Machine Learning Class Project: Sentiment Classification

1 Introduction

The task of sentiment classification requires predicting whether some text has a positive or a negative sentiment. This task is a well studied natural language processing task, and can be used to inputs as diverse as product or movie reviews (to automatically ascertain whether the reviewer likes the movie/product or not), news or social media text about famous people or companies (to ascertain how public opinion changes about them) and psychotherapy transcripts (to track how the patient's affective state is changing during a therapy session).

In this project, we will be working with the well-studied *Movie Review Dataset*, which consists of a collection of reviews from *Rotten Tomatoes*. The dataset was originally introduced by Pang and Lee (2005) and has since become a standard benchmark to study the problem of text classification.

- Pang, Bo, and Lillian Lee. "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales." In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), pp. 115-124. 2005.

2 Task definition

Your goal for this project is to explore different classifiers that predict the sentiment of a featurized movie review. The instances for classification are movie reviews, and the labels are either *positive* (denoted by 1) or *negative* (denoted by 0).

We provide three different feature representations of the data, and you may use some or all of them (and also optionally mix-and-match), and the different learning algorithms we see in class. Through the semester, you will be submitting predictions of various classifiers to Kaggle. The mechanics of the project are described in the last section.

3 Data

Each example for classification is a single movie review. For example, we may have a review such as:

> *writer-director burger imaginatively fans the embers of a dormant national grief and curiosity that has calcified into chronic cynicism and fear .*

This review is rated as a *positive* one. Two important points to note:

1. We do not provide the raw text of the reviews and instead only give you feature representations.
2. We do not use the standard train/test splits of the data and instead construct our own splits. One side effect of this is that you cannot use data that is not provided to you.

3.1 Data splits

In this project, you will be working with 10,130 examples that have been split into three parts described below:

1. `train`: This is the training split of the data, on which you will be training models after hyper-parameter tuning. We have not split the training data into multiple folds for cross-validation; we expect you to do that on your own. The training split consists of 7,089 examples.

2. `test`: This is the test set that you can use to evaluate your models locally. The test set consists of 1,520 examples.

3. `eval`: This is the "evaluation" set with 1,521 examples. We have hidden the labels for these examples. The idea is that you use your models to make a prediction on these examples, and then upload the predictions to Kaggle, where your model's performance will be ranked on a leaderboard. Kaggle uses a random half these examples for a public leaderboard that will be visible to everyone in class, and the other half for a private leaderboard that is only visible to the instructors.

3.2 Feature representations of the reviews

Instead of having you work with the raw text directly, we have pre-processed the data and a few different feature sets. All features are provided to you as `.csv` files.

1. `tfidf`: The tfidf representation of text is short for *term frequency–inverse document frequency*, and is a popular document representation that seeks to reflect how important is a word to a document in a corpus. It improves on a simpler bag of words representation by weighting each word in the bag-of-words such that frequent words are weighted lower. To avoid the dimensionality from becoming very large, we have restricted the tfidf features to those from the 5000 most frequent words.

2. `spacy-embeddings`: The tfidf representation is usually very sparse, with only a small fraction of the 5,000 features being non-zero for any vector. The spacy-embeddings contain dense 300 dimensional vectors where each document is represented by the average of its "word embeddings". Word embeddings are vectors that are trained to capture word meaning. The specific word embeddings we use here were obtained using a popular NLP library called `spaCy`.

3. `roberta`: The roberta embeddings are obtained using a very popular transformer-based language model called RoBERTa. As is the case with the spacy-embeddings, these are also dense but the vectors here are 768 dimensions long. These are obtained from the `[CLS]` token of the last layer of the roberta language model.

The description of the three feature representations here is deliberately brief. If you would like to know more about these representations, feel free to explore the corresponding Wikipedia articles which do a reasonable job of describing them, or use the office hours to discuss them. For the purpose of this class, you can think of these as three different feature representations of the same data.

You are welcome to use combinations of these features or try out feature space expansions, neural networks of various kinds and other non-linear methods.

3.3 Data files

We have three data splits (`train`, `test`, `eval`), and four different kinds of features (`tfidf`, `spacy-embeddings`, `roberta`). Each of these are available in a different file. All features are in the csv format.

All the data is available in the `data` directory, organized as follows:

Filename	Feature	Data split
`data/tfidf/tfidf.train.csv`	`tfidf`	`train`
`data/tfidf/tfidf.test.csv`		`test`
`data/tfidf/tfidf.eval.anon.csv`		`eval`
`data/spacy-embeddings/spacy-embeddings.train.csv`	`spacy-embeddings`	`train`
`data/spacy-embeddings/spacy-embeddings.test.csv`		`test`
`data/spacy-embeddings/spacy-embeddings.eval.anon.csv`		`eval`
`data/roberta/roberta.train.csv`	`roberta`	`train`
`data/roberta/roberta.test.csv`		`test`
`data/roberta/roberta.eval.anon.csv`		`eval`

In addition, the directory also contains a file called `data/eval.ids`. This file has as many rows as the `data/*.eval.anon` files. Each line consists of an example id, that uniquely identifies the evaluation example. The ids from this file will be used to match your uploaded predictions on Kaggle.

4 Evaluation

Since the data is constructed to be balanced, we will use standard accuracy to evaluate classifiers.

The examples are all split randomly among the three splits. So we expect that the cross-validation performance on the training set and the accuracy scores on the test set and the public and private splits of the evaluation set will be similar.

5 Submission format

Kaggle accepts a csv file with your predictions on the examples in the evaluation data. There should be a header line containing `example_id,label`. Each subsequent line should consist of two entries: The example id (from the file `data/eval.ids`) and the prediction (0 or 1).

We have provided two sample solutions for your reference:

1. `sample-submissions/all-positive.csv`: Where all examples are labeled as positive.
2. `sample-submissions/all-negative.csv`: Where all examples are labeled as negative.

6 Project rules

You should work *individually* on the project.

You cannot sign up to Kaggle from multiple accounts and therefore you cannot submit from multiple accounts.

You may submit a maximum of 4 entries per day.

The end date for the project is the day of the final exam, i.e., **May 3, 2023 11:59 PM Utah time**.

You should to submit *at least* six different non-trivial submissions to Kaggle. Here are the rules for these submissions:

1. You should use at least two feature sets. For example, you could train a classifier on one feature set, and the same classifier on a different one. These would count as different submissions. You are free to use the feature representations provided by us, or construct your own.
2. You should use at least four *different* learning algorithms we see in class. You cannot use any machine learning library for these five algorithms, and instead implement them by yourself.
3. For at most one of your six submissions, you are welcome to use a machine learning library such as PyTorch, TensorFlow or `scikit-learn`.

6.1 Milestones

The project is organized into several milestones summarized below. The deadlines and submissions will be managed via Canvas.

1. **Project information** (10 points): You will need to have registered for the Kaggle competition and made a dummy submission. On canvas, you should also submit your kaggle user details and the score you get for the dummy submission.
2. **Project checkpoint 1** (15 points): For this milestone, you will need to have downloaded the data, and also perhaps run some initial pre-processing on it. You should also have made at least one non-dummy submission on Kaggle. You should submit a one page report on Canvas that describes what you did so far, descriptive statistics about the dataset, and your plan till the next milestone.

3. **Project checkpoint 2** (30 points): This milestone is similar to the previous one. You will need to have made at least two additional submissions to Kaggle. You should submit a one-page report detailing updates after the first milestone, any challenges you have faced, and your plan for the rest of the semester.

4. Final report (45 points): By this time, you should have made at least six non-dummy submissions on Kaggle totally. You should submit a final report of at most six pages that is structured like a small research paper. Broadly speaking it should describe:

   (a) An overview of the project.

   (b) What are the important ideas you explored?

   (c) What ideas from the class did you use?

   (d) What did you learn?

   (e) A summary and discussion of results

   (f) If you had much more time, how would you continue the project?

   Each of these components will be equally weighted in the report grade.