# 目录

[目录 1](#_Toc156660950)

[作者探究 1](#_Toc156660951)

[Chapter1 1](#_Toc156660952)

[Summary: 1](#_Toc156660953)

[Data 2](#_Toc156660954)

[Supervised Text Classification And some LLMs 2](#_Toc156660955)

[1. Train a classifier 3](#_Toc156660956)

[2. Pre-Trained Embeddings (Separate feature extract and classifier) 4](#_Toc156660957)

[Zero-shot Classification 5](#_Toc156660958)

[1. Pre-trained Embeddings 5](#_Toc156660959)

[2. Natural Language Inference 6](#_Toc156660960)

[With Generative Models 6](#_Toc156660961)

[Chapter2 6](#_Toc156660962)

[Summary： 6](#_Toc156660963)

[Chapter3 7](#_Toc156660964)

[Summary: 7](#_Toc156660965)

[Chapter4 Tokens & Token Embeddings 7](#_Toc156660966)

[Summary 7](#_Toc156660967)

# 作者探究

Jay Alammar [Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)](https://jalammar.github.io/)

自然语义处理方面牛人

Maarten Grootendorst [Maarten Grootendorst](https://www.maartengrootendorst.com/)

A Machine Learning Engineer with a background in Psychology passionate about using Artificial Intelligence to make the world a slightly better place.

实验部分，Jupyter使用

# Chapter1 classification tasks

## **Summary:**

* Different techniques for performing a wide variety of classification tasks. From fine-tuning your entire model to no tuning at all! Classifying textual data there is an incredible amount of creative techniques for doing so.
* In the next chapter, we will continue with classification but focus instead on unsupervised classification. What can we do if we have textual data without any labels? What information can we extract? We will focus on clustering our data as well as naming the clusters with topic modeling techniques.

## Data

“rotten\_tomatoes” movie reviews. About 5000 positive and 5000 negative movie reviews from Rotten Tomatoes.

## Supervised Text Classification And some LLMs

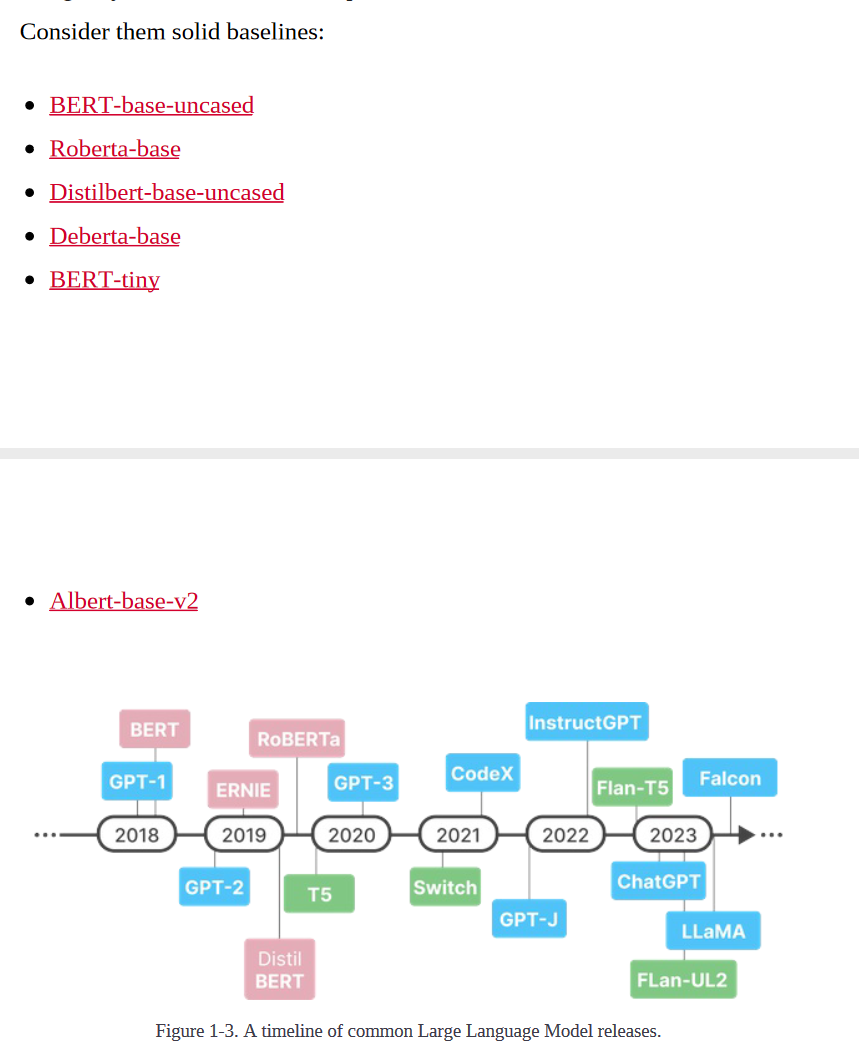
Freezing its layers: not to fine-tune the LLM.

Unfreeze some layers may be beneficial.

**BERT** is a great underlying architecture, often excel at being fine-tuned for specific tasks.

**GPT-like** models typically excel at a broad and wide variety of tasks.

There are lots of pre-trained LLM, in BERT, having BERT, RoBERTa, DistilBERT, ALBERT, DeBERTa. How to efficiently choose? Some start points. In HuggingFace’s Hub



Use “bert-base-cased” for some of examples.

### Train a classifier

1. Using a generic pre-trained LLM to convert textual data into more numerical representations. Freezing the Model
2. After LLM, train a classifier on the representations and labels. Feed forward Neural Network.

What is fine-tuning?

Fine-tuning in the context of machine learning and particularly in deep learning, refers to the process of taking a model that has already been trained on a large dataset (often called a pre-trained model) and then further training it on a smaller, more specific dataset. This process adjusts the weights and biases of the model to make it more suitable for the specific task at hand. Here's a more detailed explanation:

Some evaluation metrics

Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positives. Precision = TP / (TP + FP) where TP is the number of true positives and FP the number of false positives.

Recall (Sensitivity): Recall is the ratio of correctly predicted positive observations to all observations in the actual class. It answers the question, "**Of all the instances that are actually positive, how many were labeled correctly?**" High recall indicates a low rate of false negatives (instances that are positive but incorrectly labeled as negative). Recall = TP / (TP + FN) where TP is the number of true positives and FN the number of false negatives.

F1-Score: The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is a good way to show that a class has a balance between precision and recall. F1-Score = 2 \* (Precision \* Recall) / (Precision + Recall)

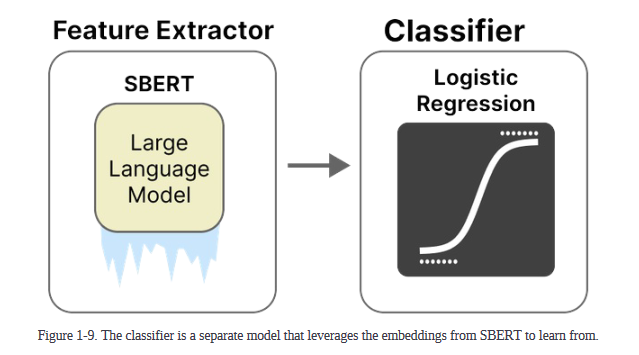
Accuracy: Accuracy is the ratio of correctly predicted observations to the total observations. It is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

Accuracy = (TP + TN) / (TP + FP + FN + TN) where TN is the number of true negatives.

Macro Average: Macro average is the average precision, recall, and F1-score between classes. Macro averaging computes the metric independently for each class and then takes the average (hence treating all classes equally).

Weighted Average: Weighted average takes into account the support of each class. This means that the metric is calculated for each class separately, but when it takes the average, it uses the support as a weight. This is useful when dealing with class imbalances, as it gives more weight to the metrics of the class with more instances.

### 2. Pre-Trained Embeddings (Separate feature extract and classifier)

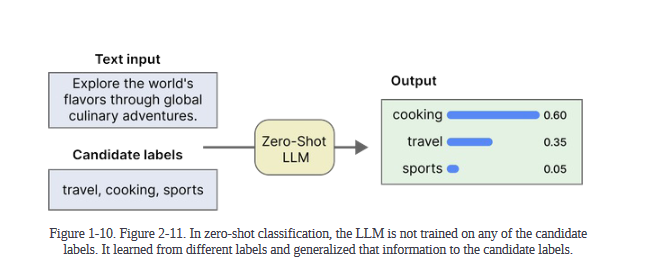


Smaller model with impressive output.

*罗吉斯特回归是一种用于二元分类的统计方法，通过逻辑函数将线性方程的输出转换为概率。*

## Zero-shot Classification

This method is a nice example of transfer learning where a model trained for one task is used for a task different than what it was originally trained for.



GPT-like models quite well

### Pre-trained Embeddings

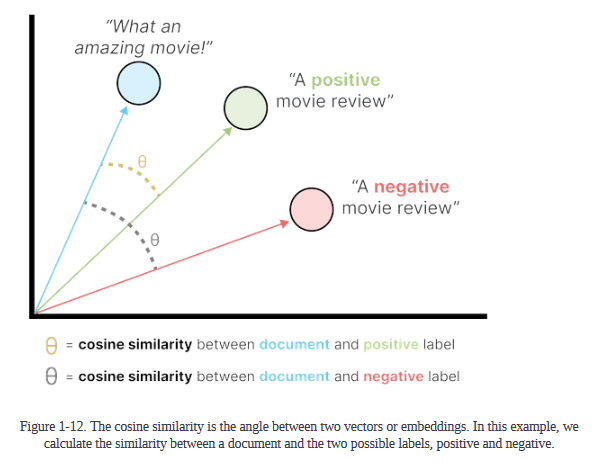
We can describe our labels based on what they should represent, for example, a negative label for movie reviews can be described as “This is a negative movie review”.

That is **describe + embed.**

Then assign labels to document using cosine similarity.

Cosine similarity, which will often be used throughout this book, is a similarity measure that checks how similar two vectors are to each other.

An intuition:



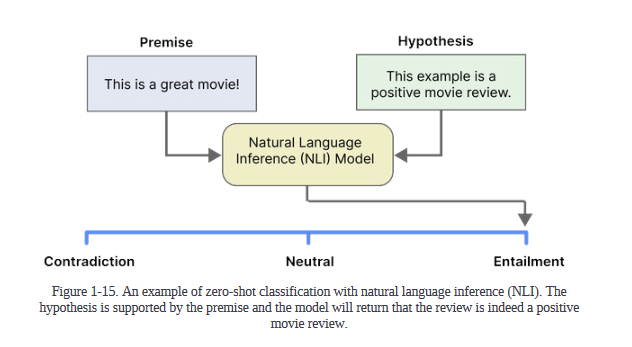
No train besides pre-trained LLM. Impressive prediction.

We can make the concrete and specific description to achieve a good accuracy.

### Natural Language Inference

For a given premise, a hypothesis is true (entailment) or false (contradiction).

Using the review as premise, then create a hypothesis asking whether the premise is about our target label.



## With Generative Models

Guide the model toward the type of answers that we are looking for. Using prompt.

Called **prompt engineering.**

Language Models are Few-Shot Learners

### In context learning

A generative model can something new with a few examples. (**in-context learning**) no fine-tuning!!

Not really “learning”

API key: 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

tenacity是一个Python库，用于简化重试逻辑的实现。这个库特别适用于网络请求和其他需要重试机制的操作。当你的代码执行某项操作时，如果遇到暂时性故障（如网络问题、暂时性的服务不可用等），tenacity能够自动地重试该操作，而你可以自定义重试的策略，比如重试的次数、重试的间隔时间等。here allows us to deal with rate limit errors

tenacity模块在处理API调用时如何应用指数退避策略 exponential backoff来自动重试请求。指数退避是一种错误处理策略，特别是针对达到速率限制错误时。该策略在初次请求失败后，会进行短暂的休眠，然后重试请求。如果请求再次失败，休眠时间将增加，这个过程会重复进行，直到请求成功或达到最大重试次数。通过这种方式，可以有效避免因速率限制而导致的错误，提高请求成功率。

**Zero shot**

虽然GPT在零样本分类任务上表现出了高性能，但如果涉及到领域特定的数据，通常通过微调(fine-tuning)方法进行学习会比在上下文中学习(in-context learning)表现得更好。这是因为在预训练过程中，模型不太可能接触到这些特定领域的数据。当模型的参数没有针对特定任务进行更新时，其适应任务特定细节的能力可能会受限。因此，为了进一步提高性能，最好是在这些数据上对GPT模型进行微调。

**Few shot**

Dd

### Named Entity Recognition

Named Entity Recognition (NER) 文本序列中检测实体

工具 SpaCy

SpaCy的一个关键特性是其处理管道（pipeline），这是一系列的处理步骤（如分词、词性标注、命名实体识别等），用于处理和分析文本。用户可以根据需要自定义这个处理管道，加入或移除特定的组件。

总的来说，SpaCy是一个功能强大、高效的NLP库，适合用于各种规模的NLP项目，从原型开发到生产部署。



1. **创建空的SpaCy模型**： 使用**spacy.blank("en")**创建一个空的英文SpaCy模型。这个模型本身不包含任何NLP处理管道，这意味着它不会进行词性标注、依存解析等操作，直到你添加具体的处理组件。
2. **定义NER任务**： 创建一个字典**task**，指定了要执行的任务类型（在这个例子中是命名实体识别**spacy.NER.v1**）和要识别的实体类型（如日期、年龄、地点、疾病和症状）。这里使用了**@llm\_tasks**作为键，表示这是一个与大型语言模型（LLM）相关的任务。
3. **选择后端**： 创建一个字典**backend**，指定了用于执行任务的后端配置。在这个例子中，选择的是通过**spacy.REST.v1**与OpenAI的API交互的REST后端，并指定了使用的模型是**gpt-3.5-turbo**。这意味着SpaCy将通过OpenAI提供的API，使用指定的GPT模型来处理文本。
4. **组合配置并创建处理管道**： 通过合并**task**和**backend**字典，并将结果配置传递给**nlp.add\_pipe("llm", config=config)**，在SpaCy模型中添加了一个名为**llm**的新管道组件。这个组件负责将文本发送到GPT模型进行处理，并将结果（如识别的命名实体）集成回SpaCy的文档对象中。
5. **执行NER任务**： 一旦管道设置完成，就可以像使用SpaCy的其他组件一样使用它来处理文本，执行命名实体识别任务。文本将通过SpaCy的处理管道发送给GPT模型，由GPT分析并返回识别的实体。

# Chapter2

## Summary：

different ways of using language models to improve existing search systems

* Dense retrieval, which relies on the similarity of text embeddings. These are systems that embed a search query and retrieve the documents with the nearest embeddings to the query’s embedding.
* Rerankers, systems (like monoBERT) that look at a query and candidate results and scores the relevance of each document to that query. These relevance scores are then used to order the shortlisted results according to their relevance to the query often producing an improved results ranking.
* Generative search, where search systems that have a generative LLM at the end of the pipeline to formulate an answer based on retrieved documents while citing its sources.

One of possible Methods evaluating

* Mean Average Precision allows us to score search systems to be able to compare across a test suite of queries and their known relevance to the test queries.

# Chapter3

## Summary:

BERTopic: By leveraging a modular structure, we used a variety of Large Language Models to create document representations and fine-tune topic representations. We extracted the topics found in ArXiv abstracts and saw how we could use BERTopic’s modular structure to develop different kinds of topic representations.

# Chapter4 Tokens & Token Embeddings

## Summary

In this chapter, we have covered LLM tokens, tokenizers, and useful approaches to use token embeddings beyond language models.

* Tokenizers are the first step in processing the input to a LLM -- turning text into a list of token IDs.
* Some of the common tokenization schemes include breaking text down into words, subword tokens, characters, or bytes.
* A tour of real-world pre-trained tokenizers (from BERT to GPT2, GPT4, and other models) showed us areas where some tokenizers are better (e.g., preserving information like capitalization, new lines, or tokens in other languages) and other areas where tokenizers are just different from each other (e.g., how they break down certain words).
* Three of the major tokenizer design decisions are the tokenizer algorithm (e.g., BPE, WordPiece, SentencePiece), tokenization parameters (including vocabulary size, special tokens, capitalization, treatment of capitalization and different languages), and the dataset the tokenizer is trained on.
* Language models are also creators of high-quality contextualized token embeddings that improve on raw static embeddings. Those contextualized token embeddings are what’s used for tasks including NER, extractive text summarization, and span classification.
* Before LLMs, word embedding methods like word2vec, Glove and Fasttext were popular. They still have some use cases within and outside of language processing. The Word2Vec algorithm relies on two main ideas: Skipgram and Negative Sampling. It also uses contrastive training similar to the one we’ll see in the contrastive training chapter.
* Token embeddings are useful for creating and improving recommender systems as we’ve seen in the music recommender we’ve built from curated song playlists.