**1.Numerical methods**

**Topic 2 Linear Algebra**

**1. For loops to multiply a matrix and a vector**

Text

Description automatically generatedA = np.array([[0, 1], [2, 3]]) B = np.array([0, 1])

for i in range(A.shape[0]):

sum = 0

for j in range(A.shape[1]):

sum += A[i][j] \* B[j]

print(sum)

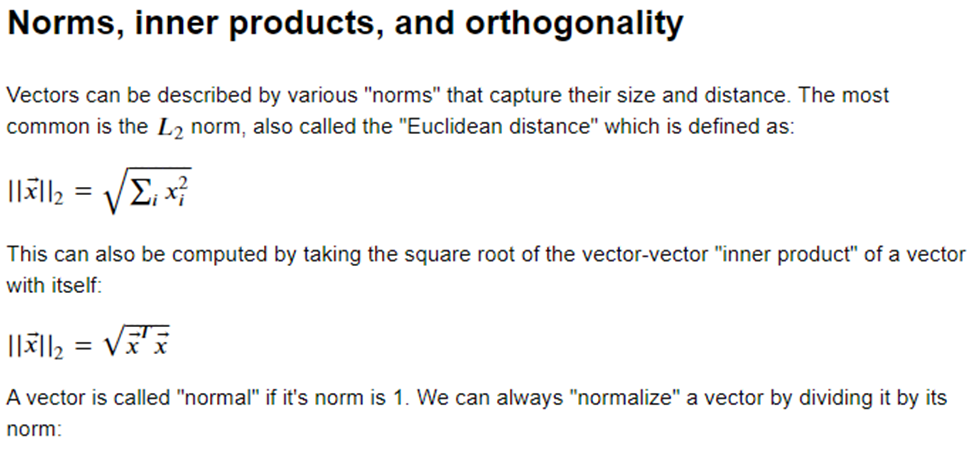
**2. "Vandermonde" matrix**

Chart, scatter chart

Description automatically generatedText

Description automatically generated with medium confidence

3.



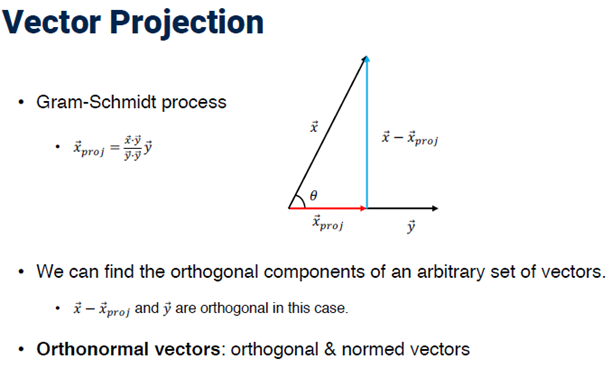
col\_0 = X\_vdm[:,0]

norm\_col\_0 = np.linalg.norm(col\_0,2)

col\_0\_normed = col\_0/norm\_col\_0

**4. Vectors are defined as "orthogonal" if their inner product is zero**

Gram-Schmidt process



**Find the orthogonal components of an arbitrary set of vectors:**

**col\_1\_ortho = col\_1\_normed - np.dot(col\_0\_normed, col\_1\_normed)\*col\_0\_normed**

**np.dot(col\_1\_ortho, col\_0\_normed)**

**有一个vector，先求它的norm2后求它的normalized(vector/vector\_norm)，然后另一个vector也z做同样处理求得col\_1\_normed。再根据以上code求得与col\_0\_normed垂直的vector**

Graphical user interface, application

Description automatically generated with medium confidence

########################################################################

e1 = v[:, 0]

e2 = v[:, 1] - np.dot(e1, v[:, 1]) / np.dot(e1, e1) \* e1

e3 = v[:, 2] - np.dot(e1, v[:, 2]) / np.dot(e1, e1) \* e1 - np.dot(e2, v[:, 2]) / np.dot(e2, e2) \* e2

e4 = v[:, 3] - np.dot(e1, v[:, 3]) / np.dot(e1, e1) \* e1 - np.dot(e2, v[:, 3]) / np.dot(e2, e2) \* e2 - np.dot(e3, v[:, 3]) / np.dot(e3, e3) \* e3

e5 = v[:, 4] - np.dot(e1, v[:, 4]) / np.dot(e1, e1) \* e1 - np.dot(e2, v[:, 4]) / np.dot(e2, e2) \* e2 - np.dot(e3, v[:, 4]) / np.dot(e3, e3) \* e3 - np.dot(e4, v[:, 4]) / np.dot(e4, e4) \* e4

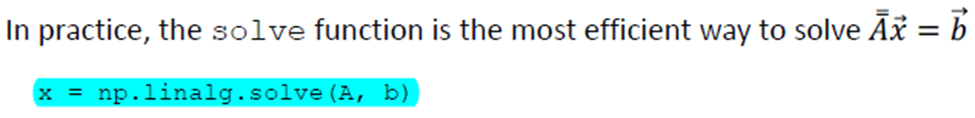
ortho\_v = np.hstack((e1.reshape(-1, 1) / np.linalg.norm(e1), e2.reshape(-1, 1) / np.linalg.norm(e2) , e3.reshape(-1, 1) / np.linalg.norm(e3), e4.reshape(-1, 1) / np.linalg.norm(e4), e5.reshape(-1, 1) / np.linalg.norm(e5)))

####################################################################

**5. Rank: The number of linearly independent columns or rows in the matrix: rank(A)≤min(m,n)**

rank\_x = np.linalg.matrix\_rank(x)

**6. Solve linear systems**



**7. Eigenvalues and Eigenvectors**

1) The eigenvectors of a symmetric matrix will always be orthonormal

2) Eigendecomposition is only possible for a square matrix. However, there is a similar concept called a "singular value decomposition", or SVD, that will work for any matrix

Graphical user interface, text, application, email

Description automatically generated Text

Description automatically generated

**8. plot import pylab as plt**

fig, ax = plt.subplots(figsize=(10, 5)) #create a new figure object (fig) with an associated axis object (ax)

**ax.plot(x,y,marker='o', ls='none')**

**ax.hist(y1 - yhat, bins = 10) 直方图**

ax.plot(x1,y1, lable=’Col 0’)

ax.set\_xlabel('Wavenumber [cm$^{-1}$]')

ax.set\_ylabel('Absorbance [unitless]')

ax.set\_title('IR of ethanol')

**Topic 3 Linear Regression**

**1. Cost funtion (Loss function) SSE=np.sum((y-yhat)\*\*2)**

**2. Simple linear regression**

A = X.T @ X b = X.T@y

w\_lsr = np.linalg.solve(A,b) yhat = X@w\_lsr

**3. Polynomial regression is still linear. (相较于simple，这里可以有x^2等，w是多个parameters)**

A = X.T @ X b = X.T@y

w\_lsr = np.linalg.solve(A,b) yhat = X@w\_lsr

**4. General Linear Regression**

this form can be used for many different types of linear regression and is referred to as a general linear model.

**5. Gaussian**

Graphical user interface, text, application, email

Description automatically generated

Text, letter

Description automatically generated

**6. scikit-learn model.coef\_.tolist()**

from sklearn.linear\_model import LinearRegression

Text, letter

Description automatically generated

(1) model = LinearRegression(fit\_intercept = False)

**Topic 4 Numerical Optimization**

**1. Automatic Differentiation (现有一个loss function，然后求它对lamda的differentiation，然后再用Scipy.minimize或是gradient descent来求最佳lamda)**

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated with medium confidence

**2. Gradient Descent**

**start with some initial guess then iteratively improve it.**

Text

Description automatically generated

Text, letter

Description automatically generated

**3. Optimization with Scipy**

A picture containing text

Description automatically generated

Fitted = result.x

**4. Using general linear regression with the new features, plot the result of regression along with the original y\_peak. You may add an intercept column to determine whether adding an intercept results in a better fitting.**

# with intercept

x = np.linspace(900, 1700, 150)

features = gaussian\_features(x, 8, 100)

## intercept column

intercept = np.ones((features.shape[0], 1))

## pad the intercept column to the features matrix

features\_w\_int = np.append(intercept, features, axis = 1)

A = features\_w\_int.T@features\_w\_int

b = features\_w\_int.T@y\_peak

## w.shape will be (9,) which was originally (8,)

w = np.linalg.solve(A, b)

fig, ax = plt.subplots(figsize = (10, 4.5), dpi = 150)

ax.plot(x, y\_peak, '--k')

ax.plot(x, features\_w\_int@w, '-b')

ax.set\_xlabel('wavenumber [$cm^{-1}$]')

ax.set\_ylabel('absorbance');

**5. It is more likely to have some negative weights as a result. let's add a constraint that all weights should be positive.**

Graphical user interface, text

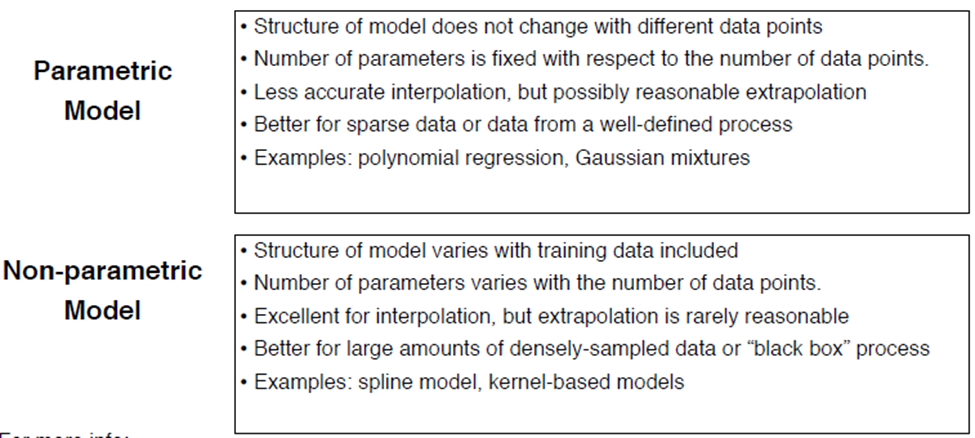
Description automatically generatedGraphical user interface, text

Description automatically generated

**2.Regression**

**Topic 1 Non-parametric models**

**1. This number of parameters to include is called a hyperparameter. Hyperparameters control the complexity of the final model**



**2. Piecewise Linear Functions ？？**

def piecewise\_linear(x\_train, x\_test=None):

if x\_test is None:

x\_test = x\_train

N = len(x\_test) #<- number of data points

M = len(x\_train) #<- number of features

X = np.zeros((N,M))

for i in range(N):

for j in range(M):

X[i,j] = max(0, x\_test[i] - x\_train[j])

return X

**3. Linear regression**

from sklearn.linear\_model import LinearRegression

model = LinearRegression(fit\_intercept = False)

model.fit(X, y\_peak)

r2 = model.score(X, y\_peak)

yhat = model.predict(X)

**4. Kernel Regression**

def rbf(x\_train, x\_test=None, gamma=1):

if x\_test is None:

x\_test = x\_train

N = len(x\_test) #<- number of data points

M = len(x\_train) #<- number of features

X = np.zeros((N,M))

for i in range(N):

for j in range(M):

X[i,j] = np.exp(-gamma\*(x\_test[i] - x\_train[j])\*\*2)

return X

**X\_train = rbf(x\_peak, gamma=gamma)**

model\_rbf = LinearRegression() #create a linear regression model instance

model\_rbf.fit(X\_train, y\_peak) #fit the model

r2 = model\_rbf.score(X\_train, y\_peak) #get the "score", which is equivalent to r^2

print('r^2 = {}'.format(r2))

**X\_test = rbf(x\_peak, x\_test=x\_test, gamma=gamma)**

yhat\_rbf = model\_rbf.predict(X\_test) #create the model prediction

**Topic2-Model\_Validation**

1. r2

Text, letter

Description automatically generated Text

Description automatically generated

**from sklearn.metrics import mean\_absolute\_error**

**MAE = mean\_absolute\_error(y\_test, yhat\_test)**

**2. Cross Validation**

**(1) Hold out cross validation**

from sklearn.model\_selection import train\_test\_split

np.random.seed(0)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_peak, y\_peak, test\_size=0.4)

**(2) k-fold Cross Validation**

from sklearn.model\_selection import KFold

kf = KFold(n\_splits = 5)

for train\_index, test\_index in kf.split(x\_peak):

x\_train, x\_test = x\_peak[train\_index], x\_peak[test\_index]

y\_train, y\_test = y\_peak[train\_index], y\_peak[test\_index]

**3. Standard Deviation of Error: One simple way of quantifying uncertainty**

error\_stdev = np.std(y3 - yhat, ddof = 2) print(error\_stdev)

**4. Gaussian Process Regression**

from sklearn.gaussian\_process import GaussianProcessRegressor

from sklearn.gaussian\_process.kernels import RBF

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

y\_peak = y\_peak.reshape(-1, 1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_peak, y\_peak, test\_size = 0.4)

gpr = GaussianProcessRegressor(kernel = RBF(1), alpha = 0.000005)

gpr.fit(x\_train, y\_train)

y\_gpr, y\_std = gpr.predict(x\_peak, return\_std = True)

**5. x\_third = x\_peak[::3]**

**Topic3-Complexity\_Optimization**

A picture containing chart

Description automatically generated

1. **Information Criteria** (the tradeoff between the number of parameters and the model error. Most common one: Bayesian IC, **Lower BIC corresponds to a better model**)

def BIC(y, yhat, k):

err = y - yhat

sigma = np.std(np.real(err))

n = len(y)

B = n\*np.log(sigma\*\*2) + k\*np.log(n)

return B



**2. def polynomial\_features(x, N):**

return np.array([x\*\*k for k in range(0,N)]).T

**3. def gaussian\_features(x, N , sigma = 25):**

# x is a vector # sigma is the standard deviation

xk\_vec = np.linspace(min(x), max(x), N)

features = []

for xk in xk\_vec:

features.append(np.exp(-((x - xk)\*\*2/(2\*sigma\*\*2))))

return np.array(features).T

4. **Regularization**

Text

Description automatically generated

5. **Kernel Ridge Regression**

from sklearn.kernel\_ridge import KernelRidge

sigma = 10 gamma = 1./(2\*sigma\*\*2) alpha = 0.1

KRR = KernelRidge(alpha=alpha, kernel='rbf', gamma=gamma)

x\_peak = x\_peak.reshape(-1,1) #we need to convert these to columns

y\_peak = y\_peak.reshape(-1,1)

KRR.fit(x\_peak, y\_peak)

x\_predict = np.linspace(min(x\_peak), max(x\_peak), 300) #create prediction data

yhat\_KRR = KRR.predict(x\_predict)

**r2\_test = KRR.score(x\_test, y\_test)**

6. **LASSO Regularization**

Ridge regression provides a good way to penalize model "smoothness", but it doesn't actually reduce the number of parameters. We can see that all of the coefficients are non-zero

from sklearn.metrics.pairwise import rbf\_kernel

sigma = 10

gamma = 1./(2\*sigma\*\*2)

**X\_train = rbf\_kernel(x\_train, x\_train, gamma=gamma)**

from sklearn.linear\_model import Lasso

sigma = 10 gamma = 1./(2\*sigma\*\*2) alpha = 1e-4

LASSO = Lasso(alpha=alpha)

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

print('The number of coefficients: {}'.format(len(LASSO.coef\_)))

x\_predict = np.linspace(min(x\_peak), max(x\_peak), 300) #create prediction data

**X\_predict = rbf\_kernel(x\_predict, x\_train, gamma=gamma)**

yhat\_LASSO = LASSO.predict(X\_predict)

coeffs = LASSO.coef\_

nonzero = [f for f in np.isclose(coeffs,0) if f == False]

print('Total number of non-zero parameters: {}'.format(len(nonzero)))

7. **GridSearchCV searches over a grid of hyperparameters, and uses cross-validation at each grid point to assess model performace Hyperparameter Tuning**

from sklearn.model\_selection import GridSearchCV

sigmas = np.array([5, 10, 15, 20, 25, 30,35, 40])

gammas = 1./(2\*sigmas\*\*2)

alphas = np.array([1e-9, 1e-5, 1e-4,1e-3, 1e-2,1e-1, 1])

parameter\_ranges = {'alpha':alphas, 'gamma':gammas}

KRR = KernelRidge(kernel='rbf')

KRR\_search = GridSearchCV(KRR, parameter\_ranges, cv=3)

KRR\_search.fit(x\_train,y\_train)

KRR\_search.best\_estimator\_, KRR\_search.best\_score\_

yhat\_KRR = KRR\_search.best\_estimator\_.predict(x\_predict)

**8. PLOT**

fig, ax = plt.subplots(figsize = (10, 5), dpi = 150)

ax.plot(x\_peak, y\_peak, '-k', label = 'original peak')

ax.plot(x\_train, y\_train\_hat, '.b', label = 'training set')

ax.plot(x\_test, y\_test\_hat, '.r', label = 'test set')

ax.legend()

ax.set\_xlabel('wavenumber [$cm^{-1}$]')

ax.set\_ylabel('absorbance');

**Topic4-High-dimensional\_Regression**

**1.vsualization of features**

N = X.shape[-1]

n = int(np.sqrt(N))

fig, axes = plt.subplots(n, n+1, figsize = (6\*n, 6\*n))

ax\_list = axes.ravel()

for i in range(N):

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

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

2. **covariance matrix**

covar = np.cov(X.T)

fig,ax = plt.subplots()

c = ax.imshow(covar)

fig.colorbar(c);

**3. Scaling Features and Outputs**

1) What could go wrong with rescaling or mean scaling? Cannot handle outliers

**2) standard scaling: X\_scaled = (X - X.mean(axis=0))/X.std(axis=0)**

**from sklearn.preprocessing import StandardScaler**

**ss = StandardScaler()**

**X\_mnist\_scaled = ss.fit\_transform(X\_mnist[:, 8:12])**

3) The structure looks totally different! This is the "correlation matrix", which tells us how correlated different features are on a scale of -1 to 1. A correlation of -1 means they are perfectly anti-correlated, while 1 means they are perfectly correlated. If any features are perfectly correlated then they are linearly dependent (and won't count toward the rank).

**4) Covariance matrix of standard scaled data = Correlation matrix**

A screenshot of a computer

Description automatically generated with medium confidence

**4. Multi-Linear Regression**

Text

Description automatically generated

**5. Dimensionality reduction**

1) forward selection

2) **PCA**

from scipy.linalg import eigvals, eig

eigvals, eigvecs = eig(corr)

PCvals, PCvecs = eigvals, eigvecs

total\_variance = np.sum(np.real(PCvals))

explained\_variance = np.real(PCvals)/total\_variance

**PC\_projection = np.dot(X\_scaled, PCvecs)**

print(PC\_projection.shape)

corr\_PCs = np.corrcoef(PC\_projection.T)

fig,ax = plt.subplots()

c = ax.imshow(corr\_PCs)

fig.colorbar(c);

**score\_list = []**

**for j in range(PC\_projection.shape[1]):**

**model = LinearRegression() #create a linear regression model instance**

**xj = PC\_projection[:,j].reshape(-1, 1)**

**model.fit(xj, y) #fit the model**

**r2 = model.score(xj, y) #get the "score", which is equivalent to r^2**

**score\_list.append([r2, j])**

**score\_list.sort()**

**score\_list.reverse()**

for r, j in score\_list:

print("{} : r^2 = {}".format(j, r))

3) Why is the model with principal components not always better than direct linear regression?

PCA only takes account of variance of the data. This does not necessarily relate to a linear correlation with the y data.

**6. from sklearn.preprocessing import StandardScaler, MinMaxScaler**

ss = StandardScaler()

X\_train\_ss = ss.fit\_transform(X\_train)

X\_test\_ss = ss.transform(X\_test)

**7.from sklearn.decomposition import PCA Homework 5**

pca = PCA()

PC\_projection = pca.fit\_transform(X\_scaled)

fig, ax = plt.subplots(figsize = (5, 4.5), dpi = 150)

ax.scatter(PC\_projection[:, 0], PC\_projection[:, 1], alpha = .4)

ax.set\_xlabel('1st principal component')

ax.set\_ylabel('2nd principal component');

**8. fig, ax = plt.subplots(figsize = (10, 4.5), dpi = 150)**

ax.plot(range(1, 41), pca.explained\_variance\_ratio\_, 'o')

ax.set\_xlabel('i-th principal component')

ax.set\_ylabel('explained variance ratio');

**9. from sklearn.linear\_model import LinearRegression**

r2 = []

for i in range(PC\_projection.shape[1]):

lr = LinearRegression(fit\_intercept = True)

lr.fit(PC\_projection[:, i].reshape(-1, 1), y)

r2.append(lr.score(PC\_projection[:, i].reshape(-1, 1), y))

fig, ax = plt.subplots(figsize = (10, 4.5), dpi = 150)

ax.plot(range(1, 41), r2, 'o')

ax.set\_xlabel('i-th principal component')

ax.set\_ylabel('$\mathrm{r^2}$');

**3.Classification**

**Topic1-Classification\_Basics**

1. Two distinct types of classification models: discriminative and generative

**(1) Discriminative models 学习出一条线或是一个function，区分各个点属于一个class的probability**

Graphical user interface, text, application, email

Description automatically generatedChart

Description automatically generated

**(2) Generative models identifying similarities between points within a given class ？？**

Graphical user interface, text

Description automatically generatedChart, bubble chart

Description automatically generated

2. **Accuracy, Precision, Recall, and F1 scores**

A screenshot of a computer

Description automatically generated with medium confidence

def acc\_prec\_recall(y\_model, y\_actual):

TP = np.sum(np.logical\_and(y\_model == y\_actual, y\_model == 1))

TN = np.sum(np.logical\_and(y\_model == y\_actual, y\_model == 0))

FP = np.sum(np.logical\_and(y\_model != y\_actual, y\_model == 1))

FN = np.sum(np.logical\_and(y\_model != y\_actual, y\_model == 0))

acc = (TP + TN) / (TP + TN + FP + FN)

if TP == 0:

prec = 0

recall = 0

else:

prec = TP / (TP + FP)

recall = TP / (TP + FN)

return acc, prec, recall

**3. Confusion matrices**

Chart, scatter chart

Description automatically generatedIn a confusion matrix **the diagonal elements correspond to true positives and true negatives**, while the off-diagonal elements correspond to false postives and false negatives, with **false positives above the diagonal and false negatives below**

**4. from sklearn.svm import SVC**

np.random.seed(1)

X\_mc, y\_mc = make\_blobs(n\_samples = 200, centers = 3, cluster\_std = 0.5\*noisiness, n\_features = 2)

model = SVC(kernel = 'linear', C = 1, decision\_function\_shape = 'ovr')

model.fit(X\_mc, y\_mc)

y\_mc\_hat = model.predict(X\_mc)

fig, axes = plt.subplots(1, 2, figsize = (15, 6))

axes[0].scatter(X\_mc[:, 0], X\_mc[:, 1], c = clrs[y\_mc])

x\_min, x\_max = X\_mc[:, 0].min() - 1, X\_mc[:, 0].max() + 1

y\_min, y\_max = X\_mc[:, 1].min() - 1, X\_mc[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.1), np.arange(y\_min, y\_max, 0.1))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

axes[1].contourf(xx, yy, Z, alpha = 0.4)

axes[1].scatter(X\_mc[:, 0], X\_mc[:, 1], c = clrs[y\_mc\_hat])

axes[0].set\_title('Original Data')

axes[1].set\_title('Prediction');

**5. def add\_intercept(X):**

intercept = np.ones((X.shape[0],1))

X\_intercept = np.append(intercept,X,1)

return X\_intercept

**def linear\_classifier(X, w):**

X\_intercept = add\_intercept(X)

p = np.dot(X\_intercept, w)

return p > 0

**6. Classification loss function ???**

Diagram

Description automatically generated with low confidence ;A picture containing text

Description automatically generated; 

**若是分错了类，第二个不等式就是正的，第三个会给一个大于零的数，minimize这个cost就可以减小分类错误 from scipy.optimize import minimize**

**(1) max cost function**

**def max\_cost(w, X, y):**

X\_intercept = add\_intercept(X)

Xb = np.dot(X\_intercept,w)

return sum(np.maximum(0, -y\*Xb))

print(max\_cost(w, X, y))

**(2) counting loss function**

**def n\_wrong(w, X = X, y = y):**

X\_intercept = add\_intercept(X)

Xb = np.dot(X\_intercept,w)

return sum(np.maximum(0, np.sign(-y\*Xb)))

print(n\_wrong(w,X,y))

**Topic2-Generalized\_Linear\_Models**

**1. Logistic regression as loss function**

Graphical user interface, text, application

Description automatically generated

**def softmax\_cost(w, X = X, y = y):**

**X\_intercept = add\_intercept(X)**

**Xb = np.dot(X\_intercept, w)**

**exp\_yXb = np.exp(-y \* Xb)**

**return sum(np.log(1 + exp\_yXb))**

from scipy.optimize import minimize

result = minimize(softmax\_cost, w, args = (X, y))

w\_logit = result.x

prediction = linear\_classifier(X, w\_logit)

fig, axes = plt.subplots(1, 2, figsize = (15, 6))

axes[0].scatter(X[:, 0], X[:, 1], c = clrs[y\_blob + 1])

axes[1].scatter(X[:, 0], X[:, 1], c = clrs[prediction + 1])

**#plot line**

m = -w\_logit[1] / w\_logit[2]

b = -w\_logit[0] / w\_logit[2]

axes[1].plot(X[:, 0], m\*X[:, 0] + b, ls = '-')

axes[0].set\_title('Original Data')

axes[1].set\_title('Prediction');

**2. Margin cost**

def margin\_cost(w, X = X, y = y):

X\_intercept = add\_intercept(X)

Xb = np.dot(X\_intercept,w)

return sum(np.maximum(0, 1 - y \* Xb))

**3. Support Vector Machine**

1) 

2) def regularized\_cost(w, X = X, y = y, alpha = 1):

X\_intercept = add\_intercept(X)

Xb = np.dot(X\_intercept, w)

cost = sum(np.maximum(0, 1 - y\*Xb))

cost += alpha\*np.linalg.norm(w[1:], 2)

return cost

**4. Non-linearity and Kernels**

1) We have seen lots of ways to find discrimination lines for linearly separable datasets, but they do not work well for non-linearly separable datasets

2) 

3) We can add this as a third feature

X\_new = np.exp(-(X[:,0]\*\*2 + X[:,1]\*\*2))

X\_new = X\_new.reshape(-1, 1)

X\_nonlinear = np.append(X, X\_new, 1)

print(X\_nonlinear.shape)

fig, axes = plt.subplots(1, 2, figsize = (15, 6))

axes[0].scatter(X\_nonlinear[:, 0],X\_nonlinear[:, 1],c = clrs[y\_circles + 1])

axes[0].set\_xlabel('$x\_0$')

axes[0].set\_ylabel('$x\_1$')

axes[1].scatter(X\_nonlinear[:, 0],X\_nonlinear[:, 2],c = clrs[y\_circles + 1])

axes[1].set\_xlabel('$x\_0$')

axes[1].set\_ylabel('$x\_2$');

5. the data **is not linearly separable** in the transformed space. To make this more general can use the **"kernel" idea from "kernel ridge regression",** and construct a new "kernel matrix"

from sklearn.metrics.pairwise import rbf\_kernel

X\_kernel = rbf\_kernel(X, X, gamma=1)

print(X\_kernel.shape)

**6. SVC for non-linear separable data**

from sklearn.svm import SVC # "Support vector classifier"

model = SVC(kernel = 'rbf', gamma = 1, C = 1000)

model.fit(X, y)

y\_predict = model.predict(X)

fig, axes = plt.subplots(1, 2, figsize = (15, 6))

axes[0].scatter(X[:, 0], X[:, 1], c = clrs[y])

axes[1].scatter(X[:, 0], X[:, 1], c = clrs[y\_predict]);

9. def plot\_svc\_decision\_function(model, ax=None, plot\_support=True):

Diagram

Description automatically generated

**Topic3-Alternate\_Classification\_Models**

**1. KNN**: The class of a point is determined by letting its k-nearest neighbors "vote" on which class it should be in. The point is assigned to whichever class has the most votes.

**def distance(x1, x2):**

# we will use the numpy 2-norm to calculate Euclidean distance:

return np.linalg.norm(x1 - x2, 2) #<- the 2 is optional here since 2 is the default.

**def get\_neighbor\_idxs(x, x\_list, k):**

dist\_pairs = []

for i,xi in enumerate(x\_list):

dist = distance(x, xi)

dist\_pairs.append([dist, i, xi]) #<- gives us the distance for each point

dist\_pairs.sort() #<- sort by distance

k\_dists = dist\_pairs[:k] #<- take the k closest points

kNN\_idxs = [i for di, i, xi in k\_dists] #<- we will get the indices of neighbors instead of the point itself.

return kNN\_idxs

**from scipy.stats import mode**

**def assign\_class(x, x\_list, y\_list, k): #<- now we need to know the responses**

neighbors = get\_neighbor\_idxs(x, x\_list, k)

y\_list = list(y\_list) #<- this ensures that indexing works properly if y\_list is a `pandas` object.

votes = [y\_list[i] for i in neighbors]

assignment = mode(votes)[0][0] #<- we won't deal with ties for this simple implementation

return assignment

**def kNN(X, k, X\_train, y\_train):**

**y\_out = []**

**for xi in X:**

**y\_out.append(assign\_class(xi, X\_train, y\_train, k))**

**y\_out = np.array(y\_out)**

**return y\_out**

**from sklearn.neighbors import KNeighborsClassifier**

**knn = KNeighborsClassifier(n\_neighbors = 20)**

**knn.fit(X, y)**

**y\_predict = knn.predict(X)**

**2. Naive Bayes Classification:** A totally different approache is the "Naive Bayes" classifier, which is a generative model. The model is "naive" because it naively assumes that the data in each class follows a Gaussian distribution, and that the features are not correlated.

from sklearn.naive\_bayes import GaussianNB

NB = GaussianNB()

NB.fit(X, y)

y\_predict = NB.predict(X)

**3. Decision Trees**

from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()

tree.fit(X, y)

y\_tree = tree.predict(X)

**Topic4-High-dimensional\_Classification**

**1. from sklearn.metrics.pairwise import rbf\_kernel**

from sklearn.metrics import accuracy\_score, confusion\_matrix

X\_kernel = rbf\_kernel(X\_perov, X\_perov, gamma = 0.02)

**2. from sklearn.svm import SVC**

model = SVC(kernel = 'rbf', gamma = 100, C = 1000)

model.fit(X\_perov[:, 3:5], y\_perov)

y\_predict = model.predict(X\_perov[:, 3:5])

**3. GridSearchCV and SCV**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_perov, y\_perov, test\_size=0.33)

from sklearn.model\_selection import GridSearchCV

from sklearn.utils import shuffle

X\_train, y\_train = shuffle(X\_train, y\_train) #Shuffle everything just for good measure

sigmas = np.array([1e-3, 1e-2, 1e-1, 1, 10, 100])

gammas = 1. / 2 / sigmas\*\*2

alphas = np.array([1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1])

Cs = 1 / alphas

parameter\_ranges = {'C': Cs, 'gamma': gammas}

svc = SVC(kernel = 'rbf')

svc\_search = GridSearchCV(svc, parameter\_ranges, cv = 3)

svc\_search.fit(X\_train, y\_train)

svc\_search.best\_estimator\_, svc\_search.best\_score\_

**4. confusion\_matrix**

from sklearn.metrics import confusion\_matrix

from sklearn.svm import SVC

svc = SVC(kernel = 'rbf', gamma = 10, C = 10)

svc.fit(X[::2], y\_class[::2])

y\_predict = svc.predict(X)

cm = confusion\_matrix(y\_class, y\_predict)

tn, fp, fn, tp = cm.reshape(-1,)

accuracy = (tn + tp) / (tn + fp + fn + tp)

precision = tp / (tp + fp)

recall = tp / (tp + fn)

coefs = LASSO.coef\_

dropped = np.array(range(X.shape[1]))[coefs==0]

**5. from sklearn.metrics import accuracy\_score, precision\_score, recall\_score**

yhat = linear\_classifier(X\_blob, w\_opt)

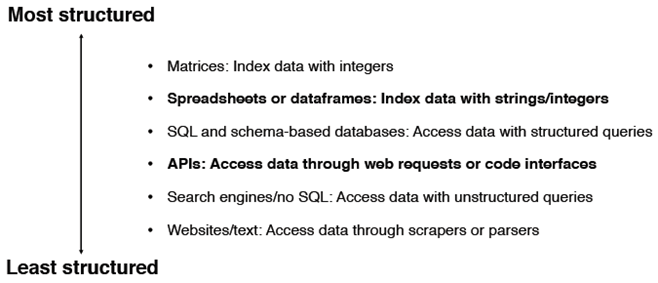
accuracy = accuracy\_score(y\_blob\_rescaled, yhat)

precision = precision\_score(y\_blob\_rescaled, yhat)

recall = recall\_score(y\_blob\_rescaled, yhat)

**4. Data Management**

**Topic1-Data\_Organization**

**1.**

**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

**mean\_no\_express = df\_no\_express.mean(axis = 0)**

**var\_no\_express = df\_no\_express.var(axis = 0)**

8) 返回所有含null 的行：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.**

**This is the raw text that describes the website, in this case it is HTML. It is possible to extract data directly from HTML, but it is challenging and tedious. Packages such as Beautiful Soup can make the process much easier, but we won't cover it in this course.**

3) import json

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

etoh = json.load(f)

4) print(etoh\_simple['PC\_Compounds'][0]['bonds'])

print(etoh\_simple['PC\_Compounds'][0]['atoms'])

？？ 如何读取的

**2. Application Programming Interfaces (RESTful API's)**

(1) First, we need to understand the structure of the query to decide how to search. From the documentation:

prolog: https://pubchem.ncbi.nlm.nih.gov/rest/pug

input: /compound/name/ethanol

operation: /cids

output: /TXT

r = requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/ethanol/cids/TXT')

print(r.text)

def get\_full(chemical): r=requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/{}/record/json'.format(chemical))

chem\_json = json.loads(r.text)

return chem\_json

**3. Python API's**

import pubchempy as pcpy

**4. Write a function that takes an arbitrary chemical name or CAS number and returns the number of C-H bonds.**

def countCH(name):

c = pcpy.get\_compounds(name, 'name')

bonds = c[0].bonds

count = 0

for bond in bonds:

if c[0].atoms[bond.aid1 - 1].element == 'C' and c[0].atoms[bond.aid2 - 1].element == 'H':

count += 1

elif c[0].atoms[bond.aid1 - 1].element == 'H' and c[0].atoms[bond.aid2 - 1].element == 'C':

count += 1

return count

countCH('ethanol')

**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) 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)

**2. def show\_image(digit\_data, n, ax=None):**

if ax is None:

fig, ax = plt.subplots()

img = digit\_data[n].reshape(8,8)

ax.imshow(img, cmap='binary', vmin=0, vmax=16)

n = 9

show\_image(X\_mnist, n)

print('Digit: {}, Min: {}, Max: {}'.format(y\_mnist[n],X\_mnist.min(),X\_mnist.max()))

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')Chart, scatter chart

Description automatically generated

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 Computing the accuracy of unsupervised models**

* 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**

**1. Features which are already provided are transformed in various ways to improve the efficiency or performance of the model.**

**2. One-Hot Encoding**

Graphical user interface, text, application

Description automatically generated

df\_onehot = pd.get\_dummies(df\_features\_output)

**3. Linear Feature Combinations**

def linear\_combination(X, W):

return X@W

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X, y\_dow)

r2\_LR = model.score(X, y\_dow)

r2s\_rand = []

m\_range = range(1, 50)

for m in m\_range:

W\_rand = np.random.rand(X.shape[1], m)

Xtilde\_rand = linear\_combination(X, W\_rand)

model.fit(Xtilde\_rand, y\_dow)

r2 = model.score(Xtilde\_rand, y\_dow)

r2s\_rand.append(r2)

**4.Partial Least Squares (PLS)** **a supervised dimensional reduction algorithm**

from sklearn.cross\_decomposition import PLSRegression

r2s\_PLS = []

m\_PLS = range(1, X.shape[1]) #PLS does not allow more components than original features

for m in m\_PLS:

model = PLSRegression(n\_components = m)

model.fit(X, y\_dow)

r2 = model.score(X, y\_dow)

r2s\_PLS.append(r2)

fig, ax = plt.subplots()

ax.plot(m\_range, [r2\_LR] \* len(m\_range), '-', label = 'All Features')

ax.plot(m\_range, r2s\_rand, '-', label = 'Random Projections')

ax.plot(40 \* np.ones(100), np.linspace(0, 0.75, 100), '--')

ax.plot(m\_range, r2s\_PCA, '-', label = 'PCA')

ax.plot(m\_PLS, r2s\_PLS, '-', label = 'PLS')

ax.set\_xlabel('# of Components')

ax.set\_ylabel('$r^2$ for Linear Model')

ax.legend();

**5. Linear Discriminant Analysis (LDA) Lecture code (LDA vs. PCA) 我的理解是用于classification problem降维**

The basic steps of LDA are:

* Compute the centerpoint of each class
* Compute intraclass covariance
* Compute interclass covariance
* Calculate the largest eigenvalues/eigenvectors of the composite covariance matrix
* Use hyperplanes perpendicular to the linear discriminant vectors to assign classes

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()

lda.fit(X\_mnist, y\_mnist)

X\_LDA = lda.transform(X\_mnist)

print(X\_LDA.shape)

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

pca = PCA(n\_components = 9)

X\_pca = pca.fit\_transform(X\_mnist)

kmeans = KMeans(n\_clusters = 10)

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

y\_predict\_pca = np.zeros(y\_mnist.shape)

for i in range(10):

collect\_label = y\_mnist[y\_pca == i]

count\_label = np.bincount(collect\_label)

common\_label = np.argmax(count\_label)

y\_predict\_pca[y\_pca == i] = common\_label

from sklearn.metrics import confusion\_matrix

import seaborn as sns

cm = confusion\_matrix(y\_mnist, y\_predict\_pca)

fig, ax = plt.subplots(figsize = (6, 5), dpi = 150)

sns.heatmap(cm, annot = True, fmt = 'd', ax = ax);

**6. Symbolic Regression (non-linear combinations of features)**

def assign\_labels(y = y\_mnist, y\_hat = y\_kmeans):

for i in range(0,10):

mostcommon=statistics.mode(y[y\_kmeans==i])

y[y\_kmeans==i]=mostcommon

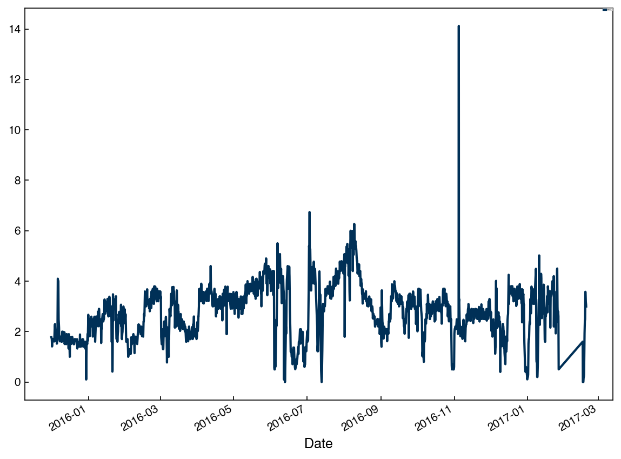
y\_predict=y

return y\_predict

**Topic2-Time\_Series\_Analysis**

**1. dow\_df = dow\_df.set\_index('Date')**

**2.dow\_df.plot()**



**3. Missing Values and Interpoloation**

* forward fill: use the prior value
* backward fill: use the subsequent value
* linear interpolation: average the prior and subsequent values
* non-linear interpolation: fit non-linear functions like splines using multiple prior/subsequent values

interp\_df = co2\_df

interp\_df['forward\_fill'] = interp\_df['co2'].fillna(method='ffill')

interp\_df['back\_fill'] = interp\_df['co2'].fillna(method='bfill')

interp\_df['linear\_interp'] = interp\_df['co2'].interpolate(method='linear')

interp\_df['spline\_interp'] = interp\_df['co2'].interpolate(method='spline', order=3)

**4. Moving Average Smoothing**

fig, ax = plt.subplots()

for window in [1,2, 4, 12, 26, 52]:

rolling\_df = co2\_df['co2\_interp'].rolling(window)

moving\_avg = rolling\_df.mean()

moving\_avg['01/01/1990':'01/01/2000'].plot(ax = ax, alpha = 0.5, label = str(window))

ax.legend();

**5. Exponential Smoothing**

def exp\_smoothing(x, alpha):

y = [x[0]] #start with x

for t in range(1, len(x)):

y.append(alpha \* x[t] + (1 - alpha) \* y[t-1])

return y

**6. Autocorrelation captures the fact that data will be correlated with itself at different times.**

from statsmodels.tsa.stattools import acf

autocorr = acf(dataset, nlags = 40)

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

\_ = plot\_acf(dataset, lags = 100)

\_ = plot\_pacf(dataset, lags = 100)

**7. Stationarity**

(1) A stationary dataset has the following properties:

* mean does not vary in time
* variance does not vary in time
* autocorrelation does not vary in time

(2) The Dickey-Fuller test evaluates the null hypothesis that a dataset is not stationary. If the resulting 𝑝 -value is high, then the dataset is not stationary.

from statsmodels.tsa.stattools import adfuller

p\_val = adfuller(co2\_df['co2\_interp'])[1]

print("Probability the data is stationary: {}".format(1 - p\_val))

**8. Differencing make a dataset more stationary**

We can do this very easily using the shift method of pandas, which simply shifts the entire dataframe.

co2\_df['co2\_diff'] = **co2\_df['co2\_interp'] - co2\_df['co2\_interp'].shift(1)**

co2\_df['co2\_diff'].plot();

**9. Model Fitting**

frequencies = [13, 23, 24, 25, 26, 26.5, 27]#, 27,28,29]

offsets = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

def make\_sine\_features(x, frequencies, offsets):

all\_feats = [x, x\*\*2]

names = ['x', 'x^2']

for freq in frequencies:

for offset in offsets:

new\_feat = np.sin((np.pi/freq)\*x - (offset/freq)\*np.pi)

all\_feats.append(new\_feat)

names.append('sin((pi\*x)/{} + {})'.format(freq, offset))

all\_feats = np.array(all\_feats).T

return all\_feats, names

X, names = make\_sine\_features(weeks, frequencies, offsets)

fig, ax = plt.subplots()

ax.plot(weeks, X[:, 2:])

ax.set\_xlim(0, 52)

ax.set\_title('Various Sine Curves')

ax.set\_xlabel('Weeks');

**10. ARIMA Modeling**

(1)先split 不是random的

train\_ratio = 0.75

N\_train = int(train\_ratio\*len(weeks))

N\_test = len(weeks) - N\_train

past\_weeks = weeks[:N\_train]

past\_co2 = y[:N\_train]

future\_weeks = weeks[-N\_test:]

future\_co2 = y[-N\_test:]

(2) predict past trend

past\_trend = model.predict(X\_past)

(3) residual reconstructed

recon\_resid = np.zeros(len(model\_resid))

recon\_resid[0] = model\_resid[0]

**for i in range(1, model\_resid\_diff.shape[0]):**

recon\_resid[i] = recon\_resid[i - 1] + model\_resid\_diff[i - 1]

past\_recon\_resid = recon\_resid[:N\_train]

(4) reconstructed past

recon\_past = np.array(past\_trend + past\_recon\_resid)

(5) construct future

X\_future, names = make\_sine\_features(future\_weeks, frequencies, offsets)

y\_future = model.predict(X\_future) + past\_recon\_resid[-1]

11. Auto-regressive Models

from statsmodels.tsa.arima\_model import ARIMA

model = ARIMA(train, order=(p,d,q))

p: 可以看partial autocorrelation 第一个peak

d: differencing 次数

q: 看autocorrelation的第一个peak