NOTE:
The attached python notebook was run through JupyterLab on Anaconda. If dependencies are proving to be an issue, please also try opening the .ipynb in a similar fashion.
Python version 3.10.0

Accessing Data:

Given the .gz input data file, I couldn't initially view the raw data itself. I tried using GVIM to get an idea of how the input data was formatted, and eventually unzipped the .gz within the python notebook itself, reading in its contents as text.

Preprocessing Data:

Noticing that the format of each line of the training data was of the form:

"CLASS_SUBCLASS Here is the question ?"

And that a lot of the questions with common 'class' values shared certain keywords, I realized that it would be valuable to compare the words in each question to the keywords that indicate the corresponding class and subclass values.

Splitting the data by line, I separated each line into two halves—one with the class and one with the question—and saved both of these values at the same index in lists of questions and classes respectively.

Using scikit-learn's CountVectorizer, I transformed each string question into a one-hot encoded matrix that represented the presence or absence of a particular word, among all distinct words present in all questions. This method would weight the presence of a particular word with respect to a corresponding class, and create a correlation between the two, essentially recognizing keywords as words with high correlations.

Partitioning Data:

Given that I would have to test my model after it has been appropriately trained, I had to create a set of test questions and classes. Accounting for the fact that my only supplied input was the training.data.gz, I randomly partitioned the preprocessed 'questions' and 'classes' lists and selected a testing set.

The scikit-learn method train_test_split() takes in both x (question) and y (class) values as parameters, returning four lists of partitioning data, with corresponding x and y values maintained by list index.

Training Model:

Once the data had been split into training and testing sets, it was only a question of which type of model I would decide to use. I chose the LinearSVC() classifier—in part because it was my first inclination, and I understand it as a fundamental, fairly reliable classifier—but also because we're dealing with x and y data, which immediately indicated some kind of graphical (possibly linear) correspondence to me. I have also had some experience using svm.SVC() with a linear kernel in the past, and have noticed that it outperforms several other classifiers consistently.

Training the model only required the model.fit() method, which requires the x and y training data.

Assessing Model:

The model can only be assessed based on the test data, given that the model has been fitted based on the train data. The model.predict() method requires the train features and x test data, and returns a list of predictions which can then be compared to the y test data to measure accuracy.

The sklearn-metrics library provides a method called accuracy_score() which takes in these two values and returns a decimal representation of accuracy.

Evaluation Results:

According to the accuracy_score() function, the LinearSVC() classifier yields an accuracy of 0.976, or almost 98%. Taking a deeper dive into the predictions, this number arises from the fact that 55 out of the 2318 predictions didn't match, and the success rate of 2263 out of 2318 is indeed 97.6%.

It is important to consider that this accuracy was achieved only after the adjustment of certain parameters—initially, the test_size parameter of train_test_split was set to 0.30 as opposed to 0.15, resulting in an accuracy of 95%. This score is still remarkably high, probably owing to the fact that there are some repeat entries in the initial dataset, making it a relatively agreeable and easily predictable set for a newly-trained classifier. Of course, reducing the size of the test data essentially increases the training set, allowing the model more information on which it can base its predictions.

A .csv file containing the predictions and truth values has been included with the python notebook and this documentation, for greater visibility regarding the classes which were incorrectly predicted by the model.