Opinion Miner

Contents

Introduction

Opinion mining involves extracting aspects from free-form text along with their associated sentiments from one or more documents. This has many uses in industry, such as enabling companies to evaluate customer sentiment toward different aspects of their products at scale. Because user feedback is often written, sophisticated natural language processing is required to analyse it systematically. This notebook solves the task using two interacting subsystems: aspect extraction and sentiment analysis.

Aspect extraction is modelled as a sequence-labelling problem using conditional random fields (CRFs) to predict labels from features derived from the reviews. Sentiment analysis introduces two contrasting solutions for assigning sentiment to the aspects extracted by the first subsystem, a rules-based approach and a transformer-based approach. Both variants are implemented and critically compared.

Running this Notebook

Note on runtime

The entire notebook runs on my device, an Apple MacBook Pro M3 Max, in approximately 25 minutes without any pre-calculation. 13 minutes of that time is spent on sentiment analysis training. I implemented a caching functionality and pre-calculated the weights, but I cannot find a way to upload them with my assignment. I tried to upload the model weights but ePortfolio has a limit of 256mb and the weights are 400mb. Therefore, you will probably have to train from scratch and if your device is significantly less powerful than mine, it may take a while. I apologise for this, I assumed there would be a way to make this work and didn't realise until submitting the day before. In addition, roughly 9 minutes is spent on aspect extraction hyperparameter tuning, but no caching functionality was implemented here. Many apologies if this makes marking difficult and please let me know if I can help.

To run this notebook from scratch, create a Python virtual environment and install project dependencies as specified in "./requirements.txt".

1. Install Python v3.9.6
2. Navigate to this project's directory and execute the following:

```
python3.9 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip wheel setuptools
```

```
pip install -r requirements.txt
python -m spacy download en_core_web_lg
```

3. Ensure the notebook kernel points to the newly created venv

The below code imports all relevant dependencies for this notebook. It also establishes the `Constants` class, which will define common constants used throughout the notebook.

In [3]:
```
"""
Import all dependencies for the project and establish common constants and utility functions.
"""
import warnings

warnings.filterwarnings(
    "ignore",
    message="`resume_download` is deprecated",
    category=FutureWarning,
    module="huggingface_hub.file_download",
)
warnings.filterwarnings("ignore", message=".*pin_memory.*MPS.*", category=UserWarning)
warnings.filterwarnings("ignore", message=r".*The component 'matcher' does not have any pattern

import collections
import os
import re
import time
from pathlib import Path
from statistics import mean
from typing import List
from typing import Tuple

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import nltk
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import spacy
import torch
from joblib import Parallel, delayed
from nltk.corpus import wordnet as wn
from seqeval.metrics import f1_score as seq_f1
from sklearn.model_selection import ParameterGrid, KFold
from sklearn_crfsuite import CRF
from sklearn.metrics import f1_score as sk_f1, precision_score, recall_score, precision_recall_
from spacy.matcher import PhraseMatcher, Matcher
from torch.utils.data import Dataset
from transformers import (
    BertTokenizerFast,
    BertForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding,
    set_seed
)
from transformers.utils.logging import set_verbosity_error
from typing import Dict, Any
from bertviz import head_view
from scipy.stats import linregress
from IPython.display import display, HTML

os.environ["TOKENIZERS_PARALLELISM"] = "false"
set_verbosity_error()
pd.set_option("display.max_colwidth", None)
pd.set_option("display.max_rows", 50)
set_seed(8)

nlp = spacy.load("en_core_web_lg", disable=["ner"])
nltk.download("wordnet")
nltk.download("omw-1.4")

class Constants():
    ADJECTIVE_POS = "ADJ"
```

```
    B_ASP = "B-ASP"
    I_ASP = "I-ASP"
    OTHER = "O"
    SAVE_DIR = "models/aspect-bert-targeted"
```

```
/Users/jwestgomila/.pyenv/versions/3.9.6/lib/python3.9/site-packages/tqdm/auto.py:21: TqdmWarnin
g: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedoc
s.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/jwestgomila/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/jwestgomila/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [4]:
```
class FigureUtilities():
    TABLE_NUMBER = 1
    FIGURE_NUMBER = 1
    TABLE_NUMBER_MAP = dict()
    FIGURE_NUMBER_MAP = dict()

    @staticmethod
    def table_caption(table_id, caption):
        if FigureUtilities.TABLE_NUMBER_MAP.get(table_id):
            print(f"Table {FigureUtilities.TABLE_NUMBER_MAP[table_id]}. {caption}")
        else:
            print(f"Table {FigureUtilities.TABLE_NUMBER}. {caption}")
            FigureUtilities.TABLE_NUMBER_MAP[table_id] = FigureUtilities.TABLE_NUMBER
            FigureUtilities.TABLE_NUMBER += 1

    @staticmethod
    def figure_caption(fig_id, caption):
        if FigureUtilities.FIGURE_NUMBER_MAP.get(fig_id):
            print(f"Figure {FigureUtilities.FIGURE_NUMBER_MAP[fig_id]}. {caption}")
        else:
            print(f"Figure {FigureUtilities.FIGURE_NUMBER}. {caption}")
            FigureUtilities.FIGURE_NUMBER_MAP[fig_id] = FigureUtilities.FIGURE_NUMBER
            FigureUtilities.FIGURE_NUMBER += 1
```

Dataset Ingestion

The below code sets up the parser used to ingest the customer review files in the "./Data/..." directory. The parser ignores the two "Readme.txt" files found in "./Data/Customer_review_data" and "./Data/CustomerReviews-3_domains/". If bytes cannot be decoded, they are replaced with the Unicode replacement character.

Throughout the dataset, a new individual review is identified by a line starting with a "[t]" string, indicating the start of a review title, with each subsequent line being a sentence, an annotated sentence, another review title or the end of the file. Sentences begin immediately after the "##" pattern, and annotations, if applicable, appear immediately before it. This parser aggregates sentences into review objects, associating annotations with their respective sentences where applicable. All of "./Data/CustomerReviews-3_domains/" and 2 files in "./Data/Reviews-9-products" are missing review separators; in these cases, each sentence is treated as a separate review to increase test data quantity.

Ground Truth

Aspects will be converted to lower case and stripped of extraneous whitespace whenever they are uesd in evaluation throughout this notebook.

In [5]:
```
"""
Defines the review parsing utility and ingests the datasets.
"""
class ReviewParser():
    PATTERN = re.compile(
        r'\s*(?P<feature>[^,\[]+?)\[(?P<sentiment>[+-]?\d+)\](?:\[(?P<extra>[^\]]+)\])?\s*(?:,|
        re.VERBOSE)

    REVIEW_MARK = re.compile(r'^\s*\[t]', re.I)
```

```
    def parse_line(self, raw: str):
        annotation_part, sentence = raw.split("##", 1)
        sentence = sentence.strip()
        annotations = (
            [m.groupdict() for m in self.PATTERN.finditer(annotation_part)]
            if annotation_part.strip()
            else [])
        return {"text": sentence, "annotations": annotations}

    def parse_reviews(self, directory, encoding='cp1252'):
        reviews = []
        for fname in sorted(os.listdir(directory)):
            if fname.lower().startswith("readme"):
                continue

            path = os.path.join(directory, fname)
            with open(path, encoding=encoding, errors='replace') as f:
                lines = f.readlines()

            marker_idxs = [i for i, line in enumerate(lines) if self.REVIEW_MARK.match(line)]
            has_markers = len(marker_idxs) > 1 or (len(marker_idxs) == 1 and marker_idxs[0] !=

            review_id = 0
            current_review = {"review_id": None, "sentences": []}

            def flush():
                nonlocal review_id, current_review
                if current_review["sentences"]:
                    reviews.append(current_review)
                review_id += 1
                stem = Path(fname).stem
                current_review = {
                    "review_id": f"{stem}-{review_id:03}",
                    "product": stem,
                    "sentences": [],
                }

            if has_markers:
                for raw in lines:
                    if self.REVIEW_MARK.match(raw):
                        flush()
                        continue
                    if "##" not in raw:
                        continue
                    line = self.parse_line(raw)
                    current_review["sentences"].append(line)
                flush()
            else:
                for raw in lines:
                    if "##" not in raw:
                        continue
                    flush()
                    line = self.parse_line(raw)
                    current_review["sentences"].append(line)

            flush()

        return reviews

    @staticmethod
    def get_sentences(reviews):
        return [sentence["text"] for review in reviews for sentence in review["sentences"]]

parser = ReviewParser()
cust_sents = parser.parse_reviews("./Data/Customer_review_data")
prod9_sents = parser.parse_reviews("./Data/Reviews-9-products")
reviews3_sents = parser.parse_reviews("./Data/CustomerReviews-3_domains")

all_reviews = cust_sents + prod9_sents + reviews3_sents

data = {
    "Dataset": [
        "Customer_review_data",
        "Reviews-9-products",
```

```
        "CustomerReviews-3_domains",
        "TOTAL"
    ],
    "Number of Reviews": [
        len(cust_sents),
        len(prod9_sents),
        len(reviews3_sents),
        len(all_reviews)
    ]
}

df = pd.DataFrame(data)
display(df)
FigureUtilities.table_caption("review_metrics_table", "Showing review count per dataset, with a
```

	Dataset	Number of Reviews
0	Customer_review_data	313
1	Reviews-9-products	1083
2	CustomerReviews-3_domains	2099
3	TOTAL	3495

Table 1. Showing review count per dataset, with an additional row at the bottom showing the tota
l count.

Dataset Qualitative Evaluation

The datasets are of mixed provenance, with Customer review data and CustomerReviews-3 domains containing Amazon product reviews annotated as part of opinion mining studies (Hu and B. Liu, 2004b; Q. Liu et al., 2015), but Reviews-9-products is of unknown origin.

Some of the annotations are of suspect quality. For example, in "./Data/Customer_review_data/Apex AD2600 Progressive-scan DVD player.txt", there is the review: "play[-2], disney movie[-2]##many of our disney movies do n't play on this dvd player". "Disney movie" is not a feature of the DVD player and a better annotator would instead extract an implicit feature like "compatibility". However, manual inspection reveals many of the annotations are of high quality; given the historical context of these datasets, they are fit for purpose.

The English language follows a Zipf distribution (Zipf, 1949). Plotting whole corpus token log frequency against log rank reveals a Zipf-like distribution (Figure 1), meaning if the system performs well on this corpus, it should generalise well to other domains.

In [6]:
```
"""
Plotting distribution of token counts and comparing to a Zipf distribution.
"""
def plot_zipf():
    all_sentences = ReviewParser.get_sentences(all_reviews)
    alpha_words = [token.text for text in all_sentences for token in nlp(text) if token.is_alph

    freq = collections.Counter(alpha_words)
    alpha_words_counts = np.array(sorted(freq.values(), reverse=True))

    ranks = np.arange(1, len(alpha_words_counts) + 1)
    log_r = np.log10(ranks)
    log_f = np.log10(alpha_words_counts)
    slope, intercept, r_val, *_ = linregress(log_r, log_f)
    c_guideline = log_f[0] + log_r[0]
    y_guideline = c_guideline - log_r

    plt.figure(figsize=(5, 4))
    plt.scatter(log_r, log_f, s=6, alpha=0.35, label='data')
    plt.plot(
        log_r,
        intercept + slope * log_r,
        'r',
        label=f'fit(slope={slope:.2f}, $R^2$={r_val ** 2:.2f})')
    plt.plot(
        log_r,
        y_guideline,
        'k--',
```

```
        linewidth=1,
        label='Zipf guideline (slope≈−1)')
    plt.xlabel(r'$\log_{10}(\mathrm{rank})$')
    plt.ylabel(r'$\log_{10}(\mathrm{frequency})$')
    plt.title(f'Zipf Rank–Frequency Plot')
    plt.legend()
    plt.tight_layout()
    plt.show()
    FigureUtilities.figure_caption("zipf_plot", "Plotting log frequency against rank reveals th

plot_zipf()
```

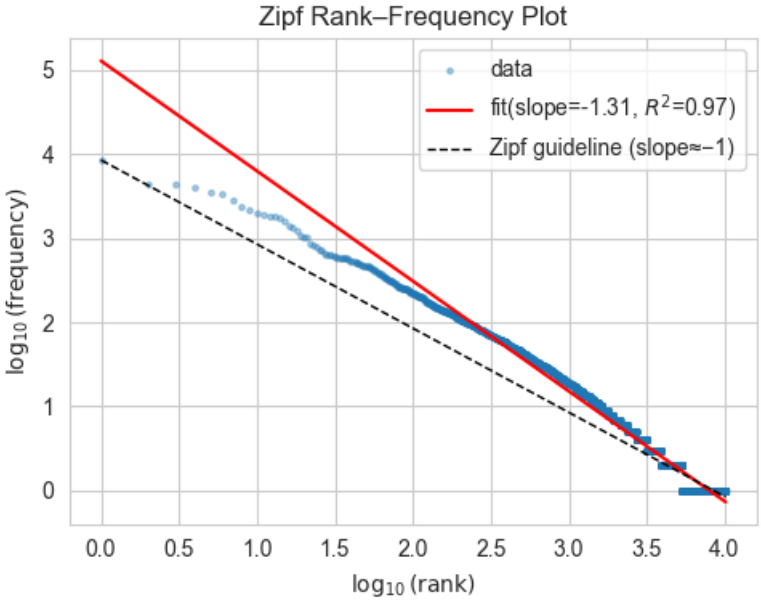

Figure 1. Plotting log frequency against rank reveals the corpus resembles a Zipf distribution, confirming that the frequency of a word is inversely proportional to its rank.

Dataset Partitioning

The dataset is partitioned into an 80% training set and a 20% test set using stratified random sampling by product name and overall review sentiment. Stratifying by product ensures proportional domain coverage across splits; users tend to discuss similar aspects within a domain, so an imbalance here could degrade system performance. Overall review sentiment is computed by aggregating sentence-level scores and normalising to 1, -1, or 0 according to the sign of the total. Maintaining proportional product and sentiment representation is key to obtaining representative system evaluation metrics.

Limitations

Only 55.8% of sentences in training and 54.7% in test are annotated (Table 1), so the trainable dataset is about half of what it appears. This sparsity can reduce $\mathrm{Precision}$ and Recall, but it also helps reduce overfitting by preserving a balance between annotated and unannotated sentences. Down-sampling unannotated sentences could be explored, though it would likely trade Recall for $\mathrm{Precision}$ downstream.

In [7]:
```
"""
Review partitioning functionality, designed to split the dataset into training and test sets wi
Displays partition results.
"""
class ReviewPartitionUtility:
    """
    Review partition utility class encapsulating various partitioning behaviours.
    """
    @staticmethod
    def review_sentiment_label(review):
        """
        Generate an aggregate sentiment label from a review.

        :param review: the review
        :return: the aggregate sentiment label
        """
        scores = [int(annotation["sentiment"]) for sentence in review["sentences"] for annotati
```

```
                    sentence["annotations"]]

        if not scores:
            return 0

        mean_scores = mean(scores)
        if mean_scores > 0:
            return 1
        elif mean_scores < 0:
            return -1
        else:
            return 0

    @staticmethod
    def count_sentences(reviews):
        """
        Count the number of sentences in a list of reviews.

        :param reviews: a list of reviews
        :return: the number of sentences
        """
        return sum(len(r["sentences"]) for r in reviews)

    @staticmethod
    def count_annotated(reviews):
        """
        Count the number of annotated sentences in a list of reviews.

        :param reviews: the reviews
        :return: the number of annotated sentences
        """
        count = 0
        for review in reviews:
            for sent in review["sentences"]:
                count += len(sent["annotations"])
        return count

    @staticmethod
    def stratify(reviews):
        """
        Stratifies reviews with respect to product name and aggregate sentiment polarity.

        Removes product/sentiment pairs with fewer than two examples.
        :param reviews: the reviews
        :return: the stratified reviews
        """
        y_strat = [(review['product'], ReviewPartitionUtility.review_sentiment_label(review)) f

        print(f"Loaded {len(reviews)} reviews")
        print(f"Created {len(y_strat)} stratification keys")

        low_sentiment_sample_reviews = {x[0] for x in collections.Counter(y_strat).items() if x

        print(f"Found {len(low_sentiment_sample_reviews)} product/sentiment pairs with less tha

        y_strat, reviews = zip(*[pair for pair in zip(y_strat, reviews) if pair[0] not in low_s

        print(f"{len(reviews)} reviews remain after low sample filtering")
        print(f"{len(y_strat)} stratification keys remain after low sample filtering\n")

        return sklearn.model_selection.train_test_split(
            reviews,
            y_strat,
            test_size=0.2,
            random_state=8,
            stratify=y_strat
        )

# Calculate the train/test split
train_reviews, test_reviews, y_train, y_test = ReviewPartitionUtility.stratify(all_reviews)

# Show train/test split reviews, sentences and annotated sentences counts
def display_train_test_split_annotation_balance():
    data = {
        "Split": ["Train", "Test"],
```

```
        "Reviews": [
            len(train_reviews),
            len(test_reviews)
        ],
        "Sentences": [
            ReviewPartitionUtility.count_sentences(train_reviews),
            ReviewPartitionUtility.count_sentences(test_reviews),
        ],
        "Annotated": [
            ReviewPartitionUtility.count_annotated(train_reviews),
            ReviewPartitionUtility.count_annotated(test_reviews),
        ],
    }

    df = pd.DataFrame(data)
    print()
    display(df)
    FigureUtilities.table_caption("annot_metrics", "Showing the annotation balance between the

# Show train/test split by product and aggregate sentiment
def display_train_test_split_annotation_metrics():
    train_counts = collections.Counter(y_train)
    test_counts = collections.Counter(y_test)
    labels = sorted(set(train_counts) | set(test_counts))

    df = pd.DataFrame({
        'label': labels,
        'train_count': [train_counts[label] for label in labels],
        'test_count': [test_counts[label] for label in labels],
    })
    display(df)
    FigureUtilities.table_caption("annot_balance_split", "Showing the review split per product

display_train_test_split_annotation_metrics()
display_train_test_split_annotation_balance()
```

```
Loaded 3495 reviews
Created 3495 stratification keys
Found 1 product/sentiment pairs with less than two samples
3494 reviews remain after low sample filtering
3494 stratification keys remain after low sample filtering
```

	label	train_count	test_count
0	(Apex AD2600 Progressive-scan DVD player, -1)	50	12
1	(Apex AD2600 Progressive-scan DVD player, 0)	2	0
2	(Apex AD2600 Progressive-scan DVD player, 1)	28	7
3	(Canon G3, -1)	4	1
4	(Canon G3, 0)	2	0
5	(Canon G3, 1)	30	8
6	(Canon PowerShot SD500, -1)	20	5
7	(Canon PowerShot SD500, 0)	83	21
8	(Canon PowerShot SD500, 1)	80	20
9	(Canon S100, -1)	2	1
10	(Canon S100, 0)	7	2
11	(Canon S100, 1)	31	8
12	(Computer, -1)	50	12
13	(Computer, 0)	237	59
14	(Computer, 1)	138	35
15	(Creative Labs Nomad Jukebox Zen Xtra 40GB, -1)	27	7
16	(Creative Labs Nomad Jukebox Zen Xtra 40GB, 0)	5	1
17	(Creative Labs Nomad Jukebox Zen Xtra 40GB, 1)	44	11
18	(Diaper Champ, -1)	9	2
19	(Diaper Champ, 0)	2	0
20	(Diaper Champ, 1)	29	7
21	(Hitachi router, -1)	2	1
22	(Hitachi router, 1)	22	5
23	(Linksys Router, -1)	8	2
24	(Linksys Router, 0)	3	1
25	(Linksys Router, 1)	27	7
26	(MicroMP3, -1)	14	3
27	(MicroMP3, 0)	2	0
28	(MicroMP3, 1)	25	6
29	(Nikon coolpix 4300, -1)	4	1
30	(Nikon coolpix 4300, 1)	23	6
31	(Nokia 6600, -1)	11	3
32	(Nokia 6600, 0)	2	0
33	(Nokia 6600, 1)	26	7
34	(Nokia 6610, -1)	2	1
35	(Nokia 6610, 0)	5	1
36	(Nokia 6610, 1)	25	6
37	(Router, -1)	81	20
38	(Router, 0)	510	128
39	(Router, 1)	112	28
40	(Speaker, -1)	46	12
41	(Speaker, 0)	316	79
42	(Speaker, 1)	189	47

	label	train_count	test_count
43	(ipod, -1)	46	12
44	(ipod, 0)	295	74
45	(ipod, 1)	82	21
46	(norton, -1)	26	6
47	(norton, 0)	3	1
48	(norton, 1)	8	2

Table 2. Showing the review split per product and sentiment score for the partitioned datasets.

	Split	Reviews	Sentences	Annotated
0	Train	2795	8509	4752
1	Test	699	1780	973

Table 3. Showing the annotation balance between the partitioned datasets.

Opinion Word Lexicon

An opinion word lexicon is sourced externally from tidytext (Julia Silge and David Robinson, 2025) and cached locally in `./Data/sentiments.csv`. This vocabulary contains $6,783$ opinion words, of which $1,777$ are adjectives. It is used in both subsystems; aspect extraction employs the adjectives to identify sentences containing opinion words, while the rules-based sentiment analysis algorithm uses the full lexicon as a seed for orientation propagation.

In [8]:
```
"""
Fetches an opinion word lexicon from https://raw.githubusercontent.com/juliasilge/tidytext, cac
"""
def get_or_download(path="Data/sentiments.csv", url="https://raw.githubusercontent.com/juliasil
    fp = Path(path)
    if fp.exists() and fp.stat().st_size > 0:
        print(f"Loading existing file: {path}")
        return pd.read_csv(fp)

    print(f"File missing or empty. Building data and writing to {path}")
    df = pd.read_csv(url)
    df.to_csv(path, index=False, encoding="utf-8")
    return df

def get_adjectives_from_external_source():
    sentiments = get_or_download()
    opinion_words = sentiments['word']
    seed_opinion_words = set(opinion_words)
    opinion_words_counter = collections.Counter(opinion_words)
    non_unique_opinion_words = [entry[0] for entry in opinion_words_counter.items() if entry[1]

    print(f"Found {len(non_unique_opinion_words)} non-unique opinion words in seed lexicon: {no
    print(f"Retrieved {len(seed_opinion_words)} opinion words\n")

    seed_opinion_words_docs = list(nlp.pipe(seed_opinion_words, disable=["ner", "lemmatizer"]))

    # Filter to adjectives
    adjective_opinion_words = [
        token.text
        for doc in seed_opinion_words_docs
        for token in doc
        if token.pos_ == Constants.ADJECTIVE_POS and token.is_alpha
    ]

    df_adj = pd.DataFrame({"Adjective Opinion Words": adjective_opinion_words})
    display(df_adj)
    FigureUtilities.table_caption("opinion_words_table", f"Identified {len(adjective_opinion_wo

    return sentiments, adjective_opinion_words

sentiments, adjective_opinion_words = get_adjectives_from_external_source()
```

```
Loading existing file: Data/sentiments.csv
Found 3 non-unique opinion words in seed lexicon: ['envious', 'enviously', 'enviousness']
Retrieved 6783 opinion words
```

	Adjective Opinion Words
0	maniacal
1	appalling
2	injudicious
3	superficial
4	complementary
...	...
1772	wary
1773	obsessive
1774	irascible
1775	bland
1776	uneconomical

1777 rows × 1 columns

```
Table 4. Identified 1777 adjectives in opinion words from the sentiments dataset, and opinion wo
rds are identified by calculating their part-of-speech.
```

Aspect Extraction

Conditional random fields (CRFs) have been shown to be highly effective at aspect extraction by framing the task as a sequence labelling problem (Shu, Xu and Liu, 2017). CRFs are discriminative probabilistic graphical models that directly model the conditional probability of a label sequence $\mathbf{y}$ given an observation sequence $\mathbf{x}$ (Lafferty, McCallum and Pereira, 2001). $\mathbf{x}$ denotes the sequence of feature vectors to be labelled, and $\mathbf{y}$ denotes the corresponding sequence of labels.

In the general graphical model formulation, the CRF score decomposes into contributions from vertices (state features) and edges (transition features); vertices capture the compatibility between a label and the observed features at a position, while edges capture the compatibility between neighbouring labels (Lafferty, McCallum and Pereira, 2001). In linear–chain CRFs, the graph structure is a chain over time steps $t = 1, \ldots, T$. The conditional probability is often written as local state feature functions f_k that depend on the current label y_t, the preceding label y_{t-1}, and the observation vector $\mathbf{x}_t$ (Sutton and McCallum, 2012):

$$p_\theta(\mathbf{y} \mid \mathbf{x}) \propto \prod_{t=1}^{T} \exp\Big\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\Big\}$$

This makes the factorisation into local state feature functions explicit, where each is scaled by a learned parameter θ_k (Sutton and McCallum, 2012). The algorithm learns the weights by estimating the conditional maximum likelihood by minimising, with respect to θ, the negated conditional log-likelihood of the predictions (Lavergne, Cappé and Yvon, 2010). Thus, the loss function used to train the model is:

$$\ell(\theta) = -\log p_\theta(\mathbf{y} \mid \mathbf{x}) = \log Z_\theta(\mathbf{x}) - \mathrm{score}_\theta(\mathbf{y}, \mathbf{x})$$

Where $\log Z_\theta(\mathbf{x})$ is the normalization factor and score_θ is the joint impact of $\mathbf{y}$ and $\mathbf{x}$ under parameters θ. The loss function is optimised iteratively using limited-memory BFGS, a quasi-Newton method that enables efficient training of large CRF models (Sha and Pereira, 2003). At inference time, a linear-chain CRF finds the most likely label sequence $\arg\max_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x})$ with Viterbi, a dynamic programming algorithm that combines state and transition feature scores (Sutton, McCallum, and others, 2012).

In [9]:
```
"""
Displaying a transformer layer figure.
"""
display(HTML('<img src="assets/opinion_miner_system_design_CRF.png" alt="Aspect extraction diag

FigureUtilities.figure_caption(
    "aspect_extraction_diagram",
```

```
    "System design diagram outlining the high level architecture of this CRF-based aspect extra
)
```

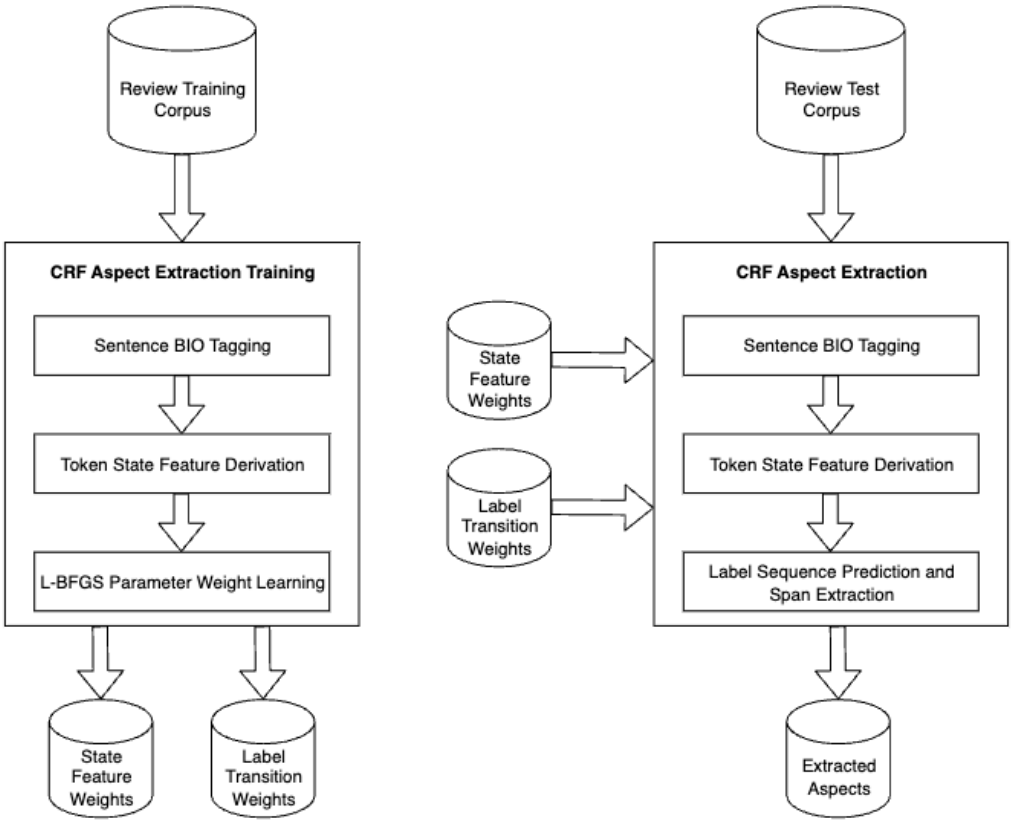

Figure 2. System design diagram outlining the high level architecture of this CRF-based aspect extraction subsystem.

Aspect Extraction Preprocessing

CRFs learn to predict labels in a sequence based on transition and feature probabilities. The input to this opinion mining pipeline is a series of largely unstructured text reviews. Therefore, they must be transformed into a state consumable by the CRF algorithm, including extracting ground truth aspects and per token feature function values.

Sentences are tagged using the BIO format (Ramshaw and Marcus, 1999), in which aspect spans are identified. Three tags are defined, B-ASP to signal the token is the start of an aspect, I-ASP to signal the continuation of an aspect, and O to signal any other token not related to an aspect. There are nine possible pairs of the three BIO labels, six of which are legal.

From	To	Legal
O	O	Yes
O	B-ASP	Yes
B-ASP	I-ASP	Yes
B-ASP	O	Yes
I-ASP	I-ASP	Yes
I-ASP	O	Yes
O	I-ASP	No
B-ASP	B-ASP	No
I-ASP	B-ASP	No

Ground Truth

CRF outputs ordered label sequences, but order is irrelevant here; aspect extraction requires identifying aspects within a sentence irrespective of position. To reduce spurious mismatches, extracted aspects are normalised for

case and whitespace (e.g., "use" in *"EASE OF USE/DESIGN: ... easy to use MP3 player"*). True negatives are excluded since non-aspects are undefined, and BIO tags are assigned by matching the longest spans in each sentence with the aspect present in the ground truth annotation for that sentence; the annotations are direct excerpts from the reviews.

The system aims to summarise sentiment per feature, but semantically similar features with different forms (e.g., "battery" vs. "batteries") can create unwanted sparsity. Collapsing by lemma reduces this, though it complicates integration with downstream sentiment analysis subsystems since the lemmatised form may not appear in the sentence, and lemmas are context dependent. Thus, the ground truth is not standardised by lemmatisation.

Original Sentence	Ground Truth Aspect	Example Lemmatised Ground Truth Aspect
battery[+2]##(batteries last longer too !)	Battery	Battery
batteries[-3]## This was obviously ... replaceable batteries!!	Batteries	Battery

Aspect Extraction Methodology

This subsystem uses CRFsuite (Okazaki, 2007), a library for linear-chain conditional random fields. It abstracts the complexity of CRF models by supporting automatic transition features learning, flexible state feature modeling, and configurable hyperparameters. A number of state feature functions are implemented from the literature, each attempting to provide the CRF model with token derived indicators to learn to identify aspects.

Context Window

A context window of length two is applied to token and part-of-speech (POS) tagging features, enabling the algorithm at token i to consider the token and POS tags at positions $i-2$, $i-1$, i, $i+1$ and $i+2$ (Sarkar, 2016). This expanded context allows the algorithm to model context-dependent local patterns to improve sequence labelling performance.

Token

The algorithm includes the current token as a feature. If the lexicon of aspects for a domain is compact, the training data is likely to contain many of the target aspects and so this feature can be a powerful feature (Jakob and Gurevych, 2010).

Part-of-Speech Tag

The POS tag provides a form of lexical disambiguation by assigning a tag based on local context (Jakob and Gurevych, 2010). For example, the token "can" is tagged as a verb in "I can swim" but as a noun in "a can of fish."

Opinion Sentence

This feature labels all tokens co-occurring in a sentence that also contains opinion words (Jakob and Gurevych, 2010). Aspects are likely to occur in the same sentence as opinion words, because natural relationships exist between them, as opinion words are used to modify aspects (Qiu et al., 2011).

Dependency Relations

A token's dependency information is represented as a state feature, including the head word and dependency relations with parent and child tokens, with the goal of allowing the model to use the dependency relationships between tokens to identify aspects (Wang et al., 2016). This can be effective because there exist natural relationships between, for example, opinion words and features.

Regularisation

Both Lasso (L_1) (Tibshirani, 1996) and Ridge (L_2) (Hoerl and Kennard, 1970) regularisation are employed to prevent the model from overfitting to the training data. L_1 regularisation seeks to move features with small

weights to zero and L_2 regularisation acts as a smoothing mechanism by discouraging the model from assigning very large weights to individual features.

The objective function for the CRF algorithm with regularisation is:

$$\ell(\theta) = -\log p_\theta(\mathbf{y} \mid \mathbf{x}) \; + \; c_1 \, \|\theta\| \; + \; \frac{c_2}{2} \, \|\theta\|^2$$

Where c_1 and c_2 are the regularisation coefficients and θ the parameter weights.

In [10]:
```
"""
Generates BIO tags and per token state features for training and test datasets.
"""
def is_opinion_token(token):
    return True if token.pos_ == Constants.ADJECTIVE_POS or token.text.lower() in adjective_opi

def get_opinion_token_ids(doc):
    ids = set()
    for i, token in enumerate(doc):
        if is_opinion_token(token):
            ids.add(i)
    return ids

def token_to_word(token):
    return token.lemma_.lower()

def token_to_pos(token):
    return token.tag_

def token_to_shape(token):
    t = token.text
    if t.isupper():
        return "all_upper"
    if t.islower():
        return "all_lower"
    if t.istitle():
        return "title_case"
    if t.isdigit():
        return "all_digit"
    return token.shape_

def word_context(doc, index, window_length=1):
    state_features = dict()

    state_features["WORD"] = token_to_word(doc[index])
    state_features["POS"] = token_to_pos(doc[index])
    state_features["SHAPE"] = token_to_shape(doc[index])

    for j in range(1, window_length + 1):
        if index >= j:
            state_features[f"-{j}WORD"] = token_to_word(doc[index - j])
            state_features[f"-{j}POS"] = token_to_pos(doc[index - j])
            state_features[f"-{j}SHAPE"] = token_to_shape(doc[index - j])

        if index + j < len(doc):
            state_features[f"+{j}WORD"] = token_to_word(doc[index + j])
            state_features[f"+{j}POS"] = token_to_pos(doc[index + j])
            state_features[f"+{j}SHAPE"] = token_to_shape(doc[index + j])

    return state_features

def opinion_sentence_context(doc):
    state_features = dict()
    state_features["IS_OPINION_SENTENCE"] = len(get_opinion_token_ids(doc)) > 0
    return state_features

def dependency_context(doc, index):
    state_features = dict()
    token = doc[index]

    state_features["REL_TO_HEAD"] = token.dep_

    head = token.head if token.head is not token else token
    state_features["HEAD_WORD"] = head.lemma_.lower()
    state_features["HEAD_POS"] = head.pos_
```

```
    child_deps = sorted({c.dep_ for c in token.children})
    child_poss = sorted({c.pos_ for c in token.children})
    state_features["CHILD_DEPS"] = ",".join(child_deps) if child_deps else "<none>"
    state_features["CHILD_POS"] = ",".join(child_poss) if child_poss else "<none>"

    return state_features

def doc_to_state_features(doc):
    x_features = []
    for i, t in enumerate(doc):
        state_features = {"bias": 1.0}

        word_context_state_features = word_context(doc, i, window_length=2)
        state_features.update(word_context_state_features)

        opinion_sentence_state_features = opinion_sentence_context(doc)
        state_features.update(opinion_sentence_state_features)

        dependency_state_features = dependency_context(doc, i)
        state_features.update(dependency_state_features)

        x_features.append(state_features)
    return x_features

def sentence_doc_and_tags(sent_dict):
    text = sent_dict["text"]
    doc = nlp(text)
    annotations = sent_dict.get("annotations", [])

    matcher = PhraseMatcher(nlp.vocab, attr="LOWER")
    features = []
    for i, a in enumerate(annotations):
        aspect = (a.get("feature") or "").strip()
        if not aspect:
            continue
        features.append(aspect)
        matcher.add(f"ASPECT_{i}", [nlp.make_doc(aspect)])

    matches = matcher(doc)
    spans = []
    for _, start, end in matches:
        spans.append((start, end))

    spans.sort(key=lambda span: (span[1] - span[0]), reverse=True)

    # Prefer longest spans
    taken = [False] * len(doc)
    bio = [Constants.OTHER] * len(doc)
    for (start, end) in spans:
        if any(taken[i] for i in range(start, end)):
            continue
        bio[start] = Constants.B_ASP
        for i in range(start + 1, end):
            bio[i] = Constants.I_ASP
        for i in range(start, end):
            taken[i] = True

    return doc, bio

def bio_spans_from_tags(tags: List[str]) -> List[Tuple[int, int]]:
    spans = []
    i = 0
    while i < len(tags):
        if tags[i] == Constants.B_ASP:
            start = i
            i += 1
            while i < len(tags) and tags[i] == Constants.I_ASP:
                i += 1
            spans.append((start, i))
        else:
            i += 1
    return spans

def spans_to_strings(doc, spans) -> List[str]:
```

```
    chunks = []
    for (s, e) in spans:
        chunk = doc[s:e].text.strip().lower()
        chunks.append(chunk)
    return chunks

def reviews_to_features_tags(reviews):
    x_features, y_tags, docs, crf_review_index = [], [], [], []
    for i, rev in enumerate(reviews):
        for sentence in rev["sentences"]:
            doc, tags = sentence_doc_and_tags(sentence)
            x_features.append(doc_to_state_features(doc))
            y_tags.append(tags)
            docs.append(doc)
            crf_review_index.append(i)
    return x_features, y_tags, docs, crf_review_index

x_features_train, y_tags_train, train_docs, _ = reviews_to_features_tags(train_reviews)
x_features_test, y_tags_test, test_docs, crf_review_index = reviews_to_features_tags(test_revie

print(f"Used {len(x_features_train)} training sentences")
print(f"Used {len(x_features_test)} test sentences")
```

```
Used 8509 training sentences
Used 1780 test sentences
```

Example State Feature Function Outputs

The below code outputs the generated state features for row 6 of the training data.

In [11]:
```
"""
Outputs example state features for visual inspection.
"""
tokens = x_features_train[5][0:100]
state_features_df = pd.DataFrame(tokens)
state_features_df.insert(0, "Token Index", range(len(tokens)))
display(state_features_df)
FigureUtilities.table_caption("crf_state_feature_examples_table", "An example of the state feat
```

	Token Index	bias	WORD	POS	SHAPE	+1WORD	+1POS	+1SHAPE	+2WORD	+2POS	...	HEAD_WOR
0	0	1.0	i	PRP	all_upper	have	VBP	all_lower	add	VBN	...	ac
1	1	1.0	have	VBP	all_lower	add	VBN	all_lower	the	DT	...	ac
2	2	1.0	add	VBN	all_lower	the	DT	all_lower	router	NN	...	ac
3	3	1.0	the	DT	all_lower	router	NN	all_lower	raizer	NNP	...	rout
4	4	1.0	router	NN	all_lower	raizer	NNP	all_lower	kit(simple	NNP	...	raiz
5	5	1.0	raizer	NNP	all_lower	kit(simple	NNP	all_lower	installation	NN	...	ac
6	6	1.0	kit(simple	NNP	all_lower	installation	NN	all_lower	)	-RRB-	...	installatic
7	7	1.0	installation	NN	all_lower	)	-RRB-	)	to	IN	...	raiz
8	8	1.0	)	-RRB-	)	to	IN	all_lower	it	PRP	...	raiz
9	9	1.0	to	IN	all_lower	it	PRP	all_lower	and	CC	...	ac
10	10	1.0	it	PRP	all_lower	and	CC	all_lower	can	MD	...	
11	11	1.0	and	CC	all_lower	can	MD	all_lower	not	RB	...	ac
12	12	1.0	can	MD	all_lower	not	RB	all_lower	live	VB	...	li
13	13	1.0	not	RB	all_lower	live	VB	all_lower	without	IN	...	li
14	14	1.0	live	VB	all_lower	without	IN	all_lower	this	DT	...	ac
15	15	1.0	without	IN	all_lower	this	DT	all_lower	setup	NN	...	li
16	16	1.0	this	DT	all_lower	setup	NN	all_lower	!	.	...	setu
17	17	1.0	setup	NN	all_lower	!	.	!	NaN	NaN	...	witho
18	18	1.0	!	.	!	NaN	NaN	NaN	NaN	NaN	...	ac

19 rows × 23 columns

Table 5. An example of the state features generated for one of the training dataset's sentences.

Hyperparameter Selection

The L_1 and L_2 regularisation coefficients, along with the maximum number of iterations, were tuned using 5-fold cross-validation. The configuration achieving the highest average F_1^{micro} score was selected. The `all_possible_transitions` option in CRFsuite is enabled. This setting introduces weights for every possible label bigram even if unobserved, allowing the model to assign negative weights to invalid transitions, improving model generalisation performance.

50 configurations were evaluated and the best performance (Figure 3), $F_1^{\text{micro}} = 37.3 \pm 0.0164\%$, was achieved with $c_1 = 0.5$, $c_2 = 0$ and 200 iterations (Table 6). The relatively large c_1 coefficient highlights the benefit of promoting sparsity in the high-dimensional CRF model, while the zero c_2 coefficient implies that L_2 regularisation harmed performance.

In [12]:
```
"""
Executes a hyperparameter grid search on the CRF model. Using 14 concurrent workers on an Apple
"""
def cross_validation_score_for_params(x_features, y_tags, params, k=5):
    """
    Cross validation evaluation for different hyperparameter grid search parameters.

    :param x_features: review sentences as per token state features
    :param y_tags: review sentences as BIO tags
    :param params: CRF configuration to evaluate
    :param k: number of folds to test
    :return: the cross validation score
    """
    splitter = KFold(n_splits=k, shuffle=True, random_state=8)
    fold_f1s = []
    for train_index, validation_index in splitter.split(x_features, y_tags):
        x_features_train = [x_features[i] for i in train_index]
        y_tags_train = [y_tags[i] for i in train_index]
```

```
        x_features_val = [x_features[i] for i in validation_index]
        y_tags_val = [y_tags[i] for i in validation_index]

        crf = CRF(
            algorithm=params["algorithm"],
            c1=params["c1"],
            c2=params["c2"],
            max_iterations=params["max_iterations"],
            all_possible_transitions=True
        )
        crf.fit(x_features_train, y_tags_train)
        y_pred = crf.predict(x_features_val)
        f1 = seq_f1(y_tags_val, y_pred, suffix=False, average="micro")
        fold_f1s.append(f1)

    mean_f1 = np.mean(fold_f1s)
    std_f1 = np.std(fold_f1s)
    print(f"Params: {params} -> F1 {float(mean_f1):.4f} ± {float(std_f1):.4f}")
    return float(np.mean(fold_f1s)), float(np.std(fold_f1s))


def tune_crf_hyperparams_cv_parallel(x_features, y_tags, n_jobs=-1):
    param_grid = {
        "algorithm": ["lbfgs"],
        "c1": [0.0, 0.01, 0.1, 0.5, 1],
        "c2": [0.0, 0.01, 0.1, 0.5, 1],
        "max_iterations": [200, 500]
    }
    params_list = list(ParameterGrid(param_grid))

    scores = Parallel(n_jobs=n_jobs, backend="loky", verbose=10)(delayed(cross_validation_score

    means = [mean for (mean, std) in scores]
    best_i = int(np.argmax(means))
    best_params = params_list[best_i]
    best_mean, best_std = scores[best_i]

    best_model = CRF(
        algorithm=best_params["algorithm"],
        c1=best_params["c1"],
        c2=best_params["c2"],
        max_iterations=best_params["max_iterations"],
        all_possible_transitions=True
    ).fit(x_features, y_tags)

    print(f"Best params: {best_params} -> F1 {best_mean:.4f} ± {best_std:.4f}")
    return best_model, best_params, list(zip(params_list, scores))

def plot_cross_validation_results(cross_validation_results_table: List[Tuple[Dict[str, Any], Tu
    rows = []
    for params, (mean, std) in cross_validation_results_table:
        rows.append({
            "c1": params["c1"],
            "c2": params["c2"],
            "max_iterations": params["max_iterations"],
            "mean_f1": mean,
            "std_f1": std,
        })
    df = pd.DataFrame(rows)

    df["config_id"] = range(len(df))

    fig, ax = plt.subplots(figsize=(12, 6))
    sns.scatterplot(
        data=df,
        x="config_id",
        y="mean_f1",
        s=120,
        ax=ax
    )

    ax.errorbar(
        df["config_id"], df["mean_f1"],
        yerr=df["std_f1"],
        fmt="none", ecolor="gray", alpha=0.7, capsize=3
```

```
    )

    ax.set_xticks(df["config_id"])
    ax.set_xlabel("Hyperparameter Configuration Index")
    ax.set_ylabel("Cross Validation $F_1^{micro}$ (mean $\pm$ std)")
    ax.set_title("CRF Hyperparameter Cross Validation Results")
    plt.tight_layout()
    plt.show()
    FigureUtilities.figure_caption("crf_cv_table", "Hyperparameter tuning results, with combina

    top10 = (df
         .sort_values("mean_f1", ascending=False)
         .head(10)
         .drop(columns=["config_id"])
         .reset_index(drop=True))
    display(top10.style.format({"mean_f1": "{:.4f}"}))
    FigureUtilities.table_caption("crf_cv_table", "Showing the hyperparameter configurations fo

best_crf, best_params, cv_table = tune_crf_hyperparams_cv_parallel(x_features_train, y_tags_tra
plot_cross_validation_results(cv_table)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 14 concurrent workers.
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  13 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done  22 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  29 out of  50 | elapsed:  4.9min remaining:  3.6min
[Parallel(n_jobs=-1)]: Done  35 out of  50 | elapsed:  5.9min remaining:  2.5min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed:  6.8min remaining:  1.5min
Params: {'algorithm': 'lbfgs', 'c1': 0.01, 'c2': 0.01, 'max_iterations': 200} -> F1 0.3506 ± 0.0
143
Params: {'algorithm': 'lbfgs', 'c1': 0.01, 'c2': 1, 'max_iterations': 200} -> F1 0.3377 ± 0.0195
Params: {'algorithm': 'lbfgs', 'c1': 0.1, 'c2': 0.5, 'max_iterations': 200} -> F1 0.3511 ± 0.010
4
Params: {'algorithm': 'lbfgs', 'c1': 0.5, 'c2': 0.1, 'max_iterations': 200} -> F1 0.3636 ± 0.012
9
Params: {'algorithm': 'lbfgs', 'c1': 1, 'c2': 0.0, 'max_iterations': 200} -> F1 0.3528 ± 0.0146
Params: {'algorithm': 'lbfgs', 'c1': 1, 'c2': 1, 'max_iterations': 200} -> F1 0.3105 ± 0.0127
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed:  7.9min remaining:   30.1s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  9.0min finished
Params: {'algorithm': 'lbfgs', 'c1': 0.0, 'c2': 0.0, 'max_iterations': 200} -> F1 0.3594 ± 0.011
0
Params: {'algorithm': 'lbfgs', 'c1': 0.01, 'c2': 0.1, 'max_iterations': 200} -> F1 0.3700 ± 0.01
24
Params: {'algorithm': 'lbfgs', 'c1': 0.1, 'c2': 0.01, 'max_iterations': 500} -> F1 0.3585 ± 0.01
68
Params: {'algorithm': 'lbfgs', 'c1': 1, 'c2': 0.01, 'max_iterations': 200} -> F1 0.3530 ± 0.0144
Params: {'algorithm': 'lbfgs', 'c1': 1, 'c2': 1, 'max_iterations': 500} -> F1 0.3079 ± 0.0114
Best params: {'algorithm': 'lbfgs', 'c1': 0.5, 'c2': 0.0, 'max_iterations': 200} -> F1 0.3730 ±
0.0164
```

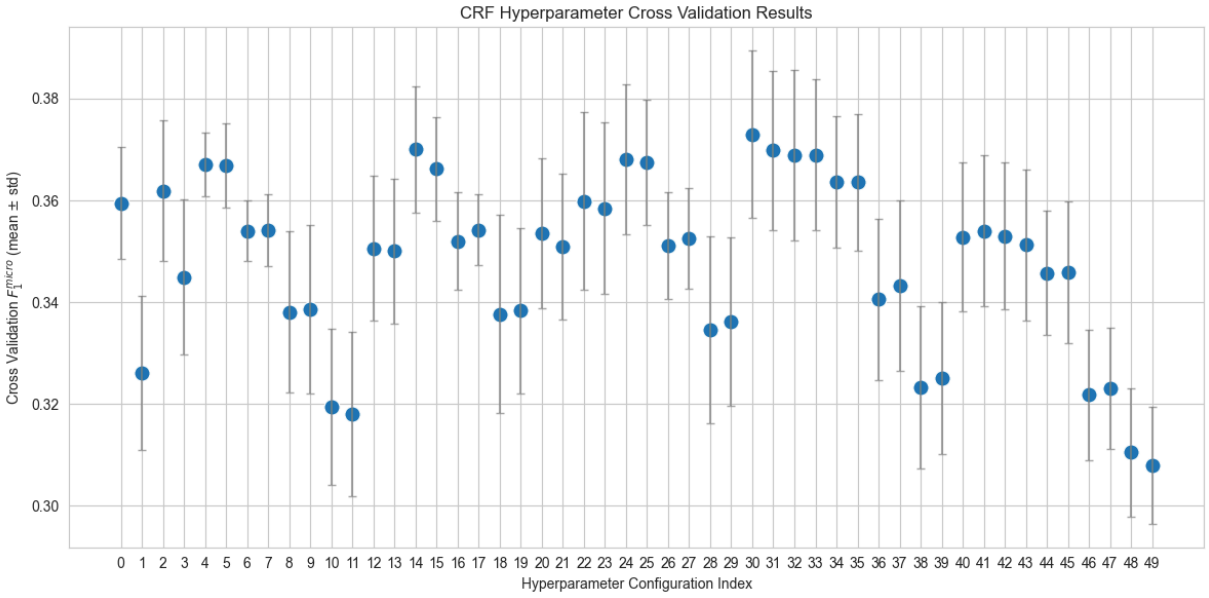

Figure 3. Hyperparameter tuning results, with combinations including varying L1 and L2 coefficients, along with two options for max iterations.

	c1	c2	max_iterations	mean_f1	std_f1
0	0.500000	0.000000	200	0.3730	0.016383
1	0.010000	0.100000	200	0.3700	0.012387
2	0.500000	0.000000	500	0.3698	0.015585
3	0.500000	0.010000	500	0.3690	0.014910
4	0.500000	0.010000	200	0.3689	0.016712
5	0.100000	0.100000	200	0.3681	0.014766
6	0.100000	0.100000	500	0.3675	0.012312
7	0.000000	0.100000	200	0.3671	0.006290
8	0.000000	0.100000	500	0.3669	0.008315
9	0.010000	0.100000	500	0.3662	0.010188

Table 6. Showing the hyperparameter configurations for the top 10 mean F1 micro scores.

CRF Transition Weights

Table 7 shows the transition features and their learned weights. Illegal sequences like $\text{I-ASP} \rightarrow \text{B-ASP}$ have large negative weights, meaning the model is unlikely to predict an illegal output sequence. However, legal transitions like $\text{B-ASP} \rightarrow \text{O}$ and $\text{I-ASP} \rightarrow \text{O}$ also have negative weights, meaning the model is less likely to predict an aspect followed by a non-aspect word.

In [13]:
```
"""
Displays CRF transition weights.
"""
def display_transition_weights():
    rows = []
    for (prev_tag, curr_tag), w in sorted(best_crf.transition_features_.items(), key=lambda tra
        rows.append({"from": prev_tag, "to": curr_tag, "weight": w})

    df = pd.DataFrame(rows)
    df["weight"] = df["weight"].round(3)
    display(df)
    FigureUtilities.table_caption("crf_transition_weights_table", "Showing alls possible BIO tr

display_transition_weights()
```

	from	to	weight
0	B-ASP	I-ASP	2.075
1	I-ASP	I-ASP	2.038
2	O	O	1.260
3	O	B-ASP	0.463
4	B-ASP	O	-0.978
5	I-ASP	O	-1.138
6	I-ASP	B-ASP	-3.144
7	B-ASP	B-ASP	-3.487
8	O	I-ASP	-9.865

Table 7. Showing alls possible BIO transitions and their learned weights.

CRF Feature Weights

Table 8 shows the five highest and lowest feature weights for each BIO tag. `WORD` token features, highlighting their strong influence on aspect identification compared to the other features. In contrast, contextual and dependency-relation features rarely appear among the top five for any label or sentiment direction.

In [14]:
```
"""
Displays the top 5 positive and negative weights for each BIO tag.
"""
```

```
def top_state_features(crf, required_label, k):
    feats = {attr: feature_label for (attr, label), feature_label in crf.state_features_.items(
    positive_weight_features = sorted(feats.items(), key=lambda x: x[1], reverse=True)[:k]
    negative_weight_features = sorted(feats.items(), key=lambda x: x[1])[:k]
    return positive_weight_features, negative_weight_features

def display_top_state_features():
    rows = []
    for label in [Constants.B_ASP, Constants.I_ASP, Constants.OTHER]:
        positive_weight_features, negative_weight_features = top_state_features(best_crf, label

        for attr, weight in positive_weight_features:
            rows.append({
                "Label": label,
                "Orientation": "Positive",
                "Feature": attr,
                "Weight": weight
            })

        for attr, weight in negative_weight_features:
            rows.append({
                "Label": label,
                "Orientation": "Negative",
                "Feature": attr,
                "Weight": weight
            })

    df_feats = pd.DataFrame(rows)
    display(df_feats)
    FigureUtilities.table_caption("crf_features_weights_table", "Showing the top 5 positive and

display_top_state_features()
```

	Label	Orientation	Feature	Weight
0	B-ASP	Positive	WORD:pipeline	6.464624
1	B-ASP	Positive	WORD:horn	6.116255
2	B-ASP	Positive	WORD:snapshot	6.031936
3	B-ASP	Positive	-1WORD:speed	5.653595
4	B-ASP	Positive	WORD:nimble	5.604997
5	B-ASP	Negative	REL_TO_HEAD:punct	-2.931322
6	B-ASP	Negative	-2POS:-LRB-	-2.829778
7	B-ASP	Negative	HEAD_WORD:through	-2.477579
8	B-ASP	Negative	WORD:thing	-2.071869
9	B-ASP	Negative	HEAD_WORD:include	-2.037183
10	I-ASP	Positive	WORD:disc	6.006086
11	I-ASP	Positive	HEAD_WORD:standout	4.962186
12	I-ASP	Positive	-1WORD:parental	4.799999
13	I-ASP	Positive	-1WORD:technical	4.729931
14	I-ASP	Positive	-2WORD:horrible	4.301058
15	I-ASP	Negative	+2WORD:that	-2.420600
16	I-ASP	Negative	+2WORD:decent	-2.342180
17	I-ASP	Negative	POS:VBZ	-2.205845
18	I-ASP	Negative	SHAPE:all_upper	-1.718598
19	I-ASP	Negative	-1WORD:menu	-1.678053
20	O	Positive	POS:PRP	6.018607
21	O	Positive	WORD:be	4.563536
22	O	Positive	POS:.	4.447223
23	O	Positive	REL_TO_HEAD:aux	4.066165
24	O	Positive	POS:DT	3.638900
25	O	Negative	WORD:keyboard	-5.272109
26	O	Negative	WORD:widescreen	-5.038053
27	O	Negative	SHAPE:Xdddx	-4.546763
28	O	Negative	WORD:dial	-3.925478
29	O	Negative	WORD:s100	-3.702065

Table 8. Showing the top 5 positive and negative state features and their weights for each BIO tag.

Aspect Extraction Evaluation

The CRF aspect extraction subsystem achieved $\text{Precision}^{\text{micro}} = 52.1\%$, $\text{Recall}^{\text{micro}} = 33.8\%$, and $F_1^{\text{micro}} = 41.0\%$ on the held-out test set. A confusion matrix is plotted to visualise performance (Figure 4).

Consider “Warning: These speakers are marketed as speakers for your TV, PlayStation, or DVD player — not your computer.” The system predicted “dvd player” and “speaker” where the ground truth is empty. “Speaker” arguably belongs in the ground truth, but “dvd player” exposes limits of hand-crafted state features and possibly the model class; knowing not to extract “dvd player” requires significant language understanding, which perhaps only a pre-trained transformer is equipped to solve.

There are also cases where predictions look valid to a human but are missing in the ground truth. For example, in Table 10 row 19, “Not a very impressive signal, can only get one maybe two bars and maybe 48mbs.” The system extracts “signal” but the ground truth is empty, raising questions about dataset suitability for real-world

evaluation. The literature reports strong CRF performance on related tasks (Shu, Xu and Liu, 2017), so the gap here likely reflects differences in datasets and feature design.

CRFs probabilistic nature can assign negative weights to legal transitions, subtly discouraging correct boundaries. Here, two legal transitions carry slightly negative weights, which can lead to over-spanning aspects. Clearer boundary features (e.g., noun and noun-phrase cues) would help. Feature engineering remains manual and domain-dependent, so important cues may be missing or suboptimal.

From	To	Weight
B-ASP	O	−0.978
I-ASP	O	−1.138

Finally, the subsystem extracts many near-duplicates that should be collapsed for user-facing summaries. For the “Computer” product type, variants of “monitor” appear separately. Post-hoc clustering of extracted aspects and labeling with a representative word would reduce unwanted sparsity without changing the CRF model.

Computer Aspect Examples
acer gd235hz monitor
acer lcd monitor
acer monitor

In [15]:
```
"""
Defining count based precision, recall and F1 score functions and evaluating the CRF model agai

Evaluating aspect extraction against the held-out test set.
"""
def get_precision(tp: int, fp: int) -> float:
    if tp + fp > 0:
        return tp / (tp + fp)
    return 0

def get_recall(tp: int, fn: int) -> float:
    if tp + fn > 0:
        return tp / (tp + fn)
    return 0

def get_f1(precision_param: float, recall_param: float) -> float:
    if precision_param + recall_param > 0:
        return 2 * precision_param * recall_param / (precision_param + recall_param)
    return 0

assert get_precision(3, 1) == 0.75
assert get_recall(3, 3) == 0.5
assert get_f1(get_precision(3, 1), get_recall(3, 3)) == 0.6

def normalize_aspect_label(aspect: str) -> str:
    """
    Normalise an aspect by converting to lower case and removing edge whitespace.

    :param aspect: the aspect to normalise
    :return: the normalised aspcet
    """
    return aspect.lower().strip()

def evaluate_aspect_extraction(y_true_aspects, y_pred_aspects, docs):
    """
    Evaluate aspect extraction performance using Precision, Recall and F1 scores.

    :param y_true_aspects: ground truth aspect values
    :param y_pred_aspects: predicted aspect values
    :param docs: spaCy processed sentences
    :return: precision, recall, F1 and mismatch examples
    """
    tp = fp = fn = 0
    test_reviews_predicted_aspects = []
    for tags_true, tags_pred, doc in zip(y_true_aspects, y_pred_aspects, docs):
        ground_truth_spans = bio_spans_from_tags(tags_true)
        pred_spans = bio_spans_from_tags(tags_pred)
```

```
        ground_truth_texts = {normalize_aspect_label(doc[s:e].text) for (s, e) in ground_truth_
        pred_texts = {normalize_aspect_label(doc[s:e].text) for (s, e) in pred_spans}

        tp += len(ground_truth_texts.intersection(pred_texts))
        fp += len(pred_texts.difference(ground_truth_texts))
        fn += len(ground_truth_texts.difference(pred_texts))

        test_reviews_predicted_aspects.append({
            "text": doc.text,
            "ground_truth": sorted(ground_truth_texts),
            "pred": sorted(pred_texts),
            "missed": sorted(ground_truth_texts - pred_texts),
            "spurious": sorted(pred_texts - ground_truth_texts),
        })

    precision = get_precision(tp, fp)
    recall = get_recall(tp, fn)
    return precision, recall, get_f1(precision, recall), test_reviews_predicted_aspects, tp, fp

def display_crf_results():
    metrics = {
        "Precision (micro)": round(precision * 100, 2),
        "Recall (micro)": round(recall * 100, 2),
        "F1 (micro)": round(f1 * 100, 2)
    }

    df = pd.DataFrame([metrics]).T
    df.columns = ["Value (%)"]

    display(df)
    FigureUtilities.table_caption("aspect_extraction_metrics", "Position agnostic aspect extrac

def plot_confusion_matrix(tp, fp, fn):
    cm = np.array([
        [tp, fn],
        [fp, 0]
    ])

    confusion_matrix_labels = ("Positive", "Negative")
    fig, ax = plt.subplots(figsize=(5, 4))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=confusion_matrix_labels,
        yticklabels=confusion_matrix_labels,
        cbar=False,
        ax=ax)

    ax.set_xlabel("Predicted")
    ax.set_ylabel("Ground Truth")
    ax.set_title("Confusion Matrix (CRF Aspect Extraction)")
    plt.show()
    FigureUtilities.figure_caption("crf_confusion_matrix", "Confusion matrix showing results fo

test_reviews_predicted_aspect_tags = best_crf.predict(x_features_test)
precision, recall, f1, test_reviews_predicted_aspects, tp, fp, fn = evaluate_aspect_extraction(

display_crf_results()
plot_confusion_matrix(tp, fp, fn)
```

	Value (%)
Precision (micro)	52.14
Recall (micro)	33.77
F1 (micro)	40.99

Table 9. Position agnostic aspect extraction metrics.

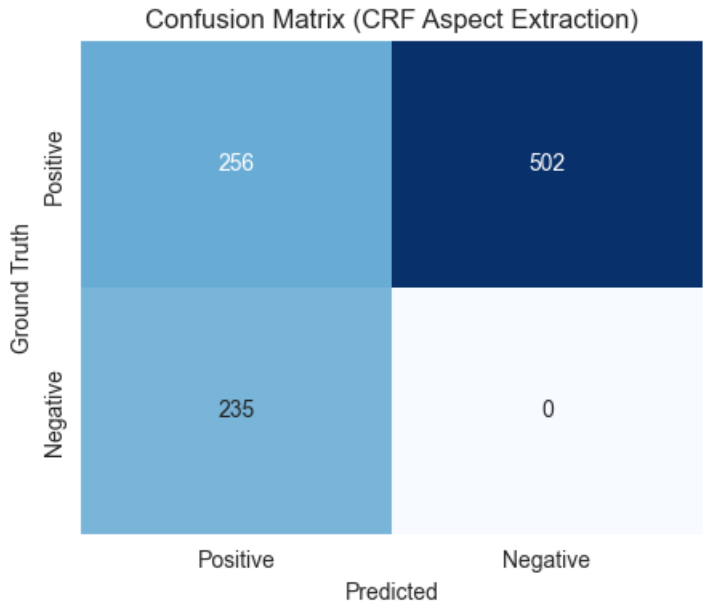

Figure 4. Confusion matrix showing results for CRF aspect extraction tested on the held-out test dataset.

Table 10 displays several examples of ground truth aspects as compared to predicted.

In [16]:
```
"""
Displaying examples of matches and mismatches output by the CRF aspect extraction test.
"""
examples = []
limits = {"Match Examples": 10, "False Negative Examples": 10, "False Positive Examples": 10}
counts = {k: 0 for k in limits}

for tags_true, tags_pred, doc in zip(y_tags_test, test_reviews_predicted_aspect_tags, test_docs
    ground_truth_spans = bio_spans_from_tags(tags_true)
    pred_spans = bio_spans_from_tags(tags_pred)

    ground_truth_texts = {normalize_aspect_label(doc[s:e].text) for (s, e) in ground_truth_span
    pred_texts = {normalize_aspect_label(doc[s:e].text) for (s, e) in pred_spans}

    if not ground_truth_texts and not pred_texts:
        continue

    matched = sorted(ground_truth_texts & pred_texts)
    missed = sorted(ground_truth_texts - pred_texts)
    spurious = sorted(pred_texts - ground_truth_texts)

    category = None
    if counts["Match Examples"] < limits["Match Examples"] and len(matched) > 0:
        category = "Match Examples"
    elif counts["False Negative Examples"] < limits["False Negative Examples"] and len(ground_t
        category = "False Negative Examples"
    elif counts["False Positive Examples"] < limits["False Positive Examples"] and len(pred_tex
        category = "False Positive Examples"

    if category:
        examples.append({
            "Category": category,
            "Sentence": doc.text,
            "Ground Truth": sorted(ground_truth_texts),
            "Predicted": sorted(pred_texts),
            "Matched": matched,
            "Missed": missed,
            "Spurious": spurious
        })
        counts[category] += 1

    if all(counts[k] >= limits[k] for k in limits):
        break

df_examples = pd.DataFrame(examples)
```

```
display(df_examples)
FigureUtilities.table_caption("aspect_extraction_examples_table", "Showing several examples of
```

	Category	Sentence	Ground Truth	Predicted	Matched	Missed	Spurious
0	Match Examples	Great Sound .	[sound]	[sound]	[sound]	[]	[]
1	False Negative Examples	EASE OF USE/DESIGN: The iPod is a very easy to use MP3 player, and if you can't figure this out, you shouldn't even own one.	[use]	[]	[]	[use]	[]
2	False Negative Examples	After getting one for Christmas from Amazon, I'm now a believer: The Digital Elph is truly the world's smallest digital camera that shoots great pictures.	[pictures]	[]	[]	[pictures]	[]
3	False Negative Examples	This is not an enterprise piece of hardware that you will find in an office or corporate environment.	[enterprise piece]	[]	[]	[enterprise piece]	[]
4	False Positive Examples	You will not be able to run 100's users using WPA and WPA2 encryption through this thing, just a handful of people on a cable modem or DSL line.	[]	[dsl line]	[]	[]	[dsl line]
5	False Negative Examples	The new hardware version does not run Linux at all and most likely will not since the memory has been cut in half.	[hardware version]	[]	[]	[hardware version]	[]
6	False Negative Examples	just want a wireless router for home or your very small business, you can just get a WRT54G from here and configure it.	[wrt54g]	[]	[]	[wrt54g]	[]
7	False Positive Examples	All wireless routers need to be configured to use Wireless Encryption, so this is just not a problem with the WRT54G.	[]	[wireless encryption]	[]	[]	[wireless encryption]
8	False Negative Examples	Another thing you should know, the metal on the flip side of the iPod is very tacky, and scratches very easily.	[flip side]	[]	[]	[flip side]	[]
9	False Negative Examples	The main problems for me is that the dsub cord from the sub to the control station is too short .	[dsub cord]	[]	[]	[dsub cord]	[]
10	Match Examples	The speaker bar response was spot-on .	[speaker bar]	[speaker bar]	[speaker bar]	[]	[]
11	False Negative Examples	After reading reviews of LCD problems, cracked, black blobs etc, I figured that it was probably a very limited problem.	[lcd]	[]	[]	[lcd]	[]
12	False Negative Examples	its definately an outstanding laptop .	[laptop]	[]	[]	[laptop]	[]
13	False Negative Examples	I have about 6 gb used on my iPod so far (well over 1000 songs), and as far as I can tell, all of them play at near-cd quality.	[quality]	[]	[]	[quality]	[]
14	False Positive Examples	This little item blew my sock off when I heard the sound of it .	[]	[item]	[]	[]	[item]
15	Match Examples	the software failed repeatedly .	[software]	[software]	[software]	[]	[]

	Category	Sentence	Ground Truth	Predicted	Matched	Missed	Spurious
16	False Positive Examples	i could n't load more than one song before the software crashed .	[]	[software]	[]	[]	[software]
17	False Positive Examples	Warning : These speakers are marketed as speakers for your TV , PlayStation , or DVD player -- not your computer .	[]	[dvd player, speakers]	[]	[]	[dvd player, speakers]
18	False Positive Examples	The AOC 2436 Monitor worked very well at first , then seemed to have problems .	[]	[aoc 2436 monitor]	[]	[]	[aoc 2436 monitor]
19	False Positive Examples	Not a very impressive signal , can only get one maybe two bars and maybe 48mbs .	[]	[signal]	[]	[]	[signal]
20	False Positive Examples	I have no issues w/ odor but baby poop doesn't start to smell really bad till they start on soilid so odor will not be a problem no matter what till then and that's many many months down the road.	[]	[odor, smell]	[]	[]	[odor, smell]
21	Match Examples	I am using JBL N38 II as surround left & right , which is again a very nice excellent speaker set .	[speaker set]	[speaker set]	[speaker set]	[]	[]
22	Match Examples	Really good sound quality , but certainly not high fidelity , and volume is not at all blaring .	[sound quality, volume]	[sound quality]	[sound quality]	[volume]	[]
23	Match Examples	great camera , i have been using this for several months and got excellent results , simple friendly usage , in many scenes indoor , outdoor , snow , close up macro etc.	[camera]	[camera]	[camera]	[]	[]
24	Match Examples	These are the only computer speakers I 've ever bought so I have no comparison to anything else .	[computer speakers]	[computer speakers]	[computer speakers]	[]	[]
25	False Positive Examples	This would be great in a dorm room or bedroom setting where you may not already have a stereo receiver and want better sound than what your TV is giving you .	[]	[sound]	[]	[]	[sound]
26	Match Examples	The replacement unit had a problem too : the little switch on the bottom was n't working properly , making it impossible to select among city forecasts in my area .	[switch]	[switch]	[switch]	[]	[]
27	Match Examples	First, I have to say that I have NEVER had the slightest problem with this camera or the software.	[camera, software]	[software]	[software]	[camera]	[]
28	Match Examples	The camera is solid, performs well, takes good pictures, and the battery lasts pretty long if you disable the LCD viewfinder.	[battery, camera, performs]	[battery, camera]	[battery, camera]	[performs]	[]
29	False Positive Examples	the keys are laid out normal so there is no guessing .	[]	[keys]	[]	[]	[keys]

Table 10. Showing several examples of matches, ground truth having more aspects than predicted and predicted having more aspects than ground truth. Anecdotally, the system often extracts an aspect that makes sense yet is not found in the ground truth.

Sentiment Analysis

Sentiment analysis in an opinion miner assigns a sentiment score to each aspect mentioned in a review. Here, sentiment is binary, -1 for negative and $+1$ for positive, determined by the opinion words used in relation to each aspect.

Two subsystem variants are implemented. The first follows the rules-based approach of Hu and Liu (2004). The second uses a BERT model (Devlin et al., 2019) with a fine-tuned classification layer.

I hypothesise that the BERT variant will outperform the rules-based system, with the gap widening as reviews become more varied and complex. BERT contrasts with a purpose-built rules algorithm by leveraging pretraining on large corpora, a high parameter count and fine-tuning, which should improve handling of cross-domain opinion words, non-adjectival cues and phrases, colloquial language, and complex opinion expressions such as sarcasm.

Both variants are evaluated on the `test_reviews` dataset given ground-truth aspects. Metrics are $\text{Precision}^{\text{macro}}$, $\text{Recall}^{\text{macro}}$, F_1^{macro}, and Coverage, enabling a fair comparison that gives equal weight to positive and negative classes. This assesses sentiment analysis independently of aspect extraction, and a holistic evaluation will later feed extracted aspects into these sentiment subsystems to simulate whole system production performance.

Rules Based Sentiment Analysis

The rules-based sentiment analysis subsystem follows Hu and Liu (2004) and has two main components, the `OrientationPrediction` and `SentenceOrientation` algorithms, described in detail below.

This subsystem variant calculates a single sentiment per sentence and cannot separate multiple aspect sentiments within the same sentence. For example, "look[+1], speed[-3]## Looks good, but the speed is terrible" results in both aspects being marked negative. In practice this limitation is likely of little consequence because sentences contain on average 1.25 annotated opinions (Figure 6).

In [17]:
```
"""
Displaying a transformer layer figure.
"""
display(HTML('<img src="assets/opinion_miner_system_design_rules-based.png" alt="Rules-based se

FigureUtilities.figure_caption(
    "opinion_miner_system_design_rules-based_diagram",
    "System design diagram outlining the high level architecture of this rules-based sentiment
)
```

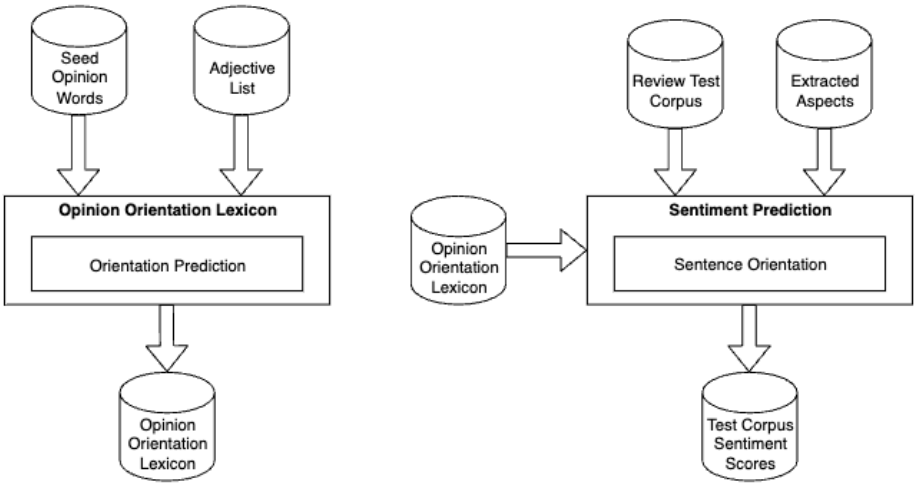

Figure 5. System design diagram outlining the high level architecture of this rules-based sentiment analysis subsystem.

In [18]:
```
"""
Showing counts of nouns, noun phrases and adjectives and average annotations per sentence.
"""
```

```
def display_pos_counts():
    all_sentences = ReviewParser.get_sentences(all_reviews)
    docs = list(nlp.pipe(all_sentences, disable=["ner", "lemmatizer"]))

    noun_set, adj_set, np_set = set(), set(), set()
    for doc in docs:
        for tok in doc:
            if tok.pos_ == "NOUN":
                noun_set.add(tok.text.lower())
            elif tok.pos_ == "ADJ":
                adj_set.add(tok.text.lower())
        for chunk in doc.noun_chunks:
            np_set.add(chunk.text.lower())

    # Display part of speech counts in a table
    data = {
        "Type": ["Nouns", "Adjectives", "Noun Phrases"],
        "Count": [len(noun_set), len(adj_set), len(np_set)]
    }

    df = pd.DataFrame(data)
    display(df)
    FigureUtilities.table_caption("rules_sentiment_pos_counts_table", "The counts of unique nou

def plot_annotations_per_sentence_counts():
    annotation_counts = [
        len(sentence["annotations"])
        for review in all_reviews
        for sentence in review["sentences"]
        if len(sentence["annotations"]) > 0
    ]

    mean_count = np.mean(annotation_counts)

    sns.set_theme(style="whitegrid")
    plt.figure(figsize=(5, 4))

    # histogram in seaborn style
    sns.histplot(
        annotation_counts,
        bins=range(1, max(annotation_counts, default=1) + 1),
        kde=False,
        edgecolor="black",
        discrete=True,
        color="C0",
        alpha=1
    )

    plt.axvline(
        mean_count,
        color='red',
        linestyle='--',
        linewidth=2,
        label=f'Mean = {mean_count:.2f}'
    )

    plt.xticks(range(1, max(annotation_counts, default=1) + 1))
    plt.xlabel('Annotations per Sentence')
    plt.ylabel('Number of Sentences')
    plt.title('Whole Dataset Annotations per Sentence Counts')
    plt.legend()
    plt.tight_layout()
    plt.show()
    FigureUtilities.figure_caption("opinion_word_count", "Showing the distribution of annotatio

display_pos_counts()
plot_annotations_per_sentence_counts()
```

	Type	Count
0	Nouns	4192
1	Adjectives	1648
2	Noun Phrases	16514

Table 11. The counts of unique nouns, noun phrases and adjectives in the corpus.

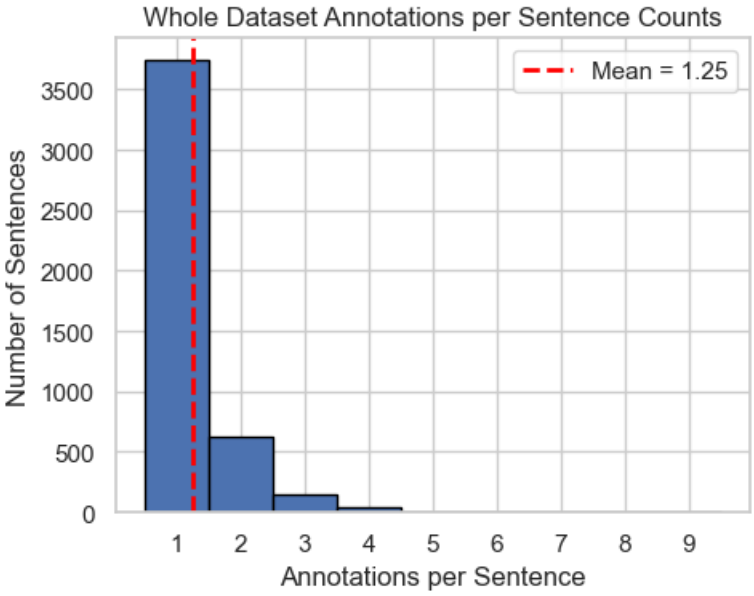

Figure 6. Showing the distribution of annotations per sentence across the whole dataset.

Rules Based Sentiment Analysis Preprocessing

Preprocessing for this subsystem variant involves constructing a lexicon of opinion words and their associated sentiments using the `OrientationPrediction` algorithm (Hu and Bing, 2004). The algorithm identifies sentiment orientation by iteratively searching WordNet (Princeton, 2010) for the synonyms and antonyms of a pre-supplied seed list, for which the sentiment direction is known. This works because adjectives generally possess the same opinion orientation as their synonyms and the opposite direction as their antonyms (Hu and Bing, 2004). With roughly $1,600$ unique adjectives across the dataset (Table 11), this yields a broad polarity-aligned lexicon.

The previously fetched tidytext adjective opinion word vocabulary, `adjective_opinion_words`, will be used as the seed lexicon for this algorithm. The lemma of the adjectives found in `train_reviews` provides the adjective list from which to propagate the sentiment polarity. Tables 12-16 provide details on the output of this algorithm, including class balance for the calculated opinion orientation lexicon, of which 41.9% of the opinion words have positive orientation and 58.1% negative. This imbalance could manifest as a bias towards negative sentiment.

Ground Truth

The ground truth for each aspect sentiment score per sentence is the sum of the annotated sentiment scores, normalised to 1 or -1. This enables a meaningful comparison between the datasets provided and the sentence-based sentiment scoring of this rules-based algorithm. Aspects used to drive the effective opinions component of this algorithm will be as extracted from the ground truth annotations for independent subsystem evaluation.

Original Sentence	Ground Truth Aspect	Ground Truth Sentiment	Standardised Ground Truth Sentiment
battery[+2]##(batteries last longer too !)	Battery	+2	+1
batteries[-3]## This was obviously ... replaceable batteries!!	Batteries	−3	−1
look[+1], speed[−3]## Looks good, but the speed is terrible	Look, Speed	+1, −3	−1, −1

The algorithm predicts sentiment for every sentence, regardless of whether an aspect is present. In a production setting, sentiment would only be attached to sentences with extracted aspects. To reflect this, the metrics do not consider sentences without annotated aspects.

In [19]:
```
"""
The OrientationPrediction algorithm to determine the sentiment polarity of the opinion word lex
"""
```

```
POSITIVE = "positive"
NEGATIVE = "negative"

def normalise(word: str) -> str:
    return word.strip().lower().replace(" ", "_")

def invert_orientation(o: str) -> str:
    o = (o or "").lower()
    if o.startswith("pos"):
        return NEGATIVE
    return POSITIVE

def explore_synset_frontier(word, max_hops=3):
    # Ensure deterministic propagation order and thus algorithm behaviour by sorting a key stag
    frontier = []
    for pos in (wn.ADJ, wn.ADJ_SAT):
        frontier.extend(wn.synsets(word, pos=pos))
    frontier = sorted(set(frontier), key=lambda s: s.name())

    visited = set(frontier)
    cluster = list(frontier)

    for _ in range(max_hops):
        next_frontier = []
        for s in frontier:
            for candidate in sorted(s.similar_tos(), key=lambda c: c.name()):
                if candidate not in visited:
                    visited.add(candidate)
                    next_frontier.append(candidate)
        if not next_frontier:
            break
        next_frontier = sorted(set(next_frontier), key=lambda s: s.name())
        cluster.extend(next_frontier)
        frontier = next_frontier

    return cluster

def adjective_synsets(word):
    word = normalise(word)
    cluster = explore_synset_frontier(word)
    syns, ants = set(), set()

    for s in cluster:
        for lemma in s.lemmas():
            syns.add(normalise(lemma.name()))
            for ant in lemma.antonyms():
                ants.add(normalise(ant.name()))
    return syns, ants

def orientation_search(adjective_list, seed_map):
    newly_assigned = []
    not_in_wordnet = []
    seed_keys = set(seed_map.keys())

    for word in sorted(adjective_list):
        key = normalise(word)
        if key in seed_map:
            continue

        syns, ants = adjective_synsets(key)

        if not syns and not ants:
            not_in_wordnet.append(key)
            continue

        hit_syns = [s for s in syns if s in seed_keys]
        if hit_syns:
            orientations = {seed_map[s] for s in hit_syns}
            if len(orientations) == 1:
                seed_map[key] = orientations.pop()
                seed_keys.add(key)
                newly_assigned.append(key)
                continue

        hit_ants = [a for a in ants if a in seed_keys]
```

```
        if hit_ants:
            orientations = {seed_map[a] for a in hit_ants}
            if len(orientations) == 1:
                seed_map[key] = invert_orientation(orientations.pop())
                seed_keys.add(key)
                newly_assigned.append(key)
                continue

    return len(newly_assigned), not_in_wordnet

def orientation_prediction(adjective_list, seed_map):
    seed_map.update({normalise(k): v for k, v in list(seed_map.items())})
    total_new = 0
    discarded = set()

    while True:
        size1 = len(seed_map)
        num_added, not_in_wn = orientation_search(adjective_list, seed_map)
        total_new += num_added
        discarded.update(not_in_wn)
        size2 = len(seed_map)
        if size1 == size2:
            break

    unresolved = [normalise(word) for word in adjective_list if normalise(word) not in seed_map

    return {
        "seed_map": seed_map,
        "newly_added_total": total_new,
        "discarded_not_in_wordnet": sorted(set(discarded)),
        "unresolved_after_propagation": sorted(set(unresolved)),
    }

def display_orientation_balance(result):
    orientation_counts = collections.Counter("positive" if (orientation or "").lower().startswi

    total = orientation_counts.get("positive", 0) + orientation_counts.get("negative", 0)
    rows = [
        {"Orientation": "positive", "Count": orientation_counts.get("positive", 0),
         "Share (%)": round(100 * orientation_counts.get("positive", 0) / total, 1)},
        {"Orientation": "negative", "Count": orientation_counts.get("negative", 0),
         "Share (%)": round(100 * orientation_counts.get("negative", 0) / total, 1)},
    ]
    df_overall_orientation = pd.DataFrame(rows)

    display(df_overall_orientation)
    FigureUtilities.table_caption(
        "overall_orientation_counts",
        "Overall counts of positive versus negative entries in the learned opinion orientation
    )

def display_orientation_prediction_summaries(result):
    orientation_prediction_df = pd.DataFrame([{
        "Newly Added": result["newly_added_total"],
        "Discarded (Not in WordNet)": len(result["discarded_not_in_wordnet"]),
        "Unresolved After Propagation": len(result["unresolved_after_propagation"]),
        "Final Seed Size": len(result["seed_map"]),
    }])

    display(orientation_prediction_df)
    FigureUtilities.table_caption(
        "orientation_prediction_summary",
        "Summary of the orientation prediction algorithm.\n"
    )

    orientation_examples_df = (
        pd.DataFrame(list(result["seed_map"].items()), columns=["opinion_word", "orientation"])
        .sort_values("opinion_word")
        .reset_index(drop=True)
    )

    display(orientation_examples_df.head(20))
    FigureUtilities.table_caption(
        "orientation_prediction_examples",
        "Example opinion words and their inferred orientations.\n"
```

```
    )

    discarded_df = pd.DataFrame(
        result["discarded_not_in_wordnet"], columns=["Opinion Words Discarded (Not in WordNet)"
    )
    display(discarded_df.head(20))
    FigureUtilities.table_caption(
        "orientation_prediction_discarded",
        "Sample of opinion words discarded because no matching WordNet synsets were found.\n"
    )

    unresolved_df = pd.DataFrame(
        result["unresolved_after_propagation"], columns=["Opinion Words Unresolved After Propag
    )
    display(unresolved_df.head(20))
    FigureUtilities.table_caption(
        "orientation_prediction_unresolved",
        "Sample of opinion words left unresolved after propagation by the orientation predictio
    )

train_reviews_opinion_words = sorted({
    token.lemma_ for review in train_reviews
    for sentence in review["sentences"]
    for token in nlp(sentence["text"])
    if token.pos_ == Constants.ADJECTIVE_POS
})
seed = {
    row["word"]: row["sentiment"]
    for i, row in sentiments.iterrows()
    if row["word"] in adjective_opinion_words
}
result = orientation_prediction(train_reviews_opinion_words, seed)

display_orientation_balance(result)
display_orientation_prediction_summaries(result)
```

	Orientation	Count	Share (%)
0	positive	907	41.9
1	negative	1257	58.1

Table 12. Overall counts of positive versus negative entries in the learned opinion orientation lexicon.

	Newly Added	Discarded (Not in WordNet)	Unresolved After Propagation	Final Seed Size
0	478	312	200	2164

Table 13. Summary of the orientation prediction algorithm.

	opinion_word	orientation
0	1st	positive
1	20th	positive
2	21st	positive
3	2nd	positive
4	3rd	positive
5	4th	positive
6	5th	positive
7	able	positive
8	abnormal	negative
9	abominable	negative
10	about	positive
11	above	positive
12	abrasive	negative
13	abrupt	negative
14	absolute	positive
15	absurd	negative
16	abundant	positive
17	abusive	negative
18	abysmal	negative
19	acceptable	positive

Table 14. Example opinion words and their inferred orientations.

	Opinion Words Discarded (Not in WordNet)
0	*
1	+
2	+1/3
3	-
4	-lrb-
5	-rrb-
6	-touchy
7	.00000001
8	.txt
9	2000pro
10	24hrs
11	2mp
12	3hp
13	3ply
14	5ghz
15	66mhz
16	98db
17	=)
18	\
19	aac

Table 15. Sample of opinion words discarded because no matching WordNet synsets were found.

	Opinion Words Unresolved After Propagation
0	academic
1	addicted
2	administrative
3	advertised
4	aesthetic
5	alphabetical
6	amateur
7	american
8	analog
9	anti
10	antibacterial
11	appreciated
12	artistic
13	atlantic
14	attendant
15	attractable
16	automatic
17	average
18	avid
19	big

Table 16. Sample of opinion words left unresolved after propagation by the orientation predictio
n algorithm.

Rules Based Sentiment Analysis Methodology

The `SentenceOrientation` algorithm (Hu and Bing, 2004) attributes sentiment based on the adjectives in a sentence, using heuristics for edge cases such as negations and neutral outcomes. Adjectives are identified through POS tagging, and their sentiment scores come from the `OrientationPrediction` algorithm. Negation words within ± 5 tokens flip the orientation, with the list of negations drawn from the original paper and the `train_reviews` dataset. If the sum is $+1$ or -1, that value is assigned. Otherwise, the algorithm falls back to the sentence's effective opinion, defined as the sentiment of the nearest opinion words to extracted aspects. If this is inconclusive, the score is inherited from the previous sentence, defaulting to 0.

The implementation only recognises positive and negative sentiment. While per-sentence aggregation can yield 0, the dataset annotations always assign non-zero aspect-level sentiment. Furthermore, introducing a neutral class would break comparability with the BERT subsystem, which directly predicts per-aspect non-zero sentiment. Neutral outcomes are therefore excluded from final evaluation, though coverage is reported to account for these missing datapoints.

In [20]:
```
"""
The SentenceOrientation algorithm to determine the sentiment of a sentence.
"""
NEGATION_WORDS = {"not", "never", "no", "hardly", "barely", "n't", "nt", "but", "however"}
SENTENCE_ORIENTATION_POSITIVE = 1
SENTENCE_ORIENTATION_NEGATIVE = -1
SENTENCE_ORIENTATION_NEUTRAL = 0

def invert_orientation(o: int) -> int:
    if o == SENTENCE_ORIENTATION_POSITIVE: return SENTENCE_ORIENTATION_NEGATIVE
    if o == SENTENCE_ORIENTATION_NEGATIVE: return SENTENCE_ORIENTATION_POSITIVE
    return o
```

```
def word_orientation(token, doc, seed_map, window):
    key = token.lemma_.lower()
    if key not in seed_map:
        return 0

    ori = SENTENCE_ORIENTATION_POSITIVE if seed_map[key] == "positive" else SENTENCE_ORIENTATIO

    index = token.i
    context = doc[max(0, index - window): min(len(doc), index + window + 1)]
    if any(tok.text.lower() in NEGATION_WORDS for tok in context):
        ori = -ori

    return ori

def feature_effective_opinions(sent_doc, sent_aspects, seed_map, window):
    """
    Find effective opinions for provided aspects within a sentence.
    Uses spaCy's Matcher to locate exact multi-word aspect spans.

    :param sent_doc: spaCy Doc object (tokenized sentence)
    :param sent_aspects: list of aspect strings (multi-word allowed)
    :param seed_map: dict of known opinion words -> orientation ("positive"/"negative"), keyed
    :param window: max distance to search around aspect tokens
    :return: list[str] opinion lemmas (possibly prefixed with "__NEG__:") nearest to each aspec
    """
    matcher = Matcher(sent_doc.vocab)
    for aspect in sent_aspects:
        words = aspect.lower().split()
        pattern = [{"LOWER": word} for word in words]
        matcher.add(aspect, [pattern])

    results = []
    matches = matcher(sent_doc)
    for match_id, start, end in matches:
        # Search for nearest seed word by lemma within window
        best = None
        best_dist = 10 ** 9

        for j, tok in enumerate(sent_doc):
            if tok.lemma_.lower() in seed_map:
                d = min(abs(j - start), abs(j - (end - 1)))
                if d <= window and d < best_dist:
                    best, best_dist = tok, d

        if best:
            # Check for negation before opinion word
            left = sent_doc[max(0, best.i - window): best.i]
            flip = any(x.text.lower() in NEGATION_WORDS for x in left)
            effective = "__NEG__:" + best.lemma_.lower() if flip else best.lemma_.lower()
            results.append(effective)

    return results

def score_opinion_token(token, seed_map):
    flip = False
    if token.startswith("__NEG__:"):
        flip = True
        token = token.split(":", 1)[1]

    key = token
    if key not in seed_map:
        return 0

    s = SENTENCE_ORIENTATION_POSITIVE if seed_map[key] == "positive" else SENTENCE_ORIENTATION_
    return -s if flip else s

def sentence_orientation(doc, sent_aspects, seed_map, prev_orientation, window):
    score = 0

    for w in doc:
        score += word_orientation(w, doc, seed_map, window)

    if score > 0:
        return SENTENCE_ORIENTATION_POSITIVE
    if score < 0:
```

```
        return SENTENCE_ORIENTATION_NEGATIVE

    for token in feature_effective_opinions(doc, sent_aspects, seed_map, window):
        score += score_opinion_token(token, seed_map)

    if score > 0:
        return SENTENCE_ORIENTATION_POSITIVE
    if score < 0:
        return SENTENCE_ORIENTATION_NEGATIVE

    return prev_orientation

def document_orientation(reviews, seed_map, test_reviews_predicted_aspects, use_predicted_aspec
    all_sents = [sent["text"] for review in reviews for sent in review["sentences"]]
    docs = list(nlp.pipe(all_sents))

    i = 0
    results = []
    for review in reviews:
        prev = SENTENCE_ORIENTATION_NEUTRAL
        review_result = []
        for sent in review["sentences"]:
            doc = docs[i]
            aspects = test_reviews_predicted_aspects[i]["pred"] if use_predicted_aspects else t
            score = sentence_orientation(doc, aspects, seed_map, prev, window=5)
            review_result.append({
                "sentence": sent,
                "sentiment": score,
                "pred_aspects": test_reviews_predicted_aspects[i]["pred"],
                "aspects": test_reviews_predicted_aspects[i]["ground_truth"],
            })
            prev = score or prev
            i += 1
        results.append(review_result)
    return results

def display_rules_based_examples(test_review_results):
    flat_rows = []
    for review_id, review_result in enumerate(test_review_results):
        for sent_id, entry in enumerate(review_result):
            flat_rows.append({
                "review_id": review_id,
                "sent_id": sent_id,
                "sentence": entry["sentence"]["text"],
                "sentiment": entry["sentiment"],
                "ground_truth_aspects": ", ".join(entry["aspects"]) if entry["aspects"] else ""
            })

    df = pd.DataFrame(flat_rows)
    display(df)
    FigureUtilities.table_caption("rules_sentiment_results_examples_table", "Rules based sentim

test_review_results = document_orientation(test_reviews, result["seed_map"], test_reviews_predi
```

```
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
```

Rules Based Sentiment Analysis Evaluation

The algorithm achieves a good level of performance, with $\text{Precision}^{\text{macro}} = 72.4\%$, $\text{Recall}^{\text{macro}} = 70.5\%$ and $F_1^{\text{macro}} = 71.2\%$. $\text{Coverage} = 94.6\%$, indicating that the subsystem has balanced class performance and predicts an accurate sentiment for the majority of aspects. A confusion matrix is plotted to visualise performance (Figure 7).

There is no evidence of a negative sentiment bias despite the sentiment orientation class imbalance introduced by the `OrientationPrediction` algorithm. Furthermore, it is perhaps surprising that enforcing an assumption as strong as a single sentiment per sentence could yield good performance, although this assumption is likely to break down in different contexts, for example, if a corpus has significantly more than 1.25 average opinion words per sentence (Figure 5).

In [33]:
```
"""
Rules-based sentiment analysis evaluation on the held-out test set.
"""
def evaluate_sentence_orientation(test_review_results):
    expected, predicted = [], []
    neutral = 0

    for test_review_result in test_review_results:
        for row in test_review_result:
            expected_feature_sentiment = {}
            actual_feature_sentiment = {}

            for annotation in row["sentence"]["annotations"]:
                sentiment_score = 1 if int(annotation["sentiment"]) > 0 else -1
                expected_feature_sentiment[annotation["feature"]] = sentiment_score
                actual_feature_sentiment[annotation["feature"]] = row["sentiment"]

            for aspect, ground_truth in expected_feature_sentiment.items():
                pred = actual_feature_sentiment.get(aspect, 0)
                if pred in (-1, 1):
                    expected.append(ground_truth)
                    predicted.append(pred)
                else:
                    neutral += 1

    precision, recall, f1, _ = precision_recall_fscore_support(expected, predicted, labels=[1,

    # Back counts out from confusion matrix
    cm = confusion_matrix(expected, predicted, labels=[1, -1])
    tp, fn = int(cm[0, 0]), int(cm[0, 1])
    fp, tn = int(cm[1, 0]), int(cm[1, 1])

    return precision, recall, f1, tp, tn, fp, fn, neutral

def plot_confusion_matrix(tp, tn, fp, fn, title="Confusion Matrix (SentenceOrientation Sentimen
    cm = np.array([[tp, fn],
                   [fp, tn]])
    confusion_matrix_labels = ("Positive", "Negative")
    fig, ax = plt.subplots(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=confusion_matrix_labels,
                yticklabels=confusion_matrix_labels,
                cbar=False, ax=ax)
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Ground Truth")
    ax.set_title(title)
    plt.show()

def display_rules_based_eval_results(precision, recall, f1, tp, tn, fp, fn, coverage, covered):
    metrics = {
        "Precision (macro)": round(precision * 100, 1),
        "Recall (macro)": round(recall * 100, 1),
        "F1 (macro)": round(f1 * 100, 1),
        "Coverage": round(coverage * 100, 1),
    }
    df = pd.DataFrame([metrics]).T
    df.columns = ["Value (%)"]
    display(df)
    FigureUtilities.table_caption("rules_sentiment_results_table", "Rules-based sentiment analy

    # Sentiment distribution as percentages (including neutrals)
    total = covered + z
    pos_pct = 100 * (tp + fn) / total if total else 0.0
    neg_pct = 100 * (fp + tn) / total if total else 0.0
    neu_pct = 100 * z / total if total else 0.0

    dist = {"Positive": round(pos_pct, 1),
            "Negative": round(neg_pct, 1),
            "Neutral": round(neu_pct, 1)}
    df = pd.DataFrame([dist]).T
    df.columns = ["Value (%)"]
    display(df)
    FigureUtilities.table_caption("rules_review_sentences_sentiment_table", "Class breakdown of
```

```
def plot_rules_based_confusion_matrix(tp, tn, fp, fn):
    plot_confusion_matrix(tp, tn, fp, fn)
    FigureUtilities.figure_caption("rules_review_sentences_sentiment_plot", "Confusion matrix s

precision, recall, f1, tp, tn, fp, fn, z = evaluate_sentence_orientation(test_review_results)
covered = tp + tn + fp + fn
coverage = covered / (covered + z) if (covered + z) > 0 else 0.0

display_rules_based_eval_results(precision, recall, f1, tp, tn, fp, fn, coverage, covered)
plot_rules_based_confusion_matrix(tp, tn, fp, fn)
```

	Value (%)
Precision (macro)	72.4
Recall (macro)	70.5
F1 (macro)	71.2
Coverage	94.6

Table 17. Rules-based sentiment analysis macro precision, recall, F1, and coverage.

	Value (%)
Positive	66.4
Negative	28.2
Neutral	5.4

Table 18. Class breakdown of rules-based sentiment assignments on the test dataset.

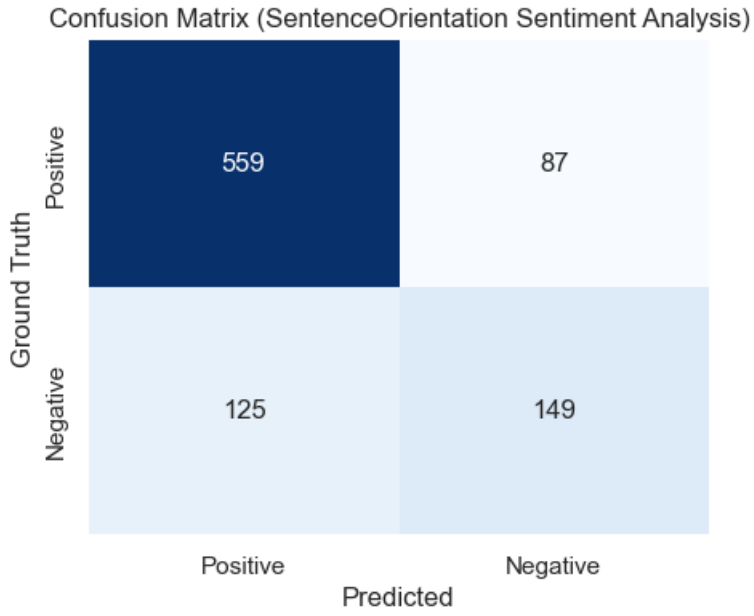

Figure 7. Confusion matrix showing test dataset results for the SentenceOrientation algorithm.

BERT Sentiment Analysis

The second sentiment analysis subsystem variant leverages BERT (Devlin et al., 2019), a transformer-based architecture. Transformer models use self-attention, which captures contextual dependencies across an entire sequence rather than fixed-size windows (Figure 8). This is valuable in sentiment analysis where polarity often depends on long-range context and subtle linguistic cues. BERT extends the transformer by jointly considering left and right context, producing richer representations, and it also applies sub-word tokenisation with the WordPiece algorithm to handle typos, rare words, and morphological variation (Hugging Face, 2025).

BERT is pre-trained on large English corpora using masked language modelling and next sentence prediction, and then fine-tuned on smaller task-specific datasets (Géron, 2022, p. 621). This approach makes BERT a flexible, powerful basis for sentiment analysis. Compared to the rules-based subsystem, it is more context-sensitive and better able to address multiple aspects per sentence, non-literal language, negation, and comparative statements.

In [34]:
```
"""
Displaying a transformer layer figure and a system design diagram for the BERT based sentiment
"""
display(HTML('<img src="assets/TLD.png" alt="Attention formula" width="400"/>'))

FigureUtilities.figure_caption(
    "transformer_layer_diagram",
    "A transformer layer has three distinct components, self-attention, a feedforward network a
)

display(HTML('<img src="assets/opinion_miner_system_design_BERT.png" alt="BERT sentiment analys

FigureUtilities.figure_caption(
    "opinion_miner_system_design_bert_diagram",
    "System design diagram outlining the high level architecture of the BERT sentiment analysis
)
```

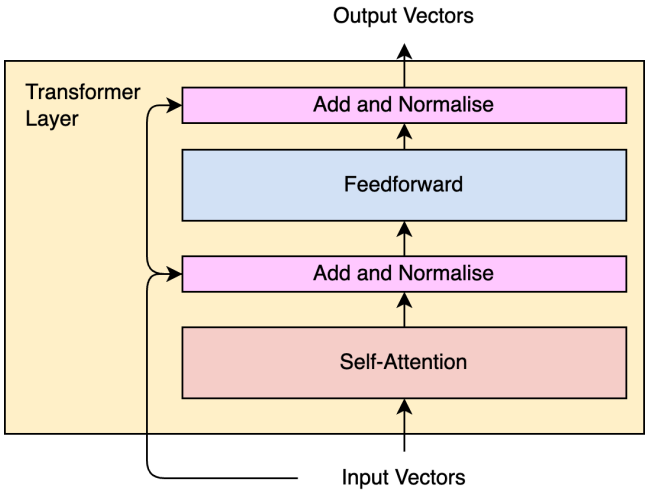

Figure 8. A transformer layer has three distinct components, self-attention, a feedforward netwo
rk and residual connections (adapted from Russell and Norvig, 2022).

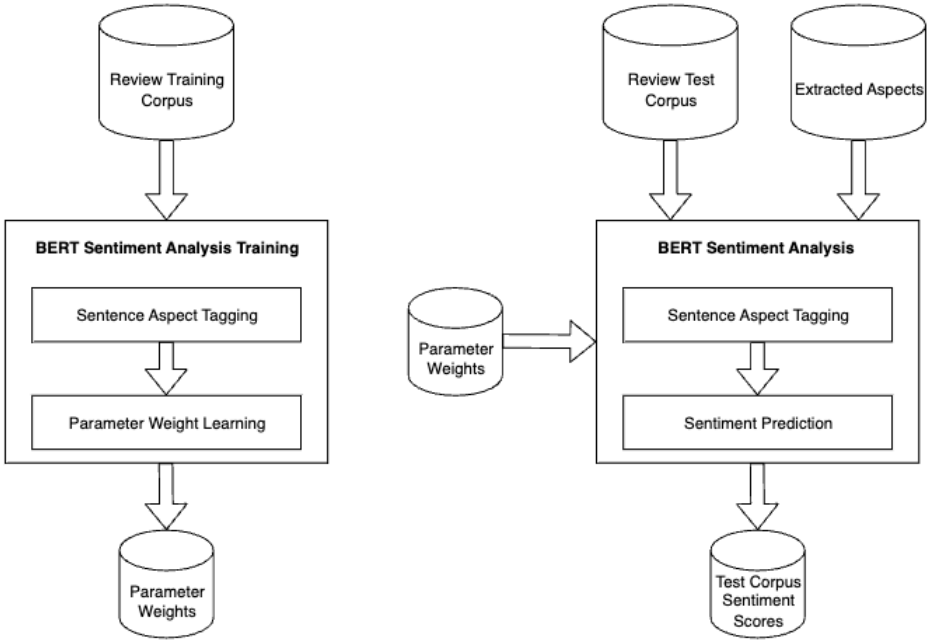

Figure 9. System design diagram outlining the high level architecture of the BERT sentiment anal
ysis subsystem.

BERT Sentiment Analysis Preprocessing

The training dataset was split into an 80% validation set and a 20% development set, stratified by product and aggregate sentiment. Because sentences are annotated with aspects only when they convey non-neutral

sentiment, the datasets support only a two-class BERT classifier: positive and negative. Therefore, unannotated sentences are excluded from training, development, and test sets.

Ground Truth

The ground truth for each aspect sentiment score per sentence is the normalised annotated sentiment score, reflecting the capability of this subsystem variant to attribute sentiment to individual aspects. Scores are normalised to 1 or -1 according to the sign of the annotated sentiment score.

Original Sentence	Ground Truth Aspect	Ground Truth Sentiment	Standardised Ground Truth Sentiment
battery[+2]##(batteries last longer too !)	Battery	+2	+1
batteries[-3]## This was obviously ... replaceable batteries!!	Batteries	−3	−1
look[+1], speed[−3]## Looks good, but the speed is terrible	Look, Speed	+1, −3	+1, −1

In [35]:
```
"""
Create new validation, development and test datasets suitable for BERT training.
"""
LABEL2ID = {"NEG": 0, "POS": 1}
ID2LABEL = {v: k for k, v in LABEL2ID.items()}

def build_bert_compatible_dataset(revs):
    review_entities = dict()
    review_objects = []
    for i, review in enumerate(revs):
        review_entities["product_id"] = review["product"]
        for sent in review["sentences"]:
            for annot in sent["annotations"]:
                review_entities["review_index"] = i
                review_entities["sentence"] = sent["text"]
                review_entities["target"] = annot["feature"]
                review_entities["label"] = "POS" if int(annot["sentiment"]) >= 1 else "NEG"
                review_objects.append(review_entities.copy())
    return review_objects

def load_your_splits():
    val_reviews, dev_reviews, y_val, y_dev = ReviewPartitionUtility.stratify(train_reviews)

    train = build_bert_compatible_dataset(val_reviews)
    dev = build_bert_compatible_dataset(dev_reviews)
    test = build_bert_compatible_dataset(test_reviews)

    return train, dev, test, val_reviews, dev_reviews

train_rows, dev_rows, test_rows, val_reviews, dev_reviews = load_your_splits()

data = {
    "Split": ["Validation", "Development"],
    "Reviews": [
        len(val_reviews),
        len(dev_reviews)
    ],
    "Sentences": [
        ReviewPartitionUtility.count_sentences(val_reviews),
        ReviewPartitionUtility.count_sentences(dev_reviews),
    ],
    "Annotated": [
        ReviewPartitionUtility.count_annotated(val_reviews),
        ReviewPartitionUtility.count_annotated(dev_reviews),
    ],
}

df = pd.DataFrame(data)
display(df)
```

```
Loaded 2795 reviews
Created 2795 stratification keys
Found 0 product/sentiment pairs with less than two samples
2795 reviews remain after low sample filtering
2795 stratification keys remain after low sample filtering
```

	Split	Reviews	Sentences	Annotated
0	Validation	2236	6841	3847
1	Development	559	1668	905

To prepare the data for aspect-based sentiment analysis, the subsystem marks aspects within each sentence. Special tokens, `<TGT>` and `</TGT>`, are inserted around the first occurrence of the target expression, providing clear boundary signals during training. These tokens are also added to the BERT tokenizer's vocabulary so trainable embeddings could be learned. Each example is then encoded with the marked sentence and the target term as a paired input, producing the datasets for training, model selection or early stopping, and evaluation.

In [36]:
```
"""
Transform datasets into a suitable format by marking aspects in sentences.
"""
START_TOK, END_TOK = "<TGT>", "</TGT>"

def mark_target(sentence: str, target: str) -> str:
    s_low, t_low = sentence.lower(), target.lower()
    i = s_low.find(t_low)
    if i == -1:
        return sentence
    j = i + len(t_low)
    return sentence[:i] + f"{START_TOK} " + sentence[i:j] + f" {END_TOK}" + sentence[j:]

class ABSADataset(Dataset):
    def __init__(self, rows, tokenizer, max_len=256, has_labels=True):
        self.rows = rows
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.has_labels = has_labels

    def __len__(self):
        return len(self.rows)

    def __getitem__(self, i):
        r = self.rows[i]
        marked = mark_target(r["sentence"], r["target"])
        enc = self.tokenizer(
            marked,
            r["target"],
            truncation=True,
            padding=False,
            max_length=self.max_len,
            return_tensors=None,
        )

        if self.has_labels and ("label" in r) and (r["label"] is not None):
            lab = r["label"]
            lab_id = LABEL2ID[lab] if isinstance(lab, str) else int(lab)
        else:
            lab_id = -100

        enc["labels"] = torch.tensor(lab_id, dtype=torch.long)
        return enc

if os.path.isdir(Constants.SAVE_DIR) and os.path.exists(os.path.join(Constants.SAVE_DIR, "confi
    print("Loading cached model/tokenizer…")
    tokenizer = BertTokenizerFast.from_pretrained(Constants.SAVE_DIR)
    model = BertForSequenceClassification.from_pretrained(Constants.SAVE_DIR, output_attentions
else:
    print("Fresh model/tokenizer…")
    tokenizer = BertTokenizerFast.from_pretrained("bert-base-uncased", output_attentions=True)
    tokenizer.add_tokens([START_TOK, END_TOK], special_tokens=True)
```

```
    model = BertForSequenceClassification.from_pretrained(
        "bert-base-uncased",
        num_labels=len(LABEL2ID),
        id2label=ID2LABEL,
        label2id=LABEL2ID,
    )
    model.resize_token_embeddings(len(tokenizer))

train_ds = ABSADataset(train_rows, tokenizer)
dev_ds = ABSADataset(dev_rows, tokenizer)
test_ds = ABSADataset(test_rows, tokenizer)
```

```
Loading cached model/tokenizer…
```

BERT Sentiment Analysis Methodology

The pre-trained 110 million parameter `bert-base-uncased` model is fine-tuned for sequence classification using the Hugging Face Transformers library. Training runs for 5 epochs with a batch size of 16. The `BertForSequenceClassification` model is used, which adds a classification layer on top of `bert-base-uncased`. All parameters are fine-tuned by minimising cross-entropy loss on the training dataset (Devlin et al., 2019), with the classifier trained to output the correct label, `POS` or `NEG`. Model selection and early stopping use the held-out development set.

Performance is evaluated using F_1^{macro}, giving equal weight to both classes. The final model and tokenizer are cached for reuse in later opinion mining experiments.

In [37]:
```
"""
Fine tuning the BERT model for sentence-based binary sentiment analysis classification.

Takes approximately 15 minutes to run on an Apple M3 MAX CPU with MPS enabled.
"""
def compute_metrics(eval_pred):
    logits, labels = eval_pred

    if isinstance(logits, tuple):
        logits = logits[0]

    preds = logits.argmax(-1)
    acc = (preds == labels).mean()
    f1_macro = sk_f1(labels, preds, average="macro", zero_division=0)
    precision_macro = precision_score(labels, preds, average="macro", zero_division=0)
    recall_macro = recall_score(labels, preds, average="macro", zero_division=0)
    return {
        "accuracy": float(acc),
        "f1_macro": float(f1_macro),
        "precision_macro": float(precision_macro),
        "recall_macro": float(recall_macro),
    }

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

if torch.cuda.is_available():
    device = torch.device("cuda")
    print("Using CUDA GPU")
elif torch.backends.mps.is_available():
    device = torch.device("mps")
    print("Using Apple MPS")
else:
    device = torch.device("cpu")
    print("Using CPU")

args = TrainingArguments(
    output_dir="outputs/aspect-bert",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    greater_is_better=True,
    fp16=torch.cuda.is_available(),
```

```
    no_cuda=not torch.cuda.is_available(),
    save_total_limit=2
)

trainer = Trainer(
    model=model.to(device),
    args=args,
    train_dataset=train_ds,
    eval_dataset=dev_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

if not os.path.isdir(Constants.SAVE_DIR) or not os.path.exists(os.path.join(Constants.SAVE_DIR,
    print("Training…")
    trainer.train()
    trainer.save_model(Constants.SAVE_DIR)
    tokenizer.save_pretrained(Constants.SAVE_DIR)

model.config.output_attentions = False
```

```
Using Apple MPS
```

The below code evaluates the development set using the final trained model weights. Performance is strong with $F_1^{\text{macro}} = 90.5\%$.

In [38]:
```
pred = trainer.predict(dev_ds)
dev_metrics = pred.metrics
df_dev = pd.DataFrame([dev_metrics])
display(df_dev)
FigureUtilities.table_caption("dev_ds", "Development set evaluation BERT sentiment analysis per
```

	test_loss	test_accuracy	test_f1_macro	test_precision_macro	test_recall_macro	test_runtime	test_samp
0	0.413113	0.918232	0.905249	0.907614	0.903003	10.6321	

Table 19. Development set evaluation BERT sentiment analysis performance metrics.

BERT Sentiment Analysis Evaluation

The fine-tuned BERT model achieves $\text{Precision}^{\text{macro}} = 91.1\%$, $\text{Recall}^{\text{macro}} = 89.9\%$, $F_1^{\text{macro}} = 90.5\%$ when tested using the held-out test dataset, implying a well-balanced classifier with strong performance. A confusion matrix is plotted to visualise performance (Figure 10).

In [39]:
```
"""
Assess model performance on the held-out test set and report the results.
"""
pred = trainer.predict(test_ds)

test_metrics = pred.metrics
df_test = pd.DataFrame([test_metrics])
display(df_test)
FigureUtilities.table_caption("test_ds", "Test set evaluation final BERT sentiment analysis per

y_true = pred.label_ids
y_pred = pred.predictions.argmax(axis=-1)

# Plot a confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=[1, 0])

plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Positive", "Negative"],
            yticklabels=["Positive", "Negative"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (BERT Sentiment Analysis)")
plt.show()
FigureUtilities.figure_caption("bert_cm", "Confusion matrix showing BERT classification perform
```

	test_loss	test_accuracy	test_f1_macro	test_precision_macro	test_recall_macro	test_runtime	test_samp
0	0.392026	0.920863	0.904962	0.911454	0.899173	9.985	

Table 20. Test set evaluation final BERT sentiment analysis performance metrics.

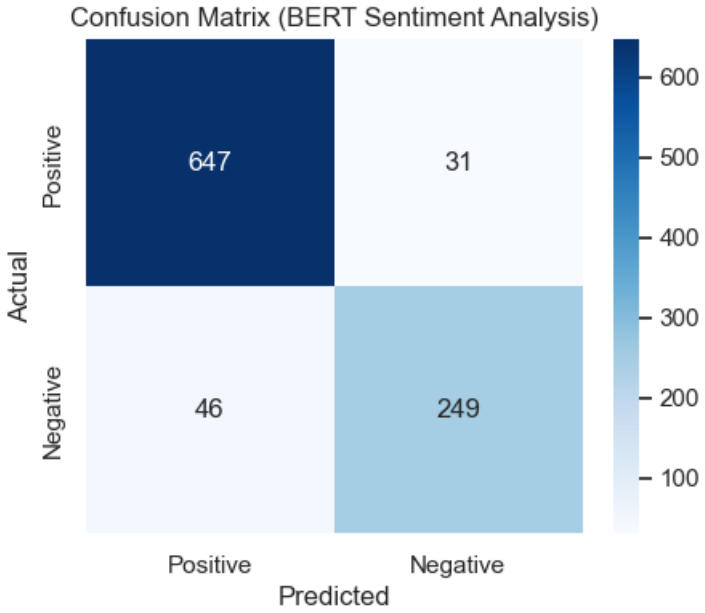

Figure 10. Confusion matrix showing BERT classification performance on the held-out test set.

Sentiment Analysis Comparative Evaluation

As hypothesised, the BERT subsystem significantly outperforms the rules-based subsystem, with improvements of $\text{Precision}^{\text{macro}}_{\text{BERT}-\text{Rules Based}} = 18.7\%$, $\text{Recall}^{\text{macro}}_{\text{BERT}-\text{Rules Based}} = 19.4\%$, $F^{\text{macro}}_{1,\text{BERT}-\text{Rules Based}} = 19.3\%$, and $\text{Coverage}^{\text{macro}}_{\text{BERT}-\text{Rules Based}} = 5.4\%$.

The rules-based subsystem is simpler and more transparent, which may matter in domains where interpretability is essential, such as sentiment-driven investment decisions. However, the performance gap is substantial, making it difficult to justify selection over BERT. Moreover, while rules-based is simpler in principle, it is more difficult to design and maintain, as it is hand-crafted. In contrast, BERT based systems can be deployed with minimal effort through frameworks such as Hugging Face, without needing deep knowledge of the underlying architecture. The rules-based subsystem is also limited to producing a single sentiment per sentence, which may prove severely detrimental if applied to domains in which multiple aspects and opinions per sentence are common.

Computational complexity provides another point of contrast. Training BERT over 5 epochs with a batch size of 16 is more expensive than running the `SentenceOrientation` algorithm. However, in production, inference time is more important. Table 22 and Figure 11 show that the inference time of both subsystem variants scales linearly with input size, but BERT has a higher baseline cost due to its larger architecture. Thus, for small inputs or latency-sensitive applications like high-frequency trading, rules-based may be preferable. For larger datasets, BERT's lower scaling exponent makes it more efficient overall.

Rules-based is an unsupervised algorithm, whereas BERT is supervised. In contexts with little or no training data, rules-based may be the better choice as BERT requires fine-tuning. Despite being fit to training data, the BERT subsystem variant clearly overcomes the test dataset's $2:1$ class imbalance towards positive reviews (Table 18), showing little sign of bias.

As previously noted, BERT's advantages extend to handling colloquial language and complex opinion expressions, among others, making it suitable for cross-domain application and complex sentence structures. Given its significant performance advantage, the BERT subsystem is likely to be the model of choice for most applications.

Attention

Tables 23-25 illustrate how BERT sentiment analysis can produce more granular aspect-based scores. In the test sentence “Looks good, but the speed is terrible.” two aspects appear with opposite polarity. The rules-based algorithm cannot handle this, as it assigns only a single sentiment per sentence. In contrast, the transformer links each aspect to its correct opinion word. For example, "speed" with "terrible" rather than "good". This is enabled by self-attention, which quantifies the strength of relationships between each aspect and all other tokens in the sequence.

In [40]:
```
"""
Whole system metrics and empirically inferring sentiment analysis subsystem runtimes.
"""
rules_based_metrics = {
    "Precision (macro)": round(precision * 100, 1),
    "Recall (macro)": round(recall * 100, 1),
    "F1 (macro)": round(f1 * 100, 1),
    "Coverage (on annotated sentences)": round(coverage * 100, 1),
}

# Extract scalars from df_test row 0 and scale to %
p = round(float(df_test.loc[0, "test_precision_macro"]) * 100, 1)
r = round(float(df_test.loc[0, "test_recall_macro"]) * 100, 1)
f = round(float(df_test.loc[0, "test_f1_macro"]) * 100, 1)

bert_metrics = {
    "Precision (macro)": p,
    "Recall (macro)": r,
    "F1 (macro)": f,
    "Coverage (on annotated sentences)": 100.0,
}

# Combine into one DataFrame
df_combined = pd.DataFrame({
    "Rules Based Score (%)": pd.Series(rules_based_metrics),
    "BERT Score (%)": pd.Series(bert_metrics),
})
df_combined["Difference (BERT - Rules Based) Score (%)"] = df_combined["BERT Score (%)"] - df_c

display(df_combined)
FigureUtilities.table_caption(
    "bert_vs_rules_metrics",
    "Comparison of macro-averaged precision, recall, F1, and coverage between the rules-based a
)

def benchmark_inference_scaling(test_reviews, test_rows, seed_map, tokenizer, trainer, window=5
    fractions = [i / 10 for i in range(1, 11)]
    records = []

    for f in fractions:
        n_rev = max(1, int(len(test_reviews) * f))
        n_rows = max(1, int(len(test_rows) * f))
        reviews_subset = test_reviews[:n_rev]
        rows_subset = test_rows[:n_rows]

        t0 = time.perf_counter()
        _ = document_orientation(reviews_subset, seed_map, test_reviews_predicted_aspects)
        rules_elapsed = time.perf_counter() - t0

        n_sents = sum(len(r["sentences"]) for r in reviews_subset)
        ds = ABSADataset(rows_subset, tokenizer)

        t1 = time.perf_counter()
        _ = trainer.predict(ds)
        bert_elapsed = time.perf_counter() - t1

        records.append({
            "Fraction": f,
            "Reviews Subset": n_rev,
            "Rules Based Sentences Processed": n_sents,
            "BERT Sentences Processed": n_rows,
            "Rules Based Time Elapsed (ms)": rules_elapsed * 1000.0,
            "BERT Time Elapsed (ms)": bert_elapsed * 1000.0,
        })

    return pd.DataFrame(records)
```

```
df_scaling = benchmark_inference_scaling(
    test_reviews=test_reviews,
    test_rows=test_rows,
    seed_map=result["seed_map"],
    tokenizer=tokenizer,
    trainer=trainer
)

display(df_scaling)
FigureUtilities.table_caption(
    "inference_scaling",
    "Inference time vs. test set fraction for rules-based and BERT subsystems. "
    "BERT does not process unannotated sentences, explaining the discrepancy between "
    "\"Rules Based Sentences Processed\" and \"BERT Sentences Processed\".\n"
)

def _estimate_loglog_exponent(x_sizes, y_times):
    m = (np.array(x_sizes) > 0) & (np.array(y_times) > 0)
    x = np.log(np.array(x_sizes)[m])
    y = np.log(np.array(y_times)[m])
    if len(x) < 2:
        return np.nan, np.nan, m
    slope, intercept = np.polyfit(x, y, 1)
    return slope, intercept, m

exponent_rules, int_rules, mask_rules = _estimate_loglog_exponent(
    df_scaling["Rules Based Sentences Processed"],
    df_scaling["Rules Based Time Elapsed (ms)"]
)
exponent_bert, int_bert, mask_bert = _estimate_loglog_exponent(
    df_scaling["BERT Sentences Processed"],
    df_scaling["BERT Time Elapsed (ms)"]
)

plot_rows = []
for x, y in zip(
        df_scaling["Rules Based Sentences Processed"][mask_rules],
        df_scaling["Rules Based Time Elapsed (ms)"][mask_rules],
):
    plot_rows.append({"system": "Rules-Based", "sentences": float(x), "time_ms": float(y)})

for x, y in zip(
        df_scaling["BERT Sentences Processed"][mask_bert],
        df_scaling["BERT Time Elapsed (ms)"][mask_bert],
):
    plot_rows.append({"system": "BERT", "sentences": float(x), "time_ms": float(y)})

df_plot = pd.DataFrame(plot_rows)

sns.set_theme(style="whitegrid")
plt.figure(figsize=(6, 5))

ax = sns.scatterplot(
    data=df_plot,
    x="sentences",
    y="time_ms",
    hue="system",
    style="system",
    markers={"Rules-Based": "o", "BERT": "s"},
    s=60,
    legend=True,
)

x_fit_rules = np.linspace(df_plot.loc[df_plot.system == "Rules-Based", "sentences"].min(),
                          df_plot.loc[df_plot.system == "Rules-Based", "sentences"].max(), 200)
y_fit_rules = np.exp(int_rules) * x_fit_rules ** exponent_rules * 1.0
sns.lineplot(x=x_fit_rules, y=y_fit_rules, ax=ax, linestyle="--", label=f"Rules-Based gradient

x_fit_bert = np.linspace(df_plot.loc[df_plot.system == "BERT", "sentences"].min(),
                         df_plot.loc[df_plot.system == "BERT", "sentences"].max(), 200)
y_fit_bert = np.exp(int_bert) * x_fit_bert ** exponent_bert * 1.0
sns.lineplot(x=x_fit_bert, y=y_fit_bert, ax=ax, linestyle="--", label=f"BERT gradient = {expone

ax.set_xscale("log")
```

```
ax.set_yscale("log")
ax.set_xlabel("Sentences processed (log scale)")
ax.set_ylabel("Time elapsed (ms, log scale)")
ax.set_title("Empirical Time Complexity Inference")

ymin, ymax = ax.get_ylim()
pmin = int(np.floor(np.log10(ymin)))
pmax = int(np.ceil(np.log10(ymax)))

yticks = [m * (10 ** p) for p in range(pmin, pmax + 1) for m in (1, 2, 5)]
yticks = [t for t in yticks if ymin <= t <= ymax]

ax.set_yticks(yticks)
ax.yaxis.set_major_formatter(ticker.FuncFormatter(
    lambda y, _:
    f"{int(y):,}" if y >= 1 else f"{y:g}"
))

xmin, xmax = ax.get_xlim()
pmin_x = int(np.floor(np.log10(xmin)))
pmax_x = int(np.ceil(np.log10(xmax)))

xticks = [m * (10 ** p) for p in range(pmin_x, pmax_x + 1) for m in (1, 2, 5)]
xticks = [t for t in xticks if xmin <= t <= xmax]

ax.set_xticks(xticks)
ax.xaxis.set_major_formatter(ticker.FuncFormatter(
    lambda x, _:
    f"{int(x):,}" if x >= 1 else f"{x:g}"
))

plt.legend()
plt.tight_layout()
plt.show()
FigureUtilities.figure_caption("time_complexity", "Empirical inference time scaling showing log

encoder = model.bert
encoder.config.output_attentions = True

def show_attention(sentence, target):
    marked = mark_target(sentence, target)
    inputs = tokenizer(marked, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**inputs, output_attentions=True)

    attention = outputs.attentions
    viz_tokens = tokenizer.convert_ids_to_tokens(inputs["input_ids"][0])

    viz = head_view(attention, viz_tokens)
    display(viz)

def attention_to_target(sentence: str, target: str, layer: int = -1):
    marked = mark_target(sentence, target)
    batch = tokenizer(marked, return_tensors="pt")
    with torch.no_grad():
        out = encoder(**batch, output_attentions=True)
    atts = out.attentions

    A = atts[layer][0]
    A = A.mean(dim=0)

    tokens = tokenizer.convert_ids_to_tokens(batch["input_ids"][0].tolist())

    i_start = tokens.index(START_TOK)
    i_end = tokens.index(END_TOK)
    tgt_col = A[:, i_start + 1:i_end].mean(dim=1)  # (T,)

    scores = tgt_col.numpy()
    return tokens, scores

def top_attenders(sentence: str, target: str, k=10, **kwargs):
    tokens, scores = attention_to_target(sentence, target, **kwargs)
    index = np.argsort(scores)[::-1]
    return [(tokens[i], float(scores[i])) for i in index[:k]]
```

```
def top_attenders_df(sentence: str, targets, k=20, **kwargs) -> pd.DataFrame:
    rows = []
    for t in targets:
        pairs = top_attenders(sentence, t, k=k, **kwargs)  # [(token, score), ...]
        for rank, (token, score) in enumerate(pairs, start=1):
            rows.append({"target": t, "token": token, "attention": float(score), "rank": rank})
    df = pd.DataFrame(rows).sort_values(["target", "rank"]).reset_index(drop=True)
    return df

s = "Looks good, but the speed is terrible."
targets = ["speed", "looks"]

def print_attention_tables():
    df_tidy = top_attenders_df(s, targets=[targets[0]])
    display(df_tidy)
    FigureUtilities.table_caption("speed_attention_table", "Displays the attention scores attri

    df_tidy = top_attenders_df(s, targets=[targets[1]])
    print()
    display(df_tidy)
    FigureUtilities.table_caption("looks_attention_table", "Displays the attention scores attri

def print_predictions_table():
    rows = [
        {"sentence": s, "target": targets[0]},
        {"sentence": s, "target": targets[1]},
    ]

    # Dataset + predict
    ds = ABSADataset(rows, tokenizer, has_labels=False)
    pred = trainer.predict(ds)

    # Unpack logits if tuple, then argmax per item
    logits = pred.predictions[0] if isinstance(pred.predictions, tuple) else pred.predictions
    pred_ids = logits.argmax(-1)

    df = pd.DataFrame(
        {
            "target": [r.get("target", "") for r in rows],
            "pred_id": [int(y) for y in pred_ids],
        }
    )
    df["pred_label"] = df["pred_id"].map(lambda i: ID2LABEL[int(i)])

    if any("sentence" in r for r in rows):
        df["sentence"] = [r.get("sentence", "") for r in rows]
    if any("product_id" in r for r in rows):
        df["product_id"] = [r.get("product_id", "") for r in rows]

    cols = [c for c in ["product_id", "sentence", "target", "pred_id", "pred_label"] if c in df
    df = df[cols]
    print()
    display(df)
    FigureUtilities.table_caption("bert_polarity_table", "Demonstrating the BERT model's abilit

print_attention_tables()
print_predictions_table()

model.config.output_attentions = False
```

	Rules Based Score (%)	BERT Score (%)	Difference (BERT - Rules Based) Score (%)
Precision (macro)	72.4	91.1	18.7
Recall (macro)	70.5	89.9	19.4
F1 (macro)	71.2	90.5	19.3
Coverage (on annotated sentences)	94.6	100.0	5.4

Table 21. Comparison of macro-averaged precision, recall, F1, and coverage between the rules-based and BERT-based sentiment analysis subsystems. The final column reports the difference (BERT − Rules-based) in percentage points.

```
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
```

	Fraction	Reviews Subset	Rules Based Sentences Processed	BERT Sentences Processed	Rules Based Time Elapsed (ms)	BERT Time Elapsed (ms)
0	0.1	69	118	97	202.589167	1233.195000
1	0.2	139	386	194	638.370875	2020.889000
2	0.3	209	536	291	1060.291292	3085.291792
3	0.4	279	734	389	1056.781541	3692.290750
4	0.5	349	915	486	1349.528000	4709.657750
5	0.6	419	1077	583	1579.473000	5281.065334
6	0.7	489	1258	681	1871.736500	6071.299750
7	0.8	559	1381	778	2016.832709	6915.650875
8	0.9	629	1553	875	2248.147667	7899.724542
9	1.0	699	1780	973	2643.917375	8564.208375

```
Table 22. Inference time vs. test set fraction for rules-based and BERT subsystems. BERT does no
t process unannotated sentences, explaining the discrepancy between "Rules Based Sentences Proce
ssed" and "BERT Sentences Processed".
```

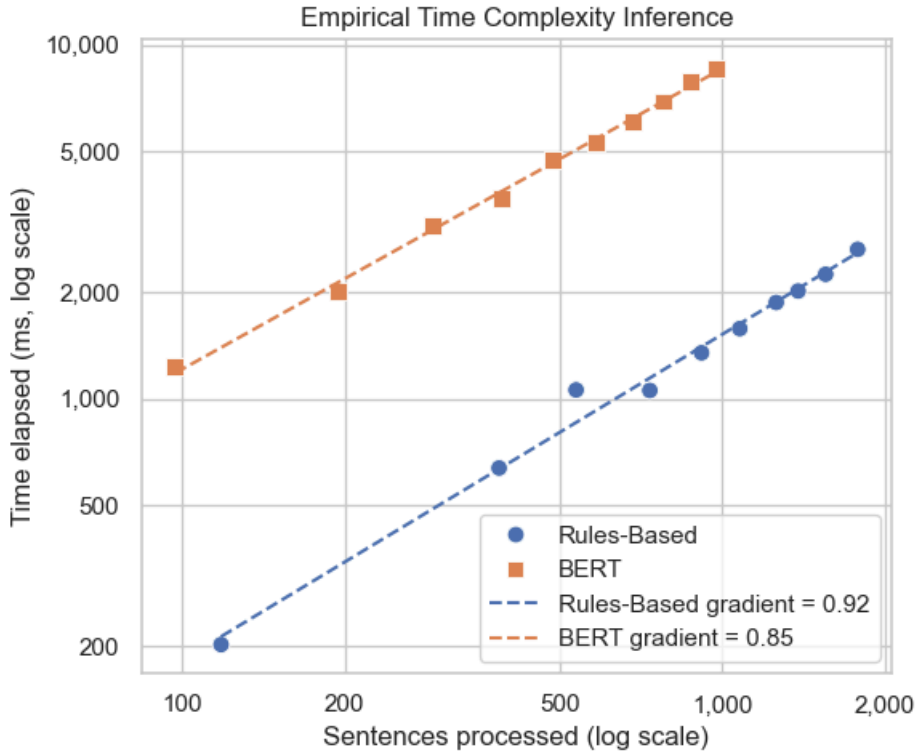

Figure 11. Empirical inference time scaling showing log time elapsed plotted against log number of sentences processed, fit to a polynomial using least squares.

	target	token	attention	rank
0	speed	speed	0.089995	1
1	speed	<TGT>	0.083890	2
2	speed	terrible	0.083641	3
3	speed	[CLS]	0.082779	4
4	speed	</TGT>	0.078352	5
5	speed	the	0.075111	6
6	speed	is	0.071944	7
7	speed	but	0.050668	8
8	speed	,	0.034715	9
9	speed	good	0.028852	10
10	speed	looks	0.024685	11
11	speed	[SEP]	0.017283	12
12	speed	.	0.007714	13

Table 23. Displays the attention scores attributed to the target aspect "speed" and other words in the sentence. Note the closest potential opinion word "terrible" has an attention score much closer to the opinion aspect's than the next possible opinion word, "good", explaining the final negative sentiment decision.

	target	token	attention	rank
0	looks	[CLS]	0.175113	1
1	looks	looks	0.145695	2
2	looks	</TGT>	0.137703	3
3	looks	<TGT>	0.134438	4
4	looks	good	0.130203	5
5	looks	,	0.104367	6
6	looks	the	0.077894	7
7	looks	is	0.063109	8
8	looks	but	0.051036	9
9	looks	terrible	0.028009	10
10	looks	speed	0.026407	11
11	looks	[SEP]	0.011570	12
12	looks	.	0.006993	13

Table 24. Displays the attention scores attributed to the target aspect "looks" and other words in the sentence. Note the closest potential opinion word "good" has an attention score much closer to the opinion aspect's than the next possible opinion word, "terrible", explaining the final positive sentiment decision.

	sentence	target	pred_id	pred_label
0	Looks good, but the speed is terrible.	speed	0	NEG
1	Looks good, but the speed is terrible.	looks	1	POS

Table 25. Demonstrating the BERT model's ability to attribute sentiment with different polarities to multiple aspects within the same sentence.

Whole System Evaluation

The whole system is evaluated by flattening the hierarchy into composite labels of the form $\text{product} \times \text{aspect} \times \text{sentiment}$. The metrics $\text{Precision}^{\text{macro}}$, $\text{Recall}^{\text{macro}}$ and F_1^{macro} are then calculated on these composites. For each composite, per-class metrics are computed and averaged. A limitation of this setup is that it does not verify whether the specific reviews contributing to each composite match between ground truth and predictions, only that their aggregated counts do.

This requires recalculating BERT subsystem derived sentiment using aspects extracted by the first subsystem. The rules-based subsystem derived sentiment must also be recalculated, since it sometimes relies on extracted aspects to resolve sentiment tie-breaks.

The BERT-based opinion miner marginally outperforms the rules-based system (Table 24-25). However, performance remains poor overall: the BERT variant achieves $\text{Precision}^{\text{macro}} = 21.5\%$, $\text{Recall}^{\text{macro}} = 19.0\%$, and $F_1^{\text{macro}} = 19.5\%$.

Considering the isolated subsystem analysis, this opinion miner's main weakness is the aspect extraction subsystem. Whole-system results are markedly worse than aspect extraction alone for two reasons. Firstly, evaluation is per review rather than per sentence. This evaluation does not de-duplicate aspects per review, so a spurious aspect repeated across sentences in the same review can inflate false positives and reduce Precision. Secondly, sentiment misclassifications from BERT cause additional false negatives, reducing Recall.

In [45]:
```
"""
Prepare whole system evaluation data structures.
"""
def build_sentence_index_map(crf_review_index):
    sent_index_map = []
    prev_review = None
    cur_sentence_index = -1
    for i, review_index in enumerate(crf_review_index):
```

```
        if review_index != prev_review:
            cur_sentence_index = 0
            prev_review = review_index
        else:
            cur_sentence_index += 1
        sent_index_map.append((review_index, cur_sentence_index))
    return sent_index_map

def build_bert_test_rows_whole_system(predicted_aspects):
    review_sentence_map = build_sentence_index_map(crf_review_index)

    rows = []
    for i, aspects_obj in enumerate(predicted_aspects):
        review_index, sentence_index = review_sentence_map[i]
        review = test_reviews[review_index]
        sentence = review["sentences"][sentence_index]

        preds = aspects_obj.get("pred", []) or []
        for aspect in preds:
            row = {
                "product_id": review["product"],
                "review_index": review_index,
                "sentence": sentence["text"],
                "target": aspect,
            }

            annotation_label = None
            norm_aspect = aspect.lower().strip()
            for annotation in sentence.get("annotations", []):
                if isinstance(annotation, dict):
                    aspect = (annotation.get("feature") or annotation.get("aspect") or annotati
                    if aspect and aspect.lower().strip() == norm_aspect:
                        s = annotation.get("sentiment")
                        if s is not None:
                            s_int = int(s)
                            annotation_label = "POS" if s_int >= 1 else "NEG"
                        break

            if annotation_label is not None:
                row["label"] = annotation_label

            rows.append(row)
    return rows

def build_bert_pred_map(test_rows_whole_system, y_pred_whole_system):
    ID2LABEL = {0: "NEG", 1: "POS"}
    product_review_map_bert = {}
    for row, yhat in zip(test_rows_whole_system, y_pred_whole_system):
        target = row.get("target")
        if not target:
            continue

        product = row["product_id"]
        aspect  = target.lower().strip()
        label   = ID2LABEL.get(int(yhat), str(int(yhat)))

        d = product_review_map_bert.setdefault(product, {}).setdefault(aspect, {})
        d.setdefault(label, set()).add(row["review_index"])

    return product_review_map_bert

def label_from_score(score: int) -> str:
    if score > 0:  return "POS"
    if score < 0:  return "NEG"
    return "NEU"

def build_rules_pred_map(test_review_results_whole_system, test_reviews):
    pred_map = {}
    for review_index, (review, review_results) in enumerate(zip(test_reviews, test_review_resul
        product = review["product"]
        review_key = review_index

        for entry in review_results:
            aspects = entry.get("pred_aspects") or []
            if not aspects:
```

```
                continue

            label = label_from_score(int(entry.get("sentiment", 0)))

            if label == "NEU":
                continue

            for aspect in aspects:
                a = aspect.lower().strip()
                d = pred_map.setdefault(product, {}).setdefault(a, {})
                d.setdefault(label, set()).add(review_key)
    return pred_map

def build_ground_truth_map_with_sentiment(test_reviews_whole_system_eval):
    gt_map = {}
    for review_index, review in enumerate(test_reviews_whole_system_eval):
        product = review["product"]

        for sentence in review.get("sentences", []):
            for annotation in sentence.get("annotations", []):
                aspect = annotation.get("feature")
                if not aspect:
                    continue

                aspect = aspect.lower().strip()
                s_raw = annotation.get("sentiment")
                s_int = int(s_raw)
                label = "POS" if s_int >= 1 else "NEG"

                d = gt_map.setdefault(product, {}).setdefault(aspect, {})
                d.setdefault(label, set()).add(review_index)
    return gt_map

def flatten_map(m):
    out = {}
    for product, aspect_map in m.items():
        for aspect, inner in aspect_map.items():
            for sentence_label, reviews in inner.items():
                out[(product, aspect, sentence_label)] = set(reviews)
    return out

def whole_system_macro_metrics(pred_map, gt_map):
    pred_units = flatten_map(pred_map)
    gt_units_sent = flatten_map(gt_map)

    use_pred = pred_units
    use_gt   = gt_units_sent
    granularity = "product x aspect x sentiment"
    class_keys = sorted(set(use_gt.keys()) | set(use_pred.keys()))

    rows = []
    for key in class_keys:
        P = use_pred.get(key, set())
        G = use_gt.get(key, set())
        tp = len(P & G)
        fp = len(P - G)
        fn = len(G - P)
        precision = get_precision(tp, fp)
        recall  = get_recall(tp, fn)
        f1   = get_f1(precision, recall)
        rows.append({
            "precision": precision,
            "recall": recall,
            "f1": f1
        })

    per_class_df = pd.DataFrame(rows)
    macro_precision = per_class_df["precision"].mean() if len(per_class_df) else np.nan
    macro_recall = per_class_df["recall"].mean() if len(per_class_df) else np.nan
    macro_f1 = per_class_df["f1"].mean() if len(per_class_df) else np.nan

    summary_df = pd.DataFrame([{
        "Key": granularity,
        "Averaged over": "Union of ground truth and predicted keys",
        "Macro Precision": round(macro_precision * 100, 2),
```

```
        "Macro Recall":    round(macro_recall * 100, 2),
        "Macro F1":        round(macro_f1 * 100, 2),
        "Num classes": len(class_keys)
    }])

    return summary_df, per_class_df

# BERT
test_rows_whole_system = build_bert_test_rows_whole_system(test_reviews_predicted_aspects)
test_ds_whole_system = ABSADataset(test_rows_whole_system, tokenizer)
pred_whole_system = trainer.predict(test_ds_whole_system)
y_pred_whole_system = pred_whole_system.predictions.argmax(axis=-1)

# Rules based
test_review_results_whole_system = document_orientation(test_reviews, result["seed_map"], test_

# Build review maps
product_review_map_rules_sent = build_rules_pred_map(test_review_results_whole_system, test_rev
product_review_map_ground_truth = build_ground_truth_map_with_sentiment(test_reviews)
product_review_map_bert = build_bert_pred_map(test_rows_whole_system, y_pred_whole_system)

summary_aspect, per_aspect = whole_system_macro_metrics(product_review_map_bert, product_review
display(summary_aspect)
FigureUtilities.table_caption("bert_final", "Whole system results using CRF for aspect extracti

summary, per_class = whole_system_macro_metrics(product_review_map_rules_sent, product_review_m
display(summary)
FigureUtilities.table_caption("rules_final", "Whole system results using CRF for aspect extract
```

```
/var/folders/wz/zn90lgl57r349v4f7djn4kxm0000gn/T/ipykernel_24561/1797536905.py:46: UserWarning:
[W036] The component 'matcher' does not have any patterns defined.
  matches = matcher(sent_doc)
```

	Key	Averaged over	Macro Precision	Macro Recall	Macro F1	Num classes
0	product x aspect x sentiment	Union of ground truth and predicted keys	21.54	18.98	19.5	810

Table 26. Whole system results using CRF for aspect extraction and BERT for sentiment analysis.

	Key	Averaged over	Macro Precision	Macro Recall	Macro F1	Num classes
0	product x aspect x sentiment	Union of ground truth and predicted keys	18.48	15.94	16.53	821

Table 27. Whole system results using CRF for aspect extraction and the rules-based algorithm for sentiment analysis.

Whole System Review Summary Examples

The below code outputs example summaries produced by the most performant variant of this opinion miner, the CRF aspect extraction subsystem paired with the BERT sentiment analysis subsystem.

In [44]:
```
"""
Produce example summaries demonstrating whole system operation.
"""
def bert_results_to_df(test_rows, y_pred, test_reviews, ID2LABEL):
    review_text_map = {}
    review_id_map = {}
    for review_index, review in enumerate(test_reviews):
        review_text_map[review_index] = " ".join(s["text"] for s in review["sentences"])
        review_id_map[review_index] = review.get("review_id", str(review_index))

    rows = []
    for row, pred in zip(test_rows, y_pred):
        aspect = row.get("target")
        if not aspect:
            continue

        review_index = int(row["review_index"])
        product = row["product_id"]
        label = ID2LABEL.get(int(pred), str(int(pred)))
```

```
        review_val = review_text_map.get(review_index, "")

        rows.append({
            "Product": product,
            "Aspect": aspect,
            "Sentiment": label,
            "Review": review_val,
        })

    df = (pd.DataFrame(rows, columns=["Product", "Aspect", "Sentiment", "Review"])
            .drop_duplicates(["Product", "Aspect", "Sentiment", "Review"])
            .sort_values(["Product", "Aspect"])
            .reset_index(drop=True))

    summary = (pd.crosstab(index=[df["Product"], df["Aspect"]], columns=df["Sentiment"])
                .rename(columns={"POS": "Positive Count", "NEG": "Negative Count"})
                .reindex(columns=["Positive Count", "Negative Count"], fill_value=0)  # ensure
                .reset_index()
               .rename_axis(index=None, columns=None))
    summary = summary[["Product", "Aspect", "Positive Count", "Negative Count"]]

    return df, summary

df_bert_view, df_bert_summary = bert_results_to_df(test_rows_whole_system, y_pred_whole_system,
display(df_bert_view)
FigureUtilities.table_caption("bert_view_df", "CRF x BERT pipeline product × aspect × sentiment

apex_df_bert_summary = df_bert_summary[df_bert_summary["Product"] == "Apex AD2600 Progressive-s
display(apex_df_bert_summary)
FigureUtilities.table_caption(
    "bert_summary_df",
    "CRF x BERT pipeline aggregated positive and negative counts for the Apex DVD Player produc
)

norton_df_bert_summary = df_bert_summary[df_bert_summary["Product"] == "norton"]
display(norton_df_bert_summary)
FigureUtilities.table_caption(
    "bert_summary_df",
    "CRF x BERT pipeline aggregated positive and negative counts for the Norton product."
)
```

	Product	Aspect	Sentiment	Review
0	Apex AD2600 Progressive-scan DVD player	apex dvd	POS	received apex dvd and the picture was great when set up with s connector . the remote is a little hard to understand . my jpeg pictures are viewable (not so sharp and vivid) on my 27 " screen but seem not as clear as when i view them on my 17 " monitor . so far the dvd works so i hope it does n't break down like the reviews i 've read . prior to christmas , it seems the reviews were very good but after that it seems the reviews went bad ! i would n't gotten this dvd based on the reviews i read after christmas .
1	Apex AD2600 Progressive-scan DVD player	apex product	POS	this is the third apex product i 've bought and they continue to impress for the price . it goes nicely with my apex 27 " in my bedroom . the build quality feels solid , it does n't shake or whine while playing discs , and the picture and sound is top notch (both dts and dd5.1 sound good) . i paid twenty dollars more than the current price , and even then it was a steal .
2	Apex AD2600 Progressive-scan DVD player	cheep media	POS	i wanted a dvd player that had basic features and would be able to play dvd - r format disc 's that i had made myself . when i 1st tried it i found that it had no problem playing the more expensive dvd - r media disc 's that other players would play but to my surprise was able to play cheep media that i had burned that i could n't get to work before in anything but the dvd burner itself that i had made the disc with . easy to use , small form factor , and looks good to boot . very happy !
3	Apex AD2600 Progressive-scan DVD player	customer service	NEG	i bought this player as a christmas present for my wife . 3/4 way through the first disk we played on it (naturally on 31 days after purchase) the dvd player froze . now it will not load any disk or just says that there is no disk . have tried to contact apex via e-mail (customerservice@apexdigital.com) , but the e-mail address is not valid . have tried calling (866-4apexgo) , but can 't get through . this product sucks , the customer service from apex sucks . i should n't have been cheap , should have bought a toshiba .
4	Apex AD2600 Progressive-scan DVD player	customer service	NEG	i have had this player for 3 months and have been able to see a total of 6 dvd 's , after the frustration and aggrivation of getting the player to do it 's job . i have tried many times to get a hold of apex who does n't seem to answer their emails and telephone number is constantly busy as (i believe) the one lonely customer service rep is probably extremely overworked . as i emailed to the the customer service department , the only thing this player has done consistantly is use enought electricity to keep the red led burning . do yourself a favor and save your money to get something of better quality , or at least works .
...	...	...	...	...
466	norton	rebate	NEG	Product is OK; however, despite the fact that I followed the rebate instructions explicitly, I am being denied the rebate because the processing agency incorrectly claims I did not send in proof of ownership of a previous product. Not much recourse for the consumer in these cases.
467	norton	software	POS	I didn't have any major problems installing this software. The only problem I had was a few components didn't quite load correctly upon startup, but a reboot fixed the problem. I had NIS 2003 and was very happy with it; this new version really doesn't seem to offer a lot of exciting new features, seems merely the same as 2003 dressed-up to look like a new version. Anyhow, this software is working fine and no problems since I installed it two weeks ago. I give it 4-stars only because the installation process - or rather problems that may surface during the installation process - can be frustrating to the average computer user. If you experience problems with installation, you can visit symantec's website, click the support tab, click the 'home or home office support' and there is an automated support assistant that can scan the Norton program files on your computer and it will tell you what is not working right and how to fix it. Can't get easier than that! Avoid like the plague. I already knew something was wrong, because much of the Spam I've seen in the past six months was from resellers of Norton products. For Norton to encourage Spammers by letting them sell their products is a very bad sign.
468	norton	software	NEG	I have read the installation instructions for both NIS 2004 and NAV 2004 prior to installation, but still ended up with the same result...junk software. Why is it that I can install any other type of software and it installs and works properly? But if I installed either one of these Norton products, neither works after installation? It can not be the computer or the owner, since I purchased McAfee Anti-Virus 8 and it installs and works fine with no problems. I have used Norton for the past 5 years and for the last 2 years, the software has gotten more and more disgraceful. I am glad that I do not work for Norton.

	Product	Aspect	Sentiment	Review
469	norton	spam	POS	Yeah - this program protects your computer all right - by locking you off the internet. Anti-Spam is Good The latest of Internet Security has antispam function that is helpful because I has received a mountain of those these days. Other than that it is all the same except the new updates. But that is good for me because Symantec softwares are a bit hard to master for me who are not savvy on computers. So far I haven't had so much complaints except that I still haven't mastered how to scandisk when using Internet Security.
470	norton	symantec	NEG	I've had the last 3 versions of NIS. All in all it hasn't been a bad product. However, Symantec has taken a significant turn for the worse with this release.

471 rows × 4 columns

Table 28. CRF x BERT pipeline product × aspect × sentiment × review, where POS and NEG indicate positive and negative, respectively.

	Product	Aspect	Positive Count	Negative Count
0	Apex AD2600 Progressive-scan DVD player	apex dvd	1	0
1	Apex AD2600 Progressive-scan DVD player	apex product	1	0
2	Apex AD2600 Progressive-scan DVD player	cheep media	1	0
3	Apex AD2600 Progressive-scan DVD player	customer service	0	2
4	Apex AD2600 Progressive-scan DVD player	dvd	0	1
5	Apex AD2600 Progressive-scan DVD player	dvd - r media disc	1	0
6	Apex AD2600 Progressive-scan DVD player	dvd player	3	1
7	Apex AD2600 Progressive-scan DVD player	dvd-player	1	0
8	Apex AD2600 Progressive-scan DVD player	picture	2	1
9	Apex AD2600 Progressive-scan DVD player	play	0	1
10	Apex AD2600 Progressive-scan DVD player	player	1	0
11	Apex AD2600 Progressive-scan DVD player	price	3	0
12	Apex AD2600 Progressive-scan DVD player	product	0	1
13	Apex AD2600 Progressive-scan DVD player	quality	1	0
14	Apex AD2600 Progressive-scan DVD player	rebate	0	1
15	Apex AD2600 Progressive-scan DVD player	remote	0	2
16	Apex AD2600 Progressive-scan DVD player	sound	1	0
17	Apex AD2600 Progressive-scan DVD player	support	0	1
18	Apex AD2600 Progressive-scan DVD player	universal remotes	0	1
19	Apex AD2600 Progressive-scan DVD player	use	1	0

Table 29. CRF x BERT pipeline aggregated positive and negative counts for the Apex DVD Player product.

	Product	Aspect	Positive Count	Negative Count
317	norton	antispam	1	0
318	norton	antivirus	1	0
319	norton	computer security system	1	0
320	norton	dial-up	0	1
321	norton	install	0	1
322	norton	installation	0	1
323	norton	nis 2004	1	0
324	norton	product	3	0
325	norton	rebate	0	1
326	norton	software	1	1
327	norton	spam	1	0
328	norton	symantec	0	1

Table 29. CRF x BERT pipeline aggregated positive and negative counts for the Norton product.

Proposed Further Work

There exist several promising directions for further work to enhance this opinion miner. Based on the literature (Shu, Xu and Liu, 2017), the CRF-based aspect extraction subsystem has a higher performance ceiling than achieved here. Improvements may come from adding additional state feature functions, such as the distance between aspects and their closest noun phrases (Jakob and Gurevych, 2010), and then performing cross-validation to evaluate the marginal contribution of each feature and identify the best configuration. Wei et al. (2021) also proposed "Masked Conditional Random Fields" to efficiently prevent illegal label transitions, which could further boost performance.

Given the strong results of the BERT sentiment subsystem and the flexibility of the ecosystem, extending BERT to also perform aspect extraction is an attractive prospect. This would allow the entire opinion miner to operate within a single unified model, jointly extracting aspects and assigning sentiment.

Finally, the questionable quality of the datasets used in this study has been noted throughout. Training and testing on larger, higher quality datasets would provide a more rigorous evaluation. SemEval offers datasets tailored to aspect-based sentiment analysis, and SemEval Task 12 (SemEval-2015 Task 12: Aspect Based Sentiment Analysis, 2015) includes annotated full reviews for multi-sentence reasoning, neutral sentiment labels to enable three-class classification and a larger corpus that supports more robust training and testing.

Conclusion

In the literature, CRFs are a leading technology for aspect extraction, but in practice it proved difficult to make work effectively. By contrast, the rules-based sentiment analysis subsystem, despite relying on simplifying assumptions about aspect-based sentiment, performed surprisingly well. This shows that sentiment analysis can be achieved with a relatively straightforward algorithm while ignoring many of the subtleties of language. As expected, the BERT-based subsystem was highly performant in isolation, even without hyperparameter tuning, suggesting that optimisation could yield additional gains.

Overall, the opinion miner developed in this study produces poor quantitative metrics, and on this basis it would not be signed off for production use. However, it is built on a principled foundation and, from qualitative inspection of product–aspect–sentiment summaries, often generates agreeable outputs, even when not matching ground truth. With the proposed further work extensions, the system could evolve into a performant opinion miner.

Appendix

Precision (macro)

$$\text{Precision}^{\text{macro}} = \frac{1}{C}\sum_{i=1}^{C}\frac{\text{TP}_i}{\text{TP}_i+\text{FP}_i}$$

Recall (macro)

$$\text{Recall}^{\text{macro}} = \frac{1}{C}\sum_{i=1}^{C}\frac{\text{TP}_i}{\text{TP}_i+\text{FN}_i}$$

F_1 (macro)

$$F_1^{\text{macro}} = \frac{1}{C}\sum_{i=1}^{C}\frac{2\times\text{Precision}_i\text{Recall}_i}{\text{Precision}_i+\text{Recall}_i}$$

Precision (micro)

$$\text{Precision}^{\text{micro}} = \frac{\sum_{i=1}^{C}\text{TP}_i}{\sum_{i=1}^{C}(\text{TP}_i+\text{FP}_i)}$$

Recall (micro)

$$\text{Recall}^{\text{micro}} = \frac{\sum_{i=1}^{C}\text{TP}_i}{\sum_{i=1}^{C}(\text{TP}_i+\text{FN}_i)}$$

F_1 (micro)

$$F_1^{\text{micro}} = \frac{2\times\text{Precision}^{\text{micro}}\times\text{Recall}^{\text{micro}}}{\text{Precision}^{\text{micro}}+\text{Recall}^{\text{micro}}}$$

Coverage

$$\text{Coverage} = \frac{\text{Number of entities predicted}}{\text{Total number of entities}}$$

References

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K., 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). pp.4171–4186.

Géron, A., 2022. Hands-on machine learning with scikit-learn, keras, and TensorFlow. 3rd edn. Sebastopol, CA: O'Reilly Media, Inc.

Hoerl, A.E. and Kennard, R.W., 1970. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics : a journal of statistics for the physical, chemical, and engineering sciences [Online], 12(1), pp.55–67. Available from: https://doi.org/10.1080/00401706.1970.10488634.

Hu, M. and Liu, B., 2004. Mining and summarizing customer reviews. Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp.168–177.

Hugging Face, 2025. WordPiece Tokenization (chapter 6) [Online]. Available from: https://huggingface.co/learn/llm-course/en/chapter6/6.

Jakob, N. and Gurevych, I., 2010. Extracting opinion targets in a single and cross-domain setting with conditional random fields. Proceedings of the 2010 conference on empirical methods in natural language processing. pp.1035–1045.

Julia Silge and David Robinson, 2025. sentiments lexicon dataset (sentiments.csv) from the tidytext package [Online]. Available from: https://raw.githubusercontent.com/juliasilge/tidytext/main/data-raw/sentiments.csv.

Lafferty, J., McCallum, A. and Pereira, F.C., 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Lavergne, T., Cappé, O. and Yvon, F., 2010. Practical very large scale CRFs. Proceedings of the 48th annual meeting of the association for computational linguistics. pp.504–513.

Okazaki, N., 2007. CRFsuite: a fast implementation of conditional random fields (crfs) [Online]. Available from: http://www.chokkan.org/software/crfsuite/.

Princeton, 2010. About WordNet [Online]. Available from: https://wordnet.princeton.edu.

Qiu, G., Liu, B., Bu, J. and Chen, C., 2011. Opinion word expansion and target extraction through double propagation. Computational linguistics, 37(1), pp.9–27.

Russell, S. and Norvig, P., 2022. Artificial Intelligence: A modern approach. Pearson Education Limited.

Sarkar, K., 2016. A CRF based POS tagger for code-mixed Indian social media text. arXiv preprint arXiv:1612.07956.

SemEval-2015 task 12: Aspect based sentiment analysis [Online], 2015. Qatar Computing Research Institute (QCRI). Available from: https://alt.qcri.org/semeval2015/task12/.

Sha, F. and Pereira, F., 2003. Shallow parsing with conditional random fields. Proceedings of the 2003 human language technology conference of the north american chapter of the association for computational linguistics. pp.213–220.

Shu, L., Xu, H. and Liu, B., 2017. Lifelong learning CRF for supervised aspect extraction. arXiv preprint arXiv:1705.00251.

Sutton, C., McCallum, A., and others, 2012. An introduction to conditional random fields. Foundations and Trends® in Machine Learning, 4(4), pp.267–373.

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), 58(1), pp.267–288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems, 30.

Wang, W., Pan, S.J., Dahlmeier, D. and Xiao, X., 2016. Recursive neural conditional random fields for aspect-based sentiment analysis. arXiv preprint arXiv:1603.06679.

Zipf, G.K., 1949. Human Behavior and the Principle of Least Effort. Cambridge, MA: Addison–Wesley.