

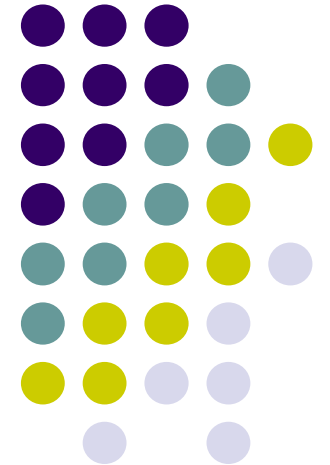
# CSC/DA 545

## Introduction to Data Mining

---

Feature Engineering

Instructor: Lin-Ching Chang



# For the next 3 weeks (old plan)

---



- Topics we will be covering...
  - Categorical Variable Encoding
  - Text Preprocessing for NLP and Machine Learning
- Feature Engineering
- Novelty and Outlier Detection
- Nature Language Processing (NLP)
- Language Modeling and ChatGPT

# For the next 3 weeks

---



- Topics we will be covering...
  - Categorical Variable Encoding
  - Feature Engineering
  - ~~• Novelty and Outlier Detection~~
  - Nature Language Processing (NLP)
  - Text Preprocessing
  - Language Modeling (n-gram)
  - ~~• ChatGPT~~

# What is Feature Engineering?

---



- The process of selecting, manipulating, and transforming raw data into features that can be used in machine learning.
- The goal of feature engineering
  - **Improve model performance (better accuracy)**
  - **Reduce data dimension**



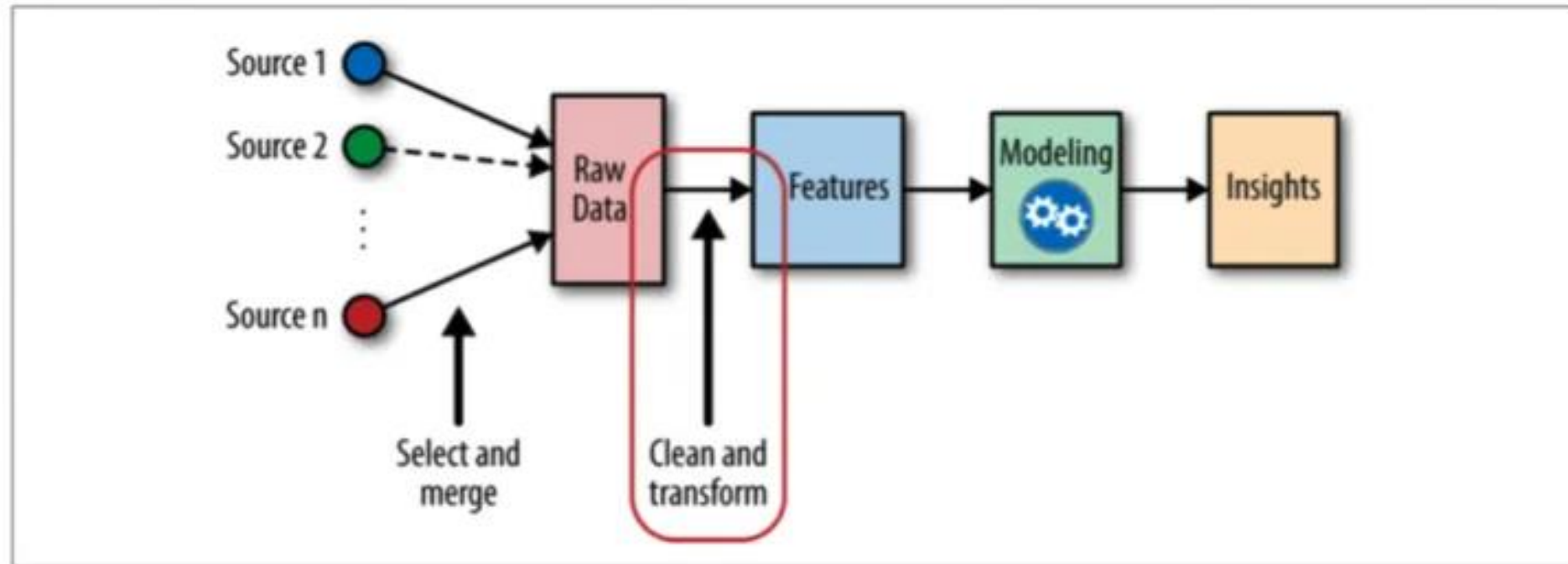
Feature engineering is the process of selecting, transforming, and extracting useful features from raw data to create inputs for a machine learning model. It involves identifying relevant variables, creating new features by combining or modifying existing ones, and cleaning and normalizing data to remove noise and inconsistencies.



The goal of feature engineering is to improve the performance of machine learning models by providing them with meaningful and informative inputs. By selecting and engineering the right features, machine learning models can more accurately identify patterns and make predictions based on the data.

Feature engineering is a crucial step in the machine learning pipeline, and it requires domain knowledge, creativity, and careful analysis of the data. It can involve techniques such as one-hot encoding, normalization, dimensionality reduction, and feature scaling, among others.

# When to perform feature engineering?



feature engineering goes here!

 @amcasari

# Common types of feature engineering:

---



- **Scaling and normalization**
- **Filling missing values**
- **Feature coding**
- **Feature construction**
- **Feature selection**
- **Feature extraction**



There are several common types of feature engineering that are used to create meaningful and informative features for machine learning models. Some of these include:



1. Feature extraction: This involves extracting new features from existing data. For example, if you have a dataset of images, you could extract features such as color histograms, texture features, or shape descriptors.
2. Feature transformation: This involves transforming existing features to make them more useful for a machine learning model. For example, you might apply a logarithmic transformation to a feature that has a skewed distribution to make it more normally distributed.
3. Feature scaling: This involves scaling features to ensure that they are on the same scale. This is important for models that are sensitive to differences in scale, such as k-nearest neighbors and support vector machines.
4. Feature selection: This involves selecting a subset of the most relevant features for a machine learning model. This can help to improve model performance and reduce overfitting.
5. Encoding categorical variables: This involves converting categorical variables into numerical features that can be used in a machine learning model. This can involve techniques such as one-hot encoding or ordinal encoding.
6. Handling missing values: This involves imputing missing values in the dataset. This can involve techniques such as mean imputation, median imputation, or regression imputation.

These are just a few examples of the types of feature engineering techniques that are commonly used. The specific techniques that are used will depend on the dataset and the machine learning model that is being used.



# Standardization and Normalization



- **Standardization:** To transform data so that it has zero mean and unit variance. Also called scaling
  - Use function `sklearn.preprocessing.scale()`
  - Parameters:
    - **X:** Data to be scaled
    - **with\_mean:** Boolean. Whether to center the data (make zero mean)
    - **with\_std:** Boolean (whether to make unit standard deviation)
- **Normalization:** to transform data so that it is scaled to the [0,1] range.
  - Use function `sklearn.preprocessing.normalize()`
  - Parameters:
    - **X:** Data to be normalized
    - **norm:** which norm to use: l1 or l2
    - **axis:** whether to normalize by row or column

# Feature coding

---



- It involves choosing a set of symbolic values to represent different categories.
- Concepts can be captured with a single column that comprises multiple values, or they can be captured with multiple columns, each of which represents a single value and has a true or false in each field.

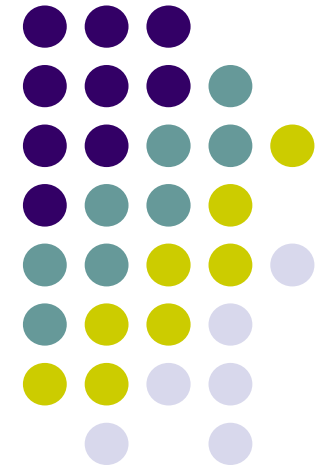
# Feature construction

---



- It creates a new feature(s) from one or more other features.
- For example, using the date you can add a feature that indicates the day of the week. With this added insight, the algorithm could discover that certain outcomes are more likely on a Monday or a weekend.

# Feature Selection



# Feature Selection



- **Sequential search strategies**

- Statistical Hypothesis Testing
- Scatter Matrix
- Sequential Forward Selection
- Sequential Backward Selection
- Plus-I Minus-r Selection
- Bidirectional Search
- Floating Search

*We will discuss some of those strategies*

- **Exponential search methods**

- Branch and Bound
- Approximate Monotonicity with Branch and Bound
- Beam Search

*We will not cover those strategies*

- **Randomized algorithms**

- Random Generation plus Sequential Selection
- Simulated Annealing
- Genetic Algorithms

Save &amp; Submit

Cancel



There are several algorithms that can be used for feature selection in machine learning. Some of the most commonly used algorithms include:



1. Filter methods: These methods use statistical measures to rank the importance of features. Common filter methods include Pearson's correlation coefficient, mutual information, and chi-squared test.
2. Wrapper methods: These methods evaluate the performance of a machine learning model with different subsets of features. Common wrapper methods include recursive feature elimination and forward/backward selection.
3. Embedded methods: These methods combine feature selection with model training. Common embedded methods include LASSO (least absolute shrinkage and selection operator) and Ridge regression.
4. Principal Component Analysis (PCA): PCA is a dimensionality reduction technique that can be used for feature selection. It transforms the original features into a smaller set of uncorrelated features, called principal components, which capture most of the variation in the original data.
5. Tree-based methods: These methods use decision trees to evaluate the importance of features. Common tree-based methods include random forests and gradient boosting.

The choice of feature selection algorithm will depend on the specific problem and dataset. It is often useful to try multiple algorithms and compare their performance to find the best approach.

Was this response better or worse?



Better



Worse



Same

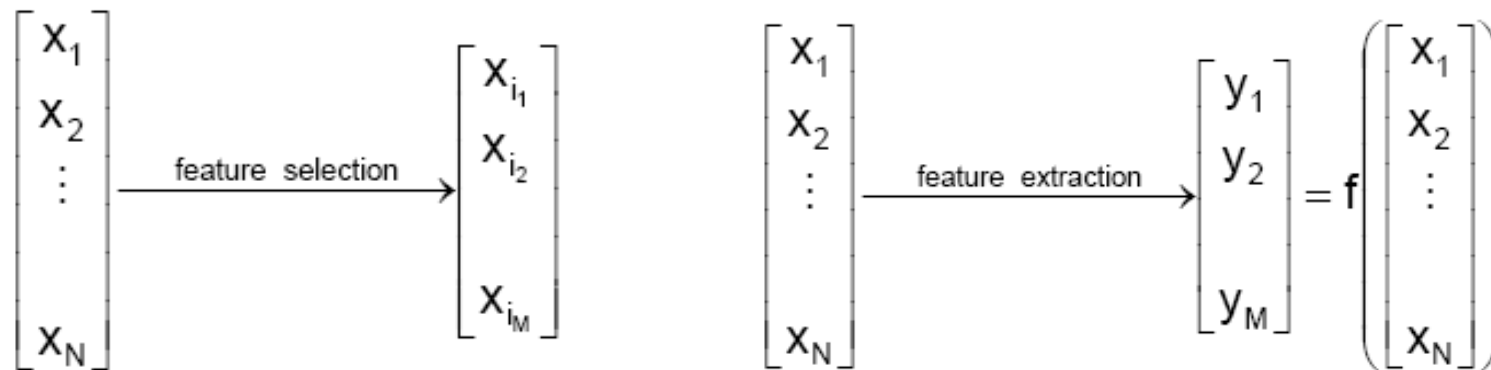


The response is not wrong, but not right either.

# Feature selection vs. Feature extraction



- There are two general approaches for performing dimensionality reduction
  - **Feature selection:** Selecting a subset of the existing features without a transformation
  - **Feature extraction:** Transforming the existing features into a lower dimensional space



# Feature selection vs. Feature extraction



- **Feature selection, also called Feature Subset Selection (FSS)**
  - Although FSS can be thought of as a special case of feature extraction, in practice it is a quite different problem
  - FSS looks at the issue of dimensionality reduction from a different perspective
  - FSS has a unique set of methodologies
- **Feature extraction**
  - We derived the “optimal” linear features for two objective functions
    - PCA
    - (Fisher’s) LDA (we will not cover this)

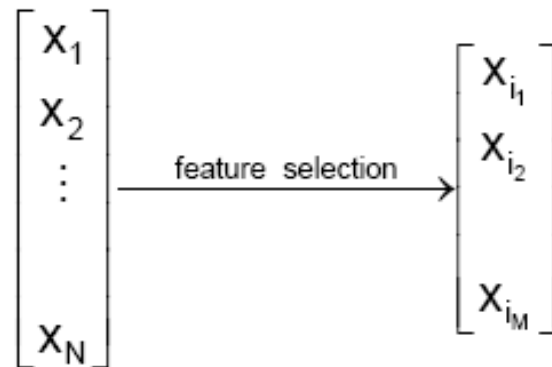


# Feature Subset Selection



- **Definition**

- Given a feature set  $X = \{x_i \mid i=1 \dots N\}$ , find a subset  $Y_M = \{x_{i_1}, x_{i_2}, \dots, x_{i_M}\}$ , with  $M < N$ , that optimizes an **objective function**  $J(Y)$ , ideally the probability of correct classification



$$\{x_{i_1}, x_{i_2}, \dots, x_{i_M}\} = \underset{M, i_m}{\operatorname{argmax}} [J\{x_i \mid i = 1 \dots N\}]$$

- **Why Feature Subset Selection?**

- Why not use the more general feature extraction methods, and simply project a high dimensional feature vector onto a low-dimensional space?

# Feature Subset Selection



- **Feature Subset Selection is necessary in a number of situations**
  - Features may be expensive to obtain
    - You evaluate a large number of features (sensors) in the test bed and select only a few for the final implementation
  - You may want to extract meaningful rules from your classifier
    - When you transform or project, the measurement units of your features (length, weight, etc.) are lost
  - Features may not be numeric
    - A typical situation in the machine learning domain
- **In addition, fewer features means fewer parameters for the machine learning task**
  - Improved the generalization capabilities
  - Reduced complexity and run-time

# The Ultimate Goal of a Classifier



- Designing a classifier that exhibit **a good generalization performance**. That is, to have a good performance (or low error rate) when dealing with data outside the training set.
- A classifier may be designed to have very small error rate over the training data set, yet its generalization performance can be very poor.
- It turns out that, in order to design a classifier with good generalization performance, **the number of training data**,  $N$ , must be large enough w.r. to its **complexity**.

# Classifier vs Features



- For a large class of classifiers the **complexity** is directly related to **the number of the features**, i.e., the dimensionality of the feature space.
- There are cases, however, where the complexity of the classifier does **not** depend on the dimensionality of the feature space, e.g., Support Vector Machines (SVM).
- In any case, **reducing the number of features** is always necessary to get rid of uninformative features, or features that carry redundant information.

# Data Preprocessing



- Prior to any feature selection, data preprocessing is often a necessary step.
- Outlier removal
  - An outlier is defined as a point that lies very far from the mean of the corresponding random variable. Such points result in large errors during training. If such points are the result of erroneous measurements, they have to be removed.
- Data normalization
  - Features with large values have large influence compared to others with small values, although this may not necessarily reflect a respective significance towards the design of the classifier.
- Missing data
  - Some of the values of certain features may be missing.

# Data Normalization



- A common technique is to **normalize each feature** via the respective estimate of the mean and variance. That is for the  $k^{\text{th}}$  feature, i.e., each feature will have **Gaussian distribution with zero mean and unit variance**.
- API Reference in sklearn.preprocessing.**StandardScaler**
  - <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing>

$$\bar{x}_k = \frac{1}{N} \sum_{i=1}^N x_{ik} \quad , \quad k = 1, 2, \dots, l$$

$$\sigma_k^2 = \frac{1}{N-1} \sum_{i=1}^N (x_{ik} - \bar{x}_k)^2$$

$$\hat{x}_{ik} = \frac{x_{ik} - \bar{x}_k}{\sigma_k}$$

# Alternative Standardization

---



- Scaling features to a range
  - between a given minimum and maximum value
  - often between zero and one
  - See `sklearn.preprocessing.MinMaxScaler` or `sklearn.preprocessing.MaxAbsScaler`

# Missing data



- Given  $N$  training feature vectors, in some of them, the values of certain features may be missing.
- **How to Handle Missing Data?**
- The missing values can be completed by a number of methods, e.g.,
  - Ignore the tuple: usually done when class label is missing (when doing classification)—not effective when the % of missing values per attribute varies considerably
  - Fill in the missing value manually: tedious + infeasible?
  - Fill in it automatically



# Fill in a missing data automatically

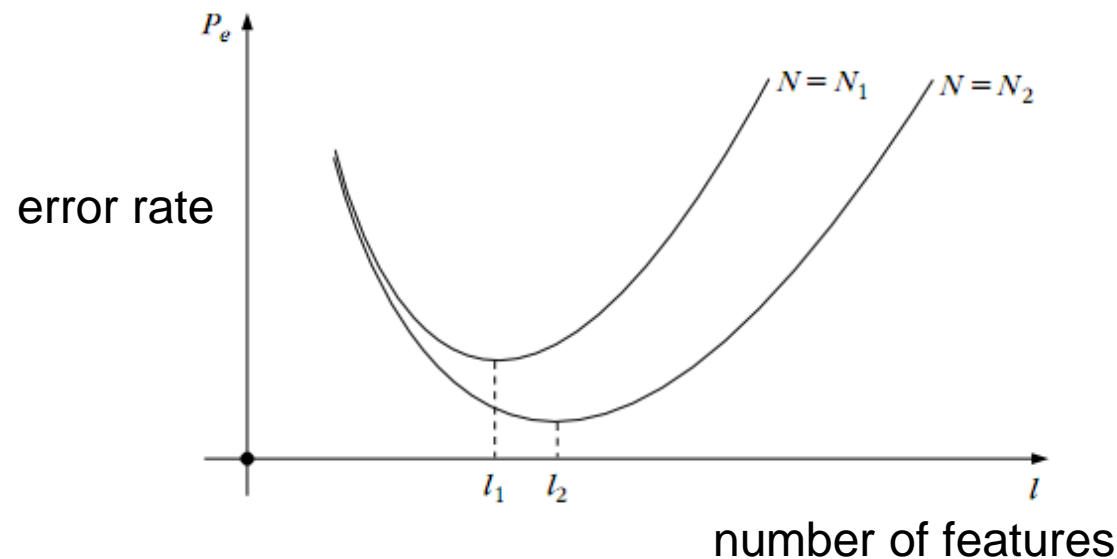


- a global constant
- a random value within a range
- A random value from a distribution
- the attribute mean, median, or mode
- the attribute mean for all samples belonging to the same class: smarter
- **the most probable value**: inference-based such as Bayesian formula or decision tree
  - Model-based methods
    - Maximum Likelihood Estimation
    - Multiple Imputation

# The Peaking Phenomenon



- **Do more features implies a better classifier?**
- If, **in an ideal world**, the class pdfs were known, then increasing the number of features would be beneficial.
- **In practice**, the general trend is that for **a finite number of training points**, increasing the number of features initially improves the generalization error rate, but after a certain value, the generalization error rate increases.



# The Main Goals in Feature Selection

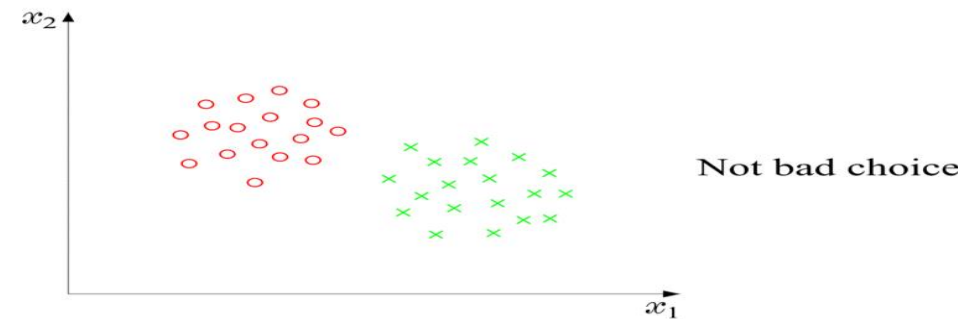
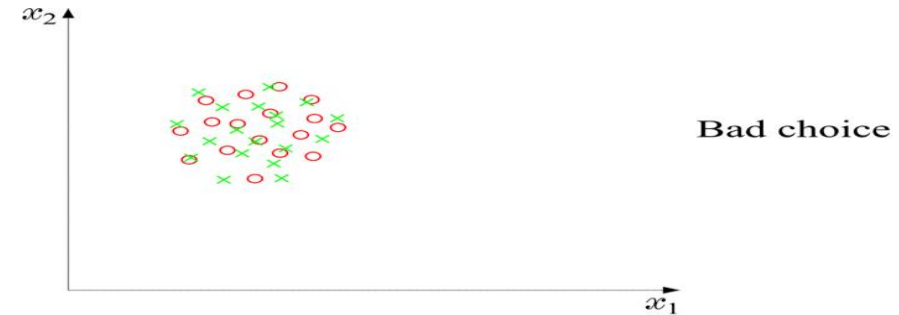


- Select the “optimum” number  $m$  of features
- Select the “best”  $m$  features
- Large  $m$  has a three-fold disadvantage:
  - High computational demands
  - Low generalization performance
  - Poor error estimates
- Given  $N$  (number of features),
  - $m$  must be large enough to learn what makes classes different and what makes patterns in the same class similar
  - $m$  must be small enough not to learn what makes patterns of the same class different
- In practice,  $m < N/3$  has been reported to be a sensible choice for a number of cases

# Feature Selection



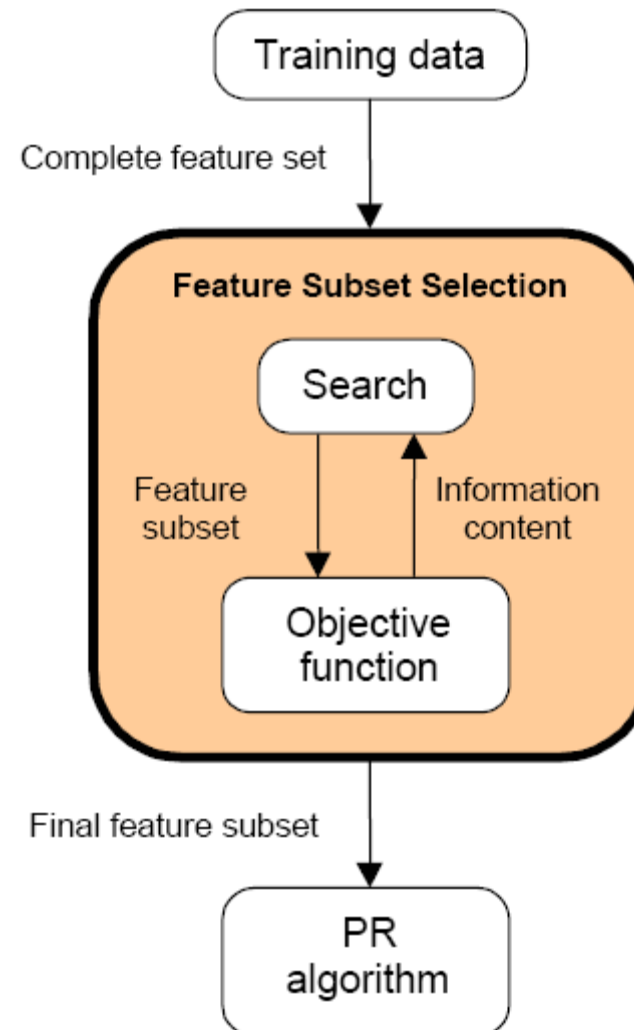
- Once  $m$  has been decided, choose the  $m$  most informative features
- Best:
  - Large between class distance,
  - Small within class variance
- Question:
  - How can you find those features?
  - search strategy
  - objective function



# Search Strategy and Objective Function



- **Feature Subset Selection requires**
  - A search strategy to select candidate subsets
  - An objective function to evaluate these candidates

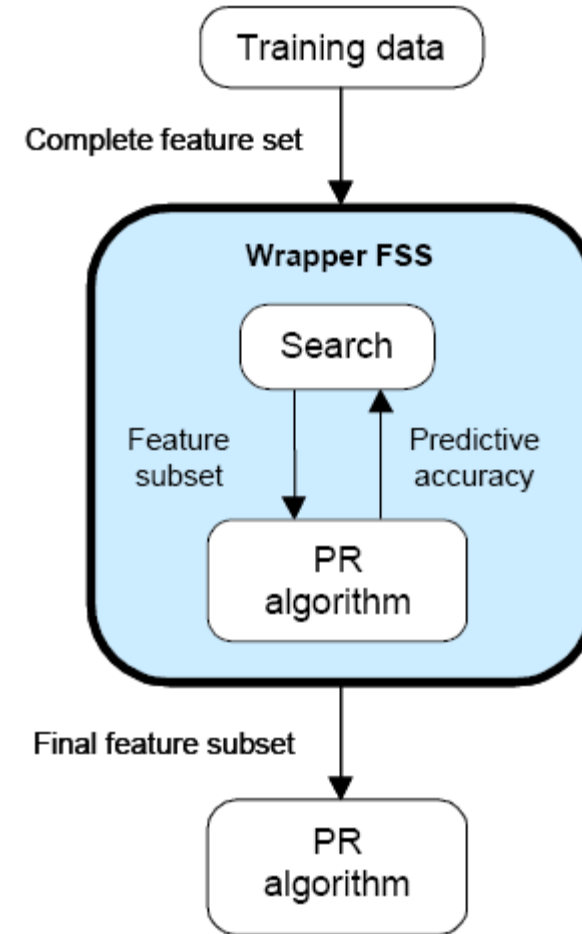
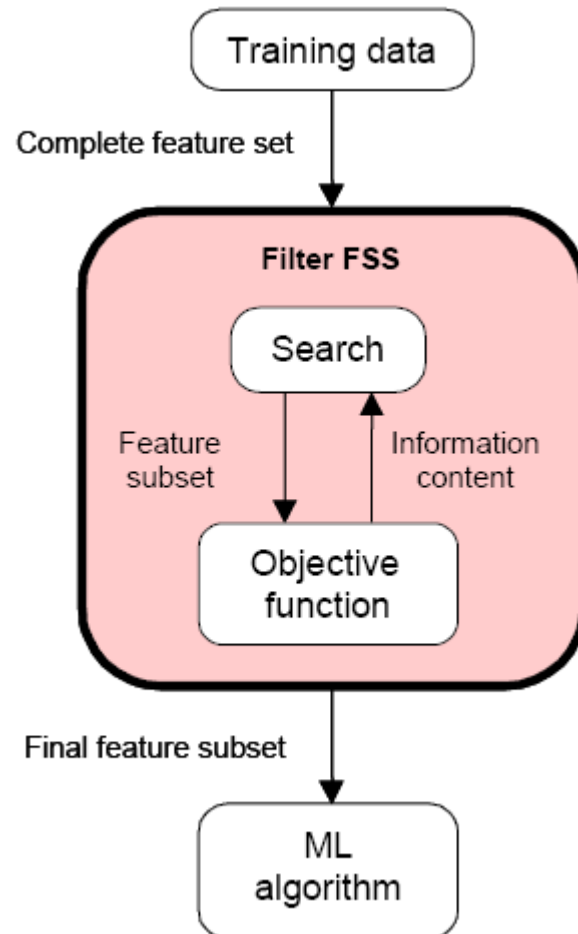


# Objective Function



- **Objective functions are divided in two groups**
  - **Filters:** The objective function evaluates feature subsets by their **information content**, typically interclass distance, statistical dependence or information-theoretic measures
  - **Wrappers:** The objective function is a **pattern classifier**, which evaluates feature subsets by their **predictive accuracy** (recognition rate on test data) by statistical resampling or crossvalidation

# Objective Function



# Filter Types



- **Distance or separability measures**
  - These methods use **distance metrics** to measure class separability, such as
    - Distance between classes: Euclidean, Mahalanobis, etc.
- **Correlation and information-theoretic measures**
  - These methods are based on the rationale that **good feature subsets**
    - (1) contain features highly correlated with (predictive of) the class (i.e. target variable)
    - (2) uncorrelated with (not predictive of) each other



# Filter Types



- **Correlation and information-theoretic measures**

- **Linear relation measures**

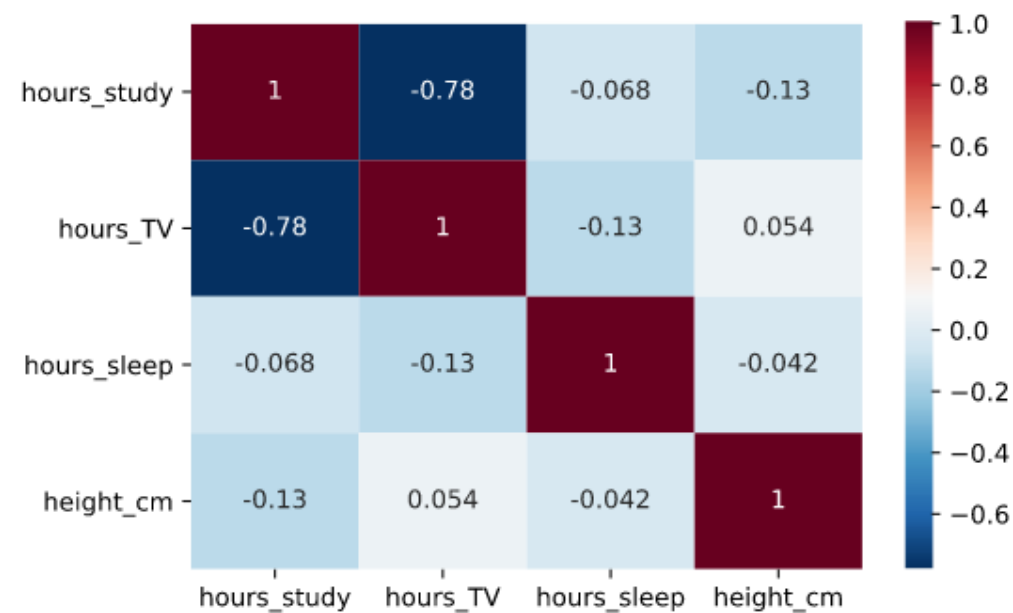
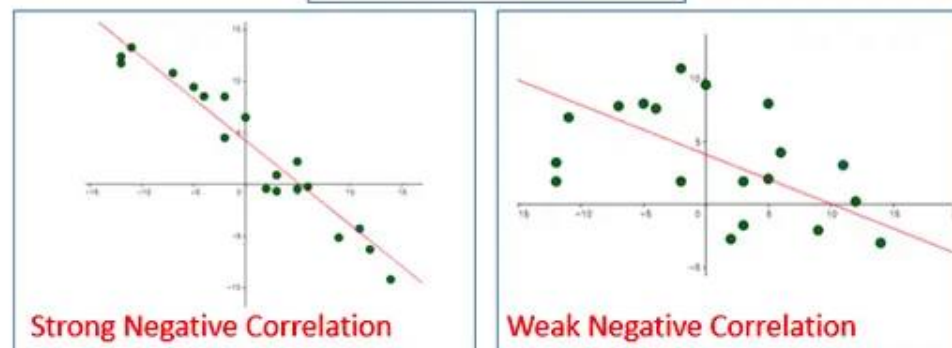
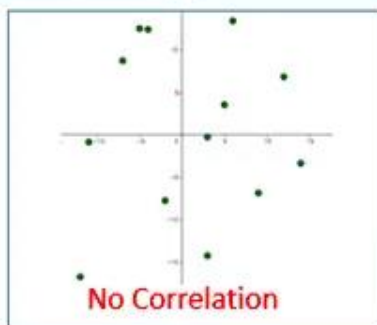
- Linear relationship between variables can be measured using the **correlation coefficient**
- Pearson correlation coefficient

$$r = \frac{\sum (x_i - \bar{x}) (y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

- **Non-Linear relation measures**

- Correlation is only capable of measuring linear dependence.
- A more powerful measure is the **mutual information**

# Correlation Coefficient – Correlation Between Features



```
import matplotlib.pyplot as plt
import seaborn as sns

corr_matrix = X_num.corr(method='pearson') # 'pearson' is default

sns.heatmap(corr_matrix, annot=True, cmap='RdBu_r')
plt.show()
```

# Filters vs. Wrappers



- **Filters**

## Advantages

- **Fast execution**: Filters generally involve a non-iterative computation on the dataset, which can execute much faster than a classifier training session
- **Generality**: Since filters evaluate the intrinsic properties of the data, rather than their interactions with a particular classifier, their results exhibit more generality: the solution will be “good” for a larger family of classifiers

- **Wrappers**

## Advantages

- **Accuracy**: wrappers generally achieve better recognition rates than filters since they are tuned to the specific interactions between the classifier and the dataset
- **Ability to generalize**: wrappers have a mechanism to avoid overfitting, since they typically use cross-validation measures of predictive accuracy

# Filters vs. Wrappers



- **Filters**

## Disadvantages

- **Tendency to select large subsets:** Since the filter objective functions are generally monotonic, the filter tends to select the full feature set as the optimal solution. This forces the user to select an arbitrary cutoff on the number of features to be selected

- **Wrappers**

## Disadvantages

- **Slow execution:** since the wrapper must train a classifier for each feature subset (or several classifiers if cross-validation is used), the method can become unfeasible for computationally intensive methods
- **Lack of generality:** the solution lacks generality since it is tied to the bias of the classifier used in the evaluation function. The “optimal” feature subset will be specific to the classifier under consideration

# Search Strategy and Objective Function



- **Search Strategy**

- Exhaustive evaluation : find the best M features out of N features
  - Feature subsets involves  $\binom{N}{M}$  combinations for a fixed value of M, and  $2^N$  combinations if M must be optimized as well
  - This number of combinations is unfeasible, even for moderate values of M and N, so a search procedure must be used in practice
  - For example, exhaustive evaluation of 10 out of 20 features involves 184,756 feature subsets; exhaustive evaluation of 10 out of 100 involves more than  $10^{13}$  feature subsets [Devijver and Kittler, 1982]
- A search strategy is therefore needed to direct the FSS process as it explores the space of all possible combination of features

- **Objective Function**

- The objective function evaluates candidate subsets and returns a measure of their “goodness”, a feedback signal used by the search strategy to select new candidates

# Search Strategies



- **There is a large number of search strategies, which can be grouped in three categories**
    - **Sequential** algorithms
      - These algorithms add or remove features sequentially, but have a tendency to become trapped in local minima
    - **Exponential** algorithms
      - These algorithms evaluate a number of subsets that grows exponentially with the dimensionality of the search space
    - **Randomized** algorithms
      - These algorithms incorporating randomness into their search procedure to escape local minima
- } *will not cover*

# Sequential Search Strategies



- Representative examples of sequential search include
  - Statistical Hypothesis Testing/Scatter Matrix
  - Sequential Forward Selection
  - Sequential Backward Selection
  - Plus-I Minus-r Selection
  - Bidirectional Search
  - Sequential Floating Selection

# Feature selection in Scikit-learn (2 methods here)



- Removing features with low variance
  - from `sklearn.feature_selection` import `VarianceThreshold`
- Univariate feature selection
  - `SelectKBest` removes all but the **highest scoring** features

By changing the 'score\_func' parameter we can apply the method for both classification and regression data.

```
select = SelectKBest(score_func=chi2, k=3)
z = select.fit_transform(x,y)
```

```
select = SelectKBest(score_func=f_regression, k=8)
z = select.fit_transform(x, y)
```

More methods later.....



# Naïve Sequential Feature Selection

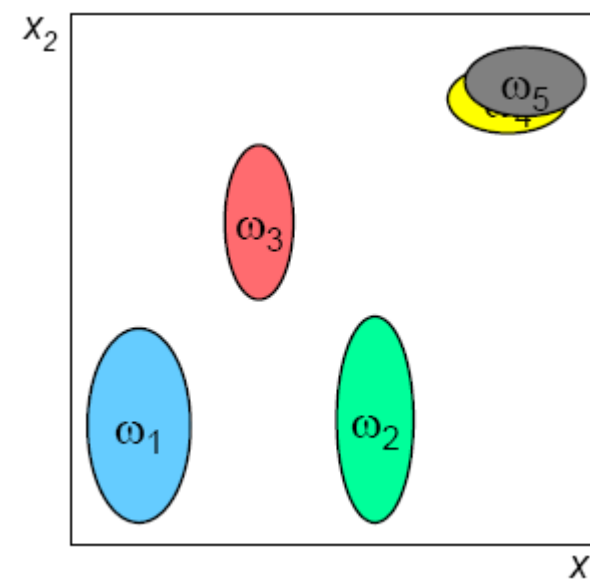
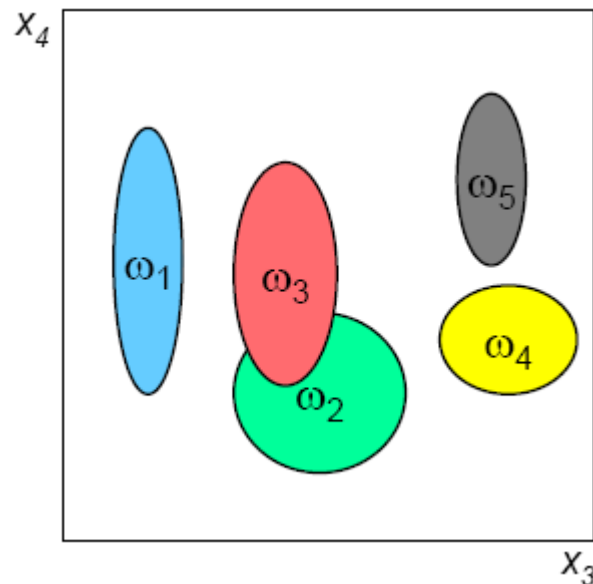


- One may be tempted to **evaluate each individual feature separately** and select those **M** features with the highest scores
  - Unfortunately, this strategy will VERY RARELY work since it does not account for **feature dependence**
- An example will help illustrate the poor performance that can be expected from this naïve approach
  - a 4-dimensional pattern recognition problem with 5 classes.
  - The objective is to select the best subset of 2 features using the naïve sequential feature selection procedure
  - Any reasonable objective function will rank features according to this sequence:  
 $J(x_1) > J(x_2) \approx J(x_3) > J(x_4)$ 
    - $x_1$  is, without a doubt, the best feature. It clearly separates  $\omega_1, \omega_2, \omega_3$  and  $\{\omega_4, \omega_5\}$
    - $x_2$  and  $x_3$  have similar performance, separating classes in three groups
    - $x_4$  is the worst feature since it can only separate  $\omega_4$  from  $\omega_5$ , the rest of the classes having a heavy overlap
    - **Question: What is the best subset of 2 features?**

# Naïve Sequential Feature Selection



- The optimal feature subset turns out to be  $\{x_1, x_4\}$ , because  $x_4$  provides the only information that  $x_1$  needs: discrimination between classes  $\omega_4$  and  $\omega_5$
- However, if we were to choose features according to the individual scores  $J(x_k)$ , we would certainly pick  $x_1$  and either  $x_2$  or  $x_3$ , leaving classes  $\omega_4$  and  $\omega_5$  non separable
  - This naïve strategy fails because it cannot consider features with complementary information



# Statistical Hypothesis Testing



- The Goal: For **each individual feature**, find whether the values, which the feature takes for the different classes, differ significantly. This is based on the values of an appropriately **chosen parameter**. That is, answer

$H_0 : \theta = \theta_0$  :The values do not differ significantly

$H_1 : \theta \neq \theta_0$  :The values differ significantly

- If they do not differ significantly reject feature from subsequent stages.

# Review: Hypothesis Testing Basics



- Let  $x$  be a random variable and the experimental samples,  $x_i = 1, 2, \dots, N$ , are assumed mutually independent

- Case 1: the variance is known**

- Compute the sample mean  $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \Rightarrow E[\bar{x}] = \frac{1}{N} \sum_{i=1}^N E[x_i] = \mu$

- Compute the variance

$$\begin{aligned} E[(\bar{x} - \mu)^2] &= E\left[\left(\frac{1}{N} \sum_{i=1}^N x_i - \mu\right)^2\right] \Rightarrow \sigma_{\bar{x}}^2 = \frac{1}{N} \sigma_x^2 \\ &= \frac{1}{N^2} \sum_{i=1}^N E[(x_i - \mu)^2] + \frac{1}{N^2} \sum_i \sum_j E[(x_i - \mu)(x_j - \mu)] \end{aligned}$$

- Test Statistic:** define the variable

$$q = \frac{\bar{x} - \hat{\mu}}{\sigma / \sqrt{N}}$$

- Hypothesis test  $H_1: E[x] \neq \hat{\mu}$   
 $H_0: E[x] = \hat{\mu}$

# Review: Hypothesis Testing Basics

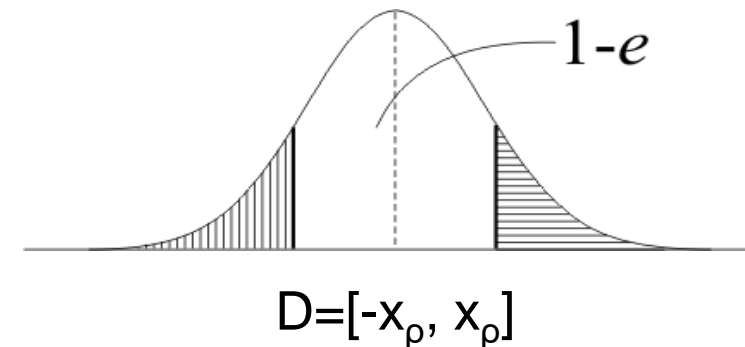


- Central limit theorem under  $H_0$   $p_{\bar{x}}(\bar{x}) = \frac{\sqrt{N}}{\sqrt{2\pi}\sigma} \exp\left(-\frac{N(\bar{x} - \hat{\mu})^2}{2\sigma^2}\right)$
- Thus, under  $H_0$   $p_q(q) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{q^2}{2}\right)$ ,  $q \approx N(0,1)$
- The decision steps:
  - Compute  $q$  from  $x_i$ ,  $i=1,2,\dots,N$
  - Choose significance level  $\rho$
  - Compute from  $N(0,1)$  tables  $D=[-x_\rho, x_\rho]$

$$q = \frac{\bar{x} - \hat{\mu}}{\sigma / \sqrt{N}}$$

if  $q \in D$  accept  $H_0$

if  $q \in \bar{D}$  reject  $H_0$



# Review: Hypothesis Testing Basics



- Let  $x$  be a random variable and the experimental samples,  $x_i = 1, 2, \dots, N$ , are assumed mutually independent

- Case 2: the unknown variance**

- Compute the sample mean  $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \Rightarrow E[\bar{x}] = \frac{1}{N} \sum_{i=1}^N E[x_i] = \mu$

- Compute the sample variance

$$\hat{\sigma}^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2 \Rightarrow E[\hat{\sigma}^2] = \sigma^2$$

- Test Statistic:** define the variable

$$q = \frac{\bar{x} - \mu}{\hat{\sigma} / \sqrt{N}}$$

- Hypothesis test  $H_1 : E[x] \neq \hat{\mu}$   
 $H_0 : E[x] = \hat{\mu}$

# Review: Hypothesis Testing Basics



- This is no longer Gaussian. If  $x$  is Gaussian, then  $q$  follows a t-distribution, with  $N-1$  degrees of freedom
- The decision steps:
  - Compute  $q$  from  $x_i, i=1,2,\dots,N$
  - Choose significance level  $\rho$
  - Compute from t-distribution table  $D=[-x_\rho, x_\rho]$

$$q = \frac{\bar{x} - \mu}{\hat{\sigma} / \sqrt{N}}$$

if  $q \in D$  accept  $H_0$

if  $q \in \bar{D}$  reject  $H_0$

# Application to Feature Selection



- The goal here is to test against zero the difference  $\mu_1 - \mu_2$  of the respective mean values of a single feature in  $\omega_1, \omega_2$ .
- Let  $x_i$   $i=1, \dots, N$ , the values of a feature in  $\omega_1$
- Let  $y_i$   $i=1, \dots, N$ , the values of the same feature in  $\omega_2$
- Assume in both classes (unknown variance or not)  $\sigma_1^2 = \sigma_2^2 = \sigma^2$
- The test becomes
$$H_0 : \Delta\mu = \mu_1 - \mu_2 = 0$$
$$H_1 : \Delta\mu \neq 0$$



# Application to Feature Selection



- Define variable  $z=x-y$
- Obviously,  $E[z]=\mu_1-\mu_2$
- Define the average  $\bar{z} = \frac{1}{N} \sum_{i=1}^N (x_i - y_i) = \bar{x} - \bar{y}$

- **Known Variance Case:**

$$q = \frac{(\bar{x} - \bar{y}) - (\hat{\mu}_1 - \hat{\mu}_2)}{\sigma \sqrt{\frac{2}{N}}}$$

- This is  $N(0,1)$  and one follows the procedure as before.

- **Unknown Variance Case:**

$$q = \frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{S_z \sqrt{\frac{2}{N}}}$$

$$S_z^2 = \frac{1}{2N-2} \left( \sum_{i=1}^N (x_i - \bar{x})^2 + \sum_{i=1}^N (y_i - \bar{y})^2 \right)$$

- $q$  is **t-distribution with  $2N-2$  degrees of freedom**, then apply appropriate tables as before.

# Application to Feature Selection



- Example: The values of a feature in two classes are:

$\omega_1$ : 3.5, 3.7, 3.9, 4.1, 3.4, 3.5, 4.1, 3.8, 3.6, 3.7

$\omega_2$ : 3.2, 3.6, 3.1, 3.4, 3.0, 3.4, 2.8, 3.1, 3.3, 3.6

- Test if the mean values in the two classes **differ significantly**, at the significance level  $\rho=0.05$

- We have

$$\omega_1 : \bar{x} = 3.73, \hat{\sigma}_1^2 = 0.0601$$

$$\omega_2 : \bar{y} = 3.25, \hat{\sigma}_2^2 = 0.0672$$

- For  $N=10$

$$S_z^2 = \frac{1}{2}(\hat{\sigma}_1^2 + \hat{\sigma}_2^2)$$

$$q = \frac{(\bar{x} - \bar{y}) - 0}{S_z \sqrt{\frac{2}{10}}}$$

$$q = 4.25$$

$$H_0 : \Delta\mu = \mu_1 - \mu_2 = 0$$

$$H_1 : \Delta\mu \neq 0$$

- From the table of the **t-distribution** with  $2N-2=18$  degrees of freedom and  $\rho=0.05$ , we obtain  $D=[-2.10, 2.10]$  and since  $q=4.25$  is outside  $D$ ,  $H_0$  is rejected.
- i.e.  $H_1$  is accepted and the feature is selected.

# Class Separability Measures



- The emphasis, so far, was on **individually** considered features. However, such an approach cannot take into account existing **correlations** among the features.
  - That is, two features may be rich in information, but if they are highly correlated we need not consider both of them.
- To this end, in order to search for possible correlations, we consider features **jointly** as elements of vectors.
  - Discard poor in information features, by means of a statistical test.
  - Choose the maximum number,  $m$ , of features to be used. This is dictated by the specific problem (e.g., the number,  $N$ , of available training patterns and the type of the classifier to be adopted).

# Class Separability Measures



- Combine remaining features to search for the “best” combination. To this end:
  - Use different feature combinations to form the feature vector. Train the classifier, and choose the combination resulting in the best classifier performance.
  - A major disadvantage of this approach is the high complexity.
  - Also, local minima, can give misleading results.
- Adopt a class separability measure and choose the best feature combination against this cost.

# Class Separability Measures- Divergence



- Let  $\underline{x}$  be the current feature combination vector.
- **Divergence:**  
To see the rationale behind this cost, consider the two – class case.
- Obviously, if on the average the value of  $\ln \frac{p(\underline{x} | \omega_1)}{p(\underline{x} | \omega_2)}$  is close to zero, then  $\underline{x}$  should be a poor feature combination.
- Define:
$$D_{12} = \int_{-\infty}^{+\infty} p(\underline{x} | \omega_1) \ln \frac{p(\underline{x} | \omega_1)}{p(\underline{x} | \omega_2)} d\underline{x}$$
$$D_{21} = \int_{-\infty}^{+\infty} p(\underline{x} | \omega_2) \ln \frac{p(\underline{x} | \omega_2)}{p(\underline{x} | \omega_1)} d\underline{x}$$
$$d_{12} = D_{12} + D_{21}$$
- $d_{12}$  is known as the divergence and can be used as a class separability measure.

# Class Separability Measures- Divergence



- For the multi-class case, define  $d_{ij}$  for every pair of classes  $w_i, w_j$  and the average divergence is defined as

$$d = \sum_{i=1}^M \sum_{j=1}^M P(\omega_i)P(\omega_j)d_{ij}$$

- Some properties:

$$d_{ij} \geq 0$$

$$d_{ij} = 0, \text{ if } i = j$$

$$d_{ij} = d_{ji}$$

- Large values of  $d$  are indicative of good feature combination.

# Class Separability Measures- Scatter Matrix



- **Scatter Matrices.** These are used as a measure of the way data are scattered in the respective feature space.

- Within-class scatter matrix

We want this to be small

$$S_i = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

$$S_1 + S_2 = S_w$$

- Between-class scatter matrix

We want this to be big

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = (w^T \mu_1 - w^T \mu_2)^2 = w^T \underbrace{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}_{S_B} w = w^T S_B w$$

- Mixture scatter matrix (total scatter matrix)

$$S_m = S_w + S_b$$

We want this to be ??

# Class Separability Measures- Scatter Matrix



- Measures based on Scatter Matrices.

$$J_1 = \frac{\text{Trace}\{S_m\}}{\text{Trace}\{S_w\}}$$

$$J_2 = \frac{|S_m|}{|S_w|} = |S_w^{-1} S_m|$$

$$J_3 = \text{Trace}\{S_w^{-1} S_m\}$$

- Other criteria are also possible, by using various combinations of  $S_m$ ,  $S_b$ ,  $S_w$ .
- The above  $J_1$ ,  $J_2$ ,  $J_3$  criteria take high values for the cases where:
  - Data are clustered together within each class.
  - The mean values for the various classes are far.





# Class Separability Measures- Scatter Matrix

- **Fisher's discriminant ratio**. In one dimension and for two equiprobable classes the determinants become:

$$|S_w| \propto \sigma_1^2 + \sigma_2^2$$

Within-class

$$|S_b| \propto (\mu_1 - \mu_2)^2$$

Between-class

and

$$\frac{|S_b|}{|S_w|} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

known as **Fischer's ratio**.

# Sequential Forward Selection (SFS)



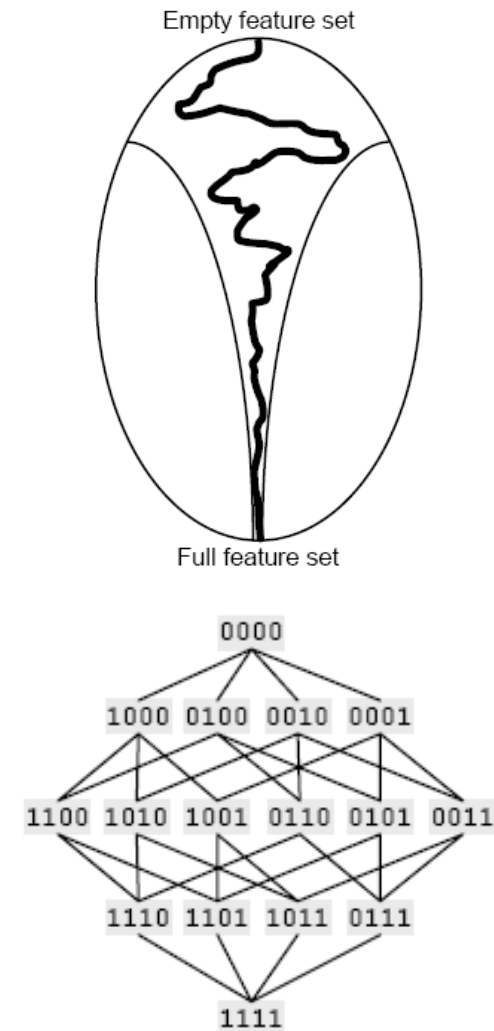
- **Sequential Forward Selection is the simplest greedy search algorithm**
  - Starting from the **empty** set, sequentially **add** the feature  $x^+$  that results in the highest objective function  $J(Y_k + x^+)$  when combined with the features  $Y_k$  that have already been selected
- **Algorithm**

1. Start with the empty set  $Y_0 = \{\emptyset\}$
2. Select the next best feature  $x^+ = \operatorname{argmax}_{x \notin Y_k} [J(Y_k + x)]$
3. Update  $Y_{k+1} = Y_k + x^+$ ;  $k = k + 1$
4. Go to 2

# Sequential Forward Selection (SFS)



- SFS performs best when the optimal subset has a small number of features
  - When the search is near the empty set, a large number of states can be potentially evaluated
  - Towards the full set, the region examined by SFS is narrower since most of the features have already been selected
- The search space is drawn like an ellipse to emphasize the fact that there are fewer states towards the full or empty sets
  - As an example, the state space for 4 features is shown. Notice that the number of states is larger in the middle of the search tree
  - The main disadvantage of SFS is that it is unable to remove features that become obsolete after the addition of other features



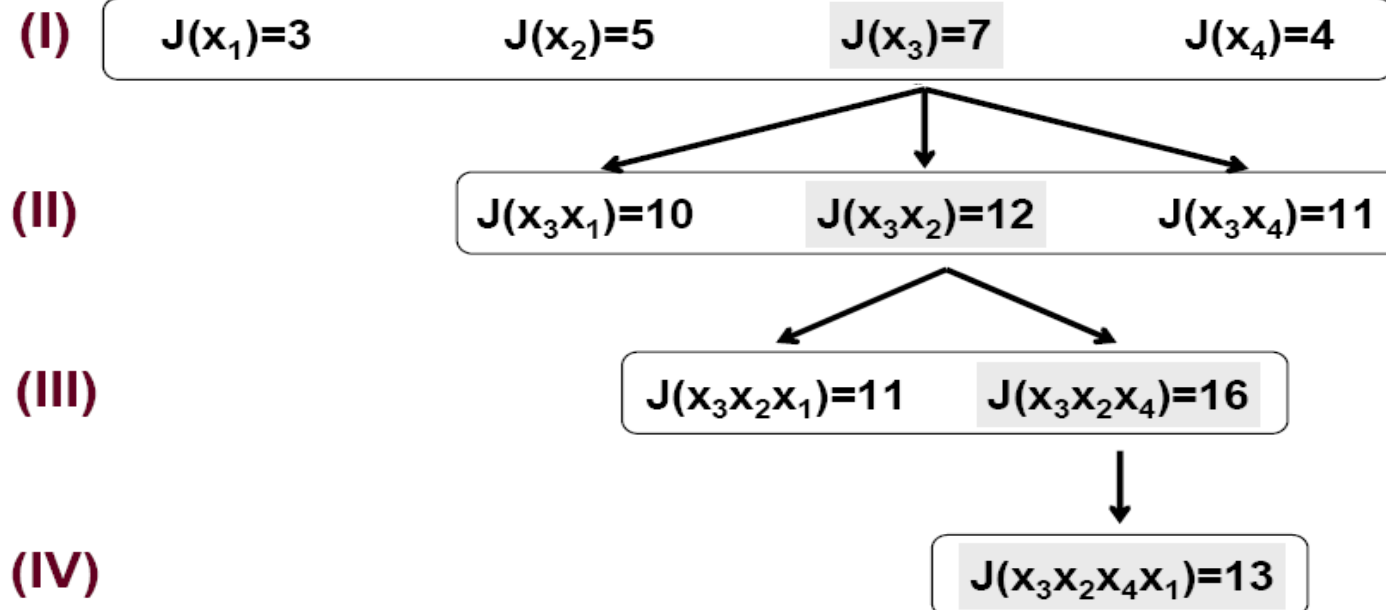
# SFS example



- Assuming the **objective function  $J(X)$**  below, perform a Sequential Forward Selection to completion

$$J(X) = -2x_1x_2 + 3x_1 + 5x_2 - 2x_1x_2x_3 + 7x_3 + 4x_4 - 2x_1x_2x_3x_4$$

- where  $x_k$  are *indicator variables* that determine if the  $k$ -th feature has been selected ( $x_k=1$ ) or not ( $x_k=0$ )
- Solution**



# Sequential Backward Selection (SBS)



- **Sequential Backward Selection works in the opposite direction of SFS**
  - Starting from the **full** set, sequentially **remove** the feature  $x^-$  that results in the smallest decrease in the value of the objective function  $J(Y-x^-)$
  - Notice that removal of a feature may actually lead to an increase in the objective function  $J(Y_k-x^-) > J(Y_k)$ . Such functions are said to be *non-monotonic*
- **Algorithm**

1. Start with the full set  $Y_0 = X$
2. Remove the worst feature  $x^- = \operatorname{argmax}_{x \in Y_k} [J(Y_k - x)]$
3. Update  $Y_{k+1} = Y_k - x^-$ ;  $k = k+1$
4. Go to 2

# Sequential Backward Selection (SBS)



- **Notes**

- SBS works best when the optimal feature subset has a large number of features, since SBS spends most of its time visiting large subsets
- The main limitation of SBS is its inability to reevaluate the usefulness of a feature after it has been discarded



# Plus-L Minus-R Selection (LRS)



- **Plus-L Minus-R is a generalization of SFS and SBS**

- If  $L > R$ , LRS starts from the empty set and repeatedly adds 'L' features and removes 'R' features
- If  $L < R$ , LRS starts from the full set and repeatedly removes 'R' features followed by 'L' feature additions

- **Algorithm**

```
1. If  $L > R$  then
    start with the empty set  $Y = \{\emptyset\}$ 
  else
    start with the full set  $Y = X$ 
    go to step 3
2. Repeat L times
     $x^+ = \operatorname{argmax}_{x \in Y_k} [J(Y_k + x)]$ 
     $Y_{k+1} = Y_k + x^+; \quad k = k + 1$ 
3. Repeat R times
     $x^- = \operatorname{argmax}_{x \in Y_k} [J(Y_k - x)]$ 
     $Y_{k+1} = Y_k - x^-; \quad k = k + 1$ 
4. Go to 2
```

# Plus-L Minus-R Selection (LRS)



- **Notes**

- LRS attempts to compensate for the weaknesses of SFS and SBS with some **backtracking** capabilities
- Its main limitation is the lack of a theory to help predict the optimal values of L and R





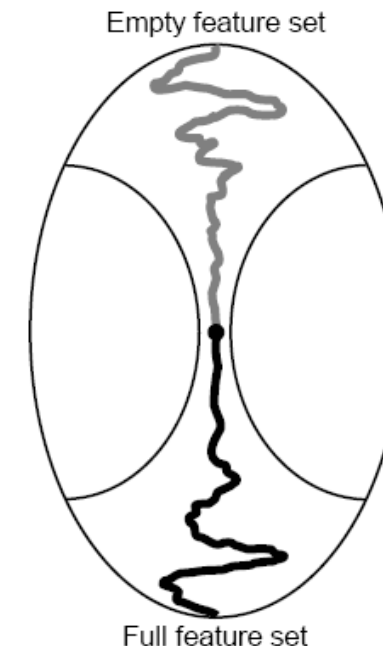
# Bidirectional Search (BDS)



- **Bidirectional Search is a parallel implementation of SFS and SBS**
  - SFS is performed from the empty set, SBS is performed from the full set
  - To guarantee that SFS and SBS converge to the same solution, we must ensure that
    - Features already selected by SFS are not removed by SBS
    - Features already removed by SBS are not selected by SFS
    - For example, before SFS attempts to add a new feature, it checks if it has been removed by SBS, and, if it has, attempts to add the second best feature, and so on. SBS operates in a similar fashion.

- **Algorithm**

1. Start SFS with the empty set  $Y_F = \{\emptyset\}$
2. Start SBS with the full set  $Y_B = X$
3. Select the best feature
$$x^+ = \underset{\substack{x \in Y_{F_k} \\ x \in Y_{B_k}}}{\operatorname{argmax}} [J(Y_{F_k} + x)]$$
$$Y_{F_{k+1}} = Y_{F_k} + x^+$$
3. Remove the worst feature
$$x^- = \underset{\substack{x \in Y_{B_k} \\ x \notin Y_{F_{k+1}}}}{\operatorname{argmax}} [J(Y_{B_k} - x)]$$
$$Y_{B_{k+1}} = Y_{B_k} - x^-; \quad k = k + 1$$
4. Go to 2



# Sequential Floating Selection



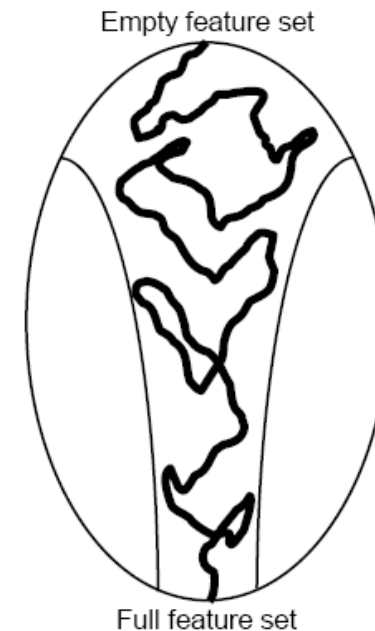
- **Sequential Floating Selection methods** are an extension to the LRS algorithms with **flexible backtracking capabilities**
  - Rather than fixing the values of 'L' and 'R', these *floating* methods **allow those values to be determined from the data**:
    - The dimensionality of the subset during the search can be through to be “floating” up and down
- **There are two floating methods**
  - **Sequential Floating Forward Selection** (SFFS) starts from the empty set
    - After each forward step, SFFS performs backward steps as long as the **objective function increases**
  - **Sequential Floating Backward Selection** (SFBS) starts from the full set
    - After each backward step, SFBS performs forward steps as long as the **objective function increases**

# Sequential Floating (Forward/Backward) Selection (SFFS and SFBS)



- SFFS Algorithm (SFBS is analogous)

1. Start with the empty set  $Y = \{\emptyset\}$
2. Select the best feature
$$x^+ = \operatorname{argmax}_{x \in Y_k} [J(Y_k + x)]$$
$$Y_k = Y_k + x^+; \quad k = k + 1$$
3. Select the worst feature\*
$$x^- = \operatorname{argmax}_{x \in Y_k} [J(Y_k - x)]$$
4. If  $J(Y_k - x^-) > J(Y_k)$  then
$$Y_{k+1} = Y_k - x^-; \quad k = k + 1$$
go to Step 3  
else  
go to Step 2



*\*Notice that you'll need to do some book-keeping to avoid infinite loops*

# Feature selection in Scikit-learn



- Removing features with low variance
  - from `sklearn.feature_selection` import `VarianceThreshold`
- Univariate feature selection
  - `SelectKBest` removes all but the highest scoring features
  - `SelectPercentile` removes all but a user-specified highest scoring percentage of features
  - using common univariate statistical tests for each feature: false positive rate `SelectFpr`, false discovery rate `SelectFdr`, or family wise error `SelectFwe`.
  - `GenericUnivariateSelect` allows to perform univariate feature selection with a configurable strategy. This allows to select the best univariate selection strategy with hyper-parameter search estimator.

# Feature selection in Scikit-learn



- Recursive feature elimination
  - select features by recursively considering smaller and smaller sets of features recursively repeated **on the pruned set** until the desired number of features to select is eventually reached.
- Feature selection using **SelectFromModel**
  - **L1-based feature selection**
- Tree-based feature selection
  - Tree-based estimators (see the **sklearn.tree** module and forest of trees in the **sklearn.ensemble** module)
- Feature selection as part of a pipeline
  - use a **sklearn.pipeline.Pipeline**

```
clf = Pipeline([
    ('feature_selection', SelectFromModel(LinearSVC(penalty="l1"))),
    ('classification', RandomForestClassifier())
])
clf.fit(X, y)
```

# Homework

---



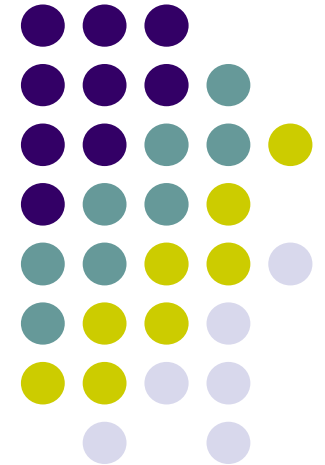
- Homework 2 will due on March 29 by 5:00 PM.
- The dataset will be available on BB. Perform feature engineering on the given dataset.
- Please submit your hw to Black Board and bring a copy of your hw to class (if possible)
- (1) Perform feature selection on the data using various algorithms in scikit learn
- (2) Perform PCA on the data, and determine how many components to choose if we want to maintain 90% of data variation.
- Extra credit: compare the performance with the full feature set and with the selected sub-set of features.

The Catholic University of America

# Introduction to Data Mining



## Feature Extraction

Instructor  
Lin-Ching Chang



# Feature Extraction Techniques



- **Principal Components Analysis (PCA)**  *Unsupervised*
  - The curse of dimensionality
  - Dimensionality reduction
  - Feature selection vs. feature extraction
  - Signal representation vs. signal classification
  - Principal Components Analysis
- **(Fisher's) Linear Discriminant Analysis (LDA) (will not cover)**  *Supervised*
  - Linear Discriminant Analysis
  - Fisher's Linear Discriminant Analysis, two classes
  - Fisher's Linear Discriminant Analysis, C classes
  - Fisher's LDA vs. PCA example
  - Limitations of LDA
  - Variants of LDA
  - Other dimensionality reduction methods



# Why Feature (Factor) Analysis?

---



- We study phenomena that can not be directly observed
  - ego, personality, intelligence in psychology
  - Underlying factors that govern the observed data
- We want to identify and operate with underlying latent factors rather than the observed data
  - E.g. topics in news articles
  - Transcription factors in genomics
- We want to discover and exploit hidden relationships
  - “beautiful car” and “gorgeous automobile” are closely related
  - So are “driver” and “automobile”
  - But does your search engine know this?
  - Reduces noise and error in results

# Why Feature (Factor) Analysis?



- We have too many observations and dimensions
  - To reason about or obtain insights from
  - To visualize
  - Too much noise in the data
  - Need to “reduce” them to a smaller set of factors
  - Better representation of data without losing much information
  - Can build more effective data analyses on the reduced-dimensional space: classification, clustering, pattern recognition
- **Combinations of observed variables** may be more effective bases for insights, even if physical meaning is obscure

# Basic Concept



- Areas of variance in data are where items can be **best discriminated** and key underlying phenomena observed
  - Areas of greatest “signal” in the data
- If two items or dimensions are highly correlated or dependent
  - They are likely to represent highly related phenomena
  - If they tell us about the same underlying variance in the data, combining them to form a single measure is reasonable
- So we want to **combine related variables**, and **focus on uncorrelated or independent ones**, especially those along which the observations have high variance
  - We want a **smaller** set of variables that **explain most** of the variance in the original data, in more compact and insightful form

# Basic Concept



- What if the dependences and correlations are not so strong or direct?
  - And suppose you have 3 variables, or 4, or 5, or 10000?
- Look for the phenomena underlying the observed covariance/ codependence in a set of variables
  - Once again, phenomena that are uncorrelated or independent, and especially those along which the data show high variance
- These phenomena are called “**factors**”, “**features**”, or “**principal components**” or “**independent components**,” depending on the methods used
  - Principal Component Analysis (PCA): based on **variance/covariance/correlation**
  - Independent Component Analysis (ICA): based on **independence**

# *The Curse of Dimensionality*

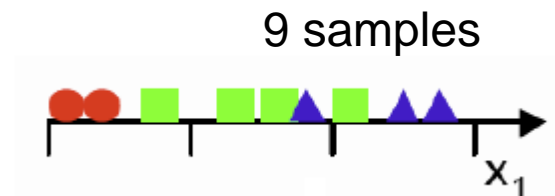


- **The curse of dimensionality is the problem caused by the exponential increase in volume associated with adding extra dimensions to a (mathematical) space.**
- **The curse of dimensionality**
  - A term coined by Bellman in 1961
  - Refers to the problems associated with multivariate data analysis as the dimensionality increases
  - We will illustrate these problems with a simple example

# The Curse of Dimensionality - Example



- **Consider a 3-class pattern recognition problem**
  - A simple approach would be to
    - Divide the feature space into uniform bins
    - Compute the ratio of examples for each class at each bin and,
    - For a new example, find its bin and choose the predominant class in that bin
- In our toy problem we decide to start with one single feature and divide the real line into 3 segments



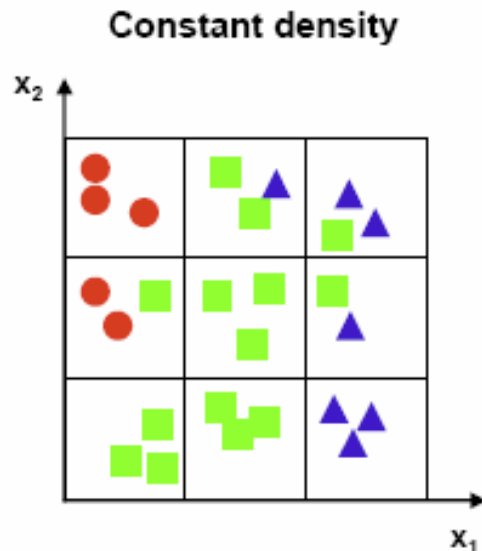
- After doing this, we notice that there exists too much overlap among the classes, so we decide to incorporate a second feature to try and improve separability.

# The Curse of Dimensionality - Example

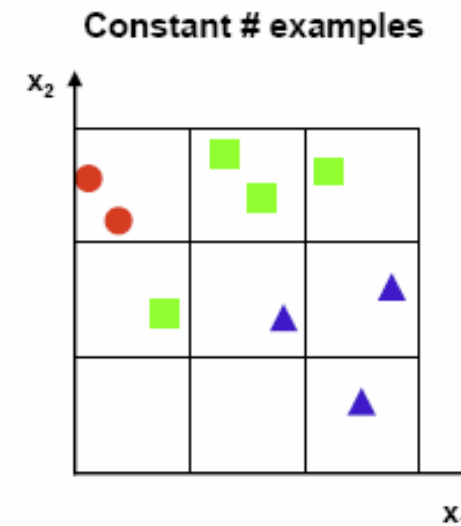


- We decide to preserve the granularity of each axis, which raises the number of bins from 3 (in 1D) to  $3^2=9$  (in 2D)
  - At this point we need to make a decision: do we maintain the **density of examples per bin** or do we keep **the number of examples** had for the one-dimensional case?

Choosing to **maintain the density** increases the number of examples from 9 (in 1D) to 27 (in 2D)



Choosing to **maintain the number of examples** results in a 2D scatter plot that is **very sparse**



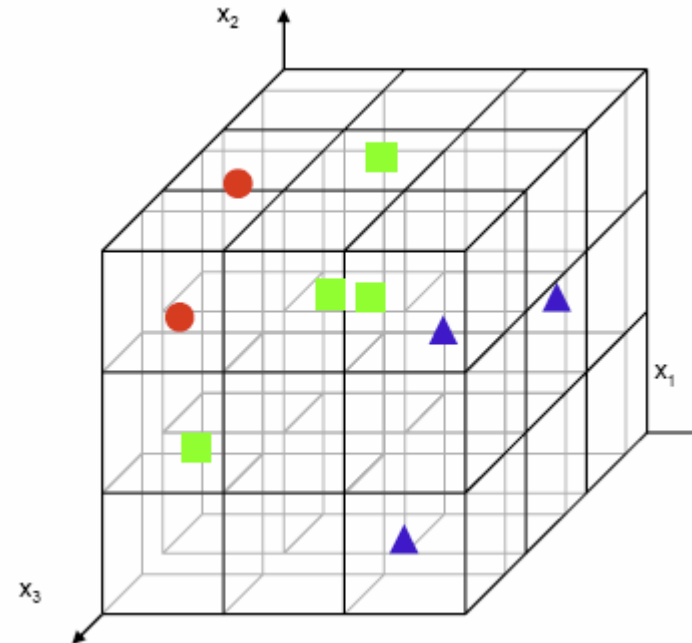
# The Curse of Dimensionality - Example



- **Moving to three features makes the problem worse:**

- The number of bins grows to  $3^3=27$

- For the same density of examples the number of needed examples becomes 81 (to maintain density)
- For the same number of examples, well, the 3D scatter plot is almost empty (become more sparse)





# *The Curse of Dimensionality*

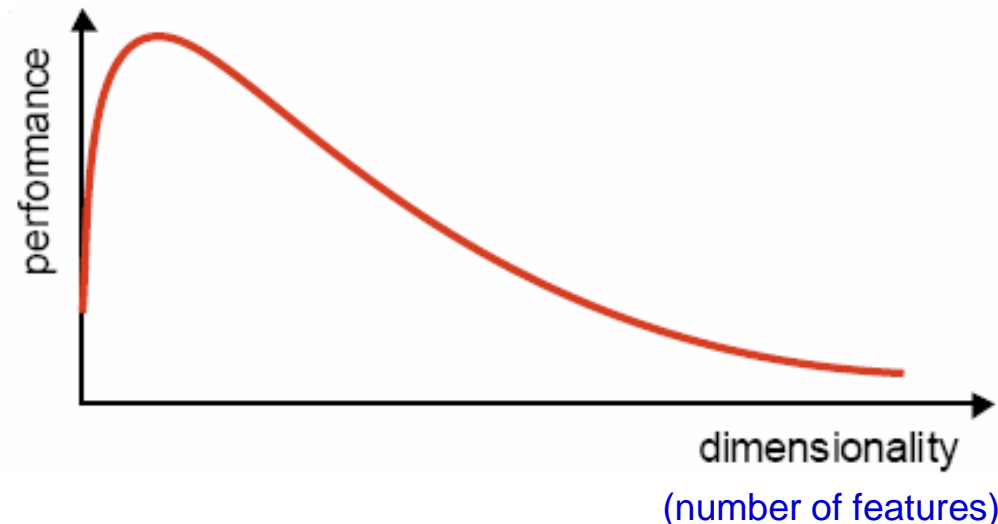


- Obviously, our approach to divide the sample space into equally spaced bins was quite inefficient
  - There are other approaches that are much less susceptible to the curse of dimensionality, **but the problem still exists**
- How do we beat **the curse of dimensionality**?
  - By reducing the dimensionality
  - By incorporating prior knowledge
  - By providing increasing smoothness of the target function

# The Curse of Dimensionality



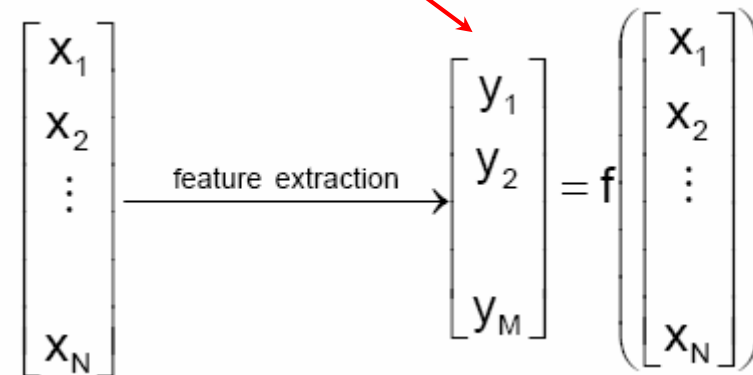
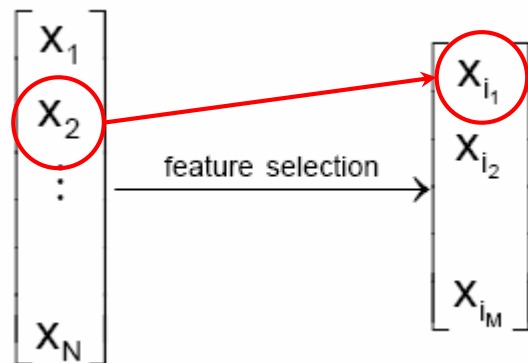
- In practice, the curse of dimensionality means that, for a **given sample size**, there is a **maximum number of features** above which the performance of our classifier will degrade rather than improve
  - In most cases, the additional information that is lost by discarding some features is (more than) compensated by a more accurate mapping in the lower dimensional space



# Dimensionality Reduction



- **Two approaches are available to perform dimensionality reduction**
  - **Feature selection**: choosing a **subset** of all the features (the ones more informative)
  - **Feature extraction**: creating a subset of **new** features by combinations of the existing features



# Dimensionality Reduction: Feature Extraction



- The problem of **feature extraction** can be stated as
  - Given a feature space  $\mathbf{x}_i \in \mathbf{R}^N$  find a mapping  $\mathbf{y} = \mathbf{f}(\mathbf{x}): \mathbf{R}^N \rightarrow \mathbf{R}^M$  with  $M < N$  such that the **transformed feature vector**  $\mathbf{y}_i \in \mathbf{R}^M$  preserves (most of) the information or structure in  $\mathbf{R}^N$ .
- How do you know such transformation is better?
  - An **optimal** mapping  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  will be one that results in **no increase in the minimum probability of error**
    - This is, a Bayes decision rule applied to the initial space  $\mathbf{R}^N$  and to the reduced space  $\mathbf{R}^M$  yield the same classification rate

# Feature Extraction



- In general, the **optimal** mapping  $y=f(x)$  will be a **non-linear** function
  - However, there is no systematic way to generate non-linear transforms
    - The selection of a particular subset of transforms is problem dependent
- For this reason, feature extraction is commonly limited to **linear transforms**:  $y=Wx$ 
  - This is,  $y$  is a linear projection of  $x$
  - NOTE: When the mapping is a non-linear function, the reduced space is called a **manifold**

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & \cdots \\ W_{21} & W_{22} & \cdots \\ \vdots & \vdots & \ddots \\ W_{M1} & W_{M2} & \end{bmatrix} \begin{bmatrix} W_{1N} \\ W_{2N} \\ \vdots \\ W_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

# Principal Components Analysis

---

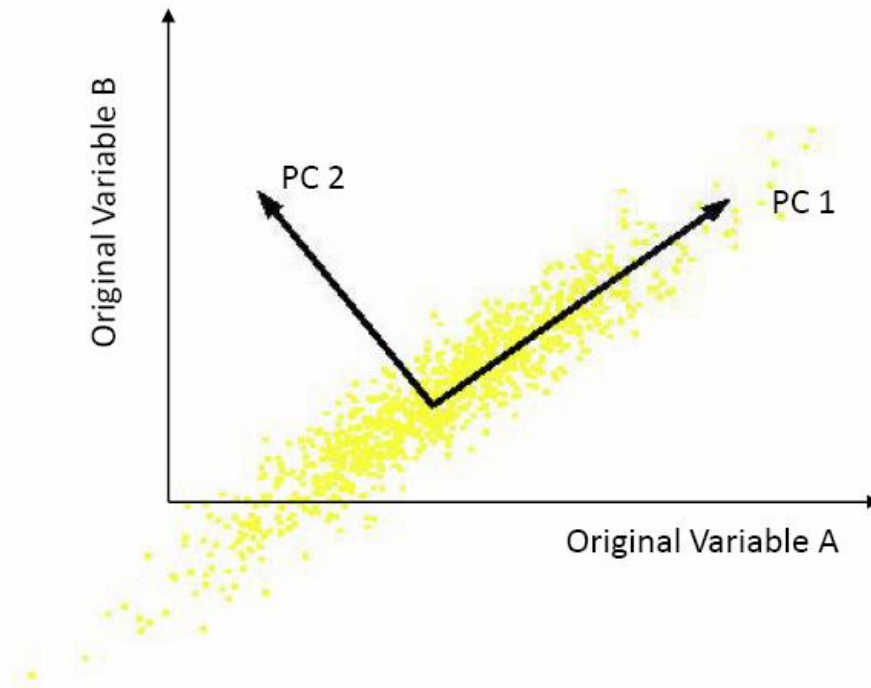


- PCA is the most common form of factor analysis
- The new variables/dimensions
  - are **linear combinations** of the original ones
  - are **uncorrelated** with one another
    - orthogonal in original dimension space
  - capture as much of the original variance in the data as possible
  - are called Principal Components

# Principal Components Analysis



- What are the new axes?



1. First principal component is the direction of **greatest variability** (covariance) in the data
1. Second is the next orthogonal (uncorrelated) direction of greatest variability

- Orthogonal directions of greatest variance in data
- Projections along PC1 **discriminate** the data most along any one axis

# Principal Components Analysis



- Principle
  - Linear projection method to **reduce** the number of parameters
  - Transfer a set of correlated variables into a new set of **uncorrelated** variables
  - **Map** the data into a space of lower dimensionality
  - Form of **unsupervised learning**
- Properties
  - It can be viewed as **a rotation** of the existing axes to new positions in the space defined by original variables
  - New axes are **orthogonal** and represent the directions with maximum variability



# How Many Principal Components (PCs)?

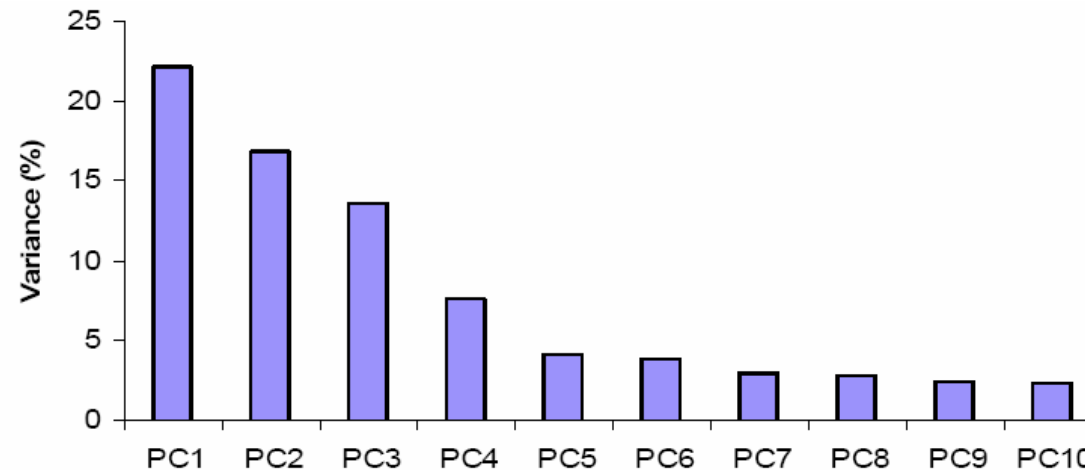


- For  $n$  original dimensions, correlation matrix is  $n$ -by- $n$ , and has up to  $n$  eigenvectors.
  - So the number of Principal Components is  $n$ .
- Where does dimensionality reduction come from?

# Dimensionality Reduction



- Can *ignore* the components of *lesser* significance.



- You do *lose some information*, but if the eigenvalues are small, you don't lose much
  - n dimensions in original data
  - calculate eigenvectors and eigenvalues
  - choose only the first p eigenvectors, based on their eigenvalues
  - final data set has only p dimensions

# PCs, Variance and Least-Squares



- The first PC retains **the greatest amount of variation** in the sample
- The *kth* PC retains the *kth* greatest fraction of the variation in the sample
- The *kth* largest eigenvalue of the correlation matrix  $C$  is the variance in the sample along the *kth* PC
- The least-squares view: PCs are a series of linear least squares fits to a sample, each orthogonal to all previous ones

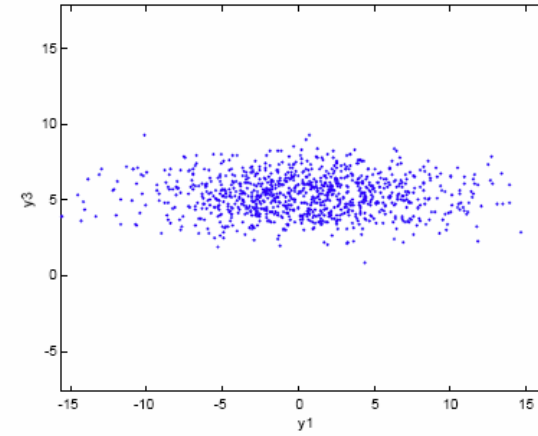
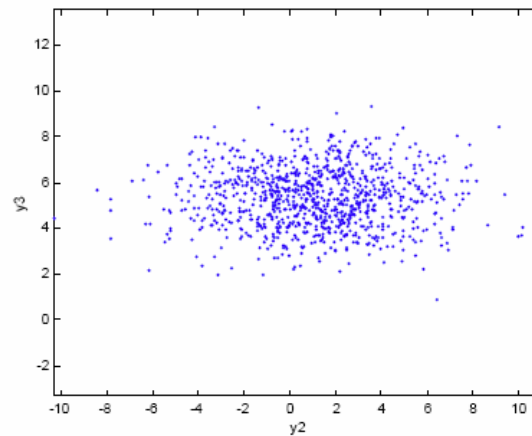
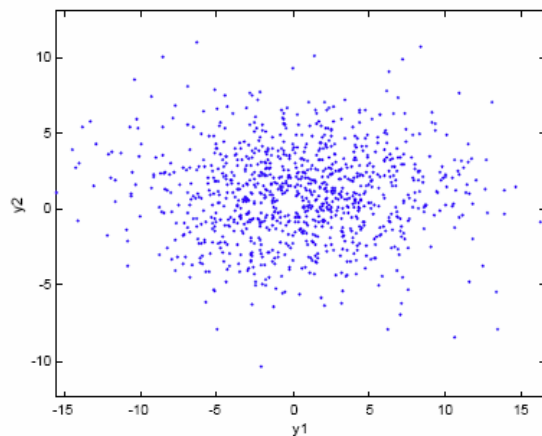
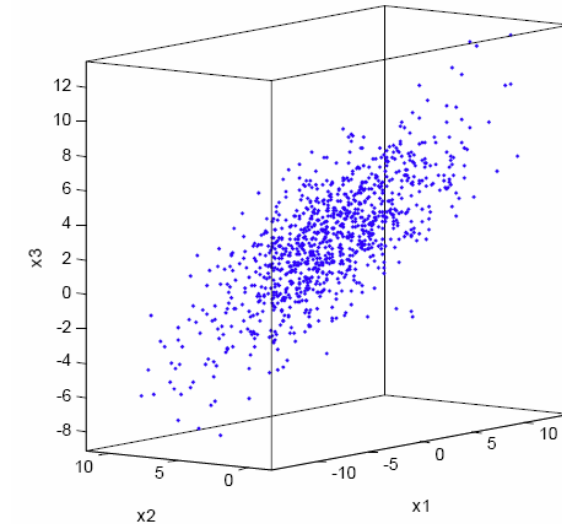
# Principal Components Analysis (PCA) Example



- In this example we have a three-dimensional **Gaussian** distribution with the following Parameters

$$\mu = [0 \ 5 \ 2]^T \text{ and } \Sigma = \begin{bmatrix} 25 & -1 & 7 \\ -1 & 4 & -4 \\ 7 & -4 & 10 \end{bmatrix}$$

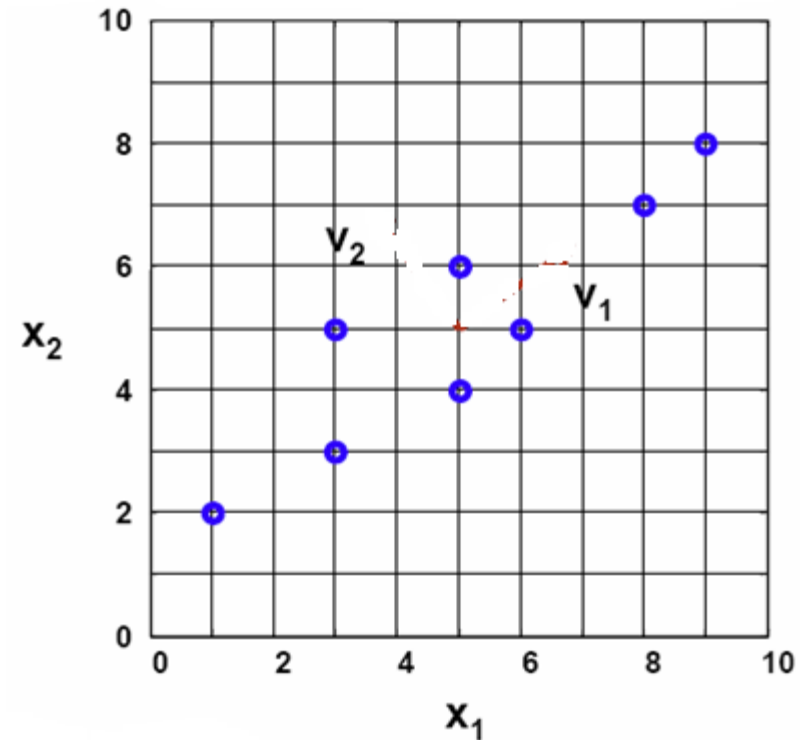
- The three pairs of principal component projections are shown below
  - Notice that the first projection has the largest variance, followed by the second projection
  - Also notice that the PCA projections **de-correlates** the axis



# Principal Components Analysis (PCA) Example



- **Compute the principal components for the following two-dimensional dataset**
  - $X=(x_1, x_2)=\{(1,2), (3,3), (3,5), (5,4), (5,6), (6,5), (8,7), (9,8)\}$
  - Let's first plot the data to get an idea of which solution we should expect
- **SOLUTION (by hand)**
  - Compute the covariance from data
  - Find the eigenvalues and eigenvectors of the covariance matrix
  - Sort the eigenvalues
  - Transform the data
    - For dimensional reduction, take the axis with larger eigenvalue



# Principal Components Analysis (PCA) Example



- $X=(x_1,x_2)=\{(1,2),(3,3),(3,5),(5,4),(5,6),(6,5),(8,7),(9,8)\}$
- **SOLUTION (by hand)**
  - Formula you need

$$\mu = E(X) = \sum x_i p_i \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\text{Variance of } X = \text{Var}(X) = \sigma^2 = \sum (x_i - \mu)^2 p_i$$

$$\text{Standard Deviation of } X = \sigma = \sqrt{\sum (x_i - \mu)^2 p_i}$$

The **covariance** between the random variables  $X$  and  $Y$ , denoted as  $\text{cov}(X, Y)$  or  $\sigma_{XY}$ , is

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = E(XY) - \mu_X \mu_Y \quad (5-26)$$

# Principal Components Analysis (PCA) Example



- $X=(x_1,x_2)=\{(1,2),(3,3),(3,5),(5,4),(5,6),(6,5),(8,7),(9,8)\}$

- **SOLUTION (by hand)**

- Compute the (biased) covariance  $\Sigma_x = \begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix}$

- The eigenvalues are the zeros of the characteristic equation

$$\Sigma_x v = \lambda v \Rightarrow |\Sigma_x - \lambda I| = 0 \Rightarrow \begin{vmatrix} 6.25 - \lambda & 4.25 \\ 4.25 & 3.5 - \lambda \end{vmatrix} = 0 \Rightarrow \lambda_1 = 9.34; \lambda_2 = 0.41;$$

- The eigenvectors are the solutions of the system

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} \lambda_1 v_{11} \\ \lambda_1 v_{12} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.59 \end{bmatrix}$$
$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} \lambda_2 v_{21} \\ \lambda_2 v_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} -0.59 \\ 0.81 \end{bmatrix}$$

# Principal Components

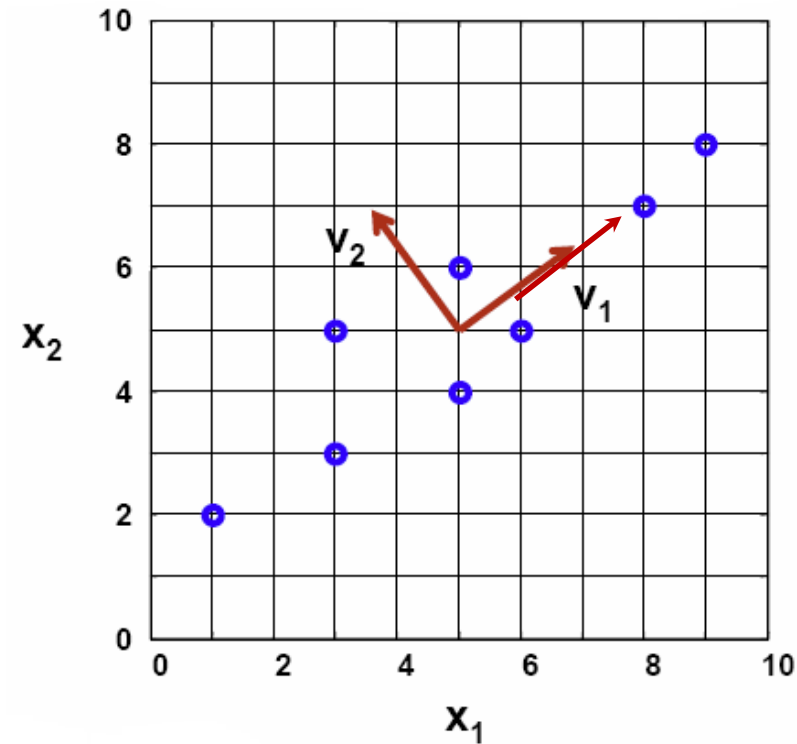


$$\lambda_1 = 9.34; \lambda_2 = 0.41;$$

$$\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.59 \end{bmatrix}$$

$$\begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} -0.59 \\ 0.81 \end{bmatrix}$$

- How much data variation covered if only the first component is used?
- $9.34 / (9.34 + 0.41) = 95.78\%$





# Transform the data



- Reduce the dimension from 2-D to 1-D
  - **two-dimensional dataset**
  - $X=(x_1, x_2)=\{(1,2), (3,3), (3,5), (5,4), (5,6), (6,5), (8,7), (9,8)\}$
- **one-dimensional dataset (try it at home if you like)**

# Limitation of PCA



- The reduction of dimensions for complex distributions may need non-linear processing
  - Curvilinear Component Analysis (CCA)
    - Non linear extension of PCA
- The main limitation of PCA is that **it does not consider class separability** since it does not take into account the class label of the feature vector
- PCA simply performs **a coordinate rotation** that aligns the transformed axes with the directions of maximum variance
- **(Note)** There is **no** guarantee that the directions of maximum variance will contain good features for discrimination

# Limitation of PCA

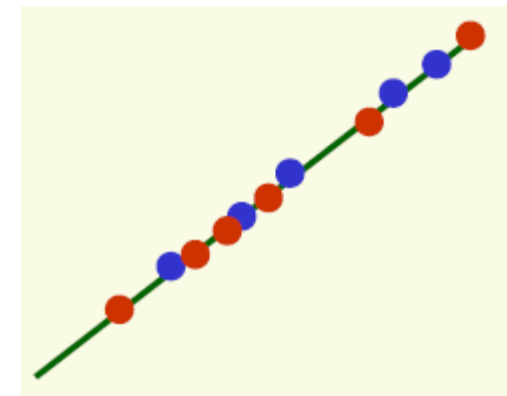
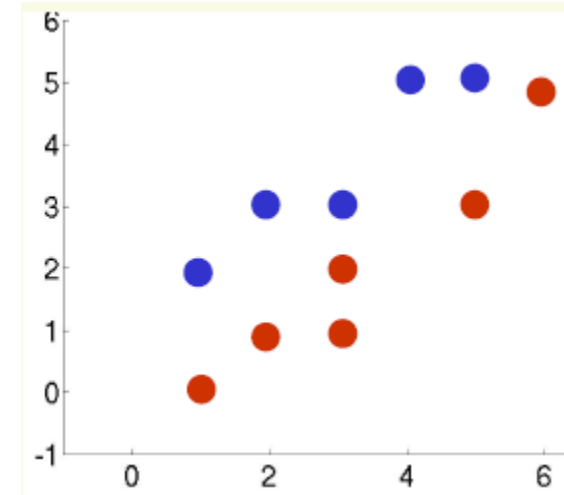


- The main limitation of PCA is that **it does not consider class separability** since it does not take into account the class label of the feature vector
- PCA simply performs **a coordinate rotation** that aligns the transformed axes with the directions of maximum variance
- **(Note)** There is **no** guarantee that the directions of maximum variance will contain good features for discrimination

# Limitation of PCA: Example



- Suppose we know the class label
  - Class 1 = [(1,2),(2,3),(3,3),(4,5),(5,5)]
  - Class 2 = [(1,0),(2,1),(3,1),(3,2),(5,3),(6,5)]
- Plot the data
  - Data can be divided to two classes visually
- Perform PCA on those data
  - PCA does not use the class label
  - PCA performs very poorly on this data
  - because the direction of largest variance is not helpful for classification.



# PCA with Scikit-learn



- `sklearn.decomposition.PCA`
- See also
  - KernelPCA: [Non-linear dimensionality reduction](#) through the use of kernels
  - SparsePCA
  - MiniBatchSparsePCA
  - TruncatedSVD
  - IncrementalPCA

# PCA with Scikit-learn



- Let's use Scikit-learn built-in ***breast\_cancer*** dataset which contains 30 features and 569 observations.
- Choose the right number of dimensions (k)

```
from sklearn.decomposition import PCA

pca_30 = PCA(n_components=30, random_state=2020)
pca_30.fit(X_scaled)
X_pca_30 = pca_30.transform(X_scaled)
```

How many components to choose?

```
pca_30.explained_variance_ratio_ * 100
```

```
array([4.42720256e+01, 1.89711820e+01, 9.39316326e+00, 6.60213492e+00,
       5.49576849e+00, 4.02452204e+00, 2.25073371e+00, 1.58872380e+00,
       1.38964937e+00, 1.16897819e+00, 9.79718988e-01, 8.70537901e-01,
       8.04524987e-01, 5.23365745e-01, 3.13783217e-01, 2.66209337e-01,
       1.97996793e-01, 1.75395945e-01, 1.64925306e-01, 1.03864675e-01,
       9.99096464e-02, 9.14646751e-02, 8.11361259e-02, 6.01833567e-02,
       5.16042379e-02, 2.72587995e-02, 2.30015463e-02, 5.29779290e-03,
       2.49601032e-03, 4.43482743e-04])
```

```
np.cumsum(pca_30.explained_variance_ratio_ * 100)
```

```
array([ 44.27202561,  63.24320765,  72.63637091,  79.23850582,
        84.73427432,  88.75879636,  91.00953007,  92.59825387,
        93.98790324,  95.15688143,  96.13660042,  97.00713832,
        97.81166331,  98.33502905,  98.64881227,  98.91502161,
        99.1130184 ,  99.28841435,  99.45333965,  99.55720433,
        99.65711397,  99.74857865,  99.82971477,  99.88989813,
        99.94150237,  99.96876117,  99.99176271,  99.99706051,
        99.99955652, 100.         ])
```

# PCA with Scikit-learn



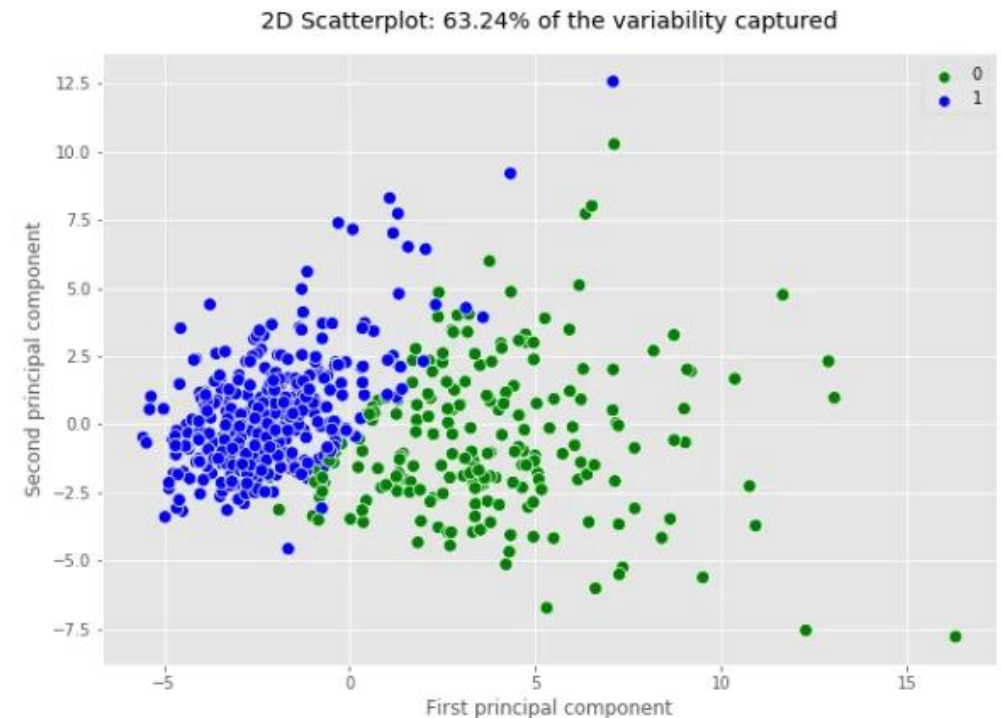
- Choose the right number of dimensions (k)

```
np.cumsum(pca_30.explained_variance_ratio_ * 100)

array([ 44.27202561,  63.24320765,  72.63637091,  79.23850582,
        84.73427432,  88.75879636,  91.00953007,  92.59825387,
        93.98790324,  95.15688143,  96.13660042,  97.00713832,
        97.81166331,  98.33502905,  98.64881227,  98.91502161,
        99.1130184 ,  99.28841435,  99.45333965,  99.55720433,
        99.65711397,  99.74857865,  99.82971477,  99.88989813,
        99.94150237,  99.96876117,  99.99176271,  99.99706051,
        99.99955652, 100.         ])
```

- Apply PCA by setting n\_components=2

```
pca_2 = PCA(n_components=2, random_state=2020)
pca_2.fit(X_scaled)
X_pca_2 = pca_2.transform(X_scaled)
```



# Homework

---



- Homework 2 will due on March 29 by 5:00 PM.
- The dataset will be available on BB. Perform feature engineering on the given dataset.
- Please submit your hw to Black Board and bring a copy of your hw to class (if possible)
- (1) Perform feature selection on the data using various algorithms in scikit learn
- (2) Perform PCA on the data, and determine how many components to choose if we want to maintain 90% of data variation.
- Extra credit: compare the performance with the full feature set and with the selected sub-set of features.





- 
- End of lecture