E0 270: Assignment 1

Due date: March 14, 2024

Instructions

1. Download and extract the assignment template. Download your test set after providing your 5 digit SR number from http://10.192.30.174:8000/download and move it inside the data folder of the assignment template. You are allowed to change the template code according to your requirements.

2. Setting up the environment, running into technical problems, and figuring out their solutions is a part of the learning process. Use Google, Stack overflow, and discuss with each other. Unfortunately, we will not be able to help you.

3. The report must be typeset in LaTeX, Ensure that everything is in a single pdf file and all pages are in correct order.

4. If you feel any particular problem in under-specified, you can safely assume that it has been done intentionally. You are free to make assumptions, but please specify the assumptions in your report. No further clarifications on the problem will be provided. However feel free to contact us if you believe that there is an error in the problem specification.

5. The grading will be relative based on the final performance achieved by your model. You are encouraged to discuss the problems amongst yourselves and engage in a healthy competition. However copying codes from others verbatim, along with the use of LLM assisted code generators are strictly forbidden, and will be penalized heavily if detected.

Data

Consider the Sentiment140 dataset https://huggingface.co/datasets/stanfordnlp/sentiment140. Each data point is a sentence, whose sentiment is the label. Since the sentences are a tweet, the maximum length is 140 characters, but also the tweets contain multiple abbreviations, urls, twitter handle references, emojis, and so on. We have preprocessed the text and removed multiple confounding aspects from the text such as 'stop-words' (https://en.wikipedia.org/wiki/Stop_word), urls and handles. We additionally modified the text by stemming and lemmatizing (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html) each word in the text. After the preprocessing, we removed all blank texts from the corpus thus obtained, and finally got a text corpus of about 1.593 million tweets and their corresponding sentiments. The sentiments were rated from 1 to 5 by human annotators. To simplify the problem setting, we have converted the sentiments to binary, i.e. currently the labels are 0 and 1 corresponding to negative and positive sentiments respectively.

Multinomial Naive Bayes

The Naive in Naive Bayes suggests that all features i.e. words in a given sentence, are independent from each other. You are required to implement a Multinomial Naive Bayes classifier, which is

used to classify the text based on aspects such as how many times a word appears in a sentnce (or associated class conditional probabilities). Since the presence of a word doesn't affect the presence of another word makes the model easy to use and update.

1. Browse through the template code that you have been provided. Specifically, look at files `prob1.py` and `utils.py`. Fill up the required sections of the code in `utils.py` (marked with TODO / NotImplementedError). You are supposed to implement a 'Count Vectorizer' (https://github.com/yashika51/Understanding-Count-Vectorizer) which takes a sentences as input and converts each sentence into a k-dimensional vector with each dimension corresponding with a word in the vocabulary, thus each dimension in the vector will contain the count of the corresponding word in the sentence.
   Caution: Please note that since the vocabulary is being considered for the entire corpora (even after taking say the top 10K words), only about $0.001 - 0.002\%$ dimensions will be filled in expectation. Since the remaining dimensions will just be 0s (indicating absence of corresponding words), you can make the computations and storage efficient by taking sparse matrices (https://docs.scipy.org/doc/scipy/reference/sparse.html). Otherwise you will get memory overflow errors.

2. Now that we have converted words into sentences, we can start applying our known machine learning algorithms on this. We will be using a generative model: Multinomial Naive Bayes (as mentioned earlier). You are supposed to modify the relevant sections in the `model.py`. Ignore (for now) the `update` parameter in the fit function.
   Hint: You can save the class conditional means for each of the features separately and consider a Bernoulli distribution by normalizing the means across classes. Remember to take store the sum of the means for inference purposes. During inference, you can find the likelihood of the new data belonging to each class and compare them to decide the class.
   Once you done implementing the code for the (naive) Bayes-optimal classifier, you can check accuracy by executing: `python prob1.py --sr_no <5 digit SR#>`
   You should get $> 75\%$ validation and test accuracy once the implementation is correct.

Bonus: You may consider implementing a TF-IDF vectorizer (https://en.wikipedia.org/wiki/Tf-idf) to improve your model's performance.

Submit the predictions generated by your best model on http://10.192.30.174:8000/submit to check your standings amongst your peers on the leaderboard (http://10.192.30.174:8000).

Active Learning

Data is easy to obtain, however corresponding annotations are often much more expensive. Human annotators are often paid by the hour to annotate a data using tools such as Amazon Mechanical Turk (https://www.mturk.com/). So, think of a situation where we let our model decide which subset of data it wants to get annotated.
Active learning is such an approach where the model selectively queries the most informative data points for labeling. Thus the annotation cost can be concentrated on only the recommended samples which can give the model more information, and not arbitrary samples which might provide redundant information. The goal is to improve model performance with fewer labeled samples by prioritizing examples that maximize learning efficiency, thus reducing annotation effort and cost.
Given an unlabeled set of data points $\mathcal{D}_{\text{Unl}}$ and a small set of labeled points $\mathcal{D}_{\text{Lab}}$, the agent has to choose which points in $\mathcal{D}_{\text{Unl}}$ to label, in the context of what is already available as labeled data. More precisely, consider a supervised learning algorithm Alg that takes a labeled dataset $\mathcal{D}_{\text{Lab}}$ and

returns a trained model $\pi_{\mathsf{Alg}}(\mathcal{D}_{\text{Lab}})$. Further, any model π has to finally perform well on some unknown distribution, quantified by some performance measure V^{π}. At each stage, the goal of the active learning agent is to choose a point $\tau \in \mathcal{D}_{\text{Unl}}$ to label so as to optimize V of the model learned on this new dataset, i.e.,

$$\tau^* = \arg\max_{\tau \in \mathcal{D}_{\text{Unl}}} V^{\pi_{\mathsf{Alg}}(\mathcal{D}_{\text{Lab}} \cup \{\tau\})}.$$

1. We relax the above constraints and assume that we can choose c datapoints at each iteration of Active Learning Strategy (ALS) for data selection. Go through the code in `prob2.py` and fill the required code starting at line 52. Consider a strategy to quantify uncertainties for the model. A simple measure of uncertainty is entropy, but one can also consider gini impurity (https://www.geeksforgeeks.org/gini-impurity-and-entropy-in-decision-tree-ml/). Note that the more the model is uncertain about a given datapoint, the more informative labelling the data will be.

2. Now notice that the class conditional feature wise counts/means in the Naive Bayes model is essentially a sufficient statistic for the previous dataset. For a new given dataset, one doesn't need to recompute the metrics after merging the dataset with the existing code. Can you find a way to combine the existing statistics with the statistics of the newly labelled datapoints? Modify the `model.py` with the case where the model is not being trained from scratch but simply updated with newly labelled datapoints. Correspondingly update the required lines in **prob2.py** where instead of appending new datapoints to the existing datapoints in order to obtain a bigger labelled dataset in `X_train_batch` and `y_train_batch` you can simply use the new datapoints to update your model in a much more computationally efficient manner.

3. Run the code 10 times by executing:
   `python prob2.py --sr_no <5 digit SR#> --run_id <int> [--is_active]`
   specifically 5 times with random selection strategy and 5 times with ALS, each with run ids $1, 2, 3, 4$ and 5. Then complete `plot.py` and run:
   `python prob2.py --sr_no <5 digit SR#> --supervised_accuracy <float>`
   to generate the required plots that contains comparison between the random strategy and ALS. You need to plot the mean and the standard deviation across 5 runs with different random seeds for each seed. Also plot a horizontal line on top to denote the supervised baseline, i.e. how much accuracy the model attains by using 100% of the labelled dataset.

4. Also modify the `prob2.py` to measure the time difference between the original model retraining the model updating and plot the times to show that the original method scales quadratically with the increase in data, but the modified method scales linearly.

Deliverables

- Completed code for implementing the Multinomial Naive Bayes Classification algorithm without using any libraries except `numpy`/`scipy` in the given incomplete code snippets.
- Completed code for the Active Learning and associated fine-tuning.
- A report containing the complete details of the algorithm and your implementation, along with justification for any extra assumptions made if necessary. Since the dataset is perfectly balanced among all classes, reporting just the negative log likelihoods and accuracies should be sufficient.

- Training plot corresponding to accuracies for both train and validation set for the active learning experiment. Also mention your best test accuracy obtained on the leaderboard along with the reduction in (percentage of data) data required to reach supervised baseline performance using ALS for data selection. Also give the training time comparison plot.

Submission

Attach **a single zip file** named in the format `Asst1_FirstName_LastName_5DigitsOfSRNo.zip` to the assignment in Teams, before the due date. The zip file should contain your code that you filled up, your report (in a pdf file named `Report_<5digitSR>.pdf`), and any plots that you put in the report. Ensure to upload the zip file on Teams submission portal on or before March 14, 2025, 11:59 pm. Ensure that you do **not** include the data files and **pycache** files in the code that you submit.