Detailed Analysis: SMS Spam Classification Solution

Solution Report

1 Overview

The solution implements a **Multinomial Naive Bayes** classifier and a **Support Vector Machine (SVM)** to classify SMS messages as "Spam" (1) or "Ham" (non-spam, 0). The core challenge is handling text data, which is sparse and high-dimensional, and avoiding numerical instability during probability calculations.

2 Part 1: Data Preprocessing

Before training, raw text must be converted into numerical feature vectors.

2.1 1. Word Normalization

Function: `get_words(message)`

- **Code Logic:**

```
message = message.lower()
return message.split(" ")

```

- **Explanation:** This function standardizes the text. Converting to lowercase ensures that "Free" and "free" are treated as the same word. Splitting by space is a simple tokenization strategy.

2.2 2. Dictionary Creation

Function: `create_dictionary(messages)`

- **Code Logic:**
  - It iterates through all messages and counts word frequencies using `collections.defaultdict(int)`.
  - **Filtering:** It only adds words to the final dictionary if `count` ≥ 5.
  - **Output:** Returns a dictionary mapping words to unique integer indices.
- **Why?** Removing rare words (appearing fewer than 5 times) reduces the feature space dimension and prevents the model from overfitting to noise (such as rare typos or unique names).

2.3 3. Feature Matrix Construction

Function: `transform_text(messages, word_dictionary)`

- **Code Logic:**

```
matrix = np.zeros((m, n)) # m = num messages, n = num unique words
for i, message in enumerate(messages):
    # ... finds word indices ...
    matrix[i, word_dictionary[word]] += 1

```

- **Explanation:** This implements the **Bag-of-Words** model. The result is a matrix where element (i, j) represents the number of times word j appears in message i. This converts variable-length text into fixed-length numeric vectors.

3 Part 2: Naive Bayes Implementation

This is the core of the assignment, implementing the Multinomial Event Model with Laplace Smoothing.

3.1 1. Training (Parameter Estimation)

Function: `fit_naive_bayes_model(matrix, labels)`

- **Math Goal:** Estimate $\phi_{k|y=1}$ (probability of word k in spam) and $\phi_{k|y=0}$ (probability of word k in ham).

- **The Formula (Laplace Smoothing):**

$$\phi_{k|y} = \frac{1 + \sum_{i \in Class} x_k^{(i)}}{n + \sum_{i \in Class} \sum_j x_j^{(i)}} \tag{1}$$

  (Where the 1 and n in the numerator/denominator are the smoothing terms).

- **Code Explanation:**

```
phij_y1 = (1 + matrix[labels==1].sum(axis=0)) / (n + matrix[labels
    ==1].sum())

```

  - `matrix[labels==1]`: Selects only the rows (messages) that are spam.
  - `.sum(axis=0)`: Sums the columns to get the total count of *each specific word* k across all spam messages.
  - `.sum()`: Sums the entire matrix subset to get the total count of *all words* in all spam messages.

3.2 2. Prediction (Log-Domain)

Function: `predict_from_naive_bayes_model(model, matrix)`

- **Problem:** Multiplying thousands of small probabilities results in **underflow**.

- **Solution:** Work in the **Log Domain**. Instead of $p(x|y) = \prod p(x_k|y)$, we compute $\sum \log p(x_k|y)$.

- **Code Explanation:**

```
return matrix @ (np.log(phi_k_y1) - np.log(phi_k_y0)) + np.log(
    phi_y / (1 - phi_y)) >= 0

```

  This effectively computes the Log Odds Ratio:

$$\log \frac{P(\text{Spam}|x)}{P(\text{Ham}|x)} \tag{2}$$

  If the final sum is ≥ 0, the probability of Spam is greater than Ham.

4 Part 3: Model Interpretability

Function: `get_top_five_naive_bayes_words`

- **Goal:** Identify which words strongly signal "Spam".
- **Metric:** $\log \frac{P(\text{word}|y=1)}{P(\text{word}|y=0)}$.
- **Code Logic:**

```
top_five ... = np.argsort(-(np.log(phi_i_y1) - np.log(phi_i_y0)))
    [:5]

```

  `np.argsort` sorts the log-probability differences in descending order. The top 5 indices map back to words like **"claim"**, **"won"**, **"prize"**, **"tone"**, **"urgent!"**.

5 Part 4: Support Vector Machine (SVM)

Function: `compute_best_svm_radius`

- **Context:** The SVM uses an RBF (Radial Basis Function) kernel. The `radius` parameter defines the influence range of a training example.
- **Logic:**
  - Performs a **Grid Search** over the list `[0.01, 0.1, 1, 10]`.
  - Trains an SVM for each radius on the training set.
  - Selects the radius that produces the highest accuracy on the validation set.
- **Result:** The optimal radius was found to be **0.1**, achieving **96.7%** accuracy.

6 Summary of Performance

- **Naive Bayes:** $\approx$ 97.8% Accuracy. Performed slightly better, likely because bag-of-words features fit the Multinomial assumption well.
- **SVM (RBF):** $\approx$ 96.8% Accuracy. Competitive, but required hyperparameter tuning.