# 简单机器学习入门教程：用Python解决简单的水果分类问题

[2018年1月4日2018年6月11日](http://www.atyun.com/14092.html) by [xiaoshan.xiang](http://www.atyun.com/author/xiaoshan-xiang) / IMG_2562465 [0](http://www.atyun.com/14092.html" \l "respond)



在这篇机器学习入门教程中，我们将使用Python中最流行的机器学习工具scikit- learn,在Python中实现几种机器学习算法。使用简单的数据集来训练分类器区分不同类型的水果。

这篇文章的目的是识别出最适合当前问题的机器学习算法。因此，我们要比较不同的算法，选择性能最好的算法。让我们开始吧!

### ****数据****

水果数据集由爱丁堡大学的Iain Murray博士创建。他买了几十个不同种类的橘子、柠檬和苹果，并把它们的尺寸记录在一张桌子上。密歇根大学的教授们对水果数据进行了些微的格式化，可以从这里下载。

下载地址：<https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/fruit_data_with_colors.txt>

让我们先看一看数据的前几行。

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | %matplotlib inline |

|  |  |
| --- | --- |
| 2 | import pandas as pd |

|  |  |
| --- | --- |
| 3 | import matplotlib.pyplot as plt |

|  |  |
| --- | --- |
| 4 | fruits = pd.read\_table('fruit\_data\_with\_colors.txt') |

|  |  |
| --- | --- |
| 5 | fruits.head() |

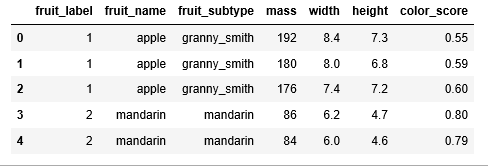


图1

数据集的每一行表示一个水果块，它由表中的几个特征表示。

在数据集中有59个水果和7个特征:

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | print(fruits.shape) |

(59, 7)

在数据集中有四种水果:

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | print(fruits['fruit\_name'].unique()) |

[“苹果”柑橘”“橙子”“柠檬”]

除了柑橘，数据是相当平衡的。我们只好接着进行下一步。

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | print(fruits.groupby('fruit\_name').size()) |

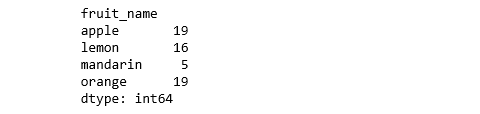


图2

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | import seaborn as sns |

|  |  |
| --- | --- |
| 2 | sns.countplot(fruits['fruit\_name'],label="Count") |

|  |  |
| --- | --- |
| 3 | plt.show() |

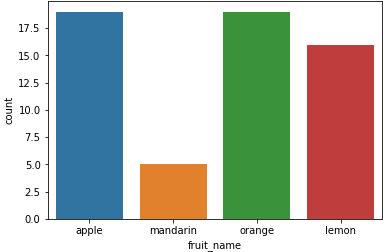


图3

### ****可视化****

* 每个数字变量的箱线图将使我们更清楚地了解输入变量的分布:

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | fruits.drop('fruit\_label', axis=1).plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False, figsize=(9,9), |

|  |  |
| --- | --- |
| 2 | title='Box Plot for each input variable') |

|  |  |
| --- | --- |
| 3 | plt.savefig('fruits\_box') |

|  |  |
| --- | --- |
| 4 | plt.show() |

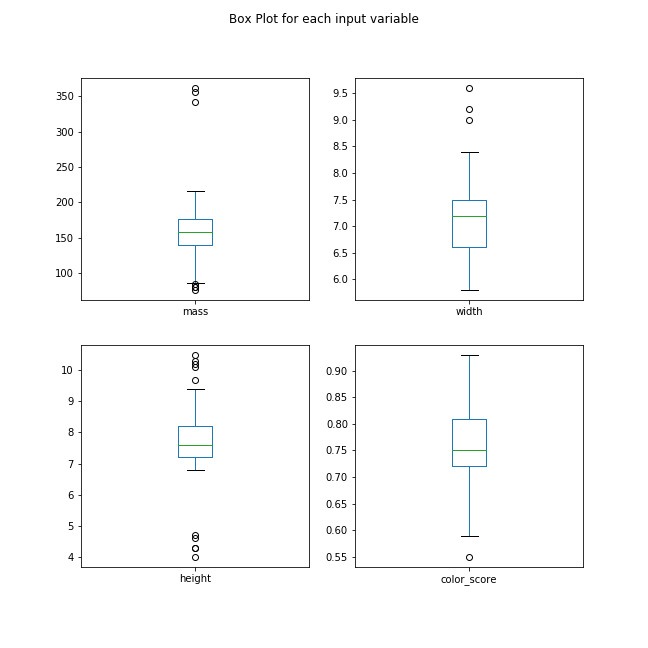


图4

* 看起来颜色分值近似于高斯分布。

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | import pylab as pl |

|  |  |
| --- | --- |
| 2 | fruits.drop('fruit\_label' ,axis=1).hist(bins=30, figsize=(9,9)) |

|  |  |
| --- | --- |
| 3 | pl.suptitle("Histogram for each numeric input variable") |

|  |  |
| --- | --- |
| 4 | plt.savefig('fruits\_hist') |

|  |  |
| --- | --- |
| 5 | plt.show() |

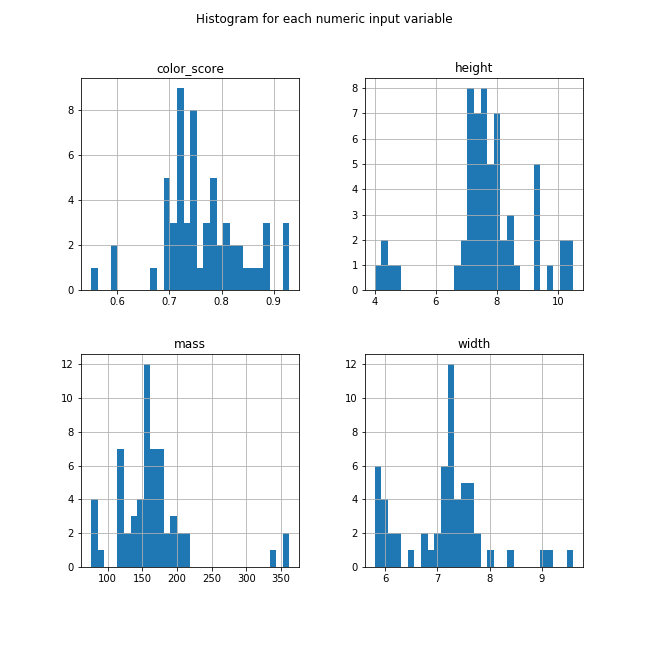


图5

* 一些成对的属性是相关的(质量和宽度)。这表明了高度的相关性和可预测的关系。

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from pandas.tools.plotting import scatter\_matrix |

|  |  |
| --- | --- |
| 2 | from matplotlib import cm |

|  |  |
| --- | --- |
| 3 | feature\_names = ['mass', 'width', 'height', 'color\_score'] |

|  |  |
| --- | --- |
| 4 | X = fruits[feature\_names] |

|  |  |
| --- | --- |
| 5 | y = fruits['fruit\_label'] |

|  |  |
| --- | --- |
| 6 | cmap = cm.get\_cmap('gnuplot') |

|  |  |
| --- | --- |
| 7 | scatter = pd.scatter\_matrix(X, c = y, marker = 'o', s=40, hist\_kwds={'bins':15}, figsize=(9,9), cmap = cmap) |

|  |  |
| --- | --- |
| 8 | plt.suptitle('Scatter-matrix for each input variable') |

|  |  |
| --- | --- |
| 9 | plt.savefig('fruits\_scatter\_matrix') |

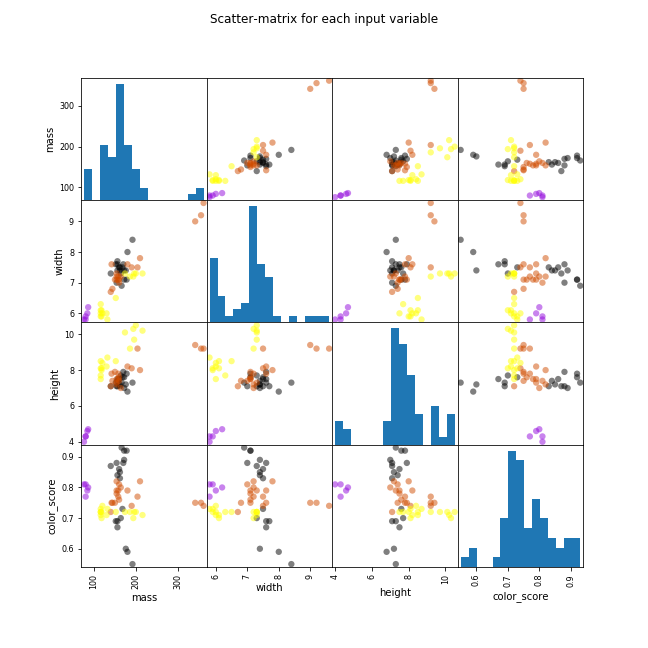


图6

### ****统计摘要****

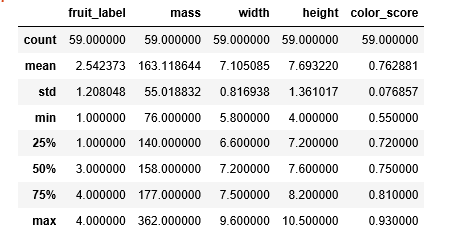


图7

我们可以看到数值没有相同的缩放比例。我们需要将缩放比例扩展应用到我们为训练集计算的测试集上。

### ****创建训练和测试集，并应用缩放比例****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.model\_selection import train\_test\_split |

|  |  |
| --- | --- |
| 2 | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0) |

|  |  |
| --- | --- |
| 3 | from sklearn.preprocessing import MinMaxScaler |

|  |  |
| --- | --- |
| 4 | scaler = MinMaxScaler() |

|  |  |
| --- | --- |
| 5 | X\_train = scaler.fit\_transform(X\_train) |

|  |  |
| --- | --- |
| 6 | X\_test = scaler.transform(X\_test) |

### ****构建模型****

****逻辑回归****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.linear\_model import LogisticRegression |

|  |  |
| --- | --- |
| 2 | logreg = LogisticRegression() |

|  |  |
| --- | --- |
| 3 | logreg.fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 4 | print('Accuracy of Logistic regression classifier on training set: {:.2f}' |

|  |  |
| --- | --- |
| 5 | .format(logreg.score(X\_train, y\_train))) |

|  |  |
| --- | --- |
| 6 | print('Accuracy of Logistic regression classifier on test set: {:.2f}' |

|  |  |
| --- | --- |
| 7 | .format(logreg.score(X\_test, y\_test))) |

训练集中逻辑回归分类器的精确度:0.70

测试集中逻辑回归分类器的精确度:0.40

****决策树****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.tree import DecisionTreeClassifier |

|  |  |
| --- | --- |
| 2 | clf = DecisionTreeClassifier().fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 3 | print('Accuracy of Decision Tree classifier on training set: {:.2f}' |

|  |  |
| --- | --- |
| 4 | .format(clf.score(X\_train, y\_train))) |

|  |  |
| --- | --- |
| 5 | print('Accuracy of Decision Tree classifier on test set: {:.2f}' |

|  |  |
| --- | --- |
| 6 | .format(clf.score(X\_test, y\_test))) |

训练集中决策树分类器的精确度:1.00

测试集中决策树分类器的精确度:0.73

****K-Nearest Neighbors（K-NN ）****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.neighbors import KNeighborsClassifier |

|  |  |
| --- | --- |
| 2 | knn = KNeighborsClassifier() |

|  |  |
| --- | --- |
| 3 | knn.fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 4 | print('Accuracy of K-NN classifier on training set: {:.2f}' |

|  |  |
| --- | --- |
| 5 | .format(knn.score(X\_train, y\_train))) |

|  |  |
| --- | --- |
| 6 | print('Accuracy of K-NN classifier on test set: {:.2f}' |

|  |  |
| --- | --- |
| 7 | .format(knn.score(X\_test, y\_test))) |

训练集中K-NN 分类器的精确度:0.95

测试集中K-NN 分类器的精确度:1.00

****线性判别分析****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis |

|  |  |
| --- | --- |
| 2 | lda = LinearDiscriminantAnalysis() |

|  |  |
| --- | --- |
| 3 | lda.fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 4 | print('Accuracy of LDA classifier on training set: {:.2f}' |

|  |  |
| --- | --- |
| 5 | .format(lda.score(X\_train, y\_train))) |

|  |  |
| --- | --- |
| 6 | print('Accuracy of LDA classifier on test set: {:.2f}' |

|  |  |
| --- | --- |
| 7 | .format(lda.score(X\_test, y\_test))) |

训练集中LDA分类器的精确度:0.86

测试集中LDA分类器的精确度:0.67

****高斯朴素贝叶斯****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.naive\_bayes import GaussianNB |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 | gnb = GaussianNB() |

|  |  |
| --- | --- |
| 4 | gnb.fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 5 | print('Accuracy of GNB classifier on training set: {:.2f}' |

|  |  |
| --- | --- |
| 6 | .format(gnb.score(X\_train, y\_train))) |

|  |  |
| --- | --- |
| 7 | print('Accuracy of GNB classifier on test set: {:.2f}' |

|  |  |
| --- | --- |
| 8 | .format(gnb.score(X\_test, y\_test))) |

训练集中GNB分类器的精确度:0.86

测试集中GNB分类器的精确度:0.67

****支持向量机****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.svm import SVC |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 | svm = SVC() |

|  |  |
| --- | --- |
| 4 | svm.fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 5 | print('Accuracy of SVM classifier on training set: {:.2f}' |

|  |  |
| --- | --- |
| 6 | .format(svm.score(X\_train, y\_train))) |

|  |  |
| --- | --- |
| 7 | print('Accuracy of SVM classifier on test set: {:.2f}' |

|  |  |
| --- | --- |
| 8 | .format(svm.score(X\_test, y\_test))) |

训练集中SVM分类器的精确度:0.61

测试集中SVM分类器的精确度:0.33

KNN算法是我们尝试过的最精确的模型。混淆矩阵提供了在测试集上没有错误的指示。但是，测试集非常小。

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 1 | from sklearn.metrics import classification\_report |

|  |  |
| --- | --- |
| 2 | from sklearn.metrics import confusion\_matrix |

|  |  |
| --- | --- |
| 3 | pred = knn.predict(X\_test) |

|  |  |
| --- | --- |
| 4 | print(confusion\_matrix(y\_test, pred)) |

|  |  |
| --- | --- |
| 5 | print(classification\_report(y\_test, pred)) |

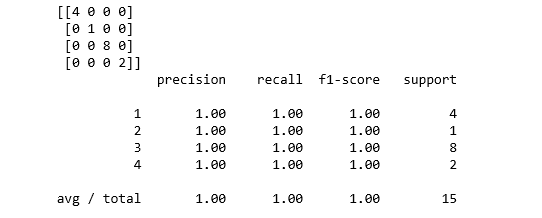


图8

### ****绘制k-NN分类器的决策边界****

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 01 | import matplotlib.cm as cm |

|  |  |
| --- | --- |
| 02 | from matplotlib.colors import ListedColormap, BoundaryNorm |

|  |  |
| --- | --- |
| 03 | import matplotlib.patches as mpatches |

|  |  |
| --- | --- |
| 04 | import matplotlib.patches as mpatches |

|  |  |
| --- | --- |
| 05 | X = fruits[['mass', 'width', 'height', 'color\_score']] |

|  |  |
| --- | --- |
| 06 | y = fruits['fruit\_label'] |

|  |  |
| --- | --- |
| 07 | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0) |

|  |  |
| --- | --- |
| 08 | def plot\_fruit\_knn(X, y, n\_neighbors, weights): |

|  |  |
| --- | --- |
| 09 | X\_mat = X[['height', 'width']].as\_matrix() |

|  |  |
| --- | --- |
| 10 | y\_mat = y.as\_matrix() |

|  |  |
| --- | --- |
| 11 | # Create color maps |

|  |  |
| --- | --- |
| 12 | cmap\_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF','#AFAFAF']) |

|  |  |
| --- | --- |
| 13 | cmap\_bold  = ListedColormap(['#FF0000', '#00FF00', '#0000FF','#AFAFAF']) |

|  |  |
| --- | --- |
| 14 |  |

|  |  |
| --- | --- |
| 15 | clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights) |

|  |  |
| --- | --- |
| 16 | clf.fit(X\_mat, y\_mat) |

|  |  |
| --- | --- |
| 17 | # Plot the decision boundary by assigning a color in the color map |

|  |  |
| --- | --- |
| 18 | # to each mesh point. |

|  |  |
| --- | --- |
| 19 |  |

|  |  |
| --- | --- |
| 20 | mesh\_step\_size = .01  # step size in the mesh |

|  |  |
| --- | --- |
| 21 | plot\_symbol\_size = 50 |

|  |  |
| --- | --- |
| 22 |  |

|  |  |
| --- | --- |
| 23 | x\_min, x\_max = X\_mat[:, 0].min() - 1, X\_mat[:, 0].max() + 1 |

|  |  |
| --- | --- |
| 24 | y\_min, y\_max = X\_mat[:, 1].min() - 1, X\_mat[:, 1].max() + 1 |

|  |  |
| --- | --- |
| 25 | xx, yy = np.meshgrid(np.arange(x\_min, x\_max, mesh\_step\_size), |

|  |  |
| --- | --- |
| 26 | np.arange(y\_min, y\_max, mesh\_step\_size)) |

|  |  |
| --- | --- |
| 27 | Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()]) |

|  |  |
| --- | --- |
| 28 | # Put the result into a color plot |

|  |  |
| --- | --- |
| 29 | Z = Z.reshape(xx.shape) |

|  |  |
| --- | --- |
| 30 | plt.figure() |

|  |  |
| --- | --- |
| 31 | plt.pcolormesh(xx, yy, Z, cmap=cmap\_light) |

|  |  |
| --- | --- |
| 32 | # Plot training points |

|  |  |
| --- | --- |
| 33 | plt.scatter(X\_mat[:, 0], X\_mat[:, 1], s=plot\_symbol\_size, c=y, cmap=cmap\_bold, edgecolor = 'black') |

|  |  |
| --- | --- |
| 34 | plt.xlim(xx.min(), xx.max()) |

|  |  |
| --- | --- |
| 35 | plt.ylim(yy.min(), yy.max()) |

|  |  |
| --- | --- |
| 36 | patch0 = mpatches.Patch(color='#FF0000', label='apple') |

|  |  |
| --- | --- |
| 37 | patch1 = mpatches.Patch(color='#00FF00', label='mandarin') |

|  |  |
| --- | --- |
| 38 | patch2 = mpatches.Patch(color='#0000FF', label='orange') |

|  |  |
| --- | --- |
| 39 | patch3 = mpatches.Patch(color='#AFAFAF', label='lemon') |

|  |  |
| --- | --- |
| 40 | plt.legend(handles=[patch0, patch1, patch2, patch3]) |

|  |  |
| --- | --- |
| 41 | plt.xlabel('height (cm)') |

|  |  |
| --- | --- |
| 42 | plt.ylabel('width (cm)') |

|  |  |
| --- | --- |
| 43 | plt.title("4-Class classification (k = %i, weights = '%s')" |

|  |  |
| --- | --- |
| 44 | % (n\_neighbors, weights)) |

|  |  |
| --- | --- |
| 45 | plt.show() |

|  |  |
| --- | --- |
| 46 | plot\_fruit\_knn(X\_train, y\_train, 5, 'uniform') |

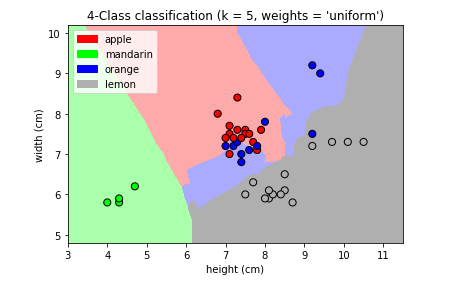


图9

[view source](http://www.atyun.com/14092.html" \l "viewSource" \o "view source)

|  |  |
| --- | --- |
| 01 | k\_range = range(1, 20) |

|  |  |
| --- | --- |
| 02 | scores = [] |

|  |  |
| --- | --- |
| 03 |  |

|  |  |
| --- | --- |
| 04 | for k in k\_range: |

|  |  |
| --- | --- |
| 05 | knn = KNeighborsClassifier(n\_neighbors = k) |

|  |  |
| --- | --- |
| 06 | knn.fit(X\_train, y\_train) |

|  |  |
| --- | --- |
| 07 | scores.append(knn.score(X\_test, y\_test)) |

|  |  |
| --- | --- |
| 08 | plt.figure() |

|  |  |
| --- | --- |
| 09 | plt.xlabel('k') |

|  |  |
| --- | --- |
| 10 | plt.ylabel('accuracy') |

|  |  |
| --- | --- |
| 11 | plt.scatter(k\_range, scores) |

|  |  |
| --- | --- |
| 12 | plt.xticks([0,5,10,15,20]) |

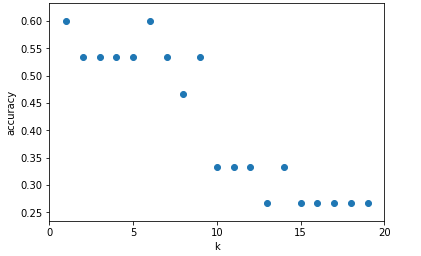


图10

对于这个特定的数据集，当k = 5时，我们获得了最高精确度。

### ****结语****

在这篇文章中，我们关注的是预测的准确度。我们的目标是学习一个具有良好泛化性能的模型。这样的模型使预测准确度最大化。通过比较不同的算法，我们确定了最适合当前问题的机器学习算法(即水果类型分类)。