**4. Data Management**

**Topic1-Data\_Organization**

1. Data structure continuum

1) the 80/20 rule of data science: 80% time for data organization and management, 20% for analysis

2) Common issues of raw data & Metadata Storage

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text

Description automatically generated

2.

Text, letter

Description automatically generated

2. Pandas data frames

1)

square brackets [] index by column name

.loc method indexes by [row\_index] or [row\_index, column]: df.loc[2:5,'x1:Primary Column Reflux Flow':'x5:Primary Column Feed Flow from Feed Column']

.iloc method indexes by [row\_number] or [row\_number, column] : df.iloc[2:5,4:9]

indexes can be lists or slices

**\*df.loc[index:index,index:index]: df.loc[2:5,] 指的是index 为2,3,4,5的row. Dd.iloc[row#:row#, column#:column#], df.iloc[2:5,]指的是第2,3,4 row**.

2) DF=df.copy(), a new object will be created with a copy of the calling object’s data and indices. Modifications to the data or indices of the copy will not be reflected in the original object

3) Filter: define a filter: filt=df[‘X1’]>100, df\_filtered=df[filt]

4) Rename: new\_columnname={‘current\_name1’:’new\_name1’, ‘current\_name2’:’new\_name2’}, df\_new=df.rename(columns=new\_columnname)

5) Set index: df\_indexbycolumn6=df.set\_index(‘column6’)

6) discard column: df\_short=df.columns[3:6] #只是选取3,4,5列的index，需要loc来获取这些列的所有数据: df.loc[:,df\_short]

7) df.loc[:,df\_short].values return an array

8) Find all null values; df[pd.isnull(df).values]

9) def is\_real\_and\_finite(x):

if not np.isreal(x):

return False

elif not np.isfinite(x):

return False

elif pd.isnull(x):

return False

else:

return True

numeric\_map = df[nondate\_cols].**applymap(is\_real\_and\_finite)**

3. Missing values

1) Three basic strategies

• Dropping observations: Deleting rows

• Dropping features/variables: Deleting columns

• Imputation: Filling in values

2) Dropping Observations: Simply drop the rows that contain non-numeric values, Useful if you have a large amount of data.

real\_rows = numeric\_map.all(axis= 1).values, df\_dropped\_obs = df[real\_rows]

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.all.html>

axis = 1 or ‘Column’, 是返回一个column。Check每一行，都是true，返回true，有一个false，这一行就返回false

3) Dropping Features/Variables

• Drop the columns that contain non-numeric values.

• Useful if many features but few observations.

• Useful if features are highly correlated.

• Useful if one feature is responsible for many/all missing values

numeric\_cols = numeric\_map.all(axis=0)

numeric\_cols

for col in numeric\_map.columns:

print(numeric\_map[col].value\_counts())

print()

True 10673

False 30

Name: x1:Primary Column Reflux Flow, dtype: int64

4) import seaborn as sns

fig, ax = plt.subplots(figsize = (17, 15), dpi = 150)

corr = df.corr()

sns.heatmap(corr, ax = ax);

Chart

Description automatically generated

corr = df.corr()

corr[‘Avg\_Delta\_Composition Primary Column’] > 0.95

5) Imputation: trying to fill in missing values based on some approximation or relationship between known variables 🡺 Regression

4. Outliers :There are two general types of approaches:

1) parametric: Assume a form of the underlying distribution, then remove data that have a sufficiently low probability of occuring given that distribution.

2) non-parameteric: Use distance metrics to identify points that are very far away from others.

3) xi = df\_dropped\_obs["x3:Input to Primary Column Bed 3 Flow"].copy()

mu = np.mean(xi), stdev = np.std(xi), z\_cutoff = 3

zi = (xi - mu)/stdev

xi\_nooutliers=xi[np.abs(zi)<z\_cutoff]Text

Description automatically generated

5. Large dataset

?? 1) HDF5

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

**Topic2-Online\_Data\_Access**

1. Online data access

1) python package: import requests,

page = requests.get('https://pubchem.ncbi.nlm.nih.gov/compound/Ethanol')

2) Beautiful Soup is a Python library for pulling data out of HTML and XML files.

3) import json

with open('data/ethanol.json') as f:

etoh = json.load(f)

4)

**Skill check: last question?**

**Homework: plot?**

**5. Exploratory Data Analysis**

**Topic1-High\_Dimensional\_Data**

1. The curse of dimensionality refers to the fact that the volume of a high-dimensional space varies exponentially with the dimension.(维数越高，sampling会呈指数增长)

2. The blessing of dimensionality is a somewhat lesser known phenomenon that occurs because of the data sparsity that arises from the curse of dimensionality. It essentially means that as the number of independent dimensions increases the data tends to be more easily separable and will look increasingly like well-separated points.(维数越高，sample越趋向于更分散)

3. The curse of dimensionality always applies, but the blessing is not guaranteed. This means that in general it is more challenging to work in high dimensions.

4.Visualizing single features

1) "summary statistics" (mean, standard deviation, min, max, etc.) of each feature

(1) np.mean(axis=0)指的是给出一行，其中是每一列的平均. np.std()

2) histogram plots

fig, axes = plt.subplots(n, n, figsize = (5 \* n, 5 \* n), dpi = 200)

ax\_list = axes.ravel() 例如8x8的image plots。需要1到64的顺序，就可以asex.ravel()

for i in range(N):

ax\_list[i].hist(X\_mnist[:, i])

ax\_list[i].set\_xlabel(i)

5. Visualizing Multiple Features

1) Scatter plots: can reveal non-linear patterns.

features = [0, 1, 2, 3, 4]

n = len(features)

fig, axes = plt.subplots(n, n, figsize = (5.5 \* n, 5 \* n), dpi = 200)

A picture containing diagram

Description automatically generatedfor i in features:

for j in features:

ax = axes[i, j]

if i == j:

ax.hist(X\_mnist[:, i])

else:

x = X\_mnist[:, j]

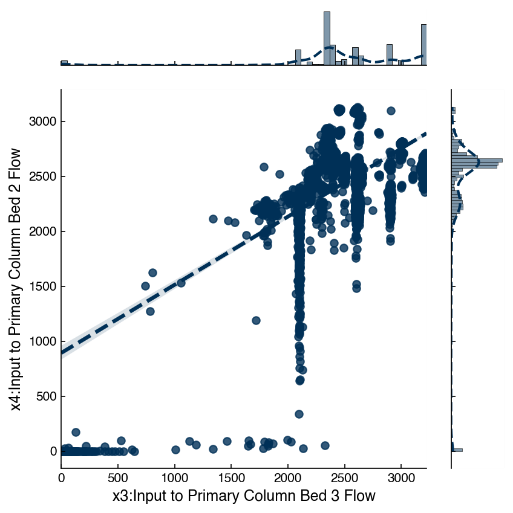
y = X\_mnist[:, i]

ax.scatter(x, y)

2) Joint polt: provide information on distribution of points

x\_col = df\_dow\_clean.columns[3]

y\_col = df\_dow\_clean.columns[4]

jp = sns.jointplot(x = x\_col, y = y\_col, data = df\_dow\_clean, kind = 'reg')

3) Correlation matrix

features = range(2, 11)

A picture containing background pattern

Description automatically generatedX\_reduced = X\_mnist[features] # select first 10 features

corr\_mnist = np.corrcoef(X\_reduced)

fig, ax = plt.subplots()

ax.imshow(corr\_mnist);

corr = mnist\_df[features].corr()

hm = sns.heatmap(corr, annot = True, fmt = '.2f', annot\_kws = {'fontsize': 10})

A picture containing text, different, scoreboard, cabinet

Description automatically generated

**Topic2-** **Topic2-Dimensionality\_Reduction**

1. There are a number of practical uses for dimensionality reduction algorithms:

* compression of data
* denoising of data
* interpretation of data
* improving model efficiency or performance

2. Assessing performance of dimensionality reduction models

* Distance: stress function
* Variance: assess the "retained variance" of the low-dimensional data
* Visualization: this becomes challenging if the low-dimensional space is > 3 dimensions
* Model performance: construct a supervised model from both the low- and high-dimensions

**3. Principal component analysis(PCA)**

1) the principal component analysis is obtained via the eigenvalues of the **covariance matrix**.

2) C = np.cov(X\_mnist.T)

eig\_vals, eig\_vecs = np.linalg.eig(C)

eig\_vecs = eig\_vecs.T

sorted\_idxs = np.argsort(eig\_vals) #this gives us the list of indices from smallest to largest

sorted\_idxs = list(sorted\_idxs)

sorted\_idxs.reverse() #this goes from largest to smallest

eig\_vals = eig\_vals[sorted\_idxs] #re-sort values

eig\_vecs = eig\_vecs[sorted\_idxs, :] #re-sort vectors

k = 20

projector = eig\_vecs[:k, :].T

X\_k = np.dot(X\_mnist,projector)

X\_reconstructed = np.dot(projector, X\_k.T).T

3**) from sklearn.decomposition import PCA**

k = 20

pca\_model = PCA(n\_components = k)

pca\_model.fit(X\_mnist)

X\_k = pca\_model.transform(X\_mnist)

X\_reconstructed = pca\_model.inverse\_transform(X\_k)

fig, axes = plt.subplots(1, 2, figsize = (10, 5))

index = 4

show\_image(X\_mnist, index, ax = axes[0])

show\_image(X\_reconstructed, index, ax = axes[1])

4) In conclusion, PCA is one of the most widely used techniques in dimensional reduction because it is:

* **Unsupervised** - We did not use the labels to determine the statistics
* **Projectable** - It is easy to project a new data point into the reduced dimensional space
* **Invertible** - It is easy to move from the low-dimensional space to the high dimensional space
* However, its weakness is **that it is linear in the original space**. It does not do well with non-linear patterns.

4. Kernel PCA: The solution to non-linearity in PCA is a familiar one: using a "kernel" to perform PCA in an even-higher dimensional space **that captures non-linearities**. The concept here is that rather than using the covariance matrix the eigenvalues of **a "kernel matrix" are used**

1) from sklearn.decomposition import KernelPCA, PCA

kPCA = KernelPCA(n\_components = k, kernel = 'rbf', gamma = gamma, fit\_inverse\_transform = True)

2) Kernel PCA is also invertible

5. Other PCA variants

* Robust PCA - good for cases where there is sparse data and/or large errors/outliers
* Partial least squares - supervised regression-based PCA that maximizes covariance between input and output
* Linear discriminant analysis - supervised classification-based PCA that maximizes inter-class variance

6. Manifold learning(Other dimensional reduction approach): Manifold learning approaches utilize **distance metrics** between points to define their similarity, and then seek **to minimize the** difference between distance metrics in the high- and low-dimensional spaces. much better suited **for highly non-linear datasets**.

1) from sklearn.manifold import **MDS** k = 2

mds = MDS(n\_components = k, n\_init = 1, max\_iter = 100) #<- note that we need to give some max\_iteration and initial guess parameters since this is iterative

X\_mds = mds.fit\_transform(X\_mnist)

2) Another popular manifold-based method is **tSNE,** a probabilistic similarity metric based on the t-distribution

from sklearn.manifold import TSNE (**visualization is better**)

tsne = TSNE(n\_components = 2, perplexity = 30.0,

early\_exaggeration = 12.0,

learning\_rate = 200.0,

n\_iter = 1000,

init = 'random',

method = 'exact')

X\_tsne = tsne.fit\_transform(X\_mnist)

3) Manifold techniques can give powerful insight into the high-dimensional structure of data; however, most suffer from several key disadvantages:

* Not projectable - the low dimensional representation only applies to the training points.
* Not invertible - no way to move back to high-dimensional space
* Slow - manifold techniques use distance matrices and hence tend to scale as 𝑁2

7. Autoencoding: the use of neural networks for dimensional reduction

This has numerous advantages:

* projectable and invertible - the link between the high/low dimensional representation is defined by the neural net
* fast and scalable - neural networks are computationally efficient
* non-linear and unsupervised - the autoencoder learns the non-linear manifold without needing labels

However, the typical cautions of neural networks apply:

* extremely large training datasets needed
* architecture and hyperparameters need to be tuned/selected
* no intuitive link between low- and high-dimensional representations

Skill check 9:

1. correlation matrix: X.corr(), X has to be dataframe matrix

2. two features correlation: corr\_value=X1.corr(X2)

3. standard scaling:

X=mnist\_df.iloc[:,8:12].to\_numpy()

X\_mnist\_scaled=(X-X.mean(axis=0))/(X.std(axis=0))

4. eigenvalues of kernel matrix: kpac.eigenvalues\_

**Topic3-Clustering**

1. Overview: Clustering algorithms seek to identify data points that are similar to each other based on a set of descriptive features. Clustering algorithms are **unsupervised since** they **do not include output labels**.

1) Types of problems/algorithms

* Expectation-maximization algorthims iteratively compute "expected" clusters and then "maximize" the parameters of the cluster to optimize the expectations.
* Density-based algorithms utilize local information about data points to identify regions where the data has similar density.
* Hierarchical algorithms map out the full network of connectivity within a dataset, then use a variable distance cutoff to assign clusters.

2) A few considerations when selecting a clustering algorithm:

* Some algorithms require **defining the number of clusters explicitly** (e.g. most expectation-maximization algorithms) while others find this implicitly based on choice of hyperparameters (e.g. density-based or hierarchical)
* Some algorithms **allow mixed membership** **where points can belong to multiple clusters** based on probabilities.
* Some algorithms **can identify/ignore outliers/noise** (e.g. density-based), while others attempt to assign clusters to all points (e.g. expectation-maximization and hierarchical).

3) Accuracy Metrics

* Silhouette score: Works best for dense, well-separated clusters. Does not work well for density-based clusters.
* Variance ratio criterion / Calinski-Harabasz score: Does not work well for density-based clusters (e.g. DBSCAN)
* Information criteria
* Classification metrics: confusion matrices, precision, recall, etc.

**Finally, it is worth noting that essentially all clustering algorithms rely on some form of distance metric**

4) Summary

* Clustering algorithms are an unsupervised approach identifying similar data points
* Silhouette score and variance ratio are common unsupervised assessments
* Clustering relies on distance metrics

2. Expectation-Maximization Models

**1) The k-means algorithm** is the simplest and most intuitive clustering algorithm.

(1) It performs remarkably well under a number of assumptions:

* Number of clusters are known
* Clusters are roughly spherical
* Clusters are separated by linear boundaries

(2) from sklearn.cluster import KMeans

model = KMeans(n\_clusters = 3)

model.fit(X\_pca)

y\_predict = model.predict(X\_pca)

centers = model.cluster\_centers\_

**2) Gaussian mixture models**: The approach is to model each cluster as a Gaussian distribution, and to model the entire dataset as a mixture of Gaussians.

(1) from sklearn.mixture import GaussianMixture

model = GaussianMixture(n\_clusters = 2,covariance\_type = ‘full’)

model.fit(X)

y\_predict = model.predict(X)

centers = model.means\_

(2) covariance\_type

* full: all entries in the covariance matrix is optimized.
* tied: only the diagonal elements will be non-zero.
* spherical: only diagonal elements will be non-zero and they will all be equal. Similar to k-means.

(3) Assess GMM: from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score

* Silhouette score; print(silhouette\_score(X, y\_predict))
* Calinski-Harabasz score: print(calinski\_harabasz\_score(X, y\_predict))
* Bayesian information criterion: GMM has built-in BIC method.: print(model.bic(X))

3) Summary

* Expectation-maximization is an iterative optimization algorithm
* k-Means is good for spherical clusters
* GMM is non-spherical and supports mixed memberships

3. Density-based models

**1) Mean shift algorithm**

* Seek the centroid of each cluster.
* A window is specified instead of a number of clusters.
* Use current centroids to identify all points within window
* Use all points within the window to compute new centroids

(1) from sklearn.cluster import MeanShift

model = MeanShift(bandwidth = 3) # bandwidth: related to size of window

model.fit(X\_tsne)

labels = model.labels\_

centroids = model.cluster\_centers\_

**2) DBSCAN**: Defines the cluster by whether a point falls within the sliding window.

•Good for highly non-linear boundaries.

•Key hyperparameters:

* eps: the radius to include in a cluster
* min\_samples: the minimum number of samples with in a window to form a cluster

(1) from sklearn.cluster import DBSCAN

model = DBSCAN(eps = 1, min\_samples = 3)

y\_predict = model.fit\_predict(X\_pca)

## **Not possible to predict the cluster of a new point**. Solution: use the DBSCAN output as

classes and train a classification model.

3) Summary

* Density-based models seek an area where data points are dense
* Mean shift algorithm updates cluster centers until convergence
* DBSCAN gathers data points with a sliding window

4. Hierarchical Models

* Construct linkages between different points.
* Visualize using dendrograms.
* Use distance cutoffs to assign clusters.
* Slow - scales as 𝑁^3.

1) Structure of a Linkage

(1)from scipy.cluster.hierarchy import linkage

X = X\_pca

Z = linkage(X, method = ’single’)

(2)Each entry has 4 members:

* 1st & 2nd: indices of two points / clusters that are being combined.
* 3rd: distance between these clusters.
* 4th: total number of points in the new cluster.

Graphical user interface, text, application

Description automatically generated

(3) Linkage Method: calculates the distance between two clusters.

* single: the minimum distance between any two points in the two clusters
* complete: the maximum distance between any two points in the two clusters
* average: an average of distances between points in the two clusters
* centroid: distance between cluster centroids
* weighted: differently weighted distances between the agglomerated cluster and one being added
* ward: distance that minimizes the variance between the clusters

**2) cophenetic coefficient**: which linkage method should we choose? This is where we can use the "cophenetic coefficient", which **measures the ratio of the distance in "linkage" space to the distance in the high-dimensional space.** Choose the option where **the cophenetic coefficient is the highest.**

(1) from scipy.cluster.hierarchy import cophenet

from scipy.spatial.distance import pdist

Dij = pdist(X, metric = ‘Euclidean’) #distance in high-D

Z = linkage(X, method=‘single’)

C, coph\_dists = cophenet(Z, Dij) #C gives the coefficient

3) Dendrogram

(1) from scipy.cluster.hierarchy import dendrogram

fig, ax = plt.subplots()

dendrogram(Z, color\_threshold = 20, ax = ax)

?color\_threshold: Distance (y-axis value) below which to identify separate branches as different colors

(2) Truncated Dendrogram

dendrogram(Z, color\_threshold = 20, truncate\_mode = ‘lastp’, p = 10, ax = ax)

* truncate\_mode = ‘lastp’: Shows the last p linkages in Z. Corresponds to Z[n-p-2:-1].
* p: the number of leaves to show in the dendrogram

4) Agglomerative Clustering:

•Clustering using the linkage structure.

•Define the number of clusters.

* Explicit: move up the tree until there are k clusters.
* Implicit: provide a linkage distance that defines separate clusters.

Graphical user interface, application

Description automatically generated with medium confidence

from scipy.cluster.hierarchy import fcluster

**clusters\_dist = fcluster(Z, 17.5, criterion = ‘distance’) #Implicit**

**clusters\_k = fcluster(Z, 4, critertion = ‘maxclust’) #Explicit**

5) Determine the Cutoffs: Inconsistency method

Graphical user interface, text

Description automatically generated

6) Summary

* Agglomerative clustering utilizes linkages between points in the dataset
* Hierarchy can be visualized by dendrograms
* Clusters can be defined implicitly or explicitly

**Topic4-Generative\_Models**

1. Generative Model Overview(**Unsupervised approach**)

1) Generative models **provide an estimate of the probability** of finding data at a particular point in feature space. These models can then be used to **generate synthetic data** that mimics the input data by sampling the probability distribution.

2) Normal distribution: The 1-dimensional normal distribution is the simplest case of a generative model

(1) from scipy.stats import norm

x = np.linspace(-3, 3, 100)

gauss = norm.pdf(x, loc = 0, scale = 0)

* pdf: probability density function
* loc: mean of the distribution
* scale: standard deviation of the distribution

(2) X\_new = norm.rvs(loc = 0, scale = 1, size = 10000) #rvs: random variate sample # We can generate new data points once we know the parameters of distribution.

x = X\_dow[:, 6]

mu = x.mean()

std = x.std()

x\_synthetic = norm.rvs(mu, std, size = 1000)

(3) Generative models are often combinations of different Gaussian distributions.

3) Summary

* Generative models learn a probability distribution
* Generative models can create synthetic data
* Simple generative model can be developed with 1-D normal distribution

2. Gaussian Mixture Models

1) from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components = 2, covariance\_type = ‘full’)

gmm.fit(X)

y = gmm.predict(X)

2) Bayesian Information Criterion: To determine the right number of Gaussians

3) **One issue with GMM's is that they do not scale well with the number of dimensions. One strategy to deal with this is to combine a GMM model with an invertible dimensionality reduction approach**

Timeline

Description automatically generated

4) Another strategy: Reduce the Number of Parameters: Let’s change the covariance\_type to spherical.

5) Summary

* GMMs learn a probability distribution as a sum of Gaussian distributions.
* Optimal number of Gaussians can be determined using the BIC.
* GMMs work for multi-dimensional data
* GMM scales poorly with number of dimensions, may need dimensional reduction

3. Kernel Density Estimation

? Text, letter

Description automatically generated

1) from sklearn.neighbors import KernelDensity

kde = KernelDensity(**bandwidth = 0.15, kernel = ‘gaussian’**)

kde.fit(x)

x\_continuous = np.linspace(min(x), max(x), 1000)

**#returns the log of probability density**

logprob = kde.score\_samples(x\_continuous)

2) generate new samples

x\_syn = kde.sample(1000) #Make 1000 new samples

x\_syn = x\_syn.reshape(-1, 1)

kde.fit(x\_syn)

logprob\_syn = kde.score\_samples(x\_continuous)

3) Summary

* KDE converts data to a probability distribution.
* KDE is good for visualization
* KDE works well with high-dimensional data.
* KDE distribution can be sampled to generate new data.

4. Not-so-naïve Bayes

1) Revisit Naïve Bayes Classification

•Naïve Bayes is a generative classification algorithm.

• It computes the posterior probability taking advantage of naïve assumptions:

* The data in each class follows a Gaussian distribution.
* The features are not correlated.

(1) X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_mnist, y\_mnist,

test\_size = 0.3, random\_state = 1)

NB = GaussianNB()

NB.fit(X\_train, y\_train)

yhat = NB.predict(X\_test)

cm = confusion\_matrix(y\_test, yhat)

2) Probability from KDE

* In this lecture, we will get probabilities 𝑃(𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠|𝑐!) from KDE.
* Based on results from KDE, each data point is assigned into a class of which the posterior probability is the highest.

(1) label = 0

X = X\_mnist[y\_mnist == label]

model = KernelDensity(bandwidth = 10, kernel = ‘gaussian’)

model.fit(X)

Graphical user interface, text, application

Description automatically generated

3) X\_train, X\_test, y\_train, y\_test =

train\_test\_split(X\_mnist, y\_mnist, test\_size = 0.3)

model = KernelDensity(bandwidth = 10, kernel = ‘gaussian’)

prediction = not\_so\_naive(X\_train, X\_test, y\_train, model)

from sklearn.metrics import accuracy\_score

4) Summary

* Generative models can remove naïve assumptions in Bayesian classification
* KDE Bayes works very well even in high dimensions
* Using KDE with Bayesian classification leads to excellent performance

**6. Feature engineering**

**Topic1-Feature\_Transformations**