Information Processing and Retrieval

Instituto Superior Técnico 2024

Lab 4: Learning in IR (document organizing, annotation, ranking)

Today we will use the 20 Newsgroup dataset[1]. scikit-learn provides access to this dataset.

```
from sklearn.datasets import fetch_20newsgroups
categories = ['comp.graphics','rec.autos','sci.crypt','talk.politics.guns']
collection = fetch_20newsgroups(subset="test", categories=categories)
```

The actual data is in text format. You need to transform it into numeric weight vectors (using for instance the term frequency or TF-IDF vector space model). As we saw in earlier labs, the scikit-learn library provides methods for this:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer( use_idf=False )
vectorspace = vectorizer.fit_transform(collection.data)
```

1 Clustering the 20 NewsGroup collection

Clustering of documents can be also achieved using scikit-learn library.

1.1. Extract the vector space of the 20 Newsgroup (use collection.data instruction to ignore the class variable), and cluster the collection using agglomerative clustering from scikit-learn.

```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering().fit(vectorspace)
print(clustering.labels_)
```

Parameterize the clustering search to use cosine as the distance function.

```
AgglomerativeClustering(n_clusters=n_clusters, linkage='average', affinity='cosine')
```

1.2. Plot the learned dendogram using the cluster.hierarchy.dendrogram package from scipy. Compare the clustering solutions produced under single and complete linkage criteria.

1.3. Evaluate the clustering solution by computing an internal measure (e.g. *silhouette*) and an external measure (e.g. *adjusted rand index*) for the produced clustering solution.

```
from sklearn.metrics import silhouette_score, adjusted_rand_score
silhouette_score(vectorspace, cluster_labels, metric='cosine')
adjusted_rand_score(cluster_labels, true_labels)
```

[1] http://qwone.com/jason/20Newsgroups/

1.4. (*homework*) Principal component analysis (PCA), latent semantic indexing (LSI) and uMAP procedures offer a way of projecting our high-dimensional vector space into a space with lower dimensionality. Map the original space into a 2-dimensional space, plotting the documents in this new space. Color the documents according to their cluster to assess their separation.

```
from sklearn.decomposition import PCA
newspace = PCA(n_components=2).fit_transform(vectorspace)
```

2 Document Classification in the 20 NewsGroup

We can standardly split the 20 NewsGroup collection into train and test sets.

```
from sklearn.datasets import fetch_20newsgroups
train = fetch_20newsgroups(subset='train')
test = fetch_20newsgroups(subset='test')

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer( use_idf=False )
trainvec = vectorizer.fit_transform(train.data)
testvec = vectorizer.transform(test.data)
```

You can see the vector representation of documents using train.data and the corresponding classes using train.target. You will notice that the classes are represented as numbers. To recover the class names, use train.target_names. You can now fit a classifier on the training data and test it on the testing data.

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(trainvec, train.target)
classes = classifier.predict(testvec)
```

The scikit-learn library also provides classes to evaluate classification results.

```
from sklearn import metrics
print metrics.accuracy_score(test.target, classes)
print metrics.classification_report(test.target, classes)
```

2.1. Implement a classifier for the 20 Newsgroups collection and measure its performance. Use for instance the above Multinomial Naïve Bayes classifier, available in scikit-learn.

2.2. Try to improve the classification by:

(a) Removing very rare words (e.g. words that occur less than 2 times) or very frequent words (e.g. words that occur in more than 90% documents) using *Vectorizer* utils by scikit-learn

(b) Compare the performance against alternative classification algorithms, such as:
- a nearest neighbour classifier (sklearn.neighbors.KNeighborsClassifier)
- the perceptron algorithm (sklearn.linear_model.Perceptron)
- support vector machines (sklearn.svm.LinearSVC)

3 Learning to Rank

Let us create a ranking method that linearly combines results from different scoring functions,

$$score(q, d) = \alpha_1 bm25(q, d) + \alpha_2 cos(q, d) + \alpha_3 freq(q, d)$$

where d is the document, q is the query, $bm25$ is the score obtained using the BM25 ranking function, cos is the score obtained using the TF_IDF ranking function, and $freq$ is the score obtained using the Term Frequency ranking function.

3.1. Implement this integrative scoring scheme. Assess how different weights α_1, α_2, and α_3 impact the *Mean Average Precision* (MAP) against each individual ranking function in isolation.

3.2. Let us try a more sophisticated approach to combine the functions by training a Logistic Regression classifier[2] on the set of queries in pri_queries.txt from the second lab.

More specifically:

(a) Create a dataset for training and testing your *Learning to Rank* (L2R) approach:

- use 70% of the queries for training and 30% for testing;
- with the training queries, build the *training dataset*. This dataset should contain, for each *(query q, document d)* pair, a set of instances $\{\mathbf{x}_1, .., \mathbf{x}_n\}$ with the format:

$$\mathbf{x}_i = (bm25(q, d), cos(q, d), freq(q, d), r)$$

  where $r = 1$ if document d is relevant for query q and $r = 0$ otherwise;
- use the same number of relevant and non-relevant documents for each query.

(b) Use the training data to learn the logistic regression classifier with relevance r as target.

(c) Run one testing query using the learnt logistic regression as your classifier (1 if the document is relevant or 0 if otherwise). Compute the *precision*, *recall*, and F_1 scores against ground relevance.

(d) Run one testing query using the learnt logistic regression as your ranking function. To order the resulting documents, use the *probability of the document being relevant* through the predict_proba method. Compute the MAP for the produced ranking.

3.3. Can additional features be further added to guide ranking? Which?

[2] https://scikit-learn.org/stable/modules/generated/sklearn.linearmodel.LogisticRegression.html

4 Pen-and-paper exercises

Consider the following collection of 4 text documents and corresponding labels (ground truth):

ID	text document	label
1	shipment of gold damaged in fire	A
2	delivery of silver arrived in silver truck	B
3	shipment of silver arrived in truck	B
4	truck damaged in fire	B

4.1. Consider documents to be represented as a Boolean model and their similarity assessed under the Manhattan distance, $\|\mathbf{d}_1 - \mathbf{d}_2\|_1 = \sum_{i=1}^{n} |d_1^i - d_2^i|$. Compute the pairwise distance matrix and the dendogram obtained with the complete (maximum) link criterion.

4.2. Assess the clustering solution from previous exercise, computing:

- **4.2.1.** the *purity* and *rand index*;
- **4.2.2.** the *silhouette* of each cluster and of the overall clustering solution.

4.3. Considering a classification of these documents, (d_1=B, d_2=B, d_3=B, d_4=A), draw the confusion matrix for this annotation attempt, computing its accuracy and F$_{\beta=2}$-measure.