K-NN (K-Nearest Neighbours) Classifier

AcadView

June 4, 2018

---

1 Overview

The KNN algorithm is a robust and versatile classifier that is often used as a benchmark for more complex classifiers such as Artificial Neural Networks (ANN) and Support Vector Machines (SVM). Despite its simplicity, KNN can outperform more powerful classifiers and is used in a variety of applications such as economic forecasting, data compression and genetics. For example, KNN was leveraged in a 2006 study of functional genomics for the assignment of genes based on their expression profiles.

2 What is k-Nearest Neighbors

The model for kNN is the entire training dataset. When a prediction is required for a unseen data instance, the kNN algorithm will search through the training dataset for the k-most similar instances. The prediction attribute of the most similar instances is summarized and returned as the prediction for the unseen instance.

The similarity measure is dependent on the type of data. For real-valued data, the Euclidean distance can be used. Other other types of data such as categorical or binary data, Hamming distance can be used.

3 How does k-Nearest Neighbors Work

The kNN algorithm is belongs to the family of instance-based, competitive learning and lazy learning algorithms.

Instance-based algorithms are those algorithms that model the problem using data instances (or rows) in order to make predictive decisions. The kNN algorithm is an extreme form of instance-based methods because all training observations are retained as part of the model.

It is a **competitive learning algorithm**, because it internally uses competition between

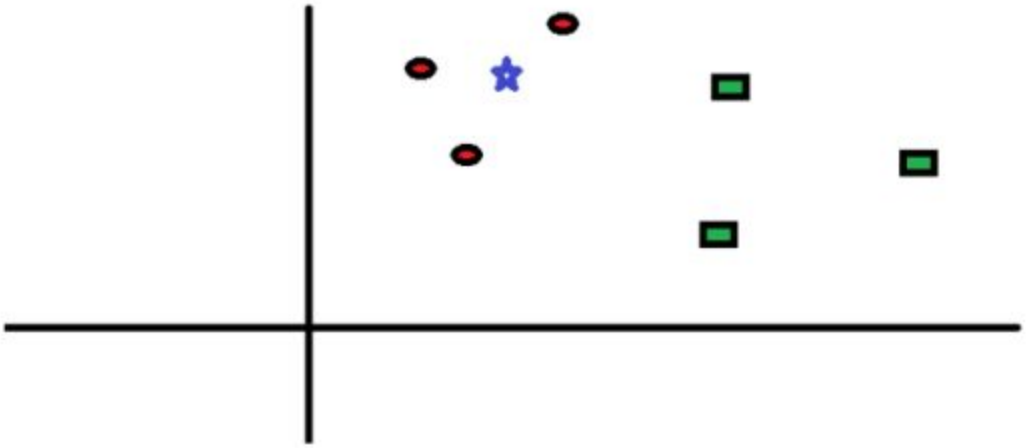
model elements (data instances) in order to make a predictive decision. The objective similarity measure between data instances causes each data instance to compete to win or be most similar to a given unseen data instance and contribute to a prediction.

Lazy learning refers to the fact that the algorithm does not build a model until the time that a prediction is required. It is lazy because it only does work at the last second. This has the benefit of only including data relevant to the unseen data, called a localized model. A disadvantage is that it can be computationally expensive to repeat the same or similar searches over larger training datasets.

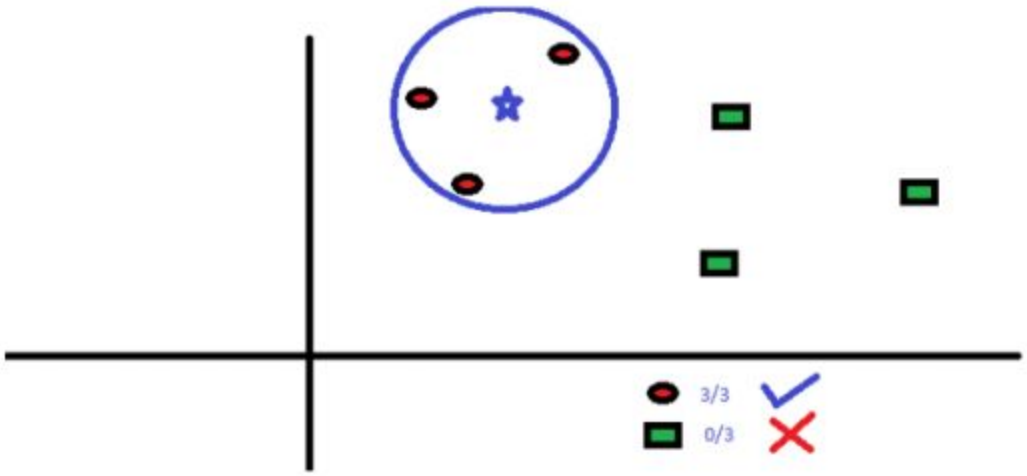
Finally, kNN is powerful because **it does not assume anything about the data,** other than a distance measure can be calculated consistently between any two instances. As such, it is called **non-parametric or non-linear** as it does not assume a functional form

.

Lets take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS) :



You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The K is KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three data points on the plane. Refer to following diagram for more details:



The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next we will understand what are the factors to be considered to conclude the best K.

4 k-NN in python through scikit-learn

4.1 Classify Flowers Using Measurements

The test problem we will be using in this tutorial is iris classification.

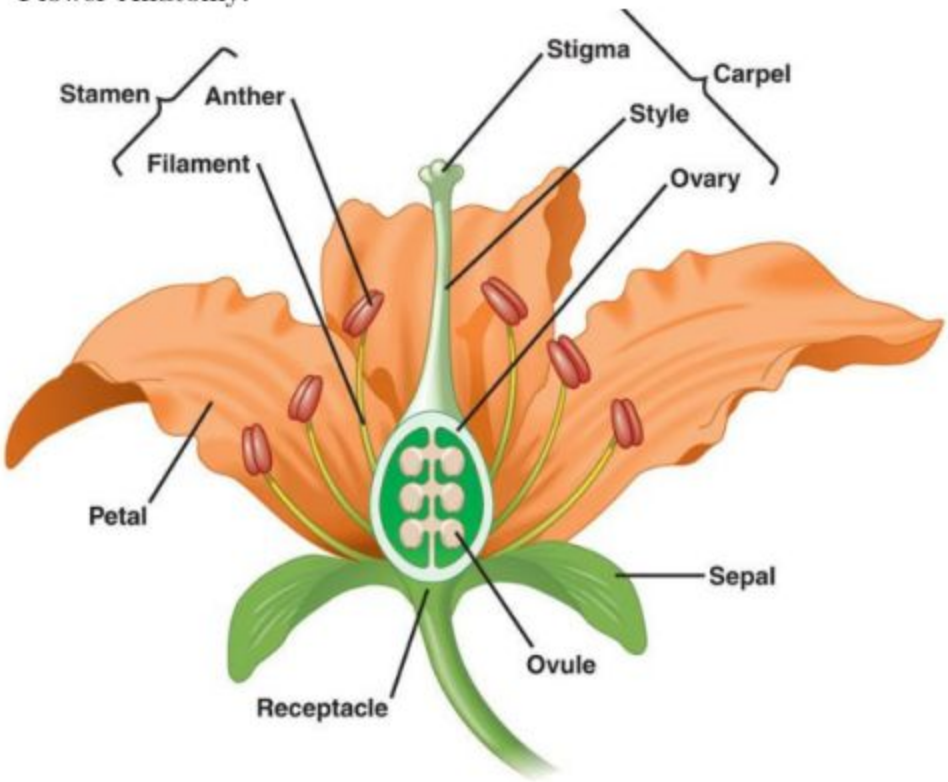
The problem is comprised of 150 observations of iris flowers from three different species. There are 4 measurements of given flowers: sepal length, sepal width, petal length and petal width, all in the same unit of centimeters. The predicted attribute is the species, which is one of setosa, versicolor or virginica.

It is a standard dataset where the species is known for all instances. As such we can split the data into training and test datasets and use the results to evaluate our algorithm implementation. Good classification accuracy on this problem is above 90% correct, typically 96% or better.

4.2 What is a Sepal?

No problem if you have no idea what a sepal was so here's some basic flower anatomy, you should find this picture helpful for relating petal and sepal length.

Flower Anatomy:



4.3 Overview of the data

Without further ado, let's see how KNN can be leveraged in Python for a classification problem. Were gonna head over to the UC Irvine Machine Learning Repository, an amazing source for a variety of free and interesting data sets.
https://archive.ics.uci.edu/ml/datasets/Iris

Go ahead and Download Data Folder ¿ iris.data and save it in the directory of your choice.

The first thing we need to do is load the data set. It is in CSV format without a header line so well use pandas read csv function.

```
# loading libraries
import pandas as pd

# define column names
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

# loading training data
df = pd.read_csv('path/iris.data.txt', header=None, names=names)
df.head()
```

Flower Anatomy:

```
# loading libraries
import numpy as np
from sklearn.cross_validation import train_test_split

# create design matrix X and target vector y
X = np.array(df.ix[:, 0:4])     # end index is exclusive
y = np.array(df['class'])       # another way of indexing a pandas df

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Now, it's time to get our hands wet. Well be using scikit-learn to train a KNN classifier and evaluate its performance on the data set using the 4 step modeling pattern:

- Import the learning algorithm
- Instantiate the model
- Learn the model
- Predict the response

scikit-learn requires that the design matrix X and target vector y be numpy arrays so lets oblige. Furthermore, we need to split our data into training and test sets. The following

code does just that.

Finally, following the above modeling pattern, we define our classifier, in this case KNN, fit it to our training data and evaluate its accuracy. We'll be using an arbitrary K but we will see later on how cross validation can be used to find its optimal value.

```
# loading library
from sklearn.neighbors import KNeighborsClassifier

# instantiate learning model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)

# fitting the model
knn.fit(X_train, y_train)

# predict the response
pred = knn.predict(X_test)

# evaluate accuracy
print accuracy_score(y_test, pred)
```

5 Pros and Cons of KNN

Pros

As you can already tell from the previous section, one of the most attractive features of the K-nearest neighbor algorithm is that is simple to understand and easy to implement. With zero to little training time, it can be a useful tool for off-the-bat analysis of some data set you are planning to run more complex algorithms on. Furthermore, KNN works just as easily with multiclass data sets whereas other algorithms are hardcoded for the binary setting. Finally, as we mentioned earlier, the non-parametric nature of KNN gives it an edge in certain settings where the data may be highly unusual.

Cons

One of the obvious drawbacks of the KNN algorithm is the computationally expensive testing phase which is impractical in industry settings. Note the rigid dichotomy between KNN and the more sophisticated Neural Network which has a lengthy training phase albeit a very fast testing phase. Furthermore, KNN can suffer from skewed class distributions. For example, if a certain class is very frequent in the training set, it will tend to dominate the majority voting of the new example (large number = more common). Finally, the accuracy of KNN can be severely degraded with high-dimension data because there is little difference between the nearest and farthest neighbor.