Chapter 4. Text Classification

Text Classification

A common task in natural language processing is classification. The goal of the task is to train a model to assign a label or class to some input text (see Figure 4-1). Classifying text is used across the world for a wide range of applications, from sentiment analysis and intent detection to extracting entities and detecting language.

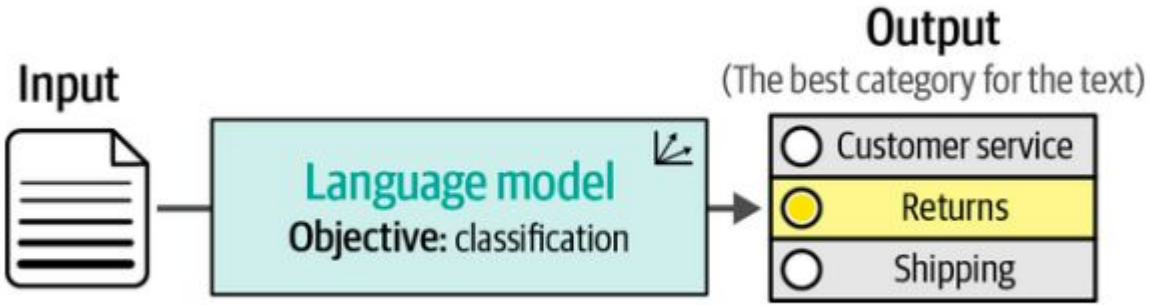

Figure 4-1. Using a language model to classify text.

used for classifying text. As illustrated in Figure 4-2, we will examine both representation and language models and explore their differences.

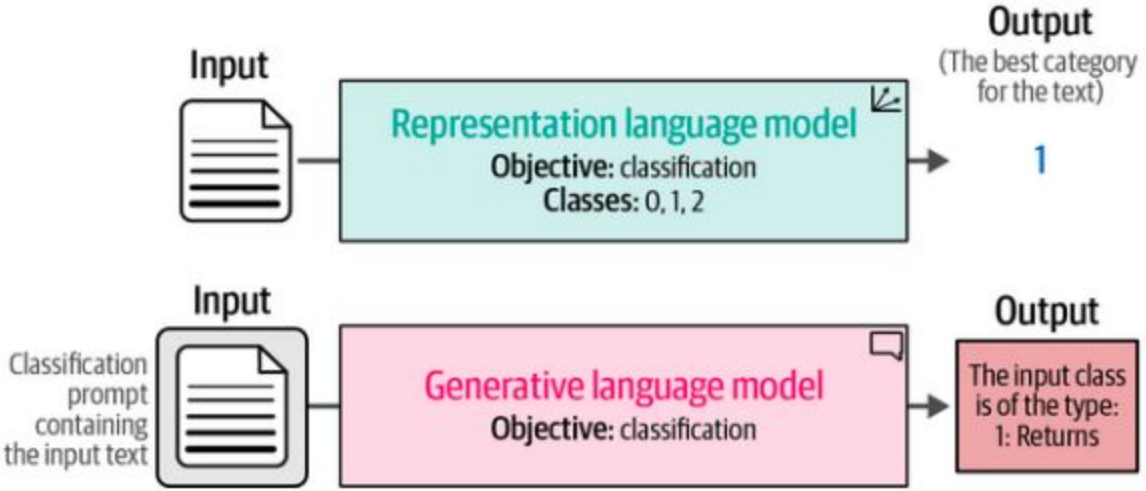

Figure 4-2. Although both representation and generative models can be used for classification, their approaches differ

A major benefit of these tokens is that they can be combined to generate representations even if they were not in the training data, as shown in Figure 4-7.

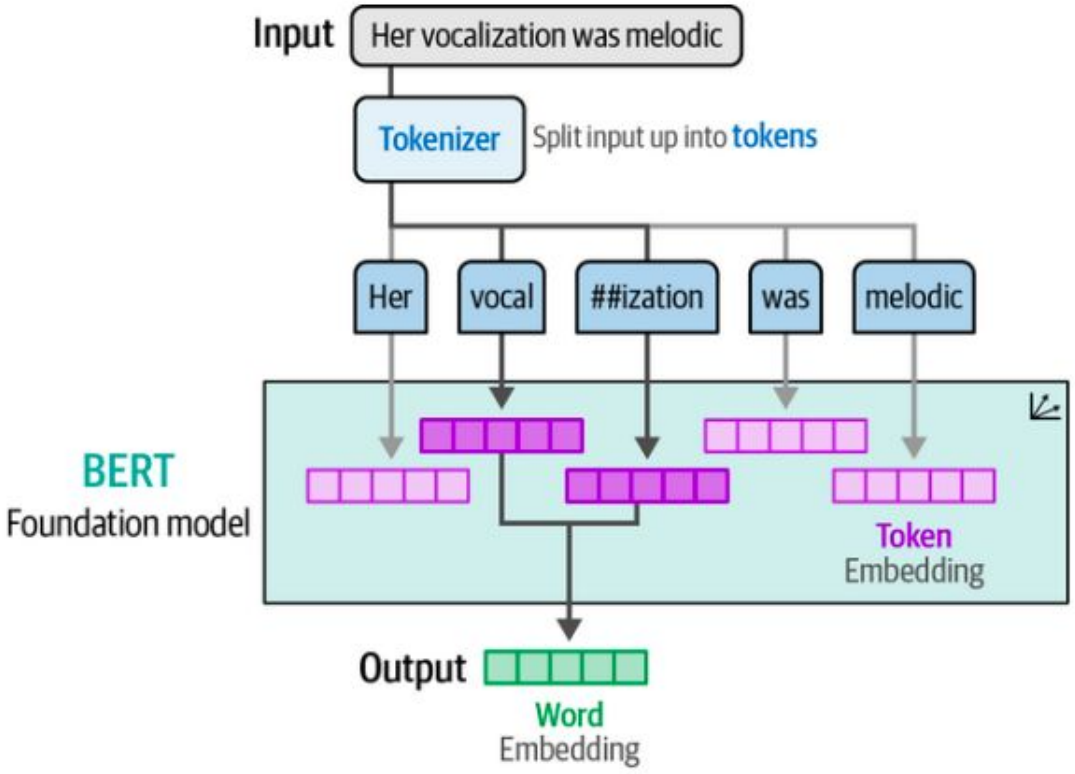

Figure 4-7. By breaking down an unknown word into tokens, word embeddings can still be generated.

Metrics

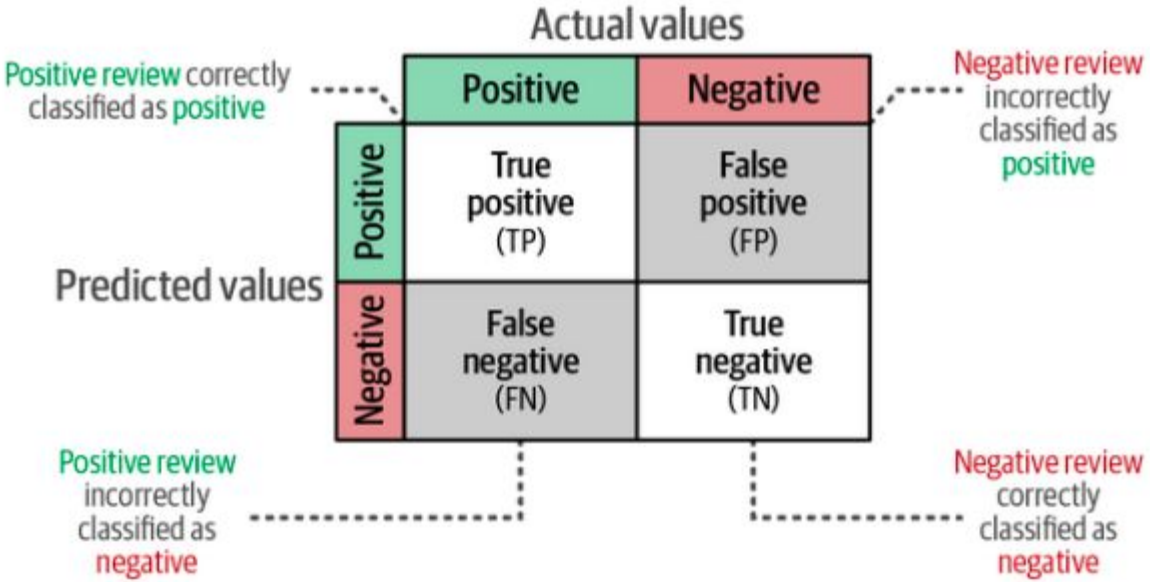

Figure 4-8. The confusion matrix describes four types of predictions we can make.

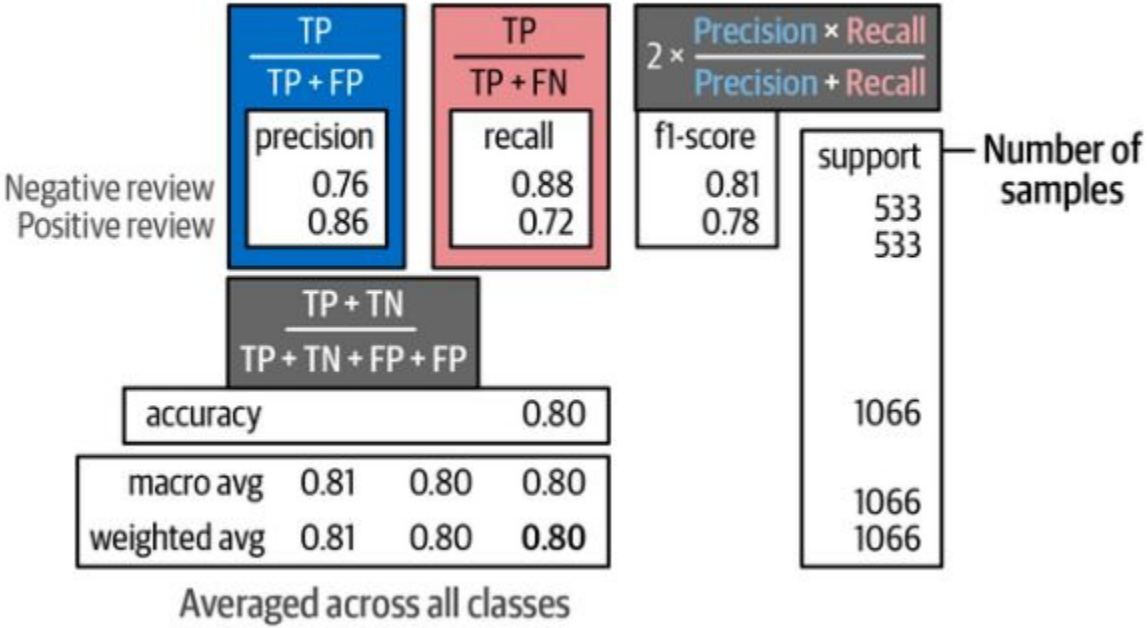

Figure 4-9. The classification report describes several metrics for evaluating a model's performance.

Using the confusion matrix, we can derive several formulas to describe the quality of the model. In the previously generated classification report we can see four such methods, namely precision, recall, accuracy, and the F1 score:

- Precision measures how many of the items found are relevant, which indicates the accuracy of the relevant results.
- Recall refers to how many relevant classes were found, which indicates its ability to find all relevant results.
- Accuracy refers to how many correct predictions the model makes out of all predictions, which indicates the overall correctness of the model.
- The F1 score balances both precision and recall to create a model's overall performance.

Text Classification with Representation Models

Classification with pretrained representation models generally comes in two flavors, either using a task-specific model or an embedding model. As we explored in the previous chapter, these models are created by fine-tuning a foundation model, like BERT, on a specific downstream task as illustrated in Figure 4-3.

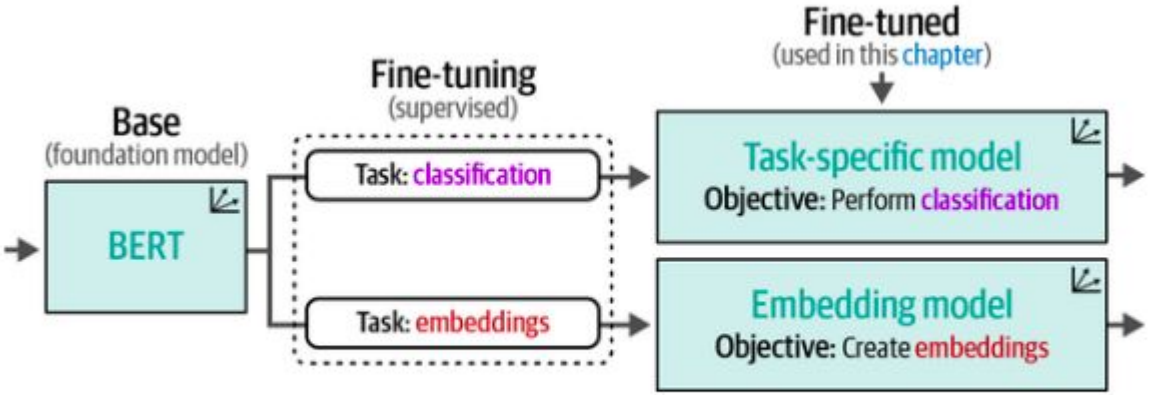

Figure 4-3. A foundation model is fine-tuned for specific tasks; for instance, to perform classification or generate general-purpose embeddings.

As we load our model, we also load the tokenizer, which is responsible for
converting input text into individual tokens, as illustrated in Figure 4-6.

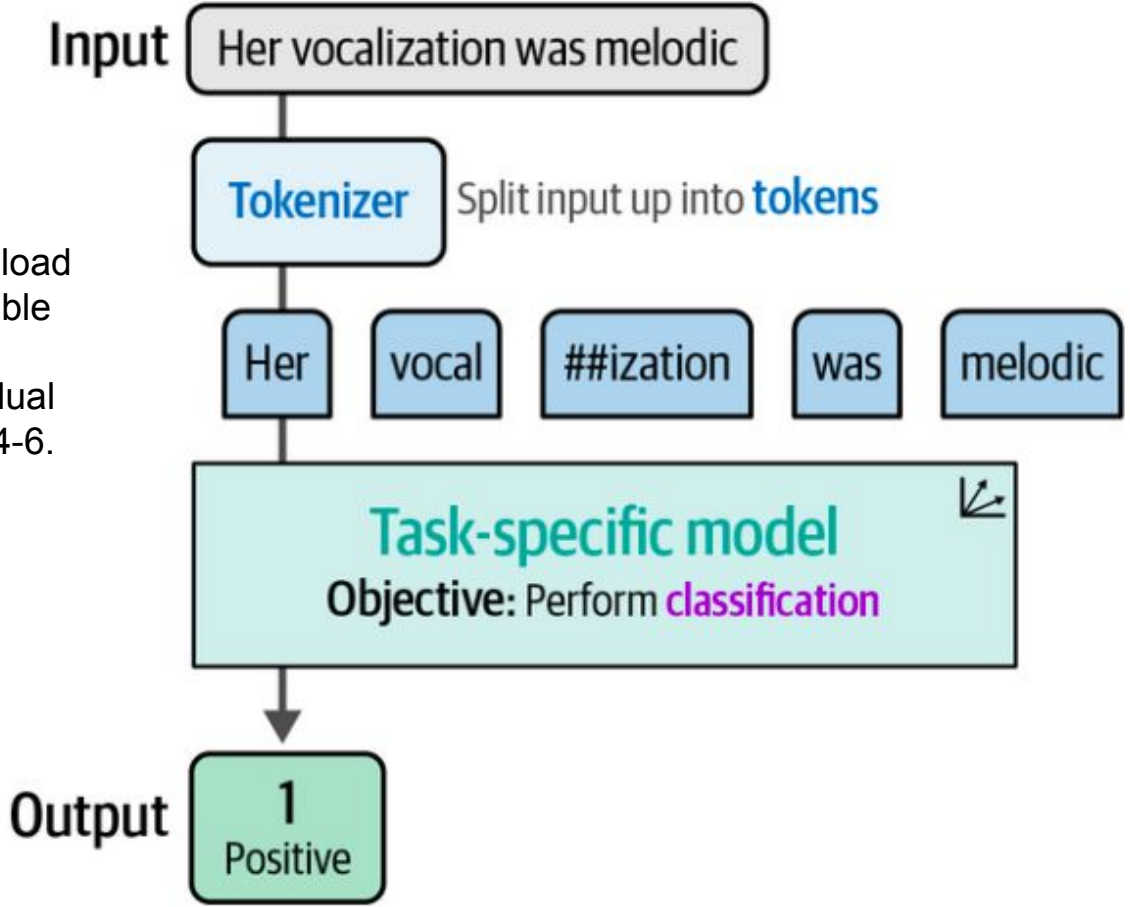

Figure 4-6. An input sentence is first fed to a tokenizer before it can be processed by the task-specific model.

Task -specific model

A **task-specific** model is typically something like "**bert-base-uncased-finetuned-SST-2**," which is BERT fine-tuned on the Stanford Sentiment Treebank (SST-2) dataset. The final layer is designed to output a probability distribution over classes (positive or negative).

How It Works

1. **Model Input**: Text is tokenized into subword units.
2. **Model Forward Pass**: The model processes these tokens.
3. **Classifier Head**: The model has a classification layer that outputs probabilities, e.g., `[0.2 (negative), 0.8 (positive)]`.
4. **Prediction**: The class with the highest probability is chosen.

Embedding Model

Now, suppose you can't find a task-specific model that perfectly suits your domain. Or maybe you want more control over your final classifier. In that case, you can take the approach of generating **embeddings** and then training a lightweight classifier, such as **Logistic Regression** or **Random Forest**, on top of those embeddings.

Step-by-Step

1. **Load an Embedding Model**: For example, `sentence-transformers/all-mpnet-base-v2`
2. **Convert Text to Embeddings**: The model generates a vector for each piece of text.
3. **Train a Classifier**: Feed these vectors into a standard classifier (e.g., Logistic Regression).
4. **Predict**: For new input, create the embedding and then run it through the classifier.

Zero-Shot Classification with Embeddings

Imagine you have a dataset but **no labels** at all. You just know the general categories you'd like to assign (e.g., "positive" vs. "negative"). **Zero-shot classification** steps in to save the day.

The "Trick" with Embeddings

1. **Create a text description of each label** (e.g., "This is a positive movie review").
2. **Embed the label descriptions** and your documents using the same embedding model.
3. **Compute the Cosine Similarity** between each document's embedding and each label's embedding.
4. **Pick the Label** with the highest similarity score.

Example

Label descriptions:

- “A very positive movie review”
- “A very negative movie review”
- Text to classify: “This movie was an absolute masterpiece!”
- Compute similarity with each label. Whichever is higher gets assigned.

This approach can yield surprisingly strong performance. Even though no labeled training data is used, the **richness** of the embedding space can accurately capture sentiment differences.

Generative Models for Text Classification

Generative models, such as **GPT-3.5**, **ChatGPT**, **Flan-T5**, and the entire GPT family, are **sequence-to-sequence** models. They take text in and produce text out.

Using Flan-T5 for Classification

1. **Load the Model**: For example, `google/flan-t5-small`.
2. **Create a Prompt**: “Is the following sentence positive or negative? ”
3. **Generate Output**: The model’s textual output might be the word “positive” or “negative.”
4. **Convert Output to a Label**: Map “positive” → `1`, “negative” → `0`.

ChatGPT: Using a Closed Source Model via API

ChatGPT, powered by the GPT-3.5 or GPT-4 family, is accessed through OpenAI's API. You can't download it to your own servers. Instead, you send text to the API, and it sends back a response.

Setting Up

1. **Create an OpenAI Account** (the free tier often covers many requests).
2. **Retrieve Your API Key**.
3. **Send Requests**: Typically, in a chat style format with roles: "system", "user", "assistant."

Example: Sentiment Classification

Prompt:

```
Predict whether the following document is a positive or negative movie review:

[DOCUMENT]

If it is positive return 1 and if it is negative return 0. Do not give any other answers.
```

- **Document**: “The acting was superb, and I laughed the whole time.”
- **Expected output**: “1” (because it’s positive).

Conclusion

Text classification is an incredibly powerful tool in NLP — and modern language models make it more accessible and accurate than ever before.

- **Representation models** (like BERT or RoBERTa) do a great job converting text into structured embeddings that capture semantic meaning.
- **Task-specific models**: Already fine-tuned for, say, sentiment analysis. Easy to use out of the box.
- **Embedding models**: Generate general-purpose embeddings. You can train a simple classifier on top or do **zero-shot** labeling by comparing embedding similarities.
- **Generative models** (like Flan-T5 or ChatGPT) output text but can be coaxed into classification by prompting.
- They offer **flexibility** and **strong performance** even without fine-tuning.
- They do, however, require mindful **prompt engineering** and can be costlier for large-scale inference.