1. 基本概念
   1. 监督学习

监督学习:训练样本中同时包含有特征和标签信息的机器学习算法.

监督学习包含**分类**和**回归**两种算法. 这类算法必须知道预测什么(这类算法必须知道目标变量的分类信息)才可进行.

数据集:

X, 特征.

, 类标签.

* 1. 监督学习的流程
     + 1. 训练部分:

获取数据  特征提取  监督学习  评价 

**2. 预测部分:**

模型  预测

**无监督学习:** 另一种机器学习算法,其样本只含有特征,不包含标签信息. 最典型的算法是聚类算法,降维算法.

* 1. **分类算法和回归算法**

**分类**算法的目标是从数个可能的分类标签中预测一个样本所属的分类标签. 例如,识别一个网站所用的语言.识别结果可以是英语,汉语,日语等语言中的一种.语言之间**没有连续性**.

**回归**算法的目标是预测一个**连续的数**(浮点型). 例如,由一个人的年龄,教育程度,住址来预测其收入.这里的收入的取值就可以是一定范围内的**任何值**.

* 1. 训练集和测试集

训练集:用于训练机器学习算法的数据样本集合.

机器学习的主要任务是**分类**. 比如,为了判断一个动物是不是鱼,我们可构建训练集如下.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **有鳃否** | **有鳍否** | **是鱼否** |
| 鲨 | 是 | 有 | 是 |
| 鲫鱼 | 是 | 有 | 是 |
| 河蚌 | 是 | 无 | 否 |
| 鲸 | 否 | 有 | 否 |
| 海豚 | 否 | 有 | 否 |

上表是一个包含5个训练样本的训练集.在该表中,”有鳃否’,”有鳍否”都可以取”有”,”无”两值,我们称它们为样本的**特征**.特征可以是数值型特征,和二值型特征. 上面表中的特征都是二值型特征.

目标变量是机器学习算法的预测结果. 上表中,“是鱼否”为**目标变量**.

为了测试机器学习算法的效果,我们通常用到两套独立的样本集: 训练集和**测试集**. 机器学习先使用训练集作为算法的输入,训练完成之后输入测试集.

* 1. 过拟合与欠拟合

如果一个模型可以对未曾见过的新数据作出精确的预测,则称该模型是**可归纳**模型. 假如我们建立了非常复杂的模型,我们总能在训练集上做到要多精确有多精确. 太复杂的模型会使得它在新数据集上预测的准确度下降.这种现象称为**过拟合**. 反之,过于简单的模型则可能导致**欠拟合**. 举例说明.

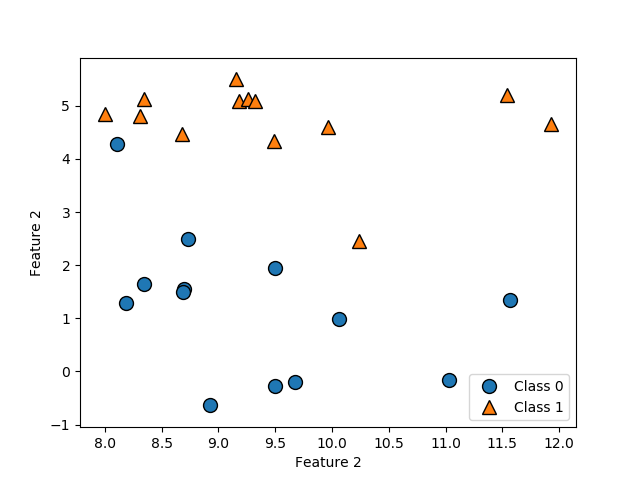
* 1. 数据集的使用举例

1. 数据集的散点图:Mglearn和matplotlib的使用.
2. #coding:utf-8
3. **import** mglearn
4. **import** matplotlib.pyplot as plt
6. #generate dataset
7. X, y = mglearn.datasets.make\_forge()
8. #plot dataset
9. mglearn.discrete\_scatter(X[:,0], X[:,1],y)
10. plt.legend(["Class 0", "Class 1"], loc=4)
11. plt.xlabel("Feature 1")
12. plt.ylabel("Feature 2")
13. **print**("X.shape:{}".format(X.shape))
14. plt.show()

问:特征数目有多少个?

数据点的数目有多少?

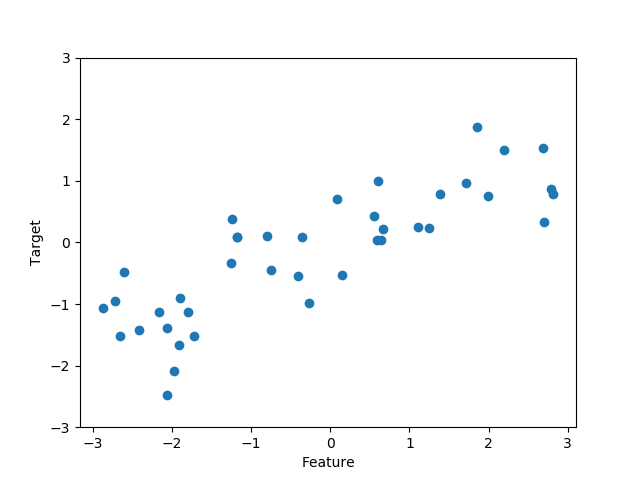
结果:



X.shape: (26, 2)

1. 回归目标与单个特征之间的关系,可用图形表示.代码如下:
2. **import** mglearn
3. **import** matplotlib.pyplot as plt
5. #generate dataset
6. X, y = mglearn.datasets.make\_wave(n\_samples=40)
7. #plot dataset
8. plt.plot(X,y,'o')
9. plt.ylim(-3,3)
10. plt.xlabel("Feature")
11. plt.ylabel("Target")
12. plt.show()

结果:



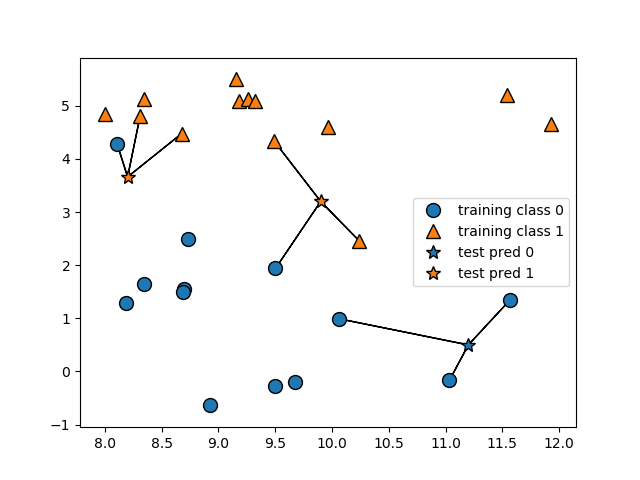
1. KNN算法

KNN算法(**k-Nearest Neighbors Algorithm**)是一种监督机器学习算法. 它通过存储训练集的数据来建立模型. 为了对新数据点作预测,该算法会在训练集中找出与该数据点最接近的一个或数个数据点,即**最近邻**.

* 1. KNN分类

1. 当我们只考虑一个最近邻时:
2. **import** mglearn
3. **import** matplotlib.pyplot as plt
4. **from** sklearn.datasets **import** load\_boston
6. boston = load\_boston()
8. **print**("Data shape:{}".format(boston.data.shape))
10. X,y = mglearn.datasets.load\_extended\_boston()
11. **print**("X.shape:{}".format(X.shape))
13. mglearn.plots.plot\_knn\_classification(n\_neighbors=1)
14. plt.show()

结果如下图:



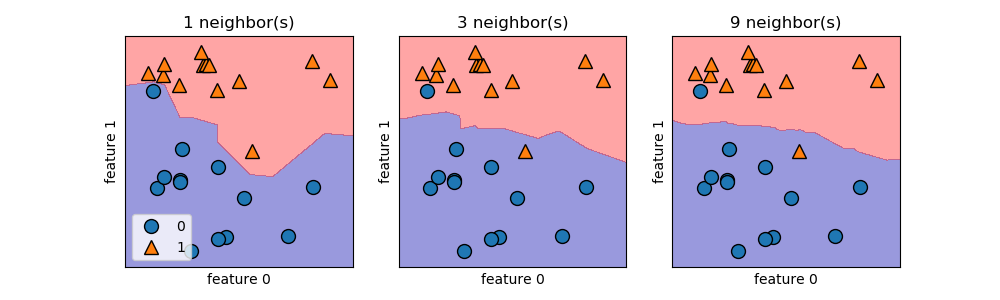
精确度计算:

1. 决策边界的计算.

代码如下:

1. **import** mglearn
2. **import** matplotlib.pyplot as plt
3. **from** sklearn.model\_selection **import** train\_test\_split
4. **from** sklearn.neighbors **import** KNeighborsClassifier
6. X,y = mglearn.datasets.make\_forge()
8. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, random\_state=0)
10. clf = KNeighborsClassifier(n\_neighbors=3)
11. clf.fit(X\_train, y\_train)
12. **print**("Test set prediction: {}".format(clf.predict(X\_test)))
13. **print**("Test set accuracy: {:.2f}".format(clf.score(X\_test,y\_test)))
15. fig,axes = plt.subplots(1,3,figsize=(10,3))
17. **for** n\_neighbors, ax **in** zip([1,3,9],axes):
18. # the fit method returns the object self, so we can instantiate and fit in one line
19. clf = KNeighborsClassifier(n\_neighbors=n\_neighbors).fit(X,y)
20. mglearn.plots.plot\_2d\_separator(clf, X,fill=True, eps=0.5,ax=ax,alpha=.4)
21. mglearn.discrete\_scatter(X[:,0],X[:,1],y, ax=ax)
22. ax.set\_title("{} neighbor(s)".format(n\_neighbors))
23. ax.set\_xlabel("feature 0")
24. ax.set\_ylabel("feature 1")
25. axes[0].legend(loc=3)
26. plt.show()

结果如下图:



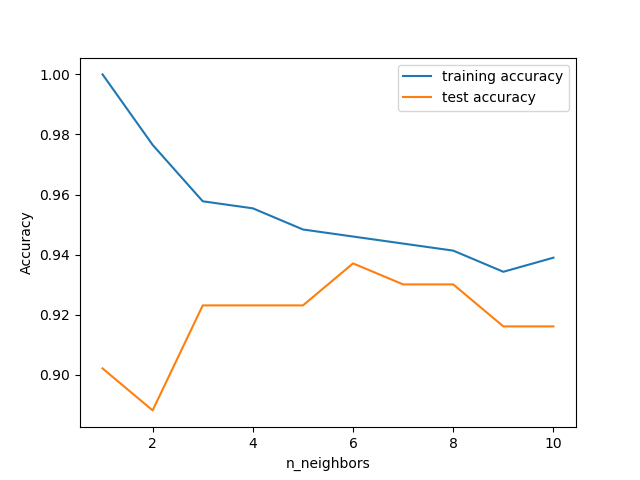
评估训练和测试的性能:

1. 训练集和测试集准确度之计算.

代码如下:

1. **from** sklearn.model\_selection **import** train\_test\_split
2. **from** sklearn.neighbors **import** KNeighborsClassifier
3. **from** sklearn.datasets **import** load\_breast\_cancer
4. **import** matplotlib.pyplot as plt
6. cancer = load\_breast\_cancer()
7. X\_train, X\_test, y\_train, y\_test = train\_test\_split(
8. cancer.data, cancer.target, stratify=cancer.target, random\_state=66)
10. training\_accuracy = []
11. test\_accuracy = []
13. # try n\_neighbors from 1 to 10
14. neighbors\_settings = range(1,11)
16. **for** n\_neighbors **in** neighbors\_settings:
17. # build the model
18. clf = KNeighborsClassifier(n\_neighbors=n\_neighbors)
19. clf.fit(X\_train,y\_train)
20. #record training set accuracy
21. training\_accuracy.append(clf.score(X\_train,y\_train))
22. #record generalization accuracy
23. test\_accuracy.append(clf.score(X\_test,y\_test))
25. plt.plot(neighbors\_settings,training\_accuracy, label="training accuracy")
26. plt.plot(neighbors\_settings,test\_accuracy, label="test accuracy")
27. plt.ylabel("Accuracy")
28. plt.xlabel("n\_neighbors")
29. plt.legend()
30. plt.show()

结果如下图:



小结:

KNN分类中,有两个参数值得关注:邻居的数目(k)和测量数据点之间距离的方式.

一般而言,k取3比较合理.但注意:不同情况,要不同地考虑!

测量数据点间的距离往往用的是欧几里得(Euclidean distance).

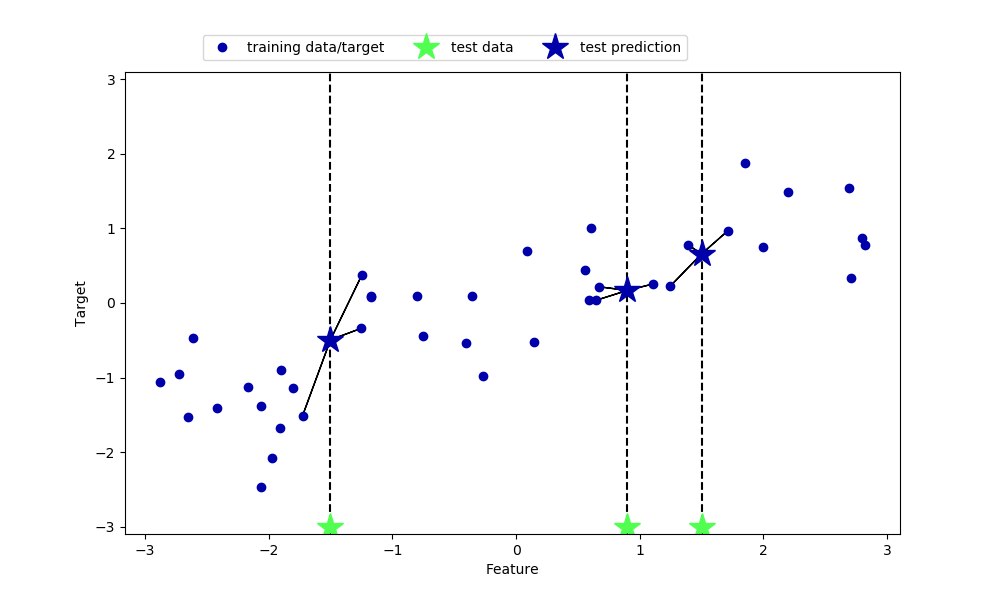
KNN算法之缺点:

* + - 1. 当数据集的特征很多时(),该算法的表现不是很好.
      2. 当样品数量很大时,预测的速度就会比较慢!

为了克服以上两个缺点, 我们需要更好的方法,如**线性模型**.

* 1. KNN回归

1. KNN回归算法示例(k=3).
2. **from** sklearn.model\_selection **import** train\_test\_split
3. **from** sklearn.neighbors **import** KNeighborsClassifier
4. **from** sklearn.datasets **import** load\_breast\_cancer
5. **import** matplotlib.pyplot as plt
6. **import** mglearn
8. X,y = mglearn.datasets.make\_forge()
9. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)
11. mglearn.plots.plot\_knn\_regression(n\_neighbors=3)
12. plt.show()

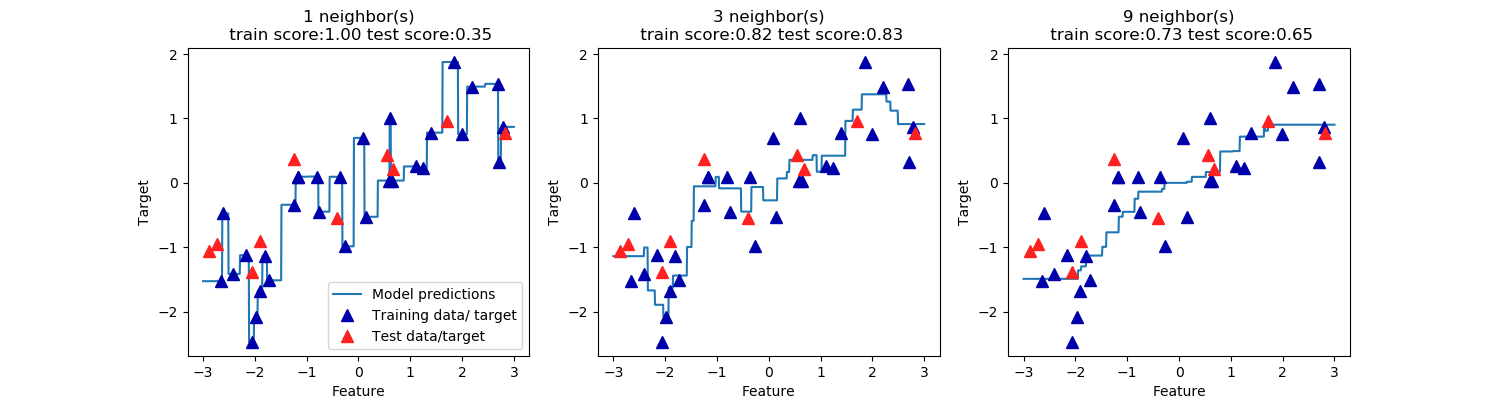


 应用于测试集:

1. **from** sklearn.neighbors **import** KNeighborsRegressor
2. **from** sklearn.model\_selection **import** train\_test\_split
3. **import** mglearn
5. X,y =  mglearn.datasets.make\_wave(n\_samples=40)
6. # split the wave dataset into a training and a test set
7. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)
9. # instantiate the model and set the num. of neighbors to 3
10. reg =  KNeighborsRegressor(n\_neighbors=3)
12. # fit the model using the training data and traning targets
13. reg.fit(X\_train, y\_train)
14. **print**("Test set predictions:\{}".format(reg.predict(X\_test)))
15. **print**("Test set R^2:{:.2f}".format(reg.score(X\_test,y\_test)))

评估KNN回归算法.

1. KNN回归算法中不同的k值的结果对比.
2. **from** sklearn.neighbors **import** KNeighborsRegressor
3. **from** sklearn.model\_selection **import** train\_test\_split
4. **import** mglearn
5. **import** numpy as np
6. **import** matplotlib.pyplot as plt
8. X,y =  mglearn.datasets.make\_wave(n\_samples=40)
9. # split the wave dataset into a training and a test set
10. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)
12. # instantiate the model and set the num. of neighbors to 3
14. fig, axes = plt.subplots(1,3,figsize=(15,4))
15. # create 1,000 data points, evenly spaced between -3 and 3
16. line = np.linspace(-3,3,1000).reshape(-1,1)
17. **for** n\_neighbors, ax **in** zip([1,3,9], axes):
18. # make predicitions using 1,3,or 9 neighbors
19. reg =  KNeighborsRegressor(n\_neighbors=n\_neighbors)
20. reg.fit(X\_train, y\_train)
21. ax.plot(line, reg.predict(line))
22. ax.plot(X\_train, y\_train,'^', c=mglearn.cm2(0), markersize=8)
23. ax.plot(X\_test, y\_test,'^', c=mglearn.cm2(1), markersize=8)
25. ax.set\_title(
26. "{} neighbor(s)\n train score:{:.2f} test score:{:.2f}".format(
27. n\_neighbors, reg.score(X\_train,y\_train),
28. reg.score(X\_test,y\_test)))
29. ax.set\_xlabel("Feature")
30. ax.set\_ylabel("Target")
31. axes[0].legend(["Model predictions", "Training data/ target", "Test data/target"], loc="best")
32. plt.show()



小结:

当使用k=1,训练集中每个点都对预测结果有很大的影响,故预测结果不稳定

当使用更大的k值, 则会导致对训练集的欠拟合.

1. 总结
   * + 1. 分类算法和回归算法的区别
       2. KNN分类算法(KNeighborsClassifier)
       3. KNN回归算法(KNeighborsRegressor模块).

1.在KNN分类中,合理地选择邻居的数目(k)和测量数据点之间距离的方式.