Statistical Learning

Due in two weeks on Moodle (May 31 + tolerance)

Homework-02

General Instructions

- You can use *any* programming language you want, as long as your work is runnable/correct/readable. Two examples:
  - **In `R`**: it would be nice to upload a well-edited and working `R Markdown` file (`.rmd`) + its `html` output.
  - **In `Python`**: it would be nice to upload a well-edited and working `Jupyter notebook` (or similia).
- Remember our **policy on collaboration**:

  Collaboration on homework assignments with fellow students is ***encouraged****.*
  However, such collaboration should be clearly acknowledged, by listing the names of the students with whom you have had discussions concerning your solution.
  You may ***not****, however, share written work or code after discussing a problem with others.*
  The solutions should be written by ***you****.*

In case of `R`

If you go for `R`, to be sure that everything is working, start `RStudio` and create an empty project called `HW1`. Now open a new `R Markdown` file (`File > New File > R Markdown...`); set the output to `HTML mode`, press `OK` and then click on `Knit HTML`. This should produce a `html`. You can now start editing this file to produce your homework submission.

- For more info on `R Markdown`, check the support webpage: R Markdown from RStudio.
- For more info on how to write math formulas in LaTex: Wikibooks.

Exercise 1: The Bayes Classifier

$\rightsquigarrow$ Your job $\leftsquigarrow$

Suppose that (Y, X) are random variable with $Y \in \{0, 1\}$ and $X \in \mathbb{R}$. Suppose that

$$(X \,|\, Y = 0) \sim \text{Unif}(-3, 1) \quad \text{and} \quad (X \,|\, Y = 1) \sim \text{Unif}(-1, 3).$$

Further suppose that $\mathbb{P}(Y = 0) = \mathbb{P}(Y = 1) = \frac{1}{2}$.

1. Define the Bayes classifier/strategy and briefly explain its role/importance in classification/prediction.
2. Find (with pen and paper) the Bayes classification rule $h_{\texttt{opt}}(x)$.
3. Simulate $n = 250$ data from the joint data model $p(y, x) = p(x \,|\, y) \cdot p(y)$ described above, and then:
   - Plot the data together with the regression function that defines $h_{\texttt{opt}}(x)$
   - Evaluate the performance of the Bayes Classifiers on these simple (only 1 feature!) data
   - Apply any other classifier of your choice to these data and comparatively comment its performance (...with respect to those of the Bayes classifiers). Of course those $n = 250$ training data should be used for training and validation too (in case there are tuning-parameters)
4. Since you are simulating the data, you can actually see what happens in repeated sampling. Hence, repeat the sampling $M = 1000$ times keeping $n = 250$ fixed (a simple `for`-loop will do it), and redo the comparison. Who's the best now? Comment.

Exercise 2: Go Gradient, Go!

For this exercise you will be working on a dataset of product reviews on Amazon that I've pre-processed in order to save you some time (see the box below for the details, also linked here). The dataset consists of a random sample of reviews of `books` and an equal number of reviews from `film/television series`.
More specifically there are ~150,000 training data and ~50,000 test data with 1334 features/words.

Your goal is to build models that predicts whether the reviews is of a `books` or a `film/television series` using word counts as features vectors.

The data are stored in a single *R-friendly* file called `amazon_review_clean.RData` and everything, response vector and feature matrix, are already splitted into training and test sets: `y_tr`, `X_tr`, `y_te` and `X_te`. Since these are very simple, *tidy* objects, it's easy to export them as ASCII files with `write.table()` to start working in `Python` or any other programming language.

The `X`–matrices are known as *term-frequency matrices* or *document-term matrices* and count how often various words are used in the text. For more info have also a look at the `Kernel for Text` slide set (pp. 142-147).
Specifically, the `X[i,j]` element counts the number of times the word w_j appears in the i^{th} review for some pre-specified set of words of interest. These matrices were produced using the cleanNLP package (see the box below for details): I included any word that occurred in at least 0.1% and no more than 50% of the reviews for a total of about 1300 words.

⇝ Your job ⇜

1. Looking at the formula at the bottom of page 8 of our notes, train a linear classifier by gradient descent (GD). Check its performance and then comment the results.
   HINT: Please notice that you can pre-compute (meaning, compute *before* the GD loop) the heavy part.

2. **Bonus**. If you are brave enough, go *stochastic* (page 40-43) and comment your "*experience*"...

Additional Code / Data Cleaning and Preparation

```r
# Packages ---------------------------------------------------------------
require(readr)
require(cleanNLP)  # https://statsmaths.github.io/cleanNLP/
require(stringi)
require(caret)

# Load & Look ------------------------------------------------------------
amazon <- read_csv("amazon.csv")
str(amazon)
table(amazon$class)
stri_wrap(amazon$text[amazon$class == "book"][1:10])

# Remove NA otherwise troubles with <clnp_annotates> below
len <- stri_length(amazon$text)
hist(len, breaks = 1000)
summary(len)  # wow! min length = 2, average length = 917, max length...30k!! XD
quantile(len, seq(0,1,.1), na.rm = T)

idx_na <- which(is.na(len))
amazon <- amazon[-idx_na,]

# Clean ------------------------------------------------------------------
?cnlp_init_stringi
?cnlp_annotate

cnlp_init_stringi()  # initialize tokenizer backend
anno <- cnlp_annotate(amazon, text_name = "text")   # take some time...

# tf-idf score -----------------------------------------------------------
```

```r
# See our notes: Kernel for Text (pp. 142-147)
# https://elearning.uniroma1.it/pluginfile.php/1029332/mod_folder/intro/Lecture_11.pdf
?cnlp_utils_tfidf

# The options in the call determine what words are included:
# a word must be used in at least <mid_df> percent of documents
# but not in more than <max_df> documents
X <- cnlp_utils_tfidf(anno$token,
                      min_df = 0.01, max_df = 0.5,
                      tf_weight = "raw")

# Take a look
dim(X)
colnames(X)
round(as.matrix(X[1:10, 1:10]), 2)

# Train-Test split --------------------------------------------------------

y <- amazon$class
X <- as.matrix(X)

set.seed(124) # for reproducibility
idx_tr <- createDataPartition(y = y, p = .75, list = F)

X_tr <- X[ idx_tr, ]; X_te <- X[-idx_tr, ]
y_tr <- y[ idx_tr ]; y_te <- y[-idx_tr]

dim(X_tr); dim(X_te); length(y_tr); length(y_te)
```