Feature Engineering

CS576 MACHINE LEARNING

Dr. Jin S. Yoo, Professor
Department of Computer Science
Purdue University Fort Wayne

Reference

- J. D. Kelleher et al., Machine Learning for Predictive Data Analytics, 2nd ed., Ch 3.6 and Ch 5.4.6
- E. Alpaydin, Introduction to Machine Learning, 4th ed., Ch 6.1-6.3
- G. James et al., An Introduction to Statistical Learning, Ch 10.2
- Géron, Hands-On Machine Learning, 3rd ed., Ch 8

Outline

What is Feature Engineering?

- Features are the attributes or characteristics of the data that the model uses to make predictions.
- **Feature engineering** is the process of selecting, transforming, or creating relevant features from raw data to improve the performance and effectiveness of machine learning models.
- Effective feature engineering can significantly enhance a model's ability to learn and make accurate predictions.
- **The goal of feature engineering** is to extract the most relevant information from the data and present it in a format that is suitable for the chosen machine learning algorithm.

Feature Engineering Tasks

- Feature engineering encompasses many tasks, including:
  - Feature Transformation
  - Feature Scaling
  - Feature Encoding
  - Feature Creation
  - Binning/Bucketing
  - Feature Selection
  - Feature Extraction (Dimensionality Reduction)
  - Handling Missing Data
  - and so on.

Feature Transformation

- **Feature Transformation**: Converting the data into a suitable format for the ML algorithm to ensure that the data meets the assumptions of the chosen algorithm.
- Feature transformation is relevant to
  - **feature scaling -** normalizing or standardizing
  - **feature encoding-** encoding categorical variables (e.g., one-hot encoding)
  - **mathematical transformation** (e.g., applying logarithm, square root)
  - **text feature transformation** – tokenization, stemming, and vectorization (e.g., TF-IDF) to convert textual content to numerical features
  - and so on.

Feature Scaling

- Having continuous features that cover very different ranges can cause difficulty for some machine learning algorithm
  - E.g., age in [16, 96] vs salary in [10,000, 100,000]
- **Feature scaling** is a process for ensuring that features are on the same scale to prevent one feature from dominating the learning process.
- Feature scaling is particularly important for algorithms sensitive to scale, such as gradient descent-based methods.
- **Normalization** techniques can be used to change a continuous feature to fall within a specified range while maintaining the relative differences between the values for the feature.

Normalization

- **Range normalization** performs a linear scaling of the ordinal values of the continuous feature to fall within a specific range [*low*, *high*] as follows:

$$a_i' = \frac{a_i - min(a)}{max(a) - min(a)} \times (high - low) + low$$

  - where a_i' is the normalized feature value, a_i is the original value, $\min(a)$ is the minimum value of feature a, $\max(a)$ is the maximum value of a, and low and $high$ are the minimum and maximum values of the desired range.
  - **Typical ranges: [0, 1] and [-1, 1]**

- Range normalization (also known as **min-max scaling**) is quite sensitive the presence of outliers in a dataset.

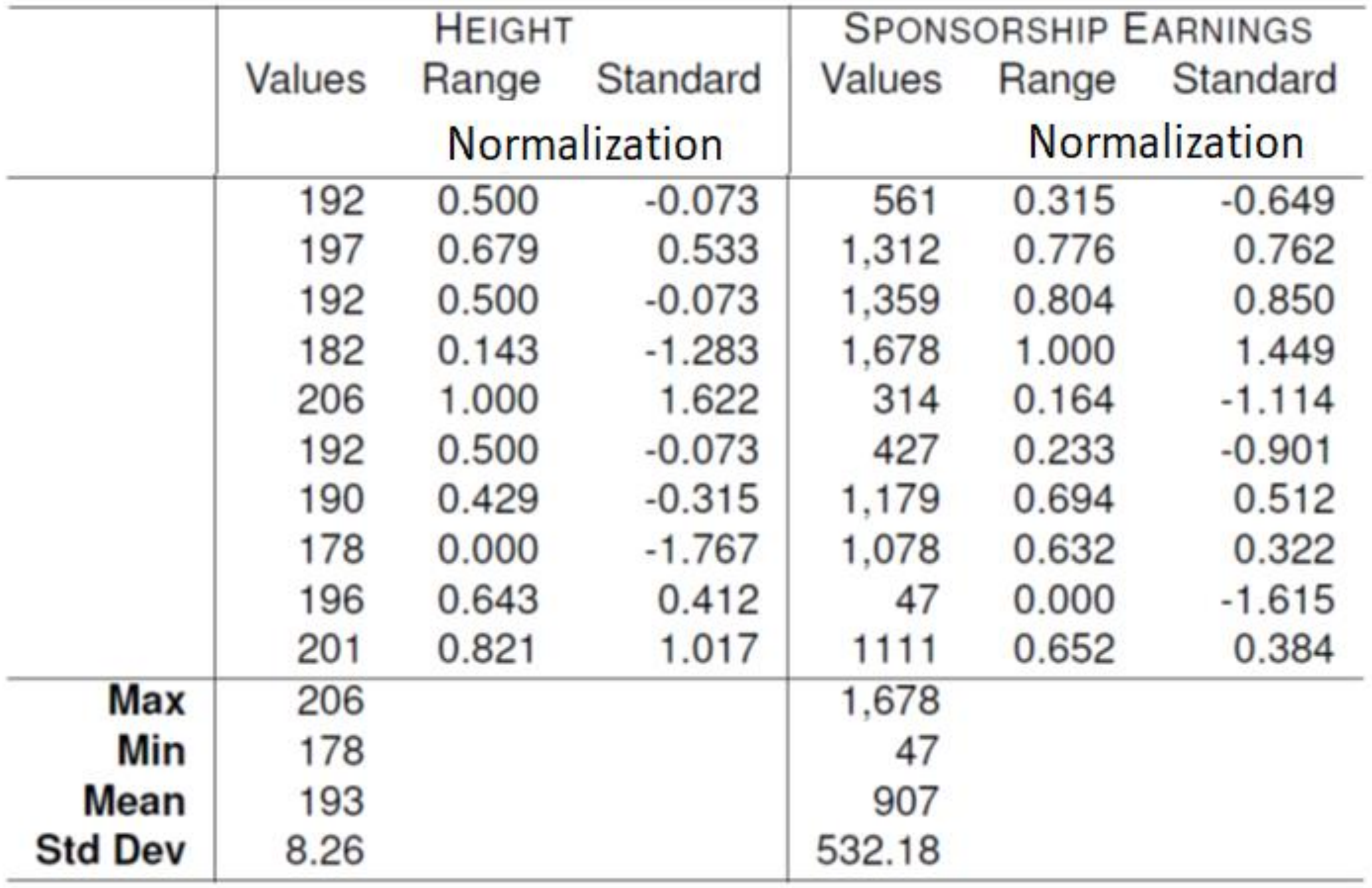
Normalization (Standardization)

- Another way to normalize data is to **standardize** it into **standard scores**.
- A standard score measures **how many standard deviations a feature value is from the mean for that feature.**
- The **standard score** is calculated as follows:

$$a_i' = \frac{a_i - \overline{a}}{sd(a)}$$

  - where a_i' is the normalized feature value, a_i is the original value, $\bar{a}$ is the mean of feature a, and sd(a) is the standard deviation for a
- Standardizing feature values have **a mean of 0 and a standard deviation of 1**
- The standardized value is in **a range of [-1, 1]**
- **Assumption**: Original feature values are normally distributed.

Example



	HEIGHT			SPONSORSHIP EARNINGS		
	Values	Range	Standard	Values	Range	Standard
		Normalization			Normalization	
	192	0.500	-0.073	561	0.315	-0.649
	197	0.679	0.533	1,312	0.776	0.762
	192	0.500	-0.073	1,359	0.804	0.850
	182	0.143	-1.283	1,678	1.000	1.449
	206	1.000	1.622	314	0.164	-1.114
	192	0.500	-0.073	427	0.233	-0.901
	190	0.429	-0.315	1,179	0.694	0.512
	178	0.000	-1.767	1,078	0.632	0.322
	196	0.643	0.412	47	0.000	-1.615
	201	0.821	1.017	1111	0.652	0.384
Max	206			1,678		
Min	178			47		
Mean	193			907		
Std Dev	8.26			532.18		

Table. The result of normalizing a small sample of the HEIGHT and SPONSORSHIP EARNINGS features from the professional basketball squad dataset.

Feature Encoding

- **Feature Encoding** is a process for converting categorical variables into numerical representations that can be used by ML algorithms.
- This includes techniques like **one-hot encoding**, **label encoding**, and **ordinal encoding**.
  - **One-hot encoding** process converts categorical variables into binary vectors.
    - E.g., [1, 0, 0] (Apple), [0, 1, 0] (Banana), [0, 0, 1] (Oracle)
  - **Label encoding** process converts categorical data into numerical values. Each unique label in a categorical feature is assigned a unique integer value.
    - E.g., 0 (Apple), 1 (Banana), 2 (Orange), 1 (Banana)

Mathematical Transformation

- **Logarithm Transformation** is used to compress the range of data values that span a large range into a smaller, more manageable range.
  - The most commonly used logarithms are
    - the **natural logarithm** (base e) : $\boldsymbol{y = \ln(x)}$, $\boldsymbol{y = \log_e x}$
    - the **base 10 logarithm**: $\boldsymbol{y = \log x}$, $\boldsymbol{y = \log_{10} x}$
  - It's particularly useful when dealing with data that has a multiplicative or exponential relationship.
- **Square Root Transformation** is used to compress the range of values that vary across a wide range while giving more emphasis to smaller values.
  - Square root transformation **:** $\boldsymbol{y = \sqrt{x}}$
  - This can help in cases where the data distribution is positively skewed and you want to make it more symmetrical

Feature Creation

- **Feature creation** involves generating new features from the existing ones, often by applying domain knowledge, mathematical transformations, or aggregations, or encoding categorical variables.
- The **goal** of feature creation is to enhance the representation of the data, making it more suitable for a specific task or improving the performance of a machine learning model.
- **Example:** if you have a dataset containing a person's birth date, you could create a new feature representing their age at the time of the observation. This newly created feature might be more relevant for certain analyses or predictions than the original birth date.

Outline

- Introduction
- Feature Transformation
  - Feature Scaling
  - Feature Encoding
  - Feature Creation
- ☞ **Binning/Bucketing**
- Feature Selection
- Feature Extraction (Dimensionality Reduction)
- Summary

Binning

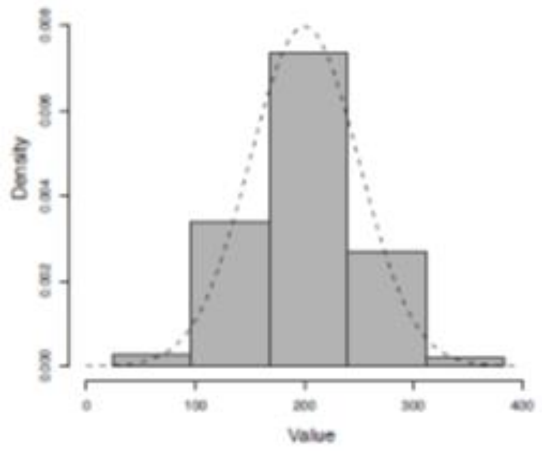
- **Binning** (also known as **bucketing** or **discretization**) involves dividing a continuous feature into a set of discrete **intervals** or **bins**.
- Each **bin** represents a range of values, and data instances falling within that range are assigned to the corresponding bin.
- For certain analysis and models, binning can be used for **dealing with categorical data.**
  - It converts a continuous feature into a categorical feature.
  - Bins corresponds to the levels of the new categorical feature.
- Binning can also be useful when **handling outliers**, **simplifying data**, and **capturing non-linear relationships**.

Binning Techniques

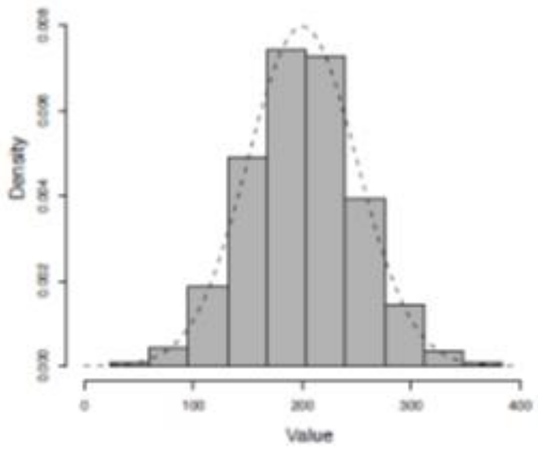
- **Two popular binning techniques** are:
  - **Equal Width Binning (Equi-Width Binning)**
  - **Equal Frequency Binning (Equi-Frequency Binning)**

Equal-Width Binning

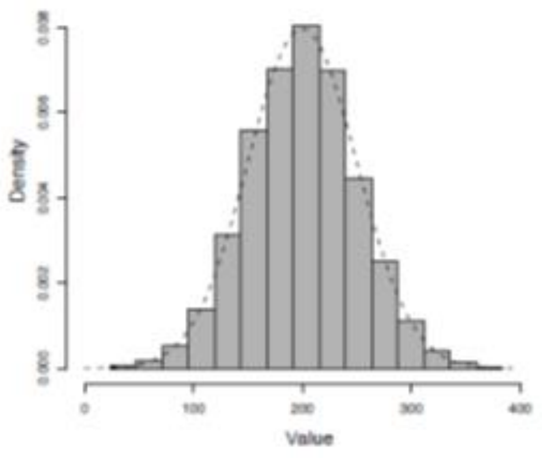
- In **equal width binning**, the range of values for the continuous feature is divided into **equally sized intervals** or **bins**, i.e., b bins each of size $\frac{\max value - \min value}{b}$
- Example:



(a) 5 Equal-width bins



(b) 10 Equal-width bins

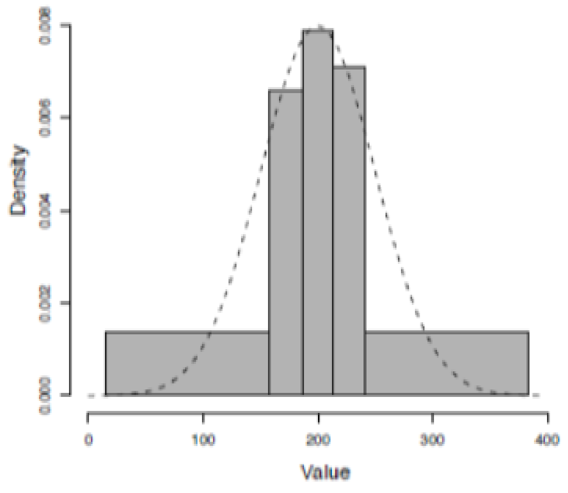


(c) 15 Equal-width bins

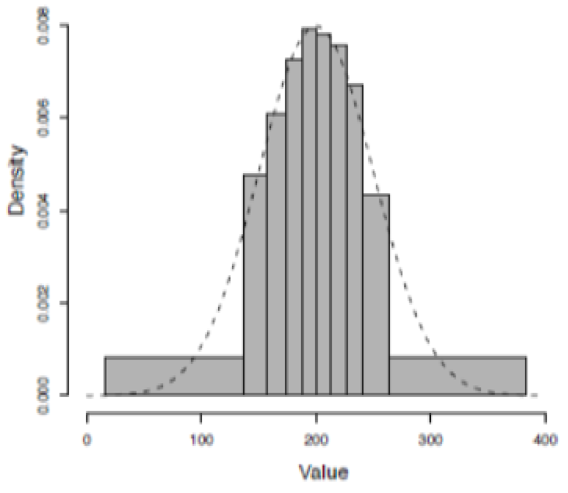
- This technique might not capture the underlying data distribution well, especially if the distribution is skewed.

Equal Frequency Binning

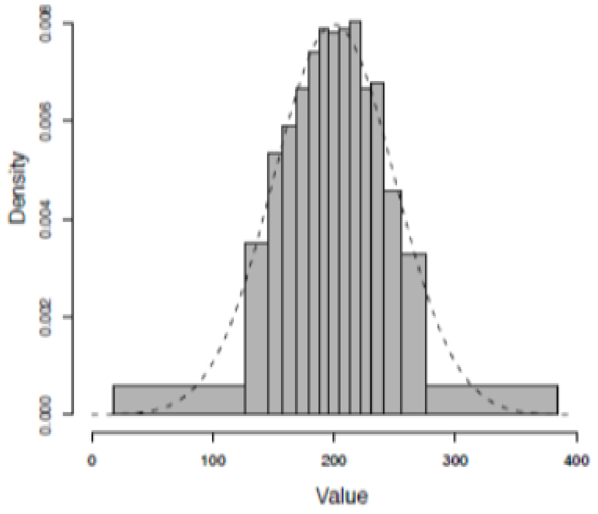
- In **equal frequency binning**, the goal is to create bins with approximately the same number of data instances.
- The width of the bins can vary based on the data distribution.
- This technique helps ensure that each bin captures a similar portion of the data, which can be particularly useful when dealing with skewed distributions.



(a) 5 Equal-frequency bins



(b) 10 Equal-frequency bins



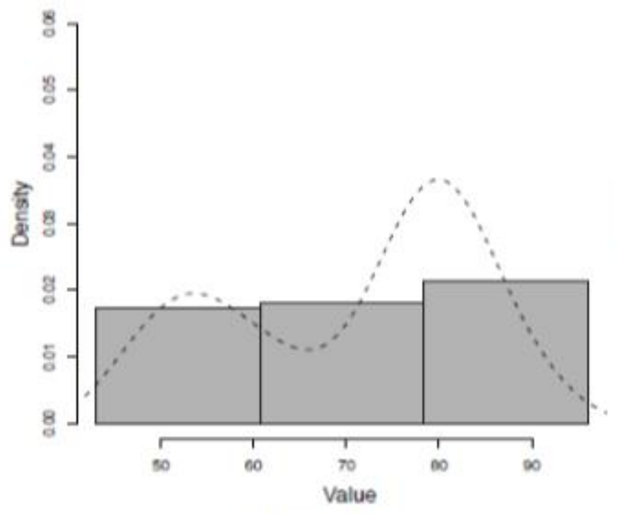
(c)15 Equal-frequency bins

Drawbacks of Binning

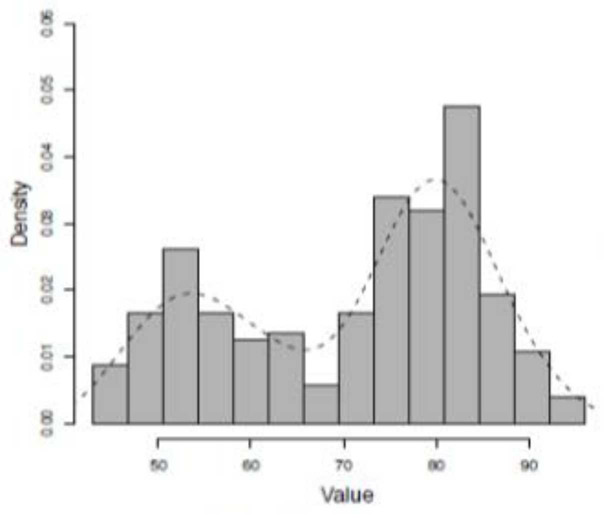
- **Information Loss**: Binning can lead to information loss, as the continuous nature of the original data is converted into discrete intervals. That might impact on model performance.
- **Sensitivity to Bin Size:** The choice of bin size can affect the results and interpretation of the analysis.
  - If we set the number of bins to a very low number we may lose a lot of information
  - If we set the number of bins to a very high number then we might have very few instances in each bin or even end up with empty bins.

Example: Number of Bins

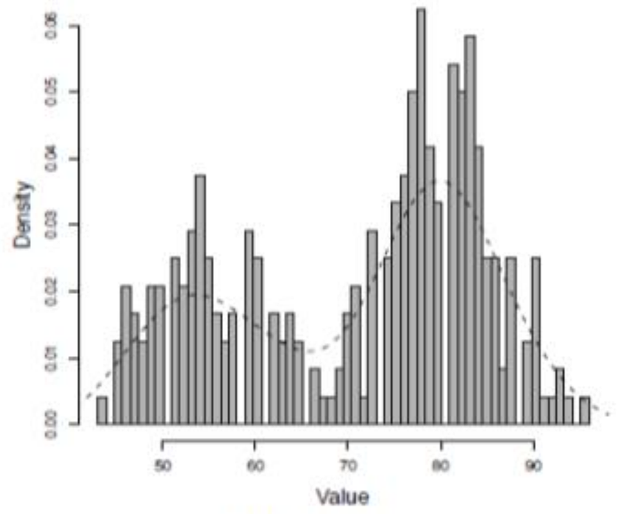
: a multimodal distribution



(a) 3 bins



(b) 14 bins



(c) 60 bins

In figure, ideally the histogram heights should follow the dashed line.

(a) doesn’t accurately represent the real distribution of values in the underlying continues feature.

(b) can be considered a reasonable representation of the feature

(c) shows a greater variance in the heights and some empty bins.

Outline

- Introduction
- Feature Transformation
  - Feature Scaling
  - Feature Encoding
  - Feature Creation
  - Binning/Bucketing
- ☞ **Feature Selection**
- Feature Extraction (Dimensionality Reduction)
- Summary

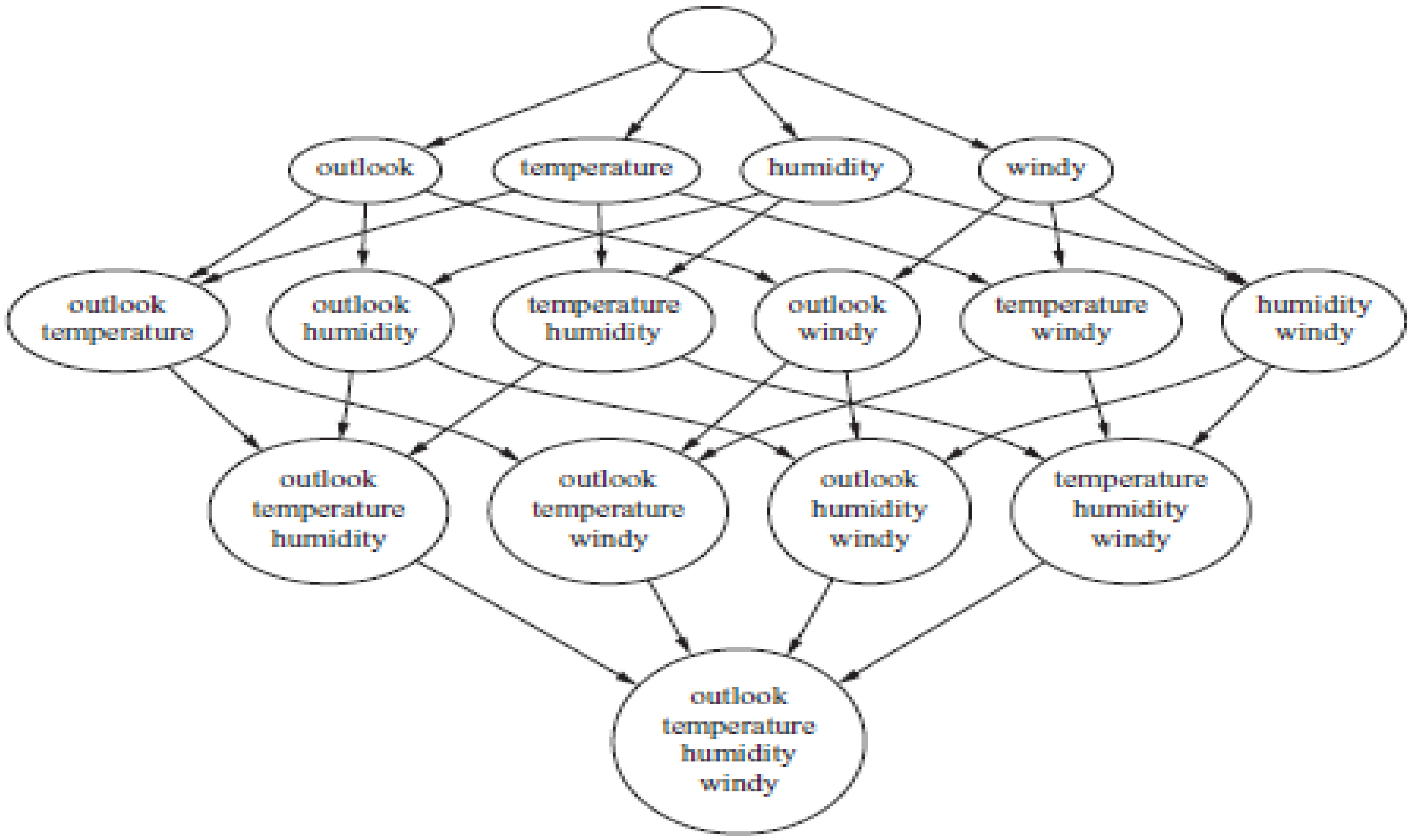
Feature Selection

- **Feature selection** involves choosing a subset of the available features (variables or attributes) that are most informative and impactful for the task at hand.
- The **goal of feature selection** is finding the best subset of the set of features which
  - contains the least number of dimensions that most contribute to the model performance, e.g., accuracy
  - is individually correlated well with prediction but have little intercorrelation

  and, removing the remaining irrelevant or redundant features
- Feature section **can involve domain knowledge, statistical tests, and algorithms that assess feature importance.**

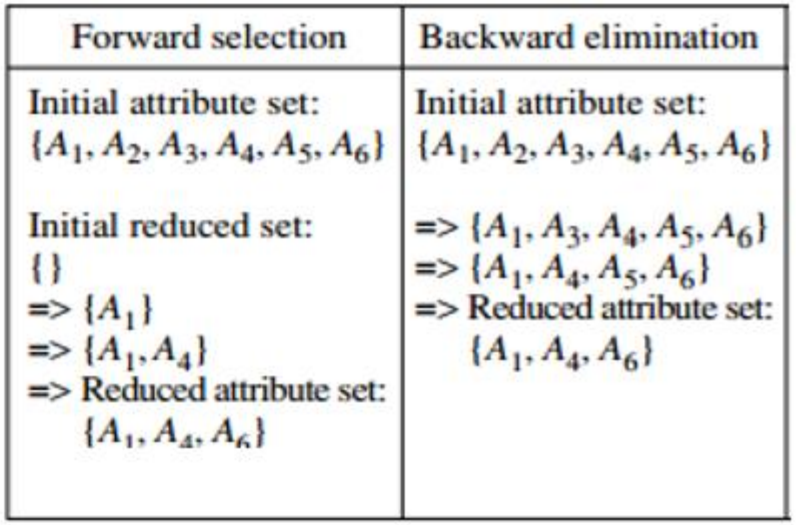
Feature Search Space

- **2^d subsets of *d* features**
  - We cannot test all of 2^d subsets of *d* features - An exhaustive search for the optimal subset of attributes is prohibitively expensive.



Heuristic Subset Search Methods

- We employee heuristics to get a reasonable (but not optimal) solution in reasonable (polynomial) time.
- Search the space of attribute subsets, evaluating each one - ***Greedy search***
- ***Forward search***: Start with no features and add them one by one, at each step adding the one that decreases the error the most.
- ***Backward search***: Start with all features and remove one at a time, if possible, to decrease the error the most
- ***Floating search***: Add different *k* features, remove *l* features at each step



Forward selection	Backward elimination
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: $\{\}$ => $\{A_1\}$ => $\{A_1, A_4\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$

Evaluation Methods

		What is evaluated?	
		Attributes	Subsets of attributes
Evaluation Method	Independent	Filters	Filters
	Learning algorithm		Wrappers

- **Filter methods** make an independent assessment based on general characteristics of the data
- **Wrapper methods** evaluate the subset using the machine learning algorithm that will ultimately be employed for learning

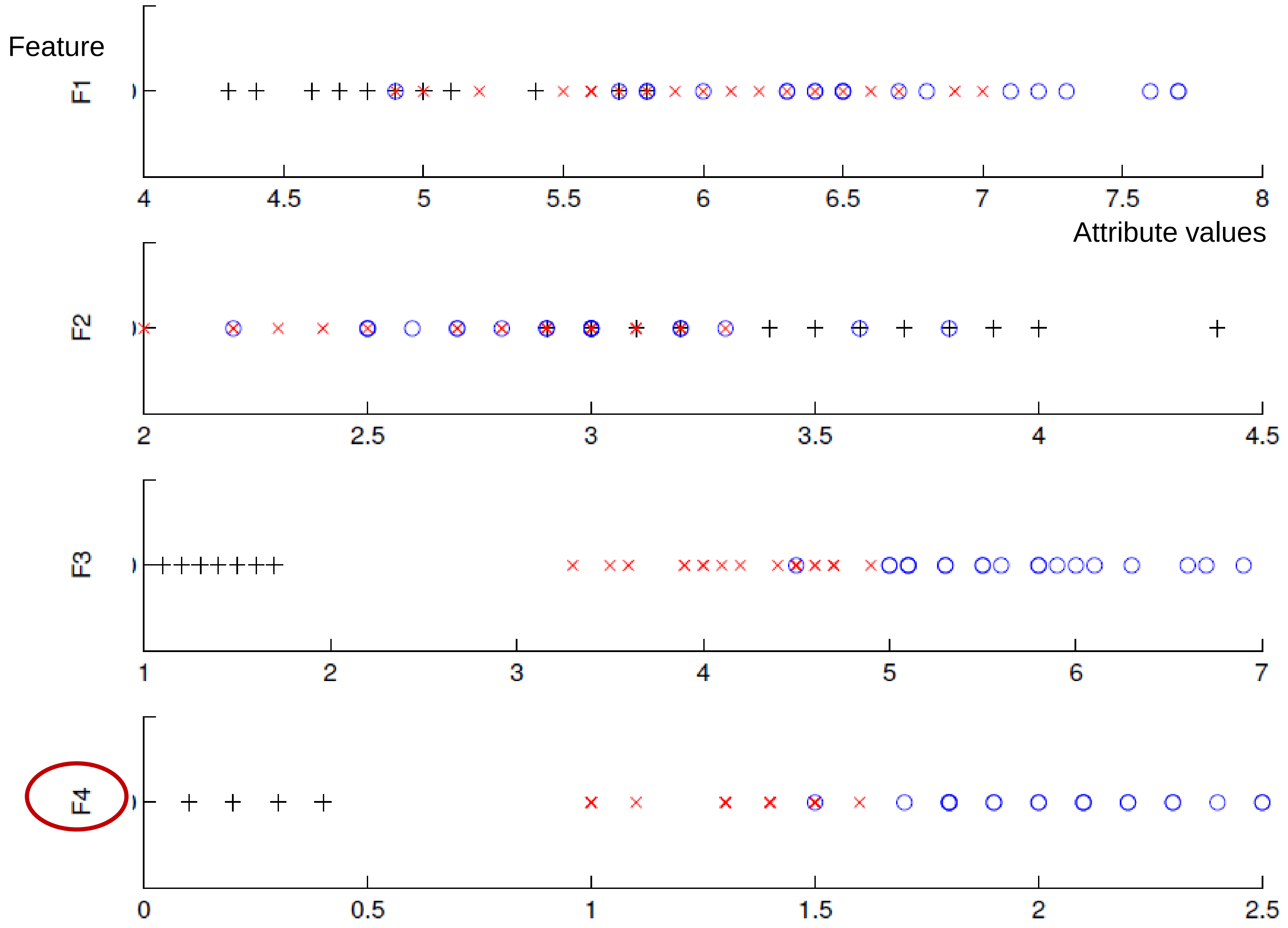
Forward Search Algorithm

- **Sequential Forward Search Algorithm**
  - *F:* a set of input features, x_i, $i=1, \ldots d$
  - *E(F):* the error incurred on the validation samples when only the features in *F* are used.
  - *E* is either *means square error* or *misclassification error*, according to the application
- **Initially *F*= Ø.**

1. At each iteration, find the best new feature that decreases the error the most.
   - For all possible features x_i, train a model on the training set and calculate $E (F \cup x_i)$ on the validation set
   - Choose a feature x_j that cause the least error, $\boldsymbol{j = \arg\min_i E(F \cup x_i)}$
2. **Add x_j to *F* if $E (F \cup x_j) < E (F)$**

- Step 1 and 2 are repeated until any further addition does not decrease *E* (or decreases *E* only slightly, i.e., < a user-defined threshold)

Example: Forward Search on Iris data



Classification accuracy (1 – error) with F1 is 0.76

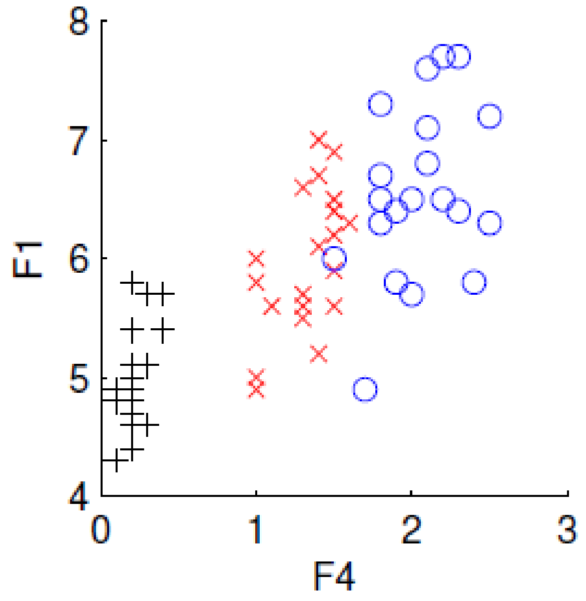
Accuracy with F2 is 0.57

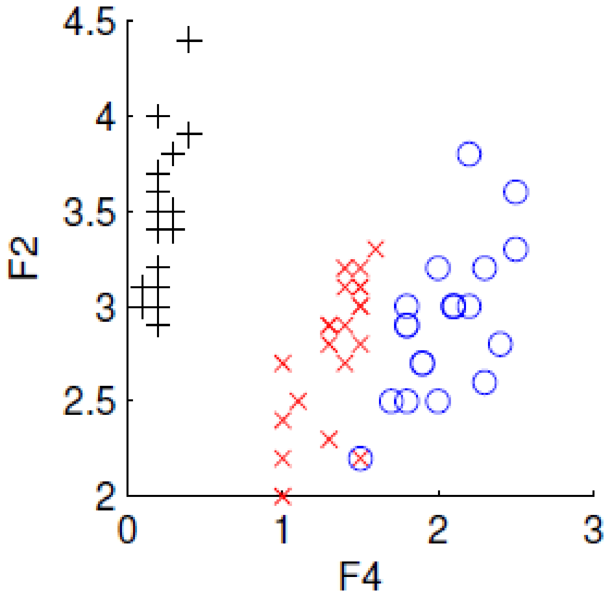
Accuracy with F3 is 0.92

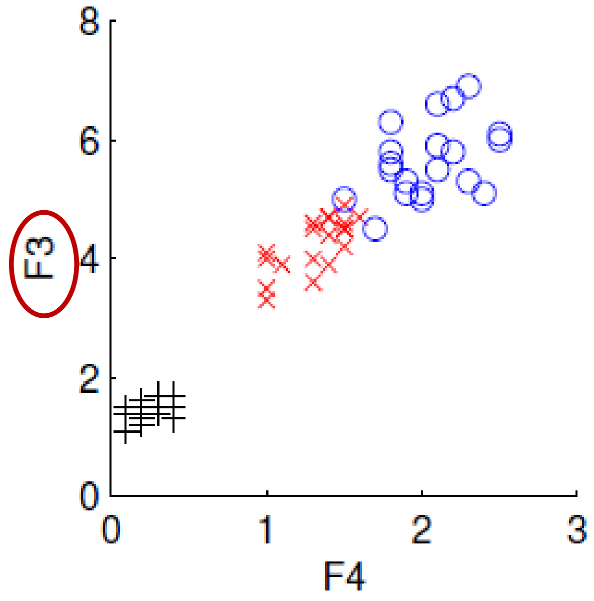
Accuracy with F4 is 0.94

F4 is chosen

Adding Additional Feature to F4







Classification accuracy with F4 and F1 is 0.87

Classification accuracy with F4 and F2 is 0.92

Classification accuracy with F4 and F3 is 0.96

F3 is chosen

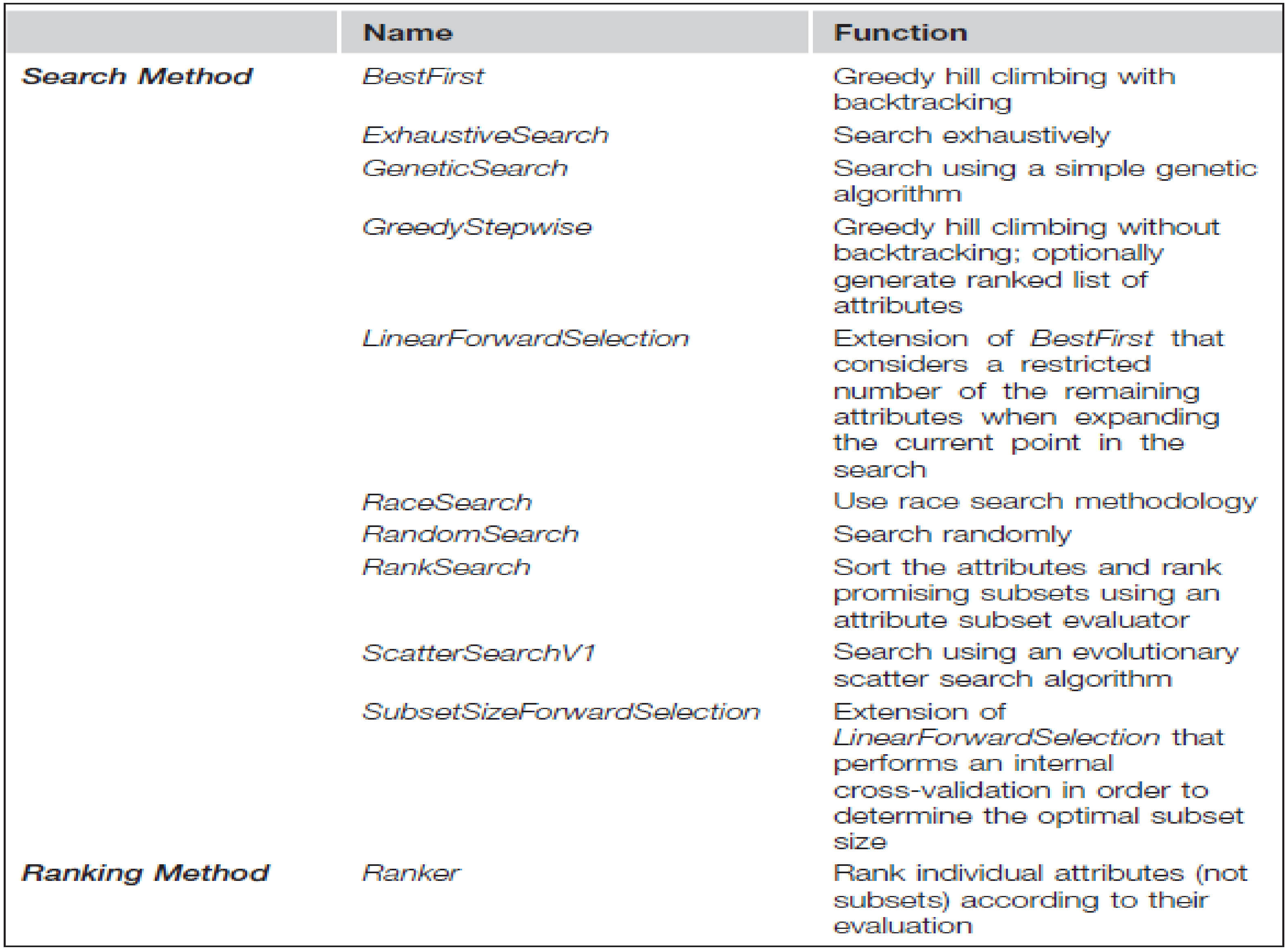
Backward Search Algorithm

- **Sequential Backward Elimination Algorithm**
  - *F:* a feature set of input features, $x_{i,}$ i=1, ...d
  - *E(F):* the error incurred on the validation samples when only the features in *F* are used.
  - *E* is either *means square error* or *misclassification error*, according to the application
- **Initially *F*=*F*.**

1. At each iteration, find the best feature that decreases the error the most when the feature is removed.
   - For all possible features $x_{i\,,}$ train a model on the training set and calculate $E\ (\ F - x_i\ )$ on the validation set
   - Choose the feature x_j that cause the least error, $\boldsymbol{j = \arg\min_i E(F - x_i)}$
2. **Remove x_j from F if $E\ (\ F - x_j\ ) < E\ (\ F\ )$**

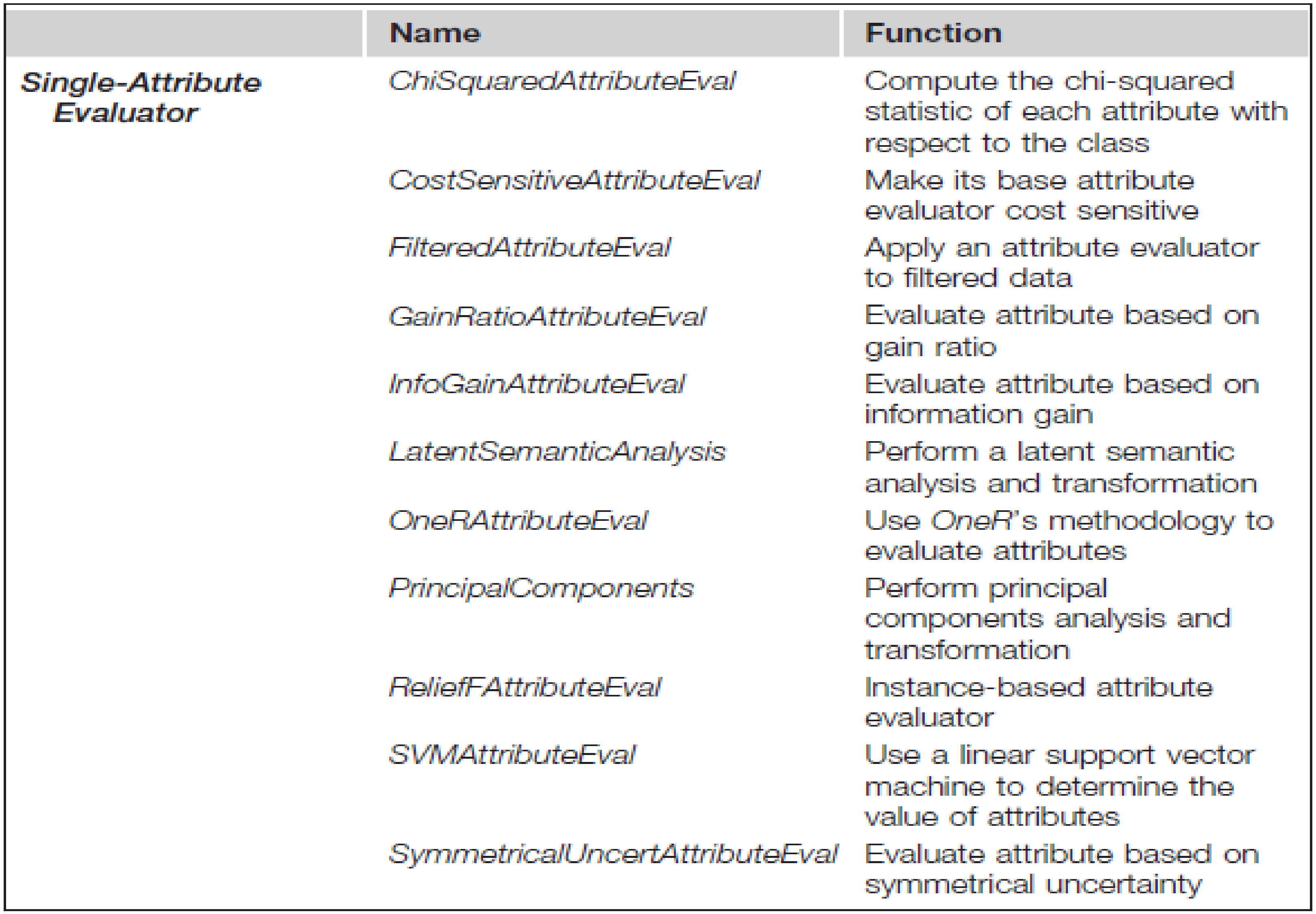
- Step 1 and 2 are repeated if removing a feature does not decrease *E* (or decreases *E* only slightly, i.e., < a user-defined threshold)

Feature Search Algorithms in Weka ML Tool



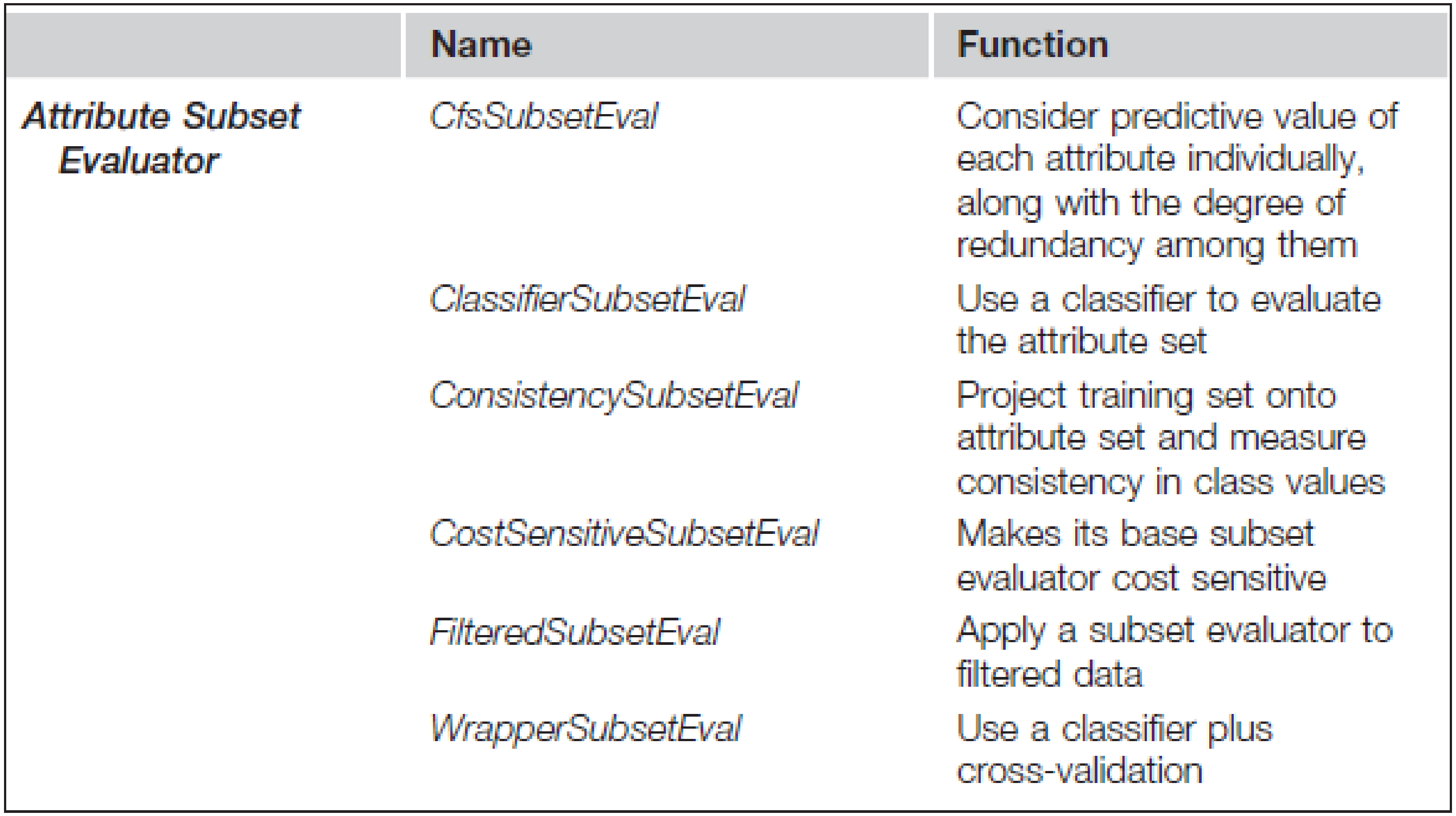
	Name	Function
Search Method	*BestFirst*	Greedy hill climbing with backtracking
	ExhaustiveSearch	Search exhaustively
	GeneticSearch	Search using a simple genetic algorithm
	GreedyStepwise	Greedy hill climbing without backtracking; optionally generate ranked list of attributes
	LinearForwardSelection	Extension of *BestFirst* that considers a restricted number of the remaining attributes when expanding the current point in the search
	RaceSearch	Use race search methodology
	RandomSearch	Search randomly
	RankSearch	Sort the attributes and rank promising subsets using an attribute subset evaluator
	ScatterSearchV1	Search using an evolutionary scatter search algorithm
	SubsetSizeForwardSelection	Extension of *LinearForwardSelection* that performs an internal cross-validation in order to determine the optimal subset size
Ranking Method	*Ranker*	Rank individual attributes (not subsets) according to their evaluation

Evaluation Metrics in Weka ML Tool



	Name	Function
Single-Attribute Evaluator	*ChiSquaredAttributeEval*	Compute the chi-squared statistic of each attribute with respect to the class
	CostSensitiveAttributeEval	Make its base attribute evaluator cost sensitive
	FilteredAttributeEval	Apply an attribute evaluator to filtered data
	GainRatioAttributeEval	Evaluate attribute based on gain ratio
	InfoGainAttributeEval	Evaluate attribute based on information gain
	LatentSemanticAnalysis	Perform a latent semantic analysis and transformation
	OneRAttributeEval	Use *OneR*'s methodology to evaluate attributes
	PrincipalComponents	Perform principal components analysis and transformation
	ReliefFAttributeEval	Instance-based attribute evaluator
	SVMAttributeEval	Use a linear support vector machine to determine the value of attributes
	SymmetricalUncertAttributeEval	Evaluate attribute based on symmetrical uncertainty

Evaluation Metrics in in Weka ML Tool



	Name	Function
Attribute Subset Evaluator	*CfsSubsetEval*	Consider predictive value of each attribute individually, along with the degree of redundancy among them
	ClassifierSubsetEval	Use a classifier to evaluate the attribute set
	ConsistencySubsetEval	Project training set onto attribute set and measure consistency in class values
	CostSensitiveSubsetEval	Makes its base subset evaluator cost sensitive
	FilteredSubsetEval	Apply a subset evaluator to filtered data
	WrapperSubsetEval	Use a classifier plus cross-validation

Outline

- Introduction
- Feature Transformation
  - Feature Scaling
  - Feature Encoding
  - Feature Creation
  - Binning/Bucketing
- Feature Selection
- ☞ **Feature Extraction (Dimensionality Reduction)**
- Summary

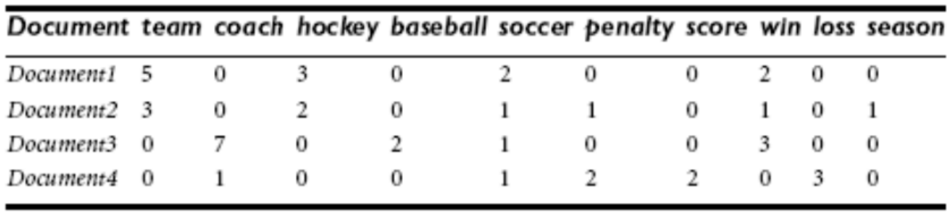
High-Dimensional Data

- Many Machine Learning problems involve thousands or even millions of features for each training instance.

  lots of features = High-dimensions

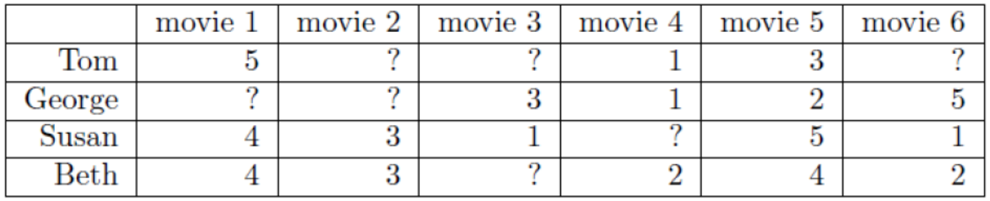
- Document classification
  - # of features per document

  : thousands of words/unigrams, millions of bigrams, contextual information



Document	team	coach	hockey	baseball	soccer	penalty	score	win	loss	season
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

- Movie ratings - Netflix
  - About 480000 users × 17800 movies



	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6
Tom	5	?	?	1	3	?
George	?	?	3	1	2	5
Susan	4	3	1	?	5	1
Beth	4	3	?	2	4	2

- MEG Brain Imaging
  - 120 locations × 500 time points × 20 objects
- Any high-dimensional image data

Curse of Dimensionality

- In high-dimensional space, many things behave very differently.
  - Most instances in a high-dimensional dataset are likely to be far way from each other.
    - With increase of dimensions, the average distance of any two points dramatically increases.
  - So, a new instance will likely be a far way from any training instance, making predictions much less reliable than in lower dimensions
- A high-dimensional dataset **makes training extremely slow**
- It can also **make it much harder to find a good solution**.
- This problem is often referred to as the ***curse of dimensionality.***

Real-World Data

- Fortunately, in most real-world problems**, training instances are *not* spread out uniformly across all dimensions.** Many features are almost constant, while others are highly correlated.
- As a result, **all training instances actually lie within (or close to) a much lower-dimensional *subspace* of the high-dimensional space.**

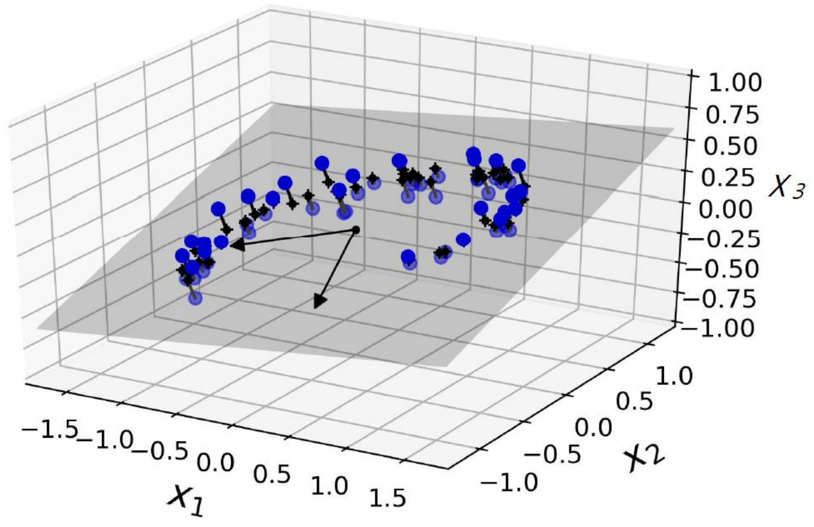


Figure. A 3D dataset lying close to a 2D subspace

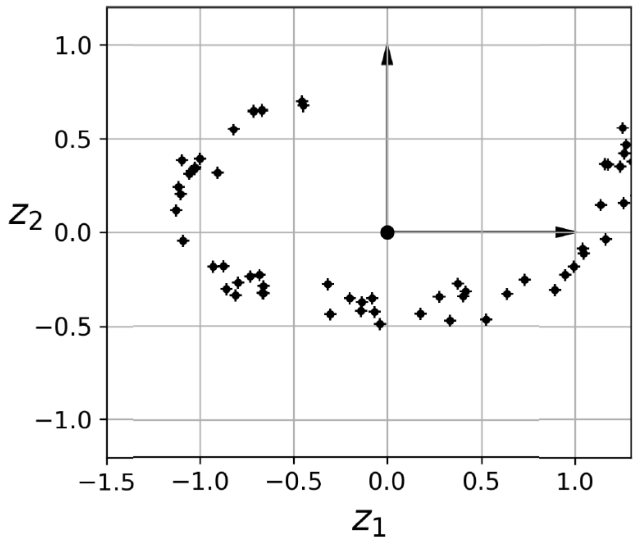
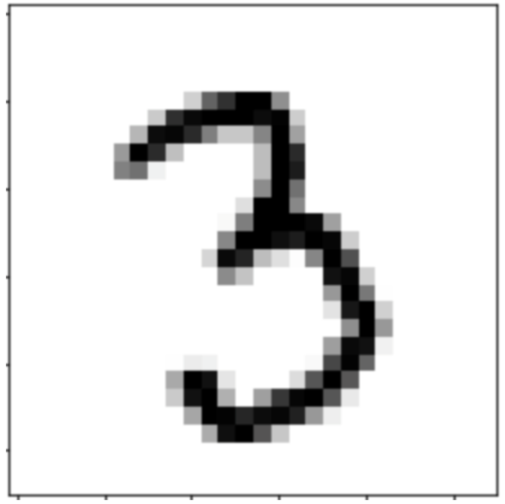


Figure. The new 2D dataset after projection

Reducing Dimensionality

- In real-world problem, the original high-dimensional data can be projected onto a much smaller feature space, resulting in **dimensionality reduction**.
- **Example**: In the image data of digits handwritten, the pixels on the image borders are almost always white. We can completely drop these pixels from training set without losing much information



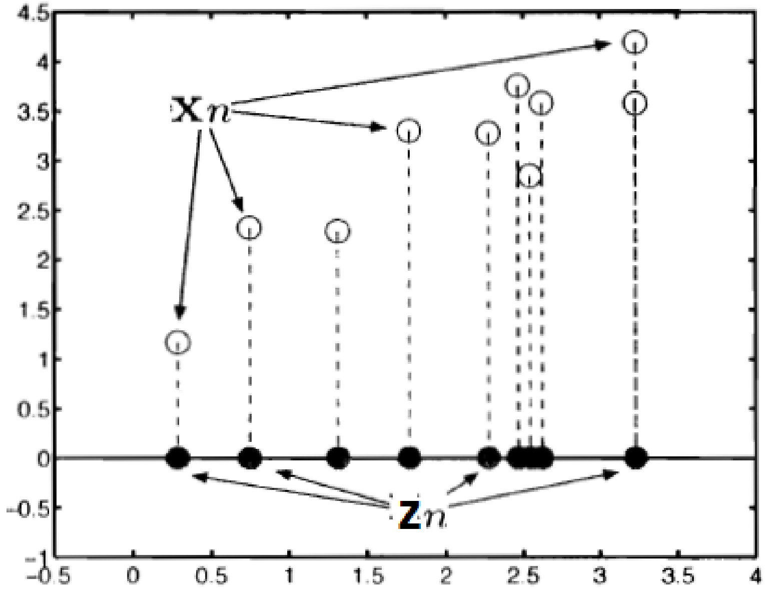
Benefits of Dimensionality Reduction

- **Remove noise** in data
- **Saves the cost** of observing the feature to be unnecessary
  - Lower resource requirements for storing/processing data
- **More efficient use of resources**
  - Reduces time complexity: Less computation
  - Reduces space complexity: Less disk
- Statistically, fewer dimensions indicates **better generalization**, Also, fewer parameters in many models
- **Simpler models might be more robust** on small datasets
- **More interpretable**; **simpler explanation**
- Usually speed up training but it may not always lead to a better or simple solution depending on the dataset.
- **Visualize data more easily** (structure, groups, outliers, etc) if plotted in 2 or 3 dimensions

Feature Extraction

- **Feature extraction** is finding a new set of *k* features ($k < d$) that are combinations of the original *d* features
- We are interesting in finding a mapping from the inputs in the original *d*-dimensional space to a new ($k < d$) -dimensional space, with minimum loss of information.
- This process is known as **projection**.
  - Let the original data set be **x** where $x_i \in \mathbf{x}, i = 1, \ldots, d$
  - Project the original x_i dimensions to new $k < d$ dimensions, z_j , $j = 1,\ldots,k$
  - The projection of **x** on the direction ***w*** is denoted to

$$z = \mathbf{w}^T \mathbf{x}$$



Feature Extraction Methods

- The most widely used feature extraction methods are
  - ***Principal Component Analysis* (PCA)**
  - ***Kernel PCA***
  - ***Independent Component Analysis* (ICA)**
  - ***Linear Discriminant Analysis* (LDA)**

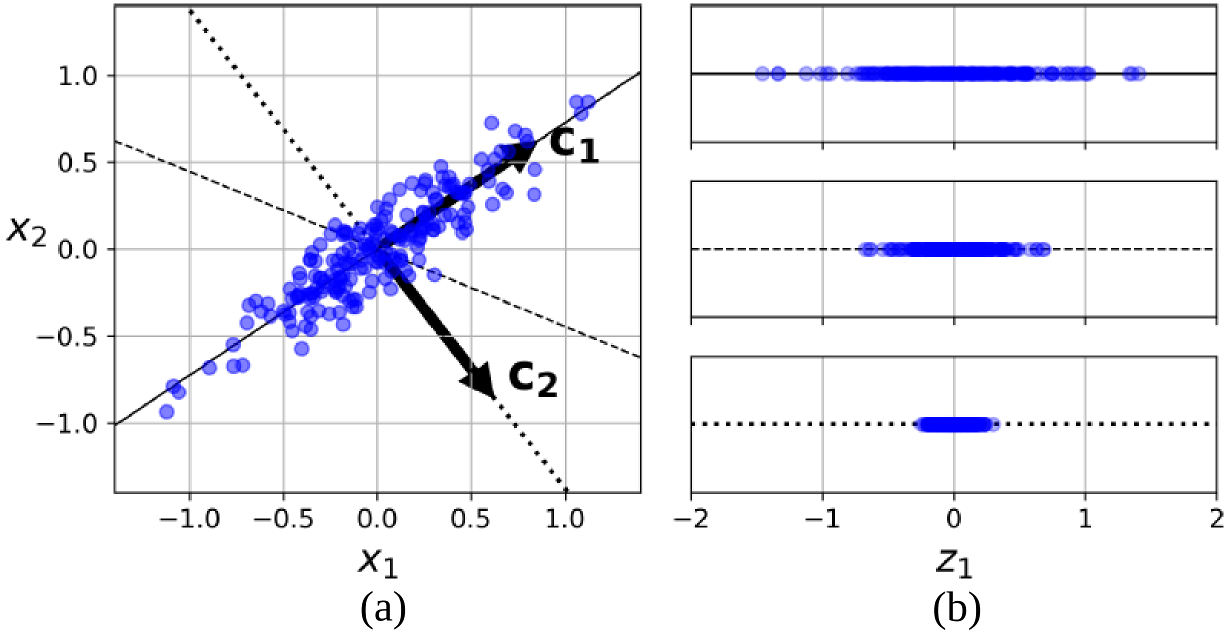
- Feature extraction methods are also powerful **unsupervised learning techniques** for extracting hidden (potentially lower dimensional) structure from high dimensional datasets.

Principal Components Analysis

- ***Principal Component Analysis* (PCA)** is by far the most popular dimensionality reduction algorithm.
- PCA is a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called ***principal components***.
- PCA could broadly be classed as **projection** techniques
  - First PCA identifies the hyperplane that lies closest to the data, and then PCA projects the data onto it
- PCA is an **unsupervised approach**, since it involves only a set of descriptive features, and no associated target feature.
- PCA also serves as **a tool for data visualization**.

Idea behand PCA

- Before you can project the training set onto a lower-dimensional hyperplane, you first need to choose the right hyperplane.
- It seems reasonable to select the axis that preserves the maximum amount of variance, as it will most likely lose less information than the other projections.

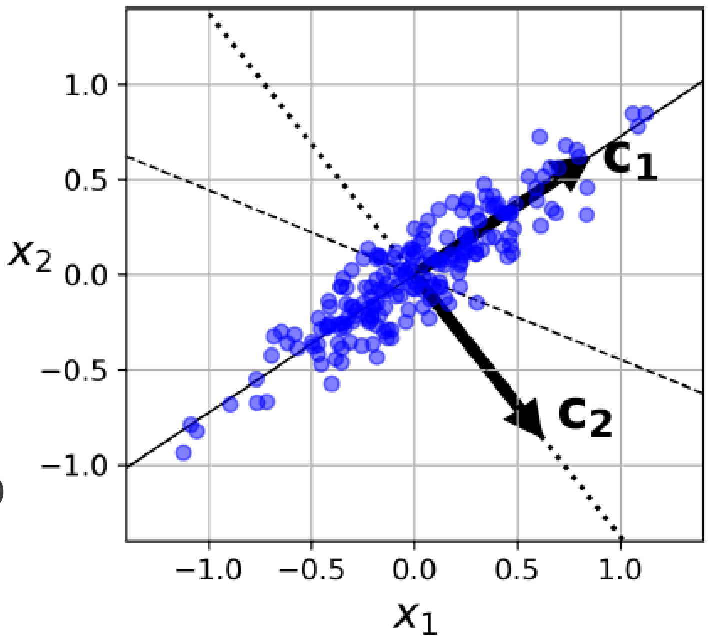


This project on the solid line (C1) preserves the maximum variance.

Figure: (a) a simple 2D dataset, along with three different axes (i.e., one-dimensional hyperplanes) (b) the result of the projection of the dataset onto each of these axes.

Coordinate System for PCA

- PCA uses a special coordinate system that depends on the set of data points as follows
  - Place the first axis in the direction of greatest variance of the points to maximize the variance along that axis.
  - The following axis should be always constrained to be perpendicular to the first axis. Subject to this constraint, choose the second axis in the way that maximizes the variance along it.
  - Continue, choosing each axis to maximize its share of the remaining variance.



Principal Component (PC)

- The unit vector that defines the *i*th axis is called the ***ith principal component* (PC)**.
- The first two PCs are represented by the orthogonal arrows in the plane, and the third PC would be orthogonal to the plane (pointing up or down).

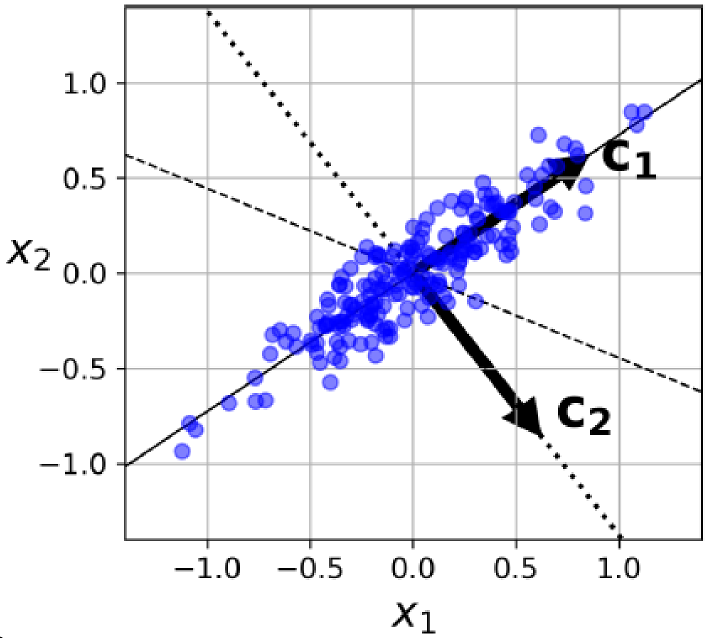


Figure: the 1st PC is **c1** and the 2nd PC is **c2**.

PCA Algorithm

- How can we find the principal components of a data set?
    - Via an **eigen-decomposition of the covariance matrix**

- **Basic PCA algorithm**
    - Start from $d \times n$ data matrix X
    - Recenter with subtracting mean from each row of X

$$\mathrm{X}_c \leftarrow \mathrm{X} - \overline{\mathrm{X}}$$

    - **Compute covariance matrix**:

$$\Sigma \leftarrow \frac{1}{d}\mathrm{X}_c^T \mathrm{X}_c$$

    - Find **eigenvectors and eigenvalues** of Σ
    - **Principal components** are k eigenvectors with highest eigen values.
        - The k eigenvectors with nonzero eigenvalues are the dimensions of the reduced space

PCA Algorithm

- Alternatively,
  - PCs can also be found via ***Singular Value Decomposition*** (SVD) of the data matrix **X**
  - SVD is a standard matrix factorization technique that can decompose the data set matrix **X** into the matrix multiplication of three matrices $\mathbf{U}\,\Sigma\,\mathbf{V}^T$, where **V** contains all the principal components that we are looking for, as shown in

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

How to Create the Reduced Dataset

- Once you have identified all the principal components, you can reduce the dimensionality of the dataset down to *d* dimensions by projecting it onto the hyperplane defined by the first *d* principal components.
- To project the training set onto the hyperplane and obtain a reduced dataset $\mathbf{X}_{d-proj}$, we can simply compute the matrix multiplication of the data set matrix **X** by the matrix $\mathbf{W}_d$, defined as the matrix containing the first *d* principal components (i.e., the matrix composed of the first *d* columns of **V**), as $\mathbf{X}_{d-proj} = \mathbf{X}\mathbf{W}_d$

Proportion of Variance

- The **proportion of variance (PoV)** explained by the k principal components is defined as $\frac{\lambda_1+\lambda_2+\cdots+\lambda_k}{\lambda_1+\lambda_2+\cdots+\lambda_k+\cdots+\lambda_d}$, where λ_i are sorted in descending order
- It is also called **explained variance ratio**.

k : Number of PCs

- **If the dimensions are highly correlated**, there will be a small number of eigenvectors with large eigenvalues and <u>*k* will be much smaller than *d*</u> and <u>a large reduction in dimensionality may be attained.</u>
  - This is typically the case in many image and speech processing tasks where nearby inputs (in space or time) are highly correlated.
- **If the dimensions are not correlated**, <u>*k* will be as large as *d* and there is no gain through PCA</u>.

How to Choose *k*

- In practice, even if all eigenvalues are greater than 0, if some eigenvalues are small and have little contribution to variance, they may be discarded. Then we take into account the leading *k* components that explain more than, e.g., 95 percent, of the variance.

- If we use principal components as a summary of our data, how many principal components are sufficient?
  - No simple answer to this question, as cross-validation is not available for this purpose.

How to Choose *k*

- ***Scree graph*** is the plot of variance explained as a function of the number of eigenvectors kept.
  - By visually analyzing it, one can also decide on *k*.

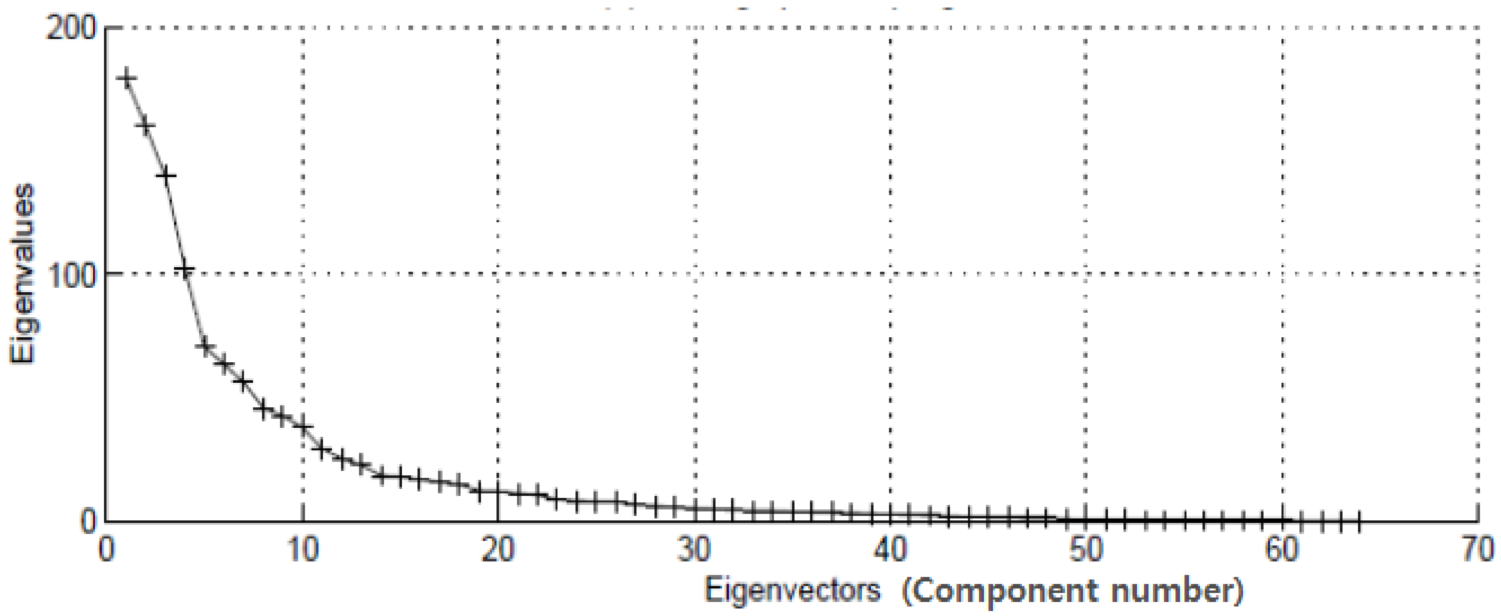


Figure Scree graph.

How to Choose *k*

- To decide *k*, we may look for an "elbow" in **the *plot of proportion of variance***
- At the "elbow", adding another eigenvector (principle component) does not significantly increase the variance explained.

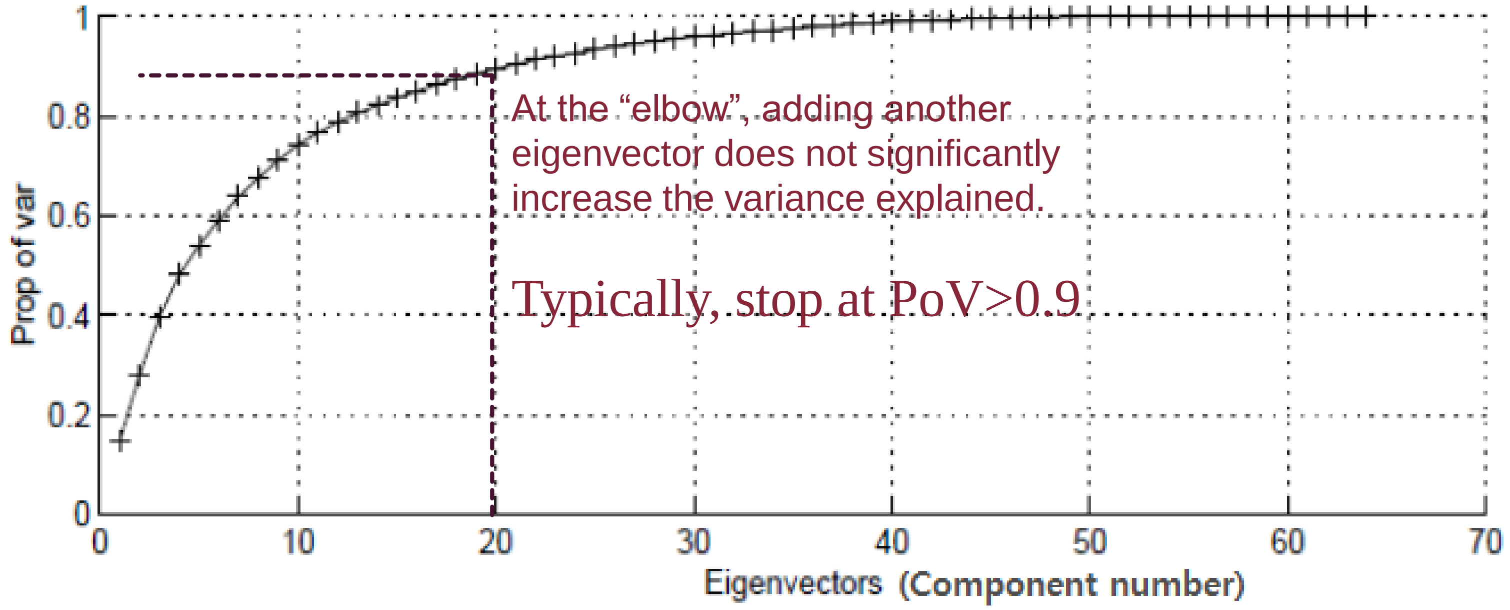


Figure Proportion of variance explained.
The first twenty eigenvectors explain 90 percent of the variance.

Outline

- Introduction
- Feature Transformation
  - Feature Scaling
  - Feature Encoding
  - Feature Creation
  - Binning/Bucketing
- Feature Selection
- Feature Extraction (Dimensionality Reduction)

☞ **Summary**

Summary

- Feature engineering encompasses several tasks, including:
  - Feature Transformation
  - Feature Scaling
  - Feature Encoding
  - Feature Creation
  - Binning/Bucketing
  - Feature Selection
  - Feature Extraction (Dimensionality Reduction)
    - PCA (Principal Component Analysis)
  - and so on.

Summary (cont.)

- Good feature engineering can lead to better model accuracy, faster convergence, reduced overfitting, and improved interpretability of the machine learning model.
- Effective feature engineering requires a deep understanding of the domain and the problem at hand.
- It often involves an iterative process of experimentation, where the impact of different feature engineering techniques on the model's performance is evaluated.

Summary (cont.)

- In data preparation, we are changing the data that we will use to subsequently train predictive models.
- Notice that if we change the data too much, the models we build might not relate well to the original data.
- It is important to make a delicate balance that we need to strike between preparing the data for machine learning algorithms and keeping the data true to the underlying processes that generate it.
- Well-designed evaluation experiments are the best way to find this balance.

APPENDIX

- Principal Component Computation

Notations

- The data points (samples) of a set of features $\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_d$ are represented by matrix X of size $d \times n$

$$\mathrm{X} = \begin{bmatrix} \mathrm{x}_{11} & \mathrm{x}_{12} & \dots & \mathrm{x}_{1d} \\ \mathrm{x}_{21} & \mathrm{x}_{22} & \dots & \mathrm{x}_{2d} \\ \dots & \dots & \dots & \dots \\ \mathrm{x}_{n1} & \mathrm{x}_{n2} & \dots & \mathrm{x}_{nd} \end{bmatrix}$$

- The **mean vector** $\mathrm{E}[\mathrm{x}] = \boldsymbol{\mu} = [\mu_1, \dots, \mu_d]^T$ where each of its elements is the mean of one column of X
- The **variance** of X_i is denoted as σ_i^2
- The **covariance** of two variables X_i and X_j is defined as

$$\sigma_{ij} = \mathrm{Cov}(\mathrm{X}_i, \mathrm{X}_j) = E[(\mathrm{X}_i - \mu_i)(\mathrm{X}_j - \mu_j)] = E[\mathrm{X}_i \mathrm{X}_j] - \mu_i \mu_j$$

- The **covariance matrix** is denoted as Σ

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \dots & \sigma_{2d} \\ \dots & \dots & \dots & \dots \\ \sigma_{d1} & \sigma_{ij} & \dots & \sigma_d^2 \end{bmatrix}$$

The diagonal terms are the variances, the off-diagonal terms are the covariances, and the matrix is symmetric

- In vector-matrix notation,

$$\Sigma \equiv \mathrm{Cov}(\mathbf{X}) = E[(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T] = E[\mathbf{X}\mathbf{X}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T$$

First Principal Component

- The **first principal component** of a set of features, $\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_d$ is the linear combination of the features.

  $$z_1 = w_{11}\mathrm{x}_1 + w_{21}\mathrm{x}_2 + \cdots + w_{d1}\mathrm{x}_d \quad \text{(i.e., } z_1 = \boldsymbol{w}_1^T\mathbf{x}\text{)}$$

  <u>that has the largest variance</u>.

  - We constraint $\sum_{j=1}^{d} w_{j1}^2 = 1$ (i.e., $\|\boldsymbol{w}_1\| = \boldsymbol{w}_1^T\boldsymbol{w}_1 = 1$)

- That is, we seek $\boldsymbol{w}_1 = (w_{11}, \ldots, w_{d1})^T$ such that $\mathrm{Var}(z_1)$ is maximized, subject to the constraint that $\boldsymbol{w}_1^T\boldsymbol{w}_1 = 1$.

  - $\boldsymbol{w}_1 = (w_{11}, \ldots, w_{d1})^T$ defines a direction in feature space along witch the data vary the most.

- If we project the *n* data points $x_1, x_2, \ldots, x_n$ onto this direction, the projected values are the principal component scores $z_{11}, \ldots., z_{n1}$ themselves, where $z_{11}, \ldots, z_{n1}$ are referred to the ***scores*** of the first principal component.

First Principal Component

- Again, we need to find $\boldsymbol{w_1} = (w_{11}, \dots, w_{d1})^{\boldsymbol{T}}$ such that $\mathrm{Var}(z_1)$ is maximized, subject to the constraint that $\boldsymbol{w_1^T w_1} = 1$.
- That is an optimization problem:

$$\underset{w_{11},\dots,w_{d1}}{maximize} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{d} w_{ji} x_{ij} \right)^2 \right\}$$

that is, $\underset{w_{11},\dots,w_{d1}}{maximize} \left\{ \frac{1}{n} \sum_{i=1}^{n} z_{i1}^2 \right\}$, subject to $\sum_{j=1}^{d} w_{j1}^2 = 1$

Computation of First Principal Component

- The maximization objective is just the variance $Var(z_1)$ of the n values of $z_{i1}$of projected data

  Find a vector w_1 that maximizes the variance of the n values of z_{i1}, that is, $Var(z_1) = \frac{1}{n}\sum_{i=1}^{n} z_{i1}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} w_{ji}x_{ij}\right)^2$

  $$= \frac{1}{n}\sum_{i=1}^{n}(w_1^T \mathrm{x}_i)^2 = Var(w_1^T \mathrm{x}_i)$$

  $$= w_1^T \Sigma w_1, \text{ subject to } w_1^T w_1 = 1$$

- This is a constrained optimization problem.
- Lagrangian wraps constrains into the objective function.

$$\underset{w_1}{maximize}\, w_1^T \Sigma w_1 - \alpha(w_1^T w_1 - 1)$$

- This problem can be solved by using matrix computation tools (*Eigenvector* or *Singular Value Decomposition*).

Computation of First Principal Component (Cont.)

$$\underset{w_1}{maximize}\, w_1^T \Sigma w_1 - \alpha(w_1^T w_1 - 1)$$

- Taking the derivative with respect to w_1 and setting it equal to 0 ($\frac{\partial}{\partial w_1} = 0$) , we have $2\Sigma\boldsymbol{w_1} - 2\alpha\boldsymbol{w_1}$=0, therefore,$\Sigma\boldsymbol{w_1} = \alpha\boldsymbol{w_1}$.
- here, $\boldsymbol{w_1}$ is the **eigenvector** of sample correlation matrix xx^T, and $\boldsymbol{\alpha}$ the corresponding **eigenvalue**.

- $Var(z_1) = w_1^T \Sigma w_1 = w_1^T \alpha w_1 = \alpha w_1^T w_1 = \boldsymbol{\alpha}$, because $w_1^T w_1 = 1$
- Because we want to maximize the sample variance of projection, we choose the eigenvector with the largest eigenvalue.
- Therefore the first principal component is the eigenvector of the covariance matrix of the data with the largest eigenvalue, $\lambda_1 = \alpha$

Example

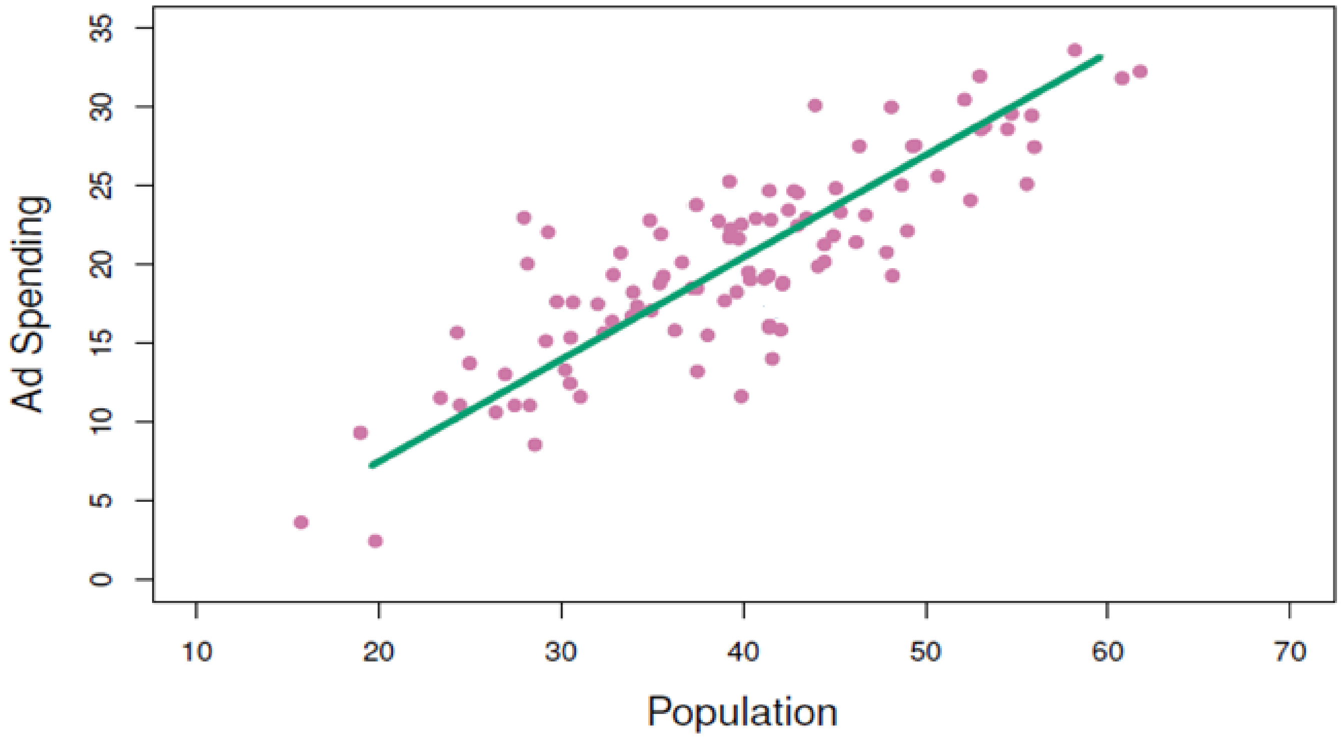


Figure: The population size (pop) and ad spending (ad) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component direction. w_{11}=0.839, $w_{21} = 0.544$

Second Principal Component

- The **second principal component** of a set of features, $x_1, x_2, \dots, x_d$ is the linear combination of the features.

  $$z_2 = w_{12}x_1 + w_{22}x_2 + \cdots + w_{d2}x_d \quad \text{(i.e., } z_2 = \boldsymbol{w}_2^T\mathbf{x}\text{)}$$

  <u>that has the largest variance among all linear combinations that are uncorrelated with</u> z_1.
- The condition of z_2 to be uncorrelated with z_1 is equivalent to constraining the direction $\boldsymbol{w_2}$ to be orthogonal (perpendicular) to the direction $\boldsymbol{w_1}$, i.e., $\boldsymbol{w_1} \cdot \boldsymbol{w_2} = \mathbf{0}$
- For the second principal component, as a Lagrange problem, we have $\underset{w_2}{maximize}\, w_2^T\Sigma w_2 - \alpha(w_2^T w_2 - 1) - \beta(w_2^T w_1 - 0)$

Computation of Second Principal Component

$$\underset{w_2}{maximize}\, w_2^T \Sigma w_2 - \alpha(w_2^T w_2 - 1) - \beta(w_2^T w_1 - 0)$$

- Taking the derivative with respect to w_2 and setting it equal to 0 , we have $2\Sigma w_2 - 2\alpha w_2 - \beta w_1$=0
- Premultiply by w_1^T and we get

$$2w_1^T \Sigma w_2 - 2\alpha w_1^T w_2 - \beta w_1^T w_1 = 0$$

  - Note that $w_1^T w_2 = 0$ because of the orthogonal constraint, $w_1^T \Sigma w_2$ is a scalar, equal to its transpose $w_2^T \Sigma w_1$, where because w_1 is the leading eigenvector of Σ, $\Sigma w_1 = \lambda_1 w_1$
  - Therefore, $w_1^T \Sigma w_2 = w_2^T \Sigma w_1 = \lambda_1 w_2^T w_1 = 0$
- β=0 . Then $\Sigma w_2 = \alpha w_2$ which implies that <u>w_2 should be the eigenvector of Σ with the second largest eigenvalue, $\lambda_2 = \alpha$.</u>

Further Principal Components

- Similarly, we can show that the other dimensions are given by the eigenvectors with decreasing eigenvalues, $\lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots$
- If we rank eigenvalues from large to small,
  - The 1st PC, $\boldsymbol{w}_1$, is the eigenvector of the sample covariance matrix Σ associated with largest eigenvalue
  - The 2nd PC, $\boldsymbol{w}_2$, is the eigenvector of the sample covariance matrix Σ associated with the second largest eigenvalue
  - and so on