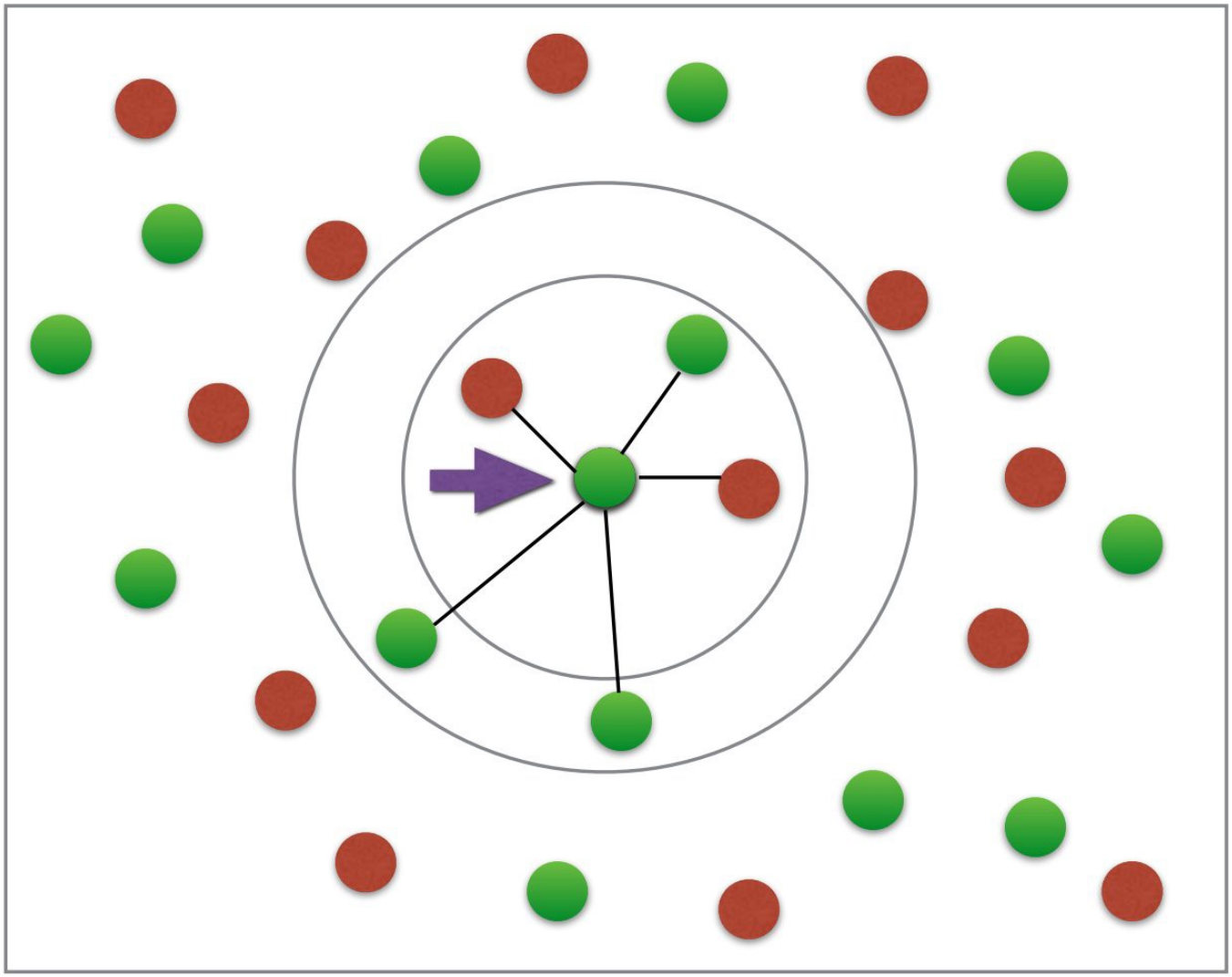
https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a

Building a k-Nearest-Neighbors (k-NN) Model with Scikit-learn

Eijaz Allibhai Follow
Sep 27, 2018 · 9 min read ★



k-NN (Image credit)

k-Nearest-Neighbors (k-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled. A supervised learning model takes in a set of input objects and output values. The model then trains on that data to learn how to map the inputs to the desired output so it can learn to make predictions on unseen data.

k-NN models work by taking a data point and looking at the 'k' closest labeled data points. The data point is then assigned the label of the majority of the 'k' closest points.

For example, if k = 5, and 3 of points are 'green' and 2 are 'red', then the data point in question would be labeled 'green', since 'green' is the majority (as shown in the above graph).

Scikit-learn is a machine learning library for Python. In this tutorial, we will build a k-NN model using Scikit-learn to predict whether or not a patient has diabetes.

Reading in the training data

For our k-NN model, the first step is to read in the data we will use as input. For this example, we are using the diabetes dataset. To start, we will use Pandas to read in the data. I will not go into detail on Pandas, but it is a library you should become familiar with if you're looking to dive further into data science and machine learning.

```
import pandas as pd

#read in the data using pandas
df = pd.read_csv('data/diabetes_data.csv')

#check data has been read in properly
df.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

4	0	137	40	35	168	43.1	2.288	33	1

Next, let's see how much data we have. We will call the 'shape' function on our dataframe to see how many rows and columns there are in our data. The rows indicate the number of patients and the columns indicate the number of features (age, weight, etc.) in the dataset for each patient.

行代表病人的数量 列代表特征值

```
#check number of rows and columns in dataset
df.shape
```

```
(768, 9)
```

We can see that we have 768 rows of data (potential diabetes patients) and 9 columns (8 input features and 1 target output).

将数据集拆分为输入和目标

Split up the dataset into inputs and targets

Now let's split up our dataset into inputs (X) and our target (y). Our input will be every column except 'diabetes' because 'diabetes' is what we will be attempting to predict. Therefore, 'diabetes' will be our target.

糖尿病

We will use pandas 'drop' function to drop the column 'diabetes' from our dataframe and store it in the variable 'X'. This will be our input.

```
#create a dataframe with all training data except the target column
X = df.drop(columns=['diabetes'])

#check that the target variable has been removed
X.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age
0	6	148	72	35	0	33.6	0.627	50

1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

We will insert the 'diabetes' column of our dataset into our target variable (y).

```
#separate target values
y = df['diabetes'].values

#view target values
y[0:5]
```

```
array([1, 0, 1, 0, 1])
```

Split the dataset into train and test data

Now we will split the dataset into into training data and testing data. The training data is the data that the model will learn from. The testing data is the data we will use to see how well the model performs on unseen data.

Scikit-learn has a function we can use called 'train_test_split' that makes it easy for us to split our dataset into training and testing data.

```
from sklearn.model_selection import train_test_split

#split dataset into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)
```

'train_test_split' takes in 5 parameters. The first two parameters are the input and target data we split up earlier. Next, we will set 'test_size' to 0.2. This means that 20% of all the

data will be used for testing, which leaves 80% of the data as training data for the model to learn from. Setting 'random_state' to 1 ensures that we get the same split each time so we can reproduce our results.

将“ stratify ”设置为y将使我们的训练拆分代表y变量中每个值的比例

Setting 'stratify' to y makes our training split represent the proportion of each value in the y variable. For example, in our dataset, if 25% of patients have diabetes and 75% don't have diabetes, setting 'stratify' to y will ensure that the random split has 25% of patients with diabetes and 75% of patients without diabetes.

将确保随机分割

Building and training the model

Next, we have to build the model. Here is the code:

```
from sklearn.neighbors import KNeighborsClassifier

# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 3)

# Fit the classifier to the data
knn.fit(X_train,y_train)
```

First, we will create a new k-NN classifier and set 'n_neighbors' to 3. To recap, this means that if at least 2 out of the 3 nearest points to an new data point are patients without diabetes, then the new data point will be labeled as 'no diabetes', and vice versa. In other words, a new data point is labeled with by majority from the 3 nearest points.

一个新的数据点将被标记为最接近的3个点

We have set 'n_neighbors' to 3 as a starting point. We will go into more detail below on how to better select a value for 'n_neighbors' so that the model can improve its performance.

Next, we need to train the model. In order to train our new model, we will use the 'fit' function and pass in our training data as parameters to fit our model to the training data.

将使用“ 拟合 ”功能并将训练数据作为参数传递，以使我们的模型适合训练数据

Testing the model

Once the model is trained, we can use the 'predict' function on our model to make predictions on our test data. As seen when inspecting 'y' earlier, 0 indicates that the

认为 显示；表示

patient does not have diabetes and 1 indicates that the patient does have diabetes. To save space, we will only show print the first 5 predictions of our test set.

```
#show first 5 model predictions on the test data
knn.predict(X_test)[0:5]
```

```
array([0, 0, 0, 0, 1])
```

We can see that the model predicted 'no diabetes' for the first 4 patients in the test set and 'has diabetes' for the 5th patient.

Now let's see how our accurate our model is on the full test set. To do this, we will use the 'score' function and pass in our test input and target data to see how well our model predictions match up to the actual results.

得分功能

看看我们的模型预测与实际结果有多吻合

```
#check accuracy of our model on the test data
knn.score(X_test, y_test)
```

```
0.6688311688311688
```

Our model has an accuracy of approximately 66.88%. It's a good start, but we will see how we can increase model performance below.

Congrats! You have now built an amazing k-NN model!

交叉验证

K代表的意思是分为几组

k-Fold Cross-Validation

数据集随机分成 K 组

Cross-validation is when the dataset is randomly split up into 'k' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group as been used as the test set.

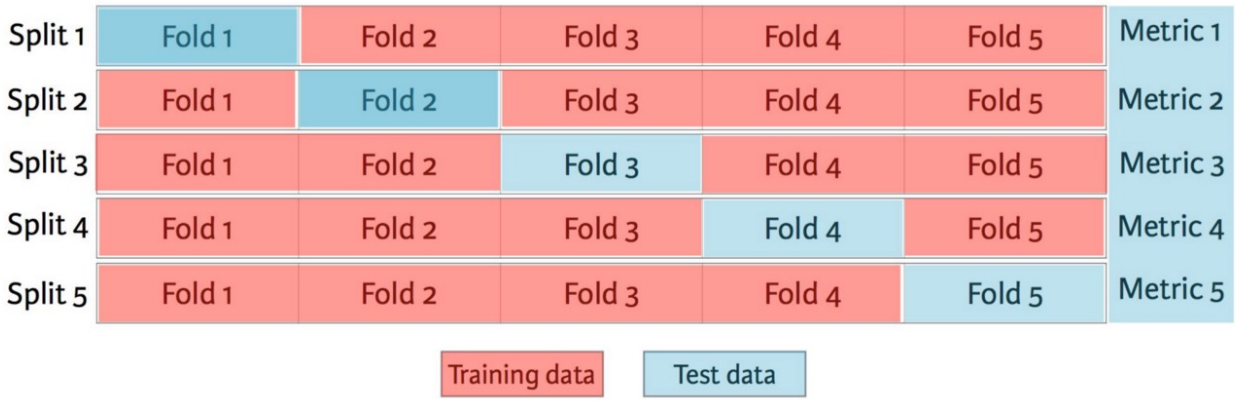
测试集

其余的用作训练集

然后重复这个过程，直到使用唯一的组作为测试集

For example, for 5-fold cross validation, the dataset would be split into 5 groups, and the model would be trained and tested 5 separate times so each group would get a chance to be the test set. This can be seen in the graph below.

被训练和测试5次



5-fold cross validation (image credit)

The train-test-split method we used in earlier is called 'holdout'. Cross-validation is better than using the holdout method because the holdout method score is dependent on how the data is split into train and test sets. Cross-validation gives the model an opportunity to test on multiple splits so we can get a better idea on how the model will perform on unseen data.

In order to train and test our model using cross-validation, we will use the 'cross_val_score' function with a cross-validation value of 5. 'cross_val_score' takes in our k-NN model and our data as parameters. Then it splits our data into 5 groups and fits and scores our data 5 seperate times, recording the accuracy score in an array each time. We will save the accuracy scores in the 'cv_scores' variable.

参数

分别对数据进行5次评分

我们将准确性得分保存在“ cv_scores ”变量中

To find the average of the 5 scores, we will use numpy's mean function, passing in 'cv_score'. Numpy is a useful math library in Python.

```
from sklearn.model_selection import cross_val_score
import numpy as np

#create a new KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)

#train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)
```

```
#print each cv score (accuracy) and average them
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
[ 0.68181818  0.69480519  0.75324675  0.75163399  0.68627451]
cv_scores mean:0.7135557253204311
```

Using cross-validation, our mean score is about 71.36%. This is a more accurate representation of how our model will perform on unseen data than our earlier testing using the holdout method.

使用GridSearchCV的超调模型参数

Hypertuning model parameters using GridSearchCV

最初的

When built our initial k-NN model, we set the parameter 'n_neighbors' to 3 as a starting point with no real logic behind that choice.

这种选择背后没有真正的逻辑。

Hypertuning parameters is when you go through a process to find the optimal parameters for your model to improve accuracy. In our case, we will use GridSearchCV to find the optimal value for 'n_neighbors'.

超调参数找到您模型的最佳参数以提高准确性

GridSearchCV的工作原理是在我们指定的参数范围内多次训练我们的模型。这样，我们可以用每个参数来测试我们的模型，并找出最优值，以获得最佳的精度结果。

GridSearchCV works by training our model multiple times on a range of parameters that we specify. That way, we can test our model with each parameter and figure out the optimal values to get the best accuracy results.

For our model, we will specify a range of values for 'n_neighbors' in order to see which value works best for our model. To do this, we will create a dictionary, setting 'n_neighbors' as the key and using numpy to create an array of values from 1 to 24.

我们使用网格搜索的新模型将采用新的k-NN分类器，即param_grid和交叉验证值5，以便找到“ n_neighbors ”的最佳值.

Our new model using grid search will take in a new k-NN classifier, our param_grid and a cross-validation value of 5 in order to find the optimal value for 'n_neighbors'.

```
from sklearn.model_selection import GridSearchCV

#create new a knn model
knn2 = KNeighborsClassifier()

#create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': np.arange(1, 25)}
```

```
#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn2, param_grid, cv=5)

#fit model to data
knn_gscv.fit(X, y)
```

After training, we can check which of our values for 'n_neighbors' that we tested performed the best. To do this, we will call 'best_params_' on our model.

```
#check top performing n_neighbors value
knn_gscv.best_params_
```

```
{'n_neighbors': 14}
```

最佳值

We can see that 14 is the optimal value for 'n_neighbors'. We can use the 'best_score_' function to check the accuracy of our model when 'n_neighbors' is 14. 'best_score_' outputs the mean accuracy of the scores obtained through cross-validation.

best_score_ ' 输出通过交叉验证获得的分数的平均准确性

```
#check mean score for the top performing value of n_neighbors
knn_gscv.best_score_
```

```
0.7578125
```

By using grid search to find the optimal parameter for our model, we have improved our model accuracy by over 4%!

Thanks for reading! The GitHub repository for this tutorial (jupyter notebook and dataset) can be found here. 他的git 资源
https://github.com/eijaz1/k-NN-Tutorial-Scikit-learn

If you would like to keep updated on my machine learning content, follow me :)

Machine Learning Scikit Learn Artificial Intelligence AI Data Science

About Help Legal

Get the Medium app