# 301110 Applications of Big Data Assignment

Nick Tothill

Spring 2020

## 1 Description

For this assignment, you will need to create a complete program to perform sentiment classification for movie reviews from feature extraction to classification. For a given review, your program should be able to predict whether it is positive i.e. like the movie, or negative, i.e. dislike the movie.

## 2 About the data

You will use a large movie review dataset containing a set of 25,000 movie reviews for training, and 25,000 for testing. You can download the data from `vUWS` under the assignments folder named `aclImdb.zip`. You can also visit the following website for more information about the dataset:
`http://ai.stanford.edu/∼amaas/data/sentiment/`
or download data directly from there. Unzip the data to your local directory. In the `aclImdb/` directory created by the zip file (∼500MB), you will find the following three items (among others):

1. train/ — feature files and raw text files for the training set
2. test/ — feature files and raw text files for the testing set
3. README — the readme file for more information on the dataset

Read the README file carefully for descriptions of the text files that contain the reviews and their naming convention. The directories with which we are concerned here are:

1. ./aclImdb/train/pos — raw text files of positive reviews in the training set
2. ./aclImdb/train/neg — raw text files of negative reviews in the training set
3. ./aclImdb/test/pos — raw text files of positive reviews in the test set
4. ./aclImdb/test/neg — raw text files of negative reviews in the test set

A full version of this data set is available at:
`hdfs://hadoop-01-149-21-172.scem.uws.edu.au:9000/users/ugbigdata/imdb/fullversion`
A tiny cut down version with much fewer files (20 each for training and 10 each for test) is also available at:
`hdfs://hadoop-01-149-21-172.scem.uws.edu.au:9000/users/ugbigdata/imdb/tinyversion`
The tiny version is for experimentation.

## 3 Task 1. Feature extraction (15 points)

Use the MapReduce model to convert all text data into matrices. Convert ratings to vectors. These will be used for classification in Task 2. Use TF-IDF to vectorise the text files. See previous practical classes and lectures materials for TF-IDF. One step further though is to represent each text file (review) as a very long and sparse vector as the following. Assume wordslist is the final list of distinct words contained in all reviews and its length is N. Then each review will be a vector of length N, with each position associated with the word in wordlist and the value being either 0, if the corresponding word is absent in the review, or the word's TF-IDF. For example, if wordlist = ['word1', 'word2', 'word3', 'word4'] and review 1 contains word1 and word4, then the vector representation of review 1 is [0.1, 0, 0, 0.4] assuming TF-IDF of word 1 and word 4 in review 1 is 0.1 and 0.4 respectively. Note that TF is calculated from one single document while IDF is obtained from all documents in the collection.

### 3.1 Requirements:

#### 3.1.1 Req. 1

A Map-reduce model is a must. Implement it using Hadoop streaming. All data are available on SCEM HDFS. The recommendation is to work on the tiny version of the data to make the code work. You may try your code on the full version. However, the application to full version is not required.

#### 3.1.2 Req. 2

Generate two matrices: `training_data`, `training_targets`, and two vectors: `test_data`, `test_targets`. `training_data` should have $N$ rows and $D$ columns with each row corresponding to each review in the training set ($N$ is the total number of reviews in the training set and $D$ is the total number of words). $N$ and $D$ vary depending on which version of the data you use. `training_targets` should have $N$ elements each of which is the rating of the review. `test_data` and `test_targets` are similarly defined.

Notes:

1. If feature extraction is too difficult for you, you can use pre-computed bag of words features included in this data set. Refer to the appendix and README file for details. If pre-computed features are used, a 60% penalty will be incurred for this task, i.e. the maximum marks you can get from this task is 6 if you do so.

2. Using a map-reduce model to extract TF-IDF is mandatory. If not used, a 20% penalty for this task will be incurred. There is no constraint on how to form the training and test matrices and vectors. There are many versions of TF-IDF. There is no preference for which version to use.

3. You can use data frame (using pandas package) instead of matrices and vectors to store training and test data and targets.

Marking scheme for task 1:

- Text file reading (1pt): read the text files for TF-IDF extraction.
- Rating scores extraction (3pts): parse the name of text files to extract ratings.
- TF-IDF extraction (8pts): use MapReduce class to extract TF-IDF for each text file.
- Forming matrices and target vectors (or data frames) (3pts): collect TF-IDFs to form training and test data for task 2.

## 4 Task 2. Classification (15 points)

Construct a classification model for review sentiment prediction, meaning that: *given a customer movie review (taken from the test set), your program should be able to predict whether it is positive or negative.*
There is no limitation on how many classifiers and what specific model you should use. You can simply pick one that works for you for this task, either from those covered in lecture and practical class materials or any other classifiers from any python packages. A good starting point is the `scikit-learn` (i.e. sklearn) package.

A few things you need to address in your python program are listed as requirements below.

### 4.1 Requirements:

#### 4.1.1 Req. 1

Data pre-processing. In task 1, you extracted the ratings vectors for training and test. These are raw ratings. As we are interested in sentiment prediction, i.e. to predict either the review is positive or negative, you need to convert all ratings $>5$ as positive class and all ratings $<=5$ as negative class. Choose a coding scheme, e.g. 0 for positive, 1 for negative.

#### 4.1.2 Req. 2

Normalisation. Apply at least one normalisation scheme and compare the performance of the classifier(s) with and without normalisation.

#### 4.1.3 Req. 3

Training and model selection. Use cross validation to select the best parameters for your classifier. There may be many parameters to tune in some classifiers (such as random forest classifier — RFC). You can focus on the most important one(s) such as `max_depth` and `n_estimators` in RFC. Refer to the `scikit-learn` package documentation for details.

*Hint: you can start with a small subset of the training set to test a few parameters to get a feel of what range the parameters should be that make the model perform well in terms of prediction accuracy. Then turn on large scale cross validation on the whole training set.*

#### 4.1.4 Req. 4

Test on test data. After model selection, apply the best model, i.e. the model with the parameters that produce the best cross validation scores, to test data, make a prediction for each review, and record prediction accuracy

. Note:

1. Always train your classifier(s) ONLY on training data including cross validation. After model selection, apply the best model on test data to evaluate the performance.

2. Good performance, i.e. higher accuracy on test data, is not essential for this task. However, if your classifier has accuracy lower than about 60%, it usually means that there are some mistakes somewhere in your code. So try to score as high an accuracy as possible.

3. You are encouraged to try many classifiers. If the coding is right, this should not be too difficult.

Marking scheme for task 2:

- Data pre-processing (1pts): convert ratings to positive and negative coding scheme.

- Normalisation and comparison (3pts): apply normalisation and compare performance difference with and without it.

- Training on training data (3pts): training performed on training data.
- Cross validation (6pts): apply cross validation on training data.
- Testing on test data (2pts): best model applied to test data and accuracy produced.

## 5 Other Marking Criteria

Your program will be marked against both functional and operational requirements. Functional requirements accounts for 80% of the mark, which measure how well your program achieves the expected functionalities and are further broken down into the items listed in marking schemes in those tasks. In addition to function requirements, your program should also meet the operational and style requirements, as follows:
**Readability** (5%): Comments should be included in your program to explain the main idea of your design; use meaningful variable and function names; do not declare variables that are not used in the program
**Modularity** (10%): Your program should make use of functions or classes wherever possible to achieve modular design and maximise reusability.
**Usability** (5%): Your program should be easy to use by the user. These include displaying messages for user interaction, performing adequate input validation, and allowing the users to choose the locations of the data file for model training.