Lab 3: Naive Bayes & Logistic Regression

Introduction

In this lab session, you'll delve into the realm of text classification, specifically focusing on movie review sentiment analysis. You will develop a system capable of classifying movie reviews into positive or negative sentiments. This system will be powered by two fundamental machine learning models: Naive Bayes and Logistic Regression.

You'll work with a dataset of movie reviews from Imdb[1], learning to process and understand textual data. The goal is to teach your models to automatically determine the sentiment of a review. Here's a glimpse of the reviews you'll analyze:

- "An enthralling experience, thoroughly enjoyable." - Positive
- "It was a tedious and uninteresting movie." - Negative
- "A masterpiece of visual storytelling." - Positive
- "The plot was predictable and dull." - Negative

Unlike the previous practice where you created a character-level language model for name generation, this practice involves understanding text at a word level. You'll learn how to represent text data through techniques like Bag of Words and then use it to train your classification models.

Learning Outcomes

By the end of this lab, students will be able to:

- Transform text data into a numerical representation suitable for machine learning algorithms, particularly focusing on the Bag of Words model.
- Understand and apply the principles of Naive Bayes and Logistic Regression models for the purpose of sentiment analysis in text.
- Evaluate and interpret the performance of text classification models using various metrics such as accuracy, precision, recall, and F1-score.
- Develop an understanding of text classification concepts and challenges.

[1] You can find the database here

Deliverables

For the successful completion of this laboratory session, you are required to submit the following files:

1. `data_processing.py` - This Python file contains all the functions related to data loading, preprocessing, and and bag-of-words encoding.
2. `naive_bayes.py` - This Python file should contain the `NaiveBayes` class with methods for training the model and classifying text data, along with the necessary functions for estimating class priors and conditional probabilities.
3. `logistic_regression.py` - This file must include the `LogisticRegression` class with functionalities for initializing weights, training the model, and predicting class labels or probabilities.
4. `utils.py` - This file shall have the `evaluate_classification` function to measure the performance of the classification models.

These files should be compressed into a single `.zip` archive and named with your student identifier. The `.zip` file must then be uploaded to the designated Moodle assignment task before the deadline.

Exercises

Exercise 1: Oh ____! Here we go again

In this exercise, you will develop the function `read_sentiment_examples()` in the `data_processing.py` file. This function will read from a `.txt` file and transform each line into a structured format suitable for machine learning models.

The steps to complete this function are:

1. Open the file, read each line, splitting the line into a sentence and its corresponding sentiment label.
2. Process each sentence by:
   (a) Tokenizing the sentence into words. You might want to use the `tokenize` function from the `utils.py` file, although it is not the optimal for Sentiment Analysis.
   (b) Optionally, perform further text preprocessing like lowercasing or removing punctuation (not mandatory in this lab, but useful in real-world scenarios).
3. Pair each tokenized sentence with its sentiment label to create `SentimentExample` objects.
4. Return the list of `SentimentExample` objects.

The `SentimentExample` class has been created for you and is stored in the `utils.py` file. You will use this function to later read both the `train` and `test` dataset from their namefellow files.

> Hint #1:
>
> The `.txt` files have the label separated by a tabulation (`"\t"`). Beware of multiple tabulations in a sentence.

Exercise 2: The Art of Vocabulary Construction

In this exercise, you'll develop the `build_vocab()` function. This function will take the preprocessed movie reviews and create a vocabulary - a mapping of words to unique indices. This vocabulary is essential for converting text data into a numerical format that machine learning models can understand and process.

The function should analyze the `SentimentExample` objects, extract all unique words across all examples, and then assign each word a unique index. If you are having a flashback, keep it to yourself, no one will believe you.

Exercise 3: And You Have my BoW!

In this exercise, you will create the `bag_of_words()` function. This function is fundamental in transforming text data into a numerical format that machine learning models can interpret. Specifically, you will convert lists of words (from movie reviews) into bag-of-words vectors using the vocabulary you've built.

A Bag of Words (BoW) representation involves counting the frequency of each word in the vocabulary within a given text or, alternatively, simply marking their presence or absence (binary representation). Your function should be capable of generating both these types of representations based on the `binary` parameter.

After you complete this function, you are now able to represent the sentence information in a numerical format that Machine Learning models can understand.

> Hint #2:
>
> Remember to exclude words that are not present in the vocabulary.

Exercise 4: (Prior) Knowledge is power

In this exercise, you'll implement the `estimate_class_priors()` method in the `NaiveBayes` class in the `naive_bayes.py`. This method lays the groundwork for your Naive Bayes classifier by calculating the prior probabilities of each class (positive and negative sentiments) in your dataset.

Your mission involves:

1. Computing the frequency of each sentiment class in your training dataset.
2. Determining the prior probability of each class by dividing its frequency by the total number of examples.

Exercise 5: Smoothing Things Over

In this exercise, you'll implement the `estimate_conditional_probabilities()` method in your `NaiveBayes` class. This method is all about understanding the likelihood of encountering specific words in each sentiment class, a key aspect of the Naive Bayes classifier.

Your task involves using the frequency of each word in the vocabulary with the frequency bag-of-words procedure for different sentiment classes. However, the twist here is using Laplace smoothing - a technique to prevent zero probabilities for unseen words.

The process includes:

1. Counting how often each word appears in the examples of each class.

2. Applying Laplace smoothing to ensure no word has a zero probability.
3. Storing and returning these probabilities in a structured format.

Exercise 6: Calculating the Odds, Naively

In this exercise, you'll tackle the implementation of the `estimate_class_posteriors()` method in the `NaiveBayes` class. This crucial function is at the heart of Naive Bayes classification, where the real magic of combining prior knowledge and word probabilities takes place.

Here's what you'll do:

1. Convert the provided `SentimentExample` into a Bag of Words (BoW) vector using your vocabulary.
2. Compute the log probabilities for each class by combining the BoW vector with the conditional probabilities of words and the class priors.
3. Return the log posterior probabilities for each class, which represent the likelihood of the given example belonging to each class.

Exercise 7: A Prediction is Worth a Thousand Words

In this exercise, you'll complete the `predict()` and `predict_proba()` functions in the `NaiveBayes` class.

First, you'll:

1. Calculate the log probability over all sentiment classes for a given review.
2. For the `predict()` function:
   - Return the class label that has the highest posterior probability.
3. For the `predict_proba()` function:
   - Return these probabilities, providing insights into the classifier's confidence and decision-making process.

Hint #3:

You might want to check the torch.argmax() function for the second step.

Exercise 8: Weight a Minute, Let's Initialize!

In this exercise, you'll implement the `initialize_parameters()` method of the `LogisticRegression` class from the `logistic_regression.py` file. This task is about setting the stage for your logistic regression model by initializing its weights.

In this function, you will:

1. Set a random seed for reproducibility, making sure that every run of your model starts with the same initial weights.
2. Initialize the weights for your logistic regression model as zero (or using any distribution you find suitable).

3. Ensure that the dimensions of the weights align with the number of features in your dataset, plus one for the bias.

Exercise 9: Squashing Expectations

In this exercise, you'll implement the `sigmoid` function in the `LogisticRegression` class. This function is the heart of logistic regression, transforming linear predictions into probabilities.

The `sigmoid` function, defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

squashes input values to lie between 0 and 1. Implement it using `torch` functions.

Exercise 10: You are Lossing it!

In this exercise, you'll implement the `binary_cross_entropy_loss` function.

Binary Cross-Entropy Loss, defined as

$$y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

quantifies the difference between the predicted probabilities and the actual labels. More generally, for N examples $(x_1, y_1), \ldots, (x_N, y_N)$, the average loss

$$-\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

In this expression:

- N is the number of samples.
- y_i is the true label of the i-th sample (either 0 or 1).
- $\hat{y}_i$ is the predicted probability of the i-th sample belonging to class 1.

Exercise 11: Finding the Perfect Fit

In this exercise, you'll delve into the fit method of the `LogisticRegression` class. This is where your logistic regression model learns from the data, adjusting its weights to make better predictions.

In this exercise, you'll:

1. Initialize the weights using the `initialize_parameters` method.
2. Iterate over your dataset, performing the following steps in each epoch:
   (a) Compute predictions using the current weights and the sigmoid function.
   (b) Calculate the loss using the binary cross-entropy formula.
   (c) Compute the gradient of the loss with respect to the weights.
   (d) Update the weights by taking a step in the direction opposite to the gradient, scaled by the learning rate.

3. Optionally, print the loss at regular intervals to monitor the training process.

Food for thought #1:

Now that you can fit a `LogisticRegression` model, how can you obtain which words have more influence to classify a review as positive? And as negative?

Exercise 12: Soft Probabilities, Hard Decisions

In this exercise, you'll implement the `predict` and `predict_proba` methods in the `LogisticRegression` class.

In `predict_proba`, you'll:

1. Apply the sigmoid function to compute the probability of each sample belonging to the positive class.
2. Return these probabilities, providing a nuanced view of the model's predictions.

Switching to `predict`, your task will be:

1. Use the probabilities from `predict_proba` and classify each sample.
2. Return class labels (0 or 1) based on a threshold, typically 0.5, turning probabilities into definitive predictions.

These methods are your final step in the logistic regression journey, turning raw data into actionable insights.

Exercise 13: Measuring Success

In this exercise, you will implement the `evaluate_classification` function from the `utils.py` file. This function is your tool for quantitatively assessing the performance of your sentiment analysis model.

Your mission is to calculate key metrics:

- Accuracy: The proportion of total predictions that were correct.
- Precision: The proportion of positive identifications that were actually correct.
- Recall: The proportion of actual positives that were identified correctly.
- F1-Score: The harmonic mean of precision and recall, balancing both metrics.

By computing these metrics, you'll gain insights into the strengths and weaknesses of your model, guiding your efforts to improve its performance.

Test your work!

After implementing each function, it's crucial to validate your work. We have prepared test functions for each required function. These tests will help ensure that your implementations meet the specified requirements. If a test function returns `None`, it indicates that the corresponding function might not be completed and the test is skipped. It is essential to pass all tests to ensure the functionality and accuracy of your code.

Additionally, a `main.py` file is provided. This file runs the necessary functions to predict the sentiment of the `test` reviews based on your implementations. Moreover, it gives you the classification metrics for both Naive Bayes and Logistic Regression. After completing all exercises, run `main.py` to see your NLP model in action.