TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Datenverarbeitung
PD Dr. Martin Kleinsteuber

Information Retrieval in High Dimensional Data

Lab #3, 18.05.2017

Data Preparation

In this exercise we will work with the IMDB sentiment dataset. This dataset contains movie reviews, each with a positive or negative sentiment (quantized by 1 for positive and 0 for negative). The labeled training and testing data is provided on Moodle.

Reading and preprocessing the data

To import the tsv file, it is recommended to use the pandas package. The provided file can be imported as follows

```
import pandas as pd
train = pd.read_csv("labeledTrainData.tsv", header=0, \
                    delimiter="\t", quoting=3)
```

- What data type is the variable train? Which values does it contain? Print some examples.

The text strings contain HTML tags, which have to be removed. To do this, use the bs4 package

```
# Import BeautifulSoup into your workspace
from bs4 import BeautifulSoup

# Initialize the BeautifulSoup object on a single movie review
example1 = BeautifulSoup(train["review"][0], 'lxml').get_text()
```

The imported text contains punctuation, numbers, and all (common) words. For now, we assume that these are not beneficial to the task of sentiment classification, and we want to remove them. Punctuation and numbers can be removed using the regular expressions (re) package

```
import re
# Use regular expressions to do a find-and-replace
letters_only = re.sub("[^a-zA-Z]",           # The pattern to search for
                      "␣",                   # The pattern to replace it with
                      example1.get_text() )  # The text to search
print letters_only
```

It is also beneficial, to convert all letters to lower case and to split the strings into individual words.

```
lower_case = letters_only.lower()        # Convert to lower case
words = lower_case.split()               # Split into words
```

For now, we also want to remove common words that do not carry much meaning, such as 'a', 'is', or 'the'. These are often referred to as stop words. A list of stop words can be obtained with the NLTK package:

```
import nltk
nltk.download()  # Download text data sets, including stop words
from nltk.corpus import stopwords # Import the stop word list
print stopwords.words("english")
```

- Write a function called `review_prepro` that takes as an input a raw review string and returns a preprocessed review, i.e. a string with HTML tags removes, all lower case letters, no stop words.
  Then apply this function to the entire training set. Return the list `clean_train_reviews`, which contains all the cleaned reviews.

Creating Features from a Bag of Words

For generating a bag of words model, we will use the scikit-learn package. Use the following code

```
from sklearn.feature_extraction.text import CountVectorizer

# define the vectorizer
vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = None,    \
                             preprocessor = None, \
                             stop_words = None,   \
                             max_features = 5000)

# fit the vectorizer to the data
train_data_features = vectorizer.fit_transform(clean_train_reviews)
# convert to numpy array
train_data_features = train_data_features.toarray()
```

Black box classifier

To do something meaningful with the generated data, we will use a prebuilt classifier, train it on the training data and then evaluate the learned classifier on the test data `labeledTestData.tsv`.

First, preprocess the test data the same way as the training data and return the variable `test_data_features`. (Hint: use `vectorizer.transform`)

To train a classifier with logistic regression use the following code

```
from sklearn.linear_model import LogisticRegression as LR

model = LR()
model.fit( train_data_features, train["sentiment"] )

p = model.predict_proba( test_data_features )[:,1]
output = pd.DataFrame( data={"id":test["id"], "sentiment":p} )
```

Evaluate result

We will use the Area Under Curve (AUC) metric to measure performance. An AUC score of 0.5 is the same as a random classifier, the closer to 1 the score is the better.

```
from sklearn.metrics import roc_auc_score as AUC

auc = AUC( test["sentiment"].values, p[:,1] )
```

More sophisticated methods

- Use a prebuilt TF-IDF vectorizer and play around with its settings such as stop words and n-grams and the performance of an LR classifier.

  ```
  from sklearn.feature_extraction.text import TfidfVectorizer
  vectorizer = TfidfVectorizer( max_features = 5000,
                                ngram_range = ( 1, 1 ),
                                sublinear_tf = True )
  ```