Assignment 08

Machine Learning, Summer term 2018
Norman Hendrich, Marc Bestmann, Philipp Ruppel
June 04, 2018

Solutions due by June 10

Assignment 08.1 (Sparse Matrices, 5×0.5 points)

Very large matrices are often *sparse*, that is, many or even most matrix elements are zero. The *scipy.sparse* module provides Python data structures for memory-efficient layout of sparse matrices and a set of common linear algebra operations on the matrices.

a. Please read the documentation for the *scipy.sparse* module, `docs.scipy.org/doc/scipy/reference/sparse.html`.

   Why are there (seven) different types (*bsr_matrix ... lil_matrix*) of sparse matrices? Explain the memory layout of *csr_matrix*.

b. Which of these classes is best for inserting new non-zero elements into an existing sparse matrix? Which of these classes are best for matrix-vector dot products and matrix-matrix multiplications? Why?

c. The *sparsity* of a matrix is defined as the ratio of zero matrix elements to the total number of matrix elements. How can you calculate the sparsity of a matrix using the scipy API?

d. Generate a tridiagonal random matrix A of size $n \times n$; that is, a matrix with random non-zero elements only on its main diagonal and the first upper and lower diagonals.

e. Linear algebra operations are provided in the *scipy.sparse.linalg* module. Run *spsolve()* to solve the linear equation $A \cdot x = b$, where $b = \mathbb{1}$ is a vector of all ones.

   Measure the memory usage and the execution time for different values of n (e.g. $n = 10, 100, 1000, 10000, \ldots$) using both the sparse matrix representation and the "traditional" dense matrix represenation, e.g., by calling A.todense(). What is the largest value of n that works on your computer?

Assignment 08.2 (Bag-of-Words Text Classification , 5×0.5 points)

The *20 Newsgroups* data set is a collection of approximately 20.000 newsgroup documents, partitioned across 20 different usenet newsgroups. It has become a popular data set for experiments in text classification and clustering.

a. Please see `scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html` for a description of the dataset. First, download the four-newsgroups subset from the full dataset:

```
from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(
  categories=['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.charistian'],
  shuffle=True, random_state=41 )
```

   Play with *twenty_train* to access the texts, the corresponding labels (newsgroup indices), and the names of the newsgroups. How many texts have been loaded?

b. Follow the tutorial and use *sklearn.feature_extraction.text.CountVectorizer* to tokenize and generate the list of words from the training data. How many words were found? How do you access the list of words? How do you find the numerical index of a word in the feature vectors?

c. Continue the tutorial and train the *MultinomialNB* Bayes classifier using the feature vectors generated by *CountVectorizer*. Test and classify a few texts (strings) of your own. Then finish the tutorial with building the *Pipeline* and classify the texts from the test set.

d. What is the motivation for the *stop_words* argument of *CountVectorizer*? Try running the full example (including tokening, training the classifier, and running classification on the test set) again when using *stop_worlds='english'*. How many words are found? What is the performance (accuracy

e. Read the paragraph about *term frequencies* and normalization using *TfidfTransformer* or *TfidfVectorizer*. What is the motivation for working with frequencies instead of word counts?

Assignment 08.3 (Text classification pipeline, 4×0.5 points)

Download the example Python scripts from `scikit-learn.org/stable/auto_examples/text/document_classification_20newsgroups.html`. This again loads (a subset of or the complete) 20 newsgroups dataset, builds the dictonary and Tfidf frequencies, and then trains and runs several classifiers from the sklearn library.

a. Run the script with the default settings and with the *–report* and *–confusion_matrix* command line arguments. This uses the whole input texts for training and classification.

b. Repeat, but now with the *–filtered* argument. This removes the email headers, signatures, and quoting from the input documents. Compare the classifier scores (e.g. accuracy, precision) with the above runs.

c. Repeat with *–all_categories*; this can take a while. (Note: Just uncomment the broken print). Look at the confusion matrices. Can you explain some of the mispredictions?

d. Is any one classifier the clear winner for this task?