分类算法

主讲老师：袁 方

OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. Logistic Regression
6. Naive Bayes
7. Decision Tree
8. 分类器选型

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN

T=True，预测正确，真的； F=False，预测错误，假的

预备知识 - Metric

混淆矩阵 Confusion Matrix

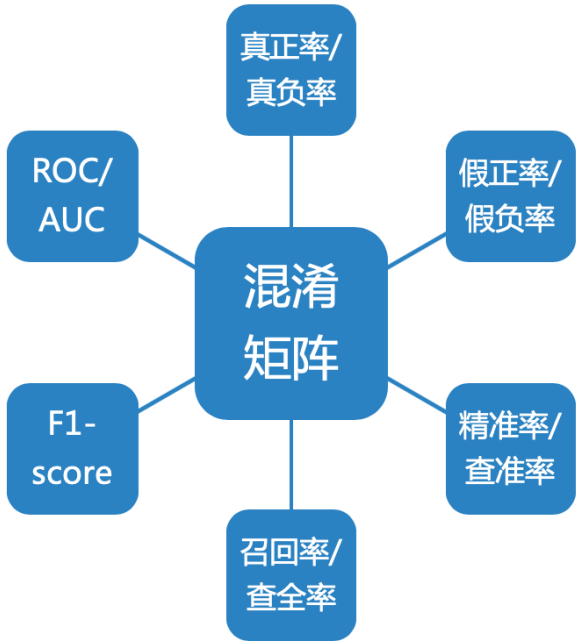
	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN

真实类别	预测类别	预测与真实的评价
1（正类）	1	TP
1	0	FN
0（负类）	1	FP
0	0	TN

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



真正率 (True Positive Rate, TPR)

灵敏度（sensitivity）

$$TPR = TP / (TP + FN)$$

真负率 (True Negative Rate, TNR)

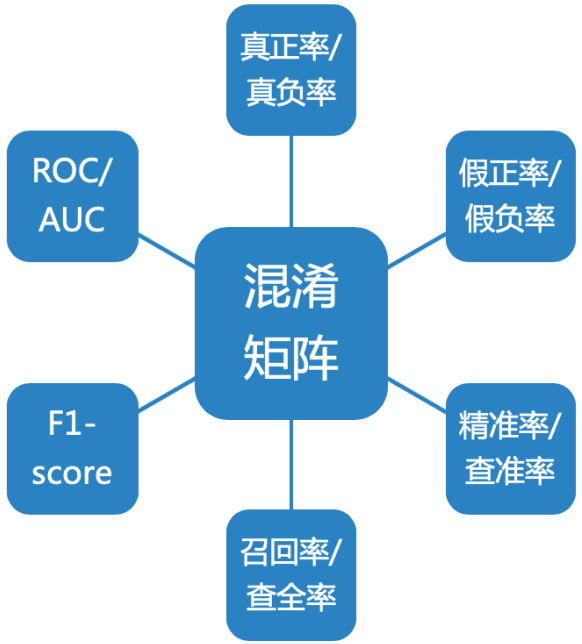
特指度（specificity）

$$TNR = TN / (TN + FP)$$

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



假正率 （False Positive Rate, FPR）

$$FPR = FP / (FP + TN)$$

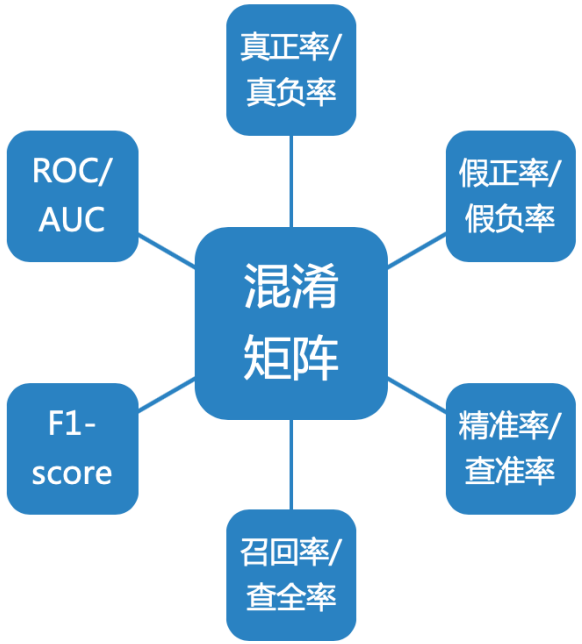
假负率（ False Negative Rate , FNR）

$$FNR = FN / (TP + FN)$$

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



精准率/查准率 (Precision)

$$P = TP/(TP + FP)$$

召回率/查全率 (Recall)，真正率 TPR

$$R = TP/(TP + FN)$$

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN

真实类别	预测类别	预测与真实的评价
1	1	TP
1	1	TP
1	1	TP
1	0	FN
0	0	TN
0	1	FP
0	1	FP
0	0	TN

精准率/查准率 (Precision)

$$\mathrm{P} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FP}) = \frac{3}{3+2} = \frac{3}{5}$$

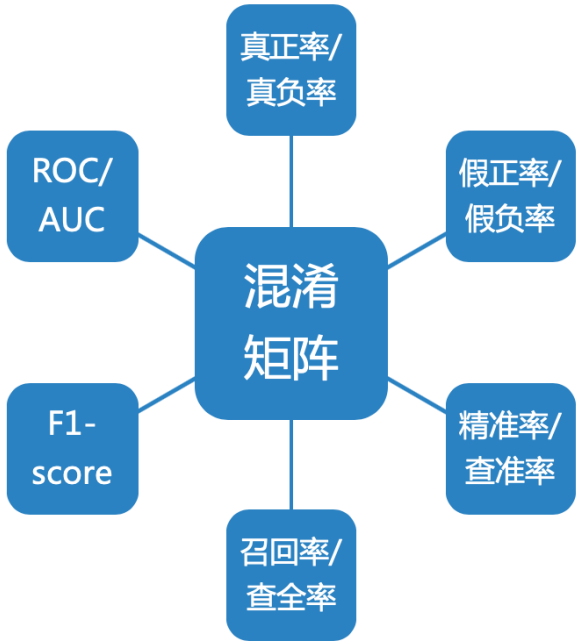
召回率/查全率 (Recall)

$$\mathrm{R} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN}) = \frac{3}{3+1} = \frac{3}{4}$$

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



F1-score：

精确率和召回率的调和平均数

$$\frac{1}{\mathrm{F}_1} = \left(\frac{1}{\mathrm{P}} + \frac{1}{\mathrm{R}}\right)/2$$

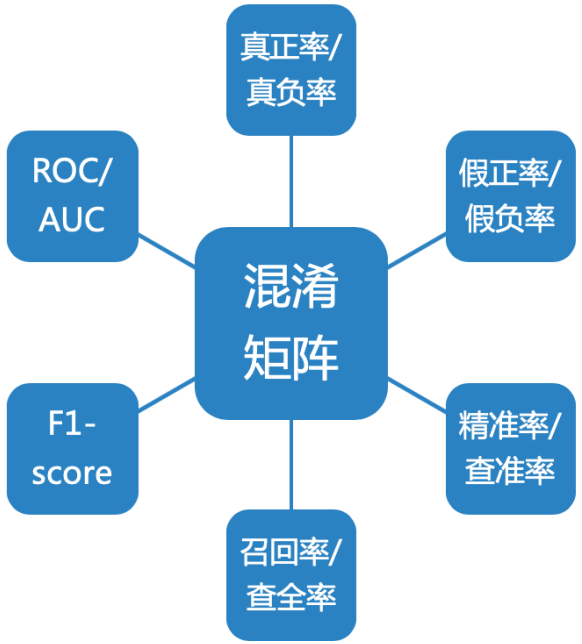
$$\mathrm{P} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$$
$$\mathrm{R} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$$

$$\mathrm{F}_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



Receiver Operating Characteristic Curve (ROC)

二分类模型返回一个概率值，通过调整**阈值**，即大于该阈值为正类，反之负类，可以得到多个(FPR,TPR)点，描**点画图得到的曲线即为**ROC曲线

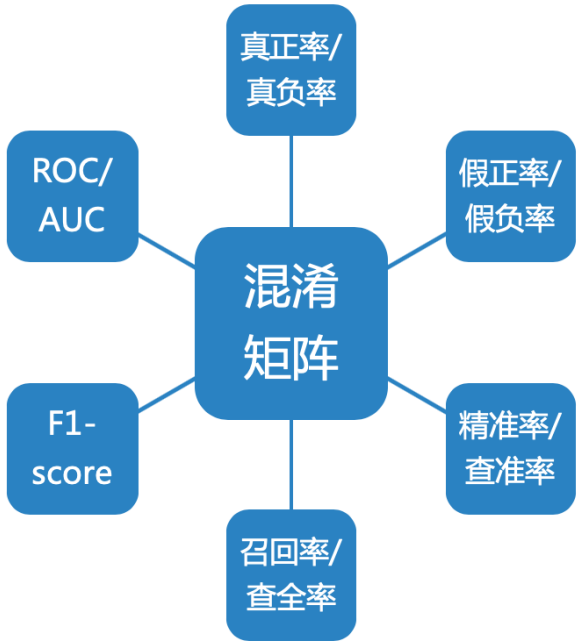
$$FPR = FP / (FP + TN)$$

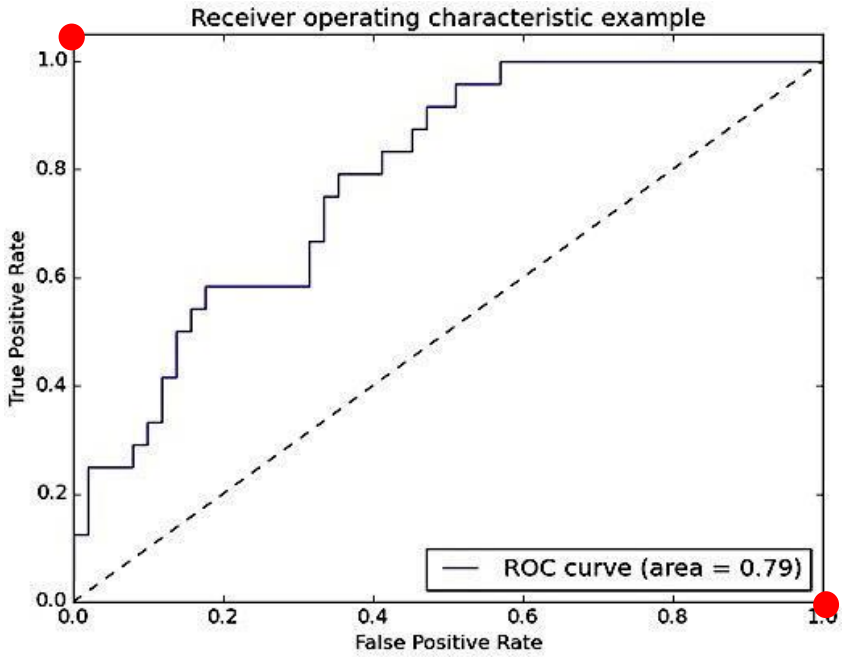
$$TPR = TP / (TP + FN)$$

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



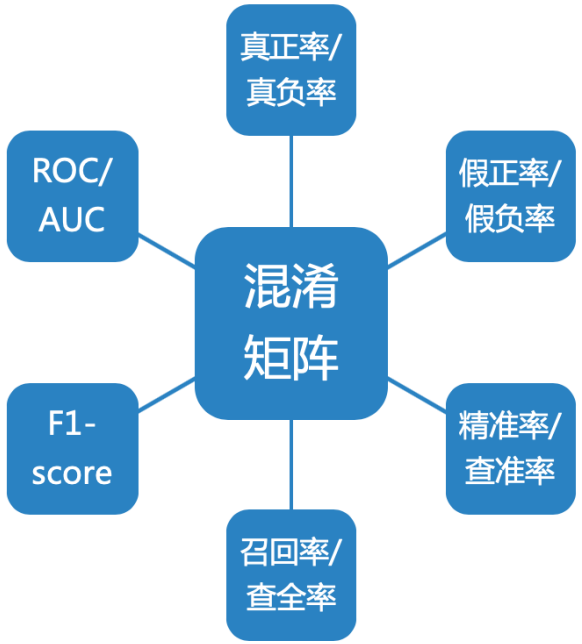


TPR=1, FPR=0 理想；TPR=0, FPR=1 恶梦

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN



Area Under Curve-(AUC)

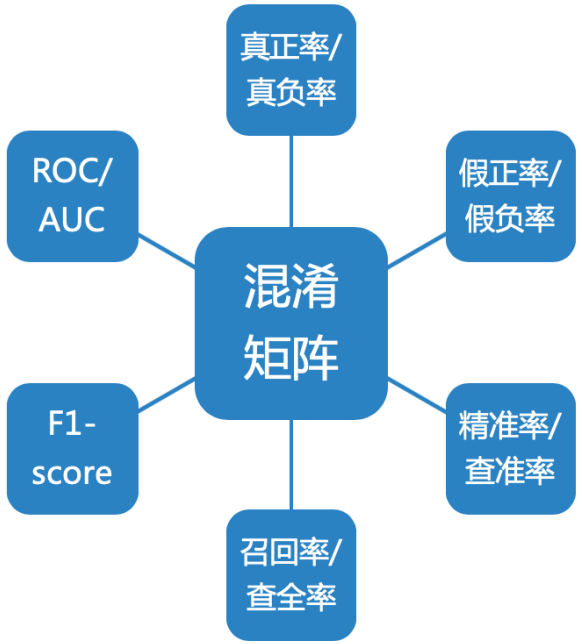
ROC曲线下面积，一般取值范围[0.5,1]

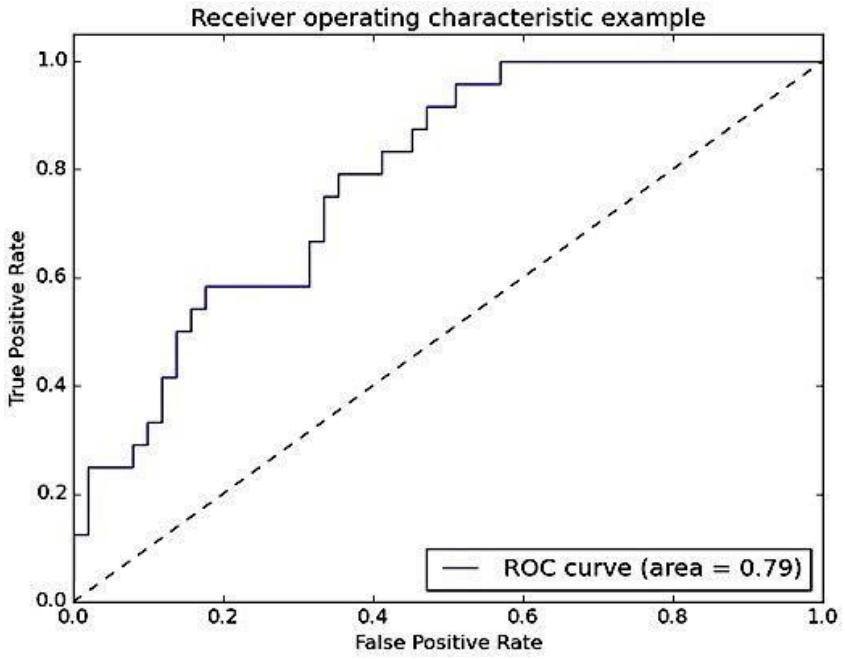
AUC越大，分类模型效果越好

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN





AUC: 0.79

预备知识 - Metric

混淆矩阵 Confusion Matrix

	预测为正类 (P = Positive，正)	预测为负类 (N = Negative，负)
实际为正类	TP	FN
实际为负类	FP	TN

Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score (y_true, y_pred[, …])	Accuracy classification score.
metrics.auc (x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.average_precision_score (y_true, y_score)	Compute average precision (AP) from prediction scores
metrics.brier_score_loss (y_true, y_prob[, …])	Compute the Brier score.
metrics.classification_report (y_true, y_pred)	Build a text report showing the main classification metrics
metrics.cohen_kappa_score (y1, y2[, labels, …])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix (y_true, y_pred[, …])	Compute confusion matrix to evaluate the accuracy of a classification
metrics.f1_score (y_true, y_pred[, labels, …])	Compute the F1 score, also known as balanced F-score or F-measure
metrics.fbeta_score (y_true, y_pred, beta[, …])	Compute the F-beta score
metrics.hamming_loss (y_true, y_pred[, …])	Compute the average Hamming loss.
metrics.hinge_loss (y_true, pred_decision[, …])	Average hinge loss (non-regularized)
metrics.jaccard_similarity_score (y_true, y_pred)	Jaccard similarity coefficient score
metrics.log_loss (y_true, y_pred[, eps, …])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef (y_true, y_pred[, …])	Compute the Matthews correlation coefficient (MCC)
metrics.precision_recall_curve (y_true, …)	Compute precision-recall pairs for different probability thresholds
metrics.precision_recall_fscore_support (…)	Compute precision, recall, F-measure and support for each class
metrics.precision_score (y_true, y_pred[, …])	Compute the precision
metrics.recall_score (y_true, y_pred[, …])	Compute the recall
metrics.roc_auc_score (y_true, y_score[, …])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve (y_true, y_score[, …])	Compute Receiver operating characteristic (ROC)
metrics.zero_one_loss (y_true, y_pred[, …])	Zero-one classification loss.

OUTLINES

1. 预备知识
2. **分类问题定义-Classification**
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. Logistic Regression
6. Naive Bayes
7. Decision Tree
8. 分类器选型

分类问题定义-Classification

Male or Female

Bull or Bear

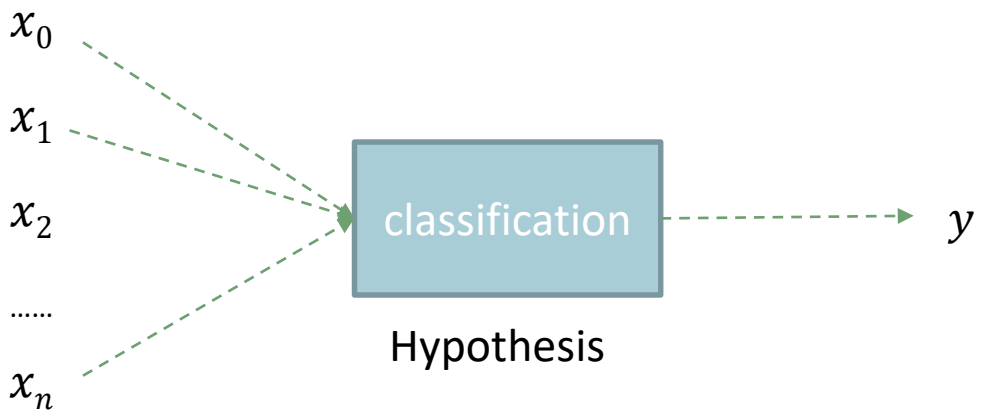
Business/Political/Sport

目标：两个或多个离散的类别

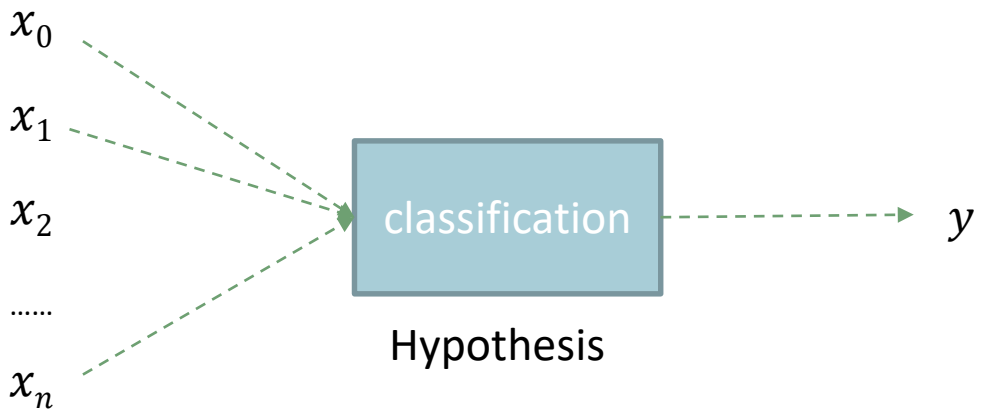
分类问题定义-Classification

Male or Female

目标：两个或多个离散的类别



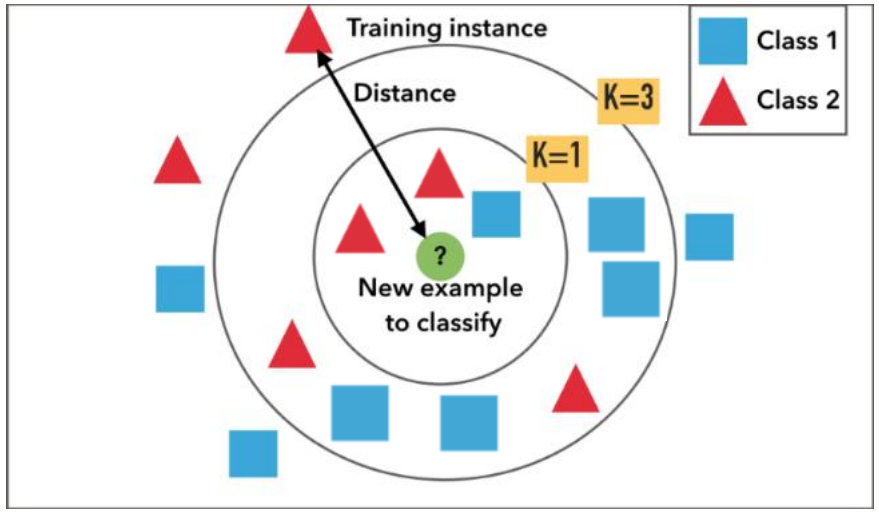
分类问题定义-Classification



OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. **K-Nearest Neighbor (KNN)**
4. Support Vector Meachine(SVM)
5. Logistic Regression
6. Naive Bayes
7. Decision Tree
8. 分类器选型

Classifier-KNN



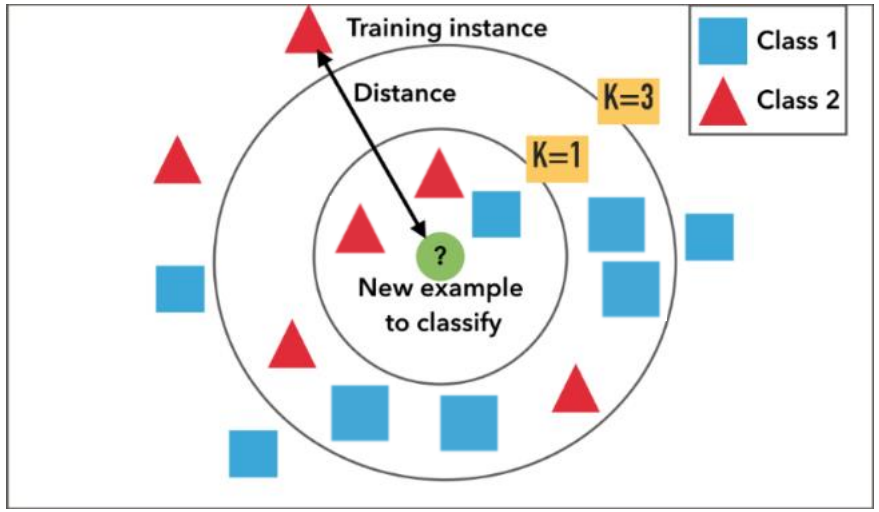
K最近邻 (KNN，K-Nearest Neighbor)

数据挖掘分类技术中最简单的方法之一

所谓K最近邻，就是**K个最近的邻居**

每个样本都可以用它最接近的**K个邻居**来代表

Classifier-KNN



Distance?

Euclidean Distance

欧氏距离是**最易于理解**的一种距离计算方法，源自**欧氏空间**中两点间的距离公式。

二维平面上两点a (x1,y1) 与 b (x2,y2) 间的欧氏距离?

Classifier-KNN

Euclidean Distance

欧氏距离是**最易于理解**的一种距离计算方法，源自**欧氏空间**中两点间的距离公式。

二维平面上两点a (x1,y1) 与 b (x2,y2) 间的欧氏距离?

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

三维空间上两点a (x1,y1,z1) 与 b (x2,y2,z2) 间的欧氏距离?

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

n维空间上两点a (x11,x12,...x1n) 与 b (y21,y22,...,y2n) 间的欧氏距离?

$$d = \sqrt{\sum_{k=1}^{n} (x_{1k} - y_{2k})^2}$$

Classifier-KNN

Manhattan Distance

城市**街区距离**(City Block distance)

想象你在曼哈顿要从一个十字路口开车到另外一个十字路口，驾驶距离是两点间的直线距离吗？显然不是，除非你能穿越大楼。实际驾驶距离就是这个“曼哈顿距离”。

二维平面上两点a (x1,y1) 与 b (x2,y2) 间的Manhattan Distance?

$$d = |x_1 - x_2| + |y_1 - y_2|$$

n维空间上两点a (x11,x12,...x1n) 与 b (y21,y22,...,y2n) 间的Manhattan Distance?

$$d = \sum_{k=1}^{n} |x_{1k} - y_{2k}|$$

Classifier-KNN

Minkowski Distance 闵可夫斯基

闵氏距离不是一种距离，而是**一组距离**的定义

n维空间上两点a (x11,x12,...x1n) 与 b (y21,y22,...,y2n) 间的Minkowski Distance?

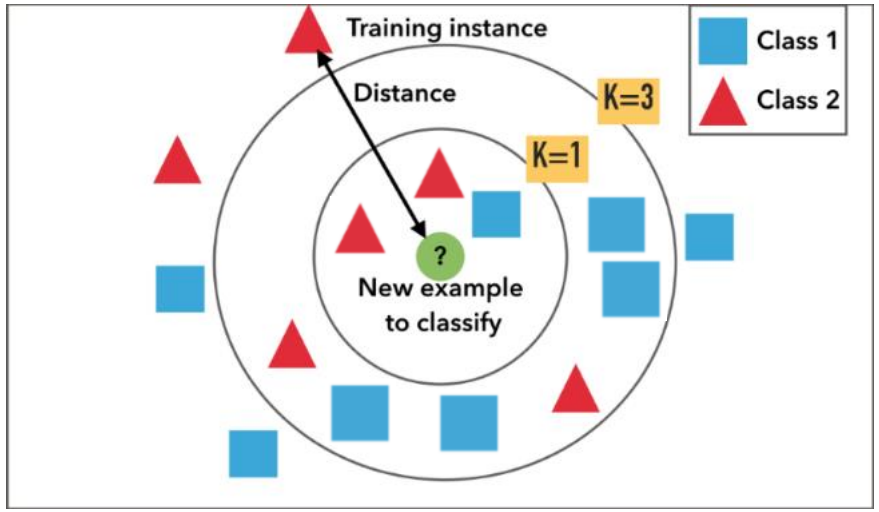
$$d = \sqrt[p]{\sum_{k=1}^{n} |x_{1k} - y_{2k}|^p}$$

Classifier-KNN

Other Distance?

How to calculate Distance in scikit-learn?

Classifier-KNN



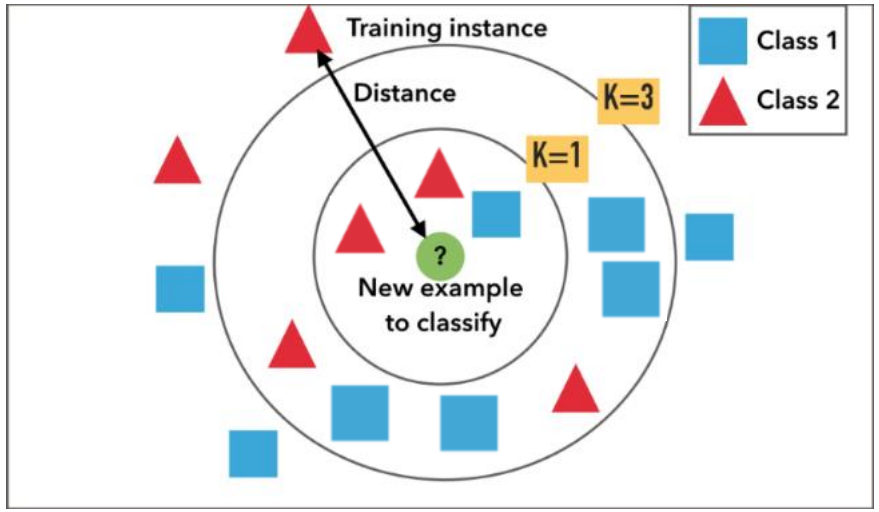
蓝方块和红三角均是已有分类数据，

判断绿色圆块是属于蓝方块或者红三角？

如果K=3：红三角占比2/3，则判断为红三角；

如果K=9：蓝方块占比5/9，则判断为蓝方块。

Classifier-KNN



算法步骤：

1. **计算**已知类别数据集中的点与当前之间的**距离**
2. 按照**距离递增**次序**排序**
3. **选取**与当前点**距离最小的k个点**
4. 确定前k个点所在的类别的**出现频率**
5. **返回**前k个点出现**频率最高**的类别作为当前点的预测分类

Classifier-KNN

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors. **KNeighborsClassifier** (*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs*) [source]

weights : str or callable, optional (default = 'uniform')

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Classifier-KNN

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors. **KNeighborsClassifier** (*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs*) [source]

algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

KDTree是依次对K维坐标轴，以中值切分构造的树，每一个节点是一个超矩形，在维数小于20时效率最高

BallTree是为了克服KD树高维失效而发明的，其构造过程是以质心C和半径r分割样本空间，每一个节点是一个超球体

Classifier-KNN

In [45]:
```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
```

In [46]:
```
np.random.seed(0)
```

In [47]:
```
indices = np.random.permutation(len(X)) # shuffle of index
```

In [48]:
```
indices
```

Out[48]:
```
array([114,  62,  33, 107,   7, 100,  40,  86,  76,  71, 134,  51,  73,
        54,  63,  37,  78,  90,  45,  16, 121,  66,  24,   8, 126,  22,
        44,  97,  93,  26, 137,  84,  27, 127, 132,  59,  18,  83,  61,
        92, 112,   2, 141,  43,  10,  60, 116, 144, 119, 108,  69, 135,
        56,  80, 123, 133, 106, 146,  50, 147,  85,  30, 101,  94,  64,
        89,  91, 125,  48,  13, 111,  95,  20,  15,  52,   3, 149,  98,
         6,  68, 109,  96,  12, 102, 120, 104, 128,  46,  11, 110, 124,
        41, 148,   1, 113, 139,  42,   4, 129,  17,  38,   5,  53, 143,
       105,   0,  34,  28,  55,  75,  35,  23,  74,  31, 118,  57, 131,
        65,  32, 138,  14, 122,  19,  29, 130,  49, 136,  99,  82,  79,
       115, 145,  72,  77,  25,  81, 140, 142,  39,  58,  88,  70,  87,
        36,  21,   9, 103,  67, 117,  47])
```

Classifier-KNN

In [49]:
```
X_train = X[indices[:-10]] # 前140
y_train = y[indices[:-10]]
X_test = X[indices[-10:]]
y_test = y[indices[-10:]]
```

In [50]:
```
knn = KNeighborsClassifier()
```

In [51]:
```
knn.fit(X_train, y_train)
```

Out[51]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [52]:
```
y_pred = knn.predict(X_test)
```

In [53]:
```
y_pred
```

Out[53]:
```
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
```

In [54]:
```
y_test
```

Out[54]:
```
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

In [55]:
```
knn.score(X_test, y_test)
```

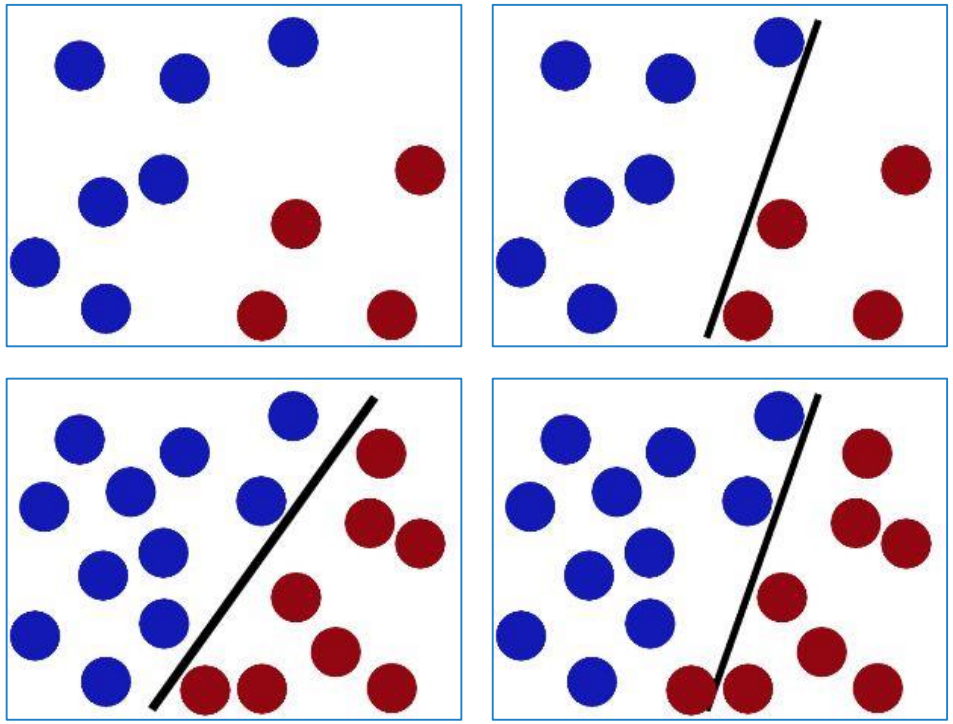
Out[55]:
```
0.9
```

OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. **Support Vector Machine(SVM)**
5. Logistic Regression
6. Naive Bayes
7. Decision Tree
8. 分类器选型

Classifier-SVM

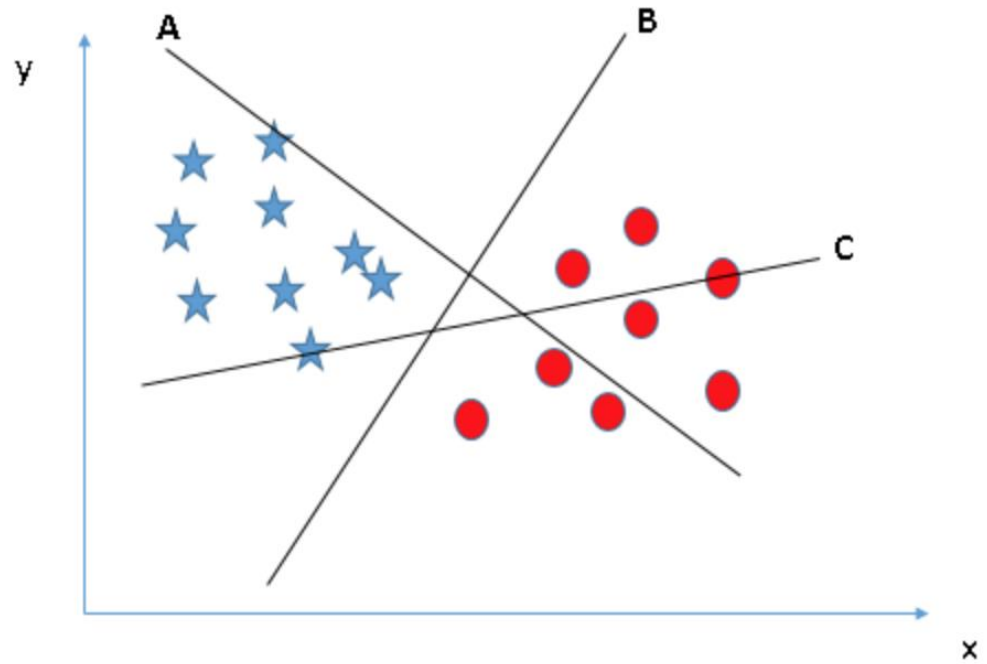
支持向量机（Support Vector Machine, SVM）的基本模型是在**特征空间上**找到**最佳的分离超平面**使得训练集上**正负样本间隔最大。**



样本越来越多，能不能继续分下去？

Classifier-SVM

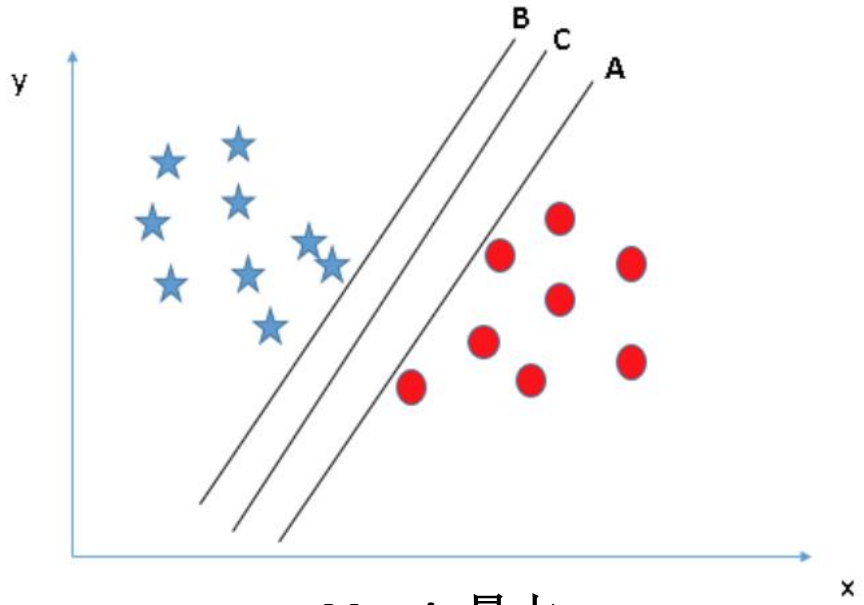
支持向量机（Support Vector Machine, SVM）的基本模型是在**特征空间上**找到**最佳的分离超平面**使得训练集上**正负样本间隔最大。**



怎么分最好？

Classifier-SVM

支持向量机（Support Vector Machine, SVM）的基本模型是在**特征空间上**找到**最佳的分离超平面**使得训练集上**正负样本间隔最大。**



Margin最大

Classifier-SVM

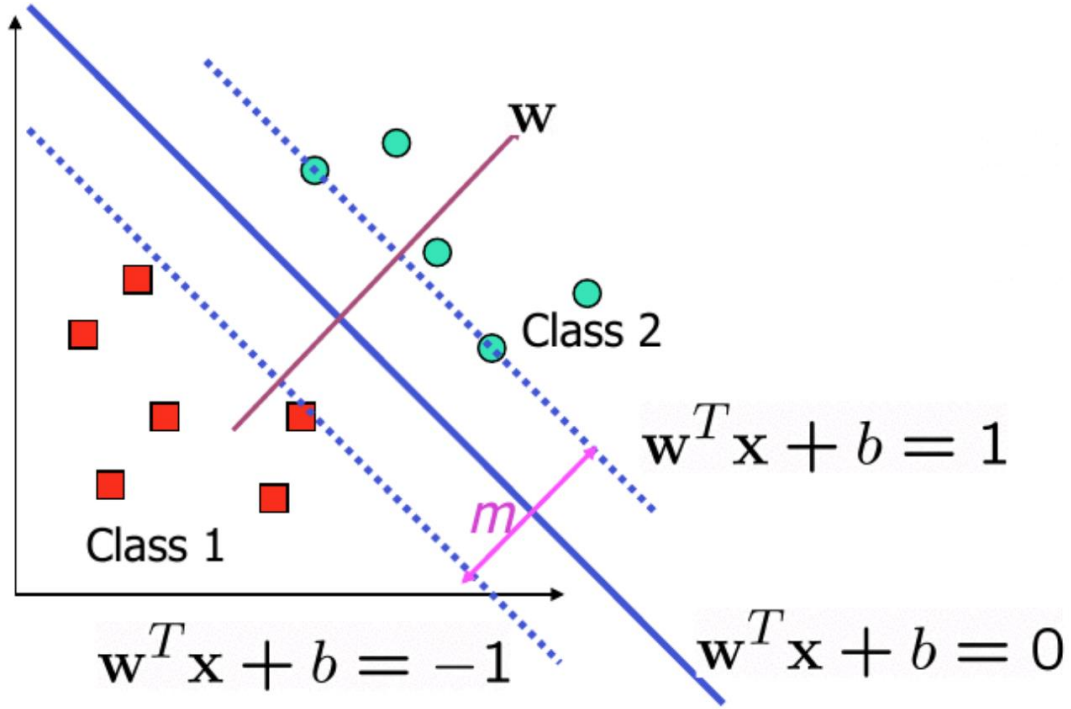
给定训练样本集$D = \{(x_1, y_1), (x_2, y_2) \cdots (x_n, y_n)\},\ y \in \{+1, -1\}$

其中，分类学习最基本的想法就是基于训练集D在特征空间中找到一个**最佳划分超平面**将正负样本分开

超平面的线性表示：$\vec{w}^T\vec{x} + b = 0$

其中$\vec{w}$为法向量，决定超平面的方向；b表示偏移量，决定超平面与原点之间的距离。

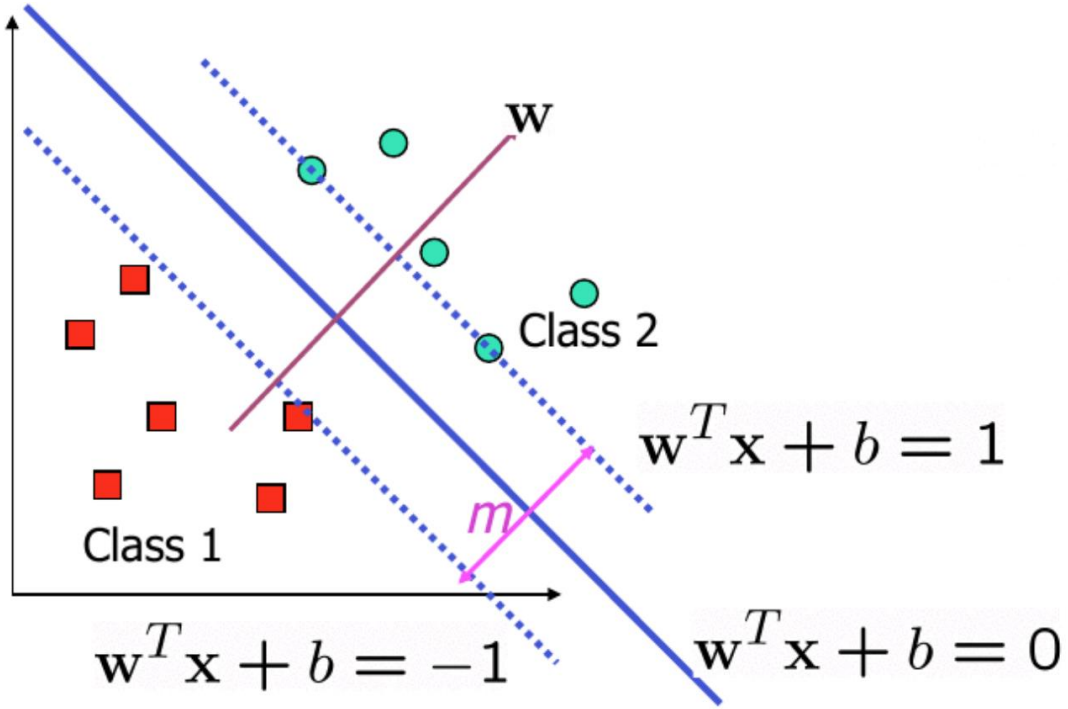
Classifier-SVM



分割超平面的**两侧**定义两个超平面H1和H2，这两个超平面分别**通过正样本**和**负样本**中**离分割超平面最近**的样本点。超平面H1和H2离分割超平面是**等距**的。

超平面H1和H2上面的点叫做**支持向量**，正负样本的间隔定义为超平面H1和H2之间的间隔，它是分割超平面**距最近正样本点距离和最近负样本点距离之和**。

Classifier-SVM



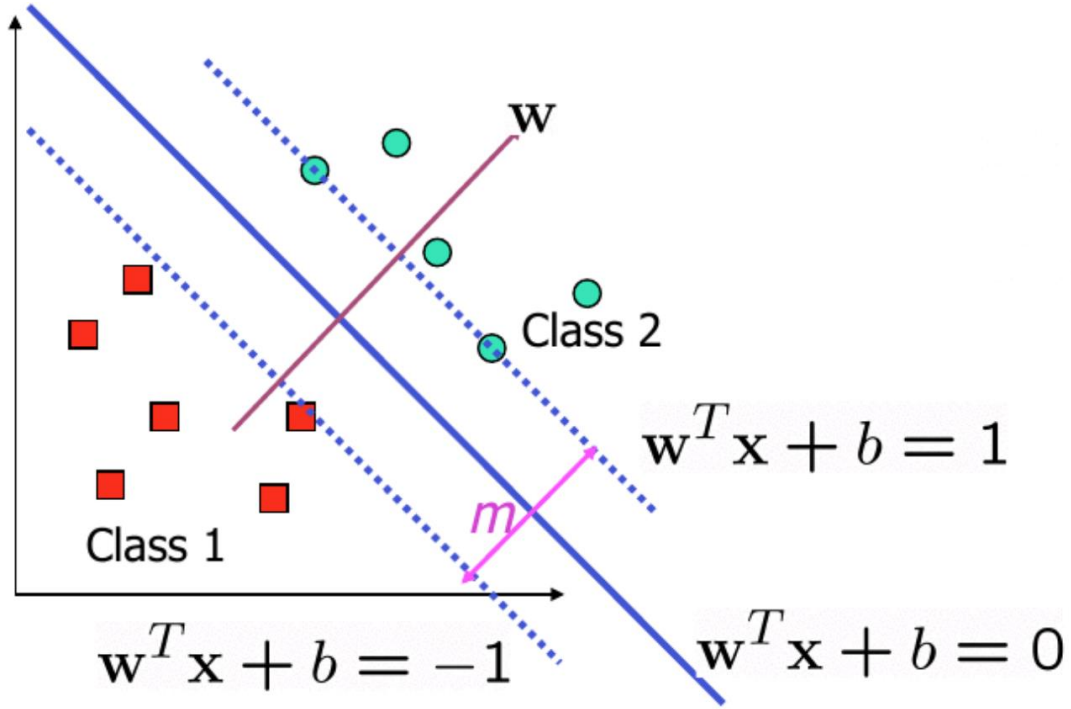
$\vec{\mathrm{w}}^{\mathrm{T}}\vec{\mathrm{x}}+\mathrm{b}\geq 1, y_i=+1$

$\vec{\mathrm{w}}^{\mathrm{T}}\vec{\mathrm{x}}+\mathrm{b}<-1, y_i=-1$

等价于：$y_i\left(\vec{\mathrm{w}}^{\mathrm{T}}\vec{\mathrm{x}}+\mathrm{b}\right)\geq+1$

目标：Margin最大，Margin?

Classifier-SVM

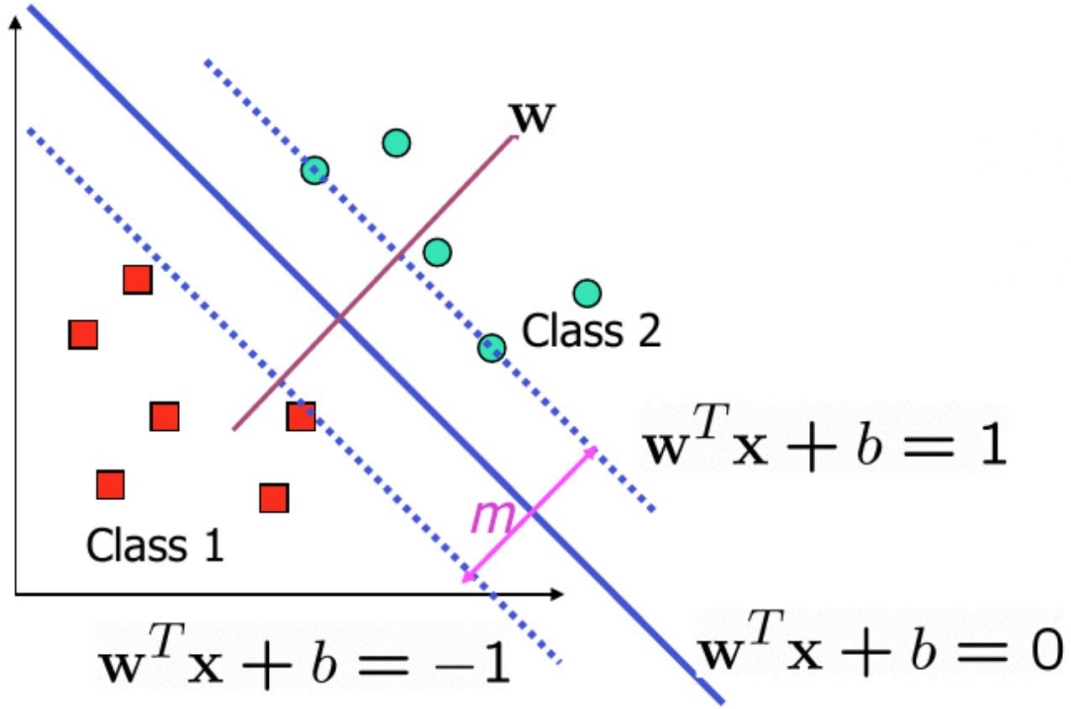


空间任意一点到超平面的距离： $\mathrm{r}=\frac{\vec{\mathrm{w}}^{\mathrm{T}}\vec{\mathrm{x}}+\mathrm{b}}{\|\mathrm{w}\|}$

支持向量： $\vec{\mathrm{w}}^{\mathrm{T}}\vec{\mathrm{x}}+\mathrm{b}=+1$上的点为$\mathrm{x}_+$; $\vec{\mathrm{w}}^{\mathrm{T}}\vec{\mathrm{x}}+\mathrm{b}=-1$上的点为$\mathrm{x}_-$

Margin: $\gamma=(\mathrm{x}_+-\mathrm{x}_-)\frac{\vec{\mathrm{w}}^{\mathrm{T}}}{\|\mathrm{w}\|}=\frac{\vec{\mathrm{w}}^{\mathrm{T}}\mathrm{x}_+}{\|\mathrm{w}\|}-\frac{\vec{\mathrm{w}}^{\mathrm{T}}\mathrm{x}_-}{\|\mathrm{w}\|}=\frac{1-\mathrm{b}}{\|\mathrm{w}\|}-\frac{-1-\mathrm{b}}{\|\mathrm{w}\|}=\frac{2}{\|\mathrm{w}\|}$

Classifier-SVM



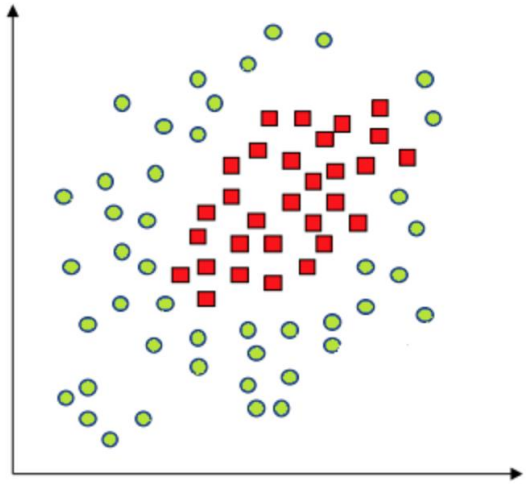
目标：$\max_{(\vec{w},b)}(\gamma)=\max(\frac{2}{\|w\|})$

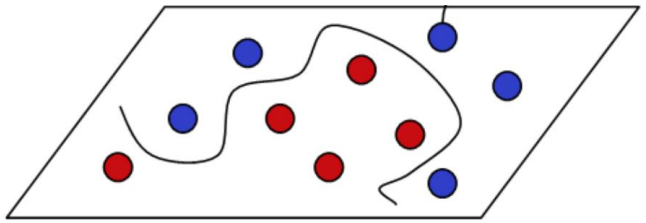
约束：$y_i(\vec{w}^T\vec{x}+b)\geq+1$

优化：拉格朗日数乘法

Classifier-SVM

SVM, **线性**模型（分类**超平面**）

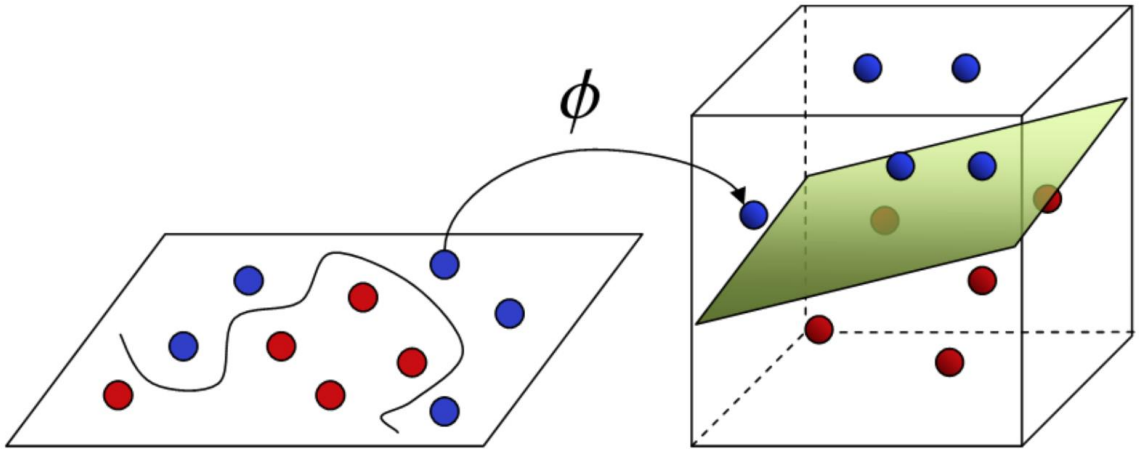


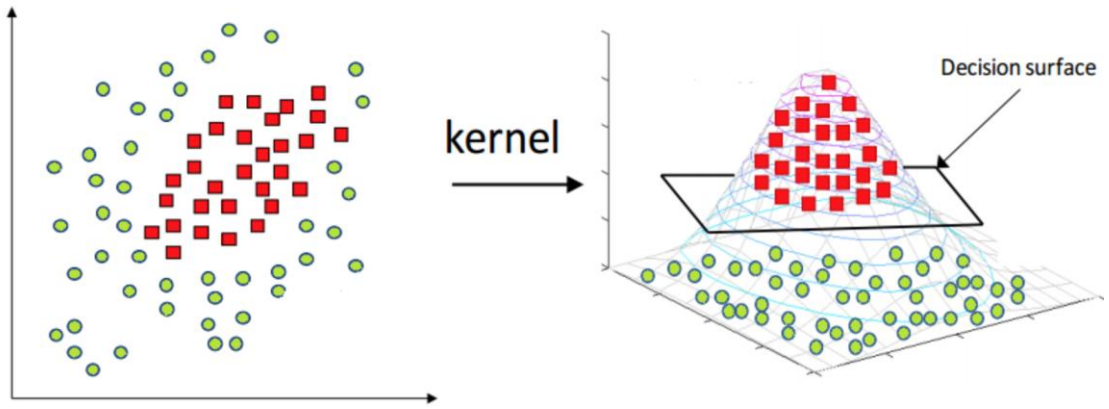


线性可分？

Classifier-SVM

SVM, 线性模型（分类面为超平面），kernel





Classifier-SVM

SVM, **线性**构型（分类面为**超平面**），kernel

$$\kappa(x, xi) = x \cdot xi$$

$$\kappa(x, xi) = ((x \cdot xi) + 1)^d$$

Radial Basis Function（RBF），径向基函数

$$\kappa(x, xi) = \exp(-||x - xi||2/\delta^2)$$

$$\kappa(x, xi) = \tanh(\eta < x, xi > +\theta)$$

Classifier-SVM

Tips:

如果**特征的数量**大到和**样本数量差不多**，则选用LR或者线性核的SVM；

如果**特征的数量小**，样本的数量正常，则选用SVM+高斯核函数；

Classifier-SVM

sklearn.svm: Support Vector Machines

The sklearn.svm module includes Support Vector Machine algorithms.

User guide: See the Support Vector Machines section for further details.

Estimators

svm.LinearSVC([penalty, loss, dual, tol, C, ...])	Linear Support Vector Classification.
svm.LinearSVR([epsilon, tol, C, loss, ...])	Linear Support Vector Regression.
svm.NuSVC([nu, kernel, degree, gamma, ...])	Nu-Support Vector Classification.
svm.NuSVR([nu, C, kernel, degree, gamma, ...])	Nu Support Vector Regression.
svm.OneClassSVM([kernel, degree, gamma, ...])	Unsupervised Outlier Detection.
svm.SVC([C, kernel, degree, gamma, coef0, ...])	C-Support Vector Classification.
svm.SVR([kernel, degree, gamma, coef0, tol, ...])	Epsilon-Support Vector Regression.
svm.l1_min_c(X, y[, loss, fit_intercept, ...])	Return the lowest bound for C such that for C in (l1_min_C, infinity) the model is guaranteed not to be empty.

Classifier-SVM

`SVC` and `NuSVC` are similar methods, but accept slightly different sets of parameters and have different mathematical formulations (see section Mathematical formulation). On the other hand, `LinearSVC` is another implementation of Support Vector Classification for the case of a linear kernel. Note that `LinearSVC` does not accept keyword `kernel`, as this is assumed to be linear. It also lacks some of the members of `SVC` and `NuSVC`, like `support_`.

sklearn.svm.SVC

class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

kernel : string, optional (default='rbf')

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

Classifier-SVM

In [17]:
```
from sklearn import svm
clf = svm.SVC()
clf.fit(iris_x_train, iris_y_train)  #调用该对象的训练方法，主要接收两个参数：训练数据集及其样本标签
```

Out[17]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [18]:
```
iris_y_predict = clf.predict(iris_x_test)  #调用该对象的测试方法，主要接收一个参数：测试数据集
iris_y_predict
```

Out[18]:
```
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
```

In [19]:
```
iris_y_test
```

Out[19]:
```
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

In [20]:
```
score=clf.score(iris_x_test,iris_y_test,sample_weight=None) #调用该对象的打分方法，计算出准确率
score
```

Out[20]:
```
0.90000000000000002
```

OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. **Logistic Regression**
6. Naive Bayes
7. Decision Tree
8. 分类器选型

Classifier-LR

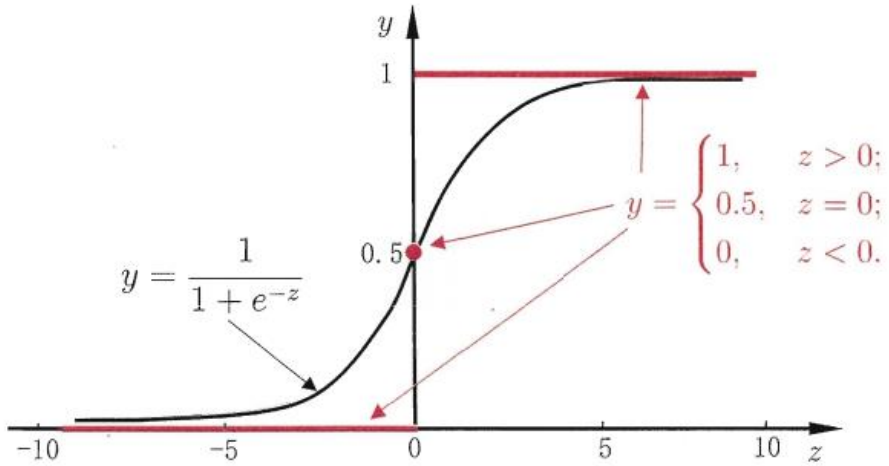
LR，Logistic Regression，**逻辑回归**

LR是一种广义线性回归（generalized linear model）

$$y = w^T x + b \longrightarrow y = \phi(z) = \phi(w^T x + b)$$

$$y \in (-\infty, +\infty) \longrightarrow y \in [0,1]$$

Classifier-LR



$$y = \begin{cases} 0 & if\ z < 0 \\ 0.5 & if\ z = 0 \\ 1 & if\ z > 0 \end{cases}$$

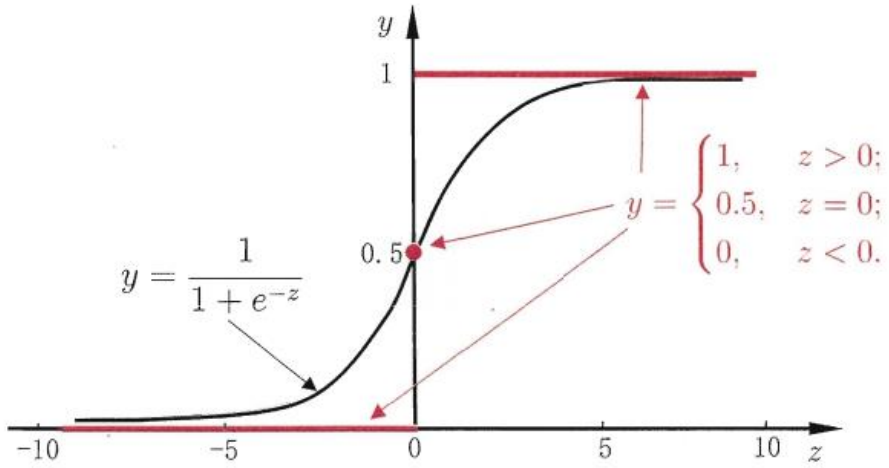
阶跃函数

函数**不连续**

$$y = \frac{1}{1 + e^{-z}}$$

Sigmoid函数**单调可微**

Classifier-LR



$$J(w) = \sum_i \frac{1}{2} (\phi(z^{(i)}) - y^{(i)})^2$$

其中，

$y^{(i)}$表示第i个样本的**真实值**，

$\phi(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$，表示第$i$个样本的**预测值**，

$z^{(i)} = w^T x^{(i)} + b$，$i$表示第$i$个样本点

Classifier-LR

常见损失函数

- **0-1**损失函数 (0-1 loss function)
- **绝对**损失函数 (absolute loss function)
- **平方**损失函数 (quadratic loss function)
- **对数**损失函数 (logarithmic loss function)

  /对数似然损失函数 (loglikelihood loss function)

Classifier-LR

常见损失函数

- 0-1损失函数 (0-1 loss function) **感知机**

$$L(Y, f(X)) = \begin{cases} 1, Y \neq f(X) \\ 0, Y = f(X) \end{cases} \quad L(Y, f(X)) = \begin{cases} 1, |Y - f(X)| \geq T \\ 0, |Y = f(X)| < T \end{cases}$$

- 绝对损失函数 (absolute loss function)

$$L(Y, f(X) = |Y - f(X)|$$

Classifier-LR

常见损失函数

- 平方损失函数 (quadratic loss function)

$$L(Y|f(X)) = \sum_{N}(Y - f(X))^2$$

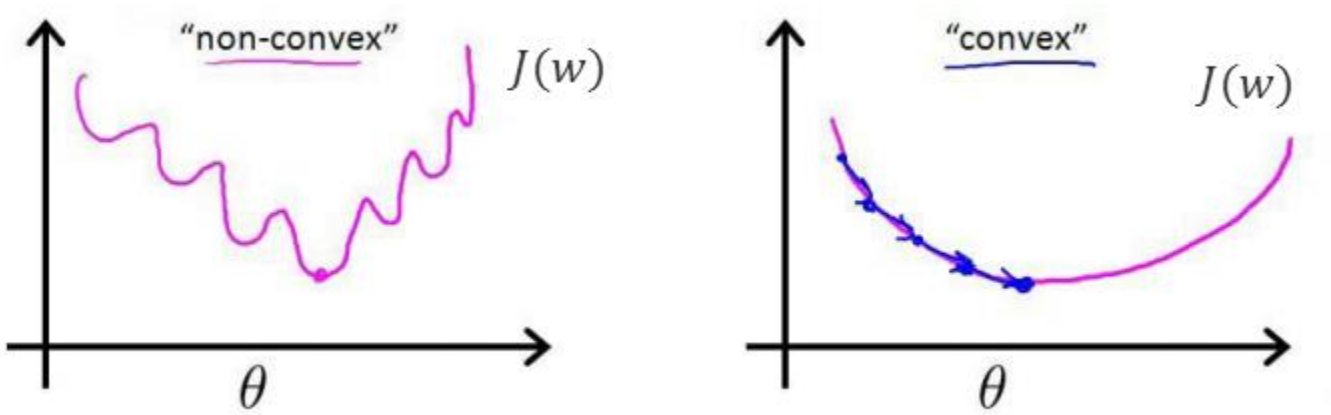
- 对数损失函数 (logarithmic loss function)

  /对数似然损失函数 (loglikelihood loss function)

$$L(Y, P(Y|X)) = -logP(Y|X)$$

Classifier-LR

$$J(w) = \sum_i \frac{1}{2} (\phi(z^{(i)}) - y^{(i)})^2$$



非凸函数
有许多局部最小值，不利于求解。

Classifier-LR

$\phi(z)$可以视为$y = 1$的后验估计

$$p(y = 1|x; w) = \phi(w^T x + b) = \phi(z)$$

$$p(y = 0|x; w) = 1 - \phi(z)$$

$$p(y|x; w) = \phi(z)^y (1 - \phi(z))^{(1-y)}$$

极大似然估计方法（Maximum Likelihood Estimate，MLE）

根据给定的训练集估计出参数w

Classifier-LR

$P(x|\theta)$

x表示某一个**具体的数据**；θ表示模型的**参数**

$f(x,y)=x^y$

指数函数

幂函数

如果θ是已知确定的， x是变量，这个函数叫做概率函数(probability function)，它描述对于不同的样本点x ，其出现概率是多少

如果x是已知确定的， θ是变量，这个函数叫做似然函数(likelihood function), 它描述对于不同的模型参数，出现x这个样本点的概率是多少

Classifier-LR

假设有一个造币厂生产某种**硬币**，现在我们拿到了一枚这种硬币，想试试这硬币是不是均匀的。即想知道抛这枚硬币，**正反面出现的概率**（记为θ）各是多少?

第一步，造数据!

拿这枚硬币抛了10次，得到的数据（x）是：反正正正正反正正正反。

第二步，假设分布!

我们想求的正面概率θ是模型参数，而抛硬币模型我们可以假设是**二项分布**。

Classifier-LR

假设有一个造币厂生产某种**硬币**，现在我们拿到了一枚这种硬币，想试试这硬币是不是均匀的。即想知道抛这枚硬币，**正反面出现的概率**（记为θ）各是多少?

第三步，构造似然函数!

$f(x,\theta)$

$= (1-\theta)\times\theta\times\theta\times\theta\times\theta\times(1-\theta)\times\theta\times\theta\times\theta\times(1-\theta)$

$= \theta^7(1-\theta)^3 = f(\theta)$

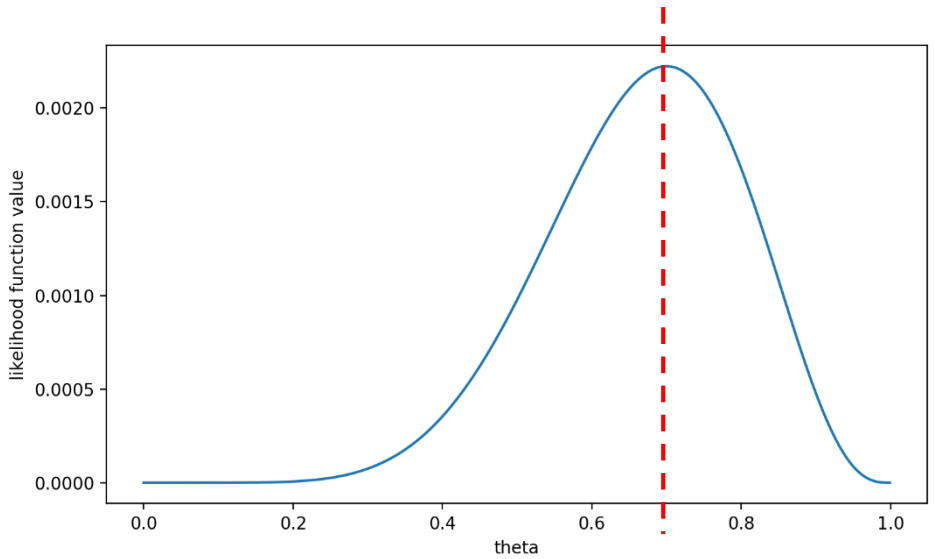
注意，这是个只关于θ的函数。

最大似然估计，顾名思义，就是要最大化这个函数。

Classifier-LR

假设有一个造币厂生产某种**硬币**，现在我们拿到了一枚这种硬币，想试试这硬币是不是均匀的。即想知道抛这枚硬币，**正反面出现的概率**（记为θ）各是多少?

第三步，求似然函数最大值



可以看出，在$\theta = 0.7$时，似然函数取得最大值

Classifier-LR

$$p(y|x;w) = \phi(z)^{y}(1-\phi(z))^{(1-y)}$$

$$L(w) = \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};w) = \prod_{i=1}^{n} \left(\phi(z^{(i)})\right)^{y^{(i)}} (1-\phi(z^{(i)}))^{1-y^{(i)}}$$

$$l(w) = lnL(w) = \sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1-y^{(i)}) ln\left(1-\phi(z^{(i)})\right)$$

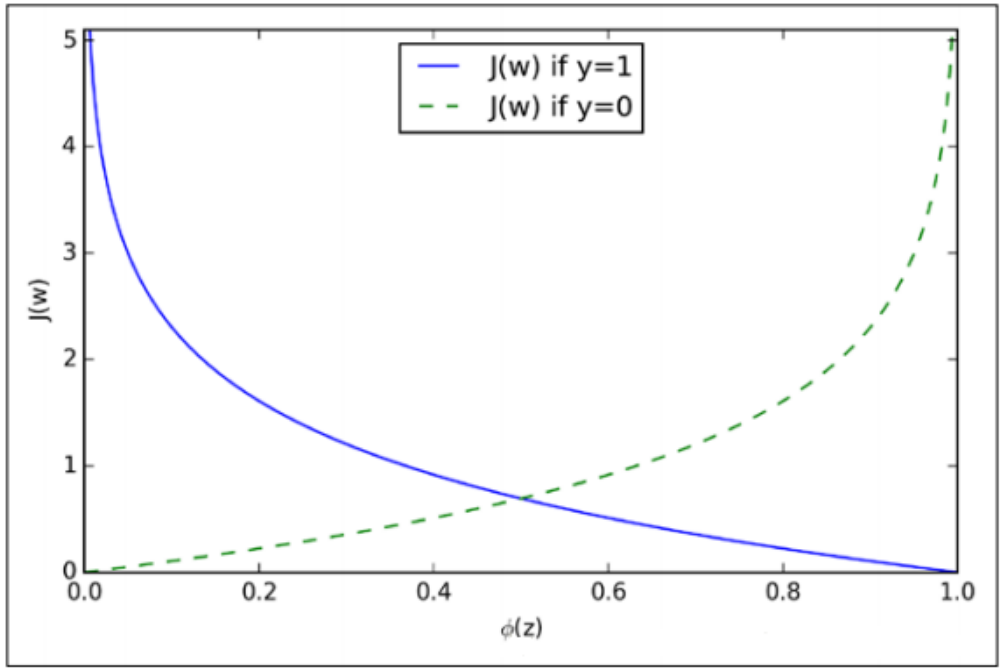
$$\underset{w}{\operatorname{argmax}}\, l(w) \quad \Rightarrow \quad \underset{w}{\operatorname{argmin}} -\frac{1}{n} l(w)$$

$$\underset{w}{\operatorname{argmin}}\, \mathcal{L}(w) = \underset{w}{\operatorname{argmin}} -\frac{1}{n} \sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1-y^{(i)}) ln\left(1-\phi(z^{(i)})\right)$$

Classifier-LR

$$\mathcal{L}(\phi(z), y; w) = -yln\big(\phi(z)\big) - (1-y)\ln\big(1-\phi(z)\big)$$

$$= \begin{cases} -ln\big(\phi(z)\big) & if\ y = 1 \\ -\ln\big(1-\phi(z)\big) & if\ y = 0 \end{cases}$$



Classifier-LR

$$\underset{w}{\operatorname{argmin}}\,\mathcal{L}(w) = \underset{w}{\operatorname{argmin}} -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1-y^{(i)}) ln\left(1-\phi(z^{(i)})\right)$$

利用梯度下降法求参数

梯度的负方向就是**函数下降最快**的方向

$$f(x) = \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + R_n(x)$$

$$f(x+\delta) - f(x) \approx f'(x)\cdot\delta$$

Classifier-LR

$$\underset{w}{\operatorname{argmin}}\,\mathcal{L}(w) = \underset{w}{\operatorname{argmin}} -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1 - y^{(i)}) ln\left(1 - \phi(z^{(i)})\right)$$

利用梯度下降法求参数

梯度的负方向就是函数下降最快的方向

$$f(x+\delta) - f(x) \approx f'(x) \cdot \delta$$

$$f'(x) \cdot \delta = ||f'(x)|| \cdot ||\delta|| \cdot \cos\theta$$

当θ=π时，也就是δ在$f'(x)$的**负方向**上时，

取得最小值，也就是下降的最快的方向了

Classifier-LR

$$\underset{w}{\operatorname{argmin}}\,\mathcal{L}(w) = \underset{w}{\operatorname{argmin}} -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1 - y^{(i)}) ln\left(1 - \phi(z^{(i)})\right)$$

利用梯度下降法求参数

沿着梯度的负方向进行下降

$$w_j := w_j + \Delta w_j, \Delta w_j = -\eta \frac{\partial \mathcal{L}(w)}{\partial w_j}$$

其中，w_j表示第j个特征的权重；η为学习率，用来控制步长。

Classifier-LR

$$\underset{w}{\operatorname{argmin}}\,\mathcal{L}(w) = \underset{w}{\operatorname{argmin}} -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1-y^{(i)}) ln\left(1-\phi(z^{(i)})\right)$$

利用梯度下降法求参数

沿着梯度的负方向进行下降

$$\frac{\partial \mathcal{L}(w)}{\partial w_j} = \sum_{i=1}^{n}\left(y^{(i)}\frac{1}{\phi(z^{(i)})} - (1-y^{(i)})\frac{1}{1-\phi(z^{(i)})}\right)\frac{\partial \phi(z^{(i)})}{\partial w_j}$$

$$= \sum_{i=1}^{n}\left(y^{(i)}\frac{1}{\phi(z^{(i)})} - (1-y^{(i)})\frac{1}{1-\phi(z^{(i)})}\right)\phi(z^{(i)})(1-\phi(z^{(i)}))\frac{\partial z^{(i)}}{\partial w_j}$$

$$= \sum_{i=1}^{n}\left(y^{(i)}(1-\phi(z^{(i)})) - (1-y^{(i)})\phi(z^{(i)})\right)x_j^{(i)} = -\sum_{i=1}^{n}\left(y^{(i)} - \phi(z^{(i)})\right)x_j^{(i)}$$

Classifier-LR

$$\underset{w}{\operatorname{argmin}}\,\mathcal{L}(w) = \underset{w}{\operatorname{argmin}} -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} ln\left(\phi(z^{(i)})\right) + (1 - y^{(i)}) ln\left(1 - \phi(z^{(i)})\right)$$

利用梯度下降法求参数

沿着梯度的**负方向**进行下降

$$w_j := w_j + \eta \sum_{i=1}^{n} \left(y^{(i)} - \phi(z^{(i)})\right) x_j^{(i)}$$

随机梯度下降法 (Stochastic gradient descent, SGD)

$$w_j := w_j + \eta \left(y^{(i)} - \phi(z^{(i)})\right) x_j^{(i)}, \qquad for\ i\ in\ range(n)$$

Classifier-LR

期望风险（expected risk）：

描述模型与**训练样本**以及**测试样本**的拟合程度。

对所有样本（包含未知和已知的样本）的预测能力，是**全局概念**，是理想化的不可求的

经验风险（empirical risk）：

描述**模型**与**训练样本**的拟合程度。

对所有训练样本的预测能力，是**局部概念**，是现实可求的

结构风险（structural risk）：

描述模型与训练样本的拟合程度，以及**模型的复杂程度。**

对过拟合现象的处理

Classifier-LR

针对训练样本时，我们**希望经验风险最小**，说明模型能够**很好的拟合训练样本；**

但是，此时模型针对测试样本，可能并不能产生很好的拟合效果。

当**模型的复杂程度过高**，虽然经验风险可能很小，但是由于高复杂度导致模型的**泛化能力变弱**，从而导致**过拟合。**

模型不仅能够比较优秀的拟合训练样本，而且能够具备比较优秀的泛化能力。

Classifier-LR

$$w = argmin \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f(x_i; w)) + \lambda \Omega(f)$$

其中，$\mathcal{L}(y_i, f(x_i; w))$表示损失函数，$N$表示训练样本的个数；

$\Omega(f)$称为**正则项**（或者惩罚项），表示**模型的复杂程度**；

λ是系数，用于权衡经验风险与模型复杂程度；

λ的取值，正是机器学习中，我们需要调节的参数。

Classifier-LR

$\Omega(f)$

称为**正则项**（regularizer）/ **惩罚项**（penalty term），表示模型的复杂程度

正则项一般是模型复杂程度的**单调递增函数**，模型**越复杂**，正则项的**值越大**

L_p范数是常用的正则项，也就是说，正则项通常是模型参数向量的范数

常见的有：L_0范数、L_1范数、L_2范数

Classifier-LR

L_0范数

描述向量中**非0元素的个数**，可以实现模型参数向量的稀疏。

实现模型参数向量的稀疏有什么好处呢？ ??

主要有以下两点：进行特征选择、提高模型可解释性。

Classifier-LR

L_1范数 (Lasso)

描述向量中各个元素的**绝对值之和**，可以实现模型参数向量的稀疏。

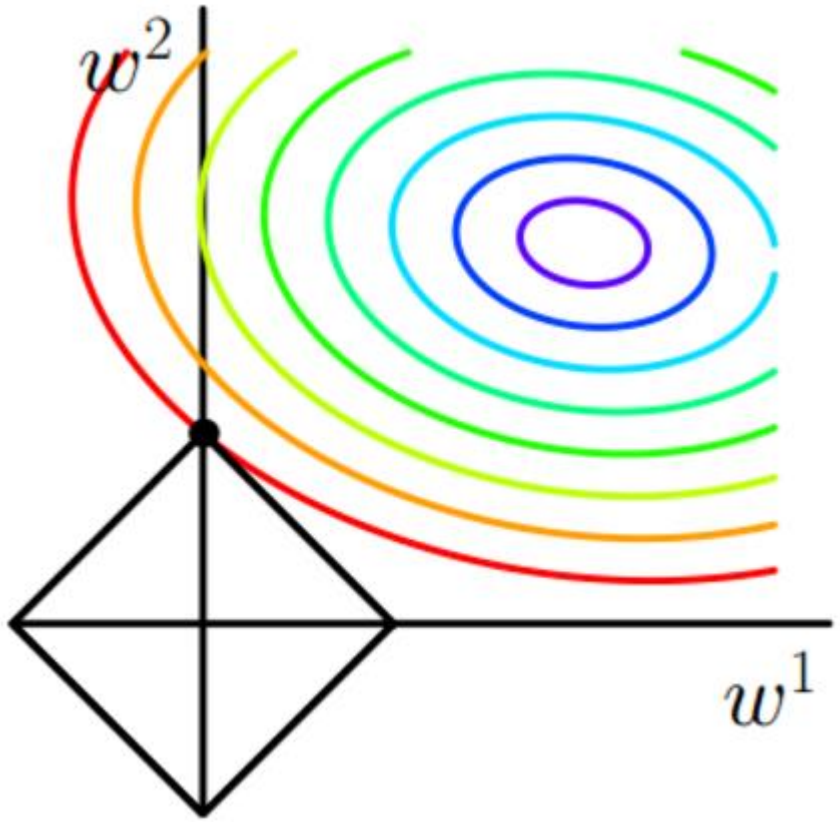
L_1范数是L_0范数的最优凸近似，L_1范数的优化求解相对L_0容易。

L_1范数不是连续可导的，求解相比L_2范数复杂

L_1范数为什么能够实现模型参数向量的稀疏呢?

Classifier-LR

$$\min_{w} \frac{1}{N} \|Y - XW\|^2, \qquad s.t. \|W\|_1 \leq C$$



Classifier-LR

L_2范数 (Ridge)

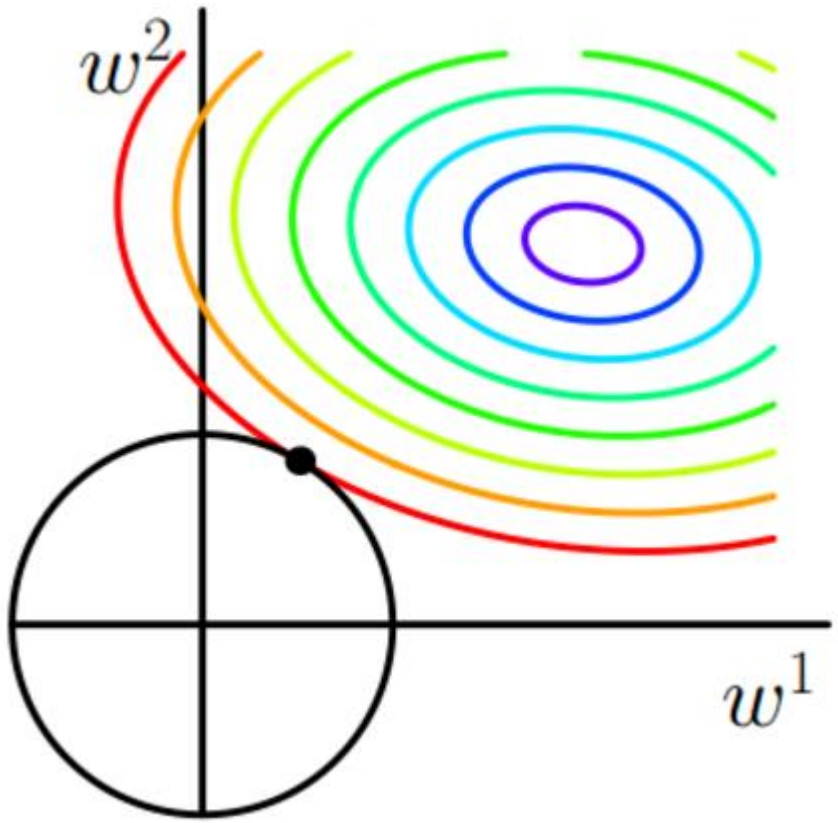
描述向量中各元素的**平方之和**，然后求**平方根**。

L2范数会选择更多的特征，这些特征都会趋近于0。

L_2范数不能实现模型参数向量的稀疏?

Classifier-LR

$$\min_{w} \frac{1}{N} \|Y - XW\|^2, \qquad s.t. \|W\|_2 \leq C$$



Classifier-LR

Tips：

L_1范数会选择少量的特征，其他的**特征都是0**；

L_2范数会选择更多的特征，这些**特征都会趋近于0**。

如果在所有特征中，只有少数特征起主要作用的情况下，

那么选择L_1范数比较合适，因为它能自动选择特征；

如果在所有特征中，大部分特征都能起作用，

而且起的作用很平均，那么使用L_2范数也许更合适。

Classifier-LR

Tips：

如果模型的特征非常多，我们希望一些**不重要特征的系数归零**，

从而让**模型的系数稀疏化**，那么选择**L1正则化。**

如果我们需要**相对精确的多元逻辑回归**模型，那么L1正则化可能就不合适了。

在调参时，如果我们目的**仅仅是为了解决过拟合问题**，一般选择**L2正则化**就可以；

但是，如果选择L2正则化之后，发现还是存在过拟合的问题，就可以考虑L1正则化。

Classifier-LR

弹性网络（Elastic Net）

通过弹性系数调节，同时使用L_1范数和L_2范数作为正则化矩阵的线性回归模型

常用于**只有很少的权重非零的稀疏模型**

当多个特征和另一个特征相关的时候弹性网络非常有用

Lasso 倾向于随机选择其中一个，而弹性网络更倾向于选择两个

在实践中，Lasso 和 Ridge 之间权衡的一个优势是：

允许在循环过程中继承 Ridge 的**稳定性**

Classifier-LR

Ridge Regression $\min_w ||Xw - y||_2{}^2 + \alpha||w||_2{}^2$

Lasso Regression $\min_w \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha||w||_1$

Elastic Regression $\min_w \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$

Logistic Regression with L2 $\min_{w,c} \frac{1}{2}w^T w + C\sum_{i=1}^{n}\log(\exp(-y_i(X_i^T w + c)) + 1).$

Logistic Regression with L1 $\min_{w,c} \|w\|_1 + C\sum_{i=1}^{n}\log(\exp(-y_i(X_i^T w + c)) + 1).$

Classifier-LR

sklearn.linear_model.LogisticRegression

class sklearn.linear_model. **LogisticRegression** (*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1*) [source]

sklearn.linear_model.SGDClassifier

class sklearn.linear_model. **SGDClassifier** (*loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, warm_start=False, average=False, n_iter=None*) [source]

Classifier-LR

In [16]:
```
from sklearn.linear_model import SGDClassifier
```

In [23]:
```
clf_1 = SGDClassifier(max_iter=1000)
```

In [24]:
```
clf_1.fit(X, y)
```

```
/Users/meizu/Work/py_env/my_scikit_learn/lib/python2.7/site-packages/sklearn
FutureWarning: max_iter and tol parameters have been added in SGDClassifier
ft unset, the default value for tol in 0.19 and 0.20 will be None (which is
ect) but will change in 0.21 to 1e-3. Specify tol to silence this warning.
  FutureWarning)
```

Out[24]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=1000,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [25]:
```
X[0:3,:]
```

Out[25]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2]])
```

In [26]:
```
clf_1.predict(X[:3, :])
```

Out[26]:
```
array([0, 0, 0])
```

In [27]:
```
clf_1.score(X, y)
```

Out[27]:
```
0.96
```

Classifier-LR

```
In [28]: from sklearn.linear_model import ElasticNet

In [35]: clf_2 = ElasticNet(random_state=0)

In [36]: clf_2.fit(X, y)

Out[36]: ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
               max_iter=1000, normalize=False, positive=False, precompute=False,
               random_state=0, selection='cyclic', tol=0.0001, warm_start=False)

In [37]: X[0:3,:]

Out[37]: array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2]])

In [38]: clf_2.predict(X[:3,:])

Out[38]: array([0.43380935, 0.43380935, 0.40979787])

In [39]: clf_2.score(X, y)

Out[39]: 0.714362142129827

In [ ]:
```

Classifier-LR

In [1]:
```
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
```

In [3]:
```
clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial')
```

In [4]:
```
clf.fit(X, y)
```

```
/Users/meizu/Work/py_env/my_scikit_learn/lib/python2.7/site-packages/sklearn/linear_m
eWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

Out[4]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=None, penalty='l2', random_state=0, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [13]:
```
X[0:3,:]
```

Out[13]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2]])
```

In [12]:
```
clf.predict(X[:3,:])
```

Out[12]:
```
array([0, 0, 0])
```

In [14]:
```
clf.predict_proba(X[:3, :])
```

Out[14]:
```
array([[9.81802911e-01, 1.81970751e-02, 1.43580537e-08],
       [9.71729527e-01, 2.82704429e-02, 3.00353141e-08],
       [9.85452757e-01, 1.45472311e-02, 1.22691633e-08]])
```

In [15]:
```
clf.score(X, y)
```

Out[15]:
```
0.9733333333333334
```

OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. Logistic Regression
6. **Naive Bayes**
7. Decision Tree
8. 分类器选型

Classifier-NB

Bayes 贝叶斯

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

$$P(L|f) = \frac{P(Lf)}{P(f)} = \frac{P(f|L)P(L)}{P(f)}$$

Classifier-NB

Naive Bayes 朴素贝叶斯

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

$$P(L|f) = \frac{P(Lf)}{P(f)} = \frac{P(f|L)P(L)}{P(f)}$$

随机变量序列$\mathrm{S} = \{x_0, x_1 \cdots, x_\mathrm{n}\},\ \mathrm{P}(\mathrm{S}) = \mathrm{P}(x_0 x_1 \cdots x_\mathrm{n})$

$$\mathrm{P}(\mathrm{S}) = \mathrm{P}(x_0 x_1 \cdots x_\mathrm{n}) = P(x_\mathrm{n}|x_0 x_1 \cdots x_{\mathrm{n}-1})P(x_0 x_1 \cdots x_{\mathrm{n}-1})$$

$$= P(x_\mathrm{n}|x_0 x_1 \cdots x_{\mathrm{n}-1})P(x_{\mathrm{n}-1}|x_0 x_1 \cdots x_{\mathrm{n}-2})P(x_0 x_1 \cdots x_{\mathrm{n}-2})$$

$$= P(x_0)\prod_{i=1}^{n} P(x_\mathrm{n}|x_0 x_1 \cdots x_{\mathrm{i}-1})$$

Naïve?

Classifier-NB

随机变量序列$\mathrm{S}=\{x_0,x_1\cdots,x_\mathrm{n}\},\ \mathrm{P}(\mathrm{S})=\mathrm{P}(x_0x_1\cdots x_\mathrm{n})$

$$\mathrm{P}(\mathrm{S})=\mathrm{P}(x_0x_1\cdots x_\mathrm{n})=P(x_\mathrm{n}|x_0x_1\cdots x_{\mathrm{n}-1})P(x_0x_1\cdots x_{\mathrm{n}-1})$$

$$=P(x_\mathrm{n}|x_0x_1\cdots x_{\mathrm{n}-1})P(x_{\mathrm{n}-1}|x_0x_1\cdots x_{\mathrm{n}-2})P(x_0x_1\cdots x_{\mathrm{n}-2})$$

$$=P(x_0)\prod_{i=1}^{n}P(x_\mathrm{n}|x_0x_1\cdots x_{\mathrm{i}-1})$$

Naïve? 假设各特征之间相互独立

$$=P(x_0)\prod_{i=1}^{n}P(x_i)$$

$$=\prod_{i=0}^{n}P(x_i)$$

Classifier-NB

Eg: 垃圾邮件分类器

- 训练数据，一些垃圾邮件和一些正常邮件。
- 分别用这两类邮件训练出两个 Language Model，亦即统计出两类邮件中各个单词出现的频率，进而计算出它的（近似）概率。
- 对于待分类的文档 d，根据贝叶斯公式，我们有

  $P(d)P(spam|d) = P(spam)P(d|spam)$

  $P(d)P(ham|d) = P(ham)P(d|ham)$，

  $P(spam|d) - P(ham|d)$ 正比于 $P(spam)P(d|spam) - P(ham)P(d|ham)$

Classifier-NB

- 其中 P(spam) 和 P(ham) 是**先验概率**，是两个**常**量，通常根据 domain specific 的知识估计出来，或者直接设置为 0.5 。
- P(d|spam) 和 P(d|ham) 就分别是两个 language model 所给出的概率了。展开来就是文档 d 的字符串里的单词分别在两个 language model 里的概率值的乘积。
- 这样，差值大于零时就可以判断为垃圾邮件，否者则为正常邮件。还可以设置一个阈值，在差值的绝对值小于这个阈值的时候表示无法判断，以降低判断的错误率。

Classifier-NB

sklearn.naive_bayes.GaussianNB

class sklearn.naive_bayes. **GaussianNB** (*priors=None*) [source]

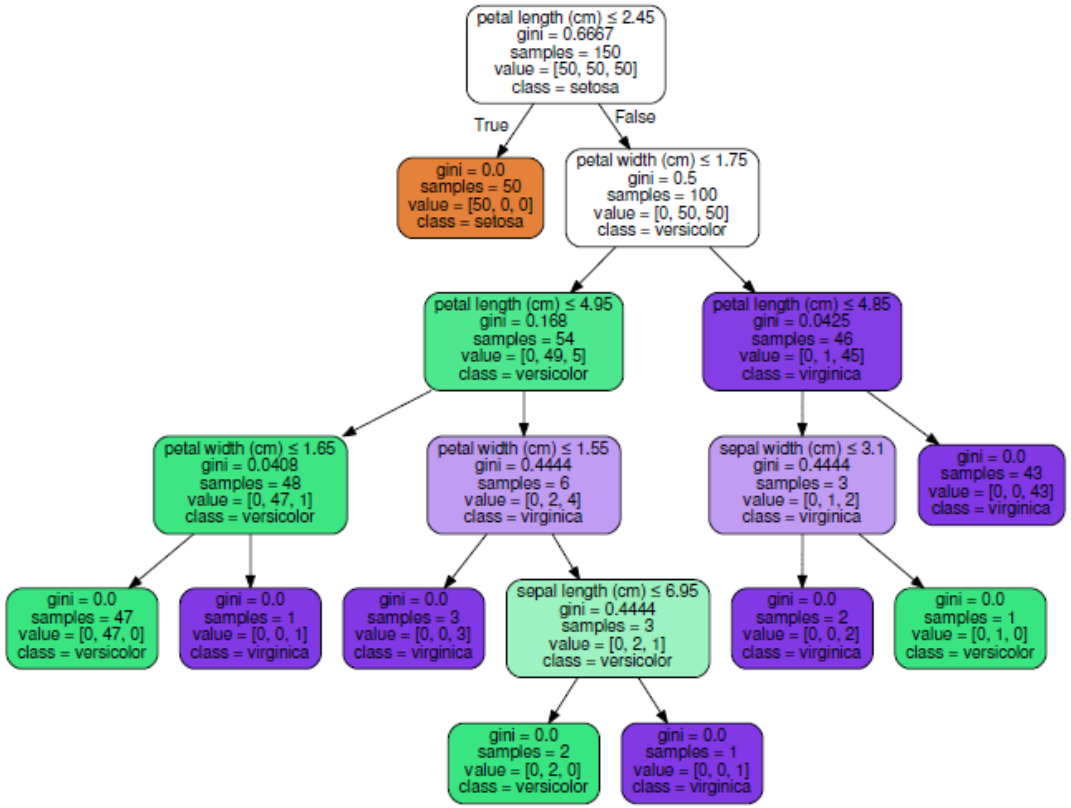
```
In [40]: from sklearn.naive_bayes import GaussianNB

In [41]: nb = GaussianNB()
         nb.fit(X, y)

Out[41]: GaussianNB(priors=None, var_smoothing=1e-09)

In [42]: X[0:3,:]

Out[42]: array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2]])

In [43]: nb.predict(X[0:3,:])

Out[43]: array([0, 0, 0])

In [44]: nb.score(X,y)

Out[44]: 0.96
```

OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. Logistic Regression
6. Naive Bayes
7. **Decision Tree**
8. 分类器选型

Classifier-DT

DT （Decision tree），决策树分类算法



Classifier-DT

GBDT for classification?

Classifier-DT

用户ID	消费金额	百度提问/回答	年龄
A	800	提问	14
B	500	回答	16
C	1200	提问	24
D	3000	回答	26
E	3000	提问	?

训练数据的均值：20岁

（这个很重要，因为一开始需要设置预测的**均值**，这样后面才会有残差）

每个样本的特征有两个：

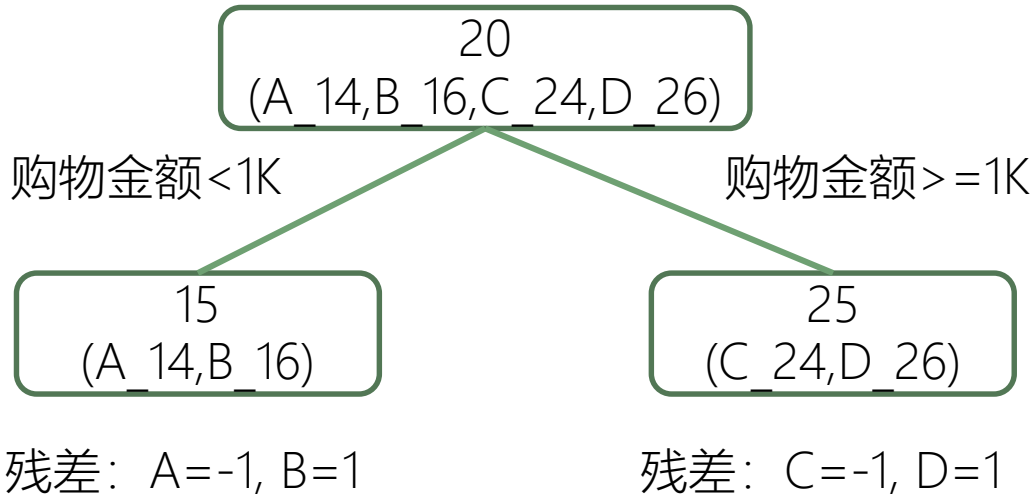
- 购物金额是否小于1K
- 经常去百度提问还是回答

Classifier-DT

预测年龄问题

训练环节

首先，输入初值20岁，根据第一个特征（具体选择哪些特征可以根据信息增益来计算选择，假设是“购物金额”），可以把4个样本分成两类



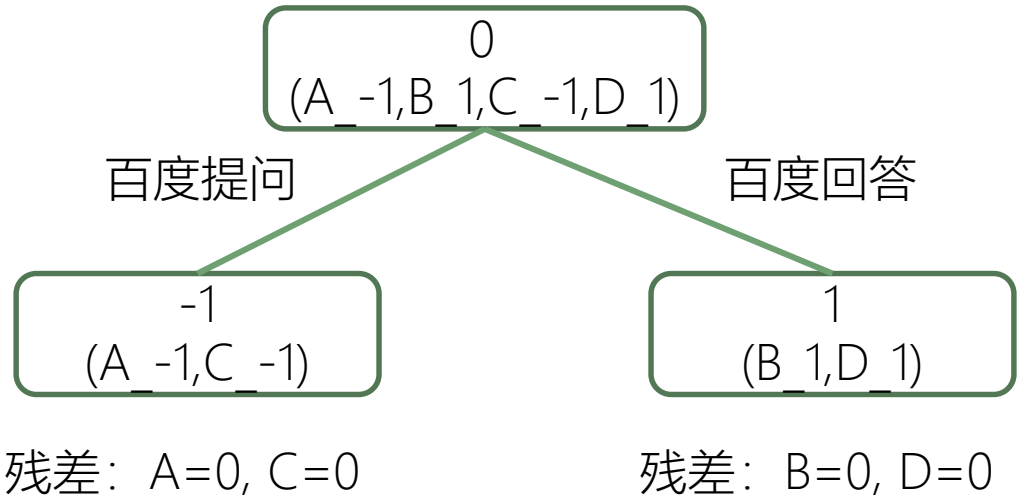
Classifier-DT

预测年龄问题

训练环节

学习第二棵决策树，我们把第一棵的残差样本（A, -1岁）、（B，1岁）、（C, -1岁）、（D, 1岁）输入，并将残差均值作为根节点

残差总和为0，停止学习



Classifier-DT

预测年龄问题

测试环节

测试样本：请预测一个**购物金额为3k，经常去百度问**淘宝相关问题的女生的年龄

我们提取2个特征：购物金额3k，经常去百度上面问问题

第一棵树 —> 购物金额大于1k —> 右叶子，初步说明这个女生25岁

第二棵树 —> 经常去百度提问 —> 左叶子，说明这个女生的残差为-1

叠加前面每棵树得到的结果：25-1=24岁，**最终预测结果为24岁**

Classifier-DT

GBDT for classification?

弱分类器的输出的结果**可减（相减有意义）**

分类中**类别相减**是没有意义的。上一轮输出的是样本 x 属于 A类，本一轮训练输出的是样本 x 属于 B类。 A 和 B 很多时候甚至都没有比较的意义，A类 - B类是没有意义的。怎么办?

softmax启发：概率相减

Classifier-DT

GBDT for classification?

二分类 or 多分类?

多分类转换为二分类问题，$A\ vs\ \bar{A}$

Classifier-DT

假设样本 X 总共有 K 类 (K = 3)。预测一个样本 x （label = 2），使用 gbdt 来判断 x 属于样本的哪一类?

Step1: 向量表示类别

0表示样本不属于该类，1表示样本属于该类。

Label=2 (第二类)，所以第二类对应的向量维度为1，其他位置为0。用 [0,1,0] 来表示。

多分类转换为二分类问题，$A\ vs\ \bar{A}$

每轮的训练的时候是同时训练三颗**树 （CART）**。第一颗树针对样本x的第一类，输入为（x, 0）。第二颗树输入针对样本x 的第二类，输入为（x,1）。第三颗树针对样本x 的第三类，输入为（x，0）

Classifier-DT

假设样本 X 总共有 K 类 (K = 3)。预测一个样本 x （label = 2），使用 gbdt 来判断 x 属于样本的哪一类?

Step2: 仿制Softmax

最终三颗树的输出结果 $f_1(x)$,$f_2(x)$, $f_3(x)$

仿照多分类的逻辑回归，使用softmax 来产生概率，则属于类别 1 的概率

$$p_1 = exp(f_1(x)) / \sum_{k=1}^{3} exp(f_k(x))$$

Classifier-DT

假设样本 X 总共有 K 类 (K = 3)。预测一个样本 x （label = 2），使用 gbdt 来判断 x 属于样本的哪一类?

Step3: 残差迭代

类别1，残差：$y_{11}(x) = 0 - p_1(x)$

类别2，残差：$y_{22}(x) = 1 - p_2(x)$

类别3，残差：$y_{33}(x) = 0 - p_3(x)$

以残差为后续学习的输入继续迭代

Classifier-DT

假设样本 X 总共有 K 类 (K = 3)。预测一个样本 x （label = 2），使用 gbdt 来判断 x 属于样本的哪一类?

样本编号	花萼长度 (cm)	花萼宽度 (cm)	花瓣长度 (cm)	花瓣宽度	label
1	5.1	3.5	1.4	0.2	1
2	4.9	3.0	1.4	0.2	1
3	7.0	3.2	4.7	1.4	2
4	6.4	3.2	4.5	1.5	2
5	6.3	3.3	6.0	2.5	3
6	5.8	2.7	5.1	1.9	3

Classifier-DT

样本编号	花萼长度(cm)	花萼宽度(cm)	花瓣长度(cm)	花瓣宽度	label
1	5.1	3.5	1.4	0.2	1

三个类别：[1, 0, 0]，[0, 1, 0]，[0, 0, 1]

三个Tree: 分别针对三个类别

以样本 1 为例。

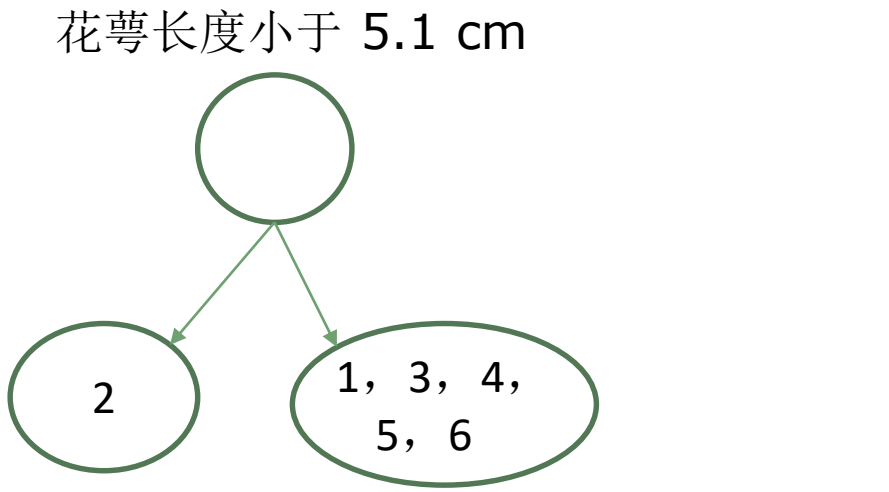
针对 Tree1 的训练样本是 [5.1,3.5,1.4,0.2,1]

针对 Tree2 的训练样本是 [5.1,3.5,1.4,0.2,0]

针对 Tree3 的训练样本是 [5.1,3.5,1.4,0.2,0]

Classifier-DT

样本编号	花萼长度(cm)	label
1	5.1	1
2	4.9	1
3	7.0	2
4	6.4	2
5	6.3	3
6	5.8	3

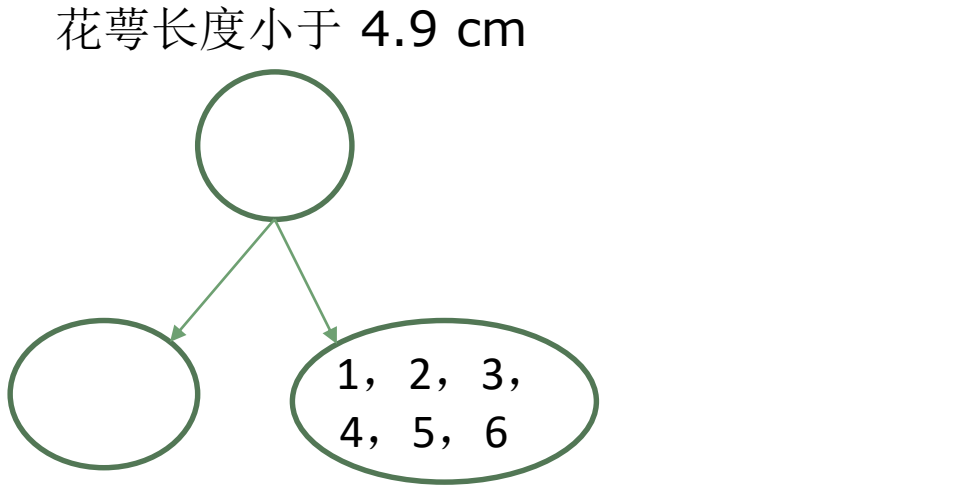


label均值：1　label均值：(1+0+0+0+0)/5=0.2

Loss：$(1-0.2)^2 + (1-1)^2 + (0-0.2)^2 + (0-0.2)^2 + (0-0.2)^2 + (0-0.2)^2 = 0.84$

Classifier-DT

样本编号	花萼长度(cm)	label
1	5.1	1
2	4.9	1
3	7.0	2
4	6.4	2
5	6.3	3
6	5.8	3



label均值：0　　label均值(1+1+0+0+0+0)/6=0.33

Loss：$(1-0.333)^2 + (1-0.333)^2 + (0-0.333)^2 + (0-0.333)^2 + (0-0.333)^2 + (0-0.333)^2 = 2.244189$

Classifier-DT

样本编号	花萼长度(cm)	label
1	5.1	1
2	4.9	1
3	7.0	2
4	6.4	2
5	6.3	3
6	5.8	3

以花萼长度为feature选取分支

取loss最小的

特征值为5.1 cm。这个时候损失函数最小为 0.8

Classifier-DT

样本编号	花萼长度(cm)	label
1	5.1	1
2	4.9	1
3	7.0	2
4	6.4	2
5	6.3	3
6	5.8	3

以花萼长度为feature选取分支

取loss最小的

特征值为5.1 cm。这个时候损失函数最小为 0.8

计划每个树下的输出：

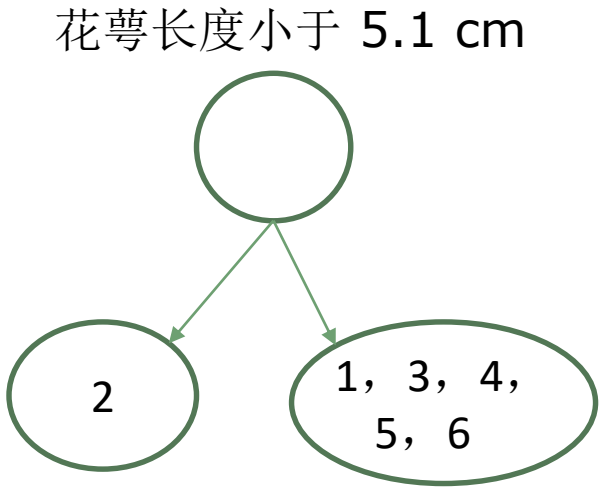
$$f(x) = \sum_{x \epsilon R_1} y_1 * I(x \epsilon R_1) + \sum_{x \epsilon R_2} y_2 * I(x \epsilon R_2)$$

Classifier-DT

以花萼长度为feature选取分支

取loss最小的

特征值为5.1 cm。这个时候损失函数最小为 0.8



label均值：1　label均值：(1+0+0+0+0)/5=0.2

计划每个树下的输出：$f(x) = \sum_{x \epsilon R_1} y_1 * I(x \epsilon R_1) + \sum_{x \epsilon R_2} y_2 * I(x \epsilon R_2)$

$$f_1(x) = 1 * 1 + 0.2 * 5 = 2$$

重复以上过程得到 $f_2(x), f_3(x)$

Classifier-DT

重复以上过程得到 $f_2(x)$, $f_3(x)$

计算最终的类别概率： $p_1 = exp(f_1(x)) / \sum_{k=1}^{3} exp(f_k(x))$

OUTLINES

1. 预备知识
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. Logistic Regression
6. Naive Bayes
7. Decision Tree
8. **分类器选型**

分类算法选型

NB - 优势

- 实现简单、训练速度快
- 适应不同量级的数据
- 支持增量式训练
- 可解释性强

NB -劣势

- 需假设条件独立
- 难以处理特征组合

分类算法选型

DT - 优势

- 计算复杂度不高
- 有效处理连续特征
- 适合小规模数据
- 可解释性强

DT -劣势

- 不擅长对数值结果进行预测
- 不支持增量式训练
- 容易出现过拟合

分类算法选型

KNN - 优势

- 可用复杂函数进行数值预测
- 对噪声数据不敏感
- 支持增量式训练
- 可解释性强

KNN -劣势

- 存在样本不均衡问题
- K值选取对结果影响大
- 计算量大、内存消耗大
- 需要完整的训练数据

分类算法选型

SVM - 优势

- 适合大规模数据
- 有效处理连续特征
- 泛化错误率低
- 结果可解释性强

SVM -劣势

- 核函数及相关参数难以确定
- 原始分类器不支持多分类
- 内存消耗大，计算耗时多
- 过程可解释性差

分类算法选型

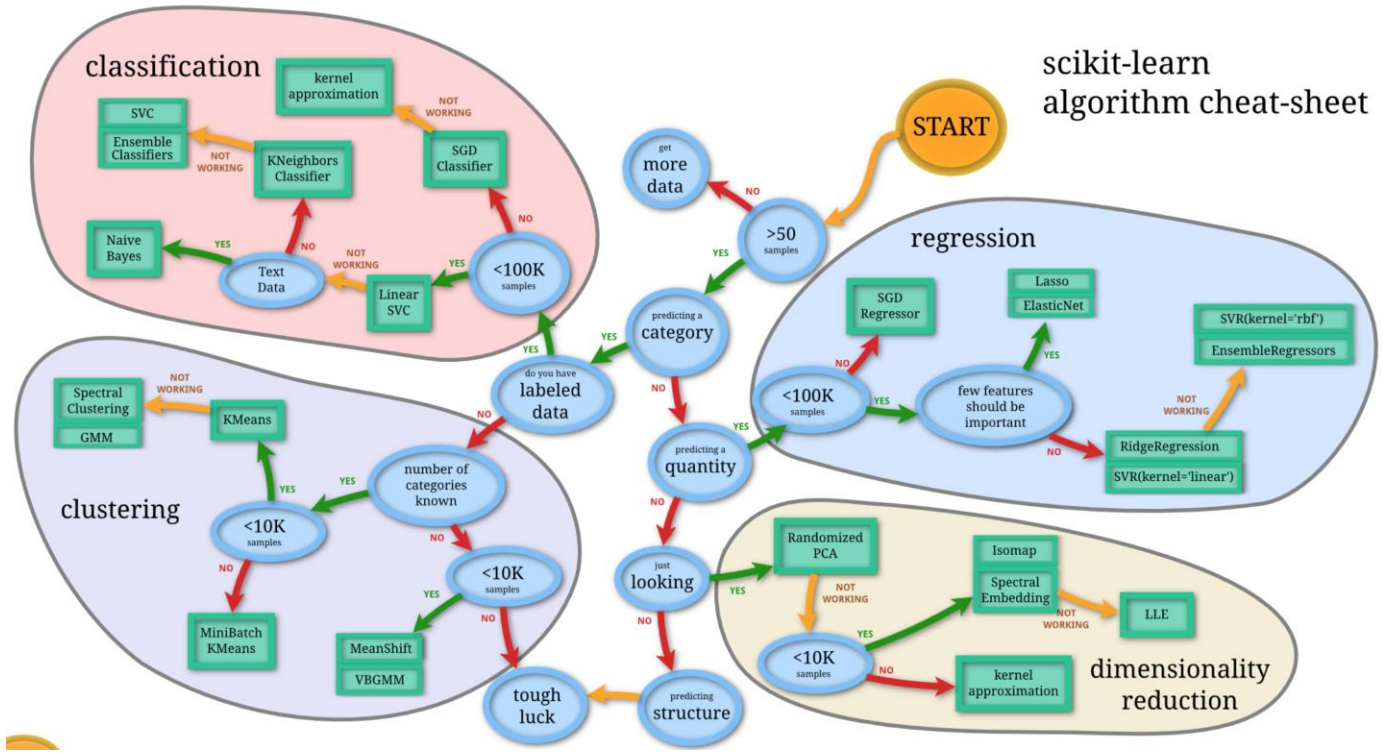
LR - 优势

- 适合大规模数据
- 计算快、存储消耗少
- 支持增量式训练
- 可解释性强

LR -劣势

- 容易欠拟合
- 分类精度可能不高
- 只能解决线性问题

分类算法选型


scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
Naive Bayes
YES
NO
Text Data
NOT WORKING
Linear SVC
YES
<100K samples
NO
get more data
NO
>50 samples
YES
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
predicting a category
YES
YES
do you have labeled data
NO
<100K samples
NO
YES
few features should be important
YES
NOT WORKING
RidgeRegression
SVR(kernel='linear')
NO
Spectral Clustering
NOT WORKING
GMM
KMeans
clustering
number of categories known
YES
YES
<10K samples
NO
NO
<10K samples
YES
NO
MiniBatch KMeans
MeanShift
VBGMM
predicting a quantity
YES
NO
just looking
YES
NO
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
tough luck
predicting structure

总结

1. 预备知识 - metrics
2. 分类问题定义-Classification
3. K-Nearest Neighbor (KNN)
4. Support Vector Machine(SVM)
5. Logistic Regression
6. Naive Bayes
7. Decision Tree
8. 分类器选型

Thanks!