AI – Homework 4: Machine Learning Recipes

Matias Rietig (mjr9r) 11/29/2017

Summary of Training Method

I implemented a k-Nearest-Neighbors algorithm in Python as the training method for the recipes. It is very straightforward to implement in the language even without using machine learning libraries (the algorithm itself is only about 20 lines of code). It takes Python dictionaries, with the key being the label and the value being the features and calculates the Euclidean distance between all the values that were given to the function. It then sorts all data it iterated over by distances and returns the majority vote of all these.

Format of Input Data

Quite a big challenge was transforming the input data into an appropriate format that can be understood and used by the algorithm and this process takes up a lot of the actual code of the final submission. I ended up reading in the *ingredients.txt,* comparing it with every single row of the *training.csv* to transform said rows into arrays of Boolean values (i.e. 0 or 1) that are ordered the same way as the original *ingredients.txt.* It looks a bit like this:

Original ingredients.txt to Array:

[“boneless chop pork”, “dried oregano”, “lemon juice”, …, “farmer cheese”, “kefir”]

Compare with Recipe (2, „greek“):

[„minced garlic”, “dried oregano”, “red wine vinegar”, “olive oil”, “boneless chop pork”, lemon juice”]

Becomes:

[1 , 1 , 1 , …, 0 , 0]

(same order and number of entries as ingredients.txt)

The cuisines are converted to numbers from 0 to 19 and appended to the end of above array. This allows us to compare Boolean values instead of Strings and use them to calculate the distances.

Performance of Training Method

Various runs and observations of the accuracy and generalization error obtained in the different folds show that there is a considerable variance in both values. The accuracy can cover a range from around 40 percent to as much as 76% (probably in a fold that included mostly recipes from common cuisines in the training set).
Below are the results of two 6-fold-cross validation runs:

Fold#	Accuracy[1]	Generalization Error[2]
1	**52.2%**	**47.8%**
2	55.9%	44.1%
3	**75.6%**	**24.4%**
4	58.5%	41.5%
5	70.2%	29.8%
6	61.9%	38.1%
Avg.	**62.4%**	**37.6%**

Table 1 - 6-fold-Cross-Validation Run (A)

Fold#	Accuracy[1]	Generalization Error[2]
1	61.9%	38.1%
2	57.5%	42.4%
3	44.1%	55.9%
4	56.5%	43.5%
5	**62.5%**	**37.5%**
6	**46.8%**	**53.2%**
Avg.	**54.9%**	**45.1%**

Table 2 - 6-fold-Cross-Validation Run (B

Considering the training set, it is probably fair to say, that the algorithm does a already a decent job teaching the combination of ingredients per cuisine. The lowest observed generalization error in the two runs above is **24.4%** (which is definitely an outlier due to a specialized fold), the highest being **53.2%** which still implies an accuracy of roughly 47 percent, being much better than either just guessing the cuisine randomly (1 out of 20, being 5%) or predicting the most common cuisine (Italian, 324/1794, about 18%).

```
------------------------------
Accuracy:  0.5217391304347826
Generalization Error:  0.4782608695652174
------------------------------
Validating with Fold 2 ...
------------------------------
Accuracy:  0.5585284280936454
Generalization Error:  0.4414715719063545
------------------------------
Validating with Fold 3 ...
------------------------------
Accuracy:  0.7558528428093646
Generalization Error:  0.24414715719063546
------------------------------
Validating with Fold 4 ...
------------------------------
Accuracy:  0.5852842809364549
Generalization Error:  0.41471571906354515
------------------------------
Validating with Fold 5 ...
------------------------------
Accuracy:  0.7023411371237458
Generalization Error:  0.29765886287625415
------------------------------
Validating with Fold 6 ...
------------------------------
Accuracy:  0.6187290969899666
Generalization Error:  0.3812709030100334
------------------------------
```

```
Accuracy:  0.6187290969899666
Generalization Error:  0.3812709030100334
------------------------------
Validating with Fold 2 ...
------------------------------
Accuracy:  0.5752508361204013
Generalization Error:  0.42474916387959866
------------------------------
Validating with Fold 3 ...
------------------------------
Accuracy:  0.4414715719063545
Generalization Error:  0.5585284280936454
------------------------------
Validating with Fold 4 ...
------------------------------
Accuracy:  0.5652173913043478
Generalization Error:  0.43478260869565216
------------------------------
Validating with Fold 5 ...
------------------------------
Accuracy:  0.6254180602006689
Generalization Error:  0.3745819397993311
------------------------------
Validating with Fold 6 ...
------------------------------
Accuracy:  0.4682274247491639
Generalization Error:  0.5317725752508361
------------------------------
```

3 - Output of the two cross validation runs from above

The algorithm probably suffers from the Curse of Dimensionality as well, meaning that the huge array of ingredients that I set up for each row is more sensitive to outliers. Using another distance (like the cosine distance) or reducing the dimensions of the array might improve the algorithm's performance.

Time Requirements

One run of the whole code takes about 16 minutes to complete on the system it was tested on. Parsing the dataset and the ingredients takes about 20 seconds and each of the cross validation runs takes about 2.30 minutes to complete.

System used for testing:

CPU: Intel i7-4700MQ @2.40 GHz

RAM: 16GB

(I didn't implement a multi-thread algorithm so the computing time can potentially be optimized a lot!)

Remarks for running Submission Code

Some standard scientific libraries for Python are required to run the submitted code. These include numpy, math, and pandas. I used the Anaconda package and prompt to run the code.

---

1 – Accuracy is computed by dividing the correct results by the total number of results (comparing the majority vote to the actual label of the data)

2 – Generalization Error is computed by the following formula: $$E[err] \approx \frac{1}{N_{valid}} \sum_{n=1}^{N_{valid}} I(\hat{y}(x_n) \neq y_n)$$