Text classification

WORKING WITH HUGGING FACE

Jacob H. Marquez
Lead Data Engineer

datacamp

Text classification: Sentiment analysis

- Labels text based on its emotional tone

Text		Category
"I love pineapple on pizza!"	→	Positive
"I don't like pineapple on pizza!"		Negative

- **Applications:** Analyzing reviews, tracking social media sentiment

Sentiment analysis: coding example

```python
from transformers import pipeline

my_pipeline = pipeline(
    "text-classification",
    model="distilbert-base-uncased-finetuned-sst-2-english"
)
print(my_pipeline("Wi-Fi is slower than a snail today!"))
```

```
[{'label': 'NEGATIVE', 'score': 0.99}]
```

Text classification: Grammatical correctness

- Evaluates text grammar for correctness

Text		Category
"This course is great!"	→	Acceptable
"Course is gravy."		Unacceptable

- **Applications:** Grammar checkers, language learning tools

Grammatical correctness: coding example

```python
from transformers import pipeline

# Create a pipeline for grammar checking
grammar_checker = pipeline(
    task="text-classification",
    model="abdulmatinomotoso/English_Grammar_Checker"
)
# Check grammar of the input text
print(grammar_checker("He eat pizza every day."))
```

```
[{'label': 'LABEL_0', 'score': 0.99}]
```

Text classification: QNLI

Question: "What state is Hollywood in?"		Category
Premise: "Hollywood is in California."	→	Entailment (True)
Premise: "Hollywood is known for its movies."	→	Not Entailment (False)

- Checks if a premise answers a question
- **Applications:** Q&A systems, fact-checking

QNLI: coding example

```python
from transformers import pipeline

classifier = pipeline(
    task="text-classification",
    model="cross-encoder/qnli-electra-base"
)

classifier("Where is Seattle located?, Seattle is located in Washington state.")
```

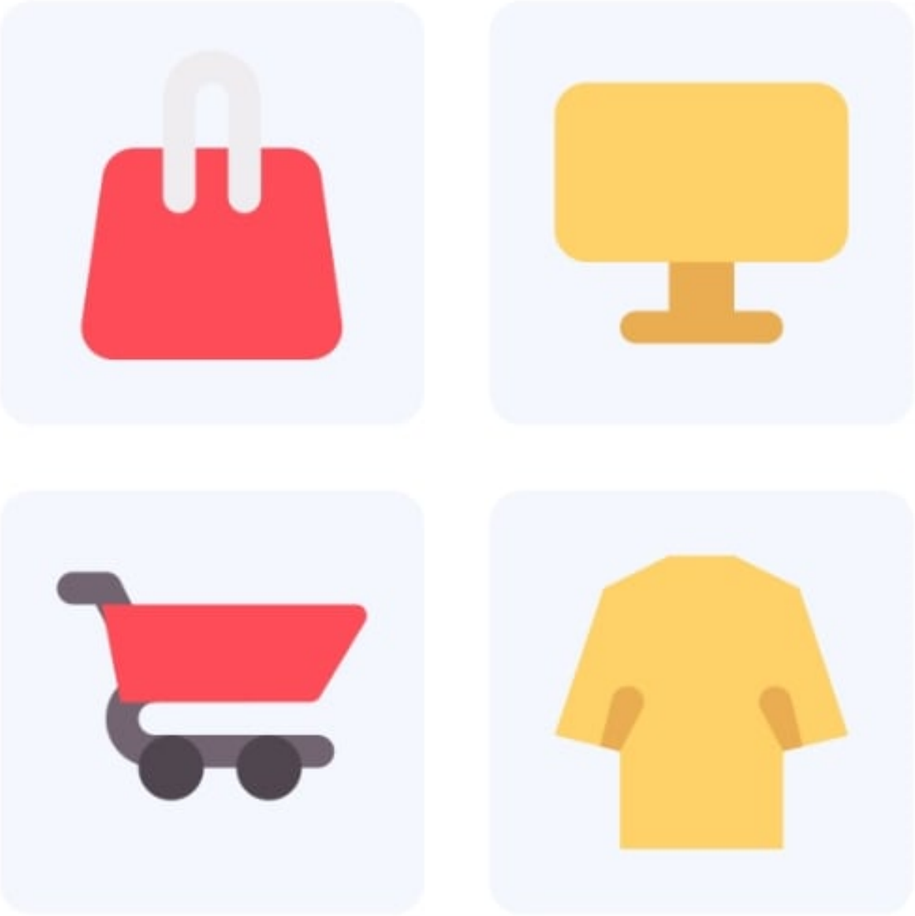
```
[{'label': 'LABEL_0', 'score': 0.997}]
```

Text classification: Dynamic category assignment

- Dynamically assigns categories based on content

```
Request:
"I want to know more about your pricing
plans."

Categories:
  • Sales: → High Confidence (True)
  • Marketing: → Moderate Confidence (False)
  • Support: → Low Confidence (False)
```

- **Applications:** Content moderation, recommendation systems



Dynamic category assignment: coding example

```
classifier = pipeline(
  task="zero-shot-classification",
  model="facebook/bart-large-mnli")

text = "Hey, DataCamp; we would like to feature your courses in our newsletter!"
categories = ["marketing", "sales", "support"]

output = classifier(text, categories)

print(f"Top Label: {output['labels'][0]} with score: {output['scores'][0]}")
```

```
Top Label: support with score: 0.8183
```

Challenges of text classification

Ambiguity

Challenges of text classification

Ambiguity

Sarcasm, Irony

Challenges of text classification

Ambiguity

Sarcasm, Irony

Multilingual

Let's practice!

WORKING WITH HUGGING FACE

Text summarization

WORKING WITH HUGGING FACE

Jacob H. Marquez
Lead Data Engineer

What is summarization?

Original Text

David G. Robinson is a data scientist at the Heap analytics company. He is a co-author of the tidytext R programming language package and the O'Reilly book, Text Mining with R. Robinson has previously worked as a chief data scientist at DataCamp and as a data scientist at Stack Overflow. He was also a data engineer at Flatiron Health in 2019.

→

Summarized Text

David G. Robinson is a data scientist. He is a co-author of the tidytext R package and the O'Reilly book.

Extractive vs. Abstractive

Extractive:

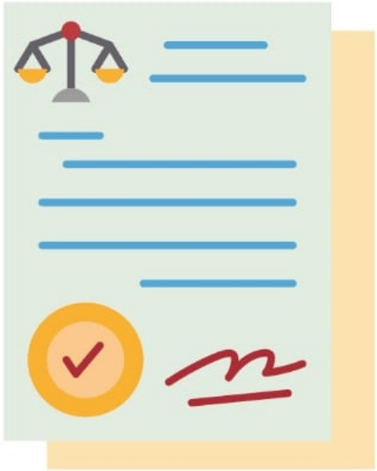
- Selects key sentences from the text
- Efficient, needs fewer resources
- Lacks flexibility; may be less cohesive

Abstractive:

- Generates new, rephrased text
- Clearer and more readable
- Requires more resources and processing

Use cases of extractive summarization

- **Legal Documents:** Highlights key clauses



- **Financial Research:** Extracts insights

Use cases of abstractive summarization

- **News Articles:** Creates concise summaries

- **Content Recommendations:** Generates compelling descriptions

Extractive summarization in action

```python
from transformers import pipeline

# Load the extractive summarization pipeline
summarizer = pipeline("summarization", model="nyamuda/extractive-summarization")
```

```python
text = "This is my really large text about Data Science..."
summary_text = summarizer(text)
```

```python
print(summary_text[0]['summary_text'])
```

```
"data science is a field that combines mathematics, statistics...."
```

Abstractive summarization in action

```python
from transformers import pipeline

# Load the abstractive summarization pipeline
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")

text = "This is my really large text about Data Science..."
summary_text = summarizer(text)
print(summary_text[0]['summary_text'])
```

```
"The global data science platform market is projected
 is projected to reach $140.9 billion by 2025..."
```

Parameters for summarization

- `min_new_tokens` & `max_new_tokens` : Control summary length

```
summarizer = pipeline(task="summarization", min_new_tokens=10, max_new_tokens=150)
```

Let's practice!

WORKING WITH HUGGING FACE

Auto Models and Tokenizers

WORKING WITH HUGGING FACE

Jacob H. Marquez
Lead Data Engineer

datacamp

Pipelines: fast and simple

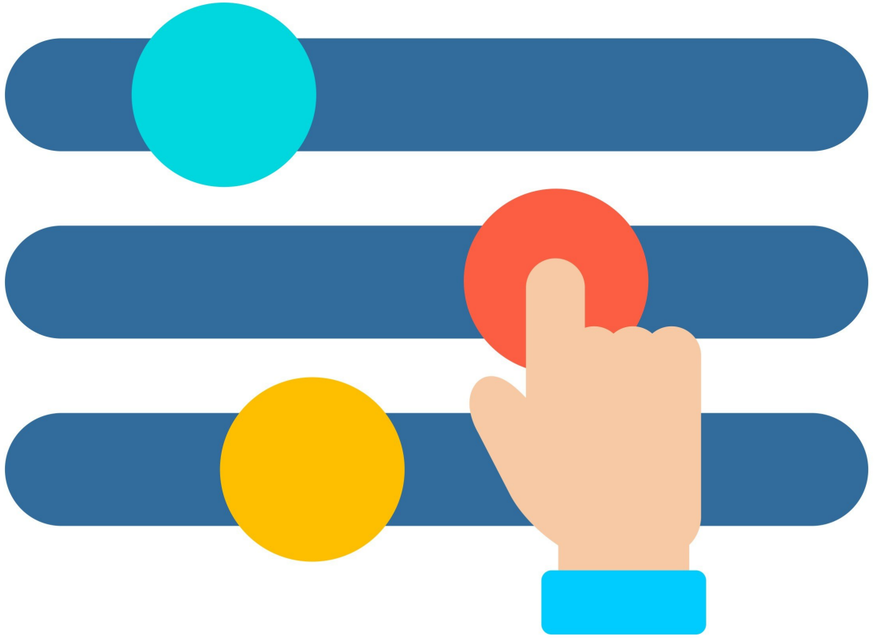
```python
from transformers import pipeline

my_pipeline = pipeline(
    "text-classification",
    model="distilbert-base-uncased-finetuned-sst-2-english"))

print(my_pipeline("Wi-Fi is slower than a snail today!"))
```

```
[{'label': 'NEGATIVE', 'score': 0.99}]
```

Auto Classes: flexible and powerful

- **Auto classes:** Flexible access to models and tokenizers
- **More control** over model behavior and outputs
- **Perfect for advanced tasks**

- Pipelines = quick; Auto classes = flexible



AutoModels

- Choose an AutoModel class to directly download a model

```
from transformers import AutoModelForSequenceClassification

# Download a pre-trained text classification model
model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert-base-uncased-finetuned-sst-2-english"
)
```

AutoTokenizers

- Prepare text input data
- Recommended to use the **tokenizer paired with the model**

```
from transformers import AutoTokenizer

# Retrieve the tokenizer paired with the model
tokenizer = AutoTokenizer.from_pretrained(
    "distilbert-base-uncased-finetuned-sst-2-english"
)
```

Tokenizing text with AutoTokenizer

- Tokenizers **clean input and split text** into tokens

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

# Tokenize input text
tokens = tokenizer.tokenize("AI: Helping robots think and humans overthink:)")
print(tokens)
```

```
['ai', ':', 'helping', 'robots', 'think', 'and',
 'humans', 'over', '##thi', '##nk', ':', ')']
```

Different models, different tokenizers

- Our model (distilbert-base-uncased):

```
['ai', ':', 'helping', 'robots', 'think', 'and', 'humans', 'over', '##thi',
'##nk', ':', ')']
```

- BERT-Base-Cased Tokenizer:

```
['AI', ':', 'Help', '##ing', 'robots', 'think', 'and', 'humans', 'over',
'##thin', '##k', ':', ')']
```

Building a Pipeline with Auto Classes

```
from transformers import AutoModelForSequenceClassification,
AutoTokenizer, pipeline

# Download the model and tokenizer
my_model = AutoModelForSequenceClassification.from_pretrained(
  "distilbert-base-uncased-finetuned-sst-2-english")
my_tokenizer = AutoTokenizer.from_pretrained(
  "distilbert-base-uncased-finetuned-sst-2-english")

# Create the custom pipeline
my_pipeline = pipeline(
  task="sentiment-analysis", model=my_model, tokenizer=my_tokenizer)
```

Use Cases for AutoModels and AutoTokenizers

- Use for more control and customization
- **Text Preprocessing:** Clean and tokenize for specific use cases
- **Thresholding:** Prioritize key categories in classification tasks
- **Complex Workflows:** Control multi-stage processing and integration

Let's practice!

WORKING WITH HUGGING FACE

Document Q&A

WORKING WITH HUGGING FACE

Jacob H. Marquez
Lead Data Engineer

datacamp

What is document question and answering?

- Answers questions from document content
- Requires a document and a question
- Provides direct or paraphrased answers

Question: "What is the total revenue of Q3?"

Memo to: Finance Department

Date: October 20, 2023

Page 1

Quarterly Financial Update

- Total revenue for Q3 was **$10.5 million**, marking a **15% increase** from the previous quarter.
- Operating expenses for Q3 totaled **$4.2 million**, resulting in a net profit of **$6.3 million**.
- Key drivers of growth included increased product demand and an expanded customer base.

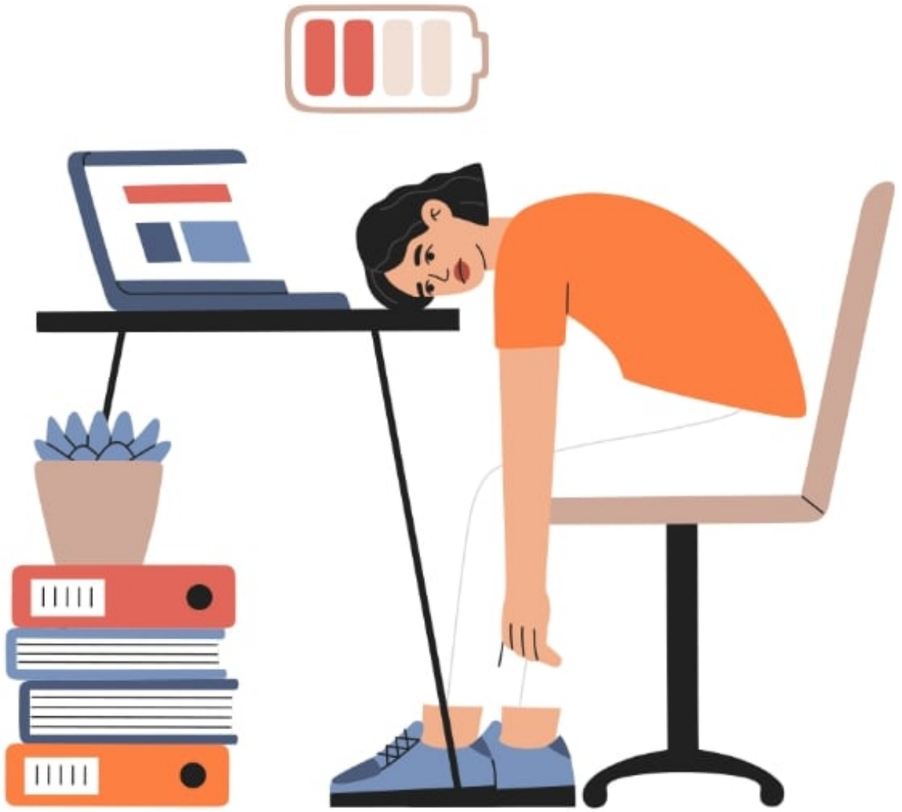
For any questions regarding this update, please contact **the finance team** directly.

Use cases for document Q&A

- **Legal:** Identify contract clauses
- **Finance:** Extract key figures
- **Support:** Retrieve answers from manuals

Automating HR queries with document Q&A

- Info stored in `US-Employee_Policy.pdf`
- Build a system to extract answers
- Save HR time and effort



Extracting text with pypdf

```python
from pypdf import PdfReader

# Load the PDF file
reader = PdfReader("US-Employee_Policy.pdf")

# Extract text from all pages
document_text = ""
for page in reader.pages:
    document_text += page.extract_text()
```

```
Welcome to the US Employee Policy document...
```

Creating a Q&A pipeline

```
# Load the question-answering pipeline
qa_pipeline = pipeline(
    task="question-answering",
    model="distilbert-base-cased-distilled-squad")

question = "How many volunteer days are offered annually?"

# Get the answer from the QA pipeline
result = qa_pipeline(question=question, context=document_text)
print(f"Answer: {result['answer']}")
```

```
Answer: 1
```

Bringing it all together

- Use `PdfReader` from `pypdf` to load and read PDF files
- Extract text with `.pages` and `.extract_text()` into `document_text`
- Set up a `question-answering` pipeline
- Pass a `question` and `context` to the pipeline
- Wrap into functions to automate queries

Let's practice!

WORKING WITH HUGGING FACE

Congratulations!

WORKING WITH HUGGING FACE

Jacob H. Marquez
Lead Data Engineer

datacamp

Chapter 1

- **Hugging Face Hub:** A platform for discovering AI models and datasets

Chapter 2

Text classification

- **Sentiment Analysis:** Understand customer emotions
- **Grammar Checks:** Identify errors in text
- **Category Assignment:** Classify requests with scores

```
pipeline(task="text-classification")
```

```
[{'label': 'POSITIVE', 'score': 0.97}]
```

Text summarization

Original Text

David G. Robinson is a data scientist at the Heap analytics company. He is a co-author of the tidytext R programming language package and the O'Reilly book, Text Mining with R. Robinson has previously worked as a chief data scientist at DataCamp and as a data scientist at Stack Overflow. He was also a data engineer at Flatiron Health in 2019.

→

Summarized Text

David G. Robinson is a data scientist. He is a co-author of the tidytext R package and the O'Reilly book.

- **Pipeline:** Specify `pipeline(task="summarization")`
- **Output Length:** Adjust with `min_new_tokens` and `max_new_tokens`

Document Q&A

- **PDF Processing:** Use `.pages` and `.extract_text()` from `pypdf` to extract text
- **Q&A Pipeline:** Specify `pipeline(task="question-answering")`

Auto Models and Tokenizers

Congratulations and Thank You!

WORKING WITH HUGGING FACE