest, which means that the prediction is an average of many decision trees. The additivity property guarantees that for a feature value, you can calculate the Shapley value for each tree individually, average them, and get the Shapley value for the feature value for the random forest.

4. Null player/Dummy?

- A feature j that does not change the predicted value - regardless of which coalition of feature values it is added to - should have a Shapley value of 0

If

$$\text{val}(S \cup \{x_j\}) = \text{val}(S)$$

for all

$$S \subseteq \{x_1, \dots, x_p\}$$

then

$$\varphi_j = 0$$

Estimation

todo

Pro

- The difference between the prediction and the average prediction is **fairly distributed among the feature values of the instance**
 - **Efficiency property** of Shapley values
 - LIME does not guarantee that the prediction is fairly distributed among the feature
 - The Shapley value might be the only method to deliver a full explanation
- The Shapley value allows **contrastive explanations**
 - Instead of comparing a prediction to the average prediction of the entire dataset, Shapley value could compare it to **a subset or even to a single data point**
 - LIME do not have contrastiveness.
- The Shapley value is the only explanation method with a solid theory
 - Methods like LIME assume linear behavior of the machine learning model locally, but there is no theory as to why this should work

Con

- Computationally expensive
- Explanations created with the Shapley value method always use **all the features**
 - SHAP can provide explanations with few features
- The Shapley value returns a simple value per feature, but no prediction model like LIME

- Cannot be used to make statements about changes in prediction for changes in the input
- Need access to the data if calculating the Shapley value for a new data instance
- May include unrealistic data instances when features are correlated
 - Permutation-based interpretation methods
 - Involving sampling values from the feature's **marginal** distribution

1.10.4.4 SHAP: SHapley Additive explanations

SHAP turns the Shapley values method into an optimization problem and uses a special kernel function to measure proximity of data instances. The results of SHAP are sparse (many Shapley values are estimated to be zero), which is the biggest difference from the classic Shapley values.

Pro

- SHAP values as the only consistent and locally accurate individualized feature attributions
- Both global and local methods

References

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. “Why should I trust you?: Explaining the predictions of any classifier.” Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM (2016)

Shapley, L. (1988). A value for n-person games. In A. Roth (Ed.), The Shapley Value: Essays in Honor of Lloyd S. Shapley (pp. 31-40). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511528446.003

Štrumbelj, E. and Kononenko, I. 2014. Explaining prediction models and individual predictions with feature contributions. Knowledge and information systems. 41, 3 (Dec. 2014), 647–665. DOI:<https://doi.org/10.1007/s10115-013-0679-x>.

Lundberg, S.M., Erion, G.G. and Lee, S.-I. 2018. Consistent Individualized Feature Attribution for Tree Ensembles. arXiv [cs.LG].

https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html

<https://www.kaggle.com/dansbecker/permutation-importance>

<https://christophm.github.io/interpretable-ml-book/pdp.html>

https://en.wikipedia.org/wiki/Shapley_value

https://www.cs.cornell.edu/courses/cs684/2004sp/scribenotes_Mar29.pdf

<https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27>

1.11 Bootstrap

Bootstrapping is a **nonparametric** approach for assessing **statistical accuracy** that substitutes computation for more traditional distributional assumptions and asymptotic results.

Bootstrapping uses the sample data to estimate relevant characteristics of the population. The sampling distribution of a statistic is then constructed empirically by **resampling** from the sample. The resampling procedure is designed to parallel the process by which sample observations were drawn from the population. For example, if the data represent an independent random sample of size n , then each bootstrap sample selects n observations with **replacement** from the original sample. The key bootstrap analogy is the following: *The population is to the sample as the sample is to the bootstrap samples.*

1. Construct an **empirical probability distribution**, F_n , from the sample by placing a probability of $1/n$ at each point, x_1, x_2, \dots, x_n of the sample (nonparametric maximum likelihood estimate of the population distribution, F)
2. **Monte Carlo Resampling:** From the empirical distribution function, F_n , draw a **random sample of the same size n with replacement** (the accuracy/variation of statistical estimates depends on the size of the sample).
3. Calculate the statistic of interest, θ_n , for this resample, yielding θ_n^*
4. Repeat steps 2 and 3 B times, where B is a large number, to create B resamples. Typically, B is at least equal to 1000 when an estimate of confidence interval around T_n is required.

Pros:

- Because it does not require distributional assumptions (such as normally distributed errors), the bootstrap can provide more accurate inferences when the data are not well behaved or when the sample size is small.
- It is possible to apply the bootstrap to statistics with sampling distributions that are difficult to derive, even asymptotically.
- It is relatively simple to apply the bootstrap to complex data-collection plans (such as stratified and clustered samples).

Cons:

- Computationally expensive
- Can only tell things about the original sample, and won't give any new information about the real population
- Bootstrap can fail, such as

- Distribution that do not have finite moments
- Small sample sizes
- Estimate extreme values from the distribution
- Estimate variance in survey sampling where the population size is N and a large sample n is taken

1.11.1 Bootstrap for Confidence Intervals

1.11.1.1 Normal-Theory CIs:

A symmetric $100(1 - \alpha)\%$ CI:

$$\hat{\theta} \pm t_{\alpha/2} \sigma_{\hat{\theta}}$$

where $\hat{\theta}$ is our estimate of θ , $\sigma_{\hat{\theta}}$ is the standard error of $\hat{\theta}$, and $t_{\alpha/2}$ is the critical value of the test statistic, i.e., $P(t \leq t_{\alpha/2}) = \alpha/2$

- Assumes that distribution of test statistic is symmetric around zero
- As $n \rightarrow \infty$: $\hat{\theta} \approx N(\theta, \sigma_{\hat{\theta}}^2)$, so that $t_{\alpha/2} = z_{\alpha/2}$

More generally, write a $100(1 - \alpha)\%$ CI as

$$\left[\hat{\theta} - t_{1-\alpha/2} \sigma_{\hat{\theta}}, \hat{\theta} - t_{\alpha/2} \sigma_{\hat{\theta}} \right]$$

where $P(t \leq t_{1-\alpha/2}) = 1 - \alpha/2$ and $P(t \leq t_{\alpha/2}) = \alpha/2$

Interpretation

Through repeated samples, e.g. 95 out of 100 confidence intervals would be expected to contain true θ with $\alpha = 0.05$

Properties used to describe a confidence interval

- Length
 - $\hat{\theta}_{\text{up}} - \hat{\theta}_{\text{lo}}$
 - Describes the overall size of the CI
- Shape
 - $\frac{\hat{\theta}_{\text{up}} - \hat{\theta}}{\hat{\theta} - \hat{\theta}_{\text{lo}}}$
 - Describes the asymmetry of the CI

First and Second Order Accurate

A confidence interval is

- **First-order accurate** if the non-coverage probability on each side differs from the nominal value by $O(n^{-1/2})$: $P(\theta < \hat{\theta}_{\text{lo}}) = \alpha + h_{\text{lo}}/\sqrt{n}$ and $P(\theta > \hat{\theta}_{\text{up}}) = \alpha + h_{\text{up}}/\sqrt{n}$
- **Second-order accurate** if the non-coverage probability on each side differs from the nominal value by

$$O(n^{-1}): P(\theta < \hat{\theta}_{\text{lo}}) = \alpha + h_{\text{lo}}/n \quad \text{and} \quad P(\theta > \hat{\theta}_{\text{up}}) = \alpha + h_{\text{up}}/n$$

1.11.1.2 t-Confidence Interval with Bootstrap Standard Error

Bootstrap SE:

$$\hat{\sigma}_B = \text{SE}^*(\hat{\theta}^*) = \sqrt{\frac{\sum_{b=1}^B (\hat{\theta}_b^* - \bar{\theta}^*)^2}{B-1}}$$

CIs:

$$\hat{\theta} \pm t_{\alpha/2} \hat{\sigma}_B$$

Pros

- Simple to form and easy to understand
- Can be applied to situations where $\sigma_{\hat{\theta}}$ is difficult to derive

Cons

- No real benefit over the standard t interval using $\hat{\sigma}$
- Tend to be too narrow for small n because $\hat{\sigma}_B$ is too narrow
- Comparable to using the MLE $\hat{\sigma} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$ instead of the unbiased estimate $s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$
- Can perform poorly if distribution is highly skewed
- Only first-order accurate

1.11.1.3 Bootstrap Confidence Intervals via Percentiles

If we have $B = 10,000$ bootstrap replications of $\hat{\theta}$: $\hat{\theta}_{(1)}^* \leq \hat{\theta}_{(2)}^* \leq \dots \leq \hat{\theta}_{(B)}^*$, define the 95% confidence interval using

$$[\hat{\theta}_{(250)}^*, \hat{\theta}_{(9750)}^*] = [\hat{\theta}_{\text{lo}}, \hat{\theta}_{\text{up}}]$$

Pros

- Simple to form and easy to understand
- Range preserving and transformation invariant
- Advantage over t -CIs with bootstrap SE when data are skewed

Cons

- Tends to be too narrow for small n (worse than t with bootstrap SE)
- Comparable to using $z_{\alpha/2} \frac{\hat{\sigma}}{\sqrt{n}}$ instead of $t_{\alpha/2} \frac{s}{\sqrt{n}}$
 - Can use an adjustment to correct for narrowness bias
- Does partial skewness correction, which adds random variability
- Only first-order accurate

1.11.1.4 Bootstrap t-Table Confidence Intervals

Given B bootstrap samples $\mathbf{x}_1^*, \dots, \mathbf{x}_B^*$, compute

$$t_b^* = \frac{\hat{\theta}_b^* - \hat{\theta}}{\hat{\sigma}_b}$$

where $\hat{\sigma}_b$ is standard error for b -th bootstrap sample

- If $\hat{\theta}$ is sample mean, then $\hat{\sigma}_b = \sqrt{\frac{\sum_{i=1}^n (x_{i(b)}^* - \bar{x}_{b^*})^2}{n^2}}$
- For other statistics, need bootstrap SE for each bootstrap sample

Given t_b^* for $b \in \{1, \dots, B\}$, define the α -th quantile q_α as $\#\{t_b^* \leq q_\alpha\} / B = \alpha$

Form the "bootstrap- t " interval: $[\hat{\theta} - q_{1-\alpha/2} \hat{\sigma}_B, \hat{\theta} - q_{\alpha/2} \hat{\sigma}_B]$

Pros

- Simple idea with intuitive procedure
- Works well for location parameters
- Second-order accurate

Cons

- Not range preserving or transformation invariant
- Formation of $\hat{\sigma}_b$ requires iterated bootstrap
- Doesn't work as well for correlation/association measures

1.11.1.5 The bias-corrected and Accelerated (BCa) Bootstrap Interval

BC_a intervals use percentiles of bootstrap distribution, but not necessarily use the 100α -th and $100(1-\alpha)$ -th percentiles

BC_a intervals have the form:

$$\left[\hat{\theta}_{(\alpha_1)}^*, \hat{\theta}_{(\alpha_2)}^* \right] = \left[\hat{\theta}_{\text{lo}}, \hat{\theta}_{\text{up}} \right]$$

where

- $\alpha_1 = \Phi \left(\hat{z}_0 + \frac{\hat{z}_0 + z_{(\alpha)}}{1 - \hat{a}(\hat{z}_0 + z_{(\alpha)})} \right)$
- $\alpha_2 = \Phi \left(\hat{z}_0 + \frac{\hat{z}_0 + z_{(1-\alpha)}}{1 - \hat{a}(\hat{z}_0 + z_{(1-\alpha)})} \right)$
- $z_{(\alpha)}$ is the 100 α -th percentile of standard normal
- $\Phi(\cdot)$ is the cdf of standard normal (pnorm)
- **Bias-correction factor \hat{z}_0 :**
 - $\hat{z}_0 = \Phi^{-1} \left(\# \left\{ \hat{\theta}_b^* < \hat{\theta} \right\} / B \right)$
 - \hat{z}_0 measures median bias of $\hat{\theta}^*$, i.e., difference between median $\left(\hat{\theta}_b^* \right)$ and $\hat{\theta}$
- **Acceleration parameter \hat{a}**
 - Calculated using a jackknife approach
 - $\hat{a} = \frac{\sum_{i=1}^n (\hat{\theta}_{(\cdot)} - \hat{\theta}_{(i)})^3}{6 \left\{ \sum_{i=1}^n (\hat{\theta}_{(\cdot)} - \hat{\theta}_{(i)})^2 \right\}^{3/2}}$
 - $\hat{\theta}_{(\cdot)} = \frac{1}{n} \sum_{i=1}^n \hat{\theta}_{(i)}$ is the average of jackknife estimates
 - $\hat{\theta}_{(i)} = s(\mathbf{x}_{(i)})$ is estimate of θ holding out x_i
 - Estimate the rate of change of the standard error of $\hat{\theta}$ with respect to the true parameter θ
 - Use to correct the (possibly unrealistic) assumption that $\hat{\theta} \approx N(\theta, \sigma_{\hat{\theta}}^2)$ assumes that $\sigma_{\hat{\theta}}$ is the same for all θ
- If $\hat{a} = \hat{z}_0 = 0$, the BC_a interval is the same as the percentile interval

Pros:

- Range preserving and transformation invariant
- Works well for a variety parameters
- Second-order accurate

Cons:

- Requires estimation of acceleration and bias-correction
- Less intuitive than other methods

1.11.2 Bootstrap Hypothesis Testing

Bootstrap hypothesis tests proceed by constructing an empirical sampling distribution for the test statistic. If T represents the test statistic computed for the original sample and T_b^* is the test statistic for the b th of B bootstrap samples, then (for a chi-square-like test statistic) the p -value for the test is $\#(T_b^* \geq T) / B$

Having obtained B bootstrap replications of the test statistic (T_b^*), the bootstrap estimate of the **achieved significance level** p -value for H_0 is

$$\hat{p}^* = \text{Prob}_{H_0} \{T_b^* \geq T\} = \frac{\sum_{b=1}^B (T_b^* \geq T)}{B}$$

1.11.3 Bootstrapping Regression Models

1.11.3.1 Random

Directly resampling the observations \mathbf{z}'_i implicitly treats the regressors X_1, \dots, X_k as random.

1. Collect the response-variable value and regressors for each sample

$$\mathbf{z}'_i \equiv [Y_i, X_{i1}, \dots, X_{ik}]$$

2. Re-sample observations $\mathbf{z}'_1, \mathbf{z}'_2, \dots, \mathbf{z}'_n$
3. Compute the regression estimator for each of the resulting bootstrap samples: $\mathbf{z}^{*'}_{b1}, \mathbf{z}^{*'}_{b2}, \dots, \mathbf{z}^{*'}_{bn}$, producing B sets of bootstrap regression coefficients, $\mathbf{b}^*_b = [\beta^*_{0b}, \beta^*_{b1}, \dots, \beta^*_{bk}]'$, which can be used in the usual manner to construct bootstrap standard errors and confidence intervals for the regression coefficients.

1.11.3.2 Fixed

Treat the X s as fixed (e.g. if the data are derived from an experimental design):

1. Estimate the regression coefficients $\beta_0, \beta_1, \dots, \beta_k$ for the original sample, and calculate the fitted value and residual for each observation:

$$\hat{Y}_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_k x_{ik}$$

$$E_i = Y_i - \hat{Y}_i$$

2. Select bootstrap samples of the residuals, $\mathbf{e}^*_b = [E^*_{b1}, E^*_{b2}, \dots, E^*_{bn}]'$, and from these calculate bootstrapped Y values, $\mathbf{y}^*_b = [Y^*_{b1}, Y^*_{b2}, \dots, Y^*_{bn}]'$, where $Y^*_{bi} = \hat{Y}_i + E^*_{bi}$
3. Regress the bootstrapped Y values on the fixed X values to obtain bootstrap regression coefficients.
 - For example, estimates are calculated by least-squares regression, then $\mathbf{b}^*_b = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{y}^*_b$ for $b = 1, \dots, B$
4. The resampled $\mathbf{b}^*_b = [\beta^*_{b0}, \beta^*_{b1}, \dots, \beta^*_{bk}]'$ can be used in the usual manner to construct bootstrap standard errors and confidence intervals for the regression coefficients.

References

Friedman, J., Hastie, T. and Tibshirani, R., 2001. The elements of statistical learning. New York, NY, USA:: Springer series in statistics.

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. An Introduction to Statistical Learning with Applications in R (Springer Texts in Statistics). In :. Springer.

<http://users.stat.umn.edu/~helwig/notes/bootci-Notes.pdf>

<https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/readings/MIT18-05S14-Reading24.pdf>

<https://faculty.washington.edu/heagerty/Courses/b572/public/GregImholte-1.pdf>

<https://faculty.washington.edu/heagerty/Courses/b572/public/GregImholte-2.pdf>

https://www.sagepub.com/sites/default/files/upm-binaries/21122_Chapter_21.pdf

1.12 Pro and Cons of Algorithms

2 Supervised Learning

2.1 Linear regression

Supervised

Regression

Let $X \in \mathbb{R}^p$ be a vector of predictors. In linear regression, we observe $Y \in \mathbb{R}$, and assume a linear model:
 $\mathbb{E}(Y|X) = \beta^T X$

2.1.1 Derivation

- **Method 1:**

A set of training data $(x_1, y_1) \dots (x_N, y_N)$ to estimate β . Each $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$ is a vector of feature measurements for the i th case.

$$y_i = \underbrace{\beta_0 + \beta_1 x_i}_{\hat{y}_i} + \epsilon_i$$

Least squares:

Pick the coefficients $\beta = (\beta_0, \beta_1, \dots, \beta_p)^T$ to minimize the **residual sum of squares**

$$\begin{aligned} \text{RSS}(\beta) &= \sum_{i=1}^N (y_i - f(x_i))^2 \\ &= \sum_{i=1}^N \left(y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j \right)^2 \end{aligned}$$

where

$$\epsilon_i = y_i - \hat{y}_i$$

and

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

In the case of one X :

The best choice of β_0 and β_1 will be chosen to minimize the *least square fit*:

$$\mathcal{Q} = \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_i))^2 = \sum_{i=1}^n \epsilon_i^2$$

Let's solve:

$$\frac{\partial \mathcal{Q}}{\partial \beta_0} = -2 \sum (y_i - (\beta_0 + \beta_1 x_i)) = 0$$

and

$$\frac{\partial \mathcal{Q}}{\partial \beta_1} = -2 \sum x_i (y_i - (\beta_0 + \beta_1 x_i)) = 0$$

\Leftrightarrow

$$\sum y_i = n\beta_0 + \beta_1 \sum x_i$$

and

$$\sum x_i y_i = -2 \sum x_i (y_i - (\beta_0 + \beta_1 x_i)) = 0$$

We get:

$$\hat{\beta}_1 = \frac{n \sum x_i y_i - (\sum x_i)(\sum y_i)}{n \sum x_i^2 - (\sum x_i)^2}$$

$$\hat{\beta}_0 = \underbrace{\frac{1}{n} \sum y_i}_{\bar{y}} - \hat{\beta}_1 \underbrace{\left(\frac{1}{n} \sum x_i \right)}_{\bar{x}}$$

• **Method 2:**

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$$

$$\frac{\partial \text{RSS}}{\partial \beta} = -2\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta)$$

$$\frac{\partial^2 \text{RSS}}{\partial \beta \partial \beta^T} = 2\mathbf{X}^T \mathbf{X}$$

Assume that \mathbf{X} has **full column rank** (one or more of its columns is equal to a linear combination of the others), and hence $\mathbf{X}^T \mathbf{X}$ is positive definite, set the first derivative to zero:

$$\mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0$$

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \underbrace{\mathbf{X} (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T}_{\text{hat matrix}} \mathbf{y}$$

hat matrix \mathbf{H} computes the orthogonal projection, and hence it is also known as a **projection matrix**

To form tests of hypothesis and confidence intervals for the parameters β_j :

Now assume that the observations y_i are **uncorrelated** and have **constant variance** σ^2 , and that the x_i are fixed (non random).

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

$$\hat{\sigma}^2 = \frac{1}{N-p-1} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

The $N - p - 1$ rather than N makes $\hat{\sigma}^2$ an unbiased estimate of σ^2 : $E(\hat{\sigma}^2) = \sigma^2$

To draw inferences about the parameters and the model, now assume the conditional expectation of Y is **linear** in X_1, \dots, X_p and assume that the deviations of Y around its expectation are **additive** and **Gaussian**.

$$Y = E(Y|X_1, \dots, X_p) + \varepsilon$$

$$= \beta_0 + \sum_{j=1}^p X_j \beta_j + \varepsilon$$

where $\varepsilon \sim N(0, \sigma^2)$

$$\hat{\beta} \sim N\left(\beta, (\mathbf{X}^T \mathbf{X})^{-1} \sigma^2\right)$$

$$(N - p - 1)\hat{\sigma}^2 \sim \sigma^2 \chi_{N-p-1}^2 \text{ (a chi-squared distribution)}$$

To test the null hypothesis that particular coefficient $\beta_j = 0$:

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma} \sqrt{v_j}} \sim t_{N-p-1}$$

where v_j is the j th diagonal element of $(\mathbf{X}^T \mathbf{X})^{-1}$

To test for the significance of groups of coefficients simultaneously (e.g., to test if a categorical variable with k levels can be excluded from a model, need to test whether the coefficients of the dummy variables used to represent the levels can all be set to zero):

$$F = \frac{(\text{RSS}_0 - \text{RSS}_1) / (p_1 - p_0)}{\text{RSS}_1 / (N - p_1 - 1)}$$

2.1.2 Assumptions

1. **Linearity and additivity** of the relationship between Y and X s
 - Scatter plot of observed/residuals versus predicted values
 - A nonlinear transformation to the dependent and/or independent variables if appropriate
2. The **X s should not be correlated** (X is column-rank deficient). Absence of this phenomenon is known as **multicollinearity**.
3. Statistical **independence of Y s**. Absence of this phenomenon is known as **autocorrelation**.
 - Durbin-Watson test:
 - Analyses **linear autocorrelation** and only between direct neighbors (**first order** effects)
 - H_0 : No first order autocorrelation
 - $d \in [0, 4]$
 - No auto-correlation: $d \in [1.5, 2.5]$
 - Minor cases of positive serial correlation ($d \in [1.2, 1.6]$):
 - Add lags of Y and/or lags of some of X s
 - Significant negative correlation in the residuals ($d > 2.6$):
 - Overdifferenced some of your variables ?????????
 - Significant correlation at the seasonal period:
 - Seasonality has not been properly accounted for in the model
 - * Seasonally adjust the variables
 - * Use seasonal lags and/or seasonally differenced variables
 - * Add seasonal dummy variables to the model (i.e., indicator variables for different seasons of the year, such as MONTH=1 or QUARTER=2)
 - Major cases of serial correlation ($d < 1$): a fundamental structural problem in the model
4. **Normality of the error distribution** $\leftarrow \varepsilon \sim N(0, \sigma^2)$
 - A normal probability plot or normal quantile plot of the residuals
 - The fractiles of error distribution versus the fractiles of a normal distribution having the same mean and variance

- If the distribution is normal, the points on such a plot should fall close to the diagonal reference line
- Violations of normality often arise when
 - The distributions of the X s and/or Y are significantly non-normal
 - The linearity assumption is violated

5. Homoscedasticity (constant variance) of the errors $\leftarrow \varepsilon \sim N(0, \sigma^2)$

- Scatter plot: residuals versus predicted values

2.1.3 Collinearity

- The columns of \mathbf{X} are **not linearly independent** $\rightarrow \mathbf{X}$ is **not of full rank** (the data vectors are confined to a q -dimensional subspace ($q < p$). It looks like we've got p different variables, but really by a change of coordinates we could get away with just q of them) $\rightarrow \mathbf{X}^T \mathbf{X}$ is **not invertible** (correlations close to one)/is **singular** (exact linear relationship among the predictors)
- When $n < p$: will always have multicollinearity
- Diagnose collinearity
 - Variance Inflation Factor (VIF):
 - * $VIF_i = \frac{1}{1-R_i^2}$
 - * For X_i : by regression X_i on all of the other X_j and computing the R_i^2 of this regression
 - * $VIF_i \geq 1$ with the variance inflation factor increasing as X_i becomes more correlated with some linear combination of the other predictors
 - Eigendecomposition:
 - * If any eigenvalues are zero, the data is multicollinear; if any are very close to zero, the data is nearly multicollinear
- How to impact estimates
 - Least squares coefficients are not uniquely defined
 - If \mathbf{x}_p is highly correlated with some of the other \mathbf{x}_k 's, the residual vector \mathbf{z}_p will be close to zero, and the coefficient $\hat{\beta}_p$ will be very unstable
 - Standard errors of the coefficient estimates will be bigger than if the predictors were uncorrelated
- Solutions

- Re-code and/or drop redundant columns in \mathbf{X}
- Principle components regression
 - * Need standardization
 - * Hard to interpret
 - * Need to decide k components
- Ridge Regression

2.1.4 Best Linear Unbiased Estimator (BLUE)

to-do

The Gauss-Markov Theorem implies that the **least squares estimates** of the parameters β have the **smallest variance** among all linear unbiased estimates. There may well exist a biased estimator with smaller mean squared error. Such an estimator would trade a little bias for a larger reduction in variance (e.g., RIDGE).

2.1.5 Pros

2.1.6 Cons

- The least squares estimates often have low bias but large variance

References:

<https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/lecture-17.pdf>

2.2 Logistic regression

Supervised

Classification

Logistic Regression is a classification algorithm that works by trying to learn a function that approximates the posterior probabilities $P(Y|X)$. It makes the central assumption that $P(Y|X)$ can be approximated as a **sigmoid function** $\sigma(z) = \frac{1}{1+e^{-z}}$ applied to a **linear combination** of input features, while at the same time ensuring that they sum to one and remain in $[0, 1]$.

For a single training datapoint $(x; y)$:

$$\begin{aligned} p(y_i = 1|x_i) &= h_{\theta}(x_i) \\ &= \sigma(z) \\ &= \sigma(\beta_0 + \beta_i x_i) \\ &= \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^m \beta_i x_i)}} \end{aligned}$$

$$\begin{aligned} p(y_i = 0|x_i) &= 1 - h_{\theta}(x_i) \\ &= \frac{e^{-(\beta_0 + \sum_{i=1}^m \beta_i x_i)}}{1 + e^{-(\beta_0 + \sum_{i=1}^m \beta_i x_i)}} \end{aligned}$$

Combine these two cases into one expression:

$$p(y_i|x_i) = [h_{\theta}(x_i)]^{y_i} [1 - h_{\theta}(x_i)]^{1-y_i}$$

2.2.1 Logit and Logistic

$$\log \frac{p(y_i|x_i)}{1 - p(y_i|x_i)} = \text{logit}(p(y_i|x_i)) = \beta_0 + \sum_{i=1}^m \beta_i x_i = \log \frac{\Pr(y = 1|X = x)}{\Pr(y = 0|X = x)} = \beta_0 + \beta^T x$$

$$\text{logit}^{-1}(\alpha) = \text{logistic}(\alpha) = \frac{1}{1 + \exp(-\alpha)} = \frac{\exp(\alpha)}{\exp(\alpha) + 1}$$

Logistic regression assumes that observations are **independent of each other** and **identically distributed**.

2.2.2 Maximum Likelihood Estimation

Maximum the (conditional) likelihood function:

$$\begin{aligned} L(x, y) &= \prod_{i=1}^m \Pr(Y = y_i | X = x_i) \\ &= \prod_{i=1}^m [h_\theta(x_i)]^{y_i} [1 - h_\theta(x_i)]^{1-y_i} \end{aligned}$$

Maximum the log likelihood:

$$\log L(x, y) = \sum_{i=1}^m y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

Minimize the -log likelihood:

$$-\log L(x, y) = -\sum_{i=1}^m y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

To find the maximum likelihood, differentiate the log likelihood with respect to the parameters, set the derivatives equal to zero, and solve.

2.2.3 Least Squared Error Estimation

Minimize sum of squared error:

$$L(x, y) = \frac{1}{2} \sum_{i=1}^m (y_i - h_\theta(x_i))^2$$

Comparisons: MLE vs. Least Squared Error

- Consistency: both loss functions can ensure consistency. The classification errors converge to optimal Bayes error as sample size goes to infinity.
- Robustness: If we think logistic regression outputs a probability distribution vector, then both methods try to **minimize the distance between the true probability distribution vector and the predicted probability distribution vector**. The only difference is in how the distance is defined. For maximum likelihood method, the distance measure is **KL-divergence**. For least squared error method, the distance is Euclidean distance. Since KL-divergence is unbounded, but (squared) Euclidean distance has an upper bound 2, it seems **logistic regression trained by minimizing squared error is more robust to outliers**.
- Convexity and Convergence: training logistic regression by **maximizing the likelihood is a convex optimization problem**, which is easier; training logistic regression by minimizing squared error is a non-convex optimization problem, which is harder.

2.2.4 Newton's Method

2.2.5 Pros

2.2.6 Cons

- Two-Class Problems: can be extended for multiclass classification, but is rarely used for this purpose.
- Unstable with well separated Classes
- Unstable with few samples

2.3 GLM

In a generalized model, $Y|X$ is assumed to follow an **exponential family distribution** $f(y; \theta, \phi) = \exp\left(\frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi)\right)$ with mean μ . We also assume a **transformation of the conditional expectation** $\mathbb{E}(Y|X)$ is a **linear function** of X for some function g :

$$g(\mathbb{E}(Y|X)) = X\beta^T$$

Here θ, ϕ are **parameters**, and a, b, c are **functions**. Any density of the above form is called an **exponential family density**. The parameter θ is called the **natural parameter**, and the parameter ϕ the **dispersion parameter**.

To estimate the parameters of an exponential family GLM: **maximum likelihood/iteratively reweighted least squares**

2.3.1 Three Components

Given predictors $X \in \mathbb{R}^p$ and an outcome Y , a generalized linear model is defined by:

1. Random component

- Specifies a distribution for $Y|X$

2. Systematic component

- Relates a parameter η to the predictors X
- A linear function of regressors
- $\eta = \beta^T X = \beta_1 X_1 + \dots + \beta_p X_p$

3. Link function/mean function $g(\cdot)$

- Connects the random and systematic components
- A smooth and invertible linearizing link function $g(\cdot)$
- Transforms the expectation of the response variable, $\mu \equiv E(Y)$ to the linear predictor:

$$g(\mu) = \eta = \beta^T X = \beta_1 X_1 + \dots + \beta_p X_p$$

- $\mu = g^{-1}(\eta) = g^{-1}(\beta_1 X_1 + \dots + \beta_p X_p)$

4. A variance function (usually not listed here)

- $\text{var}(Y_i) = \phi \text{var}(\mu)$
- The dispersion parameter ϕ is a constant

2.3.2 Assumptions

- The data y_1, y_2, \dots, y_n are independently distributed.
- The homogeneity of variance does not need to be satisfied (overdispersion, the observed variance is larger than what the model assumes, may be present)
- Errors need to be independent but not normally distributed.
- It uses maximum likelihood estimation (MLE) rather than ordinary least squares (OLS) to estimate the parameters and thus relies on large-sample approximations.
- Goodness-of-fit measures rely on sufficiently large samples, where a heuristic rule is that not more than 20% of the expected cells counts are less than 5.

2.3.3 Pros

- We do not need to transform the response Y to have a normal distribution.
- The choice of link is separate from the choice of random component thus we have more flexibility in modeling.
- If the link produces additive effects, then we do not need a constant variance. ????
- The models are fitted via Maximum Likelihood estimation; thus optimal properties of the estimators.

2.3.4 Cons

- Can have only a linear predictor in the systematic component
- Responses must be independent

2.3.5 GLM vs. Transformations of Response Variable in Linear Regression

- When transforming Y and regress the transformed response on the X
 - Model the $E[g(Y)] = \beta^T X$
 - Response variable has changed
 - Transformation must simultaneously improve linearity and homogeneity of variance
 - Transformation may not be defined on the boundaries of the sample space
- GLM:

– Model the transformed expectation of the response: $g[E(Y)] = \beta^T X$

- Not quite the same thing when (as is true except for the identity link) the link function $g(\cdot)$ is nonlinear

	log(Y) = x1 + x2	Y = x1 + x2, family = Gamma(link = "log")
Model	$E[\log(Y_i)] = \beta_0 + \beta_1 X_1 + \beta_2 X_2$	$\log(E[Y_i]) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$ $E[Y_i] = e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2}$
Make inference about	Arithmetic mean change on the log scale Geometric mean ratio on the original scale Multiplicative effects	Arithmetic means ($E[Y_i]$) on the original scale Arithmetic mean ratio on the original scale

2.3.6 Generalized Linear Models for Counts

2.3.7 Models for Overdispersed Data

2.3.7.1 Overdispersion

- May occur in **one-parameter exponential families** where the variance is supposed to be a function of the mean (the binomial or Poisson families where $E(Y) = \mu$ and $\text{Var}(Y) = \mu(1 - \mu/k)$ or $\text{Var}(Y) = \mu$, respectively)
- The actually observed variance from the data is larger than the variance imposed by the model
- The source for overdispersion may be a lack of independence in the data or a misspecification of the model

2.3.7.2 Deviance or Likelihood Ratio Statistic

- $D = 2 \left[l_S(\hat{\psi}) - l(\hat{\beta}) \right] \sim \chi_{K-p}^2$
 - $\hat{\psi}$ and $\hat{\beta}$ are the MLEs of the saturated and proposed model, respectively
 - K and p are the number of parameters in the saturated and proposed models, respectively
 - Saturated model: model that includes the maximum number of parameters that we can estimate from these data
- Can be used to test the fit of the link function and linear predictor to the data, or to test the significance of a particular predictor variable (or variables) in the model
- If D is large relative to the χ_{n-p}^2 distribution, then we have evidence against the null hypothesis that our model fits the data well

2.3.7.3 Alternative Models

- The **Negative-Binomial Model** for Count Data
- The **Quasi-Binomial** model
 - Replace the fixed dispersion parameter of 1 in the binomial distribution with a dispersion parameter ϕ to be estimated from the data.
 - Overdispersed binomial data can arise when
 - * Different individuals who share the same values of the explanatory variables nevertheless differ in their probability μ of success (*unmodelled heterogeneity*)
 - * Binomial observations are not independent (e.g., each binomial observation is for related

individuals, such as members of a family)

- The **Quasi-Poisson** Model for Count Data

- Introduce a dispersion parameter into the Poisson model
- The conditional variance of the response is $V(Y_i|\eta_i) = \phi\mu_i$
- If $\phi > 1$, therefore, the conditional variance of Y increases more rapidly than its mean
- Does not give a probability distribution for Y_i
- Cannot be estimated by maximum likelihood.
- Quasi-likelihood estimators of the regression coefficients

- **Zero-Inflated Poisson** Regression

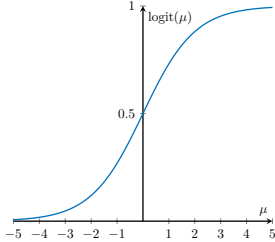
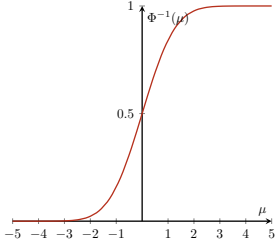
- Occurs when
 - * There are more zeroes in the data than is consistent with a Poisson (or negative-binomial) distribution
 - * Can arise when only certain members of the population are at risk of a nonzero count
- Two components:
 - * A binary logistic-regression model for membership in the latent class of individuals for whom the response variable is 0
 - * Poisson-regression model for the latent class of individuals for whom the response may be 0 or a positive count

2.3.8 Loglinear Models for Contingency Tables

2.3.9 Common distributions with typical uses and canonical link functions for GLM

Model	Random $Y X \sim$	Link $g(\mu) = \mathbf{X}\beta$	Mean $\mu(\theta)$	Variance $V(\mu)$	$a(\phi)$	Uses
Linear	$N(\mu, \sigma^2)$	u (Identity)	$\mathbf{X}\beta$	1	σ^2	$(-\infty, +\infty)$
Gamma	$\text{Gamma}(\mu, \psi)$	$\frac{1}{\mu}$ (Inverse)	$\frac{1}{\mathbf{X}\beta}$	μ^2	$\frac{1}{\psi}$	$(0, \infty)$
Inverse-Gaussian	$\text{IG}(\mu, \psi^2)$	$\frac{1}{\mu^2}$ (Inverse-square)	$\frac{1}{\sqrt{-2(\mathbf{X}\beta)}}$	μ^3	ψ^2	$(0, \infty)$
Logistic	$\text{Bern}(\mu)$	$\text{logit}(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$	$\frac{\exp(\mathbf{X}\beta)}{1+\exp(\mathbf{X}\beta)}$	$\mu(1-\mu)$	1	0/1
Logistic	$\text{Binomial}(n, \mu)$	$\text{logit}(\mu)$	$\frac{\exp(\mathbf{X}\beta)}{1+\exp(\mathbf{X}\beta)}$	$\frac{\mu(1-\mu)}{n}$	1	[0,1]
Probit		$\Phi^{-1}(\mu)$	$\Phi(\mathbf{X}\beta)$			[0,1]
Log-log		$-\ln[-\ln(\mu)]$	$\exp[-\exp(-\mathbf{X}\beta)]$			[0,1]
Complementary log-log		$-\ln[-\ln(1-\mu)]$	$1 - \exp[-\exp(-\mathbf{X}\beta)]$			[0,1]
Multinomial	$\text{Multinomial}(n, \pi)$ $\pi_j = P(Y = j X)$	$\log \frac{\pi_j}{\pi_i}$				Categorical
Possion	$\text{Pois}(\mu)$	$\ln \mu$	$\exp(\mathbf{X}\beta)$	μ	1	0, 1, 2, ...
Negative binomial	$\text{NB}(\mu, k)$	$\log\left(\frac{\mu}{k+\mu}\right)$	$\frac{ke^{(\mathbf{X}\beta)}}{1-e^{(\mathbf{X}\beta)}}$	$\mu + \frac{\mu^2}{k}$	1	0, 1, 2, ...
Loglinear		$\ln \mu$	$\exp(\mathbf{X}\beta)$	μ		Contingency table

2.3.10 Logit vs. Probit

	Logit	Probit
Y	Binary	Binary
\hat{Y}	$[0,1]$	$[0,1]$
Link	$\text{logit}(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$ 	Inverse Normal function: $\Phi^{-1}(\mu)$ 
	Slightly flatter tails	Approach the axes more quickly
Estimation	MLE	MLE
Error assumptions	A standard logistic distribution with a mean of 0 and a variance of $\pi^2/3$	Standard normal distribution with a mean of 0 and a variance of 1
Interpretations for β_j	Log-odds ratio: how the log-odds changes with a unit change in the predictor	A one-unit increase in X_j corresponds to an β_j increase in the z-score for probability of Y
Interpretations for e^{β_j}	Odds ratio: the multiplicative effect on the odds of increasing X_j by 1, while holding other X s constant	N/A
Applications	More popular in health sciences like epidemiology partly because coefficients can be interpreted in terms of odds ratios	Can be generalized to account for non-constant error variances in more advanced econometric settings (heteroskedastic probit models) and hence are used in some contexts by economists and political scientists
Good when	Extreme independent variables: independent variables where one particularly large or small value will overwhelmingly often determine whether the dependent variable is a 0 or a 1, overriding the effects of most other variables	Random effects models with moderate or large sample sizes

References:

<http://www.statslab.cam.ac.uk/~pat/All.pdf>

<http://www.stat.cmu.edu/~ryantibs/advmethods/notes/glm.pdf>

https://www.sagepub.com/sites/default/files/upm-binaries/21121_Chapter_15.pdf

<https://academic.macewan.ca/burok/Stat378/notes/glm.pdf>

<https://data.princeton.edu/wws509/notes/c5.pdf>

<http://www.marlenemueller.de/publications/HandbookCS.pdf>

<https://towardsdatascience.com/regression-analysis-generalised-linear-model-2f03c7e4cecb>

http://rstudio-pubs-static.s3.amazonaws.com/5691_192685385fc445c9b3fb1619960a20e2.html

Adekanmbi, D.B., 2017. Comparison of Probit and Logit Models for Binary Response Variable with Applications to Birth Data in South-Western, Nigeria. *American Journal of Mathematics and Statistics*, 7(5), pp.199–208.

2.4 GAM

$$\mathbb{E}(y_i|x_i) = \beta_0 + r_1(x_{i1}) + r_2(x_{i2}) + \dots + r_p(x_{ip})$$

where each r_j is an arbitrary (univariate) regression function, $j = 1, \dots, p$

References:

2.5 Linear Discriminant Analysis (LDA)

2.5.1 Linear Methods for Classification

To know the class posteriors $\Pr(G|X)$ for optimal classification:

- $f_k(x)$ is the class-conditional density of X in class $G = k$
 - Gaussian densities: linear and quadratic discriminant analysis
 - more flexible mixtures of Gaussians allow for nonlinear decision boundaries
 - Naive Bayes models assume that each of the class densities are products of marginal densities; that is, they assume that the inputs are conditionally independent in each class
- π_k : the prior probability of class k with $\sum_{k=1}^K \pi_k = 1$
- Apply the Bayes theorem:

$$\Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^K f_{\ell}(x)\pi_{\ell}}$$

2.5.2 Linear Discriminant Analysis

Supervised

Dimension Reduction

Classification

LDA, normal discriminant analysis (NDA), or discriminant function analysis finds a linear combination of features that yields the largest mean differences between the desired classes. LDA finds the boundaries around clusters of classes. It projects data points on a line so that clusters are as separated as possible, with each cluster having a relative (close) distance to a centroid.

LDA is a supervised algorithm as it takes the class label into consideration.

It is most commonly used as dimensionality reduction technique while at the same time preserving as much of the class discrimination information as possible.

2.5.2.1 Linear Discriminant Analysis as a restricted Gaussian classifier

Specifically, for LDA, model each class density as multivariate Gaussian

$$f_k(x) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}$$

and assume that the classes have a common covariance matrix $\Sigma_k = \Sigma \forall k$.

In comparing two classes k and ℓ , look at the log-ratio (*log-odds*), and the equation is linear in x :

$$\begin{aligned} \log \frac{\Pr(G = k|X = x)}{\Pr(G = \ell|X = x)} &= \log \frac{f_k(x)}{f_\ell(x)} + \log \frac{\pi_k}{\pi_\ell} \\ &= \log \frac{\pi_k}{\pi_\ell} - \frac{1}{2} (\mu_k + \mu_\ell)^T \Sigma^{-1} (\mu_k - \mu_\ell) + x^T \Sigma^{-1} (\mu_k - \mu_\ell) \end{aligned}$$

This linear log-odds function implies that the decision boundary between classes k and ℓ is linear in x ; in p dimensions a hyperplane.

Estimate the linear discriminant functions $\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$ using training data.

2.5.2.2 Derivations

Within-class scatter matrix:

$$S_w = \sum_{k=1}^K \sum_{g_i=k} (x_i - \hat{\mu}_k) (x_i - \hat{\mu}_k)^T$$

Between-class scatter matrix:

$$S_b = \sum_{k=1}^K (\hat{\mu}_k - \hat{\mu}) (\hat{\mu}_k - \hat{\mu})^T$$

- $\hat{\pi}_k = N_k/N$, where N_k is the number of class- k observations

- $\hat{\mu}_k = \sum_{g_i=k} x_i / N_k$ is the mean of class k
- $\hat{\mu}$ represents the mean of all classes
- $\hat{\Sigma} = \sum_{k=1}^K \sum_{g_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T / (N - K)$

Construct the lower dimensional space which maximizes the between-class variance and minimizes the within-class variance using **Fisher's criterion**:

$$\arg \max_W \frac{W^T S_B W}{W^T S_W W}$$

$$S_W W = \lambda S_B W$$

where λ represents the eigenvalues of the **transformation matrix** $W = S_W^{-1} S_B$ if S_W is non-singular

2.5.2.3 Time complexity

In class-independent method, the computational complexity is $O(NM^2)$ if $N > M$ (the number of features of X); otherwise the complexity is $O(M^3)$.

2.5.2.4 Pros

- Fundamental dimension reduction in LDA: need only consider the data in a subspace of dimension at most $K - 1$
- Can be used with small sample sizes (as compared to logistic regression)
- When sample sizes are equal, and homogeneity of variance/covariance holds, discriminant analysis is more accurate than logistic regression
- Optimal if and only if the classes are **Gaussian** and have **equal covariance**

2.5.2.5 Cons

- Assume that the independent variables are **normally distributed**
 - LDA's applicability extends beyond the realm of Gaussian data. However the derivation of the particular **intercept or cut-point** does require Gaussian data.
- Small Sample size

- A low number of training samples available for each class compared with the dimensionality of the sample space
- S_W is singular
- Solutions:
 - * Regularization
 - * Sub-space (e.g., PCA + LDA):
 - A non-singular intermediate space is obtained to reduce the dimension of the original data to be equal to the rank of S_W ; hence, S_W becomes full-rank, and then S_W can be inverted.
 - Lose some discriminant information
 - * Null space (Null LDA):
 - Remove the null space of S_W to make S_W full-rank; hence, invertible.
 - The drawback of this method is that more discriminant information is lost when the null space of S_W is removed
- Linearity problems
 - If the classes are **non-linearly separable**, LDA can not find a lower dimensional space
 - * LDA fails to find the LDA space when the discriminatory information are not in the *means* of classes (could be in the variance)
 - If the means of the classes are approximately equal, so the S_B and W will be zero. Hence, the LDA space cannot be calculated
 - * Solution: kernel
 - In kernel LDA, all samples are transformed non-linearly into a new space \mathcal{Z} using the function ϕ . The ϕ function is used to map the original features into \mathcal{Z} space by creating a non-linear combination of the original samples using a dot-products of it.

2.5.3 LDA vs. PCA

LDA	PCA
Supervised learning	Unsupervised learning
Separate data	Summarize data
Identify attributes that account for the most variance between classes	Identify the combination of attributes (principal components, or directions in the feature space) that account for the most variance in the data
A special case of GBM in terms of loss function (???)	More general in nature
Linear transformation	Linear transformation
Normally distributed classes and equal class covariances	
Find a subspace which maximizes the ratio of inter-class and intra-class variability	No distinction between inter-class and intra-class

2.5.4 LDA vs. LR

2.5.5 Quadratic Discriminant functions (QDA)

If the Σ_k are not assumed to be equal, the pieces quadratic in x remain.

- Separate covariance matrices must be estimated for each class
- When p is large: a dramatic increase in parameters

References

https://sebastianraschka.com/Articles/2014_python_lda.html

<https://cse.buffalo.edu/~jcorso/t/555pdf/pca-versus-lda.pdf>

<http://www.svcl.ucsd.edu/courses/ece271B-F09/handouts/Dimensionality2.pdf>

<http://www.sci.utah.edu/~shireen/pdfs/tutorials/Elhabian-LDA09.pdf>

Tharwat, A. et al., 2017. Linear discriminant analysis: A detailed tutorial. , 30(2)), pp.169–190,.

2.6 Naïve Bayes

classification generative The Naive Bayes algorithm is a classification algorithm based on Bayes rule and a set of **conditional independence assumptions**.

2.6.1 Bayes Rule

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}$$

- $P(h)$: prior probability
- $P(D|h)$: likelihood
- $P(h|D)$: posterior probability

2.6.2 Conditional Independence Assumption

Given the goal of learning $P(Y|X)$ where $X = \langle X_1, \dots, X_{ni} \rangle$, the Naive Bayes algorithm makes the assumption that each X_i is conditionally independent of each of the other X_k s given Y , and also independent of each subset of the other X_k s given Y .

Naive Bayes models assume that each of the class densities are products of marginal densities; that is, they assume that the inputs are conditionally independent in each class.

Consider the case where $X = \langle X_1, X_2 \rangle$. In this case

$$\begin{aligned} P(X|Y) &= P(X_1, X_2|Y) \\ &= P(X_1|X_2, Y)P(X_2|Y) \\ &= P(X_1|Y)P(X_2|Y) \end{aligned}$$

More generally, when X contains n attributes which satisfy the conditional independence assumption, we have

$$P(X_1, \dots, X_n|Y) = \prod_{i=1}^n P(X_i|Y)$$

2.6.3 Derivation

Assume in general that Y is any discrete-valued variable, and the attributes X_1, \dots, X_{ni} are any discrete or realvalued attributes. Our goal is to train a classifier that will output the probability distribution over

possible j values of Y , for each new instance X that we ask it to classify.

The expression for the probability that Y will take on its k th possible value, according to Bayes rule, is:

$$\begin{aligned} P(Y = y_k | X_1, \dots, X_n) &= \frac{P(Y = y_k) P(X_1, \dots, X_n | Y = y_k)}{\sum_j P(Y = y_j) P(X_1, \dots, X_n | Y = y_j)} \\ &= \frac{P(Y = y_k) \prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \prod_i P(X_i | Y = y_j)} \end{aligned}$$

Given a new instance $X^{\text{new}} = \langle X_1, \dots, X_n \rangle$, this equation shows how to calculate the probability that Y will take on any given value, given the observed attribute values of X^{new} and given the distributions $P(Y)$ and $P(X_i | Y)$ estimated from the training data.

If we are interested only in the **most probable** value of Y , then we have the Naive Bayes classification rule:

$$Y \leftarrow \operatorname{argmax}_{y_k} \frac{P(Y = y_k) \prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \prod_i P(X_i | Y = y_j)}$$

which simplifies to the following (because the denominator does not depend on y_k).

$$Y \leftarrow \operatorname{argmax}_{y_k} P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

2.6.4 Properties

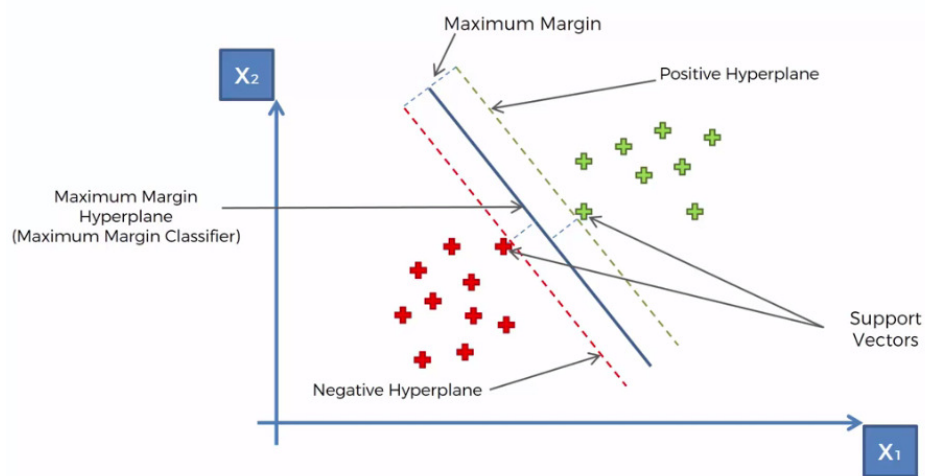
- Estimating $P(X_i | Y = y_k)$ instead of $P(X_1, \dots, X_n | Y = y_k)$ greatly reduces the number of parameters ($2^n \rightarrow 2n$) and data sparseness
- The learning step in Naive Bayes consists of estimating $P(X_i | Y = y_k)$ and $P(Y = y_k)$ based on the **frequencies in the training data**.
- There is no explicit search during training (as opposed to decision trees).
- An unseen instance is classified by computing the class that maximizes the posterior.
- Incrementality: with each training example, the prior and the likelihood can be updated dynamically: flexible and robust to errors.
- Combines prior knowledge and observed data: prior probability of a hypothesis multiplied with probability of the hypothesis given the training data.
- Probabilistic hypotheses: outputs not only a classification, but a probability distribution over all classes.
- Meta-classification: the outputs of several classifiers can be combined, e.g., by multiplying the probabilities that all classifiers predict for a given class.

- Fast to train (single scan); Fast to classify (good for lots of data)
- Not sensitive to irrelevant features
- Handles real and discrete data

2.7 SVM

2.7.1 Maximal Margin Classifier

Maximal margin hyperplane (also known as the **optimal separating hyperplane**), which is the separating hyperplane that is **farthest from the training observations**. It seeks the largest possible margin so that every observation is not only on the **correct side of the hyperplane** but also on the **correct side of the margin**.



2.7.1.1 Hyperplane

In a p -dimensional space, a **hyperplane** is a flat affine subspace of hyperplane dimension $p - 1$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p = 0$$

defines a p -dimensional hyperplane.

If a point $X = (X_1, X_2, \dots, X_p)^\top$ in p -dimensional space (i.e. a vector of length p) satisfies above equation, then X lies on the hyperplane.

If $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p > 0$, X lies to one side of the hyperplane.

If $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p < 0$, then X lies on the other side of the hyperplane.

2.7.1.2 Margin

We can compute the (perpendicular) distance from each training observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the **margin**.

Margin:

It is the distance of the separating hyperplane to the closest examples in the dataset, assuming that the dataset is linearly separable.

The **maximal margin hyperplane** is the separating hyperplane for which the margin is largest—that is, it is the hyperplane that has the **farthest minimum distance to the training observations**. We can then classify a test observation based on which side of the maximal margin hyperplane it lies.

2.7.1.3 Derivation

N pairs of training data with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. Define a hyperplane by

$$\{x : f(x) = x^T \beta + \beta_0\}$$

where β is a unit vector: $\|\beta\| = \sqrt{\beta^T \beta} = 1$ (interested only in the direction and not its length).

A classification rule induced by $f(x)$ is

$$G(x) = \text{sign}[x^T \beta + \beta_0]$$

The convex optimization problem (quadratic criterion, linear inequality constraints):

$$\max_{\beta, \beta_0, \|\beta\|=1} M$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq M, i = 1, \dots, N$$

is equivalent to:

$$\min_{\beta, \beta_0} \|\beta\|$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq 1, i = 1, \dots, N$$

is equivalent to:

$$\min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2$$

$$\text{subject to } y_i(x_i^T \beta + \beta_0) \geq 1, i = 1, \dots, N$$

2.7.1.4 Loss Function View

$$l(t) = \begin{cases} 0, & \text{if } t \geq 1 \\ \infty, & \text{if } t < 1 \end{cases}$$

2.7.1.5 Support Vectors

- Training observations are equidistant ($M = 1/||\beta||$) from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin ($2M$)
- They are vectors in p -dimensional space
- They “support” the maximal margin hyperplane in the sense vector that if these points were moved slightly then the maximal margin hyperplane would move as well
- Maximal margin hyperplane depends directly on the support vectors, but not on the other observations: a movement to any of the other observations would not affect the separating hyperplane, provided that the observation’s movement does not cause it to cross the boundary set by the margin

2.7.1.6 Cons

- Over-fitting when p is large
- Maximal margin hyperplane is extremely sensitive to a change in a single observation
- When no separating hyperplane exists, no maximal margin classifier

2.7.2 Support Vector Classifier/Soft Margin Classifier

2.7.2.1 Why SVC

- Observations that belong to two classes are not necessarily separable by a hyperplane
- Even if a separating hyperplane does exist, then there are instances in which a classifier based on a separating hyperplane might not be desirable
- SVC allows some observations to be on **the incorrect side of the margin**, or even the **incorrect side of the hyperplane**
- The fact that the support vector classifier's decision rule is based only on **the support vectors** means that it is **quite robust to the behavior of observations that are far away from the hyperplane**

Soft margin:

A hyperplane that does not perfectly separate the two classes, in the interest of

- Greater robustness to individual observations
- Better classification of most of the training observations

Linear kernel (Pearson correlation):

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij} x_{i'j}$$

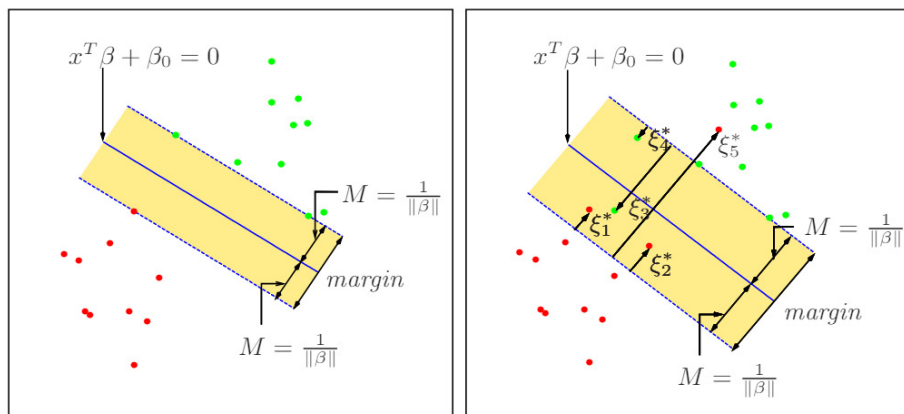


FIGURE 12.1. Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width $2M = 2/\|\beta\|$. The right panel shows the nonseparable (overlap) case. The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$; points on the correct side have $\xi_j^* = 0$. The margin is maximized subject to a total budget $\sum \xi_i \leq \text{constant}$. Hence $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin.

2.7.2.2 Soft Margin SVM: Geometric View

Define the **slack variables** $\xi = (\xi_1, \xi_2, \dots, \xi_N)$.

Modify the constraint:

$$\begin{aligned} & \max_{\beta, \beta_0, \|\beta\|=1} M \\ \text{subject to } & \begin{cases} y_i(x^T \beta + \beta_0) \geq M(1 - \xi_i), i = 1, \dots, N \\ \forall i, \xi_i \geq 0 \\ \sum_{i=1}^N \xi_i \leq C \end{cases} \end{aligned}$$

By bounding $\sum_{i=1}^N \xi_i \leq C$, the total proportional amount by which predictions fall on the wrong side of the margin were bounded.

- On the wrong side of the margin: $\xi_i > 0$
- Misclassifications (on the wrong side of the hyperplane): $\xi_i \geq 1$

Define $M = \frac{1}{\|\beta\|}$:

$$\begin{aligned} & \min_{\beta_0, \beta} \|\beta\| \\ \text{subject to } & \begin{cases} y_i(x^T \beta + \beta_0) \geq 1 - \xi_i, \forall i \\ \xi_i \geq 0 \\ \sum \xi_i \leq C \end{cases} \end{aligned}$$

Equivalent to:

$$\begin{aligned} & \min_{\beta_0, \beta} \frac{1}{2} \|\beta\|^2 \\ \text{subject to } & \begin{cases} y_i(x^T \beta + \beta_0) \geq 1 - \xi_i, \forall i \\ \xi_i \geq 0 \\ \sum \xi_i \leq C \end{cases} \end{aligned}$$

2.7.2.3 Tuning parameter C

- Bounds the sum of the ξ_i and determines the number and severity of the violations to the margin and to the hyperplane
- Generally chosen via cross-validation

- $C \uparrow \rightarrow$ More tolerant of violations to the margin \rightarrow Wider margin \rightarrow High bias but low variance
- $C \downarrow \rightarrow$ Less tolerant of violations to the margin \rightarrow Narrow margin \rightarrow Low bias but high variance

2.7.2.4 Soft Margin SVM: Loss Function View

Hinge loss

$$l(t) = \max\{0, 1 - t\}$$

$$t = y_i f(x) = y_i (x^T \beta + \beta_0)$$

- $f(x)$ is on the correct side of the hyperplane and further than distance 1 ($t > 1$): $l(t) = 0$
- $f(x)$ is on the correct side but close to the hyperplane ($0 < t < 1$): x is within the margin and $l(t) > 0$
- $f(x)$ is on the wrong side of the hyperplane ($t < 0$): the hinge loss $l(t)$ returns an even larger value, which increases linearly

Unconstrained optimization problem:

$$\min_{\beta_0, \beta} \underbrace{\frac{1}{2} \|\beta\|^2}_{\text{regularizer}} + C \underbrace{\sum_{i=1}^N \max\{0, 1 - y_i (x^T \beta + \beta_0)\}}_{\text{error term}}$$

2.7.3 Support Vector Machine

When the support vector classifier is combined with a **non-linear kernel**, the resulting classifier is known as a support vector machine.

$$\{x : f(x) = \sum \alpha_i K(x, x_i) + \beta_0\}$$

An extension of the support vector classifier that results from **enlarging the feature space** using **kernels** for nonlinear boundaries. It produces **nonlinear** boundaries by constructing a linear boundary in a large, transformed version of the feature space.

2.7.3.1 Kernel

Kernel is a function that quantifies the **similarity of two observations**

$$K(x_i, x'_i)$$

- Linear kernel: $K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij}x_{i'j}$
- Polynomial kernel of degree d : $K(x_i, x_{i'}) = (1 + \sum_{j=1}^p x_{ij}x_{i'j})^d = (1 + \langle x, x' \rangle)^d$
- Radical Basis Function (RBF) kernel:

$$\exp(-\gamma \sum_{j=1}^p (x_{ij} - x_{i'j})^2) = \exp(-\gamma \|x - x'\|^2)$$

– $\gamma \uparrow \implies$ Small bias \implies Overfitting

– Gaussian kernel: $\exp(-\frac{\|x-x'\|^2}{\delta^2})$

- Sigmoid Kernel: $\tanh(\kappa_1 \langle x, x' \rangle) + \kappa_2$

2.7.3.2 Why using kernel rather than enlarging feature space using functions of the original features

- Computational: using kernels, one need only compute $K(x_i, x_{i'})$ for all $\binom{n}{2}$ distinct pairs i, i' in the original space, which avoids curse of dimensionality
- Don't have to know the mapping function Φ
- Not all functions can be kernels:
 - Must make sure there is a corresponding Φ in some high-dimensional space

2.7.3.3 Pros

- Convex optimization problem: efficient algorithms are available to find the global minima of the objective function
- Overfitting is addressed by maximizing the margin of the decision boundary, but the user still needs to provide the type of kernel function and cost function
- Robust to noise

2.7.3.4 Cons

- Difficult to handle missing values
- High computational complexity for building the model

References

<http://mml-book.com> <https://mml-book.github.io/book/chapter12.pdf> Draft chapter (September 3, 2018) from “Mathematics for Machine Learning” c 2018 by Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. To be published by Cambridge University Press.

<http://cs229.stanford.edu/notes/cs229-notes3.pdf>

Todo:

Dual and primal form

Comparison between LR, Tree and SVM

How to choose kernels

what parameters you will need to tune during model training SVM

2.8 kNN

2.8.1 Summary

Nearest-neighbor methods use those observations in the training set \mathcal{T} closest in input space to x to form \hat{Y} . Find the k observations with x_i closest to x in input space, and average their responses.

The k -nearest neighbor fit for \hat{Y} is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

- $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample
- Closeness: Euclidean distance
- Attempt to directly implement $\hat{f}(x) = \text{Ave}(y_i | x_i \in N_k(x))$ using the **training** data
- Assume $f(x)$ is well approximated by a **locally constant** function
- Effective number of parameters of k -nearest neighbors: N/k
 - Generally bigger than p in least squares fits
 - If the neighborhoods were non-overlapping, there would be N/k neighborhoods and fit one parameter (a mean) in each neighborhood
- For k -nearest-neighbor fits, the error on the training data should be approximately an increasing function of k , and will always be 0 for $k = 1$
 - Cannot use sum-of-squared errors on the training set as a criterion for picking k , since we would always pick $k = 1$
 - Less stable than least-squares fits
- Large $k \rightarrow$ Simple model \rightarrow Under fit \rightarrow Low variance & High bias
- Small $k \rightarrow$ Complex model \rightarrow Over-fit \rightarrow High variance & Low bias

2.8.2 Pros

- Not rely on any stringent assumptions about the underlying data, and can adapt to any situation
- Less bias compared with least-squares fit

2.8.3 Cons

- Any particular sub-region of the decision boundary depends on a handful of input points and their particular positions, and is thus wiggly and unstable high variance and low bias
- If the dimension of the input space is high, the nearest neighbors need not be close to the target point (???), and can result in large errors

2.9 Classification and Regression Tree (CART)

2.9.1 Pros

- Interpretability: Trees are very easy to explain to people
- Trees can be displayed graphically, and are easily interpreted even by a non-expert
- More closely mirror human decision-making than do the regression and classification
- Trees can easily handle qualitative predictors without the need to create dummy variables

2.9.2 Cons

- **High variance:** often a small change in the data can result in a very different series of splits
 - Due to the hierarchical nature of the process: the effect of an error in the top split is propagated down to all of the splits below it.
 - Can alleviate this to some degree by trying to use a more stable split criterion, but the inherent instability is not removed.
 - **Bagging** averages many trees to reduce this variance
- Do not have the same level of predictive accuracy as some of the other regression and classification approaches
- Lack of smoothness
 - Classification with 0/1 loss: doesn't hurt much, since bias in estimation of the class probabilities has a limited effect
 - Regression: this can degrade performance
 - MARS: alleviate this lack of smoothness
- Difficulty in capturing additive structure
 - Alternative: MARS
-

2.9.3 Trees vs. Linear Models

- Trees
 - $f(X) = \sum^M c_m \cdot 1_{(X \in R_m)}$ where R_1, \dots, R_M represent a partition of feature space

- Highly non-linear and complex relationship between the features: decision trees may outperform classical approaches
- Preferred for the sake of interpretability and visualization
- Linear models
 - $f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j$
 - If the relationship between the features and the response is well approximated by a linear model: linear regression will likely work well, and will outperform regression tree that does not exploit this linear structure

2.9.4 CART: Regression Trees

Tree-based methods (for regression) partition the feature space into **box-shaped regions**, to try to make the response averages in each box **as different as possible**. The splitting rules defining the boxes are related to each through a **binary tree**, facilitating their interpretation. The predicted response for an observation is given by the **mean response of the training observations** that belong to the same terminal node.

DATA: p inputs and a response, for each of N observations: (x_i, y_i) for $i = 1, 2, \dots, N$, with $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$

GOAL: Automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have

2.9.4.1 Greedy Algorithm: Recursive Binary Splitting

- Top-down:
 - Begin at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree
- Greedy
 - At each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step
- Why greedy algorithm
 - Finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible

1. Starting with all of the data, consider a splitting variable j and split point s , and define the pair of **half-planes**

$$R_1(j, s) = \{X|X_j \leq s\} \text{ and } R_2(j, s) = \{X|X_j > s\}$$

2. Seek the splitting variable j and split point s that minimize the RSS (lead to the greatest possible reduction in RSS):

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \quad \text{and} \quad \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s))$$

3. For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible.
4. Having found the best split, partition the data into the two resulting regions and repeat the splitting process on each of the two regions
5. This process is repeated on all of the resulting regions

2.9.4.2 Tree Size: Cost-complexity Pruning

- A tuning parameter governing the model's complexity
- The optimal tree size should be adaptively chosen from the data

1. Grow a large tree T_0 , stopping the splitting process only when some minimum node size is reached

2. Large tree is pruned using COST-COMPLEXITY PRUNING/WEAKEST LINK PRUNING

- A **subtree** $T \subset T_0$ to be any tree that can be obtained by pruning T_0 : collapsing any number of its internal (non-terminal) nodes
- Terminal nodes: m
- Node m representing region R_m
- $|T|$: the number of terminal nodes in T
- Squared-error node impurity measure $Q_m(T)$:

$$N_m = \# \{x_i \in R_m\}$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$$

- Cost complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

- For each α , find the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$
 - The tuning parameter $\alpha \geq 0$: governs the tradeoff between tree size and its goodness of fit to the data
 - Large values of α result in smaller trees T_α
 - $\alpha = 0$: the solution is the full tree T_0
 - Estimation of α : achieved by five- or tenfold cross-validation
 - Choose the value $\hat{\alpha}$ to **minimize** the cross-validated sum of squares
 - As increase α from zero, branches get pruned from the tree in a nested and predictable fashion, so obtaining the whole sequence of subtrees as a function of α is easy

2.9.5 CART: Classification Trees

Predict each observation belongs to the **most commonly occurring class** of training observations in the region to which it belongs.

- Node m in a representing a region R_m with N_m observations
- Proportion of class k observations in node m :

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

- Classify the observations in node m to **the majority class in node m** : $k(m) = \arg \max_k \hat{p}_{mk}$

2.9.5.1 Measures of Node Impurity

$Q_m(T)$

- **Misclassification error:** $\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$
 - Typically used to guide cost-complexity pruning
 - Preferable if prediction accuracy of the final pruned tree is the goal
- **Gini index:**
 - Differentiable
 - More sensitive to changes in the node probabilities than the misclassification rate
 - Should be used when growing/pruning the tree
 - A small value indicates that a node contains predominantly observations from a single class
 - Random rule:
$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$
 - Need to figure out name:
 - * Classify observations to class k with probability \hat{p}_{mk} : the training error rate of this rule in the node is $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'}$
 - A measure of total variance:
 - * If we code each observation as 1 for class k and zero otherwise, the variance over the node of this 0 – 1 response is $\hat{p}_{mk} (1 - \hat{p}_{mk})$. Summing over classes k again gives the Gini index
- **Cross-entropy or deviance:** $-\sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$

- Differentiable
- More sensitive to changes in the node probabilities than the misclassification rate
- Should be used when growing/pruning the tree

2.9.5.2 Loss Matrix

- $L_{kk'}$: the loss incurred for classifying a class k observation as class k'
- No loss is incurred for correct classifications: $L_{kk} = 0 \forall k$
- Gini index
 - Multiclass: $\sum_{k \neq k'} L_{kk'} \hat{p}_{mk} \hat{p}_{mk'}$
 - Two classes: to weight the observations in class k by $L_{kk'}$

2.9.5.3 Missing Predictor Values

- Categorical variables
 - Make a new category for missing
 - Construction of **surrogate variables**: When considering a predictor for a split, use only the observations for which that predictor is not missing
- Continuous variables

TODO:

- how to handle missing data

2.10 MARS: Multivariate Adaptive Regression Splines

3 Ensemble

Ensemble learning can be broken down into two task: developing a population of base learners from the training data, and then combining them to form the composite predictor.

3.1 Bagging /Bootstrap Aggregation

- For **reducing the variance** of an estimated prediction function
- Work especially well for **high-variance, low-bias procedures**, such as trees
- Why can bootstrap reduce variance
 - Given a set of n independent observations Z_1, \dots, Z_n , each with variance σ^2 , the variance of the mean \bar{Z} of the observations is given by σ^2/n
 - AVERAGING A SET OF OBSERVATIONS REDUCES VARIANCE.

Bootstrap by taking repeated samples from the (single) training data set \rightarrow Generate B different bootstrapped training data sets \rightarrow Train our method on the b th bootstrapped training set \rightarrow Get $\hat{f}^{*b}(x)$ \rightarrow Average all the predictions, to obtain the bagging estimate:

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

3.2 Random Forest

bagging

3.2.1 Variance Reduction

The improvements in prediction obtained by bagging are **solely a result of variance reduction**:

Since each tree generated in bagging is **identically distributed (i.d.)**, the expectation of an average bias of B such trees is the same as the expectation of any one of them. This means the **bias of bagged trees is the same as that of the individual trees**, and the only hope of improvement is through **variance reduction**.

This is in contrast to **boosting**, where the trees are grown in an adaptive way to **remove bias**, and hence are not i.d.

RF improves the **variance reduction** of bagging by **reducing the correlation** between the trees, without increasing the variance too much. This is achieved in the tree-growing process through **random selection of the input variables**.

3.2.2 Algorithm

- N : the number of cases in the training set
- p : the number of input variables
- m : a constant ($m \ll p$) to sample at each node

Each tree is grown as follows:

1. **Observation bagging**

- Sample N cases at random with replacement (bootstrap) from the original data

2. Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{\min} is reached

- **Feature bagging**

- Select m variables at random out of the p and pick the best variable/split-point among the m
- * Consider only a subset of predictors at a split

- * This results in trees with different predictors at top split, thereby resulting in **decorrelated** trees and more reliable average output
 - * That's why we say random forest is **robust to correlated predictors**
 - * When we have a large number of correlated predictors: using a small value of m
 - Bagging: the special case of random forests obtained when $m = p$
 - Each tree is grown to the **largest extent possible**
 - If grown sufficiently deep, trees have relatively low bias
 - No pruning
3. Output the ensemble of trees $\{T_b\}_1^B$
 4. Predict new data by **aggregating** the predictions of the n_{tree} trees for regression ($\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$) or **majority votes** for classification

3.2.3 OOB (Out-of-Bag) Estimate of Error Rate

3.2.3.1 Steps

- At each bootstrap iteration, predict the data not in the bootstrap sample (OOB data) using the tree grown with the bootstrap sample
- Aggregate the OOB predictions
 - On average, each data point would be out-of-bag around 36% of the times
- Calculate the error rate
 - OOB error
 - * A valid estimate of the test error for the bagged model
 - * An OOB error estimate is almost identical to that obtained by N -fold cross-validation
- RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

3.2.3.2 The Forest Error Rate Depends on Two Things

- The **correlation between any two trees** in the forest
 - Increasing the correlation increases the forest error rate

- The **strength of each individual tree** in the forest
 - A tree with a **low error rate** is a strong classifier
 - Increasing the strength of the individual trees decreases the forest error rate
- Reducing m reduces both the correlation and the strength. Increasing it increases both.
- m is the only adjustable parameter to which random forests is somewhat sensitive.
- Regression tree
 - $m = p/3$
 - Minimum node size = 5
- Classification tree
 - $m = \sqrt[3]{p}$
 - Minimum node size = 1

3.2.4 Variable Importance Measures

3.2.4.1 Method 1

At each split in each tree, the **improvement in the split-criterion is the importance measure** attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.

- Regression
 - RSS
 - Record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees
 - A large value indicates an important predictor
- Classification
 - Gini index
 - Add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees

3.2.4.2 Method 2: Measure the Prediction Strength of Each Variable?

When the b th tree is grown, the OOB samples are passed down the tree, and the prediction accuracy is recorded. Then the values for the j th variable are randomly permuted in the OOB samples, and the accuracy is again computed. The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable j in the random forest.

3.2.5 Proximities?

After each tree is built, all of the data are run down the tree, and **proximities** are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are **normalized** by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

3.2.6 Parameter Tuning

- **node size** (**min samples leaf**)
 - Specify the minimum number of observations in a terminal node
 - Directly related to tree **depth**
 - Smaller node size allows for deeper, more complex trees
 - **Bias-variance trade-off**: deeper trees introduce more variance (risk of over fitting) and shallower trees introduce more bias (risk of not fully capturing unique patterns and relationships in the data)
- **m**
 - Number of randomly drawn candidate variables out of which each split is selected when growing a tree
 - Lower values of $m \rightarrow$ More different, less correlated trees \rightarrow Better stability when aggregating \rightarrow Better exploit variables with moderate effect on the response variable \rightarrow Worse performance on average (built based on suboptimal variables)
- **min samples split**:
 - Minimal size that child nodes should have for the split to be performed
- **ntree**
 - The number of trees to grow

- Should be set sufficiently high
 - * Enough trees to stabilize the error
 - * Using too many trees is unnecessarily inefficient, especially when using large data sets

3.2.7 Pros

- Robust to correlated predictors
- Solve both regression and classification problems
- Solve unsupervised ML problems
- Can handle thousands of input variables without variable selection
- Can be used as a feature selection tool using its variable importance plot.
- Takes care of missing data internally in an effective manner

3.2.8 Cons

- Difficult to interpret
- Tends to return erratic predictions for observations out of range of training data
- Can take longer than expected time to computer a large number of trees

3.3 Boosting

- Motivation: combine the outputs of many weak classifiers to produce a powerful committee (**reduce bias**)
 - **Weak classifier**: one whose error rate is only slightly better than random guessing
- Trees are grown **sequentially**
 - Each tree is grown using information from previously grown trees
 - * Unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown
 - Each tree is fit on a modified version of the original data set rather than bootstrap sampling
- Given the current model \rightarrow Fit a decision tree to the **residuals** from the model \rightarrow Add this new decision tree into the fitted function in order to update the residuals
- Each of these trees can be rather **small**, with just a few terminal nodes
 - By fitting small trees to the residuals, slowly improve \hat{f} in areas where it does not perform well

3.3.1 Three Tuning Parameters

1. The number of trees B
 - Unlike bagging and random forests, boosting can over-fit if B is too large
 - Over-fitting tends to occur slowly if at all
 - Cross-validation to select B
2. The shrinkage parameter λ
 - A small positive number
 - Control the rate at which boosting learns
 - Very small λ can require using a very large value of B to achieve good performance
3. The number d of splits (depth) in each tree
 - Controls the complexity of the boosted ensemble
 - Often $d = 1$ works well
 - In this case each tree is a **stump**: consisting of a single split

- The boosted ensemble is fitting an additive model, since each term involves only a single variable
- More generally d is the **interaction depth**, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

3.4 AdaBoost

boosting

3.4.1 Summary

The purpose of boosting is to **sequentially** apply the **weak classification algorithm** to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x)$, $M = 1, 2, \dots, M$, and the predictions from all of them are then **combined through a weighted majority vote** to produce the final prediction for $Y \in \{-1, 1\}$:

$$G(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right)$$

Where $\alpha_1, \alpha_2, \dots, \alpha_M$ are computed by the boosting algorithm, and **weight the contribution** of each respective $G_m(x) \in \{-1, 1\}$. α 's effect is to give **higher influence** to the **more accurate classifiers** in the sequence.

In Adaboost, shortcomings of weak learners are identified by high-weight data points.

AdaBoost is equivalent to forward stagewise additive modeling using the **exponential loss function**:

$$L(y, f(x)) = \exp(-yf(x))$$

3.4.2 Algorithm

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1, 2, \dots, N$
2. For $m = 1$ to M :
 - (a) Fit a classifier $G_m(x)$ to the training data using weights w_i
 - The observation weights are individually modified and the classification algorithm is reapplied to the weighted observations
 - (b) Compute $\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$
 - (c) Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
 - (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \dots, N$
 - **Misclassified** observations by the classifier $G_{m-1}(x)$ have their **weights increased**
 - Observations that are **difficult to classify correctly receive ever-increasing influence**
 - Each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence
 - The weights are decreased for those that were classified correctly
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$

3.4.3 Why Exponential Loss

- Computational: lead to a simple modular reweighting AdaBoost algorithm

3.5 Boosting Trees

3.5.1 A Single Tree as Base Learner

A tree can be expressed as:

$$T(x; \Theta) = \sum_{j=1}^J \gamma_j I(x \in R_j)$$

- $\Theta = \{R_j, \gamma_j\}_1^J$
- Tree partition the space of all joint predictor variable values into disjoint regions $R_j, j = 1, 2, \dots, J$, as represented by the terminal nodes of the tree
 - Difficult to find R_j : approximate solutions
- γ_j : a constant assigned to each such region
 - Given the R_j : estimate the γ_j is typically trivial
 - For regression: $\hat{\gamma}_j = \bar{y}_j$: the mean of the y_i falling in region R_j
- The predictive rule: $x \in R_j \Rightarrow f(x) = \gamma_j$
- J : a meta-parameter

The parameters are found by minimizing the empirical risk:

$$\hat{\Theta} = \arg \min_{\Theta} \sum_{j=1}^J \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

3.5.1.1 Why is tree an excellent base learner for boosting

- Trees have the flexibility to be weak learners by simply restricting their depth
- Separate trees can be easily added together, much like individual predictors can be added together in a regression model, to generate a prediction
- Trees can be generated very quickly

3.5.2 Boosting Trees

A sum of such trees induced in a forward stagewise manner:

$$f_M(x) = \sum_{m=1}^M T(x; \Theta_m)$$

At each step in the forward stagewise procedure one must solve:

$$\hat{\Theta}_m = \arg \min_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

3.5.3 Optimization via Gradient Descent

Induce a tree $T(x; \Theta_m)$ at the m th iteration whose predictions tree components $\mathbf{t}_m = T(x_1; \Theta_m), \dots, T(x_N; \Theta_m)$ are as **close as possible to the negative gradient**.

When using squared error, **one fits the tree T to the negative gradient values** by least squares:

$$\tilde{\Theta}_m = \arg \min_{\Theta} \sum_{i=1}^N (-g_{im} - T(x_i; \Theta))^2$$

3.5.4 Algorithms

1. Initialize $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$

2. For $m = 1$ to M :

(a) For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

(b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$

(c) For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

(d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

3. Output $\hat{f}(x) = f_M(x)$

3.5.5 Parameters Tuning

1. **Size of the constituent trees, J**

- Restrict all trees to be the same size, $J_m = J \forall m$

- A meta-parameter of the entire boosting procedure
- At each iteration a **J -terminal node regression tree** is induced
- J to be adjusted to maximize estimated performance for the data at hand by cross-validation
- $4 \leq J \leq 8$

2. Number of boosting iterations M

- Each iteration usually reduces the training risk $L(f_M)$
- Large $M \implies$ Overfitting
- Optimal number M^*
 - Monitor prediction risk as a function of M on a validation sample: the value of M that minimizes this risk is taken to be an estimate of M^* – **Early stopping**
- Regularization strategy

3. Learning Rate: Scale the contribution of each tree by a factor $0 < \nu < 1$

- Control the **learning rate** of the boosting procedure
 - Control how quickly the algorithm proceeds down the gradient descent
- Regularization strategy
- Both ν and M control prediction risk on the training data
- Smaller values of ν (more shrinkage) result in larger training risk for the same number of iterations M
 - Smaller values reduce the chance of overfitting but also increases the time to find the optimal fit
 - The best strategy: set ν to be very small ($\nu < 0.1$) and then choose M by early stopping

4. Bagging Fraction: Subsampling

- **Stochastic gradient boosting**
- Sample a fraction η of the training observations (without replacement) and grow the next tree using that subsample
- Regularization strategy
- A typical value for η can be $\frac{1}{2}$, although for large N , η can be substantially smaller than $\frac{1}{2}$
- Reduce the computing time

- In many cases it produces a more accurate model
- Help to minimize overfitting and keep from getting stuck in a local minimum or plateau of the loss function gradient (non-convex loss function)

3.5.6 Interpretations

3.5.6.1 Relative Importance of Predictor Variables

- For a single tree:

- Measure of relevance for each predictor variable X_ℓ

$$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} i_t^2 I(v(t) = \ell)$$

- At each such node t , one of the input variables $X_{v(t)}$ is used to partition the region associated with that node into two subregions; within each a separate constant is fit to the response values.
- The particular variable chosen is the one that gives **maximal estimated improvement** i_t^2 in squared error risk over that for a constant fit over the entire region
- The squared relative importance of variable X_ℓ : the **sum of such squared improvements over all internal nodes for which it was chosen as the splitting variable**
 - * Sum over the $J - 1$ internal nodes of the tree

- For boosting tree:

- The improvement in squared error due to each predictor is summed within each tree in the ensemble (i.e., each predictor gets an improvement value for each tree) . The improvement values for each predictor are then averaged across the entire ensemble to yield an overall importance value

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^M \mathcal{I}_\ell^2(T_m)$$

3.6 Gradient Boosting Machines: GBM

gradient descent boosting regression classification boosting trees

3.6.1 Summary

Given a **loss function** (e.g., squared error for regression) and a **weak learner** (e.g., regression trees), the algorithm seeks to find an additive model that **minimizes the loss function** by **gradient decent**. The algorithm is typically initialized with the best guess of the response (e.g., the mean of the response in regression). The **gradient (e.g., residual)** is calculated, and a model is then fit to the residuals to minimize the loss function. The current model is added to the **previous model**, and the procedure continues for a user-specified number of iterations. The learning procedure **consecutively** fits new models to provide a more accurate estimate of the response variable.

The loss functions applied can be arbitrary.

3.6.2 Algorithm

Inputs

- Input data $(x, y)_{i=1}^N$
- Number of iterations M
- Choice of loss-function $\Psi(y, f)$, $(\hat{f}(x) = y)$
- Choice of base-learner $h(x, \theta)$

Algorithm

- Initialize \hat{f}_0 with a constant
- For $t = 1$ to M iteration:
 - Compute the negative gradient $g_t(x)$
 - Fit a new base-learner function $h(x, \theta_t)$
 - Find the best gradient descent step size ρ_t
$$\rho_t = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^N \Psi \left[y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t) \right]$$
 - update the function estimate: $\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x_i, \theta_t)$
- End

3.6.3 Pros

- Often provides predictive accuracy that cannot be beat
- Lots of flexibility
 - Can optimize on different loss functions and provides several hyper-parameter tuning options that make the function fit very flexible
- No data pre-processing required
 - Often works great with categorical and numerical values as is
 - Handles missing data - imputation not required

3.6.4 Cons

- **Overfitting**
 - GBMs will continue improving to minimize all errors
 - The learner is tasked with optimally fitting the gradient
 - * Boosting will select the optimal learner at each stage of the algorithm
 - * Drawbacks of not finding the optimal global model as well as over-fitting the training data
 - * Subsampling, regularization/shrinkage
- **Computationally expensive**
- **Require a large grid search during tuning**
 - Number of iterations
 - Tree depth
 - Regularization parameters
- **Less interpretable** although this is easily addressed with various tools (variable importance, partial dependence plots, LIME, etc.).

References

Natekin, A. and Knoll, A., 2013. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7, p.21.

http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf

http://uc-r.github.io/gbm_regression

3.7 RF vs. GBM

RF	GBM
Ensemble of models	Ensemble of models
Trees as the base learner	Trees as the base learner
Bagging	Boosting
Variance reduction	Bias reduction
Trees: created independently	Dependent on past trees
Each tree: have maximum depth	Each tree: have minimum depth
Each tree contributes equally to the final model	Each tree contributes unequally to the final model
	More computation time

3.8 AdaBoost vs. GBM

Both use a base weak learner and they try to boost the performance of a weak learner by iteratively shifting the focus towards problematic observations that were difficult to predict. At the end, a strong learner is formed by addition (or weighted addition) of weak learners.

AdaBoost	GBM
Shift is done by up-weighting observations that were misclassified before	Identify difficult observations by large residuals computed in the previous iteration
Shortcomings identified by high-weight data points	Shortcomings identified by gradients
Exponential loss gives more weights for the samples fitted worse	Gradient boost further dissect error components to bring in more explanation (???)
A special case of GBM in terms of loss function	More general in nature

3.9 XGBoost: eXtreme Gradient Boosting

gradient decent

boosting

3.9.1 Pros

- **Regularized objective**

- XGBoost is also known as **regularized boosting** technique
 - * Learning Rate: Scale the contribution of each tree by a factor $0 < \nu < 1$
 - * Feature subsampling
- The additional regularization term helps to smooth the final learned weights to avoid **over-fitting**

- **Second-order gradients**

- Compute second-order gradients, i.e. second partial derivatives of the loss function (similar to Newton's method)
- Provide more information about the direction of gradients and how to get to the minimum of our loss function

- **Parallel Processing**

- XGBoost can make use of multiple cores on the CPU
- This is possible because of a block structure in its system design
 - * Data is sorted and stored in in-memory units called blocks
 - * Enable the data layout to be reused by subsequent iterations, instead of computing it again

- **Weighted quantile sketch**

- XGBoost has a distributed weighted quantile sketch algorithm to effectively handle weighted data

- **Handling sparse data**

- XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data
- When a value is **missing in the sparse column** the sample is classified into this **default direction**
 - * The default directions are learned from the data
 - * Having a default direction allows the algorithm to skip samples that are missing an entry for

the feature

- This technique takes advantage of the sparsity in the dataset, reducing the computation to a linear search on only the non-missing entries
- High Flexibility
 - XGBoost allow users to define custom optimization objectives and evaluation criteria

3.9.2 Parameters Tuning

- eta [default=0.3]:
 - Makes the model more robust by shrinking the weights on each step
 - Typical final values to be used: 0.01-0.2
- min child weight [default=1]
 - Defines the minimum *sum of weights* of all observations required in a child.
 - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Too high values can lead to under-fitting hence, it should be tuned using CV.
- max depth [default=6]
 - The maximum depth of a tree
 - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
 - Should be tuned using CV.
 - Typical values: 3-10
- max leaf nodes:
 - The maximum number of terminal nodes or leaves in a tree.
 - Can be defined in place of max depth. Since binary trees are created, a depth of n would produce a maximum of 2^n leaves.
 - If this is defined, XGB will ignore max depth.
- gamma [default=0]:
 - A node is split only when the resulting split gives a *positive reduction in the loss function*. Gamma specifies the minimum loss reduction required to make a split.

- (Large gamma) makes the algorithm conservative. The values can vary depending on the loss function and should be tuned.
- subsample [default=1]
 - Denotes the *fraction of observations* to be randomly samples for each tree.
 - Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.
 - Typical values: 0.5-1
- colsample bytree [default=1]
 - Denotes the *fraction of features/columns* to be randomly samples for each tree.
 - Typical values: 0.5-1
- lambda [default=1]
 - L2 regularization term on weights (analogous to Ridge regression)
 - This used to handle the regularization part of XGBoost. Though many data scientists don't use it often, it should be explored to reduce overfitting.
- alpha [default=0]
 - L1 regularization term on weight (analogous to Lasso regression)
 - Can be used in case of very high dimensionality so that the algorithm runs faster when implemented

3.10 LightGBM

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

3.10.1 Gradient-based One-Side Sampling (GOSS)

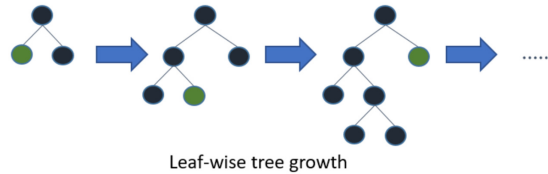
- LightGBM **keeps all the instances with large gradients** and performs random sampling on the instances with small gradients
 - Those instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain
 - To compensate the influence to the data distribution, when computing the information gain, GOSS introduces a **constant multiplier** for the data instances with small gradients
 - * Goss firstly sorts the data instances according to the absolute value of their gradients and selects the top $a \times 100\%$ instances
 - * Then it randomly samples $b \times 100\%$ instances from the rest of the data
 - * GOSS amplifies the sampled data with small gradients by a constant $\frac{1-a}{b}$ when calculating the information gain
 - * Put more focus on the under-trained instances without changing the original data distribution by much

3.10.2 Exclusive Feature Bundling

- Take advantage of sparse datasets by grouping features in a near lossless way to effectively reduce the number of features
- In a sparse feature space, many features are (almost) exclusive, i.e., they rarely take nonzero values simultaneously
 - Safely bundle such exclusive features into a single feature (**exclusive feature bundle**)
- LightGBM designs an efficient algorithm by reducing the optimal bundling problem to a **graph coloring problem** (by taking features as vertices and adding edges for every two features if they are not mutually exclusive), and solving it by a greedy algorithm with a constant approximation ratio

3.10.3 Optimization in Accuracy: Leaf-wise (Best-first) Tree Growth

- LightGBM splits the tree **leaf wise with the best fit** whereas other boosting algorithms split the **tree depth wise or level wise** rather than leaf-wise
 - It will choose the leaf with **max delta loss** to grow
 - Holding the number of leaf fixed, leaf-wise algorithms tend to achieve lower loss than level-wise algorithms
 - Leaf-wise may cause over-fitting when N is small, so LightGBM includes the `max_depth` parameter to limit tree depth. However, trees still grow leaf-wise even when `max_depth` is specified
 - Result in much better accuracy which can rarely be achieved by any of the existing boosting algorithms



Explains how LightGBM works



How other boosting algorithm works

References

<https://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/>

4 Subset Selection and Shrinkage Methods

4.1 Subset selection

Pros:

- Produces a model that is interpretable and has possibly lower prediction error than the full model

Cons:

- Discrete process: variables are either retained or discarded—it often exhibits high variance, and so doesn't reduce the prediction error of the full model

4.1.1 Best subset selection

- Fit models for all possible predictors (2^p) by cross-validated prediction error, C_p , AIC , BIC , or adjusted R^2
- The model containing all of the predictors will always have the smallest RSS and the largest R^2
- High R^2 indicates a model with a low training (not testing) error
- Drops all variables with **coefficients smaller than the M th largest** – **hard-thresholding**
- Limitations:
 - Computationally expensive
 - Infeasible for p much larger than 40

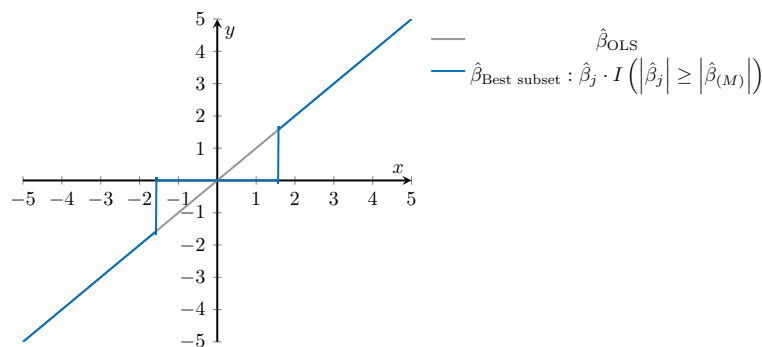


Figure 1: Thresholding functions

4.1.2 Stepwise selection

4.1.2.1 Forward

- Begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model.
- At each step, the variable that gives the **greatest additional improvement** to the fit is added to the model.
- $1 + p(p + 1)/2$
- Advantages:
 - Computational advantage over best subset selection
 - More constrained search, and will have lower variance, but perhaps more bias
- Disadvantages:
 - Will not yield a unique solution if $p \geq n$
 - Greedy algorithm: produce a nested sequence of model and ot guaranteed to yield the best model containing a subset of the p predictors

4.1.2.2 Backwards

- It begins with the full least squares model containing all p predictors, and then iteratively removes the predictor that has the **least impact on the fit**, one-at-a-time.
- $1 + p(p + 1)/2$
- Requires the number of samples n is larger than the number of variables p (so that the full model can be fit)
- Disadvantages: Not guaranteed to yield the best model containing a subset of the p predictors

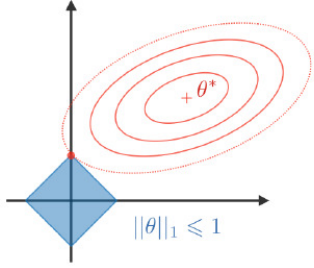
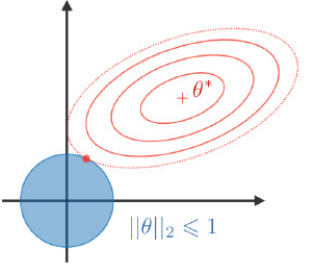
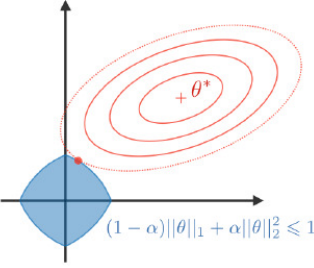
4.1.2.3 Hybrid Approaches

More closely mimic best subset selection while retaining the computational advantages of forward and backward step-wise selection

todo: https://www.julyedu.com/question/big/kp_id/23/ques_id/1572

4.2 Regularization

To control for the **variance**, but might introduce bias

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[(1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0, 1]$

4.2.1 RIDGE

4.2.1.1 Summary

Ridge regression (L_2) shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares. Ridge regression does a proportional shrinkage.

$$\beta^{\text{ridge}} = \left\{ \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p X_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\}$$

Intercept β_0 has been left out of the penalty term. Penalization of the intercept would make the procedure depend on the origin chosen for Y .

4.2.1.2 lambda

- $\lambda \geq 0$
- Complexity parameter that controls the amount of shrinkage/regularization
- The larger the value of $\lambda \rightarrow$ greater amount of shrinkage \rightarrow Larger bias, smaller variance
- $\lambda \downarrow 0$: least squares solutions

- $\lambda \uparrow \infty$: intercept only model (all coefficients will be 0) ($\hat{\beta}_{\lambda=\infty}^{ridge} = 0$)
- Choosing λ by **cross validation**
 - Choose the value of λ that minimizes the estimated MSE
 - Leave-one-out
 - * Take out one observation \rightarrow Fit the model on the remaining $n-1 \rightarrow$ Repeat \rightarrow Average
 - * Low bias, high variance
 - k -fold
 - * Split the data in k equal-sized subsets (folds) at random \rightarrow Take out one fold \rightarrow Fit the model on the remaining $k-1$ folds \rightarrow Repeat \rightarrow Average
 - * Acceptable bias, low variance

4.2.1.3 Applications of RIDGE

- High correlation between regressors
 - OLS coefficients may be poorly determined because of high correlation between regressors
- A large number of variables with similarly sized coefficients
 - RIDGE shrinks their coefficients toward zero, and those of strongly correlated variables toward each other
- Small sample size and/or $p \geq n$
 - Extreme variability in the training data

4.2.1.4 Scaling

- The value produced by the penalty component is dependant on the scale of the underlying data
- Standardize the predictors \mathbf{X} that each predictor is centered $\frac{1}{n} \sum_{i=1}^n x_{ij} = 0$ and has unit variance $\frac{1}{n} \sum_{i=1}^n x_{ij}^2 = 1$ as z
- Ensure that the penalty term is applied equally to all variables irrespective of the units in which they are measured

4.2.2 LASSO

$$\beta^{\hat{lasso}} = \{\operatorname{argmin}_{\beta} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p X_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j| \}$$

4.2.2.1 L1 sparsity

- Large λ will set some coefficients to 0
- LASSO will perform **continuous model selection**
- If group of predictors are highly correlated, lasso picks only one of them and shrinks the others to zero
- **Standardize** the predictors \mathbf{X} that each predictor is centered $\frac{1}{n} \sum_{i=1}^n x_{ij} = 0$ and has unit variance $\frac{1}{n} \sum_{i=1}^n x_{ij}^2 = 1$ as z . Without standardization, the lasso solutions would depend on the units.

4.2.2.2 Computation of the Lasso solutions

- Cannot get the optimal condition directly
 - $|\beta|$ does not have a derivation at $\beta = 0$
- Possible to obtain closed form solution for LASSO
 - Quadratic programming problem
 - $\beta^{\text{lasso}} = \mathbf{S}(\beta^{\text{OLS}}, \lambda)$
 - **S: soft-thresholding operator**

$$\mathbf{S}_{\lambda}(\beta_j) = \operatorname{sign}(\beta_j)(|\beta_j| - \lambda)_+$$

or

$$\mathbf{S}(\beta_j, \lambda) = \begin{cases} \beta_j - \lambda, & \text{if } \beta_j > \lambda \\ 0, & \text{if } |\beta_j| \leq \lambda \\ \beta_j + \lambda, & \text{if } |\beta_j| < -\lambda \end{cases}$$

- No close form solution for RIDGE
- **Least angle regression(LARS)**
 - Exploit the special structure of the LASSO problem

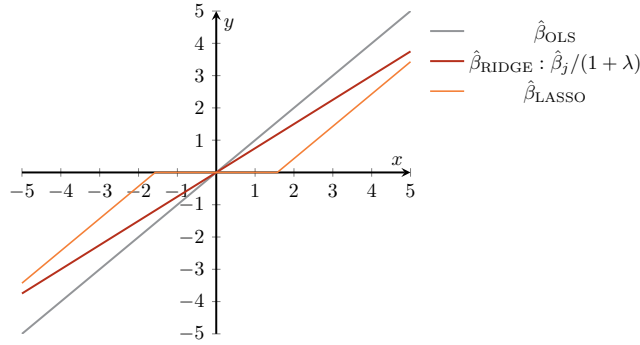


Figure 2: Thresholding functions

- Provide an efficient way to compute the solutions simultaneously for the entire path of LASSO solutions

- **Coordinate descent**

- Fix the penalty parameter λ in the Lagrangian form and optimize successively over each parameter, holding the other parameters fixed at their current values

4.2.2.3 Oracle property? Unbiased?

4.2.2.4 Pros

- Interpretation of the final model
 - The L_1 -penalty provides a natural way to encourage or enforce sparsity and simplicity in the solution
- Statistical efficiency
 - **Bet-on-sparsity principle**
 - * Assume that the underlying true signal is sparse and use an L_1 -penalty to try to recover it
 - * If the assumption is correct, can do a good job in recovering the true signal
 - * If the assumption is wrong — the underlying truth is not sparse in the chosen bases—then the L_1 -penalty will not work well
- Computational efficiency
 - L_1 -penalty is **convex**
 - Convex and the assumed sparsity can lead to significant computational advantages

- Works when face high dimensional data for which $p \gg n$

4.2.2.5 Cons

- Lasso shrinkage causes the estimates of the non-zero coefficients to be biased towards zero, and in general they are **not consistent**
 - Run the lasso to identify the set of non-zero coefficients \implies Fit an unrestricted linear model to the selected set of features
 - * Not feasible when selected set is large
 - **Relaxed Lasso**
 - * Use the lasso to select the set of non-zero predictors \implies Apply the lasso again, but using only the selected predictors from the first step
 - * Use cross-validation to estimate the initial penalty parameter for the lasso, and then again for a second penalty parameter applied to the selected set of predictors
 - * The variables in the second step have less competition from noise variables, cross-validation will tend to pick a smaller value for λ , and hence their coefficients will be shrunken less than those in the initial estimate
 - **Smoothly clipped absolute deviation (SCAD)**
 - * Modify the lasso penalty function so that larger coefficients are shrunken less severely

4.2.2.6 LASSO, RIDGE and Best Selection: Bayes Estimates with Different Priors

$|\beta_j|^q$: the log-prior density for β_j

- Variable subset selection
 - $q = 0$
 - Count the number of nonzero coefficients
- LASSO: $q = 1$
- RIDGE: $q = 2$

4.2.3 Least Angle Regression

4.2.4 Principal Components Regression

4.2.5 Partial Least Squares

References

5 Unsupervised Learning

5.1 Dimension Reduction

5.1.1 PCA

Unsupervised

Dimension Reduction

Principal component analysis is the *unsupervised* learning methodology in which the data is converted from a high dimensional space into another space with lower or equal number of dimensions. In this lower dimensional space, the dimensions are the projections of variance of data.

5.1.1.1 First Principal Component

- Principal components provide low-dimensional **linear surfaces** that are closest to the observations:
 - The first principal component has been chosen so that the projected observations are as close as possible to the original observations (average squared Euclidean distance)
- The linear combination \mathbf{X}_{v1} has the highest variance among all **linear** combinations of the features

The first principal component of a set of features \mathbf{X}_{v1} : X_1, X_2, \dots, X_p is the **normalized linear combination** of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \dots + \phi_{p1}X_p$$

that has the **largest variance** ($\sum_{j=1}^p \phi_{j1}^2 = 1$).

- Z_1 : **score** of first components (projections along these directions)
- $\phi_{11}, \dots, \phi_{p1}$: **loadings** of the first principal components
 - Principal component loading vectors: the directions in feature space along which the data vary the most

5.1.1.2 Second principal component

- \mathbf{X}_{v2} has the highest variance among all linear combinations satisfying v_2 orthogonal to v_1
- The second principal component Z_2 is a linear combination of the variables that is uncorrelated with Z_1 , and has largest variance subject to this constraint.

Together the first M principal component score vectors and the first M principal component loading vectors provide the best M -dimensional approximation (in terms of Euclidean distance) to the i th observation x_{ij} .

Scaling matters: typically scale each variable to have standard deviation one before PCA

5.1.2 ICA

5.1.3 Singular Valued Decomposition

5.2 Clustering Algorithms

Goal: to partition the observations into groups (“clusters”) so that the pairwise dissimilarities between those assigned to the same cluster tend to be smaller than those in different clusters.

Clustering algorithms fall into three distinct types:

- **Combinatorial algorithms:** work directly on the observed data with no direct reference to an underlying probability model
 - k-means
- **Mixture modeling:** supposes that the data is an *i.i.d* sample from some population described by a probability density function. This density function is characterized by a parameterized model taken to be a mixture of component density functions; each component density describes one of the clusters. This model is then fit to the data by maximum likelihood or corresponding Bayesian approaches.
- **Mode seeking/bump hunting:** take a non-parametric perspective, attempting to directly estimate distinct modes of the probability density function. Observations “closest” to each respective mode then define the individual clusters.

5.2.1 Proximity Matrices

Most of the popular clustering algorithms take a **dissimilarity matrix** as their input.

Measurements x_{ij} for $i = 1, 2, \dots, N$, on variables $j = 1, 2, \dots, p$ (**attributes**), define a dissimilarity $d_j(x_{ij}, x_{i'j})$ between values of the j th attribute, and then define

$$D(x_i, x_{i'}) = \sum_{j=1}^p d_j(x_{ij}, x_{i'j})$$

as the the dissimilarity between objects i and i' .

For $d_j(x_{ij}, x_{i'j})$:

Quantitative variables:

1. Squared distance:

$$d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$$

2. Absolute error:

$$d_j(x_{ij}, x_{i'j}) = l(|x_{ij} - x_{i'j}|)$$

3. Correlations:

$$\rho(x_{ij}, x_{i'j}) = \frac{\sum_j (x_{ij} - \bar{x}_i)(x_{i'j} - \bar{x}_{i'})}{\sqrt{\sum_j (x_{ij} - \bar{x}_i)^2 \sum_j (x_{i'j} - \bar{x}_{i'})^2}}$$

- Clustering based on correlation (similarity) is equivalent to that based on squared distance (dissimilarity) when observations are standardized.

Ordinal variables:

1. $\frac{i-0.5}{M}, i = 1, \dots, M$ for M original values

Categorical/nominal variables:

1. If the variable assumes M distinct values, these can be arranged in a symmetric $M \times M$ matrix with elements $L_{rr'} = L_{r'r}, L_{rr} = 0, L_{rr'} \geq 0$. The most common choice is $L_{rr'} = 1$ for all $r \neq r'$, while unequal losses can be used to emphasize some errors more than others.

For $D(x_i, x_{i'})$: Combining the p -individual attribute dissimilarities $d_j(x_{ij}, x_{i'j}), j = 1, 2, \dots, p$, into a single overall measure of dissimilarity $D(x_i, x_{i'})$ between two objects or observations $(x_i, x_{i'})$ processing the respective attribute values by **weighted average**:

$$D(x_i, x_{i'}) = \sum_{j=1}^p w_j \cdot d_j(x_{ij}, x_{i'j}), \sum_{j=1}^p w_j = 1$$

5.2.2 k-Means Clustering

Summary

A prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters represented by their centroids

Dissimilarity measure: [squared Euclidean distance](#)

$$d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2 = \|x_{ij} - x_{i'j}\|^2$$

Assign the N observations to the K clusters in such a way that [within each cluster](#) the average dissimilarity of the observations from the cluster mean is minimized

5.2.2.1 Algorithm

1. Randomly assign a number, from 1 to K , to each of the observations
2. Iterate until the cluster assignment stop changing
 - (a) For each of the K cluster, computer the cluster [centroid](#). The k th cluster centroid is the vector of the p feature [means](#) for the observations in the k th cluster
 - (b) Assign each observation to the cluster whose centriod is closest (Eluclidean distance)

5.2.2.2 Properties

- Iterative descent clustering methods
- Each observation belongs to at last one of the K clusters
- The clusters are [non-overlapping](#)
- Total [within cluster variation](#) (the total sum of the squared error (SSE), measured by Euclidean distance) is as small as possible
- Not guaranteed for the global minimum: result may represent a suboptimal local minimum
 - Start the algorithm with many different random choices for the starting means, and choose the solution having smallest value of the objective function
- *Top-down* procedure:
- The centriod that minimizes the SSE is the mean

5.2.2.3 How to choose number of clusters

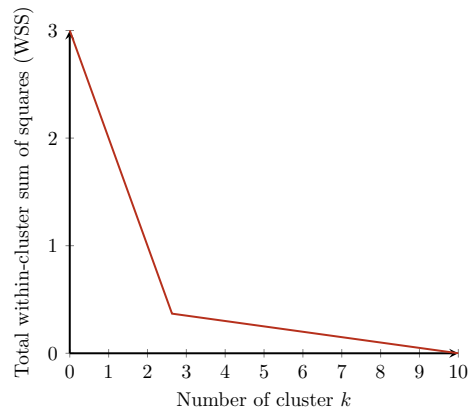
1. A choice for the number of clusters K depends on the goal:

- For data segmentation, K is usually defined as part of the problem

2. **Elbow method**

- Examine the within-cluster dissimilarity as a function of the number of clusters K .
- Looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.

- (a) Compute clustering algorithm for different values of k . For instance, by varying k from 1 to 10 clusters.
- (b) For each k , calculate the **total within-cluster sum of square (WSS)**
- (c) Plot the curve of WSS according to the number of clusters k .
- (d) The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters



3. **Average silhouette method**

$$s = \frac{b - a}{\max(a, b)}$$

- a : mean distance between a sample and all other points in the *same class*
- b : mean distance between a sample and all other points in the next nearest cluster
- Measure the quality of a clustering
 - Determine how well each object lies within its cluster
 - Estimates the average distance between clusters.
 - Ranges from -1 to 1 : a high value indicates that the object is well matched to its own cluster

and poorly matched to neighboring clusters

- (a) Compute clustering algorithm for different values of k . For instance, by varying k from 1 to 10 clusters.
- (b) For each k , calculate the average silhouette of observations (avg.sil)
- (c) Plot the curve of avg.sil according to the number of clusters k .
- (d) The location of the maximum is considered as the appropriate number of clusters

4. Gap statistic method:

- Compares the total within intra-cluster variation for different values of k with their expected values under null reference distribution of the data
- Estimates the optimal number of clusters to be the place where the gap between the two curves is largest (gap statistic)

- (a) Cluster the observed data, varying the number of clusters from $k = 1, \dots, k_{max}$, and compute the corresponding total within intra-cluster variation W_k .
- (b) Generate B reference data sets with a random uniform distribution. Cluster each of these reference data sets with varying number of clusters $k = 1, \dots, k_{max}$, and compute the corresponding total within intra-cluster variation W_{kb} .
- (c) Compute the estimated gap statistic as the deviation of the observed W_k value from its expected value W_{kb} under the null hypothesis: $Gap(k) = \frac{1}{B} \sum_{b=1}^B \log(W_{kb}) - \log(W_k)$.
- (d) Choose the number of clusters as the smallest value of K such that the gap statistic is within one standard deviation of the gap at $K+1$: $Gap(K) \geq Gap(K+1) - s_{K+1}$.

5.2.2.4 Choosing the initial centroids

1. Randomly select initial centroids:
 - Poor performance
2. Start the algorithm with many different random choices for the choosing initial centroids, and choose the solution having smallest value of the objective function
3. Take a sample of points and cluster them using a hierarchical clustering technique
 - K clusters are extracted from hierarchical clustering (computationally expensive), the the centroids of those clusters used as the initial centroids

- Used when
 - The sample is relatively small (a few hundred to a few thousand)
 - K is relatively small compared to the sample size
- 4. Select the first point at random or take the centroids of all points, for each successive initial centroid, select the point that is **farthest from any of the initial centroids** already selected
 - Obtain a set of initial centroids that is guaranteed to be not only randomly selected but also well separated
 - Can select outliers, rather than points in dense regions
 - Expensive to compute the farthest points from the current set of initial centroids
 - Improvement:
 - Apply to a sample of the points:
 - * Outliers tend not to show up in a random sample
 - * Greatly reduce the computation involved in finding the initial centroids

5.2.2.5 Space complexity

$$O((n + K)p)$$

- n : number of observations
- p : the number of attributes

5.2.2.6 Time complexity

$$O(I * K * n * p)$$

- I : number of iterations required for convergence (often small)
- Linear in the number of observations

5.2.2.7 Why Euclidean not Manhattan distance

5.2.2.8 Pros

- **Simple** and can be used for a wide variety of data types

- Quite **efficient**, even though multiple runs are often performed. Some variants, including bisecting K-means, are even more efficient, and are less susceptible to initialization problems

5.2.2.9 Cons

- Does not give a **linear ordering** of objects within a cluster
- As the number of clusters K is changed, the cluster memberships can change in arbitrary way
- Squared Euclidean distance places the highest influence on the largest distances: lack robustness against **outliers** that produce very large distances
- Not guaranteed for the global minimum: result may represent a **suboptimal local minimum**
- Cannot handle non-globular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified
- K -means is restricted to data for which there is a notion of a center (centroid)

5.2.2.10 k-means: a Variant of EM algorithm

k -means is a model that the clusters are defined by **Gaussian distributions** with **unknown means** (the θ to be estimated) and **identity covariance matrices**

- First, one makes an initial guess $\hat{\theta}^0$ of the k cluster centers.
- Then in the E-like step, one assigns each of the n points to the closest cluster based on the estimated cluster centers $\theta^{(m)}$.
- Then in the M-like step, one takes all the points assigned to each cluster, and computes the mean of those points to form a new estimate of the cluster's centroid.

EM clustering differs from k -means clustering in that at each iteration you do not choose a single $x^{(m)}$, that is, one does not force each observed point y_i to belong to only one cluster. Instead, each observed point y_i is probabilistically assigned to the k clusters by estimating $p(x|y, \theta^{(m)})$.

References:

<https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/>

<https://www-users.cs.umn.edu/~kumar001/dmbook/ch8.pdf>

Tan, P.N., 2007. Introduction to data mining. Pearson Education India.

<https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-unsupervised-learning>

5.2.3 k-medoids Clustering

- The centroid that minimizes the total sum of the abs err (manhattan) is the median

5.2.4 Agglomerative Hierarchical Clustering

Summary

Clustering techniques that produce a hierarchical clustering by starting with each point as a **singleton cluster** then repeated merging the two closest clusters until a single, all-encompassing cluster remains.

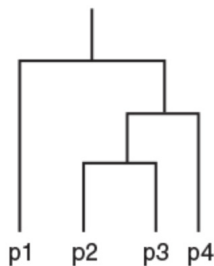
5.2.4.1 Algorithm

1. Start with each point in its own cluster
2. Identify the **closest** two clusters and merge them
3. Repeat
4. Ends when all points are in a single cluster

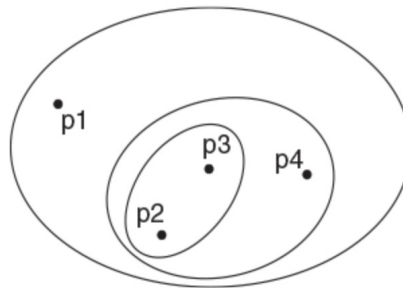
5.2.4.2 Properties

- **Agglomerative** strategies start at the bottom (*bottom-up*) and at each level recursively merge a selected pair of clusters into a single cluster. This produces a grouping at the next higher level with one less cluster. The pair chosen for merging consist of the two groups with the smallest intergroup dissimilarity.
- Monotonicity property:
 - the dissimilarity between merged clusters is monotone increasing with the level of the merger

5.2.4.3 Dendrogram



(a) Dendrogram.

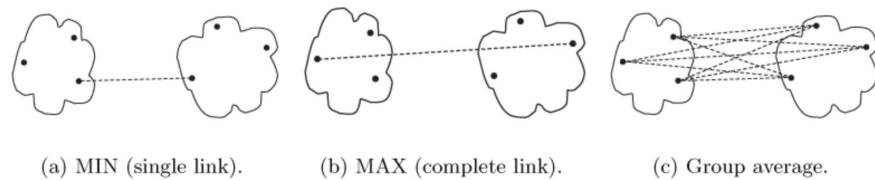


(b) Nested cluster diagram.

- A description of the clustering structure of the data as imposed by the particular algorithm employed

- Different hierarchical methods, as well as small changes in the data, can lead to quite different dendrograms
- The extent to which the hierarchical structure produced by a dendrogram actually represents the data itself can be judged by the **cophenetic correlation coefficient**:
 - Correlation between the $N(N-1)/2$ pairwise observation dissimilarities $d_{ii'}$ input to the algorithm and their corresponding cophenetic dissimilarities $C_{ii'}$ derived from the dendrogram.
 - * The cophenetic dissimilarity $C_{ii'}$ between two observations (i, i') is the inter-group dissimilarity at which observations i and i' are first joined together in the same cluster.

5.2.4.4 Dissimilarity



- **Complete linkage:**
 - The **maximum of the distance** (minimum of the similarity) between any two points in the two different clusters
 - Tend to produce compact clusters with **small diameters**
 - Can produce clusters that violate the “closeness” property (i.e., observations assigned to a cluster can be much closer to me)
 - Depend only on the ordering of the $d_{ii'}$, **invariant to monotone transformations**
- **Single linkage:**
 - The **minimum of the distance** (maximum of the similarity) between any two points in the two different clusters
 - Tend to combine, at relatively low thresholds, observations linked by a series of close intermediate observations (**chaining**)
 - Violate the “compactness” property that all observations within each cluster tend to be similar to one another, based on the supplied observation dissimilarities $\{d_{ii'}\}$
 - Tend to produce clusters with very **large diameters**
 - Depend only on the ordering of the $d_{ii'}$, **invariant to monotone transformations**
- **Average linkage:**

- The average pairwise proximity among all pairs of points in the different clusters
- Intermediate approach between the single and complete link approaches
- Its results depend on the **numerical scale** on which the observation dissimilarities $d_{ii'}$ are measured. Applying a monotone strictly increasing transformation $h(\cdot)$ can change the result.

- **Centroid linkage/Wald's method:**

- The increase in the **squared error** that results when two clusters are merged
- Uses the same objective function as K-means clustering
- Mathematically Ward's method is very similar to the **group average** method when the proximity between two points is taken to be the square of the distance between them
- Sensitive to **outliers**

5.2.4.5 Time complexity

$O(m^2 \log m)$ (m : number of observations)

5.2.4.6 Space complexity

$O(m^2)$

5.2.4.7 Pros

- Does need to commit to a particular choice of k
- By cutting off the dendrogram at various heights, different numbers of clusters emerge, and the sets of clusters are nested within one another
- It gives some partial ordering information about the samples

5.2.4.8 Cons

- Lack of a Global Objective Function:
 - Use various criteria to decide locally, at each step, which clusters should be merged (or split for divisive approaches)
 - Once a decision is made to merge two clusters, it cannot be undone at a later time
 - Cause trouble for noisy, high-dimensional data, such as document data

- The space and time complexity of hierarchical clustering severely limits the size of data sets that can be processed
- Hierarchical methods impose hierarchical structure whether or not such structure actually exists in the data.

5.2.5 DBSCAN

Summary

Density based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm. Points in low-density regions are classified as noise and omitted (does not produce a complete clustering)

5.2.6 Expectation-Maximization Algorithm

clustering MLE missing data latent variable

Summary:

The EM algorithm is an efficient iterative procedure to compute the **Maximum Likelihood Estimate** in the presence of **missing or hidden data**. EM are used in problems for which maximization of the likelihood is difficult, but made easier by **enlarging the sample with latent (unobserved) data (data augmentation)**.

5.2.6.1 Algorithm

We make a guess about the complete data X and solve for the θ that maximizes the (expected) log-likelihood of X . And once we have an estimate for θ , we can make a better guess about the complete data X , and iterate. Each iteration of the EM algorithm consists of two processes:

- In the expectation, or **E-step**, the missing data are estimated given the observed data and current estimate of the model parameters. This is achieved using the **conditional expectation**.
- In the **M-step** (maximization-step), the likelihood function is maximized under the assumption that **the missing data are known**. The estimate of the missing data from the E-step are used in lieu of the actual missing data.

1. Let $m = 0$ and make an initial estimate $\theta^{(m)}$ for θ
2. **E - step:** Given the observed data y and pretending for the moment that your current guess $\theta^{(m)}$ is correct, formulate the conditional probability distribution $p(x|y, \theta^{(m)})$ for the complete data x .
3. **E - step:** Using the conditional probability distribution $p(x|y, \theta^{(m)})$ calculated in Step 2, form the conditional expected log-likelihood, which is called the **Q-function**:

$$\begin{aligned} Q(\theta|\theta^{(m)}) &= \int_{\mathcal{X}(y)} \log p(x|\theta) p(x|y, \theta^{(m)}) dx \\ &= E_{X|y, \theta^{(m)}}[\log p(X|\theta)] \end{aligned}$$

which is a function of θ , but also depends on your current guess $\theta^{(m)}$ implicitly through the $p(x|y, \theta^{(m)})$ calculated in Step 2 .

4. **M - step:** Find the θ that maximizes the Q-function. the result is the new estimate $\theta^{(m+1)}$.
5. Let $m := m + 1$ and go back to Step 2. Iterate until the estimate **stops changing**: $\|\theta^{(m+1)} - \theta^{(m)}\| < \epsilon$ for some $\epsilon > 0$, or to iterate until the log-likelihood $\ell(\theta) = \log p(y|\theta)$ stops changing: $|\ell(\theta^{(m+1)}) - \ell(\theta^{(m)})| < \epsilon$ for some $\epsilon > 0$.

5.2.6.2 Derivation

Jensen's inequality

Let f be a **convex function** defined on an interval I . If $x_1, x_2, \dots, x_n \in I$ and $\lambda_1, \lambda_2, \dots, \lambda_n \geq 0$ with $\sum_{i=1}^n \lambda_i = 1$

$$f\left(\sum_{i=1}^n \lambda_i x_i\right) \leq \sum_{i=1}^n \lambda_i f(x_i)$$

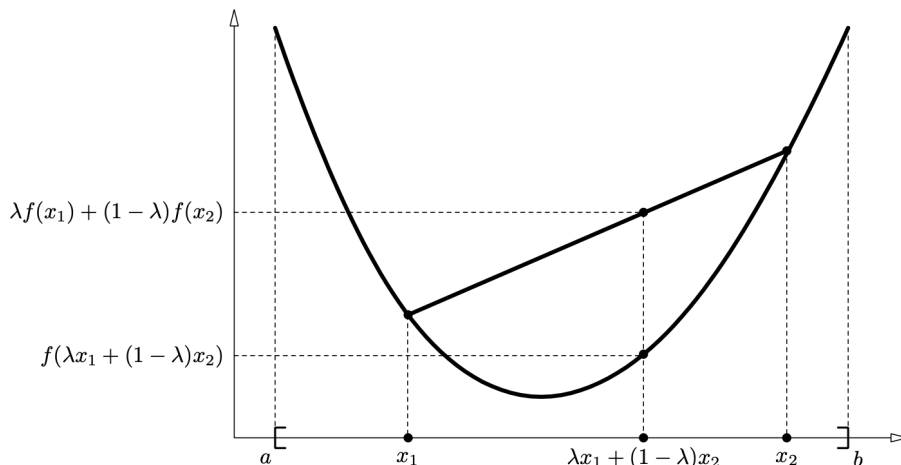


Figure 1: f is *convex* on $[a, b]$ if $f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$
 $\forall x_1, x_2 \in [a, b], \lambda \in [0, 1]$.

- Jensen's inequality provides a simple proof that the arithmetic mean is greater than or equal to the geometric mean ($\frac{1}{n} \sum_{i=1}^n x_i \geq \sqrt[n]{x_1 x_2 \cdots x_n}$)

5.2.6.3 Pros

- Conceptual simplicity
- Ease of implementation
- Monotonicity: each iteration improves $\ell(\theta)$
- The rate of convergence on the first few steps is typically quite good

5.2.6.4 Cons

- EM will not necessarily find the global maximum of the likelihood
 - Start EM from multiple random initial guesses, and choose the one with the largest likelihood as the final guess for θ
- Become excruciatingly **slow** as approaching a **local optima**
- Higher dimensionality can dramatically slow down the E-step

References

<http://cs229.stanford.edu/notes/cs229-notes8.pdf>

https://www.cs.utah.edu/~piyush/teaching/EM_algorithm.pdf

<https://www.cs.cmu.edu/~awm/15781/assignments/EM.pdf>

<http://www.mayagupta.org/publications/EMbookGuptaChen2010.pdf>

5.3 t-Distributed Stochastic Neighbor Embedding

Summary

t-SNE is a tool to **visualize high-dimensional data**. It converts similarities between data points to joint probabilities and tries to minimize the **Kullback-Leibler divergence** between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is **not convex**, i.e. with different initialization we can get different results.

5.3.1 Algorithm

t-SNE starts by converting the high-dimensional **Euclidean distances** between data points into **conditional probabilities** that represent **similarities** between x_i and x_j :

The σ_i will be specified by user. This procedure can be influenced by setting the **perplexity** of the algorithm. The perplexity can be interpreted as a smooth measure of the effective number of neighbors. The performance of t-SNE is fairly robust to changes in the perplexity, and typical values are between 5 and 50.

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i, \mathbf{x}_j\|/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i, \mathbf{x}_k\|/2\sigma_i^2)}, \quad p_{i|i} = 0$$

We define the **joint probabilities** p_{ij} in the high-dimensional space to be the summarized conditional probabilities:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

For the low-dimensional counterparts y_i and y_j of the high-dimensional data points x_i and x_j , it is possible to compute a similar joint probability q_{ij} .

A heavy-tailed distribution (Student t -distribution with $df = 1$) will be used to measure the similarities in the low-dimensional space:

$$q_{ij} = \frac{\exp(-\|(\mathbf{y}_i - \mathbf{y}_j)\|^2)}{\sum_{k \neq l} \exp(-\|(\mathbf{y}_k - \mathbf{y}_l)\|^2)}$$

If the map points y_i and y_j correctly model the similarity between the high-dimensional data points x_i and x_j , the conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equal.

t-SNE aims to find a low-dimensional data representation that minimizes the mismatch between $p_{j|i}$ and $q_{j|i}$: **minimize the Kullback-Leibler divergence** between P in the high-dimensional space, and a **Student-t based joint probability distribution** Q in the low-dimensional space with gradient

descent (and some tricks):

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

P represents the joint probability distribution over all other data points given data point x_i , and Q represents the joint probability distribution over all other map points given map point y_i .

We refer to this type of SNE as symmetric SNE because it has the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$