# ML Study Note: Feature Engineering[*]

Lizi Chen

“数据和特征决定机器学习的上限；模型和算法无限逼近这个上限。”

## 1 Feature Scaling (Normalization and Standardization)

For continuous data, normalization(归一化) helps to build a model that ‘treats’ all features ‘equally’.

For example, one feature has values range from 0 to 1000, and another one has values range from 0 to 1. In supervised learning, the model will take the first feature while almost ignore the effect from the second feature.

paraphrase this part.

### 1.1 Standardization (z-score, 零-均值标准化)

$$x'_i = \frac{x_i - mean(X)}{standard\_deviation(X)} \tag{1}$$

```python
# Use sklearn.preprocessing.StandardScaler for both training set and test set.
scaler = preprocessing.StandardScaler().fit(X_train)
scaler.transform(X_train)
# scaler can transform X_test the same way as it did on X_train
scaler.transform(X_test)
```

### 1.2 Min-Max Normalization

$$x'_i = \frac{x_i - min(X)}{max(X) - min(X)} \tag{2}$$

```python
from sklearn.preprocessing import MinMaxScaler
MinMaxScaler().fit_transform(X)
```

### 1.3 Mean Normalization

$$x'_i = \frac{x_i - ave(X)}{max(X) - min(X)} \tag{3}$$

### 1.4 L2 Normalization

$$x'_i = \frac{x_i}{\sqrt{\sum_j x_j^2}} \tag{4}$$

```python
from sklearn.preprocessing import Normalizer
Normalizer().fit_transform(X)
```

[*] Please consider star and contribute to the Github repository.

By applying feature scaling techniques, we will not only ease the visualization plotting but also improving the training speed; namely the gradient descent process. This is because by having all variables in small range i.e., [0, 1] or [−1, 1], the parameters $\theta$ descend more quickly than variables in large ranges. The Figure 1 shows that when variables value are very uneven, the gradient descend oscillates inefficiently down to the optimum, compared to the right plot.

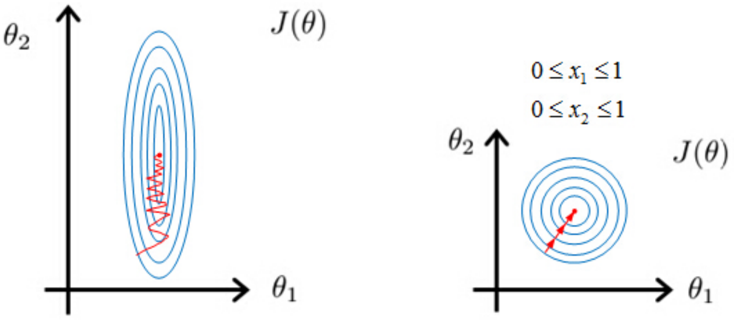


Figure 1: How feature scaling helps gradient descent.

### 1.5 Binarizer

$$x' = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{if } x < \text{threshold} \end{cases} \tag{5}$$

```
from sklearn.preprocessing import Binarizer
Binarizer(threshold = someNumber).fit_transform(X)
```

### 1.6 Scikit-learn functions:

Use of sklearn.preprocessing: Compare the effect of different scalers on data with outliers: Scikit-learn Tutorial

todo

### 1.7 Note:

**Exceptions:** We don't need to do feature normalization for Tree-based models such as Random Forest, Bagging, Boosting, etc. However, for all parameterized models or models that are based on distance/proximity, we always need feature normalization.

# 2 Correlation Analysis and Feature Selection

By analyzing how each feature relates each other, we choose features to train a model. Correlation Analysis → Feature Selection.

## 2.1 Bias and Variance:[1]

Let $\mathbf{P}$ be a probability distribution, $\mathbf{D}$ be a sample set of data from $\mathbf{P}$.
$\mu : \mathbf{P} \to \mathbf{R}$ is a real-value parameter.
$\hat{\mu} : \mathbf{D} \to \mathbf{R}$ is an estimator of $\mu$

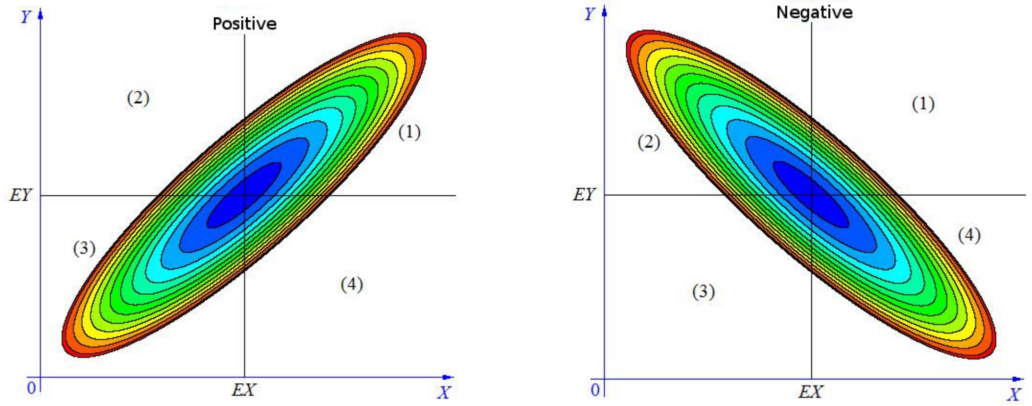
Define **bias** of $\hat{\mu}$ as $\mathbf{bias}(\hat{\mu}) = \mathbb{E}(\hat{\mu}) - \mu$.
An estimator is **unbiased** if $\mathbf{bias}(\hat{\mu}) = 0$.

Define **variance** of $\hat{\mu}$ as $\mathbf{variance}(\hat{\mu}) = \mathbb{E}(\hat{\mu}^2) - (\mathbb{E}(\hat{\mu}))^2$.

## 2.2 Covariance

Variance and standard error tells the distribution of a one-dimensional data. Covariance works in the situation where the data is two-dimensional.

$$\begin{aligned} Cov(X,Y) &= E((X - E(X)(Y - E(Y)) \\ &= E(X \cdot Y) - E(X) \cdot E(Y) \end{aligned} \tag{6}$$



(a) Covariance: Positive

(b) Covariance: Negative

Figure 2: X & Y correlation distributions with centered means.

As shown in Figure 2, the two dimension correlation distribution of X and Y are centered by their means; $E(X)$ and $E(Y)$. In the first quadrant, as marked by (1); $X > E(X)$ and $Y > E(Y)$, thus $(X - E(X))(Y - E(Y)) > 0$. When feature X and Y are positively correlated, there are more areas in the first quadrant (1) and the third quadrant (3). When they are negatively correlated, more areas in (2) and (4).

Note that when $Cov(X,Y) = 0$, X and Y does NOT necessarily independent. $Cov(X,Y) = 0$ only implies that X and Y are not correlated.

When number of random variables are greater than 2, we use **Covariance Matrix** to calculate the covariances of pairs.

$$Cov(x,y,z) = \begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix} \tag{7}$$

[1] This part is also written in the 'Bagging and Boosting Note'.

## 2.3 PC: Pearson Correlation

Pearson Correlation (also known as product moment correlation coefficient, PMCC) is a measure of the linear correlation/association between two variables X and Y, where the PC value $r = 1$ means a perfect positive correlation and the value $r = -1$ means a perfect negative correlation.

Pearson Correlation Formula:

$$r_{XY} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{8}$$

For example, we can use PC to find out whether human height and weight are correlated. Figure 3 visualize how Pearson Correlation r corresponds to the data correlation.

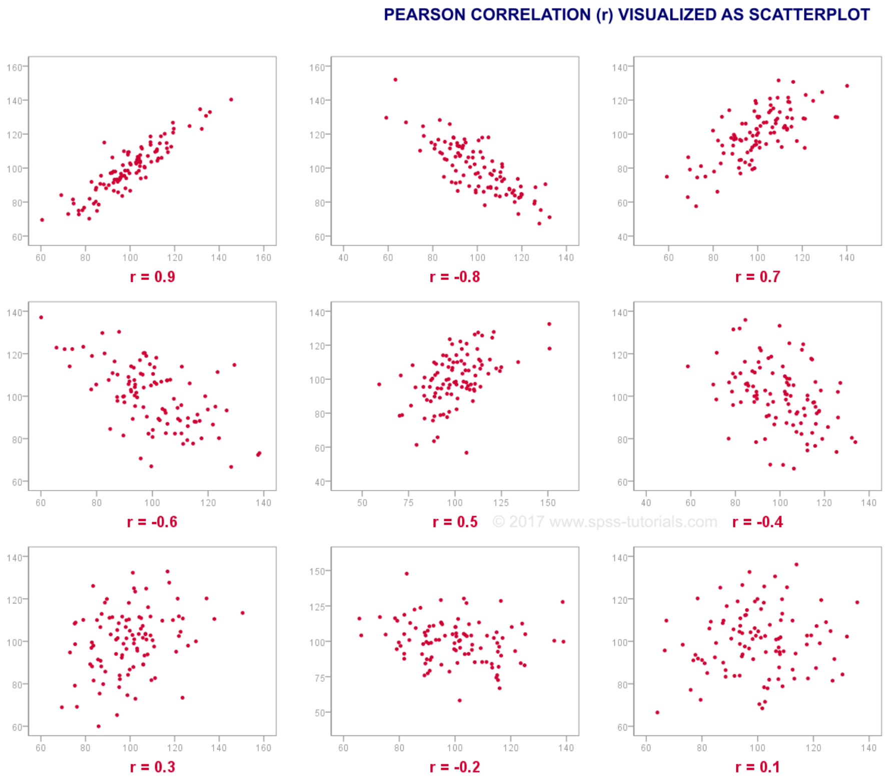


Figure 3: Source: spss-tutorials.com

**Caveats:** Correlations are very **sensitive** to outliers. Outliers are easily detected by scatter-plot and Correlation Matrix plot.

```
# scatter plot:
import matplotlib.pyplot as plt
plt.scatter(X, Y)
plt.show()
# correlation matrix plot:
plt.matshow(pd_dataframe.corr())
```

## 2.4 $R^2$, R-squared

$$R - squared = 1 - \frac{\text{explained variation}}{\text{total variation}}$$

R-squared is always between 0 and 100%:
- 0%: the model explains none of the variability of the response data around its mean.
- 100%: the model explains all the variability of the response data around its mean.

## 2.5 Kendall Rank Correlation (Kendall's tau coefficient)

## 2.6 Spearman Rank Correlation (Spearman's rho)

## 2.7 MAE: Mean Absolute Error

Mean absolute error (MAE) is a measure of difference between two continuous variables.

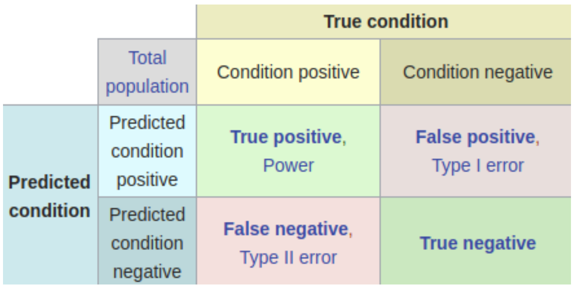
$$MAE = \frac{\sum_i^n |y_i - x_i|}{n} \tag{9}$$

## 2.8 MAPE: Mean Absolute Percentage Error

## 2.9 Classification Measurement:

$$\text{Recall/True Positive Rate} = \frac{\text{True Positive}}{\text{True Positive} + \text{True Negative}}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$



| | | True condition | |
|---|---|---|---|
| | Total population | Condition positive | Condition negative |
| Predicted condition | Predicted condition positive | True positive, Power | False positive, Type I error |
| | Predicted condition negative | False negative, Type II error | True negative |

Figure 4: Source: Wiki: Sensitivity and Specificity

**In an information retrieval context:**
Precision is the fraction of retrieved documents that are relevant to the query.

$$\text{Precision} = \frac{\{\text{Relevant Document}\} \cap \{\text{Retrieved Document}\}}{\{\text{Retrieved Document}\}}$$

Recall is the fraction of relevant documents that are successfully retrieved.

$$\text{Recall} = \frac{\{\text{Relevant Document}\} \cap \{\text{Retrieved Document}\}}{\{\text{Relevant Document}\}}$$

Explanation: When a class has 100 items, being able to retrieve 90 such items that all belong to this class implies 100% precision, yet the recall is $\frac{90}{100} = 90\%$.

## 2.10 Confusion Matrix (error matrix)

Use of `scikitlearn.metrics.confusion_matrix` can help visualize precision and recall for each class:

```
from sklearn.metrics import confusion_matrix
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
# Output:
```

```
# array([[2, 0, 0],
#        [0, 0, 1],
#        [1, 0, 2]])
```

## 2.11 $F_1$ Score and $F_\beta$ Score

$F_1$ Score is a measure of a test's accuracy, $F_1$ Score is the harmonic average of the precision and recall.

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Sometimes when you want to weigh precision or recall:

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

## 2.12 Jaccard Index

Jaccard index can show the similarity between two (or more) classes, or it can represent the proportion of shared items between two (or more) classes.

$$\text{Simple form:} \quad J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

$$\text{Generalized form:} \quad J(X, Y) = \frac{\sum_i min(x_i, y_i)}{\sum_i max(x_i, y_i)}$$

## 2.13 Common Problems:

### 2.13.1 Low Variance

When the variable has low variance data, we should consider drop it as it has no improvement on the model.

```
# Using sklearn.feature_selection.VarianceThreshold
# to remove features with low variance:
from sklearn.feature_selection import VarianceThreshold
VarianceThreshold(threshold = some_threshold).fit_transform(X)
```

### 2.13.2 High Correlation

Multicollinearity and collinearity (in multiple regression): a tutorial

todo

Scikit-learn website has example and guidance of using `SelectKBest`

**Chi-Squared Test(卡方检验) 含义: 自变量对因变量的相关性.**

explain

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
SelectKBest(chi2, k = 2).fit_transform(X_data, Y_target)
```

### 2.13.3 Missing Values

After examining the data, if there are still many missing data for some certain features, we might as well drop the feature variable. However, on a case-by-case scenario, the **drop threshold** can sometimes range from 30% to 60%.

Add other common problems, solutions, and examples.

todo

### 2.14 Recursive Feature Elimination (RFE) 递归消除特征法

RFE select features by recursively considering smaller and smaller sets of features. The size of the set of features becomes smaller because the algorithm prunes least important features in each recursion.

```python
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
estimator = SVR(kernel="linear")

# n_features_to_select: The number of features to select.
RFE(estimator, n_features_to_select = 2).fit_transform(X_data, Y_target)
```

### 2.15 SelectFromModel

1. L1-based feature selection

todo

2. Tree-based feature selection

Reference: scikit-learn tutorial

### 2.16 Analysis of Variance (ANOVA)

todo

**Null Hypothesis $\mathbf{H}_0$ & Alternative Hypothesis $\mathbf{H}_a$**

**Statistical Significance**tutorial

$F$**-test**

**P-Value**

$\alpha$**-Value - Level of Significance**

# 3 Vector Representation

## 3.1 Image

## 3.2 Properties for Text Vector Representations

- Same text have the same representation, distance of zero, maximum similarity.
- Able to compare distances between pairs of texts.
- Similarity/Distance should express the semantic comparison between texts.

## 3.3 One-Hot Encoding

对离散型特征进行 one-hot 编码是为了让距离的计算显得更加合理。
For all categorical/discrete features, represent them as multiple boolean features, one-hot. By using One-Hot encoding, we can calculate distance/proximity by ‘mapping’ the original data into a Euclidean space, this is called *embedding the vector in the Euclidean space*.

Listing 1: Python One-Hot Encoder

```
from sklearn import preprocessing

enc = preprocessing.OneHotEncoder()

# Given a dataset with three features and four samples, we let the encoder
# find the maximum value per feature and transform the data to a
# binary one-hot encoding.
enc.fit([[0,0,3],[1,1,0],[0,2,1],[1,0,2]])

array = enc.transform([[0,1,3]]).toarray()
# output: array([[1., 0., 0., 1., 0., 0., 0., 0., 1.]])
```

One way to measure similarity/distance between two text vectors is to calculate the Euclidean distance (also called L2 norm):

$$\text{Eulidean distance} = \sqrt{\sum_{i=1}(a_i - b_i)^2}$$

Listing 2: “Use numpy.linalg.norm”

```
import numpy as np
from numpy.linalg import norm
ax = np.array((0, 3, 5, 7))
bx = np.array((4, 0. 8, 0))

l2 = norm(ax - bx)
print(l2)
```

Listing 3: “Use scipy.spatial.distance”

```
from scipy.spatial import distance
l2_eu = distance.euclidean(ax, bx)
```

Also, another way is called **Cosine Similarity**, which is the cosine of the angle between 2 vectors, see Figure 5. When $cos\theta = 1$, two vectors has the maximum similarity, when $cos\theta = 0$,

Note: **Limitation in sklearn's `OneHotEncoder`:** CANNOT process `string` type features!

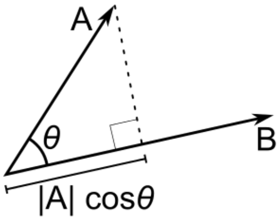


Figure 5: Projection of vector to vector B and cosine angle $cos\theta = \frac{\vec{a} \cdot \vec{b}}{||a||||b||}$

**Solution 1:** Use `pandas.get_dummies(data)` to convert categorical variable into dummy/indicator variables.
**Solution 2:** `sklearn.feature_extraction.DictVectorizer().fit_transform()` can convert a `Dict` type to matrix where categorical types are represented in 0/1.

Example of `DictVectorizer`:

```
import pandas as pd
from sklearn.feature_extraction import DictVectorizer
data =
  pd.DataFrame({'name':['Tom','Andy','David'],'age':[20,21,22],'height':[175,165,180]})
vec_data =
  DictVectorizer(sparse = False).fit_transform(data.to_dict(orient='record'))
print(arr)
# output:
# [[20. 175. 0. 0. 1.]
#  [21. 165. 1. 0. 0.]
#  [22. 180. 0. 1. 0.]]
```

## 3.4 TF, TF-IDF

TF: Term-Frequency
TF-IDF: Term-Frequency Inverse Document-Frequency

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{10}$$

, where
TF$(t, d)$ is the number of occurrences of term $t$ in document $j$,
IDF$(t)$; Inverse Document Frequency, is measured by $\left(\log \frac{1+n}{1+\text{TF}(d,t)} + 1\right)$, where the +1 is for smoothing and $n$ means the number of documents that has the term $t$.

TF-IDF assigns weight $w$ to a term $t$ in document $d$.

- $w$ is high, when $t$ occurs many times within a small set of documents.
- $w$ is low, when $t$ occurs fewer times in a document, or, when $t$ occurs in many documents.
- $w$ is at its lowest, when $t$ appears virtually in all documents, i.e., ‘a’, ‘the’, etc.

### 3.4.1 Variant of TF-IDF: WF-IDF

**Idea:** 20 occurrences of a term $t_1$ in a document may not necessarily carry 20 times more significance of a term $t_2$ that has only 2 occurrences.
**Solution:** Use logarithm:

$$\text{WF-IDF}(t, d) = \text{WF}(t, d) \times \text{IDF}(t) \tag{11}$$

, where

$$\mathrm{WF}(t,d) = \begin{cases} 1 + \log \mathrm{TF}(t,d) & \text{if } \mathrm{TF}(t,d) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

More available at Christ Manning's book Introduction to Information Retrieval, *Chapter 6.4 Variant tf-idf functions*.

### 3.5 Word2Vec

Word2Vec load in python http://mccormickml.com/2016/04/12/googles-pretrained-wo todo odel-

Word2Vec Resource: http://mccormickml.com/2016/04/27/word2vec-resources/ resource

### 3.6 GloVe, Global Vectors for Word Representation

### 3.7 VGG-ish

### 3.8 Hidden Markov Model & Maximum Entropy Model

Write in another statistical language model review note.

# 4 Dimension Reduction

Seven Techniques for Data Dimensionality Reduction https://www.knime.com/blog/seven-techniques-for-data-dimensionality-reduction Dimensionality Reduction Algorithms: Strengths and Weaknesses https://elitedatascience.com/dimensionality-reduction-algor

todo

**Questions to address in this section:** When there are too many variables, do I need to explore each and every variable? When using Random Forest, the execution time is too much due to the large number of features. Common ML Algorithms to identify the most significant variables.

**Benefits:**

- Compress data to reduce storage space.
- Reduce time required for computations - Less dimensions leads to less computation.
- Take care of multi-collinearity that improves the model performance.
- Allow data visualization (talks about in the next section).

## 4.1 Brief Intro: The Curse of Dimensionality

**In short:** When the number of features is huge relative to the number of observations in the dataset, *certain* algorithms struggle to train effective models.

Talks about exceptions.

**Mathy Explain:** Joan Bruna's Inference and Representation

Lecture 1

## 4.2 Random Forest (Ensemble Trees):

## 4.3 Backward Feature Elimination:

## 4.4 Forward Feature Construction:

## 4.5 Linear Discriminant Analysis[2] (LDA) 线性判别分析:

LDA[3] Idea: Create new axis that maximizes the distance between the means while minimizing the scatter.

What we need to do is to find a transformation $T$ that can project samples in $R^d$ space into $R^1$ space (dimension reduction):

$$y' = T(x) = w^T x$$

, where $w^T x$ is dot product between two vectors $w$ and $x$:

$$w^T x = w \cdot x = ||w|| \cdot ||x|| \cdot cos\theta$$

Here $||\mathbf{x}||\mathbf{cos}\theta$ represents the the scaler length that vector $x$ projects onto the vector $w$.

**Example steps for a two-class LDA separation:**
1. Find the center point (the mean) of class A that's projected into a lower dimension:

$$\hat{\mu}_i = T(\mu_i) = w^T \mu_i, \text{where } \mu_i \text{ is the original data center/mean for class i,}$$

$$\mu_i = \frac{1}{N} \sum_{x \in D_i} x$$

2. Calculate the variance of the projected data:

$$\hat{s}_i = \sum_{y \in Y_i} (y - \hat{\mu}_i)$$

[2] Also called Fisher's Linear Discriminant. Ronald Fisher, 1936.

[3] Latent Dirichlet Allocation also has the abbreviation of LDA.

3. Construct the LDA loss function:

$$J(w) = \frac{|\hat{\mu}_1 - \hat{\mu}_2|^2}{\hat{s}_1^2 + \hat{s}_2^2}$$

, where we call the $|\hat{\mu}_1 - \hat{\mu}_2|^2$ as *Between-class scatter*, and the $\hat{s}_1^2 + \hat{s}_2^2$ as *Within-class scatter*.
4. In order to obtain the maximum separat-ibility, we want the numerator $|\hat{\mu}_1 - \hat{\mu}_2|^2$ be as large as possible while having the denominator $\hat{s}_1^2 + \hat{s}_2^2$ as small as possible.

$$\text{Best of } w = \hat{w} = \arg\max_w J(w)$$

Figure 6 illustrates the use of different $w$ result in different *Between-class scatter* and *Within-class scatter* that lead to different separat-ibility.

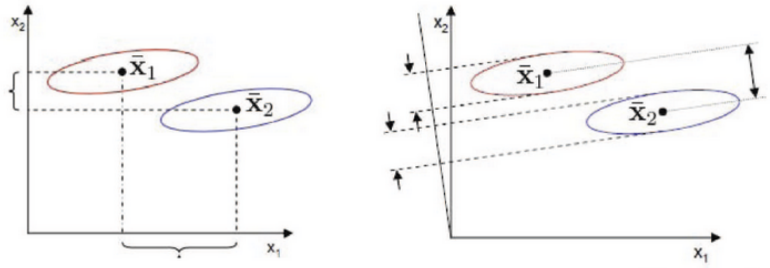


Figure 6: The left figure has larger *Between-class scatter* yet has worse separatibility. The right figure takes both factors into account.

### Find the optimal w

1. According to the nature of dot product: $\mathbf{a^T} \cdot \mathbf{b} = \mathbf{b^T} \cdot \mathbf{a}$, we can rewrite $|\hat{\mu}_1 - \hat{\mu}_2|^2$.

$$\begin{aligned} |\hat{\mu}_1 - \hat{\mu}_2|^2 &= (w^T\mu_1 - w^T\mu_2)^2 \\ &= (w^T \cdot (\mu_1 - \mu_2))^2 \\ &= w^T \cdot (\mu_1 - \mu_2) \cdot w^T \cdot (\mu_1 - \mu_2) \\ &= w^T \cdot (\mu_1 - \mu_2) \cdot (\mu_1 - \mu_2)^T \cdot w \\ \text{Let Between-class scatter matrix } \mathbf{S_B} &= (\mu_1 - \mu_2) \cdot (\mu_1 - \mu_2)^T \\ \text{We can have: } |\hat{\mu}_1 - \hat{\mu}_2|^2 &= w^T \cdot \mathbf{S_B} \cdot w \end{aligned} \tag{13}$$

2. Now rewrite $\hat{s}_1^2 + \hat{s}_2^2$.
Given that:

$$\hat{s}_i^2 = \sum_{y \in Y_i} (y - \hat{\mu}_i)^2 = \sum_{x \in X_i} (w^T x - w^T \hat{\mu}_i)^2$$

We can use the last step in Eq.13:

$$(w^T x - w^T \hat{\mu}_i)^2 = w^T((x - \mu_i)(x - \mu_i)^T)w$$

Therefore, we can have:

$$\hat{s}_i^2 = \sum_{x \in X_i} w^T((x - \mu_i)(x - \mu_i)^T)w = w^T(\sum_{x \in X_i} (x - \mu_i)(x - \mu_i)^T)w$$

Thus:

$$\hat{s}_1^2 + \hat{s}_2^2 = \underbrace{w^T(\sum_{x\in X_1}(x-\mu_i)(x-\mu_i)^T)w}_{x \text{ in } X_1,\ i=1} + \underbrace{w^T(\sum_{x\in X_2}(x-\mu_i)(x-\mu_i)^T)w}_{x \text{ in } X_2,\ i=2}$$

$$\text{Let Within-class scatter matrix } S_W = S_1 + S_2, \text{ where } S_i = \sum_{x\in X_i}(x-\mu_i)(x-\mu_i)^T \tag{14}$$

$$\begin{aligned}\hat{s}_1^2 + \hat{s}_2^2 &= w^T \cdot (S_1 + S_2) \cdot w \\ &= w^T \cdot S_W \cdot w\end{aligned}$$

Hence, from Eq.13 and Eq.14, we can have the LDA loss function $J(w)$ re-written as:

$$J(w) = \frac{|\hat{\mu}_1 - \hat{\mu}_2|^2}{\hat{s}_1^2 + \hat{s}_2^2} = \frac{w^T S_B w}{w^T S_W w} \tag{15}$$

**Use of Lagrange Multiplier to find the optimal $\widehat{w}$ :**

1. Set constraint to $w$: let $\underbrace{w^T S_W w}_{\text{denominator of } J(w)} = 1$.

Thus we want to find:

$$w = \arg\max_w \underbrace{w^T S_B w}_{J_B(w)}, \text{ given that } \phi(w) = w^T S_W w - 1 = 0$$

Construct Lagrangian function:

$$F(w) = J_B(w) - \lambda(w^T S_W w - 1) \tag{16}$$

2. We need to resolve:

$$\begin{cases} \frac{dF}{dx} &= 0, \\ \phi(w) &= 0. \end{cases} \tag{17}$$

, where $\frac{dF}{dx} = (\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \cdots, \frac{\partial F}{\partial x_n})^T$

To-be-continued.

todo

## 4.6 Principal Component Analysis (PCA) 主成分分析法:

### 4.6.1 About

PCA[4] uses Orthogonal Transformation to get sets of linearly uncorrelated variables; sets of pairs of eigenvalues and eigenvectors, as principal components. The first (largest) principal component holds the greatest variance in the data.

PCA is an useful tool in exploratory data analysis: dimension reduction, relatedness between populations.

### 4.6.2 Prerequisite

1. **Empirical Mean:** Let $x_1, x_2, \cdots, x_n$ be a st of d-dimensional real-valued data, the empirical mean (sample mean) is defined as:

$$\overline{x}_n := \frac{1}{n}\sum_{i=1}^{n} x_i \tag{18}$$

[4] invented by Karl Pearson, who also laid the foundations of statistical hypothesis testing theory, i.e., the Pearson's chi-squared test.

2. **Empirical Variance**; $S_n^2$, measures the variability of the dataset:

$$S_n^2 := \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2 \tag{19}$$

3. **Empirical Standard Deviation** is $S_n = \sqrt{S_n^2}$.

4. **Variability** refers to how spread out a group of data is, basically, variability is the same idea as variance and standard deviation.

5. **Standardization**: $y_1, y_2, \cdots, y_n$ are transformed from $x_1, x_2, \cdots, x_n$ so that $Y$ are centered at origin and have sample variance 1:

$$y_i = \frac{x_i - \overline{x}_n}{S_n}, \ 1 \le i \le n \tag{20}$$

$$\overline{y}_n = \frac{1}{n}\sum_{i=1}^{n} y_i = 0, \quad \text{and } \mathrm{Var}(\{y_i\}) = 1 \tag{21}$$

6. **Empirical cdf:**

$$F_{X,n}(x) = \frac{1}{n}\sum_{i=1}^{n} 1_{x_i \le x} \tag{22}$$

7. **Empirical Covariance:** Refers to previous section: 2.2 about covariance and covariance matrix.

$$\mathrm{Cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)(y_i - \overline{y}_n) \tag{23}$$

When $Y = X$, $\mathrm{Cov}(X,X) = \mathrm{Var}(X)$.

8. **Covariance Matrix:**

$$\Sigma_n := \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \overline{x}_n)(x_k - \overline{x}_n)^\intercal \tag{24}$$

For $x_i \in \mathbb{R}^d, d$ features, $\Sigma_n \in \mathbb{R}^{d\times d}$.

9. **Empirical Correlation Coefficient:**

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{S_n(X)\cdot S_n(Y)}$$

Proof of $\rho(X,Y) \in [-1,1]$ by Cauchy-Schwarz Inequality:

$$\left|\sum_{i=1}^{n} a_i b_i\right| \le \sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2} \tag{25}$$

Thus, when $a$ and $b$ has the highest correlation; $Y = X$, $\rho = 1$.

**Empirical Variance in a certain direction:** Let $v$ be a unit-norm vector aligened with a direction of interest. Consider the variance of $v^\intercal x_i$. The sample mean and sample variance:

$$\frac{1}{n}\sum_{i=1}^{n} v^\intercal x_i = v^\intercal \overline{x}_n$$

$$
\begin{aligned}
S_n^2(\{v^\intercal x_i\}) &= \frac{1}{n-1}\sum_{i=1}^{n}(v^\intercal x_i - v^\intercal \overline{x}_n)^2 \\
&= \frac{1}{n-1}\sum_{i=1}^{n}(v^\intercal x_i - v^\intercal \overline{x}_n)(v^\intercal x_i - v^\intercal \overline{x}_n) \\
&= \frac{1}{n-1}\sum_{i=1}^{n} v^\intercal (x_i - \overline{x}_n)(x_i^\intercal - \overline{x}_n^\intercal)v \\
&= v^\intercal \underbrace{\left(\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)(x_i^\intercal - \overline{x}_n^\intercal)\right)}_{\text{covariance matrix}} v \\
&= v^\intercal \Sigma_n v
\end{aligned}
\tag{26}
$$

Now, consider maximize the **sample variance** $S_n^2(v^\intercal x_i)$ in the direction of $v$. The largest $v$ will be the one which has the greatest variance among the data.

### 4.6.3 Steps walk-through with a naive 2D example:

Example:

Given a set of points in the Cartesian coordinate system:

$$
C = \begin{bmatrix} 1 & 1 & 2 & 4 & 2 & \leftarrow \text{X-axis} \\ 1 & 3 & 3 & 4 & 4 & \leftarrow \text{Y-axis} \end{bmatrix}
$$

In this example, our purpose is to find a dimension that has the largest variance for this dataset, which in this two dimensional Cartesian space, will be some $\vec{\mathbf{v}} = \begin{bmatrix} x_v \\ y_v \end{bmatrix}$.

We calculate the sample mean for X is $\overline{X} = 2$ and sample mean for Y is $\overline{Y} = 3$, thus after standardization, we have (Shown in Figure.7):

$$
C = \begin{bmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{bmatrix}
$$

The covariance matrix for $C$ is:

$$
Sigma_n = \frac{1}{m=4} \cdot C \cdot C^\intercal = \frac{1}{m}\begin{bmatrix} \sum_{i=1}^m x_i^2 & \sum_{i=1}^m x_i \cdot y_i \\ \sum_{i=1}^m x_i \cdot y_i & \sum_{i=1}^m y_i^2 \end{bmatrix} = \begin{bmatrix} \mathrm{Var}(X) & \mathrm{Cov}(X,Y) \\ \mathrm{Cov}(X,Y) & \mathrm{Var}(Y) \end{bmatrix}
\tag{27}
$$

In Python:

```
import numpy as np
c = np.array([[-1, -1, 0, 2, 0], [-2, 0, 0, 1, 1]])
np.cov(c)
# output:
# array([[1.5, 1. ],
#        [1. , 1.5]])
# Same as: (1/4) * np.matmul(c, c.T)
```

As shown in the covariance matrix $\Sigma_n$, it has the variance and covariance of $X$ and $Y$ in the Cartesian space. Thus, if we transform $C$ from Cartesian space to another 2-dimensional coordinate basis, we can have a new $\Sigma_n^{new}$ with other variance and covariance values.

Assume the new basis $P = \begin{bmatrix} P_x \\ P_y \end{bmatrix}$ refers to a new 2-dimensional coordinate space. Data $X$ in the current Cartesian setup will be transformed to $Y = PX$. Thus, the new covariance matrix for $Y$ is:

$$\Sigma_Y = \frac{1}{m} YY^\intercal = \frac{1}{m}(PX)(PX)^\intercal$$

, which can be rewritten to:

$$\begin{aligned} \Sigma_Y &= \frac{1}{m} PXX^\intercal P^\intercal \\ &= P\Big(\frac{1}{m}XX^\intercal\Big)P^\intercal \\ &= P\Sigma_n P \end{aligned}$$

, where the $\Sigma_n$ is the original covariance matrix in Eq.27.

Thus, our stated purpose now becomes to find the $P$ so as to have a new covariance matrix $\Sigma_Y$ such that the diagonal values (the variances of each transformed dimension itself) are maximized while the covariances become zero. In Eq.26, the variance in direction $\vec{v}$ that maximize its variance among the other dimension of the dataset, is going to be the first principal component of the data.

As for the example; graph shown in Figure.7, transforms the regular Cartesian coordinate to a new coordinate system (coordinate lines colored with green and brown), where x-axis is in the direction of $P_x = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ and y-axis is in the direction of $P_y = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$. A point in $C$ will be changed in the new coordinate system:

$$\begin{bmatrix} P_x \\ P_y \end{bmatrix} \cdot \begin{bmatrix} C_{ix} \\ C_{iy} \end{bmatrix} = \begin{bmatrix} C_i^{new} \\ C_i^{new} \end{bmatrix}$$

For example, the Blue dot in Fig.7; $(2,1)$ in the new coordinate system is

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{3}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

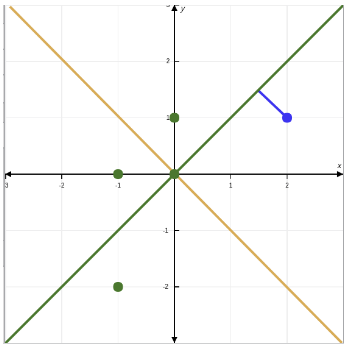


Figure 7: Points in $C$ plotted in Cartesian. Green line refers to the new x-axis, Yellow line refers to the new y-axis. (Spoiler alert: the Green line is actually in the direction of the first principal component.)

Now, in order to find the new covariance matrix $\Sigma_n^{\text{new}}$, we perform **spectral decompo-**

**sition**, such that values in the covariance matrix become zeros besides the diagonal:

$$\begin{aligned}\Sigma_n &= \frac{1}{m=4} \cdot C \cdot C^\intercal = \begin{bmatrix} \frac{6}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{6}{5} \end{bmatrix} \\ &= Q\Lambda Q^\intercal \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 \\ 0 & \frac{2}{5} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}\end{aligned}$$

Thus, we have two pairs of eigenvalue and eigenvectors:

$$\lambda_1 = 2,\ C_1 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}) \tag{28}$$

$$\lambda_2 = \frac{2}{5},\ C_2 = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}) \tag{29}$$

Thus, we have the greatest eigenvalue $\lambda_1 = 2$, and its corresponding eigenvector as the principal component, the new basis $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$.

### 4.6.4 For higher D-dimensional space:

The above example is in a 2D space. When we have a D-dimensional space; $D > 2$, we follows the same steps to acquire the principal components, except that we are now looking for $K$ mutual-orthogonal unit vectors in $K$ basis-es, $(K \leq D)$, where the data has the largest possible variance in the $K^{th}$ basis.

That is, given data D-dimensional space $X$, we calculate its covariance matrix $\Sigma$, then perform spectral decomposition:

$$\begin{aligned}\Sigma &= Q\Lambda Q^\intercal \\ &= \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{bmatrix} \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix}^\intercal \end{aligned} \tag{30}$$

$$, \text{where } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \text{ and } \Sigma u_i = \lambda_i u_i, \text{ and } u_i \perp u_j (1 \leq i \neq j \leq n)$$

Choose the top K pairs of eigenvector and eigenvalues to be the principal components for analysis. As $\lambda_1$ is the greatest, $u_1$ is the direction where the data points spread out the most.

Add sample code

### 4.6.5 Limitations

PCA depends on the scaling of the variables, which makes it sensitive to scaling.

PCA is also weak in dealing with outliers. Use 'Weighted PCA' to increase robustness of PCA.

PCA cuts out linear correlation, for data with higher correlation, we use Kernel PCA.

PCA is an non-parametric technique, which lacks of optimization tricks.

Many other variations of PCA: https://arxiv.org/pdf/1403.2877.pdf

### 4.6.6 Compare with LDA

LDA, the input training data have labels; however, PCA does not, which makes PCA an unsupervised learning algorithm.

PCA 是为了让映射后的样本具有最大的发散性；而 LDA 是为了让映射后的样本有最好的分类性能。

- PCA reduces dimensions by focusing on the genes with the most variation.
- LDA focuses on maximizing the separat-ibility among known categories.

A Comparison of PCA and LDA

todo

### 4.7 Independent Component Analysis (ICA) :

# 5 Data Visualization

## 5.1 Visualization Tools

Show sample code snippets here.
Use of matplotlib, seaborn, and Tableau

## 5.2 Big-Data Tools

Real-time Streaming Data, etc.
Show sample code snippets here.

# 6 Features in Neural Network

Feature Visualization (Distill) - How neural networks build up their understanding of images https://distill.pub/2017/feature-visualization/

read

通过深度学习来进行特征选择, Unsupervised Feature Learning.

todo

## 7 Further Readings and References

- Scikit-Learn Preprocessing Data Scikit-learn tutorial
- Tf-idf weighting Introduction to Information Retrieval, Christopher D. Manning, Stanford NLP
- 特征选择的方法: 知乎问题
- 二次型矩阵 $w^T Sw$
  知乎: 二次型的意义是什么？有什么应用？
  Equation $w_1^2 + w_2^2 - w_1 w_2 = 1$ can be re-written as:

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = 1$$

*Disclaimer: Many examples and figures may not be referenced or referenced properly, if any citation is missed or incorrect, please contact me.*

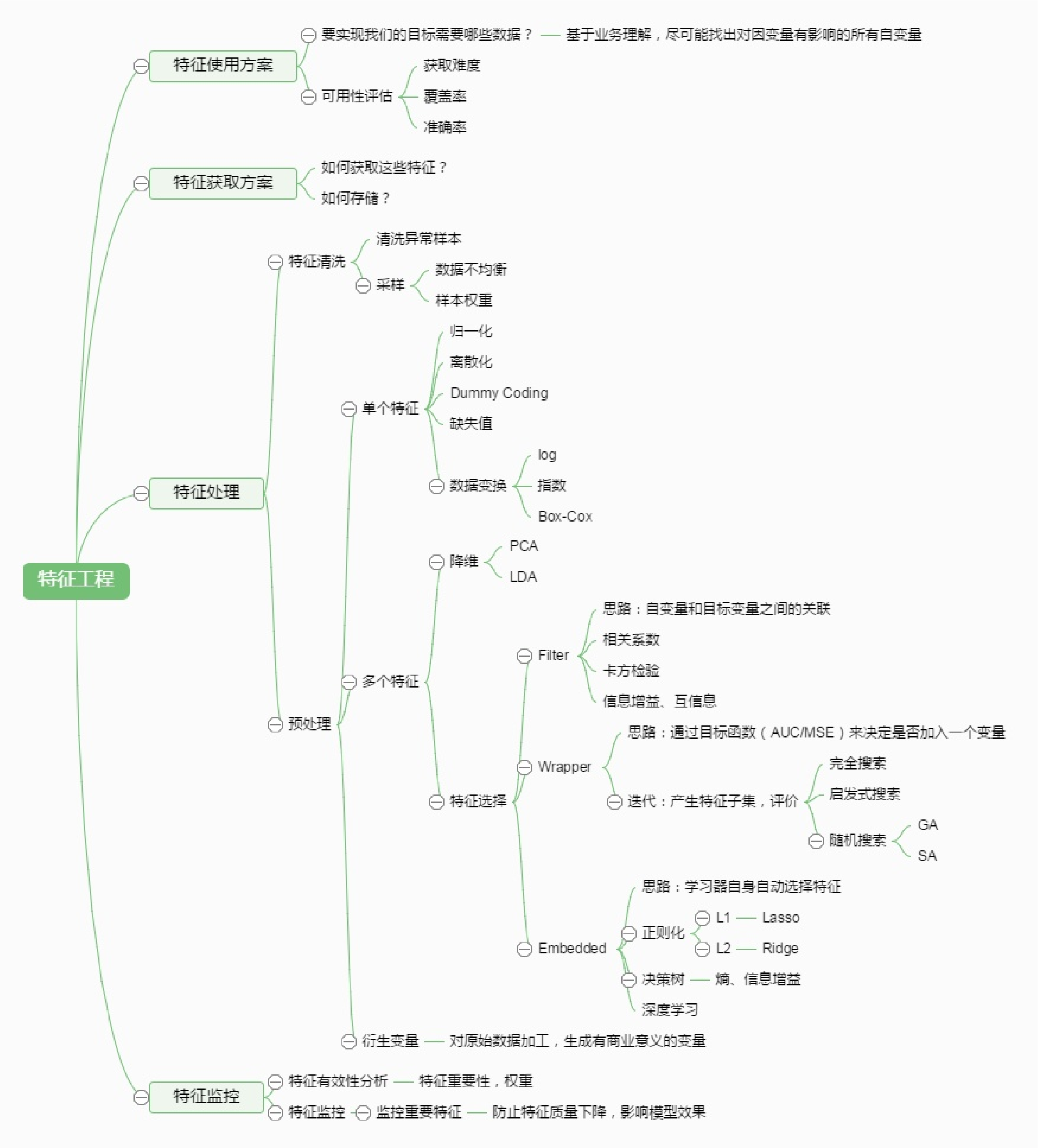


Figure 8: Feature Engineering Structure 知乎链接