**I'd be glad to explain the code in detail:**

**Imports:**

**Python**

**from keras.datasets import imdb**

**import numpy as np**

**from keras.models import Sequential**

**from keras.layers import Dense**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score**

* **keras.datasets.imdb: This line imports the imdb function from the keras.datasets module. This function is used to load the IMDB movie review sentiment classification dataset.**
* **numpy: This imports the numpy library, a powerful numerical computation library in Python.**
* **keras.models: This imports the Sequential class from the keras.models module, used to create sequential neural network models.**
* **keras.layers: This imports the Dense class from the keras.layers module, used to create dense (fully connected) layers in neural networks.**
* **sklearn.metrics: This imports several metrics functions from the sklearn.metrics module used for evaluating the performance of a classification model.**

**Data Loading and Preprocessing:**

**Python**

**(train\_data, train\_label), (test\_data, test\_label) = imdb.load\_data(num\_words=10000)**

**def vectorize\_sequences(sequences, dimensions=10000):**

**results = np.zeros((len(sequences), dimensions))**

**for i, sequence in enumerate(sequences):**

**results[i, sequence] = 1**

**return results**

**x\_train = vectorize\_sequences(train\_data)**

**x\_test = vectorize\_sequences(test\_data)**

**y\_train = np.asarray(train\_label).astype('float32')**

**y\_test = np.asarray(test\_label).astype('float32')**

* **imdb.load\_data(num\_words=10000): This line loads the IMDB dataset. The num\_words argument specifies that only the 10,000 most frequent words will be considered. Each review is represented as a list of integer indices, where each index corresponds to a word in the vocabulary.**
* **vectorize\_sequences function: This function takes a list of sequences (reviews) and converts them into a binary matrix. The matrix has dimensions (number of sequences, 10000). Each row represents a review, and a 1 at a certain index indicates the presence of that word in the review. Reviews are essentially converted into a "bag of words" representation.**
* **Data transformation:**
  + **x\_train and x\_test: These variables hold the vectorized training and testing data, respectively.**
  + **y\_train and y\_test: These variables hold the sentiment labels (positive: 1, negative: 0) for the training and testing data, converted to floats using astype('float32').**

**Model Definition:**

**Python**

**model = Sequential()**

**model.add(Dense(16, input\_shape=(10000,), activation='relu'))**

**model.add(Dense(16, activation='relu'))**

**model.add(Dense(1, activation='sigmoid'))**

**model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])**

**model.summary()**

* **Sequential model: A sequential neural network is created.**
* **Hidden layers:**
  + **model.add(Dense(16, input\_shape=(10000,), activation='relu')): The first hidden layer has 16 neurons, takes input of size 10000 (matching the vectorized data dimensions), and uses the ReLU (Rectified Linear Unit) activation function.**
  + **model.add(Dense(16, activation='relu')): The second hidden layer also has 16 neurons and uses the ReLU activation.**
* **Output layer:**
  + **model.add(Dense(1, activation='sigmoid')): The output layer has 1 neuron and uses the sigmoid activation function. Since the task is binary classification (positive or negative sentiment), the sigmoid function is suitable as it outputs values between 0 and 1, which can be interpreted as probabilities of being positive.**
* **Model compilation:**
  + **model.compile(optimizer='adam', loss='mse', metrics=['accuracy']): The model is compiled with the Adam optimizer, mean squared error (MSE) as the loss function (suitable for regression-like tasks here), and accuracy as a metric for evaluation.**

**Model Training and Evaluation:**

**Python**

**history = model.fit(x\_train, y\_train, validation\_split=0.2, epochs=20, verbose=1**

**The remaining part of the code focuses on evaluating the trained model's performance and interpreting some of the results. Here's a breakdown:**

**Model Evaluation:**

**Python**

**mse, mae = model.evaluate(x\_test, y\_test)**

**print('MSE ', mse)**

**print('MAE ', mae)**

* **model.evaluate(x\_test, y\_test): This line evaluates the model on the test data (x\_test and y\_test). The evaluation returns two metrics:**
  + **mse: Mean squared error, which measures the average squared difference between the predicted and actual sentiment labels (ideally as close to 0 as possible).**
  + **mae: Mean absolute error, which measures the average absolute difference between the predicted and actual sentiment labels (ideally as close to 0 as possible).**

**Predicting on New Data:**

**Python**

**y\_preds = model.predict(x\_test)**

**y\_preds**

**y\_test**

* **model.predict(x\_test): This line uses the trained model to predict sentiment labels for the test data (x\_test). y\_preds now holds the predicted probabilities of each review being positive (between 0 and 1).**

**Thresholding Predictions:**

**Python**

**tests = []**

**for i in y\_test:**

**tests.append(int(i))**

**preds = []**

**for i in y\_preds:**

**if i[0] > 0.5:**

**preds.append(1)**

**else:**

**preds.append(0)**

* **This code block converts the predicted probabilities (y\_preds) into binary class labels (positive or negative):**
  + **tests: This list holds the actual sentiment labels from y\_test, converted to integers (0 for negative, 1 for positive).**
  + **preds: This list holds the predicted class labels. It iterates through each predicted probability in y\_preds and assigns a class label based on a threshold of 0.5. If the probability (i[0]) is greater than 0.5, it's classified as positive (1), otherwise negative (0).**

**Evaluation Metrics:**

**Python**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score**

**print(accuracy\_score(tests, preds))**

**print(precision\_score(tests, preds))**

**print(recall\_score(tests, preds))**

* **These lines import three evaluation metrics from sklearn.metrics:**
  + **accuracy\_score: Calculates the overall accuracy of the model's predictions (percentage of correct classifications).**
  + **precision\_score: Measures the proportion of positive predictions that were actually positive (precision).**
  + **recall\_score: Measures the proportion of actual positive reviews that were correctly classified as positive (recall).**

**These metrics provide a more comprehensive picture of the model's performance beyond just mean squared error or mean absolute error. Accuracy tells you the overall percentage of correct predictions, while precision and recall offer insights into how well the model identifies positive and negative reviews accurately.**

**Looking at Examples:**

**Python**

**# word\_index is a dictionary mapping words to an integer index**

**word\_index = imdb.get\_word\_index()**

**def return\_token(tid):**

**for k, v in word\_index.items():**

**# We decode the review; note that our indices were offset by 3**

**# because 0, 1 and 2 are reserved indices for "padding", "start of sequence", and "unknown".**

**if v == tid - 3:**

**return k**

**return '?'**

**def print\_review(id\_):**

**sentence = ' '.join(return\_token(i) for i in train\_data[id\_])**

**return sentence**

**train\_label[0] # Positive**

**print\_review(1)**

**train\_label[1] # Negative**

**print\_review(2)**

* **This section retrieves the word-to-index mapping (word\_index) from the imdb dataset. This allows us to convert the integer-based reviews back into human-readable words.**
* **The return\_token function takes an integer index and looks it up in the word\_index dictionary to return the corresponding word.**
* **The print\_review function takes a review ID and uses return\_token to convert the integer indices in the review (stored in train\_data) back into words, forming a complete sentence.**
* **Finally, the code prints the sentiment labels for the first two reviews in the training data (train\_label[0] and train\_label[1]) and then uses print\_review to**