

Lab 3: Nearest Neighbor Text Classification

Edward Hunter and K. Sree Harsha

1. Introduction

In this lab, we extend our work from lab 2 to experiment with Nearest Neighbor methods by following a similar method of customizing the `template_supervised.py` file with scikit-learn code. We omit discussion of the common code pieces that we explained in detail there and focus on the Nearest Neighbor code customizations and experiment with training and testing runs with our 20 Newsgroup and Reuters datasets. Experiments will illuminate some aspects of using these methods in high-dimensional text spaces where training data is limited and how that can be addressed with dimension reduction and centroid models.

2. Create the Neighbors program

From your `/soc290` project directory make a copy of the `template_supervised.py` file called `neighbors.py` and open the copy in your editor. As before we begin by editing the prelude docstring with the new file metadata and editing the file globals to define the method and models we will support. Edit the beginning of your new file this way

```
#!/usr/bin/env python
"""
@package css
@file css/neighbors.py
@author Edward Hunter
@author Your Name Here
@brief Nearest neighbor and nearest centroid supervised learning
and evaluation methods.
"""

# Import common modules and utilities.
from common import *

# Define method and models available.
METHOD = 'Neighbors'
MODELS = ('KNN', 'Centroid')
```

The docstring prelude now carries the correct file and package name. As before add your name as an `@author`. We have also defined the `METHOD` global as the string `'Neighbors'` and the `MODELS` variable as a tuple with the strings `'KNN'` and `'Centroid'`, for K-Nearest Neighbors and Nearest Centroid, respectively, after the Nearest Neighbor methods we described in lecture.

Customizing the functions for Nearest Neighbors

Again, we just need to edit our to-do sections with scikit code to customize for the above methods. In fact, the only part that is different that lab 2 is the first to-do in `train()`. Here we

create a conditional if statement that is keyed by the choice of one of the two above methods to create scikit **vectorizer** and classifier (**clf**) objects

```
#####
# Create feature extractor, classifier.
#####
vectorizer = TfidfVectorizer(stop_words='english',sublinear_tf=True)
if model == 'KNN':
    clf = KNeighborsClassifier()
elif model == 'Centroid':
    clf = NearestCentroid()
#####
```

In this case we see that a TFIDF **vectorizer** is constructed before the if clause. That means that we use the same bag-of-words model (TFIDF) for both methods. The scikit **KNeighborsClassifier** object is created if the KNN model is specified, and the scikit **NearestCentroid** object is created if the Centroid model is specified; either way the classifier is assigned to **clf** as before.

Creating a different kind of classifier is all that is new here as compared to lab 2. The other three to-dos in train() and in the other functions and entry point are filled out exactly like before. For convenience, we repeat the sections here that have to be edited: the dimension reducer, the feature extractor and the training function look like this

```
#####
# If specified, create feature dimension reducer.
#####
dim = kwargs.get('dim', None)
if dim:
    fselector = SelectKBest(chi2, k=dim)
#####
```

```
#####
# Extract features, reducing dimension if specified.
#####
print 'Extracting text features...'
start = time.time()
x_train = vectorizer.fit_transform(data_train)
if dim:
    x_train = fselector.fit_transform(x_train, data_train_target)
    x_train = normalize(x_train)
print 'Extracted in %f seconds.' % (time.time() - start)
#####
```

```
#####
# Train classifier.
#####
print 'Training classifier...'
start = time.time()
clf.fit(x_train, data_train_target)
print 'Trained in %f seconds.' % (time.time() - start)
#####
```

In predict() we have

```
#####
# Compute features and predict.
#####
x_test = vectorizer.transform(input_data)
if dfname:
    x_test = fselector.transform(x_test)
    x_test = normalize(x_test)
pred = clf.predict(x_test)
#####
```

In eval()

```
#####
# Evaluate predictions: metrics.
#####
class_report = metrics.classification_report(data_test_target, pred,
                                             target_names=data_target_names)
conf_matrix = metrics.confusion_matrix(data_test_target ,pred)
#####
```

You can just cut and paste these common sections from the naive_bayes.py file to make life easy. And again we remove the to-dos from the entry point

```
#####
# Add method specific options.
#####
# TODO
#####

#####
# Extract method specific options.
#####
# TODO (optional): create options dict method_kwargs for train.
#####
```

So you notice from this that there is a lot of code reuse going on here. We only need slight modifications to make very different behavior in our programs. With slightly more sophisticated programming techniques, we could even make this reuse explicit in our programs so there would be no cut-and-paste involved. But for simplicity sake, we are just going to edit the templates this way. If you have a little more programming experience, know about object-oriented concepts or are just adventurous, see the instructor and we can outline an even better design you can try out or use for your future work.

3. Collecting Nearest Neighbor Performance Data and Preparing Results

Let's run our Nearest Neighbors program on the testing data to see how they perform. We'll do this for 20 Newsgroups and Reuters data with both models, and also with a dimension reduction option to see how that effects performance. After collecting and assembling results, we'll pose some questions in the next section to answer in the report.

First lets gather data for the basic KNN model on the 20 Newsgroups data. The command lines for doing this on the 3 flavors of newsgroups data are as follows (output omitted)

```
(css)$ python neighbors.py -c linear KNN 20news
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c linear KNN 20news5
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c linear KNN 20news4
. . . OUTPUT AND REPORT . . .
```

Here we have run the full 20 Newsgroups, 20news5 and 20news4 datasets all with a KNN model and generated a linear confusion matrix along with a report file for each. The contents of /soc290/reports, assuming it is initially empty or deleted will look like this

```
(css)$ ls -lora reports/
total 224
drwxr-xr-x  8 edward  272 Mar  1 14:55 .
drwxr-xr-x 26 edward  884 Mar  1 14:55 ..
-rw-r--r--  1 edward 27325 Mar  1 14:55 Neighbors_KNN_20news4_confusion.png
-rw-r--r--  1 edward  699 Mar  1 14:55 Neighbors_KNN_20news4_report.txt
-rw-r--r--  1 edward 30616 Mar  1 14:55 Neighbors_KNN_20news5_confusion.png
-rw-r--r--  1 edward  756 Mar  1 14:55 Neighbors_KNN_20news5_report.txt
-rw-r--r--  1 edward 40814 Mar  1 14:55 Neighbors_KNN_20news_confusion.png
-rw-r--r--  1 edward  3326 Mar  1 14:55 Neighbors_KNN_20news_report.txt
```

So as before, we have report files (same as seen on the command line output) and confusion image files for each of the 20 Newsgroups datasets.

Take a moment to open and examine these files and notice what you see. Does this performance look very good to you? Think about how these cases perform on average as well as how the best and worst performing individual categories look in the performance distributions. We'll ask some questions about this in the next section.

Now lets perform the same analysis but instruct the processing to limit the feature dimension by using the chi-squared statistic to select the top features in advance of training and classification. The command lines are

```
(css)$ python neighbors.py -c -d 500 linear KNN 20news
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c -d 500 linear KNN 20news5
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c -d 100 linear KNN 20news4
. . . OUTPUT AND REPORT . . .
```

Notice we used the -d option here. The argument to -d is an integer specifying the desired feature dimension, so in 20news and 20news5 we are training and classifying with a 500-dimensional bag-of-words model, while the 20news4 using a 100-dimensional model. The new reports and confusion images are now in your reports directory

```
(css)$ ls -l reports/
total 464
drwxr-xr-x 14 edward 476 Mar 1 18:02 .
drwxr-xr-x 26 edward 884 Mar 1 18:02 ..
-rw-r--r-- 1 edward 29627 Mar 1 17:56 Neighbors_KNN_20news4_100_confusion.png
-rw-r--r-- 1 edward 703 Mar 1 17:56 Neighbors_KNN_20news4_100_report.txt
-rw-r--r-- 1 edward 27325 Mar 1 14:55 Neighbors_KNN_20news4_confusion.png
-rw-r--r-- 1 edward 699 Mar 1 14:55 Neighbors_KNN_20news4_report.txt
-rw-r--r-- 1 edward 29472 Mar 1 18:02 Neighbors_KNN_20news5_500_confusion.png
-rw-r--r-- 1 edward 785 Mar 1 18:02 Neighbors_KNN_20news5_500_report.txt
-rw-r--r-- 1 edward 30616 Mar 1 14:55 Neighbors_KNN_20news5_confusion.png
-rw-r--r-- 1 edward 756 Mar 1 14:55 Neighbors_KNN_20news5_report.txt
-rw-r--r-- 1 edward 41650 Mar 1 15:06 Neighbors_KNN_20news_500_confusion.png
-rw-r--r-- 1 edward 3730 Mar 1 15:06 Neighbors_KNN_20news_500_report.txt
-rw-r--r-- 1 edward 40814 Mar 1 14:55 Neighbors_KNN_20news_confusion.png
-rw-r--r-- 1 edward 3326 Mar 1 14:55 Neighbors_KNN_20news_report.txt
```

The `_500_` and `_100_` annotations in the file name gives the dimension size when `-d` is specified.

You are probably wondering at this point how we came up with the choice of dimension. These parameters are usually determined empirically in practice. In smaller scale exploratory work we can often do this just by trial and error to find an acceptable parameter. In more complicated settings, or in a production or research setting where we must find the optimal performance, we can run the tests in a loop called a **grid search**. This means we specify the range and step resolution of the unknown parameters and run the training set across all possibilities to identify the best settings. Here we've done a little experimentation ahead of time for you to identify some reasonable dimension parameters. Take a moment to open the dimensionally reduced output files and compare them to the full dimension KNN models. Notice a difference?

Following the same steps for the Reuters data we have the following command lines and the new reports. Notice that we ask for a log confusion matrix, and for the dimension reduction case, 200 features.

```
(css)$ python neighbors.py -c log KNN reuters21578-10
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c log -d 200 KNN reuters21578-10
. . . OUTPUT AND REPORT . . .
```

Finally, let's collect results using the Nearest Centroid classifier on our datasets. The command line and results without dimension reduction are

```
(css)$ python neighbors.py -c linear Centroid 20news
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c linear Centroid 20news5
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c linear Centroid 20news4
. . . OUTPUT AND REPORT . . .
(css)$ python neighbors.py -c log Centroid reuters21578-10
. . . OUTPUT AND REPORT . . .
```

The full set of reports in /soc290/reports is

```
(css)$ ls -la reports/
total 944
drwxr-xr-x 26 edward 884 Mar 1 16:45 .
drwxr-xr-x 26 edward 884 Mar 1 14:55 ..
-rw-r--r-- 1 edward 30026 Mar 1 16:45 Neighbors_Centroid_20news4_confusion.png
-rw-r--r-- 1 edward 704 Mar 1 16:45 Neighbors_Centroid_20news4_report.txt
-rw-r--r-- 1 edward 29980 Mar 1 16:44 Neighbors_Centroid_20news5_confusion.png
-rw-r--r-- 1 edward 786 Mar 1 16:44 Neighbors_Centroid_20news5_report.txt
-rw-r--r-- 1 edward 41992 Mar 1 16:44 Neighbors_Centroid_20news_confusion.png
-rw-r--r-- 1 edward 3731 Mar 1 16:44 Neighbors_Centroid_20news_report.txt
-rw-r--r-- 1 edward 34236 Mar 1 16:45 Neighbors_Centroid_reuters21578-10_confusion.png
-rw-r--r-- 1 edward 1426 Mar 1 16:45 Neighbors_Centroid_reuters21578-10_report.txt
-rw-r--r-- 1 edward 28734 Mar 1 15:07 Neighbors_KNN_20news4_100_confusion.png
-rw-r--r-- 1 edward 703 Mar 1 15:07 Neighbors_KNN_20news4_100_report.txt
-rw-r--r-- 1 edward 27325 Mar 1 14:55 Neighbors_KNN_20news4_confusion.png
-rw-r--r-- 1 edward 699 Mar 1 14:55 Neighbors_KNN_20news4_report.txt
-rw-r--r-- 1 edward 29472 Mar 1 15:07 Neighbors_KNN_20news5_500_confusion.png
-rw-r--r-- 1 edward 785 Mar 1 15:07 Neighbors_KNN_20news5_500_report.txt
-rw-r--r-- 1 edward 30616 Mar 1 14:55 Neighbors_KNN_20news5_confusion.png
-rw-r--r-- 1 edward 756 Mar 1 14:55 Neighbors_KNN_20news5_report.txt
-rw-r--r-- 1 edward 41650 Mar 1 15:06 Neighbors_KNN_20news_500_confusion.png
-rw-r--r-- 1 edward 3730 Mar 1 15:06 Neighbors_KNN_20news_500_report.txt
-rw-r--r-- 1 edward 40814 Mar 1 14:55 Neighbors_KNN_20news_confusion.png
-rw-r--r-- 1 edward 3326 Mar 1 14:55 Neighbors_KNN_20news_report.txt
-rw-r--r-- 1 edward 33714 Mar 1 15:18 Neighbors_KNN_reuters21578-10_200_confusion.png
-rw-r--r-- 1 edward 1425 Mar 1 15:18 Neighbors_KNN_reuters21578-10_200_report.txt
-rw-r--r-- 1 edward 33652 Mar 1 15:18 Neighbors_KNN_reuters21578-10_confusion.png
-rw-r--r-- 1 edward 1421 Mar 1 15:18 Neighbors_KNN_reuters21578-10_report.txt
```

As before, the results section of the lab report should have the following 9 pages, titled as below.

1. Data 1: KNN 20news
2. Data 2: KNN 20news5 and 20news4
3. Data 3: KNN 20news, dim=500
4. Data 4: KNN 20news5 dim=500, 20news4 dim=100
5. Data 5: KNN Reuters
6. Data 6: KNN Reuters, dim=200
7. Data 7: Nearest Centroid, 20news
8. Data 8: Nearest Centroid, 20news5 and 20news4
9. Data 8: Nearest Centroid, Reuters

4. Analyzing the Data and Preparing the Lab Report

In the analysis section at the beginning of the report, provide the following sections with discussion of the following questions. When done, add a cover page with your name, email and lab title. Save the document as a pdf and email to the instructor and TA.

Analysis of Results: 20news, 20news5, 20news4 and Reuters

1. How would you characterize the performance of overall in each of the tests? Justify your argument with specific references to the average precision and recall and their distributions across document categories in each case.
2. How does this performance compare to the Naïve Bayes TFIDF model? Based on our discussion in lecture, how can you explain this difference?
3. We learned in lecture that KNN classifiers have quite strong performance guarantees. Describe these guarantees in your own words. Do you think these performance guarantees are valid for these results? Why or why not?
4. Why and how might performance be improved in these models?

Analysis of Results: 20news dim=500, 20news5 dim=500, 20news4 dim=100 and Reuters dim=200

1. How would you characterize the performance of overall in each of the tests? Justify your argument with specific references to the average precision and recall and their distributions across document categories in each case.
2. Dimension reduction has the effect of markedly improving performance over full-dimensional models. Describe why reduction in feature dimension has the effect you observe. How do you think this is related to the data sample size?

Analysis of Results: Centroid Models

1. How do the centroid models compare to both their full and reduced dimension equivalents?
2. What does the performance of these centroid models say about how these particular data are distributed geometrically in document space? Do you think centroid models will always perform this way (i.e. with any dataset)? Under what conditions might they perform less well?