MEME19403: Data Pre-processing and Dimension Reduction

Dr Liew How Hui

June 2021

Revision

According to `https://en.wikipedia.org/wiki/Data_analysis`, data analysis involve:

- inspecting data — extract and store in Python as Pandas's DataFrame & EDA (last two weeks)
- cleansing data — isnull, dropna, ... (last week)
- transforming data (today)
  - Features & label encoding: categorical data → numeric
  - Pre-processing: "scaling" of the numeric data
  - Dimension reduction: When $n \ll p$.
- modelling data (following weeks)

Goal: discovering useful information, informing conclusions, and supporting decision-making.
Practice: Apply them in your assignment.

Outline

1. **Features & Label Encoding**
   - One-Hot Encoding
   - Ordinal Encoding
   - Discretisation
2. **Pre-processing**
3. **Linear Dimension Reduction: PCA**

Feature Encoding

Most predictive models only deal with numbers of certain form, so when categorical data are met, they need to be converted to "numbers" (usually integers). When the categorical data is nominal, the encodings below are popular:

- pd.get_dummies, sklearn.preprocessing.OneHotEncoder(?)

When the categorical data is ordinal, the encodings below are popular:

- sklearn.preprocessing.{OrdinalEncoder, LabelEncoder}

Outline

1. **Features & Label Encoding**
   - One-Hot Encoding
   - Ordinal Encoding
   - Discretisation
2. **Pre-processing**
3. **Linear Dimension Reduction: PCA**

One-Hot Encoding

```
pd.get_dummies(data, prefix=None,
  prefix_sep='_', dummy_na=False, columns=None,
  sparse=False, drop_first=False, dtype=None)
```

Example:

```
1  import pandas as pd
2  Default = pd.read_excel("Default.xlsx", index_col=0) # Need openpyxl
3  print("Loaded Excel Data ...\n", Default)
4  X = pd.get_dummies(Default[['student','balance','income']],
5                     drop_first=True)
6  Y = Default['default'].apply(lambda x: 0 if x=='No' else 1)
7  print("One-hot encoded X...\n", X)
8
9  import statsmodels.api as sm
0  X = sm.add_constant(X)
1  model = sm.Logit(Y, X)
2  train = model.fit()  # Use the method .fit() to train model
3  print(train.params)
4  print(train.summary())
```

One-Hot Encoding (cont)

```
sklearn.preprocessing.OneHotEncoder(*,
  categories='auto', drop=None, sparse=True,
  dtype=<class 'numpy.float64'>,
  handle_unknown='error')
```

Useful parameters:

- drop = None : retain all features
- drop = 'first' : drop the first category in each feature. It is using in logistic regression.
- drop = 'if_binary' : features with < 2 categories are not changed.
- handle_unknown = 'error' : Python with raise error and stop running
- handle_unknown = 'ignore' : the feature will be set to all zeros

One-Hot Encoding (cont)

Example (using OneHotEncoder):

```
1  import pandas as pd
2  Default = pd.read_excel("Default.xlsx", index_col=0) # Need openpyxl
3  print("Loaded Excel Data ...\n", Default)
4  from sklearn.preprocessing import OneHotEncoder
5  encoder = OneHotEncoder(drop='first')
6  Student = encoder.fit_transform(Default[['student']])
7  encoder.categories_
8  X = pd.DataFrame({'student_Yes':Student.toarray().flatten()},
9                   index=Default.index).join(
0                   Default[['balance','income']])
1  Y = Default['default'].apply(lambda x: 0 if x=='No' else 1)
2  print("One-hot encoded X...\n", X)
3
4  import statsmodels.api as sm
5  X = sm.add_constant(X)
6  model = sm.Logit(Y, X)
7  train = model.fit()  # Use the method .fit() to train model
8  print(train.params)
9  print(train.summary())
```

Outline

1. **Features & Label Encoding**
   - One-Hot Encoding
   - Ordinal Encoding
   - Discretisation
2. **Pre-processing**
3. **Linear Dimension Reduction: PCA**

Ordinal Encoding

Although One-hot encoding is 'logical' in theory, but it is expensive in computation. It is not too unreasonable to encode (semi-)ordered data as 'integers' using 'ordinal encoding'.

According to `https://github.com/scikit-learn/scikit-learn/pull/12866` and `https://github.com/scikit-learn/scikit-learn/issues/5442`, sklearn's decision tree does not yet support categorical data properly. If the categorical data is not ordinal, this is not good — the splits in the decision tree may not make sense. Using a OneHotEncoder is the only current valid way, allowing arbitrary splits not dependent on the label ordering, but is computationally expensive. (Ref: `https://stackoverflow.com/questions/38108832/passing-categorical-data-to-sklearn-decision-tree`)

Ordinal Encoding (cont)

Both `OrdinalEncoder` and `LabelEncoder` can be used as for ordinal encoding. The difference is in the ability to deal with the shape of the data:

- `OrdinalEncoder`: handles 2D data with the shape (n_samples, n_features)
- `LabelEncoder`: **only** handles 1D data with the shape (n_samples,)

`OrdinalEncoder` is intended for "features" (often a 2D array), whereas `LabelEncoder` is for the "target variable" (often a 1D array).

Ordinal Encoding (cont)

Another difference between the encoders is the name of their learned parameter:

- `OrdinalEncoder` learns `categories_`
- `LabelEncoder` learns `classes_`

Ordinal Encoding (cont)

Consider a data with unlabelled gender data.

```
import pandas as pd
# https://www.sheffield.ac.uk/mash/statistics/datasets
# The data were collected on 200 high school students and are scores on various tests, including a
#
# Id         | Identity of the student      | Nominal
# gender     | Gender (0: Male, 1:Female)   | Binary
# ice_cream  | Favourite Flavour (1: Vanilla, 2: Chocolate, 3: Strawberry) | Nominal
# video      | Score on the video game      | Scale/Continuous
# puzzle     | Score on the puzzle          | Scale/Continuous
#
df = pd.read_csv("Ice_cream_R.csv")
print(df.dtypes)
X_train = df[['gender', 'video', 'puzzle']]
y_train = df['ice_cream']

#
# independent variables: female, video, puzzle
# dependent variable: ice_cream
#
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pylab as plt
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
print(model)
plot_tree(model)
plt.show()
```

The Python program runs with errors.

Ordinal Encoding (cont)

Getting familiar with Python's Ordinal Encoding with the famous 'play tennis/golf" data with many categorical data:

```
import pandas as pd
df = pd.read_csv('playtennis.csv')
from sklearn.preprocessing import OrdinalEncoder,\
                                  LabelEncoder
X = df[['outlook','temperature','humidity','wind']]
enc = OrdinalEncoder()
X_ordinal = enc.fit_transform(X)
print("Feature categories:\n", enc.categories_)
out_enc = LabelEncoder()
y = out_enc.fit_transform(df['playtennis'] )
print("Output classes:\n", out_enc.classes_)
print(out_enc.inverse_transform([0,1,1,0]))
```

Ordinal Encoding (cont)

Introducing label encoding to get the program running.

```
import pandas as pd
# https://www.sheffield.ac.uk/mash/statistics/datasets
df = pd.read_csv("Ice_cream_R.csv")
print(df.dtypes)

from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()  # Since 1 column only
df.gender = enc.fit_transform(df.gender)
print(enc.classes_)
print(df.dtypes)

X_train = df[['gender', 'video', 'puzzle']]
y_train = df['ice_cream']

from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pylab as plt
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
print(model)
plot_tree(model)
plt.show()
```

Outline

1. **Features & Label Encoding**
   - One-Hot Encoding
   - Ordinal Encoding
   - Discretisation
2. **Pre-processing**
3. **Linear Dimension Reduction: PCA**

Discretisation

Some predictive models works much better with discrete data (e.g. Decision trees, Bayesian models). Discretisation (or quantisation or binning) provides a way to partition **continuous features** into **discrete values** using ideas similar to constructing histograms for continuous data. Histograms focus on counting features which fall into particular bins, whereas discretisation focuses on assigning feature values to these bins. However, we need to beware the introduction nonlinearity to linear models with discretisation.

Discretisation & Interaction Effects

Sklearn provides discretisation (for applications in trees and neural networks) & interaction for linear models:

- KBinsDiscretizer: implements different binning strategies. The 'uniform' strategy uses constant-width bins. The 'quantile' strategy uses the quantiles values to have equally populated bins in each feature. The 'kmeans' strategy defines bins based on a k-means clustering procedure performed on each feature independently.
- LabelBinarizer, MultiLabelBinarizer
- PolynomialFeatures: Generate polynomial and interaction features. For application in statistics to generate interaction effect.

Discretisation (cont)

Example (k-bin uniform partitioning):

```
from sklearn.preprocessing import KBinsDiscretizer
X = np.array([[ -3., 5., 15 ], [  0., 6., 14 ], [  6., 3., 11 ]])
enc = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
enc.fit(X)
Xt = enc.transform(X)
```

- Column 1: Cut interval $[col.min, col.max] = [-3, 6]$ into 3 uniform pieces:
  $[-3, -3 + \frac{6-(-3)}{3} = 0) \cup [0, 3) \cup [3, 6]$
- Column 2: Cut interval $[3, 6]$ into
  $[3, 4) \cup [4, 5) \cup [5, 6]$
- Column 3: Cut interval $[11, 15]$ into
  $[11, 12\frac{1}{3}) \cup [12\frac{1}{3}, 13\frac{2}{3}) \cup [13\frac{2}{3}, 15]$

Discretisation (cont)

To convert the data back into the original feature space (i.e. decoding), the “inverse_transform“ function converts the binned data into the original feature space. Each value will be equal to the mean of the two bin edges.

```
print("The partitioning edges:\n", enc.bin_edges_)
enc.inverse_transform(Xt)
```

We won’t get back the original data but the ‘mean’ of the original data.
Note that `https://scikit-learn.org/stable/modules/preprocessing.html` provides an example with different bins, i.e. n_bins=[3,2,2] and non-uniform partition using pd.qcut.

Discretisation (cont)

LabelBinarizer and MultiLabelBinarizer are discretisations intended for the supervised learning target variable in **multiclass** classification.

LabelBinarizer can be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels. It is used in image recognition of single object.

MultiLabelBinarizer converts a collection of group labels to the indicator matrix format. It is used in image recognition of multiple objects.

Discretisation (cont)

Example (LabelBinarizer):

```
>>> enc = LabelBinarizer()
>>> y=['apple','orange','durian','orange','durian']
>>> enc.classes_
array(['apple', 'durian', 'orange'], dtype='<U6')
>>> enc.fit_transform(y)
array([[1, 0, 0],
       [0, 0, 1],
       [0, 1, 0],
       [0, 0, 1],
       [0, 1, 0]])
>>> enc.inverse_transform(np.array([[1,0,1,0]]))
[('apple', 'orange')]
```

Discretisation (cont)

Example (MultiLabelBinarizer):

```
>>> enc = MultiLabelBinarizer()
>>> y=[['apple','orange','durian'],
       ['apple','pear'],
       ['orange','durian']]
>>> enc.classes_
>>> enc.fit_transform(y)
array([[1, 1, 1, 0],
       [1, 0, 0, 1],
       [0, 1, 1, 0]])
>>> enc.inverse_transform(np.array([[1,0,1,0]]))
[('apple', 'orange')]
```

Outline

1. Features & Label Encoding
   - One-Hot Encoding
   - Ordinal Encoding
   - Discretisation
2. **Pre-processing**
3. Linear Dimension Reduction: PCA

Pre-processing

The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

- StandardScaler (or scale, for PCA)
- MinMaxScaler, MaxAbsScaler
- RobustScaler: Scale features using statistics that are robust to outliers
- Normalizer (or normalize, norm='l1', 'l2', 'max'): Normalise samples individually (row) to unit norm. Use in text classification / clustering.
- Binarizer: Binarize data (set feature values to 0 or 1) according to a threshold

Pre-processing (cont)

Less popular / more specialised scalers(?):

- QuantileTransformer: Transform features using quantiles information
- PowerTransformer: Apply a power transform featurewise to make data more Gaussian-like
- FunctionTransformer
- KernelCenterer: Centre a kernel matrix

Pre-processing (cont)

Example: MinMaxScaler, MaxAbsScaler and StandardScaler

Practice: scaleminmax.py

Pre-processing (cont)

```
1 from sklearn.preprocessing import Normalizer
2 X = [[4, 1, 2, 2], [1, 3, 9, 3], [5, 7, 5, 1]]
3 enc = Normalizer()
4 enc = enc.fit(X)
5 Xenc = enc.transform(X)
```

Regarding the calculation in Xenc:

$$row_1 = \frac{[4, 1, 2, 2]}{\sqrt{4^2 + 1^2 + 2^2 + 2^2}} = [0.8, 0.2, 0.4, 0.4]$$

Application: Normalise text `https://scikit-learn.org/stable/auto_examples/text/plot_document_clustering.html`

Discretisation (cont)

```
Binarizer(*, threshold=0.0, copy=True)
```

A binariser can be used to turn ‘gray image’ to ‘black & white image’.

Outline

1. Features & Label Encoding
   - One-Hot Encoding
   - Ordinal Encoding
   - Discretisation
2. Pre-processing
3. **Linear Dimension Reduction: PCA**

Principal Component Analysis (PCA)

According to `https://scikit-learn.org/stable/modules/decomposition.html`, **principal component analysis** (**PCA**) is used to decompose a *multivariate dataset* in a set of successive orthogonal components that explain a maximum amount of the variance.

PCA (cont)

Given unlabelled numeric data X of shape $n \times p$.

Motivations:

If one of the column in X is constant (e.g. 0), can we remove it? $\rightarrow$ remove "redundant" data!

What is the statistical concept that measures the correlation between columns in statistics?

PCA (cont)

Answer: Covariance matrix of X.
Steps to calculate PCA:
Step 1: Check the variance of each columns and decide if scaling is necessary. If scaling is necessary,

$$s_{\cdot j} = \sqrt{\frac{\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}{n-1}} \Rightarrow x_{\cdot j} \rightarrow \frac{x_{\cdot j} - \overline{x}_j}{s_{\cdot j}} =: \widetilde{X}$$

Otherwise, shift the data to "column mean" / centre $\overline{\boldsymbol{x}}$,

$$x_{\cdot j} \rightarrow x_{\cdot j} - \overline{x}_j =: \widetilde{X}$$

PCA (cont)

Step 2: Compute the covariance matrix of X

$$\Sigma := \text{Cov}(X) = \frac{1}{n-1}\widetilde{X}^T\widetilde{X}$$

Step 3: Compute the eigenvalues λ and eigenvectors $\boldsymbol{e}$ of Σ.

- $\det(\Sigma - \lambda I) = 0$
- Since Σ is symmetric non-negative definite, i.e.

$$f(\boldsymbol{x}) = \boldsymbol{x}\Sigma\boldsymbol{x}^T \geq 0, \quad \forall \boldsymbol{x} \in \mathbb{R}^n.$$

Here $\boldsymbol{x}$ is $1 \times p$ matrix (a row vector).

PCA (cont)

Spectral Theorem implies that the Σ in $f(\boldsymbol{x})$ can be decomposed as

$$\Sigma = \lambda_1 \boldsymbol{e}_1^T \boldsymbol{e}_1 + \lambda_2 \boldsymbol{e}_2^T \boldsymbol{e}_2 + \cdots + \lambda_q \boldsymbol{e}_q^T \boldsymbol{e}_q, \quad q = \min\{n, p\}$$

Here,

- λ_i, $i = 1, ..., q$ reflects the "variation" of the covariance matrix Σ
- $\boldsymbol{e}_i$, $i = 1, ..., q$ reflects the "direction" of the variation of the matrix Σ
- $\boldsymbol{e}_i \cdot \boldsymbol{e}_j = 0$, $i \neq j$
- $\boldsymbol{e}_i \cdot \boldsymbol{e}_i = 1$, $i = 1, ..., q$

PCA (cont)

Ways to find λ_i and $\boldsymbol{e}_i$:

- Hand calculation: Solve $\det(\Sigma - \lambda I) = 0$ (only feasible for $p \leq 3$; then solve $(\Sigma - \lambda_i I)\boldsymbol{e} = \boldsymbol{0}$ to obtain $\boldsymbol{e}_i$;
- QR algorithm from Numerical Methods (related to power method);
- SVD algorithm: Instead of calculating eigenvalues for Σ, apply matrix decomposition:

$$\widetilde{X} = USV^*, \quad V = [\boldsymbol{e}_1^T, ..., \boldsymbol{e}_q^T]$$

Here, $S = \text{diag}(\sigma_1, ..., \sigma_q)$, $\lambda_i = \sigma_i^2$.

PCA (cont)

Step 4: Construct the principal components / scores (of $\widetilde{X}$):

$$
\begin{aligned}
PC_1 &= e_{11}\tilde{X}_1 + e_{12}\tilde{X}_2 + \cdots + e_{1p}\tilde{X}_p, \\
PC_2 &= e_{21}\tilde{X}_1 + e_{22}\tilde{X}_2 + \cdots + e_{2p}\tilde{X}_p, \\
&\vdots \\
PC_q &= e_{q1}\tilde{X}_1 + e_{q2}\tilde{X}_2 + \cdots + e_{qp}\tilde{X}_p
\end{aligned}
$$

where $\boldsymbol{e}_k = (e_{k1}, e_{k2}, ..., e_{kp})$, $k = 1, \cdots, q$.
Matrix representation:

$$PC = \widetilde{X}V.$$

PCA (cont)

- Proportion of Variance, i.e.

$$PVE_i = \frac{\lambda_i}{\lambda_1 + \lambda_2 + \cdots + \lambda_p}, \quad i = 1, \cdots, q.$$

- Cummulative Proportion of Variance, i.e.

$$CPVE_i = \frac{\lambda_1 + \cdots + \lambda_i}{\lambda_1 + \lambda_2 + \cdots + \lambda_p} = PVE_1 + \cdots + PVE_i.$$

$i = 1, \cdots, q.$

- Biplot: PC2 vs PC1, i.e. the **scores**

$$(\widetilde{X}\boldsymbol{e}_1,\ \widetilde{X}\boldsymbol{e}_2)$$

PCA (cont)

PCA in Python Sklearn:

- Step 0: Original Data, X
- Step 1: from sklearn.preprocessing import scale
  - pca = sklearn.decomposition.PCA().fit(X)
  - pca = sklearn.decomposition.PCA().fit(scale(X))
  - n: pca.n_samples_
  - p: pca.n_features_
  - Centre: pca.mean_
- Step 3:
  - λ_i: pca.explained_variance_
  - $V = [\boldsymbol{e}_1^T, ..., \boldsymbol{e}_q^T]$: pca.components_
  - q: pca.n_components_
- PVEs: pca.explained_variance_ratio_

PCA (cont)

Example (May 2021 Test Question): Given the following data.

x	y
4.49	2.5
3.38	1.95
2.58	1.93
3.75	2.38
2.88	1.82
4.29	2.53
3.11	2.04
4.18	2.43
3.94	2.57
4.86	2.95

PCA (cont)

Example (cont):

1. Find the **eigenvalues** and **proportion of variance** of the PCA of the data by using hand calculation with appropriate mathematical formulas. (3.5 marks)
2. Write down the Python commands to obtain the proportion of variance, eigenvalues and eigenvectors of the PCA of the data. Then write down the output generated by the Python commands. (1.5 marks)

PCA (cont)

Example (cont): Answer to Part 1.
First, find the column means:

$$\begin{bmatrix} 3.746 & 2.31 \end{bmatrix} \qquad \text{[0.5 mark]}$$

Then calculate covariance matrix

$$Cov(X) = \frac{1}{10-1} \begin{bmatrix} 0.744 & 0.19 \\ -0.366 & -0.36 \\ -1.166 & -0.38 \\ 0.004 & 0.07 \\ -0.866 & -0.49 \\ 0.544 & 0.22 \\ -0.636 & -0.27 \\ 0.434 & 0.12 \\ 0.194 & 0.26 \\ 1.114 & 0.64 \end{bmatrix}^T \begin{bmatrix} 0.744 & 0.19 \\ -0.366 & -0.36 \\ -1.166 & -0.38 \\ 0.004 & 0.07 \\ -0.866 & -0.49 \\ 0.544 & 0.22 \\ -0.636 & -0.27 \\ 0.434 & 0.12 \\ 0.194 & 0.26 \\ 1.114 & 0.64 \end{bmatrix} = \begin{bmatrix} 0.5516 & 0.2497 \\ 0.2497 & 0.1298 \end{bmatrix}. \qquad \text{[1 mark]}$$

PCA (cont)

Example (cont): Answer to Part 1 (cont).
The eigenvalues can be found by solving the following quadratic equation:

$$\begin{vmatrix} 0.5516 - \lambda & 0.2497 \\ 0.2497 & 0.1298 - \lambda \end{vmatrix} \qquad \text{[1 mark]}$$

$$= \lambda^2 - 0.6814\lambda + 0.00924759 = 0.$$

The eigenvalues are

$$\lambda_1 = 0.667547, \quad \lambda_2 = 0.013853 \qquad \text{[1 mark]}$$

PCA (cont)

Example (cont): Answer to Part 2.

The Python code is listed below: . . . . . [1 mark]

```
import numpy as np
from sklearn.decomposition import PCA

X = np.array(
[[4.49, 2.5 ], [3.38, 1.95], [2.58, 1.93], [3.75, 2.38],
 [2.88, 1.82], [4.29, 2.53], [3.11, 2.04], [4.18, 2.43],
 [3.94, 2.57], [4.86, 2.95],]
)
print(X)
pca = PCA()
pca.fit(X)

print("Centre=", pca.mean_)
print("PVE=", pca.explained_variance_ratio_)
print("Eigenvalues=", pca.explained_variance_)
print("Eigenvectors=\n",pca.components_)
```

The output is . . . . . [0.5 mark]

```
Centre= [3.746 2.31 ]
PVE= [0.97974463 0.02025537]
Eigenvalues= [0.66758058 0.01380165]
Eigenvectors=
 [[ 0.90697657  0.42118107]
 [ 0.42118107 -0.90697657]]
```

PCA (cont)

Exercise: Work with data with 3 columns.

If time permits:
Practical: Work with ISLR Chapter 10: p_pca.py