Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Bayesian Classification

Jesús Fernández López
4º Software Development
MACHINE LEARNING

·Index:

·Introduction:

Bayesian classification is a probabilistic approach to categorize data based on Bayes' Theorem. It is particularly effective for tasks where uncertainty or variability is involved. In a Bayesian classifier, we compute the probability that a given input belongs to a specific class, such as "positive" or "negative" in sentiment analysis. The classifier assumes that the presence of specific features in the data is conditionally independent of one another, given the class label. This approach, known as Naive Bayes, allows for a simplified calculation of probabilities, making it highly efficient and scalable for large datasets.

In this assignment, the goal is to build a Naive Bayesian classifier to predict the sentiment of movie reviews based on the text, determining whether the review is positive or negative. This model uses prior knowledge of the data (training set) to estimate the likelihood of certain words appearing in positive or negative reviews, applying Laplace smoothing to handle unseen words and avoid zero probabilities.

·Task 1: Splitting and Counting the Reviews

Create a function that reads the file and separates it into training data and evaluation data based on the split indicator. The function should return four lists:
- Training data, containing all reviews of the training set
- Training labels, containing all associated sentiment labels for the training data
- Test data, containing all reviews of the test set
- Test labels, containing all associated sentiment labels for the test data

In this task, I implemented a Python function called split_reviews to read the movie reviews dataset and divide it into training and test sets based on the 'Split' column. The function also counts and prints the number of positive and negative reviews for each set. Below is a step-by-step explanation of the code:

The function begins by reading the Excel file containing the movie reviews using the pandas library. The data is stored in a DataFrame for easy manipulation (line 6). Then The dataset is filtered based on the 'Split' column. Reviews labeled as "train" are included in the training set, and those labeled as "test" are included in the test set. We extract both the reviews ('Review' column) and their corresponding sentiment labels ('Sentiment' column) (lines 8 - 20)

```
4. def split_reviews(file_path):
5.     # Cargamos el archivo Excel con las reseñas
6.     df = pd.read_excel(file_path)
7.
8.     # Filtramos el dataframe para crear el conjunto de entrenamiento basado en la columna 'Split'
9.     # y extraemos solo las reseñas de entrenamiento en forma de lista.
10.    train_data = df[df['Split'] == 'train']['Review'].tolist()
11.
12.    # Filtramos las filas donde la columna 'Split' es igual a 'train' (el conjunto de entrenamiento).
13.    # Luego, seleccionamos la columna 'Review' que contiene las reseñas y convertimos esa columna en una lista con .tolist().
14.    train_labels = df[df['Split'] == 'train']['Sentiment'].tolist()
15.
16.    # Hacemos lo mismo para el conjunto de test, extrayendo las reseñas de evaluación.
17.    test_data = df[df['Split'] == 'test']['Review'].tolist()
18.
19.    # Extraemos las etiquetas de sentimiento del conjunto de evaluación.
20.    test_labels = df[df['Split'] == 'test']['Sentiment'].tolist()
```

Further to that, the function should print on the console:
- the number of positive reviews in the training set
- the number of negative reviews in the training set
- the number of positive reviews in the evaluation set
- the number of negative reviews in the evaluation set

Now for both the training and test sets, the function counts how many reviews are labeled as 'positive' and 'negative' using the count() method on the lists of labels (lines 22 - 28). Additionally, the function prints the counts of positive and negative reviews for both the training and test sets (lines 30 - 33). Finally, the function returns four lists: the reviews and sentiment labels for both the training and test sets, allowing further processing in later tasks (line 36)

```
23.    # Contamos cuántas reseñas son positivas y cuántas negativas en el conjunto de entrenamiento.
24.    train_positive = train_labels.count('positive')
25.    train_negative = train_labels.count('negative')
26.
27.    # Contamos cuántas reseñas son positivas y cuántas negativas en el conjunto de evaluación.
28.    test_positive = test_labels.count('positive')
29.    test_negative = test_labels.count('negative')
30.
31.    # Mostramos en la consola el número de reseñas positivas y negativas del conjunto de entrenamiento.
32.    print(f"Training set: {train_positive} positive reviews, {train_negative} negative reviews")
33.    # Mostramos en la consola el número de reseñas positivas y negativas del conjunto de test.
34.    print(f"Test set: {test_positive} positive reviews, {test_negative} negative reviews")
35.
36.    # Devolvemos cuatro listas: las reseñas de entrenamiento, sus etiquetas, las reseñas de test y sus etiquetas.
37.    return train_data, train_labels, test_data, test_labels
38.
39.# Ejemplo de uso, donde llamamos a la función pasándole la ruta del archivo Excel
40.train_data, train_labels, test_data, test_labels = split_reviews('movie_reviews.xlsx')
```

In summary, this function successfully splits the data into training and evaluation sets, counts the number of positive and negative reviews in each, and prints the results to the console.

·Task 2: Extract relevant features

Create a function that goes through all reviews in the training data extracted in task 1. Some of the reviews contain non-alphanumeric (e.g. “.”, “,”, “!”, etc.) characters that should be removed before processing. Remove all such extra characters from the reviews, convert the reviews to lower case and split the review content into individual words. Now count the number of occurrences of each word in the training set. The function should take the data set (i.e. the training data constructed in task 1) as input parameter. Further to that, it should have an input parameter for specifying the minimum word length and the minimum number of word occurrence. Using this mapping of words to number of occurrences in the training set, extract all the words from the reviews that meet these minimum requirements. The function should return these words as list.

For Task 2, I implemented a function called extract_features to process the movie reviews in the training dataset and extract relevant words based on their length and frequency.

The function removes all non-alphanumeric characters manually by iterating over each character in the review and keeping only letters, numbers, and spaces. Any other character (such as punctuation) is replaced by a space (line 52)

After cleaning, the text is converted to lowercase to ensure that words like "Great" and "great" are treated as the same (line 57)

The cleaned review is then split into individual words using the split() method, which breaks the text based on spaces (line 60)

Then, a dictionary *(word_counts)* is used to track how many times each word appears in the training dataset. Only words that meet the minimum word length requirement are considered (lines 63 - 68).

Once all reviews are processed, the function filters out words that do not meet the minimum occurrence threshold. Only words that appear at least a certain number of times (as specified by min_occurrences) are included in the final list of relevant features (lines 75 - 77).

And finally the values are returned (line 79)

```python
def extract_features(train_data, min_word_length=3, min_occurrences=2):
    # Diccionario para almacenar la frecuencia de cada palabra
    word_counts = {}

    # Procesar cada reseña en el conjunto de entrenamiento
    for review in train_data:
        # 1. Eliminar caracteres no alfanuméricos: Recorremos cada carácter y mantenemos solo letras y números
        review_clean = ''.join([char if char.isalnum() or char.isspace() else ' ' for char in review])
        #Esto recorre cada carácter de la reseña y verifica si es alfanumérico (char.isalnum()) o un espacio en blanco (char.isspace()).
        # Si no lo es, lo reemplazamos por un espacio, Ya que luego hacemos split(), no importa si quedan espacios de más

        # 2. Convertir la reseña a minúsculas
        review_clean = review_clean.lower()

        # 3. Dividir la reseña en palabras
        words = review_clean.split()

        # 4. Contar la frecuencia de cada palabra
        for word in words:
            if len(word) >= min_word_length:  # Consideramos solo las palabras con longitud mínima
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1

        # 5. Filtrar palabras que cumplen con los requisitos de frecuencia mínima
        # word_counts es un diccionario en el que cada clave es una palabra, y el valor es el número de veces que esa palabra aparece en las reseñas.
        # items() es un método que devuelve una vista de pares clave-valor (en este caso, palabra y frecuencia) del diccionario.
        # Trabajamos desempaquetando los elementos
        filtered_words = []  # Inicializamos la lista de palabras filtradas
        for word, count in word_counts.items():
            if count >= min_occurrences:
                filtered_words.append(word)  # Agregamos la palabra a la lista filtrada

    return filtered_words
```

·Task 3: Count Feature Frequencies

Use the function created in task 2 to extract the set of all words of a minimum length and with a minimum number of occurrences from the reviews in the training set. Now create a function that goes through all positive reviews in the training set and counts for each of these words the number of reviews the word appears in. Do the same for all negative reviews as well. The function should take the review set to be searched and the set of words to look for as input parameters and should return as output a dictionary that maps each word of the input set to the number of reviews the word occurred in in the input set. If a word is not found in any review in the input set it should map to 0

In this task, I implemented a Python function called count_feature_frequencies to count the frequency of extracted words in positive and negative reviews separately. The function returns a dictionary that maps each word to the number of reviews in which it appears. The function takes two parameters: a list of reviews and a list of words whose frequencies need to be counted.

A dictionary named word_frequencies is created, where each word is initialized to zero using a dictionary comprehension (line 88). The function iterates over each review in the provided list (line 91), each review is cleaned by removing non-alphanumeric characters and converting it to lowercase, this prepares the text for further processing (line 93-94), and then the cleaned review is then split into individual words (line 95).

Now to avoid counting the same word multiple times in a single review, the list of words is converted into a set called unique_words (line 98) and the function checks each unique word to see if it exists in the word_frequencies dictionary. If it does, the count for that word is incremented.

```python
def count_feature_frequencies(reviews, words):
    # Inicializamos un diccionario para almacenar la frecuencia de cada palabra en reseñas
    word_frequencies = {word: 0 for word in words}

    # Iterar sobre cada reseña en el conjunto de reseñas
    for review in reviews:
        # Convertimos la reseña a minúsculas y la dividimos en palabras
        review_clean = ''.join([char if char.isalnum() or char.isspace() else ' ' for char in review])
        review_clean = review_clean.lower()
        review_words = review_clean.split()

        # Usamos un conjunto para evitar contar múltiples apariciones de la misma palabra en una reseña
        unique_words = set(review_words)

        # Contamos las palabras únicas que están en el conjunto de palabras a contar
        for word in unique_words:
            if word in word_frequencies:
                word_frequencies[word] += 1

    return word_frequencies
```

In summary, this function efficiently counts how many times each word from the extracted features appears in the specified set of reviews, allowing us to understand the distribution of words in positive and negative sentiments

·Task 4: Calculate Feature Likelihoods and Priors

Consider each word extracted in task 2 as a binary feature of a review indicating that a word is either present in the review or absent in the review. Using the function created in task 3 to count the number of reviews each of these features is present in calculate the likelihoods $P[\textit{word is present in review}|\textit{review is positive}]$ **and** $P[\textit{word is present in review}|\textit{review is negative}]$ **for each word in the feature vector. Create a function that calculates these likelihoods for all words applying Laplace smoothing with a smoothing factor** $\alpha = 1$**. The function should take the two mappings created in task 3 and the total number of positive/negative reviews obtained in task 1 as input and return a dictionary mapping each feature word to the likelihood probability that a word is present in a review given its class being either positive or negative. Also calculate the priors** $P[\textit{review is positive}]$ **and** $P[\textit{review is negative}]$ **by considering the fraction of positive/negative reviews in the training set.**

The function calculate_likelihoods_and_priors takes as input the word frequencies from positive and negative reviews, the total number of positive and negative reviews, and a smoothing factor alpha.

A dictionary likelihoods is initialized to store the likelihoods of each word for both positive and negative reviews (line 124). Then To compute the likelihoods, the function first calculates the total number of words in positive and negative reviews. This total includes Laplace smoothing by adding alpha * len(word_frequencies) to the sum of word frequencies (lines 127 - 128).

The likelihoods for each word are computed as the conditional probabilities P(word | positive) and P(word | negative) using Laplace smoothing (lines 131 - 142). And finally, the prior probabilities for positive and negative reviews are computed as the proportion of positive and negative reviews in the training set (lines 146 - 147).

```python
def calculate_likelihoods_and_priors(word_frequencies_pos, word_frequencies_neg, train_positive, train_negative,
                                     alpha=1):
    # Inicializamos los diccionarios para las verosimilitudes
    likelihoods = {}

    # Número total de palabras en reseñas positivas y negativas (with Laplace smoothing that explains the (+ alpha * len(word_fre...) )
    total_positive_words = sum(word_frequencies_pos.values()) + alpha * len(word_frequencies_pos)
    total_negative_words = sum(word_frequencies_neg.values()) + alpha * len(word_frequencies_neg)

    # Calculamos la verosimilitud de cada palabra
    for word in word_frequencies_pos:
        # P(word | positive) con Laplace smoothing
        # likelihood_pos is the probability of encountering a specific word given that the review is positive

        # word_frequencies_pos.get(word, 0): This retrieves the frequency (count) of the word from the positive reviews. If the word does not appear in positive reviews, .get() returns 0.
        # + alpha: This is where Laplace smoothing comes in. By adding alpha (usually 1), we ensure that the likelihood never becomes zero, even if the word doesn’t appear in any positive review.
        likelihood_pos = (word_frequencies_pos.get(word, 0) + alpha) / total_positive_words
        # P(word | negative) con Laplace smoothing
        # likelihood_neg is the probability of encountering a specific word given that the review is negative
        likelihood_neg = (word_frequencies_neg.get(word, 0) + alpha) / total_negative_words
        # Guardamos las verosimilitudes en el diccionario
        likelihoods[word] = {'positive': likelihood_pos, 'negative': likelihood_neg}

    # Cálculo de las probabilidades a priori
    # The prior is the overall probability of a review being positive, regardless of the words in the review
    prior_positive = train_positive / (train_positive + train_negative)
    prior_negative = train_negative / (train_positive + train_negative)

    return likelihoods, prior_positive, prior_negative
```

In summary, this function calculates both the likelihoods of words appearing in positive or negative reviews and the prior probabilities of a review being positive or negative. These calculations are essential for constructing the Naive Bayes classifier in the next tasks.

·Task 5: Maximum Likelihood Classification

Use the likelihood functions and priors created in task 4 to now create a Naïve Bayes classifier for predicting the sentiment label for a new review text. Remember to use logarithms of the probabilities for numerical stability. The function should take as input the new review text as string as well as the priors and likelihoods calculated in task 4. It should produce as output the predicted sentiment label for the new review (i.e. either "positive" or "negative").

In this task, I implemented a Naive Bayes classifier to predict the sentiment of a new movie review (either "positive" or "negative"). The classifier uses the likelihoods and prior probabilities computed in Task 4. To avoid numerical underflow when multiplying small probabilities, I used the logarithms of the probabilities, which ensures numerical stability.

The function naive_bayes_predict takes as input the new review text, the likelihoods for each word, and the prior probabilities for positive and negative reviews, then the review has to be cleaned again by removing non-alphanumeric characters, converting it to lowercase, and splitting it into individual words (lines 169 - 171).

As we are told, the log probabilities for both positive and negative reviews are initialized with the logarithms of their respective priors (lines 174 - 175).

For each word in the review, if it exists in the likelihoods dictionary, we add the log of the word's likelihood for both positive and negative reviews (lines 178 - 182).

And finally the function compares the logarithmic probabilities for both classes. If the log probability of a positive review is greater than that of a negative review, the sentiment is predicted as "positive"; otherwise, it is predicted as "negative".

```python
def naive_bayes_predict(review, likelihoods, prior_positive, prior_negative):
    # Limpiar y procesar la nueva reseña
    review_clean = ''.join([char if char.isalnum() or char.isspace() else ' ' for char in review])
    review_clean = review_clean.lower()
    words = review_clean.split()

    # Iniciar los valores de logaritmo de las probabilidades a priori
    log_prob_positive = math.log(prior_positive)
    log_prob_negative = math.log(prior_negative)

    # Calcular las probabilidades logarítmicas de que la reseña sea positiva o negativa
    for word in words:
        if word in likelihoods:
            # Sumar los logaritmos de las verosimilitudes para reseñas positivas y negativas
            log_prob_positive += math.log(likelihoods[word]['positive'])
            log_prob_negative += math.log(likelihoods[word]['negative'])

    # Predecir la clase con la mayor probabilidad logarítmica
    if log_prob_positive > log_prob_negative:
        return 'positive'
    else:
        return 'negative'
```

In summary, this function uses a Naive Bayes classifier with logarithmic probabilities to predict the sentiment of a given movie review, based on the likelihoods of words and the prior probabilities of each class.

·Task 6: Evaluation of Results

Create a k-fold cross-validation procedure for splitting the training set into k folds and train the classifier created in tasks 2-5 on the training subset. Evaluate the classification accuracy, i.e. the fraction of correctly classifier samples, on the evaluation subset and use this procedure to calculate the mean accuracy score. Compare different accuracy scores for different choices (1,2,3,4,5,6,7,8,9,10) of the word length parameter as defined in task 2. Select the optimal word length parameter and evaluate the resulting classifier on the test set extracted in task 1. The final evaluation should contain: - The confusion matrix for the classification - The percentage of true positive, true negatives, false positives and false negatives - The classification accuracy score, i.e. the fraction of correctly classified samples

In this task, I implemented a k-fold cross-validation procedure to evaluate the Naive Bayes classifier. The classifier is trained and validated on different folds of the training set, and its accuracy is evaluated for different word length parameters. The best word length is then used to evaluate the model on the test set. Additionally, I computed the confusion matrix and other evaluation metrics.

First of all the function is defined as k_fold_cross_validation, it takes in the training data, training labels, the number of folds (k), and a list of minimum word lengths to test. Then two parameters are initialized, best_accuracy to zero and best_word_length to one. These will be used to track the highest accuracy achieved and the corresponding word length parameter (lines 200 - 202)

In the for loop we initialize the list accuracies to store the accuracy results for each fold of the current minimum word length, and then we create an instance of the KFold class, specifying the number of splits (n_splits=k). The shuffle=True option randomizes the data, and random_state=42 ensures reproducibility (lines 205 - 209)

Now we create a loop that iterates over the indices returned by the split method of the KFold object, providing indices for training and validation sets in each fold (line 211), and using list comprehensions, we create the training and validation datasets for the current fold by selecting the respective indices from train_data and train_labels (lines 213 - 216)

Then we call the extract_features function to process the training fold data and obtain a list of relevant words based on the current minimum word length (line 219). The next two following lines create two lists: one for positive reviews and another for negative reviews in the training fold, using a list comprehension and the zip function to pair reviews with their labels (lines 222 - 225), lists created in order to call the count_feature_frequencies function for both positive and negative reviews (lines 226 - 227). This results in frequency counts for each word in both sets.

In lines 230 - 231 we count the total number of positive and negative labels in the current training fold, storing them in train_positive and train_negative respectively, and we use it for calling the calculate_likelihoods_and_priors function. It returns the likelihoods for each word, as well as the prior probabilities for positive and negative reviews (lines 232 - 234)

Line 237 uses a list comprehension to generate predictions for each review in the validation fold by calling the naive_bayes_predict function with the validation data and the previously calculated likelihoods and priors

Then the predicted labels are compared against the true labels in the validation fold using the accuracy_score function, and appends the accuracy to the accuracies list (lines 241 - 242)

Lastly for this function, we calculate the mean accuracy across all folds for the current minimum word length by summing the accuracies and dividing by the number of folds (line 245) and we check if the current mean accuracy is greater than the previously recorded best accuracy. If it is, update best_accuracy and set best_word_length to the current minimum word length (lines 248 - 250). Then values returned (line 252).

```
200.def k_fold_cross_validation(train_data, train_labels, k=5, min_word_lengths=[1, 2, 3, 4, 5]):
201.    best_accuracy = 0
202.    best_word_length = 1
203.
204.    # Probar diferentes longitudes mínimas de palabra
205.    for min_word_length in min_word_lengths:
206.        accuracies = []
207.
208.        # Preparar k-fold
209.        kf = KFold(n_splits=k, shuffle=True, random_state=42)
210.
211.        for train_index, val_index in kf.split(train_data):
212.            # Dividimos los datos en conjunto de entrenamiento y validación
213.            train_fold_data = [train_data[i] for i in train_index]
214.            val_fold_data = [train_data[i] for i in val_index]
215.            train_fold_labels = [train_labels[i] for i in train_index]
216.            val_fold_labels = [train_labels[i] for i in val_index]
217.
218.            # Extraemos las palabras relevantes del conjunto de entrenamiento
219.            filtered_words = extract_features(train_fold_data, min_word_length=min_word_length)
220.
```

```python
                # Calculamos las frecuencias de las palabras en reseñas positivas y negativas
                positive_reviews = [review for review, label in zip(train_fold_data, train_fold_labels) if
                                    label == 'positive']
                negative_reviews = [review for review, label in zip(train_fold_data, train_fold_labels) if
                                    label == 'negative']
                word_frequencies_pos = count_feature_frequencies(positive_reviews, filtered_words)
                word_frequencies_neg = count_feature_frequencies(negative_reviews, filtered_words)

            # Calculamos las verosimilitudes y las probabilidades a priori
            train_positive = train_fold_labels.count('positive')
            train_negative = train_fold_labels.count('negative')
            likelihoods, prior_positive, prior_negative = calculate_likelihoods_and_priors(
                word_frequencies_pos, word_frequencies_neg, train_positive, train_negative
            )

            # Validamos el modelo en el conjunto de validación
            predictions = [naive_bayes_predict(review, likelihoods, prior_positive, prior_negative) for review in
                           val_fold_data]

            # Calculamos la precisión en el conjunto de validación
            accuracy = accuracy_score(val_fold_labels, predictions)
            accuracies.append(accuracy)

        # Calculamos la precisión media para esta longitud de palabra
        mean_accuracy = sum(accuracies) / len(accuracies)

        # Actualizamos la mejor longitud de palabra si es necesario
        if mean_accuracy > best_accuracy:
            best_accuracy = mean_accuracy
            best_word_length = min_word_length

    return best_word_length, best_accuracy
```

In summary, The function k_fold_cross_validation performs k-fold cross-validation (with k = 5). It tries different values for the minimum word length parameter (e.g., 1, 2, 3, etc.), training and validating the classifier on different folds of the training data. For each fold, the classifier is trained on k-1 folds and validated on the remaining fold. The average accuracy is calculated for each word length, and the best word length is selected based on the highest mean accuracy.

- - - - - - - - - - - - -

Then we focus on the function evaluate_on_test which evaluates the final model on the test set using the best word length. In this function the words are extracted from the training data, and the likelihoods and prior probabilities are computed again. The classifier then makes predictions on the test set, and the confusion matrix is generated using confusion_matrix. The final accuracy is also calculated using accuracy_score. Let's explain the code:

We start calling the extract_features function to extract the relevant words from the entire training data using the best word length determined from k-fold cross-validation (line 258).

Similar to before, lines 261 and 262 separates the full training dataset into positive and negative reviews for frequency counting and then lines 263 and 264 count the frequencies of the relevant words in both positive and negative reviews from the training dataset.

Then we count the total number of positive and negative labels in the entire training set (line 276 - 268) and calculate the likelihoods and prior probabilities for the test evaluation based on the full training data (lines 269 - 270).

In order to generate predictions for the test data we use a list comprehension to generate predictions for each review in the test data by calling the naive_bayes_predict function (line 274), and calls the confusion_matrix function to create a confusion matrix using the true labels and the predictions for the test data. It specifies that the labels to be used are 'positive' and 'negative' (line 277).

At this moment we can compute the accuracy of the predictions on the test data using the accuracy_score function (line 280). And finally we return the data (line 282).

```
256. def evaluate_on_test(test_data, test_labels, best_word_length, train_data, train_labels):
257.     # Extraemos las palabras relevantes del conjunto de entrenamiento usando la mejor longitud
258.     filtered_words = extract_features(train_data, min_word_length=best_word_length)
259.
260.     # Calculamos las frecuencias de palabras en reseñas positivas y negativas
261.     positive_reviews = [review for review, label in zip(train_data, train_labels) if label == 'positive']
262.     negative_reviews = [review for review, label in zip(train_data, train_labels) if label == 'negative']
263.     word_frequencies_pos = count_feature_frequencies(positive_reviews, filtered_words)
264.     word_frequencies_neg = count_feature_frequencies(negative_reviews, filtered_words)
```

```
266.     # Calculamos las verosimilitudes y las probabilidades a priori
```

```python
    train_positive = train_labels.count('positive')
    train_negative = train_labels.count('negative')
    likelihoods, prior_positive, prior_negative = calculate_likelihoods_and_priors(
        word_frequencies_pos, word_frequencies_neg, train_positive, train_negative
    )

    # Generamos predicciones para el conjunto de test
    predictions = [naive_bayes_predict(review, likelihoods, prior_positive, prior_negative) for review in test_data]

    # Calculamos la matriz de confusión
    conf_matrix = confusion_matrix(test_labels, predictions, labels=['positive', 'negative'])

    # Calculamos la precisión final
    final_accuracy = accuracy_score(test_labels, predictions)

    return conf_matrix, final_accuracy
```

·Task 7: Personal Review Test

Choose a movie you like and a movie you hate and write a review for both. Try the classifier on your review and see if the predicted sentiment score matches your own sentiment.

In this task, I wrote two personal movie reviews: one for a movie I liked and another for a movie I disliked. I used the Naive Bayes classifier developed in the previous tasks to predict the sentiment of these reviews. The goal was to see if the classifier could correctly identify the sentiment based on the review text.

The function takes the reviews and uses the Naive Bayes classifier to predict the sentiment for each review, and prints whether the classifier predicts the review as "positive" or "negative."

(lines 295 - 325)

```
295. def test_personal_reviews(classifier_func, likelihoods, prior_positive, prior_negative):
296.    # Reseña positiva
297.     positive_review = ("I recently watched The Grand Budapest Hotel, and it was an absolute masterpiece … "
298.
306.    # Reseña negativa
307.      negative_review = ("Last night, I watched The Room, and it was honestly one of the worst movies I’ve ever seen… "
308.
315.    # Prueba el clasificador en ambas reseñas
316.      positive_prediction = classifier_func(positive_review, likelihoods, prior_positive, prior_negative)
317.      negative_prediction = classifier_func(negative_review, likelihoods, prior_positive, prior_negative)
318.
319.    # Mostrar los resultados
320.    print(f"Positive review prediction: {positive_prediction}")
321.    print(f"Negative review prediction: {negative_prediction}")
322.
323.
324. # Ejemplo de uso
325. test_personal_reviews(naive_bayes_predict, likelihoods, prior_positive, prior_negative)
```