Boosting

- Also uses voting/averaging but models are weighted according to their performance
- Iterative procedure: new models are influenced by performance of previously built ones
  - New model is encouraged to become expert for instances classified incorrectly by earlier models
  - Intuitive justification: models should be experts that complement each other
- There are several variants of this algorithm

Boosting

- Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on.
- Boosting also requires bootstrapping. However, there is another difference here. Unlike in bagging, boosting weights each sample of data.

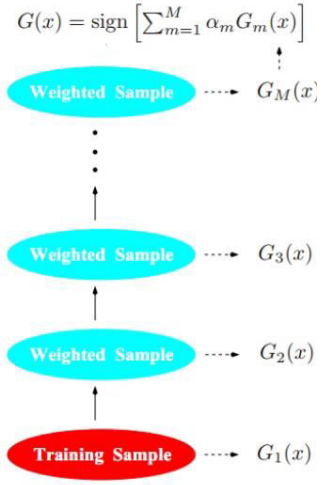
Boosting

- Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners.
- Unlike bagging that had each model run independently and then aggregate the outputs at the end without preference to any model.

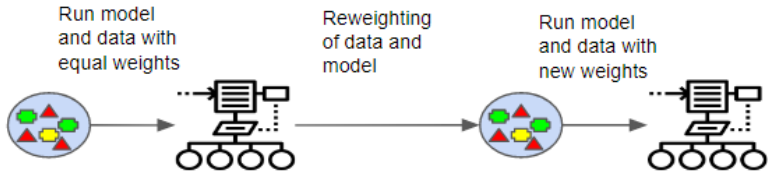
Boosting

IIMB
तेजस्वि नावधीतमस्तु

- Intuition: Ensemble many “weak” classifiers (typically decision trees) to produce a final “strong” classifier
  - Weak classifier → Error rate is only slightly better than random guessing.
- Boosting is a Forward Stagewise Additive model
- Boosting sequentially apply the weak classifiers one by one to repeatedly reweighted versions of the data.
- Each new weak learner in the sequence tries to correct the misclassification/error made by the previous weak learners.
  - Initially all of the weights are set to Wi = 1/N
  - For each successive step the observation weights are individually modified and a new weak learner is fitted on the reweighted observations.
  - At step m, those observations that were misclassified by the classifier Gm−1(x) induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly.
- Final “strong” classifier is based on weighted vote of weak classifiers



Why put weights on the samples of data?



- When boosting runs each model, it tracks which data samples are the most successful and which are not.
- The data sets with the most **misclassified** outputs are given **heavier weights.**
- These are considered to be data that have more complexity and requires more iterations to properly train the model.

AdaBoost.M1

classifier generation

```
Assign equal weight to each training instance.
For each of t iterations:
  Learn a classifier from weighted dataset.
  Compute error e of classifier on weighted dataset.
  If e equal to zero, or e greater or equal to 0.5:
    Terminate classifier generation.
  For each instance in dataset:
    If instance classified correctly by classifier:
       Multiply weight of instance by e / (1 - e).
  Normalize weight of all instances.
```

classification

```
Assign weight of zero to all classes.
For each of the t classifiers:
  Add -log(e / (1 - e)) to weight of class predicted
  by the classifier.
Return class with highest weight.
```

AdaBoost.M1

Step 1 – Creating First Base Learner

Step 2 – Calculating the Total Error (TE)

The total error is the sum of all the errors in the classified record for sample weights.

Step 3 – Calculating Performance of Stump

$$\text{Performance of Stump} = 1/2 \ln \left[\frac{1-TE}{TE} \right]$$

Step 4 – Updating Weights

If correct

New Sample Weight = Sample Weight * e^(Performance)

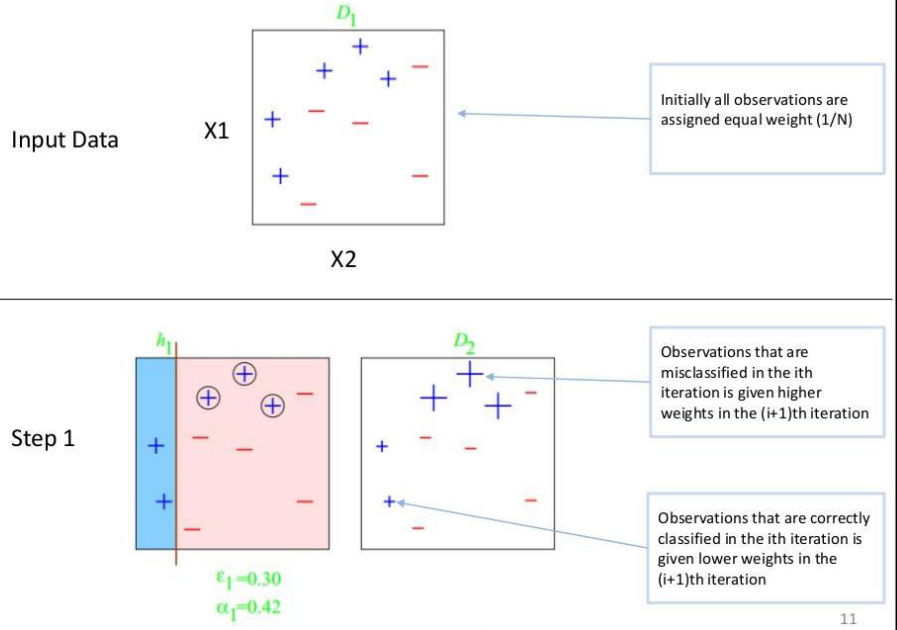
Else

New Sample Weight = Sample Weight * e^- (Performance)

Step 5 – Creating New Dataset

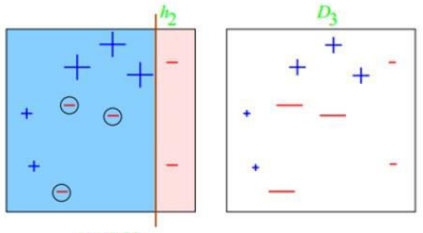
AdaBoost – Illustration

IIMB नेजस्वि नावधीतमस्तु

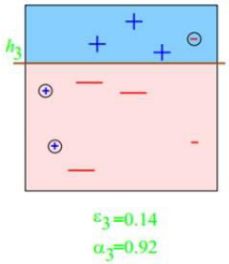


AdaBoost – Illustration

Step 2

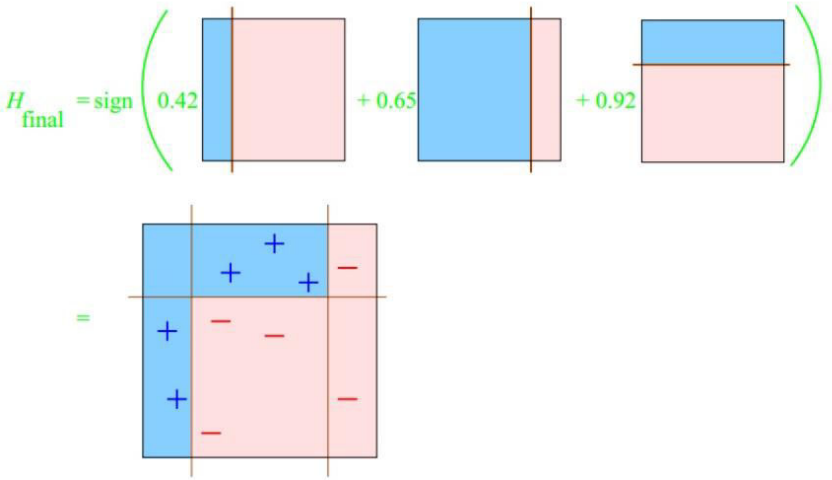


Step 3



AdaBoost – Illustration

Final Ensemble/Model



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No

← First, we'll start with some data.

Creating First Base Learner

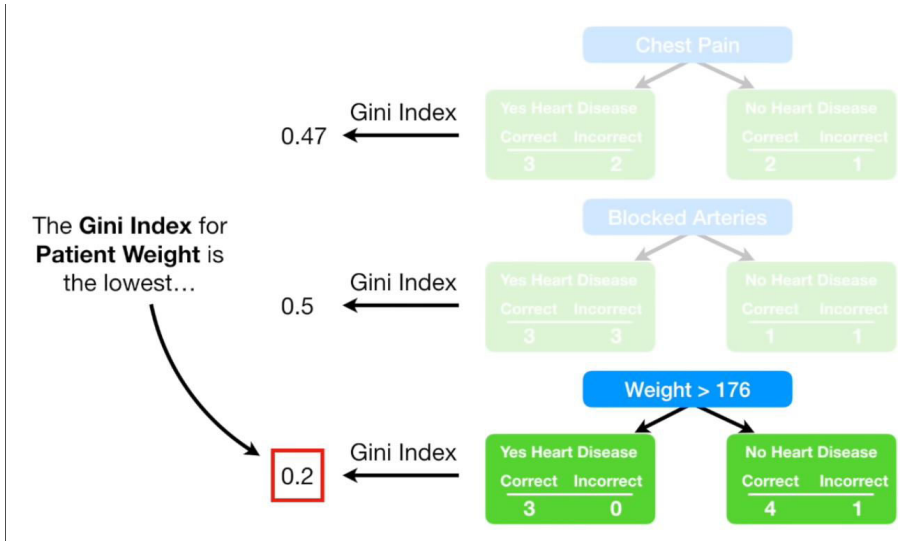
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

At the start, all samples get the same weight…

$$\frac{1}{\text{total number of samples}} = \frac{1}{8}$$

…and that makes the samples all equally important.

Step 1 Creating First Base Learner



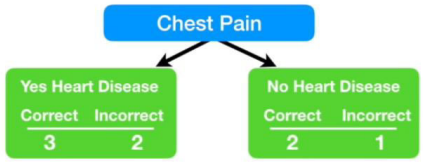
Step 2: Calculating the Total Error

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

The **Total Error** for a stump is the sum of the weights associated with the *incorrectly* classified samples.

Step 2: Calculating the Total Error

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



$$\text{Total Error} = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{3}{8}$$

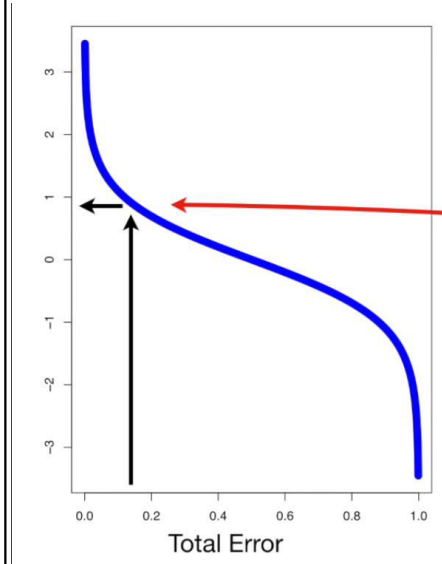
So the **Total Error** for **Chest Pain** is **3/8**.

Step 3 – Calculating Performance of Stump

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

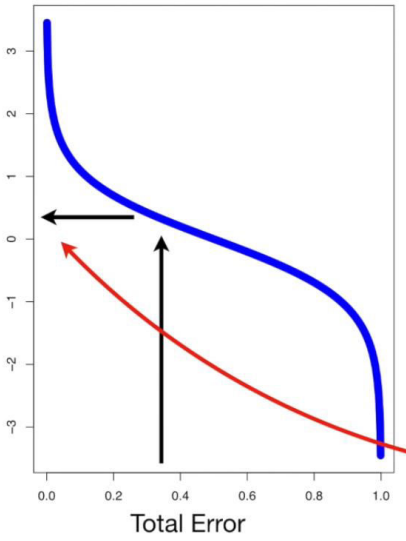
We use the **Total Error** to determine **Amount of Say** this stump has in the final classification with the following formula:

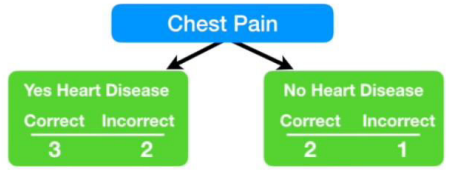
$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$



…and the **Amount of Say** that this stump has on the final classification is **0.97**.

$$\text{Amount of Say} = \frac{1}{2}\log(7) = 0.97$$



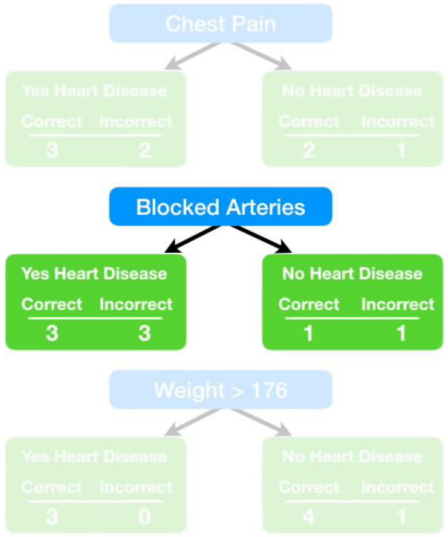


…and the **Amount of Say** that the **Chest Pain** stump would have had on the final classification is **0.42**.

$$\text{Amount of Say} = \frac{1}{2}\log(7/3) = 0.42$$

?

I'll leave the **Blocked Arteries** stump as an exercise for the viewer!



Step 4 – Updating Weights

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{\text{amount of say}}$$

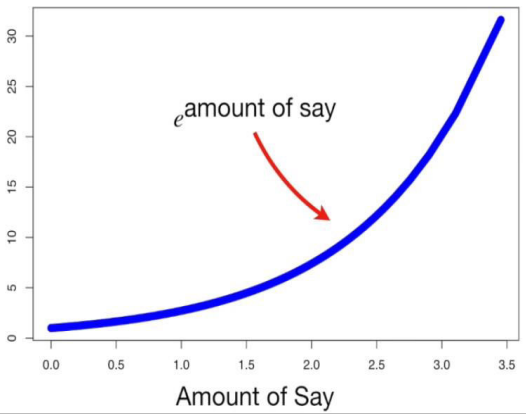
…and we will scale **1/8** with this term.

Step 4 – Updating Weights

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{\text{amount of say}}$$

$$= \frac{1}{8} e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$



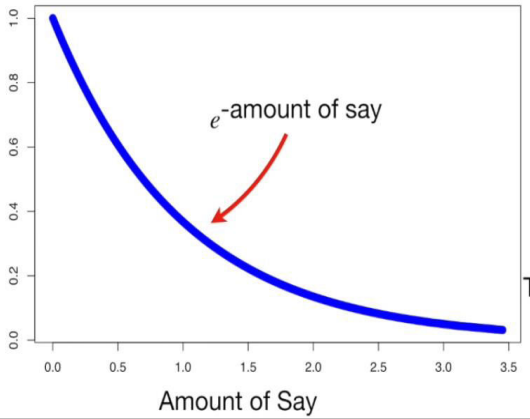
That means the new **Sample Weight** is **0.33**, which is *more* than the old one (**1/8** = **0.125**).

Step 4 – Updating Weights

$$\text{New Sample Weight} = \text{sample weight} \times e^{-\text{amount of say}}$$

$$= \frac{1}{8} e^{-\text{amount of say}}$$

$$= \frac{1}{8} e^{-0.97} = \frac{1}{8} \times 0.38 = 0.05$$



The new **Sample Weight** is **0.05**, which is *less* than the old one (**1/8** = **0.125**).

Step 5 – Creating New Dataset

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

Now we need to normalize the **New Sample Weights** so that they will add up to 1.

Step 5 – Creating New Dataset

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07

So we divide each **New Sample Weight** by **0.68** to get the normalized values.

Step 5 – Creating New Dataset

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

If the number is between **0** and **0.07**, then we would put this sample into the new collection of samples…

Step 5 – Creating New Dataset

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

If the number is between **0** and **0.07**, then we would put this sample into the new collection of samples…

Step 5 – Creating New Dataset

No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

…and if the number is between **0.07** and **0.14** (**0.07** + **0.07** = **0.14**), then we would put this sample into the new collection of samples…

Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07

…and if the number is between **0.14** and **0.21** (**0.14** + **0.07** = **0.21**), then we would put this sample into the new collection of samples…

Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

…and if the number is between **0.21** and **0.70** (**0.21** + **0.49** = **0.70**), then we would put this sample into the new collection of samples…

Step 5 – Creating New Dataset

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Ultimately, this sample was added to the new collection of samples **4** times, reflecting its larger **Sample Weight**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes
Yes	Yes	167	Yes
Yes	Yes	172	No
Yes	Yes	205	Yes
Yes	Yes	167	Yes

Remarks on Boosting

- Boosting can be applied without weights using re-sampling with probability determined by weights;
- Boosting decreases exponentially the training error in the number of iterations;
- Boosting works well if base classifiers are not too complex and their error doesn't become too large too quickly!
- Boosting reduces the bias component of the error of simple classifiers!

Decision Tree & Random Forest Algorithm

Some Practical Advices

•If the classifier is unstable (high variance), then apply bagging!

•If the classifier is stable and simple (high bias) then apply boosting!

•If the classifier is stable and complex then apply randomization injection!

•If you have many classes and a binary classifier then try error-correcting codes! If it does not work then use a complex binary classifier!

Outline

- λ Introduction
- λ Example of Decision Tree
- λ Principles of Decision Tree
  - Entropy
  - Information gain
- λ Random Forest

The problem

- Given a set of training cases/objects and their attribute values, try to determine the target attribute value of new examples.
  - Classification
  - Prediction

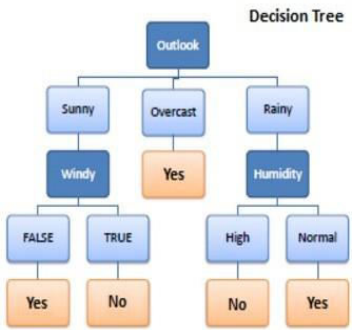
Key Requirements

- **Attribute-value description:** object or case must be expressible in terms of a fixed collection of properties or attributes (e.g., hot, mild, cold).
- **Predefined classes (target values):** the target function has **discrete output values** (bollean or multiclass)
- **Sufficient data:** enough training cases should be provided to learn the model.

A simple example

Predictors | Target

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



Principled Criterion

- Choosing the most useful attribute for classifying examples.
- Entropy
  - A measure of homogeneity of the set of examples
  - If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one
- Information Gain
  - Measures how well a given attribute separates the training examples according to their target classification
  - This measure is used to select among the candidate attributes at each step while growing the tree

Information Gain

Step 1 : Calculate entropy of the target

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

Information Gain (Cont'd)

Step 2 : Calculate information gain for each attribute

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = P(Sunny)*E(3,2) + P(Overcast)*E(4,0) + P(Rainy)*E(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = E(PlayGolf) – E(PlayGolf, Outlook)
= 0.940 – 0.693 = 0.247

Information Gain (Cont'd)

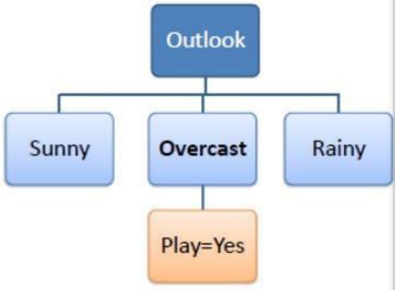
Step 3: Choose attribute with the largest information gain as the decision node.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

Information Gain (Cont'd)

Step 4a: A branch with entropy of 0 is a leaf node.

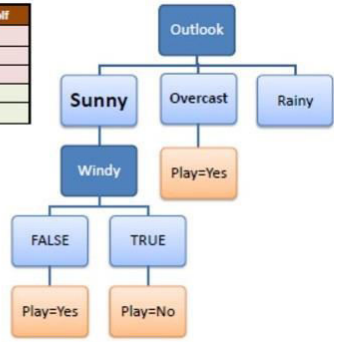
Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes
Hot	High	FALSE	Yes



Information Gain (Cont'd)

Step 4b: A branch with entropy more than 0 needs further splitting.

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Information Gain (Cont'd)

Step 5: The algorithm is run recursively on the non-leaf branches, until all data is classified.

Random Forest

- Decision Tree : one tree
- Random Forest : more than one tree

What is random forests

- An ensemble classifier using many decision tree models.
- Can be used for classification or Regression.
- Accuracy and variable importance information is provided with the results.

The Algorithm

- Let the number of training cases be N, and the numbe variables in the classifier be M.
- The number m of input variables are used to determine decision at a node of the tree; m should be much less than M.
- Choose a training set for this tree by choosing N times replacement from all N available training cases. Use the rest of cases to estimate the error of the tree, by predicting their classe
- For each node of the tree, randomly choose m variables on w to base the decision at that node. Calculate the best split base these m variables in the training set.
- Each tree is fully grown and not pruned.

- A Random Forest is a classifier consisting of collection of tree-structured classifiers {h(x, Θk), k 1....}where the Θk are independently, identicall distributed random trees and each tree casts a un vote for the final classification of input x. Lik CART, Random Forest uses the gini index fo determining the final class in each tree.
- The final class of each tree is aggregated and voted b weighted values to construct the final classifier.

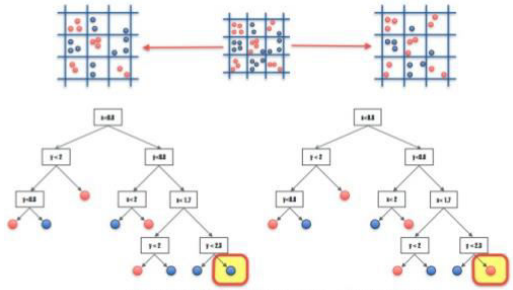
Advantages

- It produces a highly accurate classifier and learning is fast
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers or (by scaling) give interesting views of the data.
- It offers an experimental method for detecting variable interactions.

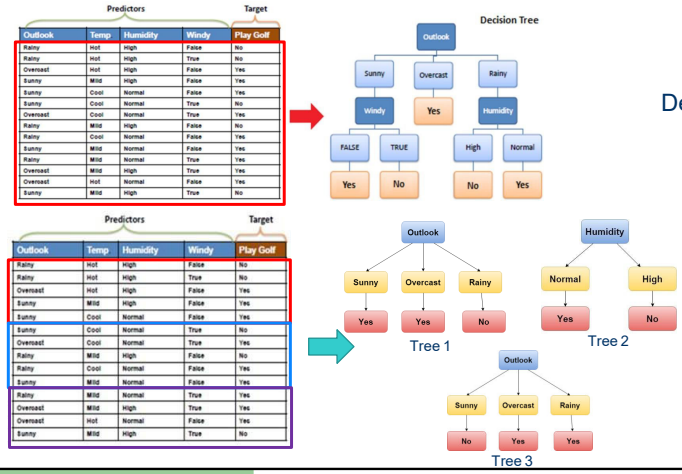
Random Forest

IIM तेजस्वि नाव

- **Problem:** Bagging still have relatively high variance
- **Goal:** Reduce variance of Bagging
- **Solution:** Along with sampling of data in Bagging, take samples of features also!
  - In other words, in building a random forest, at each split in the tree, the use only a random subset of features instead of all the features.
  - This de-correlates the trees.
  - Its mathematically proved that $\sqrt{Number\ of\ predictors}$ is a good approximate value for predictor subset size (mtry/max_features).
- **Evaluation:** A bootstrap sample uses only approximately 2/3 of the observations of original sample.
  - Remaining training data (OOB) are used to estimate error and variable importance



Decision Tree & Random Forest



Decision Tree

Random Forest

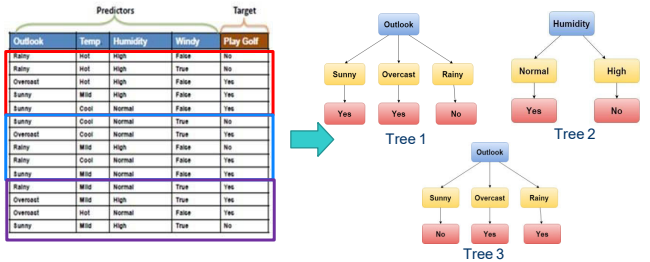
Decision Tree

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Mild	High	False	?

Result : No

Random Forest

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Mild	High	False	?



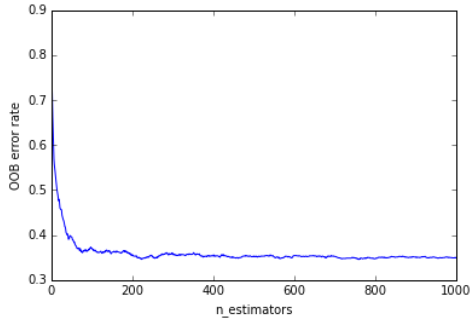
Tree 1 : No
Tree 2 : No
Tree 3 : Yes

Yes : 1
No : 2

Result : No

OOB Error Rate

- OOB error rate can be used to get a running unbiased estimate of the classification error as trees are added to the forest.



Stacking

Module 1: Regression Methods

When is it appropriate to perform a regression method? What regression models have we learned?

1. Linear Regression (simple, multiple, polynomial, interactions, model selection, Ridge & Lasso, etc...)
2. *k*-NN
3. Regression Trees

What is the main difference between these two types of models (advantages and disadvantages)? When should you use each method?

Module 2: Classification Methods

When is it appropriate to perform a classification method? What classification models have we learned?

1. Logistic Regression: same details as linear regression apply
2. *k*-NN
3. Discriminant Analysis: LDA/QDA
4. Classification Trees
5. SVM

What is the main difference between these two types of models (advantages and disadvantages)? When should you use each method?

Module 3: Ensemble Methods

What does it mean for a model to be an ensemble method?

1. Bagging Trees
2. Random Forests
3. Boosting Models
4. Neural Networks
5. Stacking Models (coming today)

What approach does each model take to improve prediction accuracy?

Bags and Forests of Trees (cont.)

Bagging:

- create an ensemble of full trees, each trained on a bootstrap sample of the training set;
- “average” the predictions

Random forest:

- create an ensemble of full trees, each trained on a bootstrap sample of the training set;
- in each tree and each split, randomly select a subset of predictors, choose a predictor from this subset for splitting;
- average the predictions

Note that the ensemble building aspects of both method are embarrassingly parallel!

Bags and Forests of Trees (cont.)

Boosting:

- Iteratively build a model from lots of little models.
- Each subsequent model predicts the residuals from the previous model, overweighting the large residuals.

Neural Networks:

- Build layers of models based on overly simple “neurons” of models.
- Uses back-propagation to efficiently communicate between output of models to update earlier models.

These methods are not as easily parallelizable.

Stacking

Motivation for Stacking

For each of our ensemble methods so far (besides Neural Networks), we have:

- Fit the base model on the same type (regression trees, for example).
- Combined the predictions in a naïve way.

Stacking is a way to generalize the ensembling approach to combine outputs of various types of model, and improves on the combination as well.

Motivation for Stacking

Recall that in boosting, the final model T, we learn is a weighted sum of simple models, T_h,

$$T = \sum_h \lambda_h T_H$$

where λ_h is the learning rate. In AdaBoost for example, we can analytically determine the optimal values of λ_h for each simple model T_h.

On the other hand, we can also determine the final model T implicitly by ***learning any model, called meta-learner, that transforms the outputs of T_h into a prediction***.

Stacked Generalization

The framework for ***stacked generalization*** or ***stacking*** (Wolpert 1992) is:

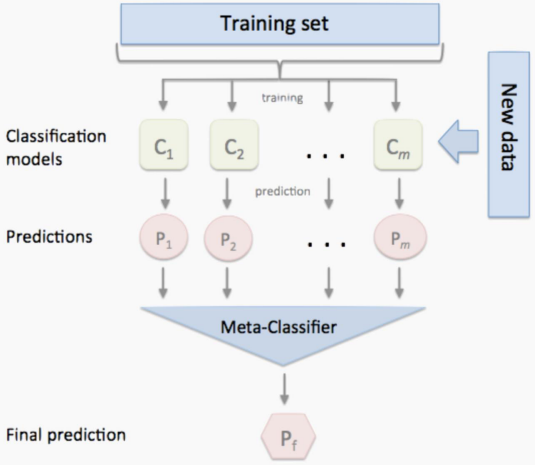
- train L number of models, T_l on the training data

$$\{(x_1, y_1), ..., (x_N, y_N)\}$$

- train a meta-learner $\tilde{T}$ on the predictions of the ensemble of models, i.e. train using the data

$$\{(T_1(x_1), ..., T_L(x_1), y_1), ..., (T_1(x_N), ..., T_L(x_N), y_N)\}$$

Stacking: an Illustration



Stacked Generalization

Stacking is a very general method,

- the models, T_l, in the ensemble can come from different classes. The ensemble can contain a mixture of logistic regression models, trees, etc.
- the meta-learner, T, can be of any type. **Note:** we want to train T on the ***out of sample*** predictions of the ensemble. For example we train T on

$$\{(T_1(x_1), ..., T_L(x_1), y_1), ..., (T_1(x_N), ..., T_L(x_N), y_N)\}$$

where $T_l(x_n)$ is generated by training T_l on

$$\{(x_1, y_1), ..., (x_{n-1}, y_{n-1}), (x_{n+1}, y_{n+1}), ...(x_N, y_N)\}$$

Stacking: General Guidelines

The flexibility of stacking makes it widely applicable but difficult to analyze theoretically. Some general rules have been found through empirical studies:

- models in the ensemble should be diverse, i.e. their errors should not be correlated.
- for binary classification, each model in the ensemble should have error rate < 1/2.
- if models in the ensemble outputs probabilities, it's better to train the meta-learner on probabilities rather than predictions.
- apply regularization to the meta-learner to avoid overfitting.

Stacking: Subsemble Approach

We can extend the stacking framework to include ensembles of models that specialize on small subsets of data (Sapp et. al. 2014), for de-correlation or improved computational efficiency:

- divide the data into J subsets
- train models, T_j , on each subset
- train a meta-learner $\tilde{T}$ on the predictions of the ensemble of models, i.e. train using the data

$$\{(T_1(x_1), ..., T_J(x_1), y_1), ..., (T_1(x_N), ..., T_J(x_N), y_N)\}$$

Again, we want to make sure that each $T_j(x_i)$ is an out of sample prediction.

Principal Components Analysis

Covariance

- Variance and Covariance are a measure of the "spread" of a set of points around their center of mass (mean)
- Variance – measure of the deviation from the mean for points in one dimension e.g. heights
- Covariance as a measure of how much each of the dimensions vary from the mean with respect to each other.
- Covariance is measured between 2 dimensions to see if there is a relationship between the 2 dimensions e.g. number of hours studied & marks obtained.
- The covariance between one dimension and itself is the variance

Covariance

$$\text{covariance}(X,Y) = \frac{\sum_{i=1}^{n} (\overline{X}_i - X)(\overline{Y}_i - Y)}{(n-1)}$$

- So, if you had a 3-dimensional data set (x,y,z), then you could measure the covariance between the x and y dimensions, the y and z dimensions, and the x and z dimensions. Measuring the covariance between x and x , or y and y , or z and z would give you the variance of the x , y and z dimensions respectively.

Covariance

- What is the interpretation of covariance calculations?
  e.g.: 2 dimensional data set
  x: number of hours studied for a subject
  y: marks obtained in that subject covariance value is
  say: 104.53
  what does this value mean?

Covariance Matrix

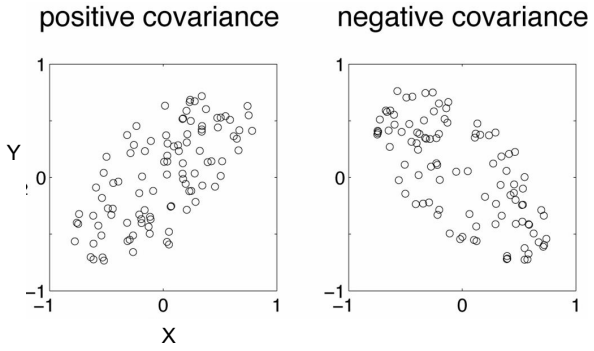
- Representing Covariance between dimensions as a matrix e.g. for 3 dimensions:

$$C = \begin{bmatrix} \text{cov}(x,x) & \text{cov}(x,y) & \text{cov}(x,z) \\ \text{cov}(y,x) & \text{cov}(y,y) & \text{cov}(y,z) \\ \text{cov}(z,x) & \text{cov}(z,y) & \text{cov}(z,z) \end{bmatrix}$$

Variances

- Diagonal is the variances of x, y and z
- cov(x,y) = cov(y,x) hence matrix is symmetrical about the diagonal
- N-dimensional data will result in NxN covariance matrix

Covariance examples



Covariance

- Exact value is not as important as it's sign.
- A positive value of covariance indicates both dimensions increase or decrease together e.g. as the number of hours studied increases, the marks in that subject increase.
- A negative value indicates while one increases the other decreases, or vice-versa e.g. active social life at PSU vs performance in CS dept.
- If covariance is zero: the two dimensions are independent of each other e.g. heights of students vs the marks obtained in a subject

PCA

- **principal components analysis (PCA)** is a technique that can be used to simplify a dataset
- It is a linear transformation that chooses a new coordinate system for the data set such that

  greatest variance by any projection of the data set comes to lie on the first axis (then called the first principal component),

  the second greatest variance on the second axis, and so on.
- PCA can be used for reducing dimensionality by eliminating the later principal components.

PCA Toy Example

Consider the following 3D points

1	2	4	3	5	6
2	4	8	6	10	12
3	6	12	9	15	18

If each component is stored in a byte, we need 18 = 3 x 6 bytes

PCA Toy Example

Looking closer, we can see that all the points are related geometrically: they are all the same point, scaled by a factor:

1	= 1 *	1	2	= 2 *	1	4	= 4 *	1
2		2	4		2	8		2
3		3	6		3	12		3
3	= 3 *	1	5	= 5 *	1	6	= 6 *	1
6		2	10		2	12		2
9		3	15		3	18		3

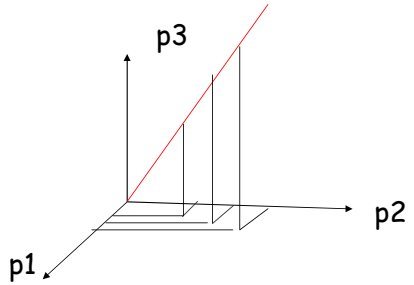
PCA Toy Example

1	= 1 *	1	2	= 2 *	1	4	= 4 *	1
2		2	4		2	8		2
3		3	6		3	12		3
3	= 3 *	1	5	= 5 *	1	6	= 6 *	1
6		2	10		2	12		2
9		3	15		3	18		3

They can be stored using only 9 bytes (50% savings!):
Store one point (3 bytes) + the multiplying constants (6 bytes)

Geometrical Interpretation:

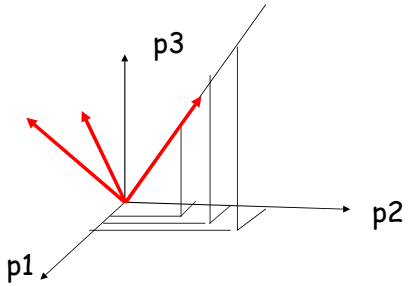
View each point in 3D space.



But in this example, all the points happen to belong to a line: a 1D subspace of the original 3D space.

Geometrical Interpretation:

Consider a new coordinate system where one of the axes is along the direction of the line:



In this coordinate system, every point has only one non-zero coordinate: we *only* need to store the direction of the line (a 3 bytes image) and the non-zero coordinate for each of the points (6 bytes).

Principal Component Analysis (PCA)

- Given a set of points, how do we know if they can be compressed like in the previous example?
  - The answer is to look into the correlation between the points
  - The tool for doing this is called PCA

PCA

- By finding the eigenvalues and eigenvectors of the covariance matrix, we find that the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset.
- This is the principal component.
- PCA is a useful statistical technique that has found application in:
  - fields such as face recognition and image compression
  - finding patterns in data of high dimension.

PCA Theorem

Let $x_1\ x_2\ \ldots\ x_n$ be a set of n N x 1 vectors and let $\bar{x}$ be their average:

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN} \end{bmatrix} \qquad \bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{i=n} \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN} \end{bmatrix}$$

Steps Involved in the PCA

Step 1: Standardize the dataset.
Step 2: Calculate the covariance matrix for the features in the dataset.
Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.
Step 4: Sort eigenvalues and their corresponding eigenvectors.
Step 5: Pick k eigenvalues and form a matrix of eigenvectors.
Step 6: Transform the original matrix.

PCA Theorem

Let X be the N x n matrix with columns $x_1 - x,\ x_2 - x,\ldots\ x_n - x$:

$$X = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix}$$

Note: subtracting the mean is equivalent to translating the coordinate system to the location of the mean.

PCA Theorem

Let Q = X X^T be the N x N matrix:

$$Q = XX^T = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix} \begin{bmatrix} (\mathbf{x}_1 - \bar{\mathbf{x}})^T \\ (\mathbf{x}_2 - \bar{\mathbf{x}})^T \\ \vdots \\ (\mathbf{x}_n - \bar{\mathbf{x}})^T \end{bmatrix}$$

Notes:

1. Q is square
2. Q is symmetric
3. Q is the *covariance* matrix [aka scatter matrix]
4. Q can be very large (in vision, N is often the number of pixels in an image!)

PCA Theorem

Theorem:

Each x_j can be written as: $\mathbf{x}_j = \bar{\mathbf{x}} + \sum_{i=1}^{i=n} g_{ji}\mathbf{e}_i$

where e_i are the n eigenvectors of Q with non-zero eigenvalues.

Notes:

1. The eigenvectors e_1 e_2 … e_n span an ***eigenspace***
2. e_1 e_2 … e_n are N x 1 orthonormal vectors (directions in N-Dimensional space)
3. The scalars g_{ji} are the coordinates of x_j in the space.

$$g_{ji} = (\mathbf{x}_j - \bar{\mathbf{x}}).\mathbf{e}_i$$

Using PCA to Compress Data

- Expressing x in terms of e_1 … e_n has not changed the size of the data
- However, if the points are highly correlated many of the coordinates of x will be zero or closed to zero.

note: this means they lie in a lower-dimensional linear subspace

Using PCA to Compress Data

- Sort the eigenvectors e_i according to their eigenvalue:

$$\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n$$

- Assuming that $\lambda_i \approx 0$ if $i > k$
- Then

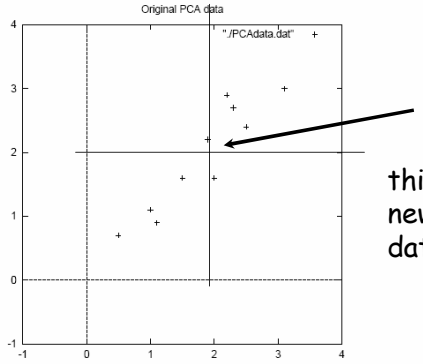
$$\mathbf{x}_j \approx \bar{\mathbf{x}} + \sum_{i=1}^{i=k} g_{ji}\mathbf{e}_i$$

PCA Example –STEP 1

http://kybele.psych.cornell.edu/~edelman/Psych-465-Spring-2003/PCA-tutorial.pdf

- DATA:

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



mean

this becomes the new origin of the data from now on

Figure 3.1: PCA example data, original data on the left, data with the means subtracted on the right, and a plot of the data

PCA Example –STEP 2

- Calculate the covariance matrix

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

- since the non-diagonal elements in this covariance matrix are positive, we should expect that both the x and y variable increase together.

PCA Example –STEP 3

- Calculate the eigenvectors and eigenvalues of the covariance matrix

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -735178656 \end{pmatrix}$$

PCA Example –STEP 3

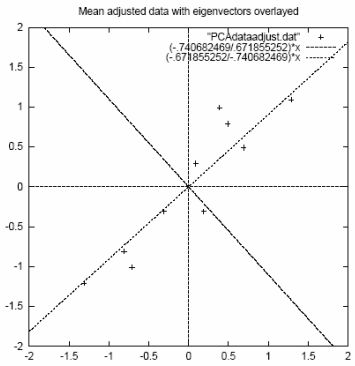


Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlayed on top.

- eigenvectors are plotted as diagonal dotted lines on the plot.
- Note they are perpendicular to each other.
- Note one of the eigenvectors goes through the middle of the points, like drawing a line of best fit.
- The second eigenvector gives us the other, less important, pattern in the data, that all the points follow the main line, but are off to the side of the main line by some amount.

PCA Example –STEP 4

- Feature Vector

  FeatureVector = (eig_1 eig_2 eig_3 … eig_n)

  We can either form a feature vector with both of the eigenvectors:

$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

  or, we can choose to leave out the smaller, less significant component and only have a single column:

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

PCA Example –STEP 5

- Deriving new data coordinates

  FinalData = RowFeatureVector x RowZeroMeanData

  RowFeatureVector is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top

  RowZeroMeanData is the mean-adjusted data *transposed*, ie. the data items are in each column, with each row holding a separate dimension.

  Note: his is essential Rotating the coordinate axes so higher-variance axes come first.

PCA Example –STEP 5

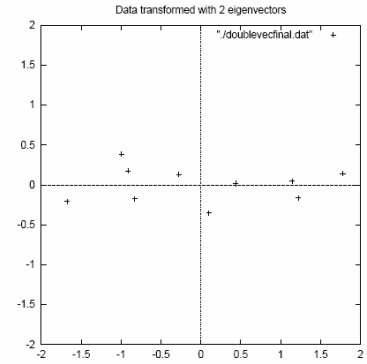
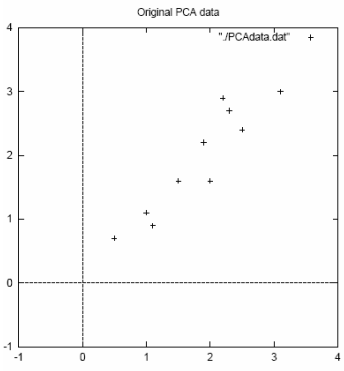


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

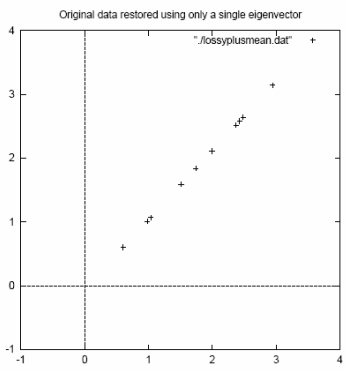
PCA Example : Approximation

- If we reduced the dimensionality, obviously, when reconstructing the data we would lose those dimensions we chose to discard. In our example let us assume that we considered only the x dimension…

PCA Example : Final Approximation



2D point cloud

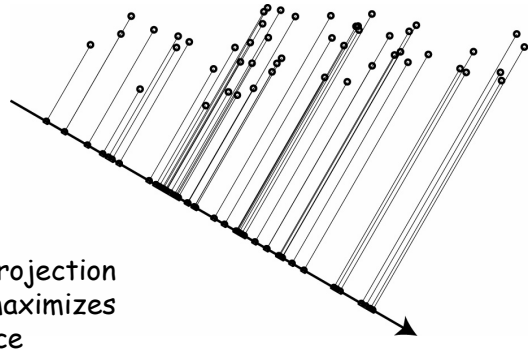


Approximation using one eigenvector basis

Another way of thinking about Principal component

- direction of maximum variance in the input space
- happens to be same as the principal eigenvector of the covariance matrix

One-dimensional projection



find projection that maximizes variance

Covariance to variance

- From the covariance, the variance of any projection can be calculated.
- Let *w* be a unit vector

$$\left\langle \left(w^T x\right)^2 \right\rangle - \left\langle w^T x \right\rangle^2 = w^T C w$$

$$= \sum_{ij} w_i C_{ij} w_j$$

Maximizing variance

- Principal eigenvector of C
  - the one with the largest eigenvalue.

$$w^* = \arg\max_{w:|w|=1} w^T C w$$

$$\lambda_{\max}(C) = \max_{w:|w|=1} w^T C w$$

$$= w^{*T} C w^*$$

Singular Value Decomposition (SVD)

Any m x n matrix X can be written as the product of 3 matrices:

$$X = UDV^T$$

Where:

- U is m x m and its columns are orthonormal vectors
- V is n x n and its columns are orthonormal vectors
- D is m x n diagonal and its diagonal elements are called the singular values of X, and are such that:

$\sigma_1 , \sigma_2 , \ldots \sigma_n , 0$

Implementing PCA

- Need to find "first" k eigenvectors of Q:

$$Q = XX^T = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix} \begin{bmatrix} (\mathbf{x}_1 - \bar{\mathbf{x}})^T \\ (\mathbf{x}_2 - \bar{\mathbf{x}})^T \\ \vdots \\ (\mathbf{x}_n - \bar{\mathbf{x}})^T \end{bmatrix}$$

Q is N x N (Again, N could be the number of pixels in an image. For a 256 x 256 image, N = 65536 !!)
Don't want to explicitly compute Q!!!!

SVD Properties

$$X = UDV^T$$

- The columns of U are the eigenvectors of XX^T
- The columns of V are the eigenvectors of X^TX
- The squares of the diagonal elements of D are the eigenvalues of XX^T and X^TX

DIMENSIONALITY REDUCTION: PCA, MDS

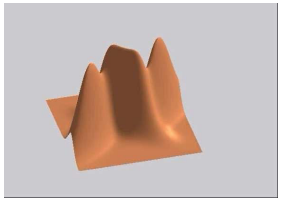
slides are due to L.Saul , A. Ng, and A. Ghodsi

Topics

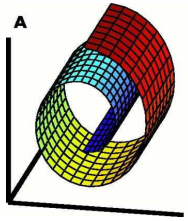
- PCA
- MDS
- IsoMap
- LLE
- EigenMaps

Types of Structure in High Dimension

- Clumps
  - Clustering
  - Density Estimation



- Low Dimensional Manifolds
  - Linear
  - NonLinear



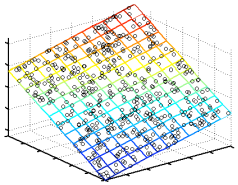
Dimensionality Reduction

- Data representation

  Inputs are real-valued vectors in a high dimensional space.

- Linear structure

  Does the data live in a low dimensional subspace?



- Nonlinear structure

  Does the data live on a low dimensional submanifold?

Dimensionality Reduction

- Question

  How can we detect low dimensional structure in high dimensional data?

- Applications
  - Digital image and speech processing
  - Analysis of neuronal populations
  - Gene expression microarray data
  - Visualization of large networks

Notations

- Inputs (**high dimensional**)

  $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$ points in R^D

- Outputs **(low dimensional)**

  $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_n}$ points in R^d ($d << D$)

- Goals

  Nearby points remain nearby.

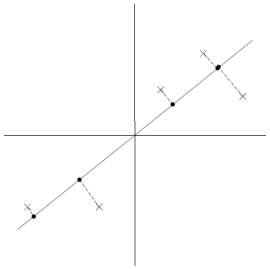
  Distant points remain distant.

Linear Methods

- PCA
- MDS

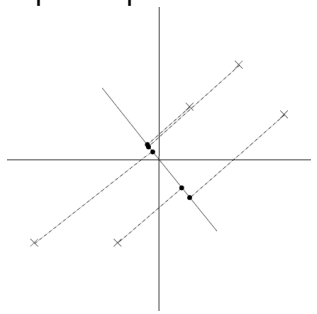
Principle Component Analysis

good representation



the projected data has a fairly large variance, and the points tend to be far from zero.

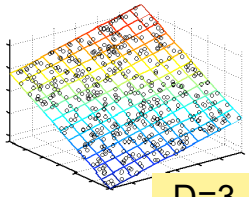
poor representation



the projections have a significantly smaller variance, and are much closer to the origin.

Principle Component Analysis

D=2, d=1



D=3, d=2

- Seek most accurate data representation in a lower dimensional space.
- The good direction/subspace to use for projection lies in the direction of largest variance.

Maximum Variance Subspace

- Assume inputs are centered: $\sum_i x_i = 0$
- Given a unit vector u and a point x, the length of the projection of x onto u is given by $x^T u$
- Maximize projected variance:

$$\text{var}(y) = \frac{1}{n}\sum_i \left(x_i^T u\right)^2 = \frac{1}{n}\sum_i u^T x_i x_i^T u$$

$$= u^T \left(\frac{1}{n}\sum_i x_i x_i^T\right) u$$

1D Subspace

- Maximizing $u^T C u$ subject to $\|u\| = 1$

where $C = n^{-1}\sum_i x_i x_i^T$ is the empirical

covariance matrix of the data,

gives the principle eigenvector of C.

d-dimensional Subspace

- to project the data into a d-dimensional subspace (d <<D), we should choose $u_1,\ldots,u_d$ to be the top d eigenvectors of C.
- $u_1,\ldots,u_d$ now form a new, orthogonal basis for the data.
- The low dimensional representation of x is given by

$$y_i = \begin{bmatrix} u_1^T x_i \\ u_2^T x_i \\ \vdots \\ u_k^T x_i \end{bmatrix} \in \Re^d.$$

Interpreting PCA

- Eigenvectors:
  principal axes of maximum variance subspace.
- Eigenvalues:
  variance of projected inputs along principle axes.
- Estimated dimensionality:
  number of significant (nonnegative) eigenvalues.

PCA summary

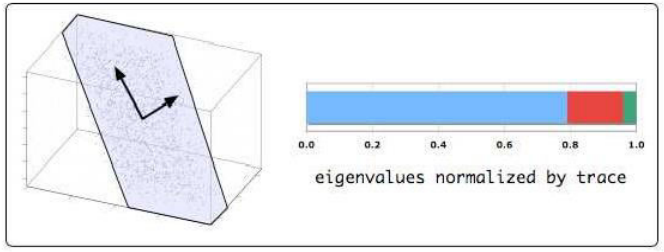
Input: $z_i \in R^D,\ i=1,..,n$ Output: $y_i \in R^d,\ i=1,..,n$

1. Subtract sample mean from the data
$$x_i = z_i - \hat{\mu}, \quad \hat{\mu} = 1/n \sum_i z_i$$
2. Compute the covariance$_n$ matrix
$$C = 1/n \sum_{i=1} x_i x_i^t$$
3. Compute eigenvectors $\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_d}$ corresponding to the ***d*** largest eigenvalues of C (d<<D).
4. The desired y is
$$y = P^t x, \ P = [e_1, \ldots, e_d]$$

Equivalence

- PCA finds the directions that have the most variance.
$$\text{var}(y) = \frac{1}{n} \sum_i \left\| P^T x_i \right\|^2$$
- Same result can be obtained by minimizing the squared reconstruction error.
$$err(y) = \frac{1}{n} \sum_i \left\| x_i - PP^T x_i \right\|^2$$

Example of PCA



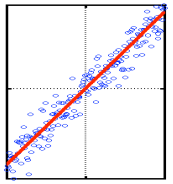
Eigenvectors and eigenvalues of covariance matrix for *n=1600* inputs in *d=3 dimensions.*

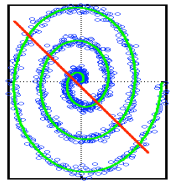
Example: faces

Eigenfaces from 7562 Images:
top left image is linear combination of the rest.
Sirovich & Kirby (1987)
Turk & Pentland (1991)

Properties of PCA

- Strengths:
  - Eigenvector method
  - No tuning parameters
  - Non-iterative
  - No local optima





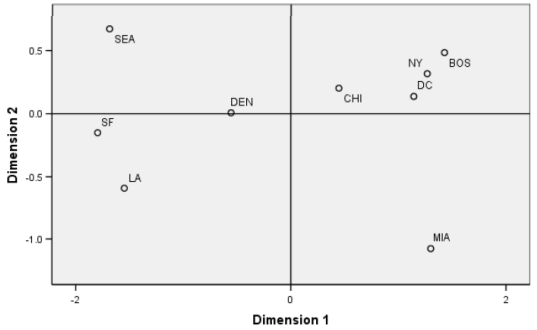
- Weaknesses:
  - Limited to second order statistics
  - Limited to linear projections

Multidimensional Scaling (MDS)

- MDS attempts to preserve pairwise distances.
- Attempts to construct a configuration of n points in Euclidian space by using the information about the distances between the n patterns.

Example : Distances between US Cities

	BOS	CHI	DC	DEN	LA	MIA	NY	SEA	SF
BOS	0	963	429	1,949	2,979	1,504	206	2,976	3,095
CHI	963	0	671	996	2,054	1,329	802	2,013	2,142
DC	429	671	0	1,616	2,631	1,075	233	2,684	2,799
DEN	1,949	996	1,616	0	1,059	2,037	1,771	1,307	1,235
LA	2,979	2,054	2,631	1,059	0	2,687	2,786	1,131	379
MIA	1,504	1,329	1,075	2,037	2,687	0	1,308	3,273	3,053
NY	206	802	233	1,771	2,786	1,308	0	2,815	2,934
SEA	2,976	2,013	2,684	1,307	1,131	3,273	2,815	0	808
SF	3,095	2,142	2,799	1,235	379	3,053	2,934	808	0



Multidimensional Scaling (MDS)

- A $n \times n$ matrix $\mathcal{D}$ is called a distance or affinity matrix if it is symmetric, $\mathtt{d}_{ii} = 0$, and $\mathtt{d}_{ij} > 0, \quad i \neq j$.
- Given a distance matrix $\mathcal{D}^{(X)}$, MDS attempts to find n data points $y_1, ..., y_n$ in d dimensions, such that if $d_{ij}^{(Y)}$ denotes the Euclidean distance between y_i and y_j, then $\mathcal{D}^Y$ is similar to $\mathcal{D}^{(X)}$.

Metric MDS

- Metric MDS minimizes

$$\min_{Y} \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij}^{(X)} - d_{ij}^{(Y)})^2$$

where

$$d_{ij}^{(X)} = \left\| x_i - x_j \right\| \quad \text{and} \quad d_{ij}^{(Y)} = \left\| y_i - y_j \right\|.$$

Metric MDS

- The distance matrix $D^{(X)}$ can be converted to a Gram matrix K by

$$K = -\frac{1}{2} H (D^{(X)})^2 H$$

where $H = I - \frac{1}{n} e e^T$ and e is the vector of ones.

Metric MDS

- K is *p.s.d*, thus it can be written as $K = X^T X$
- $\min_{Y} \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij}^{(X)} - d_{ij}^{(Y)})^2$ is equivalent to

$$\min_{Y} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i^T x_j - y_i^T y_j)^2$$

- The norm can be converted to a trace:

$$\min_{Y} Tr\left(X^T X - Y^T Y\right)^2$$

Metric MDS

- Using Singular Value Decomposition we can decompose:

$$X^T X = V \Lambda V^T$$

$$Y^T Y = Q \hat{\Lambda} Q^T$$

- Since $Y^T Y$ is *p.s.d.*, $\hat{\Lambda}$ has no negative values, thus

$$Y = \hat{\Lambda}^{1/2} Q^T$$

Metric MDS

- Returning to the minimization, we can write

$$\min_{Q,\hat{\Lambda}} Tr\left(V \Lambda V^T - Q \hat{\Lambda} Q^T\right)^2$$

$$= \min_{Q,\hat{\Lambda}} Tr\left(\Lambda - \underbrace{V^T Q}_{G} \hat{\Lambda} Q^T V\right)^2$$

$$= \min_{G,\hat{\Lambda}} Tr\left(\Lambda - G \hat{\Lambda} G^T\right)^2$$

$$= \min_{G,\hat{\Lambda}} Tr\left(\Lambda^2 + G \hat{\Lambda} G^T G \hat{\Lambda} G^T - 2 \Lambda G \hat{\Lambda} G^T\right)$$

Metric MDS

- For a fixed $\hat{\Lambda}$ we can minimize for G, obtaining

$$G = I$$

$$\min_{\hat{\Lambda}} Tr\left(\Lambda^2 + \hat{\Lambda}^2 - 2 \Lambda \hat{\Lambda} G\right)$$

$$= \min_{\hat{\Lambda}} Tr\left(\Lambda - \hat{\Lambda}\right)^2$$

Metric MDS

- To make the two matrices Λ and $\hat{\Lambda}$ similar, we can make $\hat{\Lambda}$ to be the top d diagonal elements of Λ.
- Also $G = V^T Q$ and $G = I$ imply that $V = Q$.
- Therefore,

$$Y = \hat{\Lambda}^{1/2} Q^T \quad \Rightarrow \quad Y = \hat{\Lambda}^{1/2} V^T$$

where V comprises the eigenvectors of $X^T X$ corresponding to the top d eigenvalues and $\hat{\Lambda}$ comprises the top d eigenvalues of $X^T X$.

Interpreting MDS

- Eigenvectors:
  - Ordered, scaled, and truncated to yield low dimensional embedding.
- Eigenvalues:
  - Measure how each dimension contributes to dot products.
- Estimated dimensionality:
  - Number of significant (nonnegative) eigenvalues.

Relation to PCA

	PCA	MDS
Spectral Decomposition	Covariance matrix (D x D)	Gram matrix (n x n)
Eigenvalues	Matrices share nonzero eigenvalues up to constant factor	
Results	Same	
Computation	$O((n+d)D^2)$	$O((D+d)n^2)$

Non-Metric MDS

- Transform pairwise distances: $\delta_{ij} \rightarrow g(\delta_{ij})$
  - Transformation: nonlinear, but monotonic.
  - Preserves rank order of distances.
- Find vectors y_i such that $\|y_i - y_j\| \approx g(\delta_{ij})$

$$Cost = \min_{y} \sum_{ij} \left(g(\delta_{ij}) - \|y_i - y_j\| \right)^2$$

Non-Metric MDS

- Possible objective function:

$$Cost = \sum_{ij} \left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\| - \|\mathbf{y}_i - \mathbf{y}_j\|}{\|\mathbf{x}_i - \mathbf{x}_j\|} \right)^2$$

Properties of non-metric MDS

- Strengths
  - Relaxes distance constraints.
  - Yields nonlinear embeddings.
- Weaknesses
  - Highly nonlinear, iterative optimization with local minima.
  - Unclear how to choose distance transformation.

Steps Involved in the PCA

Step 1: Standardize the dataset.
Step 2: Calculate the covariance matrix for the features in the dataset.
Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.
Step 4: Sort eigenvalues and their corresponding eigenvectors.
Step 5: Pick k eigenvalues and form a matrix of eigenvectors.
Step 6: Transform the original matrix.

1. Standardize the Dataset

Assume we have the below dataset which has 4 features and a total of 5 training examples.

f1	f2	f3	f4
1	2	3	4
5	5	6	7
1	4	2	3
5	3	2	1
8	1	2	2

Dataset matrix

First, we need to standardize the dataset and for that, we need to calculate the mean and standard deviation for each feature.

$$x_{new} = \frac{x - \mu}{\sigma}$$

Standardization formula

	f1	f2	f3	f4
μ =	4	3	3	3.4
σ =	3	1.58114	1.73205	2.30217

Mean and standard deviation before standardization
After applying the formula for each feature in the dataset is transformed as below:

f1	f2	f3	f4
-1	-0.63246	0	0.26062
0.33333	1.26491	1.73205	1.56374
-1	0.63246	-0.57735	-0.17375
0.33333	0	-0.57735	-1.04249
1.33333	-1.26491	-0.57735	-0.60812

Standardized Dataset

2. Calculate the covariance matrix for the whole dataset

The formula to calculate the covariance matrix:

For Population

$$Cov(x,y) = \frac{\Sigma (x - \bar{x}) * (y - \bar{y})}{N}$$

For Sample

$$Cov(x,y) = \frac{\Sigma (x - \bar{x}) * (y - \bar{y})}{(N-1)}$$

Covariance Formula the covariance matrix for the given dataset will be calculated as below

	f1	f2	f3	f4
f1	var(f1)	cov(f1,f2)	cov(f1,f3)	cov(f1,f4)
f2	cov(f2,f1)	var(f2)	cov(f2,f3)	cov(f2,f4)
f3	cov(f3,f1)	cov(f3,f2)	var(f3)	cov(f3,f4)
f4	cov(f4,f1)	cov(f4,f2)	cov(f4,f3)	var(f4)

Since we have standardized the dataset, so the **mean for each feature is 0** and the standard deviation is 1.

var(f1) = ((-1.0-0)² + (0.33-0)² + (-1.0-0)² +(0.33−0)² +(1.33−0)²)/5
var (f1) = 0.8

cov(f1,f2) =
((-1.0−0)*(-0.632456-0) +
(0.33−0)*(1.264911-0) +
(-1.0−0)* (0.632456-0)+
(0.33−0)*(0.000000 -0)+
(1.33−0)*(-1.264911−0))/5
cov(f1,f2 = -0.25298

In the similar way be can calculate the other covariances and which will result in the below covariance matrix

	f1	f2	f3	f4
f1	0.8	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8	0.51121	0.4945
f3	0.03849	0.51121	0.8	0.75236
f4	-0.14479	0.4945	0.75236	0.8

covariance matrix (population formula)

3. Calculate eigenvalues and eigen vectors.

An **eigenvector** is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding **eigenvalue** is the factor by which the eigenvector is scaled.

Let A be a square matrix (in our case the covariance matrix), v a vector and λ a scalar that satisfies Av = λv, then λ is called eigenvalue associated with eigenvector v of A. Rearranging the above equation,

Av-λv =0 ; (A-λI)v = 0

Since we have already know v is a non- zero vector, only way this equation can be equal to zero, if

det(A-λI) = 0

	f1	f2	f3	f4
f1	0.8 - λ	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8- λ	0.51121	0.4945
f3	0.03849	0.51121	0.8 - λ	0.75236
f4	-0.14479	0.4945	0.75236	0.8 - λ

A-λI = 0

Solving the above equation = 0

λ = 2.51579324 , 1.0652885 , 0.39388704 , 0.02503121

Eigenvectors:

Solving the (A-λI)v = 0 equation for v vector with different λ values:

$$\begin{pmatrix} 0.800000-\lambda & -0.252982 & 0.038490 & -0.144791 \\ -0.252982 & 0.800000-\lambda & 0.511208 & 0.494498 \\ 0.038490 & 0.511208 & 0.800000-\lambda & 0.752355 \\ -0.144791 & 0.494498 & 0.752355 & 0.800000-\lambda \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = 0$$

For λ = *2.51579324, solving the above equation using Cramer's rule, the values for v vector are*

v1 = 0.16195086
v2 = -0.52404813
v3 = -0.58589647
v4 = -0.59654663

Going by the same approach, we can calculate the eigen vectors for the other eigen values. We can from a matrix using the eigen vectors.

e1	e2	e3	e4
0.161960	-0.917059	-0.307071	0.196162
-0.524048	0.206922	-0.817319	0.120610
-0.585896	-0.320539	0.188250	-0.720099
-0.596547	-0.115935	0.449733	0.654547

eigenvectors(4 * 4 matrix)

***4.* Sort eigenvalues and their corresponding eigenvectors.**

Since eigenvalues are already sorted in this case so no need to sort them again.

5. Pick k eigenvalues and form a matrix of eigenvectors

If we choose the top 2 eigenvectors, the matrix will look like this:

```
       e1        e2
 0.161960 -0.917059
-0.524048  0.206922
-0.585896 -0.320539
-0.596547 -0.115935
```

Top 2 eigenvectors(4*2 matrix)

6. Transform the original matrix.

Feature matrix * top k eigenvectors = Transformed Data

```
       f1        f2        f3        f4              e1        e2          nf1       nf2
-1.000000 -0.632456  0.000000  0.260623        0.161960 -0.917059    0.014003  0.755975
 0.333333  1.264911  1.732051  1.563740   *   -0.524048  0.206922  =  -2.556534 -0.780432
-1.000000  0.632456 -0.577350 -0.173749       -0.585896 -0.320539   -0.051480  1.253175
 0.333333  0.000000 -0.577350 -1.042493       -0.596547 -0.115935    1.014150  0.000239
 1.333333 -1.264911 -0.577350 -0.608121                             1.579803 -1.228917
                        (5,4)                     (4,2)                       (5,2)
```

Data Transformation

Locally Linear Embedding LLE

Contents

1.Introduction

- High dimensional data appears frequently in statistical pattern recognition
- But the further processing does not require all the properties of the data
- We could reduce the dimension without hardly affecting the relevant features of the data
- less dimensionality ?less required space less processing time

2.What is LLE? and what can it do?

- LLE is a dimensionality reduction algorithm
- Like PCAMDS
- It is a eigenvector method
- It models linear variabilities in high dimensional data
- Simple to implement
- Its optimizations do not get into local minima

2.What is LLE? and what can it do?

- The projection of LLE identifies the underlying structures of the manifold
- PCA metric MDS project faraway points to nearby points
- capable of generating highly nonlinear embeddings
- based on simple geometric intuitions
- used in audiovisual speech synthesis
- and in visual pattern recognition

3.The Algorithm

- Compute the neighbors of each data point Xi
- Compute the weights Wij that best reconstruct each data point Xi from its neighbors by minimizing the reconstruction error rate
- Compute the vectors Yi best reconstructed by the weights Wij again by minimizing the error rate

3.1.Computing the neighbors

- There are many ways of determining the neighbor points of X
- assuming a fixed number N of neighbors for each point and compute them (compute the N nearest points to X)
- choosing all the points within a ball of fixed radius (X is the center of the ball)

3.2.Computing the weigh matrix

- We should minimize the reconstruction error which
  is represented by the equation

3.2.Computing the weigh matrix(mathematical stuff)

- Constrainted Least Squares Problem

3.2.Computing the weigh matrixCntd.

- The sum of weights for each point should be 1
- You can achieve this task in different ways e.g.
- Constraint satisfaction
- A neural network
- A genetic Algorithm!
- Eigenvectors (it comes from the German word Eigenvektoren))

3.3.Computing the projections

- Every original data point Xi is mapped to a low dimensional vector Yi
- Then we calculate the Yi by using the Wij from the last step and minimizing the following embedding cost function

3.3.Computing the projections(mathematical stuff)

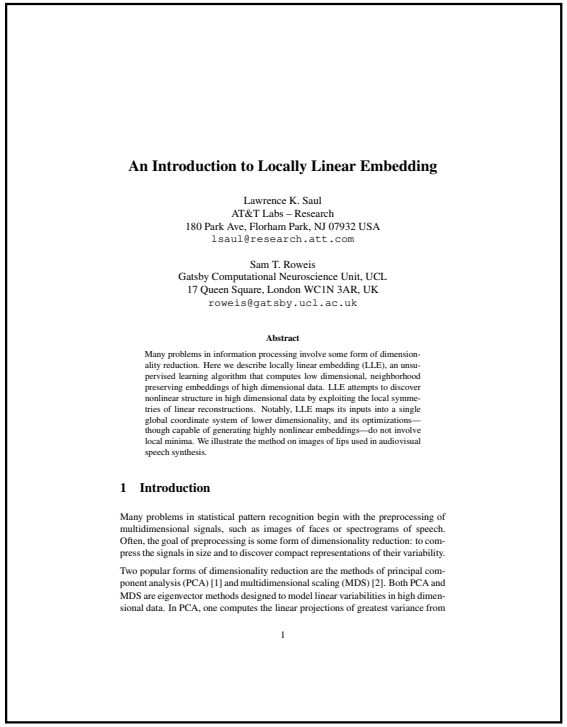
- Eigenvectors

5.final

- When the distances between the points are the important factor in your pattern recognition algorithm
- and you do not want your algorithm to take hours to be ready but your computer is a serial slow machine
- Then apply LLE before you start!

6.References

- An Introduction to Locally Linear Embedding (L.K.Saul S.T.Roweis)
- Think globally, fit locally Unsupervised Learning of Low Dimensional Manifolds
- Journal of Machine Learning Research 4 (2003) 119-155
- A mathematical resource Lineare Algebra Prof.Dr. Vogt WS02/03 Skript Universität Osnabrück



An Introduction to Locally Linear Embedding

Lawrence K. Saul
AT&T Labs – Research
180 Park Ave, Florham Park, NJ 07932 USA
lsaul@research.att.com

Sam T. Roweis
Gatsby Computational Neuroscience Unit, UCL
17 Queen Square, London WC1N 3AR, UK
roweis@gatsby.ucl.ac.uk

Abstract

Many problems in information processing involve some form of dimensionality reduction. Here we describe locally linear embedding (LLE), an unsupervised learning algorithm that computes low dimensional, neighborhood preserving embeddings of high dimensional data. LLE attempts to discover nonlinear structure in high dimensional data by exploiting the local symmetries of linear reconstructions. Notably, LLE maps its inputs into a single global coordinate system of lower dimensionality, and its optimizations—though capable of generating highly nonlinear embeddings—do not involve local minima. We illustrate the method on images of lips used in audiovisual speech synthesis.

1 Introduction

Many problems in statistical pattern recognition begin with the preprocessing of multidimensional signals, such as images of faces or spectrograms of speech. Often, the goal of preprocessing is some form of dimensionality reduction: to compress the signals in size and to discover compact representations of their variability.

Two popular forms of dimensionality reduction are the methods of principal component analysis (PCA) [1] and multidimensional scaling (MDS) [2]. Both PCA and MDS are eigenvector methods designed to model linear variabilities in high dimensional data. In PCA, one computes the linear projections of greatest variance from

the top eigenvectors of the data covariance matrix. In classical (or metric) MDS, one computes the low dimensional embedding that best preserves pairwise distances between data points. If these distances correspond to Euclidean distances, the results of metric MDS are equivalent to PCA. Both methods are simple to implement, and their optimizations do not involve local minima. These virtues account for the widespread use of PCA and MDS, despite their inherent limitations as linear methods.

Recently, we introduced an eigenvector method—called locally linear embedding (LLE)—for the problem of nonlinear dimensionality reduction[4]. This problem is illustrated by the nonlinear manifold in Figure 1. In this example, the dimensionality reduction by LLE succeeds in identifying the underlying structure of the manifold, while projections of the data by PCA or metric MDS map faraway data points to nearby points in the plane. Like PCA and MDS, our algorithm is simple to implement, and its optimizations do not involve local minima. At the same time, however, it is capable of generating highly nonlinear embeddings. Note that mixture models for local dimensionality reduction[5, 6], which cluster the data and perform PCA within each cluster, do not address the problem considered here—namely, how to map high dimensional data into a single global coordinate system of lower dimensionality.

In this paper, we review the LLE algorithm in its most basic form and illustrate a potential application to audiovisual speech synthesis[3].

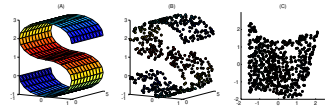


Figure 1: The problem of nonlinear dimensionality reduction, as illustrated for three dimensional data (B) sampled from a two dimensional manifold (A). An unsupervised learning algorithm must discover the global internal coordinates of the manifold without signals that explicitly indicate how the data should be embedded in two dimensions. The shading in (C) illustrates the neighborhood-preserving mapping discovered by LLE.

data that are invariant to exactly such transformations. We therefore expect their characterization of local geometry in the original data space to be equally valid for local patches on the manifold. In particular, the same weights W_{ij} that reconstruct the ith data point in D dimensions should also reconstruct its embedded manifold coordinates in d dimensions.

(Informally, imagine taking a pair of scissors, cutting out locally linear patches of the underlying manifold, and placing them in the low dimensional embedding space. Assume further that this operation is done in a way that preserves the angles formed by each data point to its nearest neighbors. In this case, the transplantation of each patch involves no more than a translation, rotation, and rescaling of its data, exactly the operations to which the weights are invariant. Thus, when the patch arrives at its low dimensional destination, we expect the same weights to reconstruct each data point from its neighbors.)

LLE constructs a neighborhood preserving mapping based on the above idea. In the final step of the algorithm, each high dimensional observation $\vec{X}_i$ is mapped to a low dimensional vector $\vec{Y}_i$ representing global internal coordinates on the manifold. This is done by choosing d-dimensional coordinates $\vec{Y}_i$ to minimize the embedding cost function:

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2 . \qquad (2)$$

This cost function—like the previous one—is based on locally linear reconstruction errors, but here we fix the weights W_{ij} while optimizing the coordinates $\vec{Y}_i$. The embedding cost in Eq. (2) defines a quadratic form in the vectors $\vec{Y}_i$. Subject to constraints that make the problem well-posed, it can be minimized by solving a sparse $N \times N$ eigenvector problem, whose bottom d non-zero eigenvectors provide an ordered set of orthogonal coordinates centered on the origin. Details of this eigenvector problem are discussed in Appendix B.

Note that while the reconstruction weights for each data point are computed from its local neighborhood—independent of the weights for other data points—the embedding coordinates are computed by an $N \times N$ eigensolver, a global operation that couples all data points in connected components of the graph defined by the weight matrix. The different dimensions in the embedding space can be computed successively; this is done simply by computing the bottom eigenvectors from eq. (2) one at a time. But the computation is always coupled across data points. This is how the algorithm leverages overlapping local information to discover global structure.

Implementation of the algorithm is fairly straightforward, as the algorithm has only one free parameter: the number of neighbors per data point, K. Once neighbors

2 Algorithm

The LLE algorithm, summarized in Fig. 2, is based on simple geometric intuitions. Suppose the data consist of N real-valued vectors $\vec{X}_i$, each of dimensionality D, sampled from some smooth underlying manifold. Provided there is sufficient data (such that the manifold is well-sampled), we expect each data point and its neighbors to lie on or close to a locally linear patch of the manifold.

We can characterize the local geometry of these patches by linear coefficients that reconstruct each data point from its neighbors. In the simplest formulation of LLE, one identifies K nearest neighbors per data point, as measured by Euclidean distance. (Alternatively, one can identify neighbors by choosing all points within a ball of fixed radius, or by using more sophisticated rules based on local metrics.) Reconstruction errors are then measured by the cost function:

$$\mathcal{E}(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2 , \qquad (1)$$

which adds up the squared distances between all the data points and their reconstructions. The weights W_{ij} summarize the contribution of the jth data point to the ith reconstruction. To compute the weights W_{ij}, we minimize the cost function subject to two constraints: first, that each data point $\vec{X}_i$ is reconstructed only from its neighbors, enforcing $W_{ij} = 0$ if $\vec{X}_j$ does not belong to this set; second, that the rows of the weight matrix sum to one: $\sum_j W_{ij} = 1$. The reason for the sum-to-one constraint will become clear shortly. The optimal weights W_{ij} subject to these constraints are found by solving a least squares problem, as discussed in Appendix A.

Note that the constrained weights that minimize these reconstruction errors obey an important symmetry: for any particular data point, they are invariant to rotations, rescalings, and translations of that data point and its neighbors. The invariance to rotations and rescalings follows immediately from the form of eq. (1); the invariance to translations is enforced by the sum-to-one constraint on the rows of the weight matrix. A consequence of this symmetry is that the reconstruction weights characterize intrinsic geometric properties of each neighborhood, as opposed to properties that depend on a particular frame of reference.

Suppose the data lie on or near a smooth nonlinear manifold of dimensionality $d \ll D$. To a good approximation, then, there exists a linear mapping—consisting of a translation, rotation, and rescaling—that maps the high dimensional coordinates of each neighborhood to global internal coordinates on the manifold. By design, the reconstruction weights W_{ij} reflect intrinsic geometric properties of the

LLE ALGORITHM

1. Compute the neighbors of each data point, $\vec{X}_i$.
2. Compute the weights W_{ij} that best reconstruct each data point $\vec{X}_i$ from its neighbors, minimizing the cost in eq. (1) by constrained linear fits.
3. Compute the vectors $\vec{Y}_i$ best reconstructed by the weights W_{ij}, minimizing the quadratic form in eq. (2) by its bottom nonzero eigenvectors.

Figure 2: Summary of the LLE algorithm, mapping high dimensional data points, $\vec{X}_i$, to low dimensional embedding vectors, $\vec{Y}_i$.

are chosen, the optimal weights W_{ij} and coordinates Y_i are computed by standard methods in linear algebra. The algorithm involves a single pass through the three steps in Fig. 2 and finds global minima of the reconstruction and embedding costs in Eqs. (1) and (2). As discussed in Appendix A, in the unusual case where the neighbors outnumber the input dimensionality ($K > D$), the least squares problem for finding the weights does not have a unique solution, and a regularization term—for example, one that penalizes the squared magnitudes of the weights—must be added to the reconstruction cost.

The algorithm, as described in Fig. 2, takes as input the N high dimensional vectors, $\vec{X}_i$. In many settings, however, the user may not have access to data of this form, but only to measurements of dissimilarity or pairwise distance between different data points. A simple variation of LLE, described in Appendix C, can be applied to input of this form. In this way, matrices of pairwise distances can be analyzed by LLE just as easily as MDS[2]; in fact only a small fraction of all possible pairwise distances (representing distances between neighboring points and their respective neighbors) are required for running LLE.

3 Examples

The embeddings discovered by LLE are easiest to visualize for intrinsically two dimensional manifolds. In Fig. 1, for example, the input to LLE consisted $N = 600$ data points sampled off the S-shaped manifold. The resulting embedding shows how the algorithm, using $K = 12$ neighbors per data point, successfully unraveled the underlying two dimensional structure.

Fig. 3 shows another two dimensional manifold, this one living in a much higher dimensional space. Here, we generated examples—shown in the middle panel of the figure—by translating the image of a single face across a larger background of random noise. The noise was uncorrelated from one example to the next. The only consistent structure in the resulting images thus described a two-dimensional manifold parameterized by the face's center of mass. The input to LLE consisted of $N = 961$ grayscale images, with each image containing a 28×20 face superimposed on a 59×51 background of noise. Note that while simple to visualize, the manifold of translated faces is highly nonlinear in the high dimensional ($D = 3009$) vector space of pixel coordinates. The bottom portion of Fig. 3 shows the first two components discovered by LLE, with $K = 4$ neighbors per data point. By contrast, the top portion shows the first two components discovered by PCA. It is clear that the manifold structure in this example is much better modeled by LLE.

Finally, in addition to these examples, for which the true manifold structure was known, we also applied LLE to images of lips used in the animation of talking heads[3]. Our database contained $N = 8188$ color (RGB) images of lips at 109×84 resolution. Dimensionality reduction of these images ($D = 27216$) is useful for faster and more efficient animation. The top and bottom panels of Fig. 4 show the first two components discovered, respectively, by PCA and LLE (with $K = 16$). If the lip images described a nearly linear manifold, these two methods would yield similar results; thus, the significant differences in these embeddings reveal the presence of nonlinear structure. Note that while the linear projection by PCA has a somewhat uniform distribution about its mean, the locally linear embedding has a distinctly spiny structure, with the tips of the spines corresponding to extremal configurations of the lips.

4 Discussion

It is worth noting that many popular learning algorithms for nonlinear dimensionality reduction do not share the favorable properties of LLE. Iterative hill-climbing methods for autoencoder neural networks[7, 8], self-organizing maps[9], and latent variable models[10] do not have the same guarantees of global optimality or convergence; they also tend to involve many more free parameters, such as learning rates, convergence criteria, and architectural specifications.

The different steps of LLE have the following complexities. In Step 1, computing nearest neighbors scales (in the worst case) as $O(DN^2)$, or linearly in the input dimensionality, D, and quadratically in the number of data points, N. For many

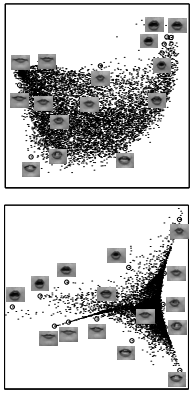


Figure 4: Images of lips mapped into the embedding space described by the first two coordinates of PCA (top) and LLE (bottom). Representative lips are shown next to circled points in different parts of each space. The differences between the two embeddings indicate the presence of nonlinear structure in the data.

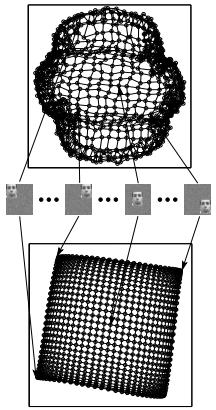


Figure 3: The results of PCA (top) and LLE (bottom), applied to images of a single face translated across a two-dimensional background of noise. Note how LLE maps the images with corner faces to the corners of its two dimensional embedding, while PCA fails to preserve the neighborhood structure of nearby images.

data distributions, however – and especially for data distributed on a thin submanifold of the observation space – constructions such as K-D trees can be used to compute the neighbors in $O(N \log N)$ time[13]. In Step 2, computing the reconstruction weights scales as $O(DNK^3)$; this is the number of operations required to solve a $K \times K$ set of linear equations for each data point. In Step 3, computing the bottom eigenvectors scales as $O(dN^2)$, linearly in the number of embedding dimensions, d, and quadratically in the number of data points, N. Methods for sparse eigenproblems[14], however, can be used to reduce the complexity to subquadratic in N. Note that as more dimensions are added to the embedding space, the existing ones do not change, so that LLE does not have to be rerun to compute higher dimensional embeddings. The storage requirements of LLE are limited by the weight matrix which is size N by K.

LLE illustrates a general principle of manifold learning, elucidated by Tenenbaum et al[11], that overlapping local neighborhoods—collectively analyzed—can provide information about global geometry. Many virtues of LLE are shared by the Isomap algorithm[11], which has been successfully applied to similar problems in nonlinear dimensionality reduction. Isomap is an extension of MDS in which embeddings are optimized to preserve "geodesic" distances between pairs of data points; these distances are estimated by computing shortest paths through large sublattices of data. A virtue of LLE is that it avoids the need to solve large dynamic programming problems. LLE also tends to accumulate very sparse matrices, whose structure can be exploited for savings in time and space.

LLE is likely to be even more useful in combination with other methods in data analysis and statistical learning. An interesting and important question is how to learn a parametric mapping between the observation and embedding spaces, given the results of LLE. One possible approach is to use $\{\vec{X}_i, \vec{Y}_i\}$ pairs as labeled examples for statistical models of supervised learning. The ability to learn such mappings should make LLE broadly useful in many areas of information processing.

A Constrained Least Squares Problem

The constrained weights that best reconstruct each data point from its neighbors can be computed in closed form. Consider a particular data point $\vec{x}$ with K nearest neighbors $\vec{\eta}_j$ and reconstruction weights w_j that sum to one. We can write the reconstruction error as:

$$\varepsilon = \left|\vec{x} - \textstyle\sum_j w_j \vec{\eta}_j\right|^2 = \left|\sum_j w_j (\vec{x} - \vec{\eta}_j)\right|^2 = \textstyle\sum_{jk} w_j w_k C_{jk}, \qquad (3)$$

where in the first identity, we have exploited the fact that the weights sum to one, and in the second identity, we have introduced the local covariance matrix,

$$C_{jk} = (\vec{x} - \vec{\eta}_j) \cdot (\vec{x} - \vec{\eta}_k). \tag{4}$$

This error can be minimized in closed form, using a Lagrange multiplier to enforce the constraint that $\sum_j w_j = 1$. In terms of the inverse local covariance matrix, the optimal weights are given by:

$$w_j = \frac{\sum_k C_{jk}^{-1}}{\sum_{lm} C_{lm}^{-1}}. \tag{5}$$

The solution, as written in eq. (5), appears to require an explicit inversion of the local covariance matrix. In practice, a more efficient way to minimize the error is simply to solve the linear system of equations, $\sum_j C_{jk} w_k = 1$, and then to rescale the weights so that they sum to one (which yields the same result). By construction, the local covariance matrix in eq. (4) is symmetric and semipositive definite. If the covariance matrix is singular or nearly singular—as arises, for example, when there are more neighbors than input dimensions ($K > D$), or when the data points are not in general position—it can be conditioned (before solving the system) by adding a small multiple of the identity matrix,

$$C_{jk} \leftarrow C_{jk} + \left(\frac{\Delta^2}{K}\right) \delta_{jk}, \tag{6}$$

where Δ^2 is small compared to the trace of C. This amounts to penalizing large weights that exploit correlations beyond some level of precision in the data sampling process.

B Eigenvector Problem

The embedding vectors $\vec{Y}_i$ are found by minimizing the cost function, eq. (2), for fixed weights W_{ij}:

$$\min_Y \quad \Phi(Y) \quad = \quad \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2. \tag{7}$$

Note that the cost defines a quadratic form,

$$\Phi(Y) = \sum_{ij} M_{ij} (\vec{Y}_i \cdot \vec{Y}_j),$$

involving inner products of the embedding vectors and the $N \times N$ matrix M:

$$M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj}, \tag{8}$$

where δ_{ij} is 1 if $i = j$ and 0 otherwise.

This optimization is performed subject to constraints that make the problem well posed. It is clear that the coordinates $\vec{Y}_i$ can be translated by a constant displacement without affecting the cost, $\Phi(Y)$. We remove this degree of freedom by requiring the coordinates to be centered on the origin:

$$\sum_i \vec{Y}_i = \vec{0}. \tag{9}$$

Also, to avoid degenerate solutions, we constrain the embedding vectors to have unit covariance, with outer products that satisfy

$$\frac{1}{N} \sum_i \vec{Y}_i \vec{Y}_i^{\top} = I, \tag{10}$$

where I is the $d \times d$ identity matrix. Note that there is no loss in generality in constraining the covariance of $\vec{Y}$ to be diagonal and of order unity, since the cost function in eq. (2) is invariant to rotations and homogeneous rescalings. The further constraint that the covariance is equal to the identity matrix expresses an assumption that reconstruction errors for different coordinates in the embedding space should be measured on the same scale.

The optimal embedding—up to a global rotation of the embedding space—is found by computing the bottom $d + 1$ eigenvectors of the matrix, M; this is a version of the Rayleitz-Ritz theorem[12]. The bottom eigenvector of this matrix, which we discard, is the unit vector with all equal components; it represents a free translation mode of eigenvalue zero. Discarding this eigenvector enforces the constraint that the embeddings have zero mean, since the components of other eigenvectors must sum to zero, by virtue of orthogonality. The remaining d eigenvectors form the d embedding coordinates found by LLE.

Note that the bottom $d + 1$ eigenvectors of the matrix M (that is, those corresponding to its smallest $d + 1$ eigenvalues) can be found without performing a full matrix diagonalization[14]. Moreover, the matrix M can be stored and manipulated as the sparse symmetric matrix

$$M = (I - W)^T (I - W), \tag{11}$$

giving substantial computational savings for large values of N. In particular, left multiplication by M (the subroutine required by most sparse eigensolvers) can be performed as

$$Mv = (v - Wv) - W^{\top}(v - Wv), \tag{12}$$

requiring just one multiplication by W and one multiplication by $W^{\top}$, both of which are extremely sparse. Thus, the matrix M never needs to be explicitly created or stored; it is sufficient to store and multiply the matrix W.

C LLE from Pairwise Distances

LLE can be applied to user input in the form of pairwise distances. In this case, nearest neighbors are identified by the smallest non-zero elements of each row in the distance matrix. To derive the reconstruction weights for each data point, we need to compute the local covariance matrix C_{jk} between its nearest neighbors, as defined by eq. (4) in appendix A. This can be done by exploiting the usual relation between pairwise distances and dot products that forms the basis of metric MDS[2]. Thus, for a particular data point, we set:

$$C_{jk} = \frac{1}{2} \left(D_j + D_k - D_{jk} - D_0 \right), \tag{13}$$

where D_{jk} denotes the squared distance between the jth and kth neighbors, $D_l = \sum_s D_{ls}$ and $D_0 = \sum_{jk} D_{jk}$. In terms of this local covariance matrix, the reconstruction weights for each data point are given by eq. (5). The rest of the algorithm proceeds as usual.

Note that this variant of LLE requires significantly less user input than the complete matrix of pairwise distances. Instead, for each data point, the user needs only to specify its nearest neighbors and the submatrix of pairwise distances between those neighbors. Is it possible to recover manifold structure from even less user input—say, just the pairwise distances between each data point and its nearest neighbors? A simple counterexample shows that this is not possible. Consider the square lattice of three dimensional data points whose integer coordinates sum to zero. Imagine that points with even z-coordinates are colored black, and that points with odd z-coordinates are colored red. The "two point" embedding that maps all black points to the origin and all red points to one unit away preserves the distance between each point and its four nearest neighbors. Nevertheless, this embedding completely fails to preserve the underlying structure of the original manifold.

Acknowledgements

The authors thank E. Cosatto, H.P. Graf, and Y. LeCun (AT&T Labs) and B. Frey (U. Toronto) for providing data for these experiments. S. Roweis acknowledges the support of the Gatsby Charitable Foundation, the National Science Foundation, and the National Sciences and Engineering Research Council of Canada.

References

[1] I.T. Jolliffe, *Principal Component Analysis* (Springer-Verlag, New York, 1989).

[2] T. Cox and M. Cox. *Multidimensional Scaling* (Chapman & Hall, London, 1994).

[3] E. Cosatto and H.P. Graf. Sample-Based Synthesis of Photo-Realistic Talking-Heads. *Proceedings of Computer Animation*, 103–110. IEEE Computer Society (1998).

[4] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science* **290**, 2323-2326 (2000).

[5] K. Fukunaga and D. R. Olsen, An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers* 20(2), 176-193 (1971).

[6] N. Kambhatla and T. K. Leen. Dimension reduction by local principal component analysis. *Neural Computation* **9**, 1493–1516 (1997).

[7] D. DeMers and G.W. Cottrell. Nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 5*, D. Hanson, J. Cowan, L. Giles, Eds. (Morgan Kaufmann, San Mateo, CA, 1993), pp. 580–587.

[8] M. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* **37**, 233–243 (1991).

[9] T. Kohonen. *Self-organization and Associative Memory* (Springer-Verlag, Berlin, 1988).

[10] C. Bishop, M. Svensen, and C. Williams. GTM: The generative topographic mapping. *Neural Computation* **10**, 215–234 (1998).

[11] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science* **290**, 2319-2323 (2000).

[12] R. A. Horn and C. R. Johnson. *Matrix Analysis* (Cambridge University Press, Cambridge, 1990).

[13] J. H. Friedman, J. L. Bentley and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3), 290-226 (1977).

[14] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide* (Society for Industrial and Applied Mathematics, Philadelphia, 2000).

Model Evaluation and Selection

Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap
- Comparing classifiers:
  - Confidence intervals
  - Cost-benefit analysis and ROC Curves

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

A confusion matrix is **a technique for summarizing the performance of a classification algorithm**.

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

There are two possible predicted classes: "yes" and "no". If we were predicting the presence of a disease

•**True positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
•**True negatives (TN):** We predicted no, and they don't have the disease.
•**False positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
•**False negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $CM_{i,j}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All
- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

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

- Perfect score is 1.0
- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*)
cancer = no	**140**	**9560**	9700	98.56 (*specificity*)
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Suppose I have 10,000 emails in my mailbox out of which 300 are spams. The spam detection system detects 150 mails as spams, out of which 50 are actually spams. What is the precision and recall of my spam detection system ?

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 boostrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

Estimating Confidence Intervals: Classifier Models M_1 vs. M_2

- Suppose we have 2 classifiers, M_1 and M_2, which one is better?
- Use 10-fold cross-validation to obtain $\overline{err}(M_1)$ and $\overline{err}(M_2)$
- These mean error rates are just *estimates* of error on the true population of *future* data cases
- What if the difference between the 2 error rates is just attributed to *chance*?
  - Use a **test of statistical significance**
  - Obtain **confidence limits** for our error estimates

Estimating Confidence Intervals: Null Hypothesis

- Perform 10-fold cross-validation
- Assume samples follow a **t distribution** with *k–1* **degrees of freedom** (here, *k=10*)
- Use **t-test** (or **Student's t-test**)
- **Null Hypothesis**: M_1 & M_2 are the same
- If we can **reject** null hypothesis, then
  - we conclude that the difference between M_1 & M_2 is **statistically significant**
  - Chose model with lower error rate

Estimating Confidence Intervals: t-test

- If only 1 test set available: **pairwise comparison**
  - For i[th] round of 10-fold cross-validation, the same cross partitioning is used to obtain $err(M_1)_i$ and $err(M_2)_i$
  - Average over 10 rounds to get $\overline{err}(M_1)$ and $\overline{err}(M_2)$
  - **t-test** computes **t-statistic** with *k-1* **degrees of freedom:**

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/k}} \quad \text{where}$$

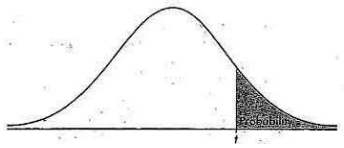
$$var(M_1 - M_2) = \frac{1}{k}\sum_{i=1}^{k}\left[err(M_1)_i - err(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))\right]^2$$

- If two test sets available: use **non-paired t-test**

where $var(M_1 - M_2) = \sqrt{\frac{var(M_1)}{k_1} + \frac{var(M_2)}{k_2}}$,

where k_1 & k_2 are # of cross-validation samples used for M_1 & M_2, resp.

Estimating Confidence Intervals: Table for t-distribution



- Symmetric
- **Significance level**, e.g., *sig = 0.05* or *5%* means M_1 & M_2 are *significantly different* for 95% of population
- **Confidence limit**, *z = sig/2*

TABLE B: *t*-DISTRIBUTION CRITICAL VALUES

	Tail probability *p*											
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level *C*											

Estimating Confidence Intervals: Statistical Significance

- Are M_1 & M_2 **significantly different**?
  - Compute *t.* Select *significance level* (e.g. *sig = 5%)*
  - Consult table for t-distribution: Find *t value* corresponding to *k-1 degrees of freedom* (here, 9)
  - t-distribution is symmetric: typically upper % points of distribution shown → look up value for **confidence limit** *z=sig/2* (here, 0.025)
  - **If t > z or t < -z**, then t value lies in rejection region:
    - **Reject null hypothesis** that mean error rates of M_1 & M_2 are same
    - Conclude: statistically significant difference between M_1 & M_2
  - **Otherwise**, conclude that any difference is **chance**

Issues Affecting Model Selection

- **Accuracy**
  - classifier accuracy: predicting class label
- **Speed**
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- **Robustness**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Predictor Error Measures

- Measure predictor accuracy: measure how far off the predicted value is from the actual known value
- **Loss function**: measures the error betw. y_i and the predicted value y_i'
  - Absolute error: $| y_i - y_i' |$
  - Squared error: $(y_i - y_i')^2$
- Test error (generalization error): the average loss over the test set
  - Mean absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{d}$ Mean squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{d}$
  - Relative absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{\sum_{i=1}^{d} | y_i - \bar{y} |}$ Relative squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{\sum_{i=1}^{d} (y_i - \bar{y})^2}$

The mean squared-error exaggerates the presence of outliers

Popularly use (square) root mean-square error, similarly, root relative squared error

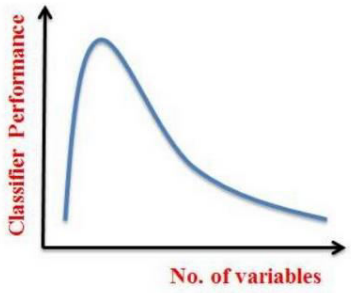
Feature Selection

Feature Selection

- It is the process of selecting a subset of relevant features for use in model construction.
- It is also called variable selection or attribute selection.
- It reduces the complexity of model.
- It either improve or maintain accuracy of model.

Curse of Dimensionality

- The amount of data required to achieve the same level of accuracy increases exponentially as the number of features increases.
- But in practice, the volume of training data available to us is fixed. Therefore, in most cases, the performance of the classifier will decrease with an increased number of variables.

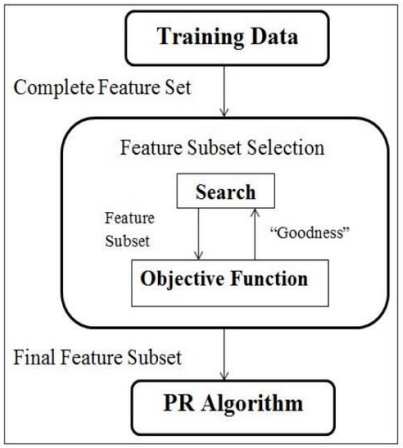


Feature Selection Steps

Feature Selection is an optimization problem.

Step 1 : Search the space of possible feature set.

Step 2 : Pick the subset that is optimal or near optimal with respect to some objective function.



Feature Selection Methods

- Feature Selection Algorithms can be broadly classified into 3 classes :
  - Filter Methods
  - Wrapper Methods
  - Embedded Methods

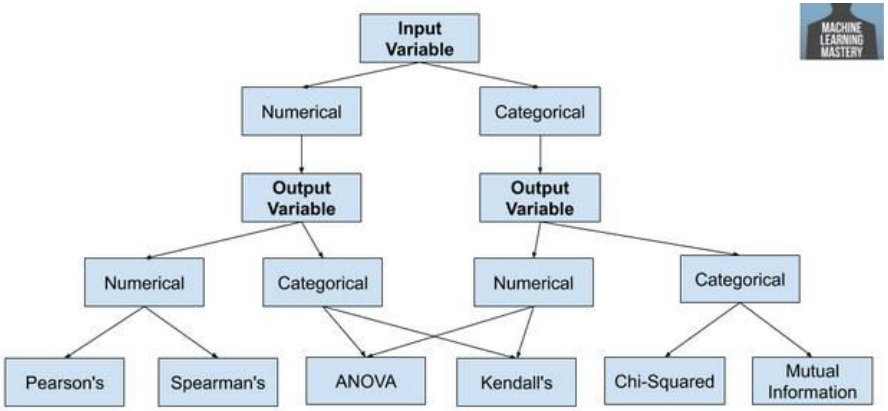
Feature\Response	Continuous	Categorical
Continuous	Pearson's Correlation	LDA
Categorical	Anova	Chi-Square

continuous variables X and Y. Its value varies from -1 to +1. Pearson's correlation is given as:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

- **LDA:** Linear discriminant analysis is used to find a linear combination of features that characterizes or separates two or more classes (or levels) of a categorical variable.
- **ANOVA:** ANOVA stands for Analysis of variance. It is similar to LDA except for the fact that it is operated using one or more categorical independent features and one continuous dependent feature. It provides a statistical test of whether the means of several groups are equal or not.
- **Chi-Square:** It is a is a statistical test applied to the groups of categorical features to evaluate the likelihood of correlation or association between them using their frequency distribution.

One thing that should be kept in mind is that filter methods do not remove multicollinearity. So, you must deal with multicollinearity of features as well before training models for your data.



Typical methods for feature selection

- Categories

	Single feature evaluation	Subset selection
filter	MI, IG, KL-D, GI, CHI	Category distance, …
wrapper	Ranking accuracy using single feature	For LR (SFO, Grafting)

- Single feature evaluation
  - Frequency based, mutual information, KL divergence, Gini indexing, information gain, Chi square statistic
- Subset selection method
  - Sequential forward selection
  - Sequential backward selection

Single feature evaluation

- Measure quality of features by all kinds of metrics
  - Frequency based
  - Dependence of feature and label (Co-occurrence)
    - mutual information, Chi square statistic
  - Information theory
    - KL divergence, information gain
  - Gini indexing

Frequency based

- Remove features according to frequency of features or instances contain the feature
- Typical scenario
  - Text mining

Mutual information

- Measure the dependence of two random variables
- Definition

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right)$$

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) dx\,dy$$

Chi Square Statistic

- Measure the dependence of two variables

$$\chi^2(t,c) = \frac{N \times (AD - CB)^2}{(A+C) \times (B+D) \times (A+B) \times (C+D)}$$

  - A: number of times feature t and category c co-occur
  - B: number of times t occurs without c
  - C: number of times c occurs without t
  - D: number of times neither c or t occurs
  - N: total number of instances

Information Gain

- Reduction in entropy caused by partitioning the examples according to the attribute

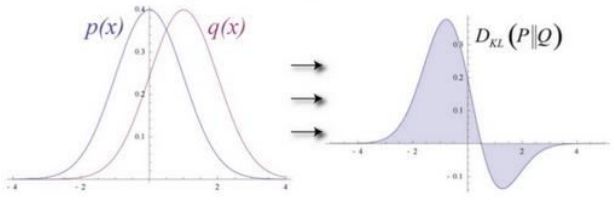
$$Gain(S,f) = Entroy(S) - (\Pr(f)Entropy(S \mid f) + \overline{\Pr}(f)Entropy(S \mid \bar{f}))$$

$$IG(t) = -\sum_i Pr(c_i)\log Pr(c_i) + Pr(t)\sum_i Pr(c_i|t)\log Pr(c_i|t) + Pr(\bar{t})\sum_i Pr(c_i|\bar{t})\log Pr(c_i|\bar{t})$$

KL divergence

- Measure the difference between two probability distribution

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\ln\frac{P(i)}{Q(i)}$$



$$D_{\mathrm{KL}}(P\|Q) = -\sum_x p(x)\log q(x) + \sum_x p(x)\log p(x) = H(P,Q) - H(P)$$

Gini indexing

- Calculate conditional probability of f given class label

$$Pr(f|c_i) = \frac{1+\mathcal{N}(f,c_i)}{|V| + \sum_{f\in V}\mathcal{N}(f,c_i)}$$

- Normalize across all classes

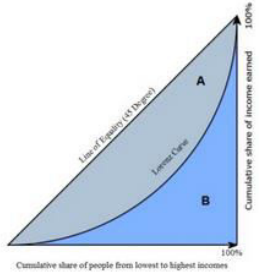
$$wcp(f,c_i) = \frac{Pr(f|c_i)}{\sum_{k=1}^{|C|} Pr(f|c_k)}$$

- Calculate Gini coefficient

$$\mathcal{G}(t) = \frac{\sum_{i=1}^{n}(2i-n-1)wcp(t,c_i)}{n^2\mu}$$

- For two categories case

$$\varsigma(t) = \frac{wcp(t,c_2) - wcp(t,c_1)}{2(wcp(t,c_2) + wcp(t,c_1))}$$



Subset selection methods

- Select subsets of features that together have good predictive power, as opposed to ranking features individually
- Always by adding new features into existing set or removing features out of existing set
  - Sequential forward selection
  - Sequential backward selection
- Evaluation
  - category distance measurement
  - Classification error

Category distance measurement

- Select features subset with large category distance

$$J(x) = \frac{1}{2}\sum_{i=1}^{C} P_i \sum_{j=1}^{C} P_j \frac{1}{n_i n_j} \sum_{k=1}^{n_i}\sum_{l=1}^{n_j} \ell(x_k^{(i)}, x_l^{(j)})$$

$$\ell(x_k^{(i)}, x_l^{(j)}) = (x_k^{(i)} - x_l^{(j)})^T (x_k^{(i)} - x_l^{(j)})$$

Wrapper Method

- work on the basic principles of Combinatorics.
- the learning algorithm (classifier) itself is used to perform the Feature Selection and select the top features.
- Examples :
  - forward feature selection
  - backward feature elimination
  - recursive feature elimination

- **Forward Selection:** Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.
- **Backward Elimination:** In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features.
- **Recursive Feature elimination:** It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

Embedded Methods

- combine the qualities of filter and wrapper methods.
- dynamic Feature Selection Methods.
- Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset. The problem is essentially reduced to a search problem.

5. Difference between Filter and Wrapper methods

The main differences between the filter and wrapper methods for feature selection are:

- Filter methods measure the relevance of features by their correlation with dependent variable while wrapper methods measure the usefulness of a subset of feature by actually training a model on it.
- Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally very expensive as well.
- Filter methods use statistical methods for evaluation of a subset of features while wrapper methods use cross validation.
- Filter methods might fail to find the best subset of features in many occasions but wrapper methods can always provide the best subset of features.
- Using the subset of features from the wrapper methods make the model more prone to overfitting as compared to using subset of features from the filter methods.

Summarization

- Categories

	Single feature evaluation	Subset selection
filter	MI, IG, KL-D, GI, CHI	Category distance, ...
wrapper	Ranking accuracy using single feature	For LR (SFO, Grafting)

- Filter + Single feature evaluation
  - Less time consuming, usually works well
- Wrapper + Subset selection
  - Higher accuracy, but easy overfitting

Outlook	emperatur	Humidity	Windy	Play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

Relief

RELIEF [KR92] is a feature weighting algorithm that is sensitive to feature interactions

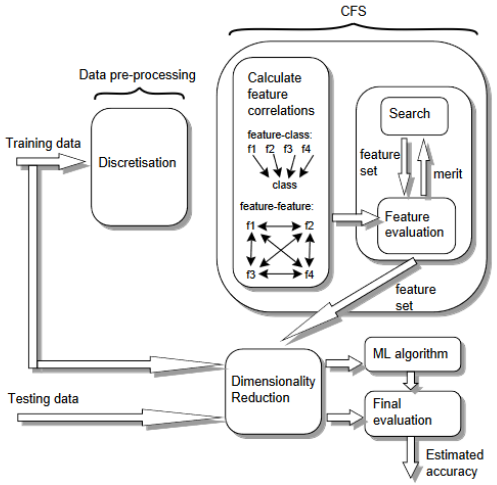
$$W_X = P(\text{different value of } X \mid \text{nearest instance of different class}) - P(\text{different value of } X \mid \text{nearest instance of same class}).$$

$$Relief_X = P(\text{different value of } X \mid \text{different class}) - P(\text{different value of } X \mid \text{same class}),$$

$$Relief_X = \frac{Gini' \times \sum_{x \in X} p(x)^2}{(1 - \sum_{c \in C} p(c)^2) \sum_{c \in C} p(c)^2},$$

$$M_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}}$$

where M_S is the heuristic "merit" of a feature subset S containing k features, $\overline{r_{cf}}$ is the mean feature-class correlation ($f \in S$), and $\overline{r_{ff}}$ is the average feature-feature inter-correlation. The numerator of Equation 4.16 can be thought of as providing an indication of how predictive of the class a set of features are; the denominator of how much redundancy there is among the features.



Outlook	emperatur	Humidity	Windy	Play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

Table shows the feature correlation matrix for the data set—relief has been used to calculate the correlatio

	Outlook	Temperature	Humidity	Wind	Class
Outlook	1.000	0.116	0.022	0.007	0.130
Temperature		1.000	0.248	0.028	0.025
Humidity			1.000	0.000	0.185
Wind				1.000	0.081

Table illustrates a forward selection search through the feature subset space along with the merit of each subset

Feature set	k	$\overline{r_{cf}}$	$\overline{r_{ff}}$	Merit
[]	0	N/A	N/A	0.0
[Outlook]	1	0.130	1.000	$\frac{1\times0.130}{\sqrt{1+1(1-1)1.0}} = 0.130$
[Temperature]	1	0.025	1.000	$\frac{1\times0.025}{\sqrt{1+1(1-1)1.0}} = 0.025$
[**Humidity**]	1	0.185	1.000	$\frac{1\times0.185}{\sqrt{1+1(1-1)1.0}} = 0.185$
[Wind]	1	0.081	1.000	$\frac{1\times0.081}{\sqrt{1+1(1-1)1.0}} = 0.081$
[**Outlook Humidity**]	2	0.158	0.022	$\frac{2\times0.158}{\sqrt{2+2(2-1)0.022}} = 0.220$
[Temperature Humidity]	2	0.105	0.258	$\frac{2\times0.105}{\sqrt{2+2(2-1)0.258}} = 0.133$
[Humidity Wind]	2	0.133	0.0	$\frac{2\times0.133}{\sqrt{2+2(2-1)0.0}} = 0.188$
[Outlook Temperature Humidity]	3	0.113	0.132	$\frac{3\times0.113}{\sqrt{3+3(3-1)0.132}} = 0.175$
[**Outlook Humidity Wind**]	3	0.132	0.0096	$\frac{3\times0.132}{\sqrt{3+3(3-1)0.0096}} = 0.226$
[Outlook Temperature Humidity Wind]	4	0.105	0.0718	$\frac{4\times0.105}{\sqrt{4+4(4-1)0.0718}} = 0.191$

The search begins with the empty set of features, which has zero merit. Each single feature addition to the empty set is evaluated; Humidity is added to the subset because it has the highest score. The next step involves trying each of the remaining features with Humidity and choosing the best (Outlook). Similarly, in the next stage Wind is added to the subset. The last step tries the single remaining feature (Temperature) with the current subset; this does not improve its merit and the search terminates. The best subset found (Outlook, Humidity, Wind) is returned.

Feature selection

What is feature selection?

- Consider our training data as a matrix where each row is a vector and each column is a dimension.
- For example consider the matrix for the data x_1=(1, 10, 2), x_2=(2, 8, 0), and x_3=(1, 9, 1)
- We call each dimension a feature or a column in our matrix.

Feature selection

- Useful for high dimensional data such as genomic DNA and text documents.
- Methods
  - Univariate (looks at each feature independently of others)
    - Pearson correlation coefficient
    - F-score
    - Chi-square
    - Signal to noise ratio
    - And more such as mutual information, relief
  - Multivariate (considers all features simultaneously)
    - Dimensionality reduction algorithms
    - Linear classifiers such as support vector machine
    - Recursive feature elimination

Feature selection

- Methods are used to rank features by importance
- Ranking cut-off is determined by user
- Univariate methods measure some type of correlation between two random variables. We apply them to machine learning by setting one variable to be the label (y_i) and the other to be a fixed feature (x_{ij} for fixed j)

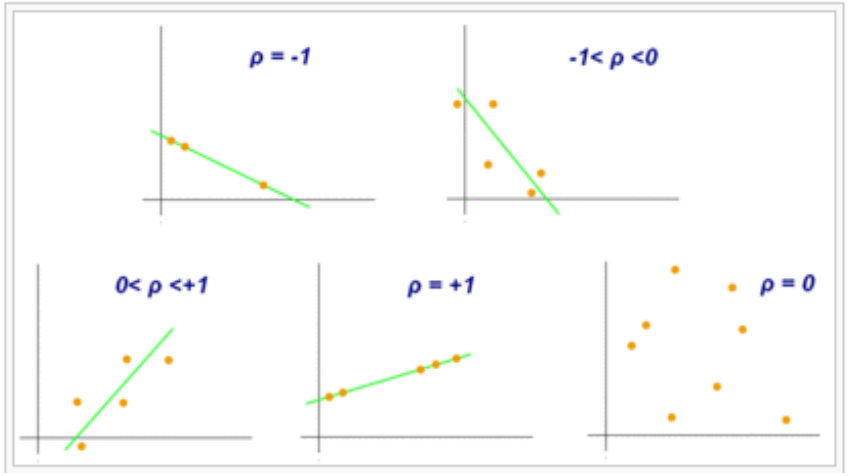
Pearson correlation coefficient

- Measures the correlation between two variables
- Formulas:
  - Covariance(X,Y) = $E((X-\mu_X)(Y-\mu_Y))$
  - Correlation(X,Y)= Covariance(X,Y)/$\sigma_X\sigma_Y$
  - Pearson correlation =

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

- The correlation r is between -1 and 1. A value of 1 means perfect positive correlation and -1 in the other direction

Pearson correlation coefficient



From Wikipedia

F-score

F-score is a simple technique which measures the discrimination of two sets of real numbers. Given training vectors $\boldsymbol{x}_k, k = 1, \ldots, m$, if the number of positive and negative instances are n_+ and n_-, respectively, then the F-score of the ith feature is defined as:

$$F(i) \equiv \frac{\left(\bar{\boldsymbol{x}}_i^{(+)} - \bar{\boldsymbol{x}}_i\right)^2 + \left(\bar{\boldsymbol{x}}_i^{(-)} - \bar{\boldsymbol{x}}_i\right)^2}{\frac{1}{n_+ - 1}\sum_{k=1}^{n_+}\left(x_{k,i}^{(+)} - \bar{\boldsymbol{x}}_i^{(+)}\right)^2 + \frac{1}{n_- - 1}\sum_{k=1}^{n_-}\left(x_{k,i}^{(-)} - \bar{\boldsymbol{x}}_i^{(-)}\right)^2}, \qquad (4)$$

where $\bar{\boldsymbol{x}}_i$, $\bar{\boldsymbol{x}}_i^{(+)}$, $\bar{\boldsymbol{x}}_i^{(-)}$ are the average of the ith feature of the whole, positive, and negative data sets, respectively; $x_{k,i}^{(+)}$ is the ith feature of the kth positive instance, and $x_{k,i}^{(-)}$ is the ith feature of the kth negative instance. The numerator indicates the discrimination between the positive and negative sets, and the denominator indicates the one within each of the two sets. The larger the F-score is, the more likely this feature is more discriminative. Therefore, we use this score as a feature selection criterion.

From Lin and Chen, Feature extraction, 2006

Chi-square test

- We have two random variables:
  - Label (L): 0 or 1
  - Feature (F): Categorical
- Null hypothesis: the two variables are independent of each other (unrelated)
- Under independence
  - P(L,F)= P(D)P(G)
  - P(L=0) = (c1+c2)/n
  - P(F=A) = (c1+c3)/n
- Expected values
  - E(X1) = P(L=0)P(F=A)n
- We can calculate the chi-square statistic for a given feature and the probability that it is independent of the label (using the p-value).
- Features with very small probabilities deviate significantly from the independence assumption and therefore considered important.

Contingency table

	Feature=A	Feature=B
Label=0	Observed=c1 Expected=X1	Observed=c2 Expected=X2
Label=1	Observed=c3 Expected=X3	Observed=c4 Expected=X4

Signal to noise ratio

- Difference in means divided by difference in standard deviation between the two classes
- $S2N(X,Y) = (\mu_X - \mu_Y)/(\sigma_X - \sigma_Y)$
- Large values indicate a strong correlation

Multivariate feature selection

- Consider the vector w for any linear classifier.
- Classification of a point x is given by wTx+w0.
- Small entries of w will have little effect on the dot product and therefore those features are less relevant.
- For example if w = (10, .01, -9) then features 0 and 2 are contributing more to the dot product than feature 1. A ranking of features given by this w is 0, 2, 1.

Multivariate feature selection

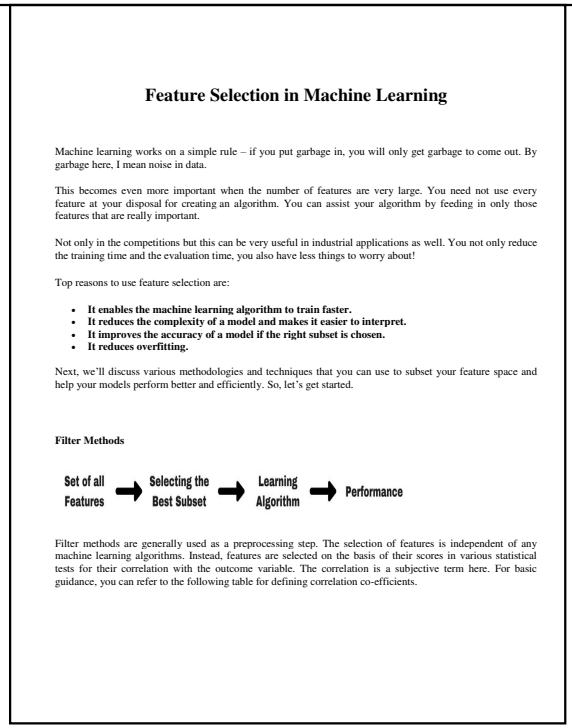
- The w can be obtained by any of linear classifiers we have see in class so far
- A variant of this approach is called recursive feature elimination:
  - Compute w on all features
  - Remove feature with smallest w_i
  - Recompute w on reduced data
  - If stopping criterion not met then go to step 2

Limitations

- Unclear how to tell in advance if feature selection will work
  - Only known way is to check but for very high dimensional data (at least half a million features) it helps most of the time
- How many features to select?
  - Perform cross-validation

Feature selection in practice

- NIPS 2003 feature selection contest
  - Contest results
  - Reproduced results with feature selection plus SVM
- Effect of feature selection on SVM
- Comprehensive gene selection study comparing feature selection methods
- Ranking genomic causal variants with SVM and chi-square



Feature Selection in Machine Learning

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, I mean noise in data.

This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important.

Not only in the competitions but this can be very useful in industrial applications as well. You not only reduce the training time and the evaluation time, you also have less things to worry about!

Top reasons to use feature selection are:

- **It enables the machine learning algorithm to train faster.**
- **It reduces the complexity of a model and makes it easier to interpret.**
- **It improves the accuracy of a model if the right subset is chosen.**
- **It reduces overfitting.**

Next, we'll discuss various methodologies and techniques that you can use to subset your feature space and help your models perform better and efficiently. So, let's get started.

Filter Methods

Set of all Features → Selecting the Best Subset → Learning Algorithm → Performance

Filter methods are generally used as a preprocessing step. The selection of features is independent of any machine learning algorithms. Instead, features are selected on the basis of their scores in various statistical tests for their correlation with the outcome variable. The correlation is a subjective term here. For basic guidance, you can refer to the following table for defining correlation co-efficients.

Feature\Response	Continuous	Categorical
Continuous	Pearson's Correlation	LDA
Categorical	Anova	Chi-Square

Statistical tools:

- **Pearson's Correlation:** It is used as a measure for quantifying linear dependence between two continuous variables X and Y. Its value varies from -1 to +1. Pearson's correlation is given as:

  For eg :

X	Y
1	10
2	20
3	30

- Both X and Y are tightly correlated with each other as X increases Y also increases.

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

- where:

  cov is the covariance
  σ_X is the standard deviation of X
  σ_Y is the standard deviation of Y

The formula for ρ can be expressed in terms of mean and expectation. Since

$$\operatorname{cov}(X,Y) = \mathbb{E}[(X-\mu_X)(Y-\mu_Y)],$$ [11]

the formula for ρ can also be written as

$$\rho_{X,Y} = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \quad \text{(Eq.2)}$$

where:

σ_Y and σ_X are defined as above
μ_X is the mean of X
μ_Y is the mean of Y
$\mathbb{E}$ is the expectation.

- **LDA:** Linear discriminant analysis is used to find a linear combination of features that characterizes or separates two or more classes (or levels) of a categorical variable.

- **ANOVA:** ANOVA stands for Analysis of variance. It is similar to LDA except for the fact that it is operated using one or more categorical independent features and one continuous dependent feature. It provides a statistical test of whether the means of several groups are equal or not.

  Univariate feature selection. Univariate feature selection works **by selecting the best features based on univariate statistical tests**. We compare each feature to the target variable, to see whether there is any statistically significant relationship between them. It is also called analysis of variance (ANOVA).

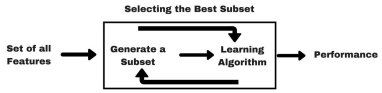
- **Chi-Square:** It is a statistical test applied to the groups of categorical features to evaluate the likelihood of correlation or association between them using their frequency distribution.

One thing that should be kept in mind is that filter methods do not remove multicollinearity. So, you must deal with multicollinearity of features as well before training models for your data.

Multicollinearity generally occurs when there are high correlations between two or more predictor variables.

Multicollinearity example : A **person's height and weight,**

Wrapper Methods



In wrapper methods, we try to use a subset of features and train a model using them. Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset. These methods are usually computationally very expensive.

Some common examples of wrapper methods are forward feature selection, backward feature elimination, recursive feature elimination, etc.

- **Forward Selection:** Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.

Algorithm 1 Sequential forward feature set generation - SFG.

function SFG(F - full set, U - measure)
  initialize: $S = \{\}$ ▷ S stores the selected features
  repeat
    $f = \text{FINDNEXT}(F)$
    $S = S \cup \{f\}$
    $F = F - \{f\}$
  until S satisfies U *or* $F = \{\}$
  return S
end function

- **Backward Elimination:** In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features.

Algorithm 2 Sequential backward feature set generation - SBG.

function SBG(F - full set, U - measure)
  initialize: $S = \{\}$ ▷ S holds the removed features
  repeat
    $f = \text{GETNEXT}(F)$
    $F = F - \{f\}$
    $S = S \cup \{f\}$
  until S does not satisfy U *or* $F = \{\}$
  return $F \cup \{f\}$
end function

- **Recursive Feature elimination:** It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

Algorithm 3 Bidirectional feature set generation - BG.

function BG(F_f, F_b - full set, U - measure)
  initialize: $S_f = \{\}$ ▷ S_f holds the selected features
  initialize: $S_b = \{\}$ ▷ S_b holds the removed features
  repeat
    $f_f = \text{FINDNEXT}(F_f)$
    $f_b = \text{GETNEXT}(F_b)$
    $S_f = S_f \cup \{f_f\}$
    $F_b = F_b - \{f_b\}$
    $F_f = F_f - \{f_f\}$
    $S_b = S_b \cup \{f_b\}$
  until (a) S_f satisfies U *or* $F_f = \{\}$ *or* (b) S_b does not satisfy U *or* $F_b = \{\}$
  return S_f if (a) *or* $F_b \cup \{f_b\}$ if (b)
end function

One of the best ways for implementing feature selection with wrapper methods is to use Boruta package that finds the importance of a feature by creating shadow features.

Boruta is a feature selection algorithm, it works as a wrapper algorithm around Random Forest.

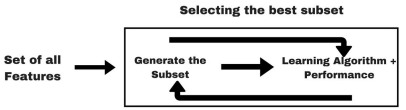
It works in the following steps:

1. Firstly, it adds randomness to the given data set by creating shuffled copies of all features (which are called shadow features).
2. Then, it trains a random forest classifier on the extended data set and applies a feature importance measure (the default is Mean Decrease Accuracy) to evaluate the importance of each feature where higher means more important.

3. At every iteration, it checks whether a real feature has a higher importance than the best of its shadow features (i.e. whether the feature has a higher Z-score than the maximum Z-score of its shadow features) and constantly removes features which are deemed highly unimportant.
4. Finally, the algorithm stops either when all features get confirmed or rejected or it reaches a specified limit of random forest runs.

Boruta diagram of running flow from creating shadows — training — comparing — removing features and back again.

Embedded Methods



Embedded methods combine the qualities'of filter and wrapper methods. It's implemented by algorithms that have their own built-in feature selection methods.

Some of the most popular examples of these methods are LASSO and RIDGE regression which have inbuilt penalization functions to reduce overfitting.

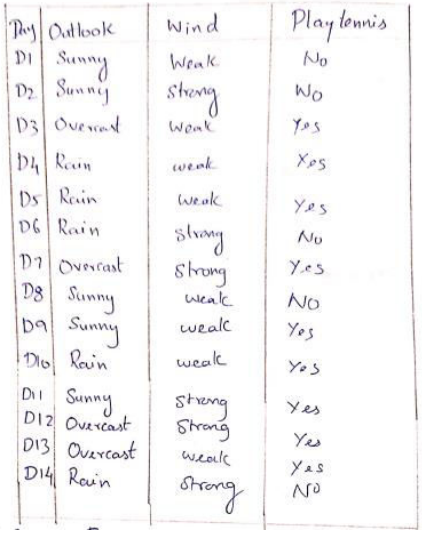
- Lasso regression performs L1 regularization which adds penalty equivalent to absolute value of the magnitude of coefficients.
- Ridge regression performs L2 regularization which adds penalty equivalent to square of the magnitude of coefficients.

Difference between Filter and Wrapper methods

The main differences between the filter and wrapper methods for feature selection are:

- Filter methods measure the relevance of features by their correlation with dependent variable while wrapper methods measure the usefulness of a subset of feature by actually training a model on it.
- Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally very expensive as well.
- Filter methods use statistical methods for evaluation of a subset of features while wrapper methods use cross validation.
- Filter methods might fail to find the best subset of features in many occasions but wrapper methods can always provide the best subset of features.
- Using the subset of features from the wrapper methods make the model more prone to overfitting as compared to using subset of features from the filter methods.

Chi-Square Test



Day	Outlook	Wind	Play tennis
D1	Sunny	Weak	No
D2	Sunny	Strong	No
D3	Overcast	Weak	Yes
D4	Rain	weak	Yes
D5	Rain	Weak	Yes
D6	Rain	Strong	No
D7	Overcast	Strong	Yes
D8	Sunny	weak	No
D9	Sunny	weak	Yes
D10	Rain	weak	Yes
D11	Sunny	Strong	Yes
D12	Overcast	Strong	Yes
D13	Overcast	weak	Yes
D14	Rain	Strong	No

Chi Square Formula

$$X^2 = \sum_{i=1}^{M} \sum_{j=1}^{k} \left(\frac{(O_{ij} - E_{ij})^2}{E_{ij}} \right)$$

Rows = M

Columns = k

O_{ij} = Observed Frequency

E_{ij} = Expected Frequency

Let us first find out the Chisquare value for outlook. ($X^2_{outlook}$)

⇒ Contruct the contigency table

Day	Outlook	Wind	Playtennis
D1	Sunny	Weak	No
D2	Sunny	Strong	No
D3	Overcast	Weak	Yes
D4	Rain	weak	Yes
D5	Rain	weak	Yes
D6	Rain	Strong	No
D7	Overcast	Strong	Yes
D8	Sunny	weak	No
D9	Sunny	weak	Yes
D10	Rain	weak	Yes
D11	Sunny	Strong	Yes
D12	Overcast	Strong	Yes
D13	Overcast	weak	Yes
D14	Rain	Strong	No

Observed Frequency Contigency Table

Outlook	Yes	No	
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5
	9	5	14

Step 2: Expected Value Calculation

$E(Sunny, Yes) = n \times P(Yes) \times P(Sunny)$

$= 14 \times \frac{9}{14} \times \frac{5}{14}$

$= 3.21$

$E(Sunny, No) = n \times P(No) \times P(Sunny)$

$= 14 \times \frac{5}{14} \times \frac{5}{14}$

Similarly calculate = 1.79 for other cases and tabulate.

Similarly calculate = 1.79 for other cases and tabulate.

	Yes	No
Sunny	3.21	1.79
Overcast	2.57	1.43
Rain	3.21	1.79

Observed Frequency Contigency Table

Outlook	Yes	No	
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5
	9	5	14

Similarly calculate = 1.79 for other cases and tabulate.

	Yes	No
Sunny	3.21	1.79
Overcast	2.57	1.43
Rain	3.21	1.79

Step 3: Similarly calculate for X^2_{wind}

Array ... ∧ formula

$$X^2 = \frac{(2-3.21)^2}{3.21} + \frac{(4-2.57)^2}{2.57} + \frac{(3-3.21)^2}{3.21} + \frac{(3-1.79)^2}{1.79} + \frac{(9-1.43)^2}{1.43} + \frac{(2-1.79)^2}{1.79}$$

$X^2_{outlook} = 3.129$

Similarly calculate for X^2_{wind}

X^2_{wind}

	Yes		No		Total
	O	E	O	E	
Strong	3	3.86	3	1.14	6
Weak	6	5.14	2	2.86	8
	9	9	5		14

$$X^2_{wind} = \frac{(3-3.86)^2}{3.86} + \frac{(3-1.14)^2}{1.14} + \frac{(6-5.14)^2}{5.14} + \frac{(2-2.86)^2}{2.86}$$

$$X^2_{wind} = 3.629$$

On comparing two scores, it is identified that the feature "Wind" is more important to determine the Output.

Day	Outlook	Wind	Play tennis
D1	Sunny	Weak	No
D2	Sunny	Strong	No
D3	Overcast	Weak	Yes
D4	Rain	weak	Yes
D5	Rain	Weak	Yes
D6	Rain	Strong	No
D7	Overcast	Strong	Yes
D8	Sunny	weak	No
D9	Sunny	weak	Yes
D10	Rain	weak	Yes
D11	Sunny	Strong	Yes
D12	Overcast	Strong	Yes
D13	Overcast	weak	Yes
D14	Rain	Strong	No

Chi Square Formula

$$X^2 = \sum_{i=1}^{M} \sum_{j=1}^{k} \left(\frac{(O_{ij} - E_{ij})^2}{E_{ij}} \right)$$

Rows = M

Columns = k

O_{ij} = Observed Frequency

E_{ij} = Expected Frequency

Let us first find out the chisquare value for outlook. ($X^2_{outlook}$)

⇒ Construct the contigency table

Observed Frequency Contigency Table

Outlook	Yes	No	
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5
	9	5	14

Step 2: Expected Value calculation

$E(Sunny, Yes) = n \times P(Yes) \times P(Sunny)$

$= 14 \times \frac{9}{14} \times \frac{5}{14}$

$= 3.21$

$E(Sunny, No) = n \times P(No) \times P(Sunny)$

$= 14 \times \frac{5}{14} \times \frac{5}{14}$

$= 1.79$

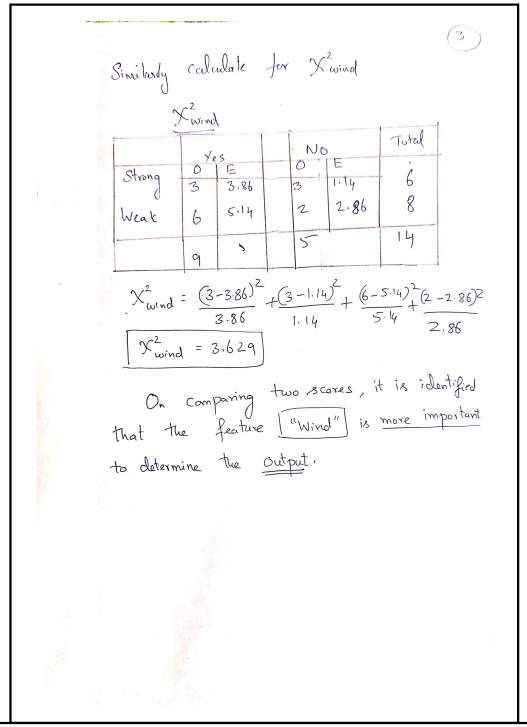
Similarly calculate for other cases and tabulate.

	Yes	No
Sunny	3.21	1.79
Overcast	2.57	1.43
Rain	3.21	1.79

Step 3: Apply in X^2 formula

$$X^2 = \frac{(2-3.21)^2}{3.21} + \frac{(4-2.57)^2}{2.57} + \frac{(3-3.21)^2}{3.21} + \frac{(3-1.79)^2}{1.79} + \frac{(0-1.43)^2}{1.43} + \frac{(2-1.79)^2}{1.79}$$

$X^2_{outlook} = 3.129$



Similarly calculate for χ^2_{wind}

χ^2_{wind}

	Yes		No		Total
	O	E	O	E	
Strong	3	3.86	3	1.14	6
Weak	6	5.14	2	2.86	8
	9		5		14

$$\chi^2_{wind} = \frac{(3-3.86)^2}{3.86} + \frac{(3-1.14)^2}{1.14} + \frac{(6-5.14)^2}{5.14} + \frac{(2-2.86)^2}{2.86}$$

$\chi^2_{wind} = 3.629$

On comparing two scores, it is identified that the feature "Wind" is more important to determine the output.

Learning from Imbalanced Data

1. The problem: Imbalanced Learning
2. The solutions: State-of-the-art
3. The evaluation: Assessment Metrics
4. The future: Opportunities and Challenges

The Nature of Imbalanced Learning Problem

The Nature of Imbalance Learning

The Problem

- ✓ Explosive availability of raw data
- ✓ Well-developed algorithms for data analysis

Requirement?

- ***Balanced*** *distribution* of data
- ***Equal*** *costs* of misclassification

What about data in reality?

Imbalance is Everywhere

- Between-class
- Within-class
- Intrinsic and extrinsic
- Relativity and rarity
- Imbalance and small sample size

Growing interest

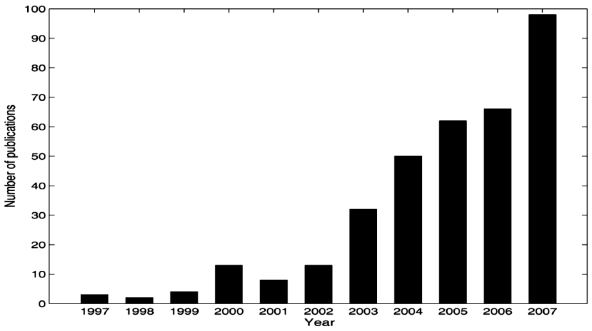


Fig. 1. Number of publications on imbalanced learning.

The Nature of Imbalance Learning

Mammography Data Set:
An example of ***between-class*** *imbalance*

	Negative/healthy	Positive/cancerous
Number of cases	10,923	260
Category	Majority	Minority
Imbalanced accuracy	≈ 100%	0-10 %

Imbalance can be on the order of 100 : 1 up to 10,000 : 1!

Intrinsic and ***extrinsic*** imbalance

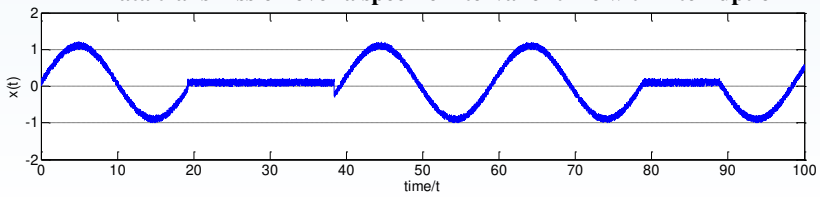
Intrinsic:

- Imbalance due to the nature of the dataspace

Extrinsic:

- Imbalance due to time, storage, and other factors
- **Example:**

Data transmission over a specific interval of time with interruption



The Nature of Imbalance Learning

Data Complexity

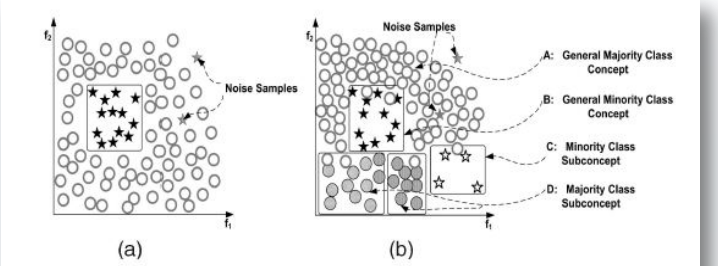


Fig. 2. (a) A data set with a between-class imbalance. (b) A high-complexity data set with both between-class and within-class imbalances, multiple concepts, overlapping, noise, and lack of representative data.

Relative imbalance and *absolute rarity*

$$Q: 1{,}000{,}000 : 1{,}000 = 1{,}000 : 1 \ ?$$

- The minority class may be outnumbered, but not necessarily rare
- Therefore they can be accurately learned with little disturbance

Imbalanced data with *small sample size*

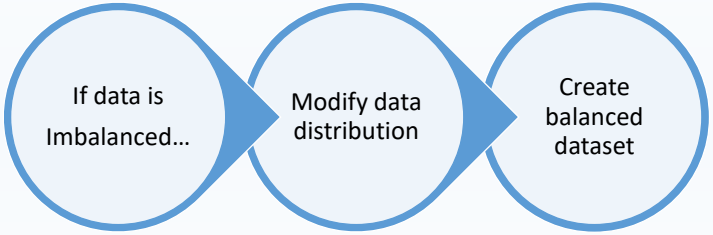
- Data with high dimensionality and small sample size
  - Face recognition, gene expression
- Challenges with small sample size:
  1. Embedded absolute rarity and within-class imbalances
  2. Failure of generalizing inductive rules by learning algorithms
     - Difficulty in forming good classification decision boundary over *more* features but *less* samples
     - Risk of overfitting

The Solutions to Imbalanced Learning Problem

Solutions to imbalanced learning

- Sampling methods
- Cost-sensitive methods
- Kernel and Active Learning methods

Sampling methods



Create balance though sampling

Sampling methods

Random Sampling

S: training data set; S_{min}: set of minority class samples, S_{maj}: set of majority class samples; ***E***: generated samples

Random oversampling

- Expand the minority
- $|S'_{min}| \leftarrow |S_{min}| + |E|$
- $|S'| \leftarrow |S_{min}| + |S_{maj}| + |E|$
- Overfitting due to multiple "tied" instances

Random undersampling

- Shrink the majority
- $|S'_{maj}| \leftarrow |S_{maj}| - |E|$
- $|S'| \leftarrow |S_{min}| + |S_{maj}| - |E|$
- Loss of important concepts

Sampling methods

Informed Undersampling

- *EasyEnsemble*
  - ***Unsupervised****: use random subsets of the majority class to create balance and form multiple classifiers*
- *BalanceCascade*
  - ***Supervised****: iteratively create balance and pull out redundant samples in majority class to form a final classifier*
    1. Generate $E \subset S_{maj}$ $(s.t. |E| = |S_{\min}|)$, and $N = \{E \cup S_{\min}\}$
    2. Induce $H(n)$
    3. Identify N^*_{maj} as samples from N that are correctly classified
    4. Remove N^*_{maj} from S_{maj}
    5. Repeat (1) and induce $H(n+1)$ until stopping criteria is met

Sampling methods

Informed Undersampling

- ***Undersampling using K-nearest neighbor (KNN) classifier***
  - NearMiss-1, NearMiss-2, NearMiss-3, and the "most distant" method
  - NearMiss-2 provides competitive results for imbalanced learning
- ***One-sided selection (OSS)***
  - Selects representative subset E from the majority class
  - Combine with the minority class $N = \{E \cup S_{\min}\}$
  - Refine N with data cleaning techniques

Sampling methods

Synthetic Sampling with Data Generation

- Synthetic minority oversampling technique (SMOTE)
  - Creates artificial minority class data using feature space similarities
  - For $\forall x_i \in S_{\min}$
    1. Randomly choose one of the k nearest neighbor $\hat{x}_i$;
    2. Create a new sample $x_{new} = x_i + (\hat{x}_i - x_i) \times \delta$, where δ is a uniformly distributed random variable.

Sampling methods

Synthetic Sampling with Data Generation

- Synthetic minority oversampling technique (SMOTE)

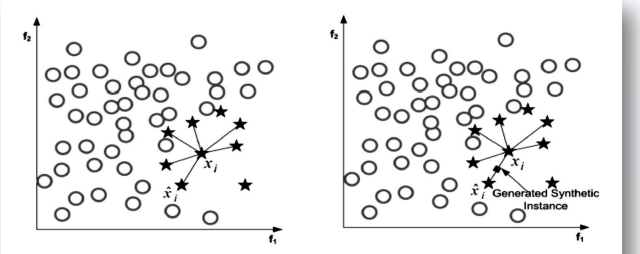


Fig. 3. (a) Example of the K-nearest neighbors for the x_i example under consideration ($K = 6$). (b) Data creation based on euclidian distance.

Sampling methods

Adaptive Synthetic Sampling

- Overcomes over generalization in SMOTE algorithm
  - Border-line-SMOTE:
    1. Determine the set of m-nearest neighbors for each $x_i \in S_{min}$, call it $S_{i:m-NN}$
    2. Identify the number of nearest neighbors in majority class, i.e., $|S_{i:m-NN} \cap S_{maj}|$
    3. Select x_i that satisfies: $\frac{m}{2} \leq |S_{i:m-NN} \cap S_{maj}| < m$

Adaptive Synthetic Sampling

- Overcomes over generalization in SMOTE algorithm
  - Border-line-SMOTE

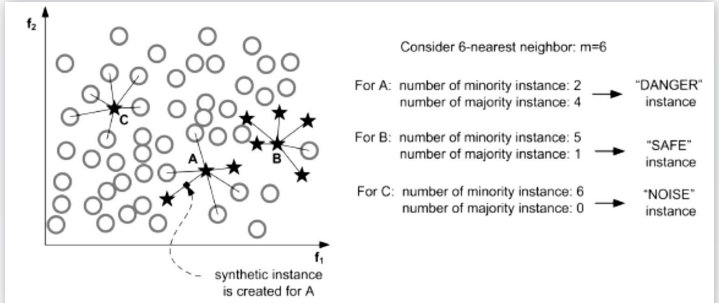


Fig. 4. Data creation based on Borderline instance.

Adaptive Synthetic Sampling

- Overcomes over generalization in SMOTE algorithm
  - ADASYN
    1. Calculate number of synthetic samples $G = \left(|S_{maj}| - |S_{\min}|\right) \times \beta$
    2. for each $x_i \in S_{\min}$, find k-nearest neighbors and calculate ratio $\Gamma_i = \frac{\Delta_i/K}{Z}, i = 1, \ldots, |S_{min}|$ as a distribution function;
    3. Identify the number of synthetic samples to be generated forx_i by $g_i = \Gamma_i \times G$
    4. Generate x_{new} using SMOTE algorithm: $x_{new} = x_i + (\hat{x}_i - x_i) \times \delta$

Sampling with Data Cleaning

- Tomek links
  1. Given a pair (x_i, x_j) where $x_i \in S_{\min}$, $x_j \in S_{maj}$, and the distance between them as $d(x_i, x_j)$
  2. If there is no instance $x_k \; s.t. \; d(x_i, x_k) <. d(x_i, x_j)$ or $d(x_j, x_k) < d(x_i, x_j)$, then (x_i, x_j) is called a Tomek link
- Clean up unwanted inter-class overlapping after synthetic sampling
- Examples:
  - OSS, condensed nearest neighbor and Tomek links (CNN + Tomek links), neighborhood cleaning rule (NCL) based on edited nearest neighbor (ENN), SMOTE+ENN, and SMOTE+Tomek

Sampling with Data Cleaning

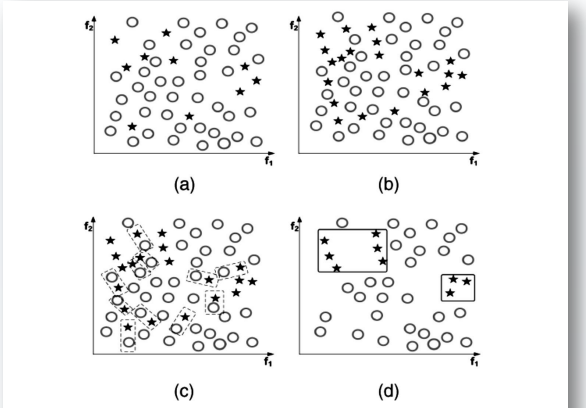


Fig. 5. (a) Original data set distribution. (b) Post-SMOTE data set. (c) The identified Tomek Links. (d) The data set after removing Tomek links.

Cluster-based oversampling (CBO) method

1. For the majority class S_{maj} with m_{maj} clusters
   I. Oversample each cluster $C_{maj:j} \subset S_{maj}, j = 1, \ldots, m_{maj}$ except the largest $C_{maj:\max}$, so that for $\forall j, |C_{maj:j}| = |C_{maj:\max}|$
   II. Calculate the number of majority class examples after oversampling as N_{CBO}
2. For the minority class S_{min} with m_{min} clusters
   I. Oversample each cluster $C_{min:i} \subset S_{min}, i = 1, \ldots, m_{\min}$ to be of the same size $N_{CBO}/m_{\min}$, so that for $\forall i, |C_{min:i}| = N_{CBO}/m_{\min}$

CBO Method

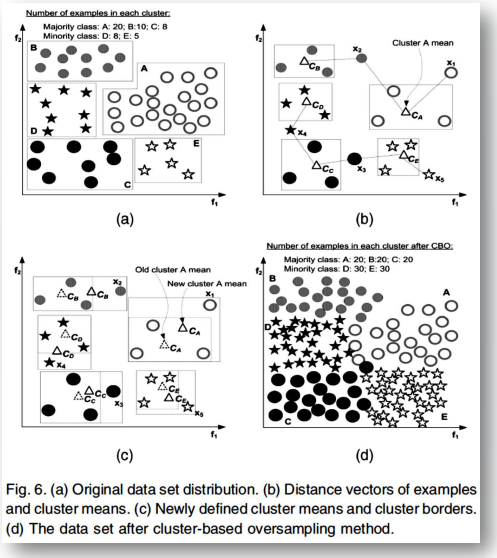


Fig. 6. (a) Original data set distribution. (b) Distance vectors of examples and cluster means. (c) Newly defined cluster means and cluster borders. (d) The data set after cluster-based oversampling method.

Integration of Sampling and Boosting

1. SMOTEBoost
   - SMOTE + Adaboost.M2
   - Introduces synthetic sampling at each boosting iteration
2. DataBoost-IM
   - AdaBoost.M1
   - Generate synthetic date of hard-to-learn samples for both majority and minority classes (usually $|E_{maj}| < |E_{min}|$)
3. JOUS-Boost

Integration of Sampling and Boosting

DataBoost-IM

1. Collect the set E of top misclassified samples (hard-to-learn samples) for both classes with subsets $E_{maj} \subset E$ and $E_{min} \subset E$
2. Identify M_L seeds from E_{maj} and M_S seeds from E_{min}, where
   $M_L = \min(\frac{|S_{maj}|}{|S_{min}|}, |E_{maj}|)$ and $M_S = \min(\frac{|S_{maj}| \times M_L}{|S_{\min}|}, |E_{\min}|)$
3. Generate synthetic set E_{syn} with subsets for both classes:
   $E_{\text{smin}} \subset E_{syn}$ and $E_{smaj} \subset E_{syn}$
   $s.t. |E_{smin}| = M_S \times |S_{\min}|$ and $|E_{smaj}| = M_L \times |S_{maj}|$

Cost-Sensitive Methods

Cost-Sensitive Learning Framework

- Define the cost of misclassifying a majority to a minority as $C(Min, Maj)$
- Typically $C(Maj, Min) > C(Min, Maj)$
- Minimize the overall cost - usually the *Bayes conditional risk* - on the training data set

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

True Class j

Predicted Class i	1	2	…	k
1	C(1,1)	C(1,2)	…	C(1,k)
2	C(2,1)	…	…	.
.	.	…	…	.
k	C(k,1)	…	…	C(k,k)

Fig. 7. Multiclass cost matrix.

Cost-Sensitive Dataspace Weighting with Adaptive Boosting

- Iteratively update the distribution function D_t of the training data according to error of current hypothesis h_t and cost factor C_i
  - Weight updating parameter $\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
  - Error of hypothesis h_t: $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$

Cost-Sensitive Dataspace Weighting with Adaptive Boosting

- Given D_t, h_t, C_i, α_t, and ε_t

1. AdaC1: $D_{t+1}(i) = \frac{D_t(i)\ exp(-\alpha_t C_i h_t(x_i) y_i)}{Z_t}$
2. AdaC2: $D_{t+1}(i) = \frac{C_i D_t(i)\ exp(-\alpha_t h_t(x_i) y_i)}{Z_t}$
3. AdaC3: $D_{t+1}(i) = \frac{C_i D_t(i)\ exp(-\alpha_t C_i h_t(x_i) y_i)}{Z_t}$
4. AdaCost: $D_{t+1}(i) = \frac{D_t(i)\ exp(-\alpha_t h_t(x_i) y_i \beta_i)}{Z_t}$,

   $\beta_i = \beta(sign(y_i, h_t(x_i)), C_i)$

Cost-Sensitive methods

Cost-Sensitive Decision Trees

1. Cost-sensitive adjustments for the decision threshold
   - The final decision threshold shall yield the most dominant point on the ROC curve
2. Cost-sensitive considerations for split criteria
   - The impurity function shall be insensitive to unequal costs
3. Cost-sensitive pruning schemes
   - The probability estimate at each node needs improvement to reduce removal of leaves describing the minority concept
   - Laplace smoothing method and Laplace pruning techniques

Cost-Sensitive methods

Cost-Sensitive Neural Network

Four ways of applying cost sensitivity in neural networks

Modifying probability estimate of outputs	Altering outputs directly	Modify learning rate	Replacing error-minimizing function
• *Applied only at testing stage* • *Maintain original neural networks*	• *Bias neural networks during training to focus on expensive class*	• *Set η higher for costly examples and lower for low-cost examples*	• *Use expected cost minimization function instead*

Kernel-Based Methods

Kernel-based learning framework

- Based on statistical learning and Vapnik-Chervonenkis (VC) dimensions
- Problems with Kernel-based support vector machines (SVMs)
  1. Support vectors from the minority concept may contribute less to the final hypothesis
  2. Optimal hyperplane is also biased toward the majority class

To minimize the **total** error → Biased toward the majority

Kernel-Based Methods

Integration of Kernel Methods with Sampling Methods

1. SMOTE with Different Costs (SDCs) method
2. Ensembles of over/under-sampled SVMs
3. SVM with asymmetric misclassification cost
4. Granular Support Vector Machines—Repetitive Undersampling (GSVM-RU) algorithm

Kernel-Based Methods

Kernel Modification Methods

1. Kernel classifier construction
   - Orthogonal forward selection (OFS) and Regularized orthogonal weighted least squares (ROWLSs) estimator
2. SVM class boundary adjustment
   - Boundary movement (BM), biased penalties (BP), class-boundary alignment(CBA), kernel-boundary alignment (KBA)
3. Integrated approach
   - Total margin-based adaptive fuzzy SVM (TAF-SVM)
4. K-category proximal SVM (PSVM) with Newton refinement
5. Support cluster machines (SCMs), Kernel neural gas (KNG), P2PKNNC algorithm, hybrid kernel machine ensemble (HKME) algorithm, Adaboost relevance vector machine (RVM), …

Active Learning Methods

Active Learning Methods

- SVM-based active learning

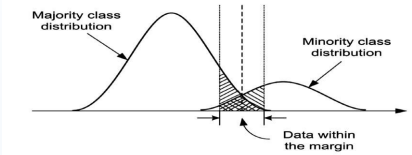


Fig. 8. Data imbalance ratio within and outside the margin [98].

- Active learning with sampling techniques
  - Undersampling and oversampling with active learning for the word sense disambiguation (WSD) imbalanced learning
  - New stopping mechanisms based on maximum confidence and minimal error
  - Simple active learning heuristic (SALH) approach

Active Learning Methods

Additional methods

- One-class learning/novelty detection methods
- Mahalanobis-Taguchi System
- Rank metrics and multitask learning
  - Combination of imbalanced data and the small sample size problem
- Multiclass imbalanced learning
  - AdaC2.M1
  - Rescaling approach for multiclass cost-sensitive neural networks
  - the ensemble knowledge for imbalance sample sets (eKISS) method

The Evaluation of Imbalanced Learning Problem

Assessment Metrics

How to evaluate the performance of imbalanced learning algorithms ?

1. Singular assessment metrics
2. Receiver operating characteristics (ROC) curves
3. Precision-Recall (PR) Curves
4. Cost Curves
5. Assessment Metrics for Multiclass Imbalanced Learning

Assessment Metrics

Singular assessment metrics

		True class: p	True class: n
Hypothesis output	Y	TP (True Positives)	FP (False Positives)
	N	FN (False Negatives)	TN (True Negatives)
Column counts:		P_C	N_C

Fig. 9. Confusion matrix for performance evaluation.

$$Accuracy = \frac{TP + TN}{P_C + N_C}$$

$$ErrorRate = 1 - accuracy$$

- Limitations of accuracy – sensitivity to data distributions

Assessment Metrics

Singular assessment metrics

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN},$$

- Insensitive to data distributions

Assessment Metrics

Singular assessment metrics

More comprehensive metrics

$$F\text{-}Measure = \frac{(1 + \beta)^2 \cdot Recall \cdot Precision}{\beta^2 \cdot Recall + Precision}$$

$\beta = 1$, usually

$$G\text{-}mean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}}$$

Assessment Metrics

Receive Operating Characteristics (ROC) curves

- $TP_{rate} = \frac{TP}{P_C}$
- $FP_{rate} = \frac{FP}{N_C}$
- Area under the curve (AUC)

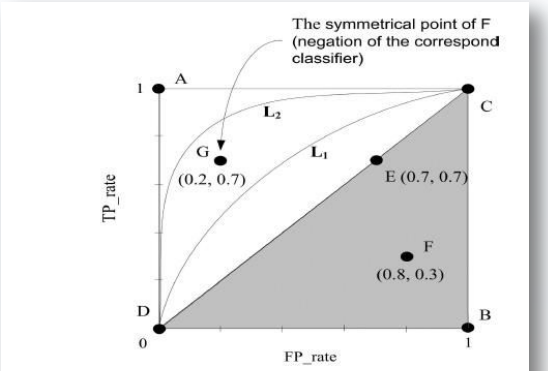


Fig. 10. ROC curve representation.

Assessment Metrics

Precision-Recall (PR) curves

- Plotting the precision rate over the recall rate
- A curve dominates in ROC space (resides in the upper-left hand) ***if and only if*** it dominates (resides in the upper-right hand) in PR space
- PR space has all the analogous benefits of ROC space
- Provide more informative representations of performance assessment under highly imbalanced data

Assessment Metrics

Cost Curves

- $PCF(+)$: the probability of an example being from the positive class
- Expected cost: $E[C] = (1 - TP - FP) \times PCF(+) + FP$

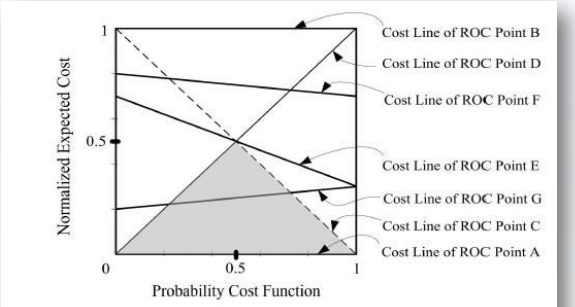


Fig. 11. Cost curve representation.

The Future of Imbalanced Learning Problem

Opportunities and Challenges

- Understanding the Fundamental Problem
- Need of a Uniform Benchmark Platform
- Need of Standardized Evaluation Practices
- Semi-supervised Learning from Imbalanced data

Opportunities and Challenges

Understanding the Fundamental Problem

1. What kind of assumptions will make imbalanced learning algorithms work better compared to learning from the original distributions?
2. To what degree should one balance the original data set?
3. How do imbalanced data distributions affect the computational complexity of learning algorithms?
4. What is the general error bound given an imbalanced data distribution?

Opportunities and Challenges

Need of a Uniform Benchmark Platform

1. Lack of a uniform benchmark for standardized performance assessments
2. Lack of data sharing and data interoperability across different disciplinary domains;
3. Increased procurement costs, such as time and labor, for the research community as a whole group since each research group is required to collect and prepare their own data sets.

Opportunities and Challenges

Need of Standardized Evaluation Practices

- Establish the practice of using the curve-based evaluation techniques
- A standardized set of evaluation practices for proper comparisons

Opportunities and Challenges

Incremental Learning from Imbalanced Data Streams

1. How can we autonomously adjust the learning algorithm if an imbalance is introduced in the middle of the learning period?
2. Should we consider rebalancing the data set during the incremental learning period? If so, how can we accomplish this?
3. How can we accumulate previous experience and use this knowledge to adaptively improve learning from new data?
4. How do we handle the situation when newly introduced concepts are also imbalanced (i.e., the imbalanced concept drifting issue)?

Opportunities and Challenges

Semi-supervised Learning from Imbalanced Data

1. How can we identify whether an unlabeled data example came from a balanced or imbalanced underlying distribution?
2. Given an imbalanced training data with labels, what are the effective and efficient methods for recovering the unlabeled data examples?
3. What kind of biases may be introduced in the recovery process (through the conventional semi-supervised learning techniques) given imbalanced, labeled data?

Reference:

This lecture notes is based on the following paper:

H. He and E. A. Garcia, "Learning from Imbalanced Data," IEEE Trans. Knowledge and Data Engineering, vol. 21, issue 9, pp. 1263-1284, 2009

WHAT ARE MISSING DATA?

- In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation.
- This situation mostly occur as a result of manual data entry procedures, equipment errors and incorrect measurements.

PROBLEMS OF MISSING DATA

-loss of efficiency

- complications in handling and analyzing the data

- bias resulting from differences between missing and complete data.

TYPES OF MISSING DATA

- MISSING COMPLETELY AT RANDOM (MCAR)

Values in a data set can miss completely at random (MCAR) if the events that lead to any particular data-item missing are independent both of observable variables and of unobservable parameters of interest, and occur entirely at random.

TYPES OF MISSING DATA (Cont'd)

- MISSING AT RANDOM (MAR)

Missing at random (MAR) is an alternative, and occurs when the missing-ness is related to a particular variable, but it is not related to the value of the variable that has missing data.

An example of this is accidentally omitting an answer on a questionnaire.

TYPES OF MISSING DATA (Cont'd)

- MISSING NOT AT RANDOM (MNAR)

This is data that is missing for a specific reason (i.e. the value of the variable that is missing is related to the reason it is missing).

An example of this is if certain question on a questionnaire tend to be skipped deliberately by participants with certain characteristics.

HOW TO DEAL WITH MISSING DATA

- REDUCING THE DATA SET

The simplest solution for the missing data imputation problem is the reduction of the data set and elimination of all missing values. This can be done by elimination of samples (rows) with missing values or elimination of attributes (columns) with missing values

HOW TO DEAL WITH MISSING DATA

REPLACE MISSING VALUE WITH MEAN

This method replaces each missing value with mean of the attribute.
The mean is calculated based on all known values of the attribute. This method is usable only for numeric attributes and is usually combined with replacing missing values with most common attribute value for symbolic attributes.

HOW TO DEAL WITH MISSING DATA

REPLACE MISSING VALUE WITH MEAN FOR THE GIVEN CLASS

This method is similar to the previous one. The difference is in that the mean is not calculated from all known values of the attribute, but only attributes values belonging to given class are used.

HOW TO DEAL WITH MISSING DATA

REPLACE MISSING VALUE WITH MOST COMMON ATTRIBUTE VALUE

Since the mean is affected by the presence of outliers it seems natural to use the median instead just to assure robustness. In this case the missing data for a given attribute is replaced by the median of all known values of that attribute in the class where the instance with the missing value belongs

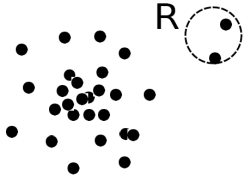
HOW TO DEAL WITH MISSING DATA

CLOSEST FIT

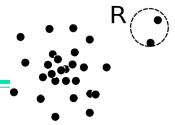
The closest fit algorithm for missing attribute values is based on replacing a missing attribute value with an existing value of the same attribute from another case that resembles as much as possible the case with missing attribute values

What Are Outliers?

- **Outlier**: A data object that **deviates significantly** from the normal objects as if it were **generated by a different mechanism**
  - Ex.: Unusual credit card purchase, sports: Michael Jordon, Wayne Gretzky, ...
- Outliers are different from the noise data
  - Noise is random error or variance in a measured variable
  - Noise should be removed before outlier detection
- Outliers are interesting: It violates the mechanism that generates the normal data
- Outlier detection vs. *novelty detection*: early stage, outlier; but later merged into the model
- Applications:
  - Credit card fraud detection
  - Telecom fraud detection
  - Customer segmentation
  - Medical analysis



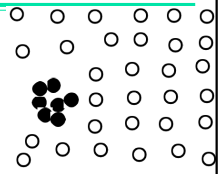
Types of Outliers (I)



Global Outlier

- Three kinds: *global, contextual* and *collective* outliers
- **Global outlier** (or point anomaly)
  - Object is O_g if it significantly deviates from the rest of the data set
  - Ex. Intrusion detection in computer networks
  - Issue: Find an appropriate measurement of deviation
- **Contextual outlier** (or *conditional outlier*)
  - Object is O_c if it deviates significantly based on a selected context
  - Ex. 80° F in Urbana: outlier? (depending on summer or winter?)
  - Attributes of data objects should be divided into two groups
    - Contextual attributes: defines the context, e.g., time & location
    - Behavioral attributes: characteristics of the object, used in outlier evaluation, e.g., temperature
  - Can be viewed as a generalization of *local outliers*—whose density significantly deviates from its local area
  - Issue: How to define or formulate meaningful context?

Types of Outliers (II)



Collective Outlier

- **Collective Outliers**
  - A subset of data objects *collectively* deviate significantly from the whole data set, even if the individual data objects may not be outliers
  - Applications: E.g., *intrusion detection*:
    - When a number of computers keep sending denial-of-service packages to each other
  - Detection of collective outliers
    - Consider not only behavior of individual objects, but also that of groups of objects
    - Need to have the background knowledge on the relationship among data objects, such as a distance or similarity measure on objects.
- A data set may have multiple types of outlier
- One object may belong to more than one type of outlier

Challenges of Outlier Detection

- Modeling normal objects and outliers properly
  - Hard to enumerate all possible normal behaviors in an application
  - The border between normal and outlier objects is often a gray area
- Application-specific outlier detection
  - Choice of distance measure among objects and the model of relationship among objects are often application-dependent
  - E.g., clinic data: a small deviation could be an outlier; while in marketing analysis, larger fluctuations
- Handling noise in outlier detection
  - Noise may distort the normal objects and blur the distinction between normal objects and outliers. It may help hide outliers and reduce the effectiveness of outlier detection
- Understandability
  - Understand why these are outliers: Justification of the detection
  - Specify the degree of an outlier: the unlikelihood of the object being generated by a normal mechanism

Outlier Detection I: Supervised Methods

- Two ways to categorize outlier detection methods:
  - Based on whether user-*labeled* examples of outliers can be obtained:
    - Supervised, semi-supervised vs. unsupervised methods
  - Based on *assumptions about normal data and outliers*:
    - Statistical, proximity-based, and clustering-based methods
- **Outlier Detection I: Supervised Methods**
  - Modeling outlier detection as a classification problem
    - Samples examined by domain experts used for training & testing
  - Methods for Learning a classifier for outlier detection effectively:
    - Model normal objects & report those not matching the model as outliers, or
    - Model outliers and treat those not matching the model as normal
  - Challenges
    - Imbalanced classes, i.e., outliers are rare: Boost the outlier class and make up some artificial outliers
    - Catch as many outliers as possible, i.e., recall is more important than accuracy (i.e., not mislabeling normal objects as outliers)

Outlier Detection II: Unsupervised Methods

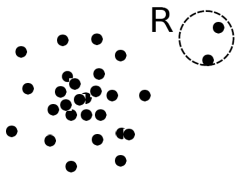
- Assume the normal objects are somewhat ``clustered'' into multiple groups, each having some distinct features
- An outlier is expected to be far away from any groups of normal objects
- Weakness: Cannot detect collective outlier effectively
  - Normal objects may not share any strong patterns, but the collective outliers may share high similarity in a small area
- Ex. In some intrusion or virus detection, normal activities are diverse
  - Unsupervised methods may have a high false positive rate but still miss many real outliers.
  - Supervised methods can be more effective, e.g., identify attacking some key resources
- Many clustering methods can be adapted for unsupervised methods
  - Find clusters, then outliers: not belonging to any cluster
  - Problem 1: Hard to distinguish noise from outliers
  - Problem 2: Costly since first clustering: but far less outliers than normal objects
    - Newer methods: tackle outliers directly

Outlier Detection III: Semi-Supervised Methods

- Situation: In many applications, the number of labeled data is often small: Labels could be on outliers only, normal objects only, or both
- Semi-supervised outlier detection: Regarded as applications of semi-supervised learning
- If some labeled normal objects are available
  - Use the labeled examples and the proximate unlabeled objects to train a model for normal objects
  - Those not fitting the model of normal objects are detected as outliers
- If only some labeled outliers are available, a small number of labeled outliers many not cover the possible outliers well
  - To improve the quality of outlier detection, one can get help from models for normal objects learned from unsupervised methods

Outlier Detection (1): Statistical Methods

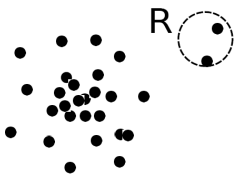
- Statistical methods (also known as model-based methods) assume that the normal data follow some statistical model (a stochastic model)
  - The data not following the model are outliers.
- Example (right figure): First use Gaussian distribution to model the normal data
  - For each object y in region R, estimate $g_D(y)$, the probability of y fits the Gaussian distribution
  - If $g_D(y)$ is very low, y is unlikely generated by the Gaussian model, thus an outlier



- Effectiveness of statistical methods: highly depends on whether the assumption of statistical model holds in the real data
- There are rich alternatives to use various statistical models
  - E.g., parametric vs. non-parametric

Outlier Detection (2): Proximity-Based Methods

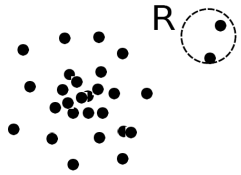
- An object is an outlier if the nearest neighbors of the object are far away, i.e., the **proximity** of the object is **significantly deviates** from the proximity of most of the other objects in the same data set
- Example (right figure): Model the proximity of an object using its 3 nearest neighbors
  - Objects in region R are substantially different from other objects in the data set.
  - Thus the objects in R are outliers



- The effectiveness of proximity-based methods highly relies on the proximity measure.
- In some applications, proximity or distance measures cannot be obtained easily.
- Often have a difficulty in finding a group of outliers which stay close to each other
- Two major types of proximity-based outlier detection
  - Distance-based vs. density-based

Outlier Detection (3): Clustering-Based Methods

- Normal data belong to large and dense clusters, whereas outliers belong to small or sparse clusters, or do not belong to any clusters
- Example (right figure): two clusters
  - All points not in R form a large cluster
  - The two points in R form a tiny cluster, thus are outliers



- Since there are many clustering methods, there are many clustering-based outlier detection methods as well
- Clustering is expensive: straightforward adaption of a clustering method for outlier detection can be costly and does not scale up well for large data sets