# Project A: Classifying Sentiment from Text Reviews

Last modified: 2023-09-28 14:53

**Status: RELEASED.**

**Due date**: Thu. Oct. 19, 2023 by 11:59pm ET (Boston time)

**Jump to**:

Background Code Datasets Problem 1: Bag-of-Words Problem 2: Open Challenge Rubric

**Turn-in links**: See What to Turn In Instructions for details

- PDF report turned in to: https://www.gradescope.com/courses/596466/assignments/3432985
- ZIP file of test-set predictions for Problem 1's Bag-of-Words Leaderboard: https://www.gradescope.com/courses/596466/assignments/3433046
- ZIP file of test-set predictions for Problem 2's Open-Ended Leaderboard: https://www.gradescope.com/courses/596466/assignments/3433292
- Reflection on Project A: <#TODO>

# Overview

This is a multi-week project with lots of open-ended programming. Get started right away!

- Release on Thu 9/28
- Form partners by Thu 10/05 (complete signup form linked below)
- Due on Thu 10/19

Suggested intermediate deadlines:

- by Tue 10/09: Complete Problem 1 code/experimentation + leaderboard submission
- by Thu 10/11: Complete Problem 1 writeup
- by Tue 10/17: Complete Problem 2 code/experimentation + leaderboard submission
- by Thu 10/19: Complete Problem 2 writeup

## Team Formation

By start of class on Thu 10/05, you should have identified your partner and signed up here:

- ProjectA Team Formation Form

Even if you decide to work alone, you should fill this form out acknowledging that.

In this project, you are encouraged to work as a team of 2 people. If you prefer, you can work individually. Individual teams still need to complete all the parts below and will be evaluated no differently than teams. We strongly recommend working in pairs to keep your workload manageable.

If you need help finding a teammate, please post to our "Finding a Partner for Project A" post on Piazza.

## Work to Complete

As a team, you will work on one *semi-open* problems, and then a *completely open* problem.

The 2 problems look at different representations of text for a common task.

- Problem 1 looks at using bag-of-word feature representations
- Problem 2 is an open-ended problem, where any feature representation is allowed

Throughout Problems 1 and 2, you will practice the *development cycle* of an ML practitioner:

- Propose a reasonable ML pipeline (feature extraction + classifier)
- Train that pipeline on available data
- Evaluate results carefully on available data
- Revise the pipeline and repeat

For all problems, we will maintain a *leaderboard* on Gradescope. You should periodically submit the predictions of your best model on the *test set* (we do not release the true labels of the test set to you in advance).

## What to Turn In

Each team will prepare *one* PDF report covering all problems.

- Suggested length 4 pages (upper limit is 6 pages)
- This document will be manually graded according to our rubric
- Can use your favorite report writing tool (Word or G Docs or LaTeX or ....)
- Should be **human-readable**. Do not include code. Do NOT just export a jupyter notebook to PDF.
- Should have each subproblem marked via the in-browser Gradescope annotation tool)

Each team will prepare a ZIP file of test-set predictions for each of Problem 1 and Problem 2.

- Each submission ZIP will contain just one plain text file: `yproba1_test.txt`
- - Each line contains float probability that the relevant example should be classified as a positive example given its features
- - Should be loadable into NumPy as a 1D array via this snippet: `np.loadtxt('yproba1_test.txt')`
- - Will be thresholded to produce hard binary predicted labels (either 0 or 1)

Each *individual* will turn in a *reflection form* (after completing the report).

- Link: <#TODO>

# Datasets, Starter Code and Code Restrictions

For all required data and code, see the projectA folder of the public assignments repo for this class:

https://github.com/tufts-ml-courses/cs135-23f-assignments/tree/main/projectA

Our starter code repo provides a few scripts helping you load the data for each problem, but otherwise offers *no* other code. This is meant to simulate the lack of code you'd have in the "real world", trying to build a text sentiment classifier from scratch using your machine learning skills.

For this assignment, you can use *any* Python package you like (sklearn, nltk, etc). You are welcome to consult the sklearn documentation website or other external web resources for snippets of code to guide your usage of different classifiers. However, you should *understand* every line of the code you use and not simply copy-paste without thinking carefully. You should also cite and acknowledge third-party code that made a significant impact on your work in your report.

Remember to keep the course collaboration policy in mind: do your own work!

# Background

We have given you a dataset of several thousand single-sentence reviews collected from three domains: imdb.com, amazon.com, yelp.com. Each review consists of a sentence and a binary label indicating the emotional *sentiment* of the sentence (1 for reviews expressing *positive* feelings; 0 for reviews expressing *negative* feelings). All the provided reviews in the training and test set were scraped from websites whose assumed audience is primarily English speakers, but of course may contain slang, misspellings, some foreign characters, and many other properties that make working with natural language data challenging (and fun!).

Your goal is to develop a binary classifier that can correctly identify the sentiment of a new sentence.

Here are some example *positive* sentences:

```
imdb          The writers were "smack on" and I think the best actors and actresses were a bonus to the show.These
characters were so real.
imdb          The Songs Were The Best And The Muppets Were So Hilarious.
yelp          Food was so goooodd.
yelp          I could eat their bruschetta all day it is devine.
```

Here are some example negative sentences:

```
amazon        It always cuts out and makes a beep beep beep sound then says signal failed.
amazon        the only VERY DISAPPOINTING thing was there was NO SPEAKERPHONE!!!!
yelp          It sure does beat the nachos at the movies but I would expect a little bit more coming from a restaurant.
yelp          I'm not sure how long we stood there but it was long enough for me to begin to feel awkwardly out of place.
```

## Dataset acknowledgment

This dataset comes from research work by D. Kotzias, M. Denil, N. De Freitas, and P. Smyth described in the KDD 2015 paper 'From Group to Individual Labels using Deep Features'. We are grateful to these authors for making the dataset available.

# Provided data

You are given the data in CSV file format, with 2400 input,output pairs in the training set, and 600 inputs in the test set.

**Training set** of 2400 examples

`x_train.csv` : input data, as text

- Column 1: 'website_name' : one of ['imdb', 'amazon', 'yelp']
- Column 2: 'text' : string sentence which represents the raw review

`y_train.csv` : binary labels to predict

- Column 1: 'is_positive_sentiment' : 1 = positive sentiment, 0 = negative

**Test set** of 600 examples

`x_test.csv` : input data, as text

- Column 1: 'website_name': as above
- Column 2: 'text': as above

# Performance metric

We will use Area under the ROC curve (AUROC) to judge your classifier's quality.

# Suggested Way to Load Data into Python

We suggest loading the sentence data using the `read_csv` method in Pandas:

```
x_train_df = pd.read_csv('x_train.csv')
tr_list_of_sentences = x_train_df['text'].values.tolist()
```

You can see a short example working Python script here: https://github.com/tufts-ml-courses/cs135-23f-assignments/blob/main/projectA/load_train_data.py

We'll often refer to each review or sentence as a single "document". Our goal is to classify each document into either the positive or negative sentiment class.

# Preprocessing

As discussed in class, there are many possible approaches to *feature representation*, the process of transforming any possible natural language document (often represented as an ordered list of words which can be of variable length) into a feature vector $x_n$ of a standard length.

In this project, we will explore several approaches, including bag-of-words vectors (explored in Problem 1). Later, you'll be allowed to try any feature representation approach you want (Problem 2).

In most cases, we suggest that you consider removing punctuation and converting upper case to lower case.

# Problem 1: Bag-of-Words Feature Representation

## Background on Bag-of-Words Representations

As discussed in class on day10, the "Bag-of-Words" (BoW) representation assumes a fixed, finite-size vocabulary of V possible words is known in advance, with a defined index order (e.g. the first word is "stegosaurus", the second word is "dinosaur", etc.).

Each document is represented as a count vector of length V, where entry at index v gives the number of times that the vocabulary word with index v appears in the document.

The key constraint with BoW representations is that each input feature must directly correspond to one human-readable unigram in a finite vocabulary.

That said, you have many design decision to make when applying a BoW representation:

- How big is your vocabulary?
- Do you exclude rare words (e.g. appearing in less than 10 documents)?
- Do you exclude common words (like 'the' or 'a', or appearing in more than 50% of documents)?
- Do you keep the count values, or only store present/absent binary values?

You are *strongly encouraged* to take advantage of the many tools that sklearn provides related to BoW representations:

- User Guide for Bag of Words tools in `sklearn.feature_extraction.text`
- sklearn.feature_extraction.text.CountVectorizer

# Goals and Tasks for Problem 1

For Problem 1, you will develop an effective BoW representation plus binary classifier pipeline, aiming to produce the best possible performance on heldout data.

You should experiment with several possible ways of performing BoW preprocessing.

You should use only a LogisticRegression classifier for this problem.

You should use best practices in hyperparameter selection techniques to avoid overfitting and generalize well to new data. Within your hyperparameter selection, you should use cross-validation over *multiple* folds to assess the range of possible performance numbers that might be observed on new data.

Your report should contain the following sections:

## 1A : Bag-of-Words Design Decision Description

Well-written paragraph describing your chosen BoW feature representation pipeline, with sufficient detail that another student in this class could reproduce it. You are encouraged to use just plain English prose, but you might include a brief, well-written pseudocode block if you think it is helpful.

You should describe and justify all major decisions, such as:

- how did you "clean" the data? (handle punctuation, upper vs. lower case, numbers, etc)
- how did you determine the final vocabulary set? did you exclude words, and if so how?
- what was your final vocabulary size (or rough estimate of size(s), if size varies across folds because it depends on the training set)?
- did you use counts or binary values or something else?
- how does your approach handle out-of-vocabulary words in the test set? (it's fine if you just ignore them, but you should be aware of this)

## 1B : Cross Validation Design Description

Well-written paragraph describing how you use cross-validation to perform both classifier training and any hyperparameter selection needed for the classifier pipeline.

For Problem 1, you *must* use cross validation with at least 3 folds, searching over at least 5 possible hyperparameter configurations to avoid overfitting.

You should describe and justify all major decisions, such as:

- What performance metric will your search try to optimize on heldout data?
- How did you execute CV? How many folds? How big is each fold? how do you split the folds?
- What off-the-shelf software did you use, if any?
- After using CV to identify a selected hyperparameter configuration, how will you then build one "final" model to apply on the test set?

## 1C : Hyperparameter Selection for Logistic Regression Classifier

Using your BoW preprocessing, plus a logistic regression classifier, your goal is to train a model that achieves the best performance on heldout data.

Here, we ask you to use a LogisticRegression classifier, and identify a concrete hyperparameter search strategy. Which hyperparameters are you searching? What concrete grid of values will you try?

Your report should include a figure and paragraph summarizing the design of this search as well as the results. Please follow the hyperparameter selection rubric.

## 1D : Analysis of Predictions for the Best Classifier

In a figure, show some representative examples of false positives and false negatives for your chosen best classifier from **1C**. Be sure to look at *heldout* examples (not examples used to train that model). It's OK to analyze examples from just one fold (you don't need to look at all K test sets in CV).

In a paragraph caption below the figure, try to characterize what kinds of mistakes the classifier makes. Do you notice anything about these sentences that you could use to improve performance? (You can *apply* these ideas later in Problem 2).

You could look at any of these questions:

- does it do better on longer sentences or shorter sentences?
- does it do better on a particular kind of review (amazon or imdb or yelp)?
- does it do better on sentences without negation words ("not", "didn't", "shouldn't", etc.)?

### 1E : Report Performance on Test Set via Leaderboard

Create your "final" classifier using the selected hyperparameters from **1C**. Apply your classifier to each test sentence in `x_test.csv`. Store your *probabilistic* predictions into a single-column plain-text file `yproba1_test.txt` (remember, we'll use AUROC as the metric to decide your rank on the leaderboard). Upload this file to our bag-of-words leaderboard.

In your report, include a summary paragraph stating your ultimate test set performance, compare it to your previous estimates of heldout performance from cross-validation, and reflect on any differences.

# Problem 2: Open-ended challenge

## Goals and Tasks for Problem 2

For this problem, your goal is to obtain the best performance on heldout data, using *any* feature representation you want, *any* classifier you want, and any hyperparameter selection procedure you want.

Here are some concrete examples of methods/ideas you could try:

- Instead of only using single words ("unigrams"), as in Problem 1, can you consider some bigrams (e.g. 'New York' or 'not bad')?
- Can you use smart reweighting techniques like term-frequency/inverse-document-frequency? See sklearn.feature_extraction.text.TfIdfVectorizer
- Try a *different* classifier in sklearn (nearest neighbor, random forest, MLP, etc.). Be sure you understand enough about this classifier to define a reasonable hyperparameter search strategy.
- Would it help to build separate classifiers for amazon, imdb, and yelp reviews?
- Would it help build features for the first-half and second-half of sentences?
- Can you use text compression methods to obtain features?
- Can you use off-the-shelf neural representations of text, like word2vec or GloVe or BERT?

We expect to see the paragraphs and figures described below, that mirror the criteria above for 1A (overall design of feature representation), 1B (overall CV experimental strategy), 1C (hyperparameter search strategy for chosen classifier), and 1D (performance analysis).

For full credit, we expect that at least 2 parts out of 2A, 2B, and 2C explore substantially different methods than those used in Problems 1A, 1B, 1C. Each choice must be plausibly motivated by improving your classifier's performance.

### 2A : Feature Representation description

Include a paragraph describing and justifying how you transformed text into fixed-length feature vectors suitable for classification. Include enough detail that another student could roughly reproduce your work.

If this is substantially similar to 1A, it is OK to say so and keep this paragraph brief (you don't need to repeat yourself).

### 2B : Cross Validation (or Equivalent) description

Include a paragraph describing and justifying how you set up your training and hyperparameter selection process, given only the provided training set. Include enough detail that another student could roughly reproduce your work.

If this is substantially similar to 1B, it is OK to say so and keep this paragraph brief.

### 2C : Classifier description with Hyperparameter search

Include a paragraph describing and justifying which classifier you selected, how you trained it, and how you performed hyperparameter search (including concrete values you explored). Include enough detail that another student could roughly reproduce your work.

Include a figure that shows how classifier performance changes over at least one hyperparameter. Please follow the hyperparameter selection rubric.

### 2D : Error analysis

In a figure, show some representative examples of false positives and false negatives for your chosen best classifier from **2C**. Be sure to look at *heldout* examples (not examples used to train that model).

In a paragraph caption below the figure, try to characterize what kinds of mistakes the classifier makes. Reflect on any key differences from the classifier in Problem 1.

### 2E : Report Performance on Test Set via Leaderboard

Apply your best pipeline from **2A - 2D** above to the test sentences in `x_test.csv`. Store your *probabilistic* predictions into a single-column plain-text file `yproba1_test.txt`. Upload this file to our Open-Ended leaderboard.

In your report, include a summary paragraph stating your ultimate test set performance. Discuss if your performance is better or worse than in Problem 1, and reflect on why you think that might be.

# Rubric for Overall Performance

We'll get a final number for this project by averaging:

- 87% : your report performance, using the rubric below
- 10% : your leaderboard submissions, using the rubric below
- 3% : completion of your reflection on the project

# Rubric for Evaluating Leaderboard Submissions

You'll submit 2 sets of predictions to our leaderboard (one each for Problem 1 and 2)

For each one, we'll give you a score between 0.0 and 1.0 where:

- 85% of points represent if you achieved a "reasonable" score (e.g. a standard pipeline trained using good practices)
- 15% of points awarded if you are within tolerance of the top 3 submissions in this class (partial credit possible, linearly interpolating between the "reasonable" score and the "top" score).

# Rubric for Evaluating PDF Report

Earning full credit on this assignment requires a well-thought-out report that demonstrates you made reasonable design decisions for feature preprocessing and classifiers and followed machine learning best practices throughout, especially for hyperparameter selection. Achieving top-scores on the leaderboard is far less important than understanding *why* some methods and choices outperform others.

Points will be allocated across the various parts as follows:

- 60%: Problem 1
- 40%: Problem 2

Within each problem, we break down the points like this

- 30%: Paragraph A on Feature representation design decisions
- 15%: Paragraph B on cross validation design decisions
- 35%: Paragraph C on training and selection for your classifier
- 15%: Paragraph D on analysis of classifier mistakes/successes
- 5%: Paragraph E reflection on heldout performance

# Hyperparameter Selection Rubric

## Figure Requirements:

Your figure should show heldout performance a range of at least 5 possible hyperparameter values controlling *model complexity* that cover both underfitting and overfitting. That is, if at all possible, at least one candidate value should show clear underfitting and at least one should show clear overfitting.

Your figure should:

- Show both training set and validation set performance trends in the same plot. Make sure you add a legend, label your axes, add a title.
- Show the *typical* performance at each hyperparameter via the *average* over multiple CV folds

An *ideal* figure will also

- Communicate *uncertainty* around this typical value, by exposing the variation across the multiple CV folds
- - A simple way to show uncertainty is show the empirical *range* across all folds, or the empirical standard deviation
- - A *better* way to do this is show a separate dot for the direct performance of each fold (so 5 dots for 5 folds).

The big idea here is that your figure should help the reader understand if one hyperparameter is definitely better than another (e.g. performance is better on most or all folds) or if there isn't much difference.

## Paragraph requirements:

In each paragraph where you describe training a classifier and selecting its hyperparameters to avoid overfitting, you should include

- 1-2 sentences: describe the potential advantages of the chosen classifier for the task at hand.
- 1-3 sentences: describe any necessary details about the training process (e.g. are there convergence issues? step-size selection issues? should you stop early to avoid overfitting?)
- 1-2 sentences: describe which model complexity hyperparameter(s) were explored, how these values control model complexity, and why the chosen candidate value grids (or random distributions) are reasonable to explore the transition between under and over fitting and find the "sweet spot" in-between.
- 1-2 sentences: describe the results of the experiment: which hyperparameter is preferred? is the evidence decisive, or uncertain?

For better readability, please **bold** the key factual takeaway for each of the 4 items above. For example,

> *Our 10-fold CV search for logistic regression searched the c hyperparameter, an L2-regularization penalty on the weights, across **20 log-spaced values between 10^-6 and 10^6***

## General tips for Figures

Please do your best to keep figures close to the related paragraph, ideally on the same page.

If a figure contains multiple elements such as multiple lines or multiple sets of bars, please make sure that they are on the same scale. Alternatively, if different scales are necessary, adjust them appropriately to maintain reasonable and easily interpretable trends. For instance, it is inappropriate to represent a line fluctuating between 0 and 1 on the same scale as a line oscillating between 100 and 1000, as this would distort the representation and interpretation of the data points.

---