Practical Assignment 1

COMP 451 - Fundamentals of Machine Learning

Prof. William L. Hamilton

Winter 2021

Preamble

- The assignment is due February 23rd at 11:59pm via MyCourses. Late work will be automatically subject to a 20% penalty, and can be submitted up to 5 days after the deadline.
- You should submit your assignment as a single zip file with the filenames indicated in the assignment README.
- You may consult with other students in the class regarding solution strategies, but you must list all the students that you consulted with on the first page of your submitted assignment and you should not share code. You may also consult published papers, textbooks, and other resources, but you must cite any source that you use in a non-trivial way (except the course notes).
- You must write your code yourself and be able to explain the solution to the professor, if asked.
- Python is strongly recommended, but you might write your solution in Java, C, or MATLAB if you prefer. The solution and autograding script will be in Python.
- You may use array processing and matrix libraries (e.g., numpy), but you may not use machine learning libraries (e.g., sklearn or PyTorch) in your solution.

Overview of the assignment

In this assignment, you will implement logistic regression and Naive Bayes models for sentiment analysis. The dataset you will work with is based on the IMDB Sentiment Classification Dataset (`https://ai.stanford.edu/~amaas/data/sentiment/`). This dataset contains positive and negative movie reviews from IMDB, and the goal is to classify whether a review is positive or negative based on the words used. We will provide a pre-processed version of this dataset, where each review is represented as a list of binary features, indicating which words are present in that review. You may examine the original dataset (linked above) to see the original text reviews.

Getting started

In order to get started for the assignment, please follow these steps:

1. Download the files at `https://cs.mcgill.ca/~wlh/comp451/files/practical_assn_1.zip`.
2. Read the README.md file in detail.

3. Make sure you can open/access the `train_dataset.tsv` and `validation_dataset.tsv` files. These are the train and validation dataset, in tab separated format. Each row in these files is a different data point. The first 1000 columns correspond to binary features and the last column corresponds to the target value for the data point.

4. If you are using Python, we suggest you use the numpy package to load the data into arrays (e.g., `data = numpy.loadtxt(train_dataset.tsv, delimiter="\t")`.

5. If using Python and numpy, following step 4 above, you can separate the features and targets as follows:

   - `X = data[:,:-1]`, which selects all the rows and all the columns except the last one.
   - `y = data[:,-1]`, which selects all the rows and only the last column.
   - Now each row `X[i]` corresponds to a vector of binary features for an example `i` and the corresponding entry in `y[i]` is the label for that example.

Task 1: Implement Naive Bayes [12 points]

Your first task is to implement a Bernoulli Naive Bayes model.

Requirements

- You must implement the Naive Bayes Algorithm as described in Chapter 5 of the class notes.
- Do not use smoothing in your implementation.
- In your implementation, you should use maximum likelihood to estimate the class prior `theta_1` (i.e., $\theta_1 = \hat{P}(y = 1)$), as well as the 2000 distinct feature likelihoods `theta_j_k` (i.e., $\theta_{j,k} = \hat{P}(\mathbf{x}[j] = 1 | y = k)$ for all $j = 0, .., 999$ and $k = 0, 1$).
- In order to facilitate autograding you must generate three `tsv` files (examples of these files are included in the assignment files):
  1. `class_priors.tsv`, which contains two lines. The first line should have your estimated value for θ_1 and the second line should contain $1 - \theta_1$.
  2. `negative_feature_likelihoods.tsv`, with 1000 lines where each line contains a $\theta_{j,0}$ value for $j = 0, ..., 999$. The order of the values in this file should match the ordering of the features in the `train_data.tsv` file. In other words, this file contains the feature likelihoods for the negative (i.e., $y = 0$) class.
  3. `positive_feature_likelihoods.tsv`, with 1000 lines where each line contains a $\theta_{j,1}$ value for $j = 0, ..., 999$. The order of the values in this file should match the ordering of the features in the `train_data.tsv` file. In other words, this file contains the feature likelihoods for the positive (i.e., $y = 1$) class.
- You can run the `autograder_test_NB.py` script to test if your files are formatted correctly. This script will also output your accuracy on the validation set.

Implementation suggestions

- We suggest that you use a Python class to represent your Naive Bayes model, using a dictionary of attributes to store the parameters of the model.
- We suggest that you implement a `fit(X, y)` function for your Naive Bayes class, which takes the training data as input and sets the model parameter attributes as a side effect. You should use the maximum likelihood expressions from class to implement this function. You will need to iterate over the training data (or use array-based arithmetic) to compute some count statistics.

- We suggest you implement a `predict(X)` function, which takes an matrix containing n training examples and outputs n predictions for the estimated $\hat{y}$ values, using the currently stored model parameters. You should use the log-likelihood ratio to implement this function.
- We suggest you implement a `save_parameters(directory_name)` function for your Naive Bayes class, which saves the learned model parameters in the specified directory.
- We suggest you implement an evaluation script to test the accuracy of your model, rather than relying completely on the autograding script.

Task 2: Implement Logistic Regression [12 points]

Your second task is to implement a logistic regression model.

Requirements

- You must implement the logistic regression as described in Chapter 6 of the class notes.
- In your implementation, you should use gradient descent to estimate the model parameters (i.e., the $\mathbf{w}$ vector). You must use full-batch gradient descent and not stochastic gradient descent. You must experiment with different learning rates and use sufficient iterations of gradient descent so that your model converges (up to a tolerance of roughly $\epsilon = 0.0005$).
- In order to facilitate autograding you must generate one tsv file `weights.tsv` that contains 1001 lines, with each line corresponding to a different feature coefficient and where the first line corresponds to the bias term.
- You can run the `autograder_test_LR.py` script to test if your files are formatted correctly. This script will also output your accuracy on the validation set.

Implementation suggestions

- We suggest that you use a Python class to represent your logistic regression model, using a dictionary of attributes to store the parameters of the model.
- We suggest that you implement a `fit(X, y, max_iters=5000, lr=0.01, tol=0.0005)` function for your logistic regression class, which takes the training data as input and sets the model parameter attributes as a side effect. You should use gradient descent to learn the parameters and we suggest using the default hyperparameters noted above for the maximum number of gradient descent iterations (`max_iters`), the learning rate (`lr`), and the tolerance ϵ for stopping (`tol`).
- We suggest you implement a `predict(X)` function, which takes an matrix containing n training examples and outputs n predictions for the estimated $\hat{y}$ values, using the currently stored model parameters. You should logistic function with a threshold of 0.5 to implement this function.
- We suggest you implement a `save_parameters(directory_name)`, which saves the learned model parameters in the specified directory.
- We suggest you implement an evaluation script to test the accuracy of your model, rather than relying completely on the autograding script.

Rubric

We will use a held-out test set to evaluate your models. However, your performance on the validation set should be a good indicator of your test set performance. The rubric for grading both the Naive Bayes and logistic regression implementations are as follows:

- 0-2 points are awarded if the solution does not run in the autograder. Up to 2 points can be obtained if the code contains a non-trivial attempt at the implementation.
- 2-6 points are awarded if the implementation runs but achieves a test accuracy of less than 70%. The TA will award discretionary points based on the quality of the code and the nature of the error in the implementation.
- 7-8 points will be awarded if the implementation achieves a test accuracy of greater than 70% but that is lower than the accuracy of our reference implementation by 5% or more. Note that both the Naive Bayes and logistic regression models have unique optimum values, so your solution should exactly match the reference implementation if it is properly done. The TA will award discretionary points based on the quality of the code and the nature of the error in the implementation.
- 9-10 points will be awarded if the implementation achieves a test accuracy of greater than 70% but that is lower than the accuracy of our reference implementation by 2% or more. Note that both the Naive Bayes and logistic regression models have unique optimum values, so your solution should exactly match the reference implementation if it is properly done. The TA will award discretionary points based on the quality of the code and the nature of the error in the implementation.
- 11-12 points will be awarded if the implementation achieves a test accuracy that is equivalent to our reference solution (with a tolerance of $\pm 1\%$). Note that both the Naive Bayes and logistic regression models have unique optimum values, so your solution should exactly match the reference implementation if it is properly done. The TA will award discretionary points based on the quality of the code; full points will be awarded as long as there are no major issues with the code (e.g., obvious and extreme inefficiencies).

Note: This is not a software engineering course, and we will not be harsh when grading code. You may lose points if your code is confusing to the point of being unreadable (e.g., no meaningful variable names, confusing or unnecessary use of data structures) or if your implementation is obviously orders of magnitude less efficient than it could be (e.g., contains unnecessary nested loops over the training data). We will not provide detailed feedback on the code quality (e.g., detailed suggestions of how to fix the code).