Machine Problem 4

Naive Bayes Classifier for Handwritten Digits

Kislaya Joshi (**kj276**) and Shreya Bajpai (**sb1437**)

A. Implementing a Naive Bayes Classifier

1. Our code includes our basic feature extractor `extract_basic_features`(*digit_data, width, height*)
2. Given the large number of features, if we use the standard multiplicative representation of joint probability, there can be drawbacks. The drawbacks is that if you multiply many probabilities, then underflow might result.
3. We opt to take the natural logarithm of both sides in our implementation of `compute_statistics`(*data, label, width, height, feature_extractor, percentage*), which abides by the conditions indicated in a.3.
4. Now that we have a trained model, we want to be able to make predictions, which is why you will find our completed functions: `classify`(*data, label, width, height, feature_extractor*) and `compute_class`(*features*)
5. At this point, we will evaluate the effectiveness of the basic Naive Bayes classifier at varying levels of training data:
    a. **Accuracy values**:
        i. For 10% of trained data: 73.2%
        ii. For 20% of trained data: 77.2%
        iii. For 30% of trained data: 79%
        iv. For 40% of trained data: 79%
        v. For 50% of trained data: 79.9%
        vi. For 60% of trained data: 79.9%
        vii. For 70% of trained data: 80.5%
        viii. For 80% of trained data: 81.5%
        ix. For 90% of trained data: 81.3%
        x. For 100% of trained data: 81.7%
6. The *k* value we use is 0.0001 because it leads us to the most accurate performance of our classifier. Our classifier performs quite well under the Naive Bayes, with over 81.7% when given 100% of the training data. We assume that we don't have a large enough sample, but given the increase in performance as the percentage of training data increases, we can induce that the accuracy of our classifier would continue to increase.

B. Enhancing Features

1. The advanced features we chose to implement include:
    a. Checking the frequency of the body of the digit data. We chose the most encompassing feature, which is checking the shape (#) of the number we are given. This is the most accurate way to determine what the digit data actually is, even though it takes the longest time.
    b. Checking the frequency of the boundary (edges that encompass the body of the number,'+') of the digit data. This gives us insight about how large and how many pixels might be contained in the body, without having to search the body. This is a more efficient feature than our first one, but is less reliable.
    c. Checking frequency of the whitespaces occurring in the digit data. This feature provides us further information about the physical attributes of the digit data. We can determine the space that is wasted, and use this as a classification tool.
    d. Accuracy Values for our Advanced features:
        i. For 10% of trained data: 64.3%
        ii. For 20% of trained data: 64.6%
        iii. For 30% of trained data: 64.7%
        iv. For 40% of trained data: 65.4%
        v. For 50% of trained data: 67.2%
        vi. For 60% of trained data: 67.5%
        vii. For 70% of trained data: 67.5%
        viii. For 80% of trained data: 67%
        ix. For 90% of trained data: 66.6%
        x. For 100% of trained data: 66.8%
2. We report the performance of our combined features now:
    a. For 10% of trained data: 73.7%
    b. For 20% of trained data: 78.6%
    c. For 30% of trained data: 80.1%
    d. For 40% of trained data: 80.2%
    e. For 50% of trained data: 81.1%%
    f. For 60% of trained data: 81%
    g. For 70% of trained data: 80.9%
    h. For 80% of trained data: 82.4%
    i. For 90% of trained data: 82.3%
    j. For 100% of trained data: 82.5%

As you can note from the above accuracy rates of our basic features implementation and our advanced features, there is a drastic increase in performance when we combine

the features. This motivated us to create our `extract_final_features`(*digit_data, width, height*) in the B.3. as a combined features implementation.

3. Our code includes `extract_final_features`(*digit_data, width, height*). We have chosen to implement our `extract_final_features` as a **combination of both the basic and advanced features** since this gives us the highest accuracy.
4. Bonus Implementation