TERMINOLOGY

data set: 记录的集合

instance/sample/feature vector: 每条记录是关于一个事件或对象的描述，也称为特征向量

attribute/feature: 反映事件或对象在某方面的表现或性质的事项

label: 关于示例的结果的信息

classification: 预测的是离散值

regression: 预测的是连续值

clustering: 训练集中的样例自动形成若干组，对应一些潜在的规律划分

supervised learning: 训练数据有标记信息，包括分类和回归

unsupervised learning: 训练数据没有标记信息，包括聚类

overfitting (过拟合，过配): 学习器把训练样本自身的某些特点当做了所有潜在样本都会具有的一般性质，从而导致泛化 (generalization) 性能下降，方差大

underfitting (欠拟合，欠配): 学习器对训练样本的一般性质尚未学好，偏差大

评估方法: 1. hold-out (留出法): 直接将数据集划分为两个互斥的集合，其中一个作为训练集，另一个测试集，一般采用若干次随机划分、重复进行试验取均值

2. cross validation (交叉验证法): 将数据集划分为 k 个大小相似的互斥子集，每个子集都尽可能保持数据分布的一致性，通过分层采样得到，然后每次用 k-1 个子集的并集作为训练集，余下的子集作为测试集，这样就得到 k 组训练/测试集，取结果均值，又称为“k 折交叉验证”，一般采用 10 次 10 折交叉验证

3. bootstrap (自助法): 每次随机从数据集中取出一个样本，拷贝放入训练集中，然后将该样本放回初始数据集中，重复执行 m 次后，得到包含 m 个样本的训练集，可知一个样本不被取到的概率为 0.368，一般用于数据集较小且难以有效划分训练/测试集的情况

性能度量: 1. 回归任务常用均方误差 (mean squared error)

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}(f(\boldsymbol{x}_i)-y_i)^2$$

2. 分类任务常用错误率和精度

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i)\neq y_i)\quad \mathrm{acc}(f;D)\ =\ \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i)=y_i)$$

对于某些问题采用查准率 (precision) 、查全率 (recall) 和 F1-score

真实情况	预测结果	
	正例	反例
正例	TP (真正例)	FN (假反例)
反例	FP (假正例)	TN (真反例)

$$P=\frac{TP}{TP+FP}$$

$$R=\frac{TP}{TP+FN}$$

一般来说，查准率较高时，查全率较低；查全率较高时，查准率较低

作出查准率-查全率曲线图，曲线下的面积可作为比较不同学习器的性能优劣

F1-score 是基于查准率和查全率的调和平均数

$$F1=\frac{2\times P\times R}{P+R}=\frac{2\times TP}{\text{样例总数}+TP-TN}$$

偏差与方差: 偏差-方差分解 (bias-variance decomposition) 用于解释学习算法泛化性能

$$E(f;D)=bias^2(\boldsymbol{x})+var(\boldsymbol{x})+\varepsilon^2$$

泛化误差可分解为偏差 (度量学习算法的期望预测和真实结果的偏离程度，刻画法的拟合能力) 、方差 (度量同样大小的训练集的变化所导致的学习性能的变化，刻画数

据扰动所造成的影响) 和噪声 (度量在当前任务上任何学习算法所能达到的期望泛化误差的下界，刻画学习问题本身的难度) 之和

对于给定的学习任务，在训练不足时，学习器的拟合能力不够强，数据扰动不足以使学习器产生显著变化，此时偏差主导了泛化错误率；训练程度加深后，学习器拟合能力增强，数据扰动能够被学习器学到，方差逐渐主导泛化错误率；训练充分后，轻微的数据扰动都可能显著影响学习器，若训练数据中非全局的特性被学习器学到，则发生过拟合

LINEAR REGRESSION

给定由 d 个属性描述的示例 $x=(x_1;x_2;...;x_d)$，其中 x_i 是 x 在第 i 个属性上的取值，线性模型试图学得一个通过属性的线性组合来进行预测的函数，即 $f(x)=w_1x_1+w_2x_2+...+w_dx_d+b$

给定数据集 $D=\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$，其中 $x_i=(x_{i1};x_{i2};...;x_{id})$, $y_i\in R$，线性回归试图学得一个线性模型来预测实值输出标记，即 $f(x_i)=wx_i+b$ 使得 $f(x_i)\approx y_i$

常用最小二乘法 (均方误差最小化) 找到一条直线，使所有样本到直线上的欧氏距离之和最小

$$w=\frac{\sum_{i=1}^{m}y_i(x_i-\bar{x})}{\sum_{i=1}^{m}x_i^2-\frac{1}{m}\left(\sum_{i=1}^{m}x_i\right)^2} \qquad b=\frac{1}{m}\sum_{i=1}^{m}(y_i-wx_i)$$

1. **Import pandas, numpy, matplotlib,and seaborn. Then set %matplotlib inline**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

2. **Read in the Ecommerce Customers csv file as a DataFrame called customers**

```
customers = pd.read_csv("Ecommerce Customers")
```

Check the head of customers, and check out its info() and describe() methods

```
customers.head()
```

```
customers.describe()
```

```
customers.info()
```

customers.head(n=1): Return the first n rows

customers.describe(percentiles=None, include=None, exclude=None): Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding ``NaN`` values

customers.info(verbose=None, buf=None, max_cols=None, memory_usage=None,

null_counts=None): Concise summary of a DataFrame

3. **Use seaborn to create a jointplot to compare two columns**

```
sns.jointplot(x='Time on Website',y='Yearly Amount Spent',data=customers)
```

sns.jointplot(x, y, data=None, kind='scatter', …): Draw a plot of two variables with bivariate and univariate graphs

explore these types of relationships across the entire data set

```
sns.pairplot(customers)
```

sns.pairplot(data, …): Plot pairwise relationships in a dataset, by default, this function will create a grid of Axes such that each variable in ``data`` will by shared in the y-axis across a single row and in the x-axis across a single column

Create a linear model plot (using seaborn's lmplot)

sns.lmplot(x, y, data, …): Plot data and regression model fits across a FacetGrid

4. **split the data into training and testing sets, use model_selection.train_test_split from sklearn to split the data into training and testing sets**

```
y = customers['Yearly Amount Spent']
```

```
X = customers[['Avg. Session Length', 'Time on App','Time on Website', 'Length of Membership']]
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

train_test_split(*arrays, **options): Split arrays or matrices into random train and test subsets

5. **train our model on our training data, Import LinearRegression from sklearn.linear_model**

```
from sklearn.linear_model import LinearRegression
```

Create an instance of a LinearRegression() model named lm

```
lm = LinearRegression()
```

Train/fit lm on the training data

```
lm.fit(X_train,y_train)
```

lm.fit(X, y, sample_weight=None): Fit linear model, returns an instance of self

Print out the coefficients of the model

```
print('Coefficients: \n', lm.coef_)
```

coef_: attribute of linear regression model, Estimated coefficients for the linear regression problem

6. **evaluate its performance by predicting off the test values, use lm.predict() to predict off the X_test set of the data**

```
predictions = lm.predict( X_test)
```

lm.predict(X): Predict using the linear model, returns an array representing predicted values

Create a scatterplot of the real test values versus the predicted values

```
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

plt.scatter(x, y, …): A scatter plot of y vs x with varying marker size and/or color, returns a path (matplotlib.collections.PathCollection), where x and y are array-like representing data positions

plt.xlabel(s, …): Set the x-axis label of the current axes

plt.ylabel(s, …): Set the y-axis label of the current axes

7. **Calculate the Mean Absolute Error, Mean Squared Error, and the Root Mean Squared Error**

```
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

Plot a histogram of the residuals and make sure it looks normally distributed

```
sns.distplot((y_test-predictions),bins=50);
```

Recreate the concluded dataframe

```
coeffecients = pd.DataFrame(lm.coef_,X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

LOGISTIC REGRESSION

对于分类任务，常用对数几率函数作为线性模型，称为对数几率回归，即 $y=1/1+e^{-z}$，其中 z 是实值转换为的 0/1 值，是单调可微的任意阶可导的凸函数

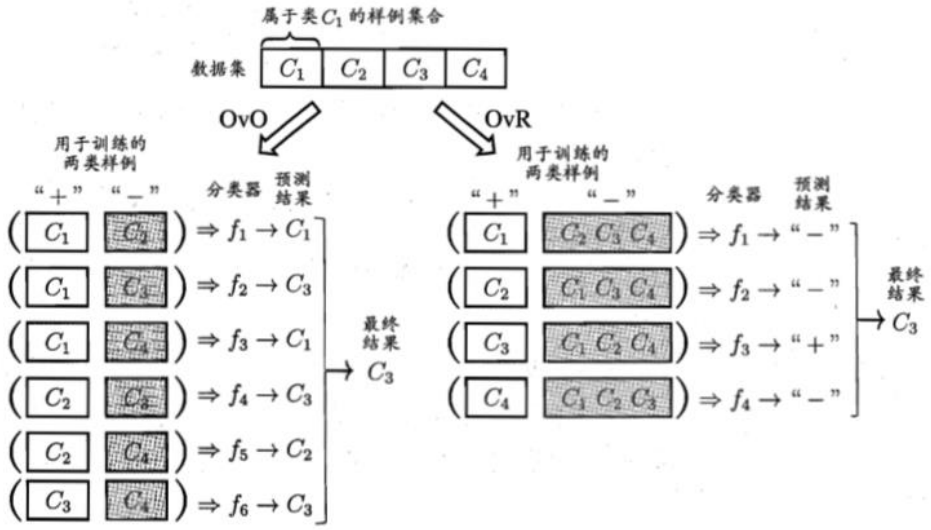
对于多分类学习任务，拆分为若干个二分类任务求解，即先对问题进行拆分，然后为拆分出的每个二分类任务训练一个分类器，集成各个分类器的预测结果得到最终的多分类结果

拆分策略: 1. 一对一 (one vs. one): 将数据集中的 N 个类别两两配对，从而产生 N(N-1)/2 个二分类任务，将预测得最多的类别作为最终分类结果

存储开销和测试时间开销较大，训练时间开销较小

2. 一对其余 (one vs. rest): 将一个类的样例作为正例，所有其他类的样例作为反例来训练 N 个分类器，若预测结果有多个正类，选择置信度最大的类别标记作为分类结果

存储开销和测试时间开销较小，训练时间开销较大



3. 多对多 (many vs. many): 每次将若干个类作为正类，若干个其他类作为反类，常用纠错输出码 (ECOC, error correcting output codes) 进行类别拆分

ECOC 工作过程: 1. 编码: 对 N 个类别做 M 次划分，每次划分将一部分类别划为正

类，一部分划为反类，形成一个二分类训练集，一共产生 M 个训练集，训练处 M 个分类器

2. 解码: M 个分类器分别对测试样本进行预测，预测标记组成一个编码，将这个预测编码与各个类别各自的编码进行比较，其中距离最小的类别作为最终预测结果

1. **Import libraries and read csv file into a pandas dataframe**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
train = pd.read_csv('titanic_train.csv')
```

2. **use seaborn to create a simple heatmap to see where we are missing data**

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

sns.heatmap(data, yticklabels=bool/list-list/int, cbar, cmap): Plot rectangular data as a color-encoded matrix, this is an Axes-level function and will draw the heatmap into the currently-active Axes if none is provided to the ax argument, where ax is the axes object to draw the plot onto

visualize some more of the data

```
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

sns.countplot(x=string, y=string, hue=string, data=dataframe, …): Show the counts of observations in each categorical bin using bars

```
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
```

sns.distplot(a=1d-array/list, bins=int, kde=bool, …): Flexibly plot a univariate distribution of observations

```
train['Age'].hist(bins=30,color='darkred',alpha=0.7)
```

dataframe/column.hist(bins=int, …): Draw histogram of the DataFrame's series using matplotlib

use cufflinks for plots

```
import cufflinks as cf
cf.go_offline()
```

```
train['Fare'].iplot(kind='hist',bins=30,color='green')
```

dataframe/column.iplot(kind=string, bins=int, …): Returns a plotly chart either as inline chart, image of Figure object

3. **fill in missing age data instead of just dropping the missing age data rows, check the average age by passenger class and fill in the mean age of all the passengers of this class**

```
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

sns.boxplot(x=string, y=string, data=dataframe, …): Draw a box plot to show distributions with respect to categories

define and apply a function to fill missing age data based on the passenger class

```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

column.apply(func, axis=0(row)/1(column), …): Applies function along input axis of DataFrame

drop the column which misses too much data and drop any other missing values

```
train.drop('Cabin',axis=1,inplace=True)
```

```
train.dropna(inplace=True)
```

dataframe.drop(labels=string, axis=0(row)/1(column), …): Return new object with labels in requested axis removed

4. **convert categorical features to dummy variables using pandas to make machine learning algorithm be able to take those features as inputs**

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

pd.get_dummies(data, …): Convert categorical variable into dummy/indicator variables

```
train = pd.concat([train,sex,embark],axis=1)
```

pd.concat(objs, axis=0(row)/1(column), …): Concatenate pandas objects along a particular axis with optional set logic along the other axes, objs is a sequence or mapping of dataframes

5. **split the data into a training set and test set, train and predict**

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

```
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
predictions = logmodel.predict(X_test)
```

6. **check precision, recall, f1-score using classification report**

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,predictions))
```

precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances

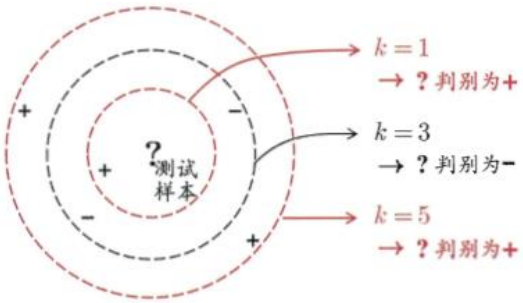
recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances

f1-score is the harmonic average of the precision and recall, its best value is 1 (perfect precision and recall) and worst is 0

K NEAREST NEIGHBORS

k 近邻学习是一种常用的监督学习方法，工作机制是给定测试样本，基于某种距离度量找出训练集中与其最靠近的 k 个训练样本，然后基于这 k 个邻居的信息来进行预测；通常在分类任务中使用投票法，在回归任务中使用平均法，也可以基于距离远近进行加权

k 近邻学习没有显式的训练过程，是懒惰学习 (lazy learning，对应于急切学习 eager learning) 的代表，在训练阶段仅仅是把样本保存起来，训练时间开销为零，待收到测试样本后再进行处理



最近邻分类器的泛化错误率不超过贝叶斯最优分类器的错误率的两倍

1. **Import libraries and read csv file into a pandas dataframe**

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

```
df = pd.read_csv("Classified Data",index_col=0)
```

index_col=0: use the first column as the index

2. **Standardize the variables**: Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(df.drop('TARGET CLASS',axis=1))
```

scaler.fit(X, …): Compute the mean and std to be used for later scaling

```
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
```

```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

scaler.transform(X, …): Perform standardization by centering and scaling

array[:-1]: it is all of the string except for its last character, which is useful for removing the trailing newline from a string

3. **split the data into train set and test set, use KNN model to train and predict**

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
                                                    test_size=0.30)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train,y_train)
```

```
pred = knn.predict(X_test)
```

4. **evaluate the predictions**

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,pred))
```

```
print(classification_report(y_test,pred))
```

5. **use the elbow method to pick a good K value**

```
error_rate = []

for i in range(1,30):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

draw a plot for the relation between error rate and K value, find the most suitable K value

```
plt.figure(figsize=(10,6))
plt.plot(range(1,30),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

DIMENSION REDUCTION AND METRIC LEARNING

降维 (dimension reduction) 是缓解维数灾难 (高维情形下出现的数据样本稀疏，距离计算困难) 的重要途径，通过数学变换将原始高维属性空间转变为一个低维子空间，在这个子空间中样本密度大幅提高，距离计算更加容易

低维嵌入: 与学习任务密切相关的低维分布

多维缩放 (multiple dimensional scaling, MDS) 使原始空间中样本之间的距离在低维空间中能够保持

假定 m 个样本在原始空间的距离矩阵为 $D\in R^{m*m}$，第 i 行 j 列的元素 $dist_{ij}$ 为样本 x_i 到 x_j 的距离，目标是获得样本在 d'维空间的表示 $Z\in R^{d'*m}$，$d'\leq d$，且任意两个样本在 d'维空间中的欧氏距离等于原始空间中的距离

令 $B=Z^TZ\in R^{m*m}$ 为降维后样本的内积矩阵，通过令降维后的样本 Z 被中心化，使内积矩阵的行与列之和均为零，得到

$$dist_{i\cdot}^2=\frac{1}{m}\sum_{j=1}^{m}dist_{ij}^2\ ,\qquad \sum_{i=1}^{m}dist_{ij}^2=\mathrm{tr}(\mathbf{B})+mb_{jj}$$

$$dist_{\cdot j}^2=\frac{1}{m}\sum_{i=1}^{m}dist_{ij}^2\ ,\qquad \sum_{j=1}^{m}dist_{ij}^2=\mathrm{tr}(\mathbf{B})+mb_{ii}$$

$$dist_{\cdot\cdot}^2=\frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}dist_{ij}^2\qquad \sum_{i=1}^{m}\sum_{j=1}^{m}dist_{ij}^2=2m\ \mathrm{tr}(\mathbf{B})$$

其中 tr(·)表示矩阵的迹 (trace)

$\mathrm{tr}(\mathbf{B})=\sum_{i=1}^{m}\|\boldsymbol{z}_i\|^2$

得到内积矩阵为

$$b_{ij}=-\frac{1}{2}(dist_{ij}^2-dist_{i\cdot}^2-dist_{\cdot j}^2+dist_{\cdot\cdot}^2)$$

对内积矩阵 B 做特征值分解，即 $B=V\Lambda V^T$，其中 $\Lambda=diag(\lambda_1,\lambda_2,\ldots,\lambda_d)$为特征值构成的对角矩阵，V 为特征向量矩阵；假定其中有 d*个非零特征值，构成对角矩阵 Λ_*，V_*为相应的特征向量矩阵；或取 $d'\ll d$ 个最大特征值构成的对角矩阵$\tilde{\Lambda}$，$\tilde{V}$为相应的特征向量矩阵，则

$$\mathbf{Z}=\mathbf{\Lambda}_*^{1/2}\mathbf{V}_*^{\mathrm{T}}\in\mathbb{R}^{d^*\times m}\qquad \mathbf{Z}=\tilde{\mathbf{\Lambda}}^{1/2}\tilde{\mathbf{V}}^{\mathrm{T}}\in\mathbb{R}^{d'\times m}$$

输入： 距离矩阵 $\mathbf{D} \in \mathbb{R}^{m \times m}$, 其元素 $dist_{ij}$ 为样本 $\boldsymbol{x}_i$ 到 $\boldsymbol{x}_j$ 的距离;
低维空间维数 d'.

过程：

1: 根据式(10.7)~(10.9)计算 $dist_{i\cdot}^2$, $dist_{\cdot j}^2$, $dist_{\cdot\cdot}^2$;
2: 根据式(10.10)计算矩阵 $\mathbf{B}$;
3: 对矩阵 $\mathbf{B}$ 做特征值分解;
4: 取 $\tilde{\mathbf{\Lambda}}$ 为 d' 个最大特征值所构成的对角矩阵, $\tilde{\mathbf{V}}$ 为相应的特征向量矩阵.

输出： 矩阵 $\tilde{\mathbf{V}}\tilde{\mathbf{\Lambda}}^{1/2} \in \mathbb{R}^{m \times d'}$, 每行是一个样本的低维坐标

主成分分析 (Principal Component Analysis, PCA) 是常用的降维方法，定义一个超平面，具有最近重构性 (样本点到这个超平面的距离都足够近) 和最大可分性 (样本点在这个超平面的投影都尽可能分开)

输入： 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
低维空间维数 d'.

过程：

1: 对所有样本进行中心化: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$;
2: 计算样本的协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$;
3: 对协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ 做特征值分解;
4: 取最大的 d' 个特征值所对应的特征向量 $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d'}$.

输出： 投影矩阵 $\mathbf{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{d'})$.

PCA 将原始高维空间转化为低维空间，舍弃了对应于最小的 d-d’个特征值的特征向量，使样本的采样密度增大，在一定程度上起到去噪的效果

当高维空间到低维空间的函数映射是非线性时，采用核化主成分分析 (kernelized PCA)

流形学习 (manifold learning) 是借鉴了拓扑流形概念 (在局部与欧氏空间同胚的空间，在局部具有欧氏空间的性质，能用欧氏距离进行距离计算) 的降维方法；若低维流形嵌入到高维空间中，则数据样本在高维空间的分布在局部上仍具有欧氏空间的性质，可以在局部建立起降维映射关系，再推广到全局，当维数降至二维或三维时，能对数据进行可视化

等度量映射 (Isometric Mapping) 认为不能直接在高维空间中计算直线距离，应使用近邻距离来近似，即基于近邻距离逼近能获得低维流形上测地线距离很好的近似，保持了近邻样本之间的距离；采用 Dijkstra 算法或 Floyd 算法来计算两点间的最短路径

输入: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
近邻参数 k;
低维空间维数 d'.

过程:

1: **for** $i=1,2,\ldots,m$ **do**
2: 确定 $\boldsymbol{x}_i$ 的 k 近邻;
3: $\boldsymbol{x}_i$ 与 k 近邻点之间的距离设置为欧氏距离, 与其他点的距离设置为无穷大;
4: **end for**
5: 调用最短路径算法计算任意两样本点之间的距离 $\mathrm{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j)$;
6: 将 $\mathrm{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j)$ 作为 MDS 算法的输入;
7: **return** MDS 算法的输出

输出: 样本集 D 在低维空间的投影 $Z=\{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_m\}$.

局部线性嵌入 (Locally Linear Embedding, LLE) 保持邻域内样本之间的线性关系

输入: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
近邻参数 k;
低维空间维数 d'.

过程:

1: **for** $i=1,2,\ldots,m$ **do**
2: 确定 $\boldsymbol{x}_i$ 的 k 近邻;
3: 从式(10.27)求得 w_{ij}, $j\in Q_i$;
4: 对于 $j\notin Q_i$, 令 $w_{ij}=0$;
5: **end for**
6: 从式(10.30)得到 $\mathbf{M}$;
7: 对 $\mathbf{M}$ 进行特征值分解;
8: **return** $\mathbf{M}$ 的最小 d' 个特征值对应的特征向量

输出: 样本集 D 在低维空间的投影 $Z=\{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_m\}$.

DECISION TREES AND RANDOM FORESTS

一棵决策树包含一个根结点、若干个内部结点和若干个叶结点

叶结点对应于决策结果，其他的每个结点对应于一个属性测试

每个结点包含的样本集合根据属性测试的结果被划分到子结点中，根结点包含样本全集，从根结点到每个叶结点的路径对应一个判定测试序列

决策树流程遵循分而治之策略，由递归过程生成

输入: 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
　　属性集 $A = \{a_1, a_2, \ldots, a_d\}$.
过程: 函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3: 　将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A = \varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6: 　将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10: 　为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11: 　**if** D_v 为空 **then**
12: 　　将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13: 　**else**
14: 　　以 TreeGenerate(D_v, $A \setminus \{a_*\}$)为分支结点
15: 　**end if**
16: **end for**
输出: 以 node 为根结点的一棵决策树

递归返回的情形: 1. 当前结点包含的样本全属于同一类别，无需划分

2. 当前属性集为空，或是所有样本在所有属性上取值相同，无法划分，此时将当前结点标记为叶结点，将其类别设定为该结点所含样本最多的类别
3. 当前结点包含的样本集合为空，不能划分，此时将当前结点标记为叶结点，将其类别设定为其父结点所含样本最多的类别

划分选择: 随着划分过程的进行，决策树的分支结点所包含的样本尽可能属于同一类别，即结点的纯度越来越高

1. information gain (信息增益) : 信息熵 (information entropy) 是度量样本集合纯度的指标 $\mathrm{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k$ p_k 为第 k 类样本所占比例， Ent(D)越小，纯度越高

   用属性 a 对样本集 D 进行划分所得的信息增益为

   $$\mathrm{Gain}(D, a) = \mathrm{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \mathrm{Ent}(D^v)$$

   信息增益越大，则使用属性 a 来进行划分所得的纯度提升越大，信息增益准则对可取值数目较多的属性有所偏好

2. 增益率 (gain ratio) : $\mathrm{Gain\_ratio}(D, a) = \frac{\mathrm{Gain}(D, a)}{\mathrm{IV}(a)}$ 其中 $\mathrm{IV}(a) = -\sum_{v=1}^{V} \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$ 为属性 a 的固有值，属性 a 的可能取值越多，即 V 越大，IV(a)的值越大，增益率准则对可能取值数目较少的属性有所偏好

   因此，先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的

剪枝 (prune) : 决策树有时会由于分支过多导致过拟合，剪枝是解决过拟合的主要手段

1. 预剪枝 (preprune) : 在决策树生成过程中，对每个结点在划分前先进行估计，若当前结点的划分不能带来决策树泛化性能提升，则停止划分并将当前结点标记为叶结点；预剪枝会使决策树中很多分支都没有进行判断，降低了过拟合的风险并且减少了决策树的训练时间和测试时间开销；预剪枝基于贪心 (greedy) 的本质禁止某些后续能够提升泛化性能的分支展开，会带来欠拟合的风险
2. 后剪枝 (postprune) : 先从训练集生成一棵完整的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将该子树替换为叶结点；后剪枝保留更多分支，欠拟合风险小，泛化性能一般更优，但训练时间开销较大

连续值处理: 采用二分法 (bi-partition) 对连续属性进行处理

假定连续属性 a 在 D 上有 n 个不同取值，排序后记为{a^1, a^2, …, a^n}，基于划分点 t 将 D 划分为子集D_t^-和D_t^+，可考察划分点集合 $\left\{\frac{a^i+a^{i+1}}{2}\middle|1\le i\le n-1\right\}$，信息增益为

$$\mathrm{Gain}(D,a)=\max_{t\in T_a}\ \mathrm{Gain}(D,a,t)\ =\max_{t\in T_a}\ \mathrm{Ent}(D)-\sum_{\lambda\in\{-,+\}}\frac{|D_t^\lambda|}{|D|}\mathrm{Ent}(D_t^\lambda)$$

选择使信息增益最大化的划分点来进行样本集合的划分

缺失值处理: 令$\tilde{D}$表示训练集 D 中在属性 a 上没有缺失值的样本子集，w_x为每个样本 x 的权重

$$\rho=\frac{\sum_{\boldsymbol{x}\in\tilde{D}}w_{\boldsymbol{x}}}{\sum_{\boldsymbol{x}\in D}w_{\boldsymbol{x}}}\quad\tilde{p}_k=\frac{\sum_{\boldsymbol{x}\in\tilde{D}_k}w_{\boldsymbol{x}}}{\sum_{\boldsymbol{x}\in\tilde{D}}w_{\boldsymbol{x}}}\quad\tilde{r}_v=\frac{\sum_{\boldsymbol{x}\in\tilde{D}^v}w_{\boldsymbol{x}}}{\sum_{\boldsymbol{x}\in\tilde{D}}w_{\boldsymbol{x}}}$$

p 表示无缺失值样本所占比例，$\widetilde{p_k}$表示无缺失值样本中第 k 类所占比例，$\widetilde{r_v}$表示无缺失值样本中在属性 a 上取值 a^v 的样本所占比例

$$\mathrm{Gain}(D,a)=\rho\times\left(\mathrm{Ent}\left(\tilde{D}\right)-\sum_{v=1}^{V}\tilde{r}_v\,\mathrm{Ent}\left(\tilde{D}^v\right)\right)$$

$$\mathrm{Ent}(\tilde{D})=-\sum_{k=1}^{|\mathcal{Y}|}\tilde{p}_k\log_2\tilde{p}_k$$

若样本 x 在划分属性 a 上的取值已知，则将 x 划入与其取值对应的子结点，且样本权值在子结点中保持为 w_x；若样本 x 在划分属性 a 上的取值未知，则将 x 同时划入所有子结点，且样本权值在与属性值 a^v 对应的子结点中调整为$\widetilde{r_v}$ * w_x，即让同一样本以不同的概率划入到不同的子结点中

由 d 个属性描述的样本对应了 d 维空间中的一个数据点，对样本分类则意味着在这个坐标空间中寻找不同类样本之间的分类边界，特点是轴平行，即分类边界由若干个与坐标轴平行的分段组成

multivariate decision tree (多变量决策树) : 也称为斜决策树，实现斜划分，非叶结点是多个属性的线性组合

随机森林 (random forest) 是 Bagging 的一个扩展变体，在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入了随机属性选择，即对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含 k 个属性的子集，然后再从这个子集中选择一个最优属性用于划分，k 代表随机程度，一般取值 $\log_2 d$，d 为属性集合总数

随机森林在 Bagging 样本扰动的基础上，增加了属性扰动，使基学习器多样性更强，提升了最终集成的泛化性能

1. **Import libraries and read csv file into a pandas dataframe**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
df = pd.read_csv('kyphosis.csv')
```

2. **split the data into a training set and a test set**

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop('Kyphosis',axis=1)
y = df['Kyphosis']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

3. **train a single decision tree, predict and evaluate**

```
from sklearn.tree import DecisionTreeClassifier
```

```
dtree = DecisionTreeClassifier()
```

```
dtree.fit(X_train,y_train)
```

```
predictions = dtree.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(classification_report(y_test,predictions))
```

4. **use SciKit learn built-in visualization for decision trees** (optional)

```
from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydot

features = list(df.columns[1:])
```

```
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,feature_names=features,filled=True,rounded=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

5. **train a random forest and compare it with decision tree model**

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
```

```
rfc_pred = rfc.predict(X_test)
```

```
print(classification_report(y_test,rfc_pred))
```

BAYESIAN CLASSIFIER

贝叶斯决策论 (Bayesian decision theory) 是概率框架下实施决策的基本方法，对分类任务而言，在所有相关概率都已知的理想情况下，贝叶斯决策论考虑如何基于这些概率和误判损失来选择最优的类别标记

假设有 N 种可能的类别标记，$y=\{c_1, c_2, \ldots, c_N\}$, λ_{ij} 是将一个真实标记为 c_j 的样本误分类为 c_i 所产生的损失，则将样本 x 分类为 c_i 所产生的期望损失 (或样本 x 上的条件风险) 为

$$R(c_i \mid \boldsymbol{x}) = \sum_{j=1}^{N} \lambda_{ij} P(c_j \mid \boldsymbol{x})$$

总体风险为

$$R(h) = \mathbb{E}_{\boldsymbol{x}}\left[R\left(h\left(\boldsymbol{x}\right) \mid \boldsymbol{x}\right)\right]$$

贝叶斯判定准则: 为最小化总体风险，在每个样本上选择使条件风险 R(c|x)最小的类别标记，即

$$h^*(\boldsymbol{x}) = \arg\min_{c \in \mathcal{Y}} R(c \mid \boldsymbol{x})$$

其中 h^*称为贝叶斯最优分类器，$R(h^*)$称为贝叶斯风险，$1-R(h^*)$反映了分类器能达到的最好性能

使用贝叶斯判定准则来最小化决策风险时，要先获得后验概率 P(c|x)，常用判别式模型 (给定 x，通过直接建模 P(c|x)来预测 c) 或者生成式模型 (先对联合概率分布 P(x,c)建模，再由此得到 P(c|x))

对于生成式模型，$\mathrm{P(c|x)} = \frac{P(x,c)}{P(x)} = \frac{P(c)P(x|c)}{P(x)}$，其中 P(c)是先验概率，P(x|c)是样本 x 相对于类别标记的条件概率，也称为似然 (likelihood)，P(x)是用于归一化的证据因子

朴素贝叶斯分类器 (naïve Bayes classifier) 解决了条件概率作为所有属性的联合概率难以从有限的训练样本中直接估得的问题，采用属性条件独立性假设，$\mathrm{P(c|x)} = \frac{P(c)P(x|c)}{P(x)} = \frac{P(c)}{P(x)}\prod_{i=1}^{d} P(x_i|c)$，朴素贝叶斯分类器的表达式为$h_{nb}(x) = \underset{c \in y}{argmax}\, \mathrm{P(c)} \prod_{i=1}^{d} P(x_i|c)$

采用拉普拉斯修正避免因训练集样本不充分而导致概率估值为零的情况，即

$$\hat{P}(c) = \frac{|D_c|+1}{|D|+N} \qquad \hat{P}(x_i \mid c) = \frac{|D_{c,x_i}|+1}{|D_c|+N_i} \qquad p(x_i \mid c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

其中 $\mu_{c,i}$和$\sigma_{c,i}^2$分别是第 c 类样本在第 i 个属性上取值的均值和方差

半朴素贝叶斯分类器 (semi-naïve Bayes classifier) 的基本想法是适当考虑一部分属性间的相互依赖信息，从而既不需要进行完全联合概率计算，又不至于忽略较强的属性依赖关系

独依赖估计 (one dependent estimator) 是常用策略，假设每个属性在类别之外最多仅依赖于一个其他属性，即$\mathrm{P(c|x)} \propto \mathrm{P(c)} \prod_{i=1}^{d} P(x_i|c, pa_i)$，其中$pa_i$是属性$x_i$所依赖的属性，称为$x_i$的父属性

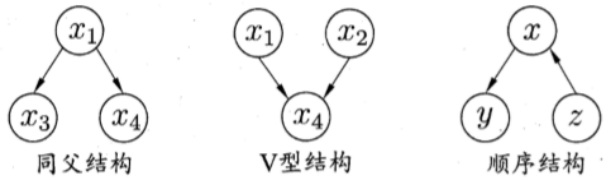
常用方法是 AODE (averaged one dependent estimator)， 是一种基于集成学习机制的独依赖分类器，假设所有属性都依赖于同一个超父属性，尝试将每个属性作为超父来构建模型，然后将那些具有足够训练数据支撑的模型集成起来作为最终结果，即

$$P(c \mid \boldsymbol{x}) \propto \sum_{\substack{i=1 \\ |D_{x_i}| \geqslant m'}}^{d} P(c, x_i) \prod_{j=1}^{d} P(x_j \mid c, x_i)$$

$$\hat{P}(c, x_i) = \frac{|D_{c,x_i}| + 1}{|D| + N_i} \quad \hat{P}(x_j \mid c, x_i) = \frac{|D_{c,x_i,x_j}| + 1}{|D_{c,x_i}| + N_j}$$

贝叶斯网 (Bayesian network/belief network 信念网) 借助有向无环图来刻画属性之间的依赖关系，使用条件概率表来描述属性的联合概率分布，由结构和参数两部分构成，每个结点对应于一个属性，若两个属性有直接依赖关系，则将它们由一条边连接起来，参数定量描述这种依赖关系

贝叶斯网中三个变量之间的典型依赖关系为



1. 同父结构中，给定父结点的取值，则两个子结点条件独立

2. V 型结构 (也称为冲撞结构) 中，给定子结点的取值，两个父结点必不独立；若子结点取值未知，两个父结点相互独立；这种独立性称为边际独立性 (marginal independence)

3. 顺序结构中，给定 x 的值，则 y 与 z 条件独立

道德图 (moral graph) : 为了分析有向图变量间的条件独立性，可使用有向分离，将有向图转变为无向图，即找出有向图中所有 V 型结构，在 V 型结构的两个父结点之间加上一条无向边，再所有有向边改为无向边

EM 算法: 常用于估计参数隐变量，是一种迭代式的方法

令 X 表示已观测变量集，Z 表示隐变量集，θ表示模型参数，若想对θ做极大似然估计，则需要最大化对数似然 LL(θ|X, Z) = lnP(X, Z|θ)，对 Z 计算期望，最大化已观测数据的对数边际似然 LL(θ|X) = lnP(X|θ) = $\ln \sum_Z P(X, Z|\theta)$

若参数θ已知，则可根据训练数据推断出最优隐变量 Z 的值 (E 步)；若 Z 的值已知，则可对参数θ做极大似然估计 (M 步)

以初始值θ^0为起点，进行迭代直至收敛: 1. 基于θ^t推断隐变量 Z 的期望，记为 Z^t

2. 基于已观测变量 X 和 Z^t对参数θ做极大似然估计，记为θ^{t+1}

ENSEMBLE LEARNING

集成学习 (ensemble learning) 通过构建并结合多个学习器来完成学习任务

集成中只包含同种类型的个体学习器，称为同质的 (homogeneous)，其中的个体学习器称为基学习器；包含不同类型的个体学习器，称为异质的 (heterogenous)，其中的个体学习器称为组件学习器 (component learner)

要获得好的集成，个体学习器应具有一定的准确性并且互相之间有差异；根据个体学习器的生成方式，集成学习方法可分为: 1. 个体学习器间存在强依赖关系，必须串行生成的序列化方法，代表为 Boosting

2. 个体学习器间不存在强依赖关系，可同时生成的并行化方法，代表为 Bagging 和随机森林

Boosting: 是一类可将弱学习器提升为强学习器的算法，先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本分布进行调整，使之前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器，重复进行此过程，直至基学习器数量达到事先指定的值，最后将这些基学习器进行加权结合

AdaBoost (自适应增强) 算法: 常用于二分类任务，可由加性模型推导

输入: 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程:

初始化样本权值分布.
1: $\mathcal{D}_1(\boldsymbol{x}) = 1/m$.

2: **for** $t = 1, 2, \ldots, T$ **do**

基于分布 $\mathcal{D}_t$ 从数据集 D 中训练出分类器 h_t.
3: $h_t = \mathfrak{L}(D, \mathcal{D}_t)$;

估计 h_t 的误差.
4: $\epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;

5: **if** $\epsilon_t > 0.5$ **then break**

确定分类器 h_t 的权重.
6: $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$;

更新样本分布，其中 Z_t 是规范化因子，以确保 $\mathcal{D}_{t+1}$ 是一个分布.
7: $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
$= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$

8: **end for**

输出: $H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$

Boosting 算法要求基学习器能对特定的数据分布进行学习，可通过重赋权法 (在训练过程中的每一轮中，根据样本分布为每个训练样本重新赋予一个权重) 或重采样法 (在每一轮学习中，根据样本分布对训练集重新进行采样，再用重采样而得的样本集对基学习器进行训练) 实现

Boosting 算法主要关注降低偏差 (bias)，因为在迭代过程中逐渐逼近真实值，因此能基于泛化性能很弱的学习器构建出很强的集成

Bagging: 是并行式集成学习方法的代表，基于自助采样法 (bootstrap sampling)，对每个采样集训练出一个基学习器，再将这些基学习器进行结合，对分类任务使用简单投票法，对回归任务使用简单平均法 (每个基学习器使用相同权重)，可用于多分类、回归任务

输入: 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程:

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$
3: **end for**

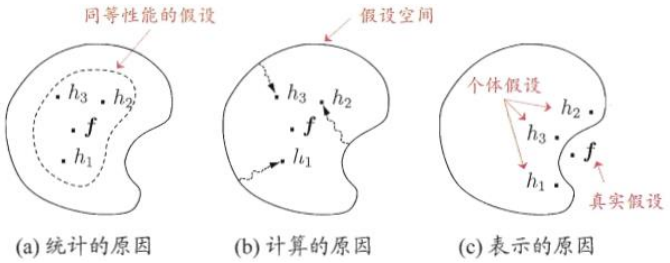
输出: $H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y)$

Bagging 算法主要关注降低方差，因为通过自助采样法和平均法降低了强学习器的过拟合

学习器结合: 1. 统计方面来看，由于学习任务的假设空间很大，可能有多个假设在训练集上达到同等性能，此时若使用单学习器可能因误选而导致泛化性能不佳

2. 计算方面来看，学习算法往往会陷入局部极小，有的局部极小点对应的泛化性能可能不佳

3. 表示方面来看，某些学习任务的真实假设可能不在当前学习算法所考虑的假设空间中，此时使用单学习器则无效



结合方法: 1.平均法: 简单平均法，加权平均法；个体学习器性能相差较大时常用加权平均法，个体学习器性能相近时常用简单平均法

2. 投票法: 绝对多数投票法 (若某标记得票过半，则预测为该标记，否则拒绝预测)，相对多数投票法 (预测为得票最多的标记)，加权投票法

3. 学习法: 通过另一个学习器来进行结合

Stacking 算法: 先从初始数据集训练出初级学习器，然后生成一个新数据集用于训练次级学习器，在新数据集中，初级学习器的输出被当做样例输入特征，而初始样本的标记仍被当做样例标记

输入: 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
初级学习算法 $\mathfrak{L}_1, \mathfrak{L}_2, \ldots, \mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.
过程:
1: **for** $t = 1, 2, \ldots, T$ **do**
2: $h_t = \mathfrak{L}_t(D)$;
3: **end for**
4: $D' = \varnothing$;
5: **for** $i = 1, 2, \ldots, m$ **do**
6: **for** $t = 1, 2, \ldots, T$ **do**
7: $z_{it} = h_t(\boldsymbol{x}_i)$;
8: **end for**
9: $D' = D' \cup ((z_{i1}, z_{i2}, \ldots, z_{iT}), y_i)$;
10: **end for**
11: $h' = \mathfrak{L}(D')$;
输出: $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$

个体学习器的分歧 (ambiguity) 为 $\mathrm{A}(h_i|x) = (h_i(x) - H(x))^2$，集成的分歧为 $\sum_{i=1}^{T} w_i A(h_i|x)$

分歧表征了个体学习器在样本 x 上的不一致性，在一定程度上反映了个体学习器的多样性

误差-分歧分解 (error-ambiguity decomposition) : $\mathrm{E} = \bar{E} - \bar{A}$，集成的泛化误差等于个体学习器泛化误差的加权均值和个体学习器的加权分歧值的差值；个体学习器准确性越高、多样性越大，则集成表现越好

CLUSTERING

聚类 (clustering) 是一种无监督学习 (训练样本的标记是未知的，通过对无标记训练样本的学习来揭示数据的内在性质和规律) 任务，聚类将数据集中的样本划分为若干个通常是不相交的子集，每个子集称为一个簇 (cluster)

聚类性能度量: 也称为聚类有效性指标，评估聚类结果的好坏；聚类结果的簇内相似度高且簇间相似度低，则聚类结果性能高

1. 将聚类结果和某个参考模型进行比较，称为外部指标

$$
\begin{aligned}
a &= |SS|, \quad SS = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j)\}, \\
b &= |SD|, \quad SD = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j)\}, \\
c &= |DS|, \quad DS = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j)\}, \\
d &= |DD|, \quad DD = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j)\}
\end{aligned}
$$

其中 λ 和 λ^*分别表示聚类簇划分和参考模型簇划分所对应的簇标记向量

Jaccard 系数: $\mathrm{JC} = \frac{a}{a+b+c}$

FM 指数: $\mathrm{FMI} = \sqrt{\frac{a}{a+b} \times \frac{a}{a+c}}$

Rand 指数: $\mathrm{RI} = \frac{2(a+d)}{m(m-1)}$，其中 m 是数据集中样本总数

外部指标的结果值均在[0, 1]区间，值越大聚类性能越好

2. 直接考察聚类结果而不利用任何参考模型，称为内部指标

$$
\begin{aligned}
\mathrm{avg}(C) &= \frac{2}{|C|(|C|-1)} \sum_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
\mathrm{diam}(C) &= \max_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) , \\
d_{\min}(C_i, C_j) &= \min_{\boldsymbol{x}_i \in C_i, \boldsymbol{x}_j \in C_j} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) , \\
d_{\mathrm{cen}}(C_i, C_j) &= \mathrm{dist}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) ,
\end{aligned}
$$

其中 dist(a, b)用于计算两个样本之间的距离，μ 代表簇的中心点 ($\frac{1}{|C|}\sum_{1 \leq i \leq |C|} x_i$)，

avg(C)表示簇 C 内样本间的平均距离，diam(C)表示簇 C 内样本间的最远距离，d_{min}

表示簇 C_i 和簇 C_j 最近样本间的距离，d_{cen} 表示簇 C_i 和簇 C_j 中心点间的距离

DB 指数: $\mathrm{DBI} = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i} \left(\frac{avg(C_i)+avg(C_j)}{d_{cen}(\mu_i, \mu_j)}\right)$

Dunn 指数: $\mathrm{DI} = \min_{1 \leq i \leq k} \left\{ \min_{j \neq i} \left(\frac{d_{min}(C_i, C_j)}{max_{1 \leq l \leq k} diam(C_l)} \right) \right\}$

DBI 的值越小聚类性能越好，DI 的值越大聚类性能越好

距离计算满足非负性、同一性、对称性和直递性的基本性质

对于连续属性，或称有序属性，常用闵可夫斯基距离

$$\mathrm{dist}_{\mathrm{mk}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{u=1}^{n} |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}}$$

欧氏距离 (p=2)

$$\mathrm{dist}_{\mathrm{ed}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2 = \sqrt{\sum_{u=1}^{n} |x_{iu} - x_{ju}|^2}$$

曼哈顿距离 (p=1)

$$\mathrm{dist}_{\mathrm{man}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_1 = \sum_{u=1}^{n} |x_{iu} - x_{ju}|$$

对于离散属性，或称无序属性，常用 VDM (value difference metric) 距离

$$\mathrm{VDM}_p(a, b) = \sum_{i=1}^{k} \left| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right|^p$$

若有 n_c 个有序属性，$n-n_c$ 个无序属性，结合式为

$$\mathrm{MinkovDM}_p(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{u=1}^{n_c} |x_{iu} - x_{ju}|^p + \sum_{u=n_c+1}^{n} \mathrm{VDM}_p(x_{iu}, x_{ju}) \right)^{\frac{1}{p}}$$

原型聚类 (prototype-based clustering) 也称为基于原型的聚类，这类算法假设聚类结构能通过一组原型刻画

1. k 均值算法，可看作高斯混合聚类在混合成分方差相等且每个样本仅指派给一个混合成分时的特例，仅在凸形簇结构 (形似椭球的簇结构) 上效果较好

2. 学习向量量化 (learning vector quantization) 假设数据样本带有类别标记，学习过程利用样本的这些监督信息来辅助聚类

输入: 样本集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
原型向量个数 q, 各原型向量预设的类别标记 $\{t_1,t_2,\ldots,t_q\}$;
学习率 $\eta\in(0,1)$.

过程:
初始化一组原型向量 $\{\boldsymbol{p}_1,\boldsymbol{p}_2,\ldots,\boldsymbol{p}_q\}$
repeat
  从样本集 D 随机选取样本 $(\boldsymbol{x}_j,y_j)$;
  计算样本 $\boldsymbol{x}_j$ 与 $\boldsymbol{p}_i$ $(1\leqslant i\leqslant q)$ 的距离: $d_{ji}=\|\boldsymbol{x}_j-\boldsymbol{p}_i\|_2$;
  找出与 $\boldsymbol{x}_j$ 距离最近的原型向量 p_{i^*}, $i^*=\arg\min_{i\in\{1,2,\ldots,q\}} d_{ji}$;
  if $y_j=t_{i^*}$ **then**
    $\boldsymbol{p}'=\boldsymbol{p}_{i^*}+\eta\cdot(\boldsymbol{x}_j-\boldsymbol{p}_{i^*})$
  else
    $\boldsymbol{p}'=\boldsymbol{p}_{i^*}-\eta\cdot(\boldsymbol{x}_j-\boldsymbol{p}_{i^*})$
  end if
  将原型向量 $\boldsymbol{p}_{i^*}$ 更新为 $\boldsymbol{p}'$
until 满足停止条件

输出: 原型向量 $\{\boldsymbol{p}_1,\boldsymbol{p}_2,\ldots,\boldsymbol{p}_q\}$

若最近的原型向量 p_{i^*}与 x_j 的类别标记相同，则令 p_{i^*}向 x_j 方向靠拢；若最近的原型向量 p_{i^*}与 x_j 的类别标记不同，则令 p_{i^*}远离 x_j 方向

3. 高斯混合聚类采用概率模型来表达聚类原型

输入: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
高斯混合成分个数 k.

过程:
初始化高斯混合分布的模型参数 $\{(\alpha_i,\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\mid 1\leqslant i\leqslant k\}$
repeat
  for $j=1,2,\ldots,m$ **do**
    根据式(9.30)计算 $\boldsymbol{x}_j$ 由各混合成分生成的后验概率, 即 $p_{\mathcal{M}}(z_j=i\mid\boldsymbol{x}_j)=\dfrac{P(z_j=i)\cdot p_{\mathcal{M}}(\boldsymbol{x}_j\mid z_j=i)}{p_{\mathcal{M}}(\boldsymbol{x}_j)}$
    $\gamma_{ji}=p_{\mathcal{M}}(z_j=i\mid\boldsymbol{x}_j)$ $(1\leqslant i\leqslant k)$ $=\dfrac{\alpha_i\cdot p(\boldsymbol{x}_j\mid\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k}\alpha_l\cdot p(\boldsymbol{x}_j\mid\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)}$.
  end for
  for $i=1,2,\ldots,k$ **do**
    计算新均值向量: $\boldsymbol{\mu}_i'=\dfrac{\sum_{j=1}^{m}\gamma_{ji}\boldsymbol{x}_j}{\sum_{j=1}^{m}\gamma_{ji}}$;
    计算新协方差矩阵: $\boldsymbol{\Sigma}_i'=\dfrac{\sum_{j=1}^{m}\gamma_{ji}(\boldsymbol{x}_j-\boldsymbol{\mu}_i')(\boldsymbol{x}_j-\boldsymbol{\mu}_i')^{\mathrm{T}}}{\sum_{j=1}^{m}\gamma_{ji}}$;
    计算新混合系数: $\alpha_i'=\dfrac{\sum_{j=1}^{m}\gamma_{ji}}{m}$;
  end for
  将模型参数 $\{(\alpha_i,\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\mid 1\leqslant i\leqslant k\}$ 更新为 $\{(\alpha_i',\boldsymbol{\mu}_i',\boldsymbol{\Sigma}_i')\mid 1\leqslant i\leqslant k\}$
until 满足停止条件
$C_i=\varnothing$ $(1\leqslant i\leqslant k)$
for $j=1,2,\ldots,m$ **do**
  根据式(9.31)确定 $\boldsymbol{x}_j$ 的簇标记 λ_j;
  将 $\boldsymbol{x}_j$ 划入相应的簇: $C_{\lambda_j}=C_{\lambda_j}\bigcup\{\boldsymbol{x}_j\}$
end for

输出: 簇划分 $\mathcal{C}=\{C_1,C_2,\ldots,C_k\}$

密度聚类假设聚类结构能通过样本分布的紧密程度确定，通常从样本密度的角度考察样本之间的可连接性，基于可连接样本不断扩展聚类簇以获得最终的聚类结果

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 是一种常用的密度聚类算法，基于一组邻域参数来刻画样本分布的紧密程度

基本概念: 1. ϵ-邻域: 包含样本集中与 x_j 的距离不大于 ϵ 的样本

2. 核心对象: 该样本的 ϵ-邻域至少包含 MinPts 个样本

3. 密度直达: 若 x_j 位于 x_i 的 ϵ-邻域中，且 x_i 是核心对象，则 x_j 由 x_i 密度直达；不满足对称性

4. 密度可达: 对 x_i 与 x_j，若存在样本序列 $p_1, p_2, \ldots, p_n$，其中 $p_1=x_i, p_n=x_j$ 且 p_{i+1} 由 p_i 密度直达，则 x_j 由 x_i 密度可达；满足直递性，不满足对称性

5. 密度相连: 对 x_i 与 x_j，若存在 x_k 使得 x_i 与 x_j 均由 x_k 密度可达，则 x_i 与 x_j 密度相连；满足对称性

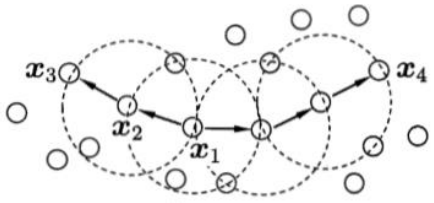


图 9.8 DBSCAN 定义的基本概念$(MinPts = 3)$: 虚线显示出 ϵ-邻域, x_1 是核心对象, x_2 由 x_1 密度直达, x_3 由 x_1 密度可达, x_3 与 x_4 密度相连.

DBSCAN 将簇定义为由密度可达关系导出的最大的密度相连样本集合，即簇满足连接性 ($x_i \epsilon C$，$x_j \epsilon C$，则 x_i 与 x_j 密度相连) 和最大性 ($x_i \epsilon C$，x_j 由 x_i 密度可达，则 $x_j \epsilon C$)

DBSCAN 算法先任选数据集中的一个核心对象，再由此出发确定其密度可达的所有样本集合，即为对应的聚类簇，样本集中不属于任何簇的样本则被认为是噪声或异常样本

输入: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
邻域参数 $(\epsilon, MinPts)$.

过程:

初始化核心对象集合: $\Omega=\varnothing$
for $j=1,2,\ldots,m$ **do**
  确定样本 $\boldsymbol{x}_j$ 的 ϵ-邻域 $N_\epsilon(\boldsymbol{x}_j)$;
  if $|N_\epsilon(\boldsymbol{x}_j)|\geqslant MinPts$ **then**
    将样本 $\boldsymbol{x}_j$ 加入核心对象集合: $\Omega=\Omega\bigcup\{\boldsymbol{x}_j\}$
  end if
end for
初始化聚类簇数: $k=0$
初始化未访问样本集合: $\Gamma=D$
while $\Omega\neq\varnothing$ **do**
  记录当前未访问样本集合: $\Gamma_{\text{old}}=\Gamma$;
  随机选取一个核心对象 $\boldsymbol{o}\in\Omega$, 初始化队列 $Q=<\boldsymbol{o}>$;
  $\Gamma=\Gamma\setminus\{\boldsymbol{o}\}$;
  while $Q\neq\varnothing$ **do**
    取出队列 Q 中的首个样本 $\boldsymbol{q}$;
    if $|N_\epsilon(\boldsymbol{q})|\geqslant MinPts$ **then**
      令 $\Delta=N_\epsilon(\boldsymbol{q})\bigcap\Gamma$;
      将 Δ 中的样本加入队列 Q;
      $\Gamma=\Gamma\setminus\Delta$;
    end if
  end while
  $k=k+1$, 生成聚类簇 $C_k=\Gamma_{\text{old}}\setminus\Gamma$;
  $\Omega=\Omega\setminus C_k$
end while

输出: 簇划分 $\mathcal{C}=\{C_1,C_2,\ldots,C_k\}$

层次聚类试图在不同层次对数据集进行划分，从而形成树形的聚类结构，可采用自底向上的聚合策略或自顶向下的分拆策略

AGNES (Agglomerative Nesting) 是一种采用自底向上聚合策略的层次聚类算法，先将数据集的每个样本看作一个初始聚类簇，然后在算法运行的每一步中找出距离最近的两个聚类簇进行合并，不断重复该过程直至达到预设的聚类簇个数

最小距离: $d_{\min}(C_i,C_j)=\min\limits_{\boldsymbol{x}\in C_i,\boldsymbol{z}\in C_j}\text{dist}(\boldsymbol{x},\boldsymbol{z})$,

最大距离: $d_{\max}(C_i,C_j)=\max\limits_{\boldsymbol{x}\in C_i,\boldsymbol{z}\in C_j}\text{dist}(\boldsymbol{x},\boldsymbol{z})$,

平均距离: $d_{\text{avg}}(C_i,C_j)=\dfrac{1}{|C_i||C_j|}\sum\limits_{\boldsymbol{x}\in C_i}\sum\limits_{\boldsymbol{z}\in C_j}\text{dist}(\boldsymbol{x},\boldsymbol{z})$

使用最小距离，AGNES 算法称为单链接；使用最大距离，AGNES 算法称为全链接；使用平均距离，AGNES 算法称为均链接

输入: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
聚类簇距离度量函数 d;
聚类簇数 k.

过程:

for $j=1,2,\ldots,m$ **do**
  $C_j=\{\boldsymbol{x}_j\}$
end for
for $i=1,2,\ldots,m$ **do**
  for $j=1,2,\ldots,m$ **do**
    $M(i,j)=d(C_i,C_j)$;
    $M(j,i)=M(i,j)$
  end for
end for
设置当前聚类簇个数: $q=m$
while $q>k$ **do**
  找出距离最近的两个聚类簇 C_{i^*} 和 C_{j^*};
  合并 C_{i^*} 和 C_{j^*}: $C_{i^*}=C_{i^*}\bigcup C_{j^*}$;
  for $j=j^*+1,j^*+2,\ldots,q$ **do**
    将聚类簇 C_j 重编号为 C_{j-1}
  end for
  删除距离矩阵 M 的第 j^* 行与第 j^* 列;
  for $j=1,2,\ldots,q-1$ **do**
    $M(i^*,j)=d(C_{i^*},C_j)$;
    $M(j,i^*)=M(i^*,j)$
  end for
  $q=q-1$
end while

输出: 簇划分 $\mathcal{C}=\{C_1,C_2,\ldots,C_k\}$

K MEANS CLUSTERING

对样本集 D={x_1, x_2, …, x_m}，k 均值算法针对聚类所得的簇划分 C={C_1, C_2, …, C_k}最小化平方误差

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} ||\boldsymbol{x} - \boldsymbol{\mu}_i||_2^2$$

其中$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ 是簇 C_i 的均值向量，该式刻画了簇内样本围绕簇均值向量的紧密程度，E 值越小则簇内样本相似度越高

k 均值算法采用贪心策略，通过迭代优化来近似求解

输入: 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
聚类簇数 k.

过程:

从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$
repeat
  令 $C_i = \varnothing \ (1 \leqslant i \leqslant k)$
  for $j = 1, 2, \ldots, m$ **do**
    计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu}_i \ (1 \leqslant i \leqslant k)$ 的距离: $d_{ji} = ||\boldsymbol{x}_j - \boldsymbol{\mu}_i||_2$;
    根据距离最近的均值向量确定 $\boldsymbol{x}_j$ 的簇标记: $\lambda_j = \arg\min_{i \in \{1,2,\ldots,k\}} d_{ji}$;
    将样本 $\boldsymbol{x}_j$ 划入相应的簇: $C_{\lambda_j} = C_{\lambda_j} \bigcup \{\boldsymbol{x}_j\}$;
  end for
  for $i = 1, 2, \ldots, k$ **do**
    计算新均值向量: $\boldsymbol{\mu}'_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
    if $\boldsymbol{\mu}'_i \neq \boldsymbol{\mu}_i$ **then**
      将当前均值向量 $\boldsymbol{\mu}_i$ 更新为 $\boldsymbol{\mu}'_i$
    else
      保持当前均值向量不变
    end if
  end for
until 当前均值向量均未更新

输出: 簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

1. **Import libraries**

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from sklearn.datasets import make_blobs
```

2. **create some data and visualize**

```
data = make_blobs(n_samples=200, n_features=2, centers=4, cluster_std=1.8,random_state=101)
```

make_blobs(n_samples: the total number of points equally divided among clusters, n_features: the number of features for each sample, centers: the number of centers to generate, cluster_std: the

standard deviation of the clusters, …): Generate isotropic Gaussian blobs for clustering, returns an array[X][Y], X is the generated samples, Y is the integer labels for cluster membership of each sample

```
plt.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')
```

3. **create the clusters**

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=4)
```

```
kmeans.fit(data[0])
```

show the performance of the clusters

```
kmeans.cluster_centers_
```

cluster_centers_: attribute for coordinates of cluster centers

```
kmeans.labels_
```

labels_: attribute for labels of each point

```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels_,cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')
```

SUPPORT VECTOR MACHINES

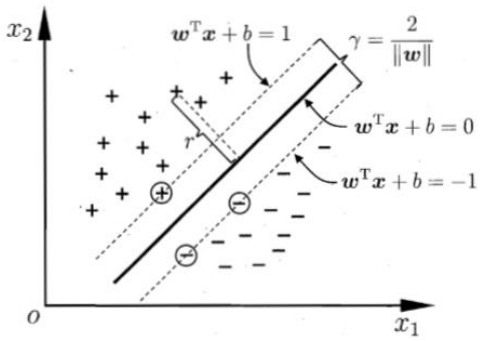
分类学习中，基于训练集在样本空间中找到一个划分超平面，将不同类别的样本分开，该超平面所产生的分类结果鲁棒性最好，对未见示例的泛化能力最强

划分超平面的线性方程为 $w^Tx+b=0$，其中 w 为法向量，决定超平面的方向；b 为位移项，决定超平面与原点之间的距离；样本空间中任意点 x 到超平面距离为$r = \frac{|w^Tx+b|}{||w||}$

假设超平面能将训练样本正确分类，则

$$\begin{cases} \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b \geqslant +1, & y_i = +1 \\ \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b \leqslant -1, & y_i = -1 \end{cases}$$

距离超平面最近的训练样本点使等号成立，称为支持向量 (support vector)，两个异类支持向量到超平面的距离之和为$\gamma = \frac{2}{||w||}$，称为间隔 (margin)



支持向量机基本型: 具有最大间隔的划分超平面，满足

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geqslant 1, \quad i = 1, 2, \ldots, m. \end{aligned}$$

可通过凸二次规划问题优化计算包求解，或拉格朗日乘子法求解相应的对偶问题

1. **Import libraries and read csv file into a pandas dataframe or use built in dataset**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

customize our own data frame from the dataset

```
df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
```

```
df_target = pd.DataFrame(cancer['target'],columns=['Cancer'])
```

2. **split the data into a training set and a test set**

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(df_feat, np.ravel(df_target),
                                                    test_size=0.30, random_state=101)
```

np.ravel(a, order=char): Return a contiguous flattened array with specific order

3. **train the support vector classifier, predict and evaluate**

```
from sklearn.svm import SVC
```

```
model = SVC()
```

```
model.fit(X_train,y_train)
```

```
predictions = model.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,predictions))
```

```
print(classification_report(y_test,predictions))
```

4. **use a GridSearch to adjust parameters if classifying does not perform well, creating a grid of parameters and just trying out all the possible combinations is called a Gridsearch. Use a built in GridSearchCV (CV stands for cross validation) which takes a dictionary that describes the parameters that should be tried and a model to train, the grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested. One of the great things about GridSearchCV is that it is a meta-estimator. It takes an estimator like SVC,**

and creates a new estimator, that behaves the same.

```
param_grid = {'C': [0.1,1, 10, 100, 1000],
              'gamma': [1,0.1,0.01,0.001,0.0001],
              'kernel': ['rbf']}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
```

GridSearchCV(estimator, param_grid, refit=True, verbose=int (the higher, the more messages), …): Exhaustive search over specified parameter values for an estimator and refit an estimator using the best found parameters on the whole dataset

inspect the best parameters in the best_params_ attribute and the best estimator in the best_eatimator_ attribute, predict again and evaluate

```
grid.best_params_
```

```
grid.best_estimator_
```

```
grid_predictions = grid.predict(X_test)
```

```
print(confusion_matrix(y_test,grid_predictions))
```

```
print(classification_report(y_test,grid_predictions))
```

TENSORFLOW BASICS

1. **Import tensorflow library**

```
import tensorflow as tf
```

2. **create a simple constant with Tensorflow**

```
hello = tf.constant('Hello World')
```

```
type(hello)
```

```
x = tf.constant(100)
```

```
type(x)
```

3. **create a TensorFlow Session, which is a class for running TensorFlow operations. A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated**

```
sess = tf.Session()
```

```
sess.run(hello)
```

→ b'Hello World'

```
type(sess.run(hello))
```

→ bytes

```
with tf.Session() as sess:
    print('Operations with Constants')
    print('Addition',sess.run(x+y))
    print('Subtraction',sess.run(x-y))
    print('Multiplication',sess.run(x*y))
    print('Division',sess.run(x/y))
```

4. **use a placeholder for a tensor to wait for a constant to appear after a cycle of operations, the value of this tensor must be fed using the feed_dict optional argument**

```
x = tf.placeholder(tf.int32)
y = tf.placeholder(tf.int32)
```

define customized operations

```
add = tf.add(x,y)
sub = tf.sub(x,y)
mul = tf.mul(x,y)
```

run operations with variable input

```
d = {x:20,y:30}
```

```
with tf.Session() as sess:
    print('Operations with Constants')
    print('Addition',sess.run(add,feed_dict=d))
    print('Subtraction',sess.run(sub,feed_dict=d))
    print('Multiplication',sess.run(mul,feed_dict=d))
```

complex operations using matrix multiplication

```
import numpy as np
a = np.array([[5.0,5.0]])
b = np.array([[2.0],[2.0]])
```

```
mat1 = tf.constant(a)
```

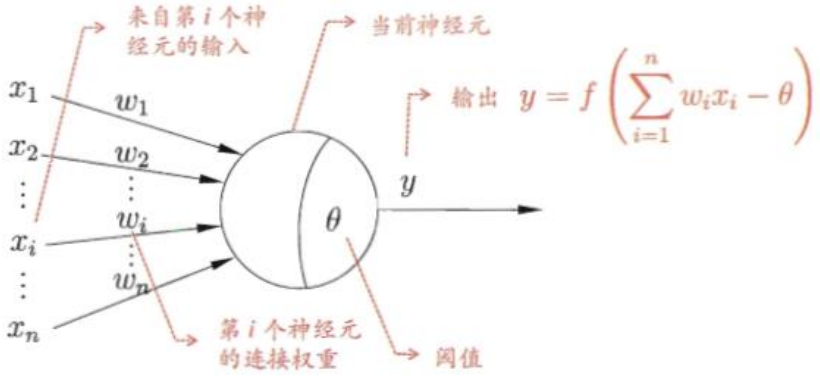
```
mat2 = tf.constant(b)
```

```
matrix_multi = tf.matmul(mat1,mat2)
```

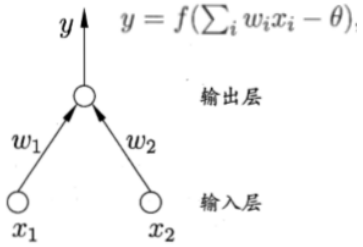
NEURAL NETS AND DEEP LEARNING

neural networks (神经网络) : 是由具有适应性的简单单元组成的广泛并行互连的网络，最基本的成分是神经元 (neuron) 模型，常用 M-P 神经元模型，其中神经元接收到

来自 n 个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过激活函数处理以产生神经元的输出



感知机 (perceptron) : 也称为阈值逻辑单元 (threshold logic unit) 由两层神经元组成，输入层接收外界输入信号后传递给输出层，输出层是 M-P 神经元，能够实现与、或、非逻辑运算



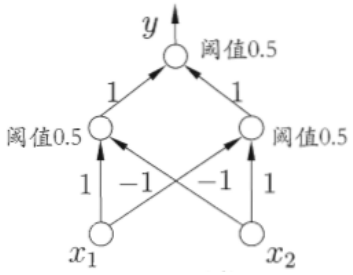
“与” $(x_1 \wedge x_2)$: 令 $w_1=w_2=1, \theta=2$, 则 $y=f(1 \cdot x_1+1 \cdot x_2-2)$, 仅 在 $x_1=x_2=1$ 时, $y=1$;

“或” $(x_1 \vee x_2)$: 令 $w_1=w_2=1, \theta=0.5$, 则 $y=f(1 \cdot x_1+1 \cdot x_2-0.5)$, 当 $x_1=1$ 或 $x_2=1$ 时, $y=1$;

“非” $(\neg x_1)$: 令 $w_1=-0.6, w_2=0, \theta=-0.5$, 则 $y=f(-0.6 \cdot x_1+0 \cdot x_2+0.5)$, 当 $x_1=1$ 时, $y=0$; 当 $x_1=0$ 时, $y=1$.

感知机只有输出层神经元进行激活函数处理，即只有一层功能神经元，学习能力有限，只能解决线性可分的问题 (若两类模式是线性可分的，即存在一个线性超平面能将它们分开)

要解决非线性可分问题 (例如异或)，使用多层功能神经元，输出层和输入层之间的神经元，称为隐含层，隐含层和输出层神经元都是有激活函数的功能神经元



常见的神经网络中，每层神经元与下一层神经元全互连，神经元之间不存在同层连接，也不存在跨层连接，称为多层前馈神经网络 (multi-layer feedforward neural networks)，输入层神经元接收外界输入，隐含层和输出层神经元对信号进行函数处理，最终结果由输出层神经元输出

误差逆传播算法 (error back propagation) 可以用来训练多层网络

输入: 训练集 $D = \{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{m}$;
学习率 η.

过程:

在$(0,1)$范围内随机初始化网络中所有连接权和阈值
repeat
  for all $(\boldsymbol{x}_k, \boldsymbol{y}_k) \in D$ **do**
    根据当前参数和式(5.3) 计算当前样本的输出 $\hat{\boldsymbol{y}}_k$;
    根据式(5.10) 计算输出层神经元的梯度项 g_j;
    根据式(5.15) 计算隐层神经元的梯度项 e_h;
    根据式(5.11)-(5.14) 更新连接权 w_{hj}, v_{ih} 与阈值 θ_j, γ_h
  end for
until 达到停止条件

输出: 连接权与阈值确定的多层前馈神经网络

$$\hat{y}_j^k = f(\beta_j - \theta_j) \quad \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$
$$= \hat{y}_j^k(1-\hat{y}_j^k)(y_j^k - \hat{y}_j^k)$$
$$= b_h(1-b_h)\sum_{j=1}^{l} w_{hj}g_j$$
$$\Delta w_{hj} = \eta g_j b_h$$
$$\Delta v_{ih} = \eta e_h x_i$$
$$\Delta\theta_j = -\eta g_j$$
$$\Delta\gamma_h = -\eta e_h$$

BP 算法的目标是要最小化训练集 D 上的累计误差

由于表示能力强大，BP 训练神经网络可能导致过拟合，训练误差持续降低时，测试误差可能上升，常用早停策略 (early stopping，即将数据分成训练集和验证集，训练集用来计算梯度、更新连接权和阈值，验证集用来估计误差，若训练集误差降低但验证集误差升高，则停止训练) 或正则化策略 (regularization，即在误差函数中增加一个用于描述网络复杂度的部分，例如增加连接权和阈值平方和这一项后，训练过程将偏好比较小的连接权和阈值，使输出更加光滑，缓解了过拟合)

深度学习: 典型的深度学习模型是多隐层神经网络，难以直接使用 BP 算法进行训练，因为误差在多隐层内逆传播时，往往会发散而不是收敛到稳定状态

通过多层处理，逐渐将初始的低层特征表示 (与输出目标之间联系不太密切的输入表示) 转化为高层特征表示后，用简单模型即可完成复杂的分类等学习任务

无监督逐层训练 (unsupervised layer-wise training) 是多隐层网络训练的常用手段，基本思想是每次训练一层隐结点，训练时将上一层隐结点的输出作为输入，本层隐结点的输出作为下一层隐结点的输入，这称为预训练 (pre-training)； 预训练全部完成后，再对整个网络进行微调训练

预训练+微调训练的方法可视为将大量参数分组，对每组先找到局部较优的设置，再基于这些局部较优的结果联合起来进行全局寻优，这样能利用模型大量参数所提供的自由度并且有效地节省了训练开销

权共享 (weight sharing) 是另一种节省训练开销的方法，即让一组神经元使用相同的连接权