UNSUPERVISED MACHINE LEARNING FOR TEXT CATEGORIZATION

(A Computational Linguistics Project)

QUESTION

How accurately can an efficient, unsupervised machine learning model categorize text files?

HYPOTHESIS

It would be able to separate both spoken language and technical documents from all other categories, because the former contains more common words and the latter contains more uncommon words.

WHAT IS UNSUPERVISED MACHINE LEARNING?

Unsupervised machine learning is a category of methods used by AI to learn to recognize patterns without using previously labeled data.

In supervised machine learning, the AI is given a training dataset which has been manually labeled and categorized, and it learns to categorize a new dataset based on the training data.

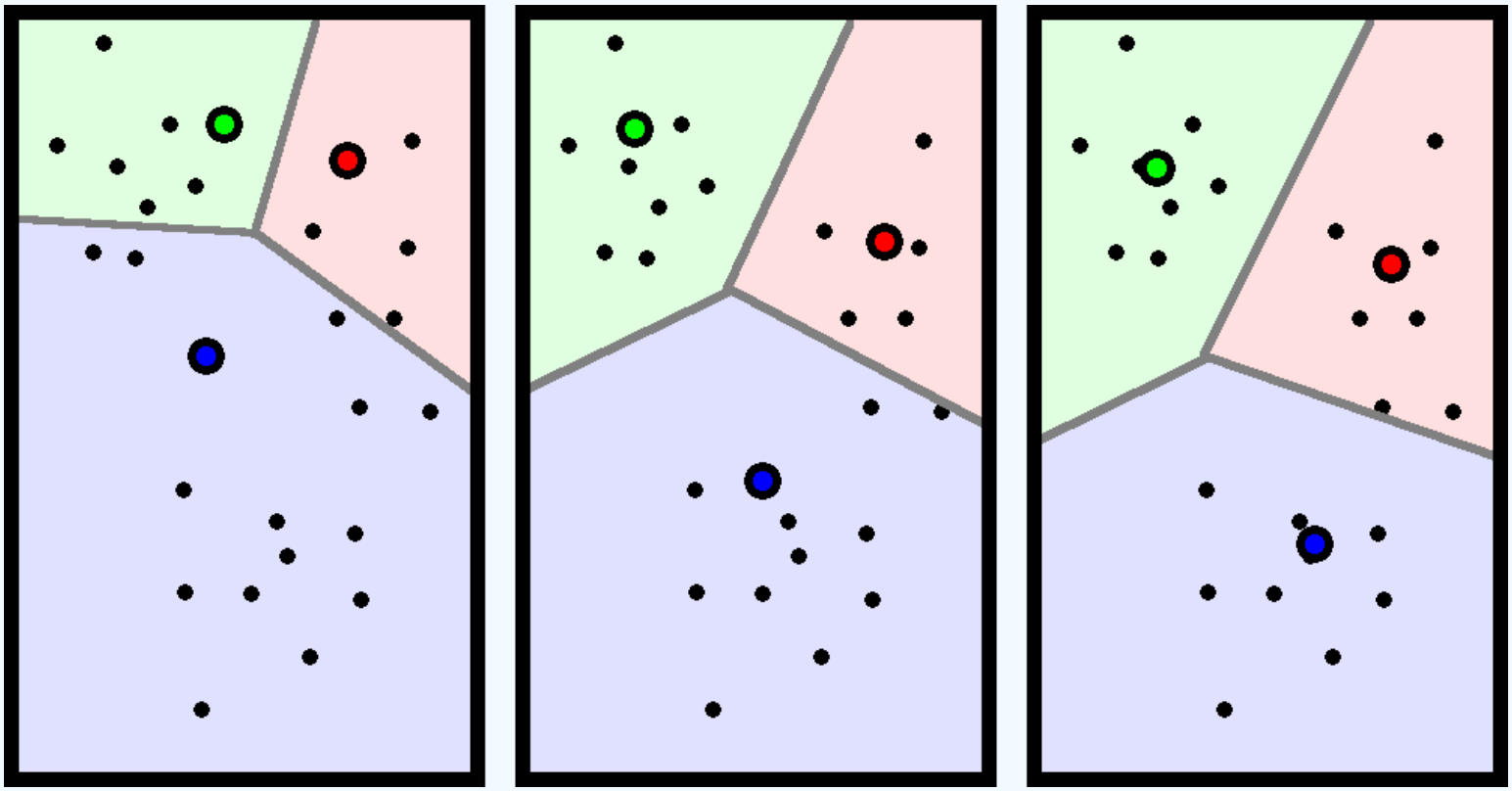
In semi-supervised machine learning, only some of the training data is labeled and the AI uses this labeled data to categorize the rest.

In unsupervised machine learning, no data is labeled, and the AI separates the data into clusters of similar unlabeled data points.

WHAT ARE DOCUMENT VECTORS?

In order to numerically cluster documents in a corpus, their word frequency information must first be converted to strings of numbers (vectors). These vectors represent the documents as points residing in a high-dimensional space. To create document vectors, a list of all words in the corpus is compiled, and the frequencies of all words in all documents are stored in a single array. Words which only occur in one document are removed, as they provide no useful information. The tf-idf values, or "term frequency-inverse document frequency" values, of the remaining words are calculated using the equation below and used as coordinates of the document vectors' terminal points. The initial points of all vectors are placed at the origin. These vectors can then be clustered with the k-means algorithm.

$$tf\ idf = \left(\frac{Word\ frequency\ in\ document}{Length\ of\ document}\right) \times \ln\left(\frac{Number\ of\ documents}{Number\ of\ documents\ containing\ word}\right)$$



An example of the K-means algorithm separating data points into 3 clusters. Here, the clustering occurs in 3 iterations. Black circles represent data points, colored circles represent centroids, and points in the same colored region are in the same cluster.

WHAT IS K-MEANS?

K-means is a clustering algorithm. It inputs a set of points and the desired number of clusters, k, into which these points are to be placed. It outputs labels for the points, which determine the cluster into which each point falls. K-means runs very quickly compared to other clustering algorithms, making it ideal for this project. This algorithm functions as follows:

1. Choose k random points, or centroids, in the vector space.
2. For every data point, find the closest centroid to that point and add it to that centroid's cluster.
3. Move every centroid to the average location of all of the points in its cluster.
4. Repeat steps 2 and 3 until the centroids stop moving.
5. Output the clusters.

MATERIALS AND METHODS

Materials

Software used:

- Python 2.7.12
- Python libraries: numpy, scipy, os, math, and time
- Emacs text editor
- Bash shell
- GNU utilities

Procedure:

1. Install software.
2. Download the Manually Annotated Sub-Corpus (MASC):
   http://anc.org/MASC/download/masc_500k_texts.zip
3. Execute the Python script (see image). It performs the following:
   - Using tf-idf, generates vectors which represent each document.
   - Using k-means, clusters the vectors with k-values ranging from 2 to 9.
   - Outputs the clusters and their corresponding values of k.

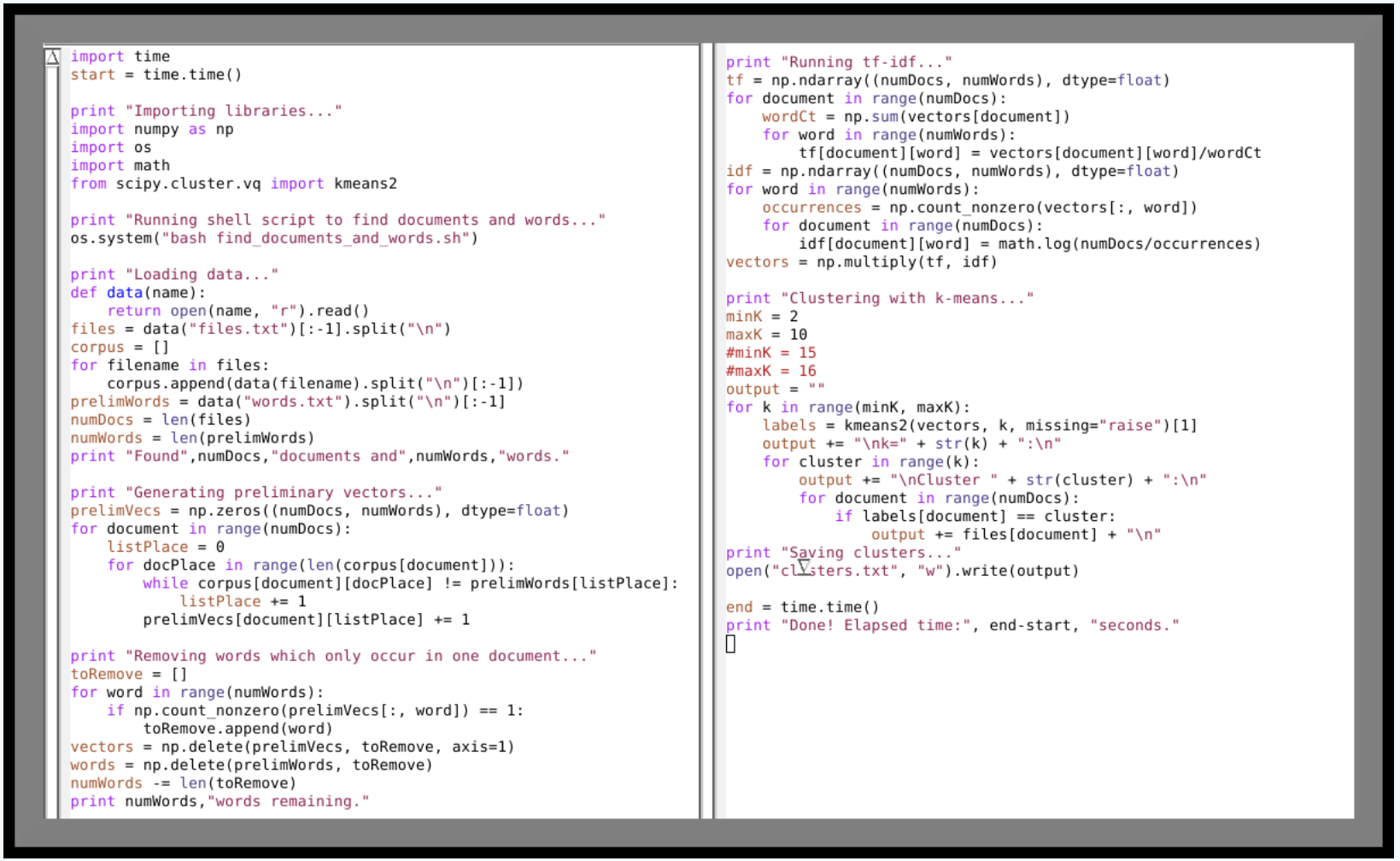
RESULTS

For all values of k between 2 and 9, k-means generated one large cluster and several smaller clusters. The smaller clusters were almost entirely composed of spam, emails, and newspaper articles. The algorithm separated these three categories reasonably well. Of the 80 emails, 96 spam messages, and 52 newspaper articles, 72 emails (90%), 63 spam messages (66%), and 36 newspaper articles (69%) were sorted into the smaller clusters.

RESULTS

	Spam/News/Email	Other Categories	Total
Smaller Clusters	171	2	173
Largest Cluster	57	160	217
Total	228	162	390

99% of the documents which were not spam, newspaper, or email were placed into the largest cluster (160/162), and 99% of the documents in the smaller clusters were spam, newspaper, or email (171/173). However, only 71% of the documents in these three categories were placed into the smaller clusters (171/288).



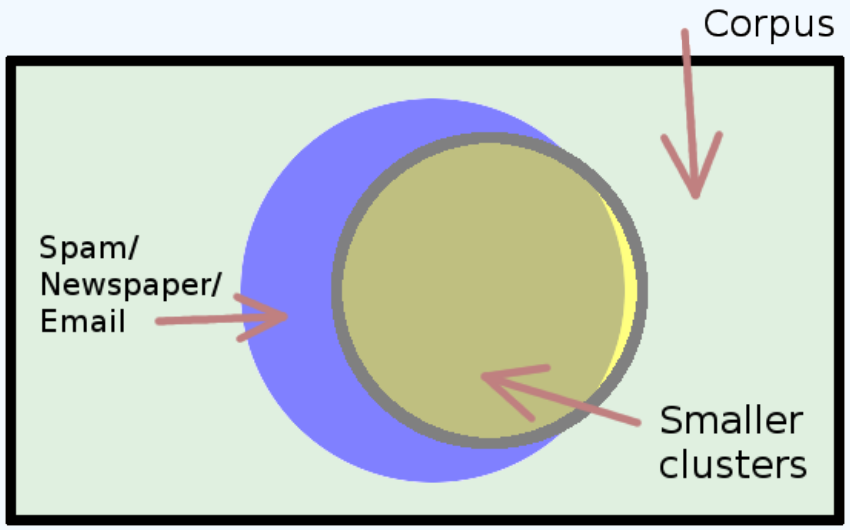
```
import time
start = time.time()

print "Importing libraries..."
import numpy as np
import os
import math
from scipy.cluster.vq import kmeans2

print "Running shell script to find documents and words..."
os.system("bash find_documents_and_words.sh")

print "Loading data..."
def data(name):
    return open(name, "r").read()
files = data("files.txt")[:-1].split("\n")
corpus = []
for filename in files:
    corpus.append(data(filename).split("\n")[:-1])
prelimWords = data("words.txt").split("\n")[:-1]
numDocs = len(files)
numWords = len(prelimWords)
print "Found",numDocs,"documents and",numWords,"words."

print "Generating preliminary vectors..."
prelimVecs = np.zeros((numDocs, numWords), dtype=float)
for document in range(numDocs):
    listPlace = 0
    for docPlace in range(len(corpus[document])):
        while corpus[document][docPlace] != prelimWords[listPlace]:
            listPlace += 1
        prelimVecs[document][listPlace] += 1

print "Removing words which only occur in one document..."
toRemove = []
for word in range(numWords):
    if np.count_nonzero(prelimVecs[:, word]) == 1:
        toRemove.append(word)
vectors = np.delete(prelimVecs, toRemove, axis=1)
words = np.delete(prelimWords, toRemove)
numWords -= len(toRemove)
print numWords,"words remaining."
```

```
print "Running tf-idf..."
tf = np.ndarray((numDocs, numWords), dtype=float)
for document in range(numDocs):
    wordCt = np.sum(vectors[document])
    for word in range(numWords):
        tf[document][word] = vectors[document][word]/wordCt
idf = np.ndarray((numDocs, numWords), dtype=float)
for word in range(numWords):
    occurrences = np.count_nonzero(vectors[:, word])
    for document in range(numDocs):
        idf[document][word] = math.log(numDocs/occurrences)
vectors = np.multiply(tf, idf)

print "Clustering with k-means..."
minK = 2
maxK = 10
#minK = 15
#maxK = 16
output = ""
for k in range(minK, maxK):
    labels = kmeans2(vectors, k, missing="raise")[1]
    output += "\nk=" + str(k) + ":\n"
    for cluster in range(k):
        output += "\nCluster " + str(cluster) + ":\n"
        for document in range(numDocs):
            if labels[document] == cluster:
                output += files[document] + "\n"
print "Saving clusters..."
open("clusters.txt", "w").write(output)

end = time.time()
print "Done! Elapsed time:", end-start, "seconds."
```

The Python code used to generate and cluster the document vectors.

CONCLUSION

The smaller clusters reliably contained documents from only three categories of the corpus, but not all documents in these categories were placed into the smaller clusters.



FURTHER RESEARCH

The language model used is the bag-of-words model, which only looks at occurrences of individual words. This could be extended to the n-grams model, which considers consecutive word pairs, triplets, and so on. "tf-idf" has many variants, and changing the variant may lead to more accurate classification. Lastly, k-means is not the only clustering algorithm used for this type of problem. Other algorithms, such as the expectation-maximization algorithm, can also be used in unsupervised machine learning and may, in some situations, yield better results.

ACKNOWLEDGMENTS

I would like to thank my father for helping with programming and formatting, Ms. Finlayson for providing feedback and guidance through the process, and Mrs. Dexter, my mentor.

Created with LaTeX on March 12, 2017.