Classifying with k-Nearest Neighbors

Harvey Alférez, Ph.D.

Introduction

- Have you ever seen **movies categorized** into **genres**?

- What defines these **genres**, and who says which movie goes into what **genre**?

- The **movies** in **one genre** are **similar** but **based on what**?

- **What makes an action movie similar to another action movie and dissimilar to a romance movie?**

Introduction (Cont.)

- Do people **kiss** in action movies, and do people **kick** in romance movies?
  - Yes, but there's probably **more kissing in romance movies** and **more kicking in action movies**.

- Perhaps if you **measured kisses, kicks, and other things per movie**, you could automatically figure out what **genre** a **movie** belongs to.

Classifying with Distance Measurements

- We'll discuss our first machine-learning algorithm: **k-Nearest Neighbors (kNN)**!!!

- k-Nearest Neighbors is easy to grasp and very effective.

k-Nearest Neighbors

Pros: High accuracy, insensitive to outliers, no assumptions about data

Cons: Computationally expensive, requires a lot of memory

Works with: Numeric values, nominal values

- The kNN works like this: we have an existing **set of example data**, **our training set**.

- We have **labels** for all of this data—we know what class each piece of the data should fall into.

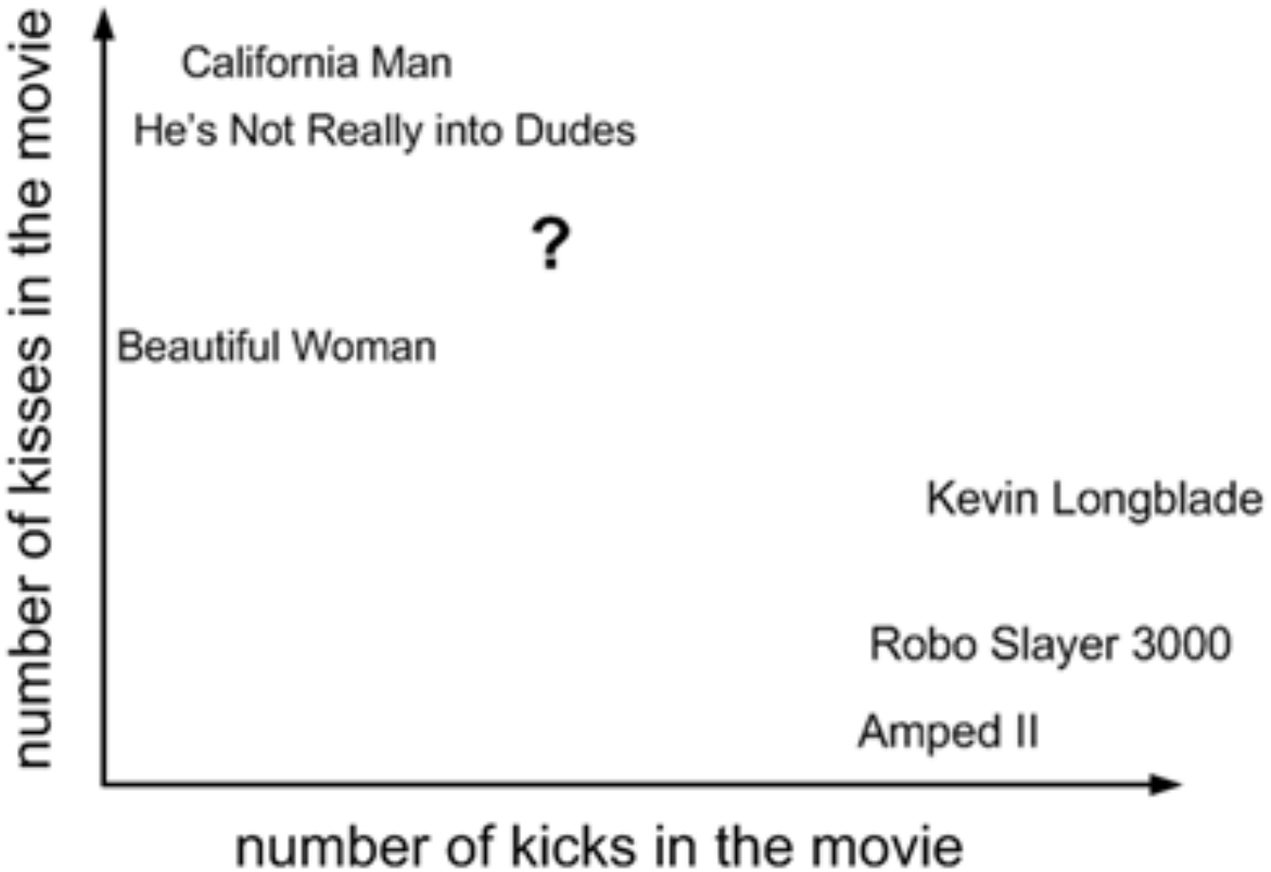
- When we're **given a new piece of data without a label**, we compare that new piece of data to the existing data, every piece of existing data.

- We then **take the most similar pieces of data** (the **nearest neighbors**) and look at their **labels**.

    - We look at the **top k most similar pieces of data from our known dataset**; this is where the k comes from. (k is an integer and it's usually less than 20.)

- Lastly, we take a **majority vote** from the k most similar pieces of data, and the majority is the new class we assign to the data we were asked to classify.

Classifying with Distance Measurements (Cont.)



Classifying with Distance Measurements (Cont.)

Table 2.1 **Movies with the number of kicks and number of kisses shown for each movie, along with our assessment of the movie type**

Movie title	# of kicks	# of kisses	Type of movie
California Man	3	104	Romance
He's Not Really into Dudes	2	100	Romance
Beautiful Woman	1	81	Romance
Kevin Longblade	101	10	Action
Robo Slayer 3000	99	5	Action
Amped II	98	2	Action
?	18	90	Unknown

Classifying with Distance Measurements (Cont.)

Movie title	Distance to movie "?"
California Man	20.5
He's Not Really into Dudes	18.7
Beautiful Woman	19.2
Kevin Longblade	115.3
Robo Slayer 3000	117.4
Amped II	118.9

Table 2.2 Distances between each movie and the unknown movie

k = 3

1. Prepare: Importing Data with Python

- kNN.py

```
from numpy import *
import operator

def createDataSet():
group = array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
labels = ['A','A','B','B']
return group, labels
```

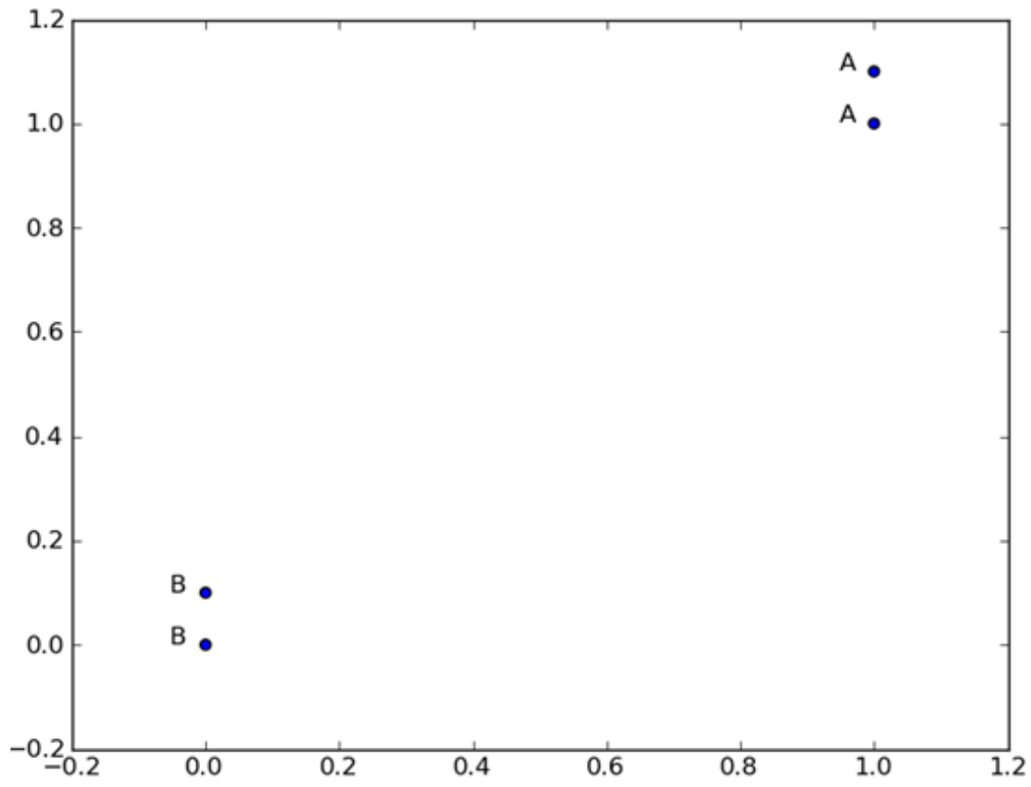


Figure 2.2 **The four data points of our very simple kNN example**

1. Prepare: Importing Data with Python

- python
- import kNN
- groups,labels = kNN.createDataSet()
- groups
- labels

2. Putting the kNN Classification Algorithm into Action

For every point in our dataset:
calculate the distance between inX and the current point
sort the distances in increasing order
take k items with lowest distances to inX
find the majority class among these items
return the majority class as our prediction for the class of inX

2. Putting the kNN Classification Algorithm into Action (Cont.)

The function `classify0()` takes four inputs: the input vector to classify called inX, our full matrix of training examples called dataSet, a vector of labels called labels, and, finally, k, the number of nearest neighbors to use in the voting. The labels vector should have as many elements in it as there are rows in the dataSet matrix. You calculate the distances (1) using the Euclidian distance where the distance between two vectors, xA and xB, with two elements, is given by

$$d = \sqrt{(xA_0 - xB_0)^2 + (xA_1 - xB_1)^2}$$

For example, the distance between points (0,0) and (1,2) is calculated by

$$\sqrt{(1-0)^2 + (2-0)^2}$$

If we are working with four features, the distance between points (1,0,0,1) and (7, 6, 9, 4) would be calculated by

$$\sqrt{(7-1)^2 + (6-0)^2 + (9-0)^2 + (4-1)^2}$$

2. Putting the kNN Classification Algorithm into Action (Cont.)

To predict the class, type the following text at the Python prompt:

```
>>> kNN.classify0([0,0], group, labels, 3)
```

The result should be B. Try to change the [0,0] entry to see how the answer changes.

Congratulations!

You just made your first classifier!

You can do a lot with this simple classifier. Things will only get easier from here on out.

How to Test a Classifier

- To test out a classifier, you **start** with **some known data**.
- **Add up the number of times the classifier was wrong / the total number of tests you gave it**.
- This will give you the **error rate**.
  - An **error rate of 0** means you have a **perfect classifier**
  - An **error rate of 1.0** means **the classifier is always wrong**.

Improving Matches from a Dating Site with kNN

- My friend Hellen has been using some online dating sites to find different people to go out with. She realized that despite the site's recommendations, she didn't like everyone she was matched with. After some introspection, she realized there were three types of people she went out with:

- People she didn't like
- People she liked in small doses
- People she liked in large doses

Improving Matches from a Dating Site with kNN (Cont.)

Hellen has asked us to help her filter future matches to categorize them.

Prepare: Parsing Data from a Text File

- datingTestSet.txt
  - 1,000 entries. A new sample is on each line. Hellen has recorded the following features:
    - Number of frequent flyer miles earned per year
    - Percentage of time spent playing video games
    - Liters of ice cream consumed per week
- **file2matrix:** This function takes a filename string and outputs two things: a matrix of training examples and a vector of class labels.

Prepare: Parsing Data from a Text File (Cont.)

Listing 2.2 Text record to NumPy parsing code

```
def file2matrix(filename):
    fr = open(filename)
    numberOfLines = len(fr.readlines())
    returnMat = zeros((numberOfLines,3))
    classLabelVector = []
    fr = open(filename)
    index = 0
    for line in fr.readlines():
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:3]
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector
```

1 **Get number of lines in file**

2 **Create NumPy matrix to return**

3 **Parse line to a list**

Prepare: Parsing Data from a Text File (Cont.)

- reload(kNN)
- datingDataMat,datingLabels = kNN.file2matrix('datingTestSet2.txt')
- datingDataMat
- datingLabels[0:20]

Prepare: Normalizing Numeric Values

- If you were to calculate the distance between person 3 and person 4 in table 2.3, you would have

$$\sqrt{(0-67)^2+(20{,}000-32{,}000)^2+(1.1-0.1)^2}$$

Table 2.3 Sample of data from improved results on a dating site

	Percentage of time spent playing video games	Number of frequent flyer miles earned per year	Liters of ice cream consumed weekly	Category
1	0.8	400	0.5	1
2	12	134,000	0.9	3
3	0	20,000	1.1	2
4	67	32,000	0.1	2

Prepare: Normalizing Numeric Values (Cont.)

- When dealing with values that lie in different ranges, it's common to **normalize** them.
- Common ranges to normalize them to are 0 to 1 or -1 to 1. To scale everything from 0 to 1, you need to apply the following formula:
  - *newValue = (oldValue-min)/(max-min)*

Prepare: Normalizing Numeric Values (Cont.)

```
>>> reload(kNN)
>>> normMat = kNN.autoNorm(datingDataMat)
>>> normMat
```

Use: Putting Together a Useful System

```
>>> kNN.classifyPerson()
percentage of time spent playing video games?10
frequent flier miles earned per year?10000
liters of ice cream consumed per year?0.5
You will probably like this person:  in small doses
```

For every point in our dataset:

calculate the distance between inX and the current point

sort the distances in increasing order

take k items with lowest distances to inX

find the majority class among these items

return the majority class as our prediction for the class of inX

Test: Testing the Classifier as a Whole program

- One way you can use the existing data is to take some portion, say 90%, to train the classifier.

- Then you'll take the remaining 10% to test the classifier and see how accurate it is.

Test: Testing the Classifier as a Whole program (Cont.)

- You can measure the performance of a classifier with the error rate.
  - In classification, the **error rate** is the number of misclassified pieces of data divided by the total number of data points tested.

- An **error rate of 0** means you have a perfect classifier, and an **error rate of 1.0** means the classifier is always wrong.

Test: Testing the Classifier as a Whole program (Cont.)

```
>>> kNN.datingClassTest()
```

http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[ 0.66666667  0.33333333]]
```