Practical Exercise — Lesson 3 (Week 5)

Single-Prediction Explainability Audit with Attention / LIME / SHAP

Time: 75–100 minutes (solo or pairs)
Tools: Google Colab, Python, `datasets`, `transformers`, `torch`, `pandas`, `numpy`, `matplotlib`, `lime`, `shap` (optional: `bertviz`)
What you will submit: a Colab notebook + a 1-page "Explanation Card" (PDF)

Learning outcomes

By the end, you will be able to:

1. Generate **local explanations** for a single prediction using **attention**, **LIME**, and **SHAP**.
2. Summarize explanations in **plain language** and **visuals** for a non-technical stakeholder.
3. Run **faithfulness checks** (mask/delete important tokens) to see if explanations predict model behavior.
4. Document **limitations** (e.g., attention ≠ causation; instability; sampling variance) and define next debugging steps.

Setup (5–10 min)

1. Open a fresh Colab (CPU is fine).
2. Install/import:
3. `!pip -q install datasets transformers torch lime shap matplotlib --upgrade`
4. `import os, numpy as np, pandas as pd, torch, matplotlib.pyplot as plt`
5. `from datasets import load_dataset`
6. `from transformers import AutoTokenizer, AutoModelForSequenceClassification, TextClassificationPipeline`
7. `os.environ["TOKENIZERS_PARALLELISM"] = "false"`
8. `SEED = 42`
9. `np.random.seed(SEED); torch.manual_seed(SEED)`
10. **Dataset (pick one):**
    - Quick binary sentiment: `glue/sst2` (validation split).
    - (Optional) Toxicity: `civil_comments` (small sampled subset).
11. **Model:** small BERT-like checkpoint for speed, e.g., `distilbert-base-uncased-finetuned-sst-2-english`.

Part A — Pick one instance and lock the baseline (10–15 min)

1. **Load data & model**
2. `ds = load_dataset("glue", "sst2")["validation"]`
3. `ckpt = "distilbert-base-uncased-finetuned-sst-2-english"`
4. `tok  = AutoTokenizer.from_pretrained(ckpt, use_fast=True)`
5. `mdl  = AutoModelForSequenceClassification.from_pretrained(ckpt, output_attentions=True)`
6. `clf  = TextClassificationPipeline(model=mdl, tokenizer=tok, return_all_scores=True, truncation=True)`
7. **Select a single example** whose prediction is confident (score ≥ 0.8) and **save** the raw text and model score.
8. Record the **baseline prediction** (label + probability). You will use this as the reference for all explanations.

Part B — Attention heat-map (last layer, averaged over heads) (15–20 min)

Purpose: Show which tokens receive the most attention from the **[CLS]** token in the final layer. This is a diagnostic view—not proof of causality.

1. **Run the model with attentions** and decode tokens:
2. `enc = tok(example_text, return_tensors="pt", truncation=True)`
3. `with torch.no_grad():`
4. `    out = mdl(**enc)`
5. `attn = out.attentions[-1].squeeze(0).mean(0)   # [heads, seq, seq] -> avg over heads => [seq, seq]`
6. `cls_to_tok = attn[0].cpu().numpy()             # attention from [CLS] (position 0) to all tokens`
7. `toks = tok.convert_ids_to_tokens(enc["input_ids"][0])`
8. **Normalize and plot** a bar or heat-map over tokens (skip special tokens). Mark top-k tokens.
9. **Write a 2–3 sentence narrative**: "The model places high final-layer attention on **X**, **Y**, **Z**, which plausibly aligns/misaligns with the predicted label because …"
10. **Limitation note (1 line):** "Attention weights are not causal attributions; they show where the model *looks*, not necessarily which features *drove* the score."

Part C — LIME on the same instance (15–20 min)

Purpose: Obtain perturbation-based word importances with a **local linear surrogate**.

1. **Create a prob-function** that returns `predict_proba` for texts:
2. `def predict_proba(texts):`
3. `    outs = clf(texts)`
4. `    # outs: list of [{'label':'NEGATIVE', 'score':...}, {'label':'POSITIVE','score':...}]`
5. `    # reorder to [neg, pos] and return 2-D array`
6. `    arr = []`
7. `    for o in outs:`
8. `        scores = {d["label"].lower(): d["score"] for d in o}`
9. `        arr.append([scores.get("negative",0.0), scores.get("positive",0.0)])`
10. `    return np.array(arr)`
11. **Run LIME**:
12. `from lime.lime_text import LimeTextExplainer`
13. `class_names = ["negative","positive"]`
14. `explainer = LimeTextExplainer(class_names=class_names, random_state=SEED)`
15. `exp = explainer.explain_instance(example_text, predict_proba, num_features=10, labels=[1])  # class 1 = positive`
16. `exp.show_in_notebook(text=example_text)  # or exp.as_list(label=1)`
17. `lime_weights = exp.as_list(label=1)      # list of (token, weight)`
18. **Interpret (2–3 sentences):** Name the top positive and negative contributors and whether they make sense.
19. **Limitation note (1 line):** "LIME can be unstable across seeds/perturbations and may split words; treat weights as directional hints."

Part D — SHAP on the same instance (15–20 min)

Purpose: Estimate Shapley values for tokens. This is slower than LIME—keep background small.

1. **Background texts** (10–20 short sentences):
2. `background = [" ".join(ds[i]["sentence"].split()[:10]) for i in np.random.choice(len(ds), 20, replace=False)]`
3. **Define model wrapper** (maps list[str] → prob for positive):
4. `def f(texts):`
5. `    return predict_proba(texts)[:,1]  # positive class probability`
6. **Explainer & values** (kernel approximator for text):
7. `import shap`
8. `masker = shap.maskers.Text(tokenizer=tok)`
9. `explainer = shap.Explainer(f, masker)`
10. `shap_values = explainer([example_text], max_evals=1500, batch_size=20)  # keep it light`
11. `shap.plots.text(shap_values[0])`
12. **Interpret (2–3 sentences):** Compare top tokens with LIME. Are they consistent?
13. **Limitation note (1 line):** "Kernel SHAP for text relies on sampling; results vary with background and budget."

If SHAP is slow or errors: reduce `max_evals` to 500; if still slow, capture `shap_values.data` and compute top-k by absolute value without plotting, or document as a limitation.

Part E — Faithfulness checks (10–15 min)

Goal: Test whether "important" tokens actually **change the prediction** when perturbed.

1. **Deletion test (AOPC-style).**
   - Rank tokens by importance for each method (attention, LIME, SHAP).
   - Iteratively **mask** the top-k tokens (replace with `[MASK]` or a neutral token) and record the positive class probability after each deletion.
   - Plot **probability vs. #tokens removed** curves (three lines: one per method). A steeper drop = more faithful explanation.

```
def mask_tokens(text, tokens_to_mask):
    words = text.split()
    for i in tokens_to_mask: words[i] = "[MASK]"
    return " ".join(words)
# Build ranked indices per method, then loop k = 1..K
```

7. **Counterfactual tiny edit.**
   - Change one top positive/negative token (e.g., "excellent"→"average"), re-predict, and record Δprobability.
   - Write a **one-liner**: "Replacing *X* with *Y* reduced the positive probability from 0.91→0.64 (-0.27), consistent with the explanations."

Part F — Synthesis & risks (5–10 min)

1. **Convergence table (top-5 tokens).** Create a 3-column table (Attention / LIME / SHAP) with token and signed weight/indicator. Note overlaps.
2. **Short narrative (≤120 words):**
   - What features are the model using?
   - Do methods agree on the **reason** for the prediction?
   - Any **spurious cues** (punctuation, usernames, identity terms)?
   - One **next step** (data fix, prompt/feature tweak, threshold).
3. **Caveats (≤2 bullets):** One limitation per method you used.

What you submit

1. **Notebook (.ipynb)** containing:
   - Baseline prediction; **attention heat-map** or bar chart; **LIME** and **SHAP** visuals/lists.

- **Deletion curve** plot comparing methods; **counterfactual edit** result.
- Convergence table; short narrative; caveats.

2. **Explanation Card (≤1 page, PDF)**
   - **Use-case & instance** (1 sentence, anonymized if needed).
   - **Prediction** (label + prob).
   - **Top tokens** (agreeing across methods, if any).
   - **Faithfulness evidence** (deletion drop, counterfactual Δprob).
   - **Risk note** (e.g., identity term drives decision).
   - **Action** (one concrete next step) + **how you'll re-measure**.

Troubleshooting & tips

- **Tokenization drift:** Use the *same* tokenizer for attention and masking. Skip special tokens (`[CLS]`, `[SEP]`).
- **Long texts:** Truncate to ≤128 tokens for speed; document the truncation.
- **LIME stability:** Set `random_state`; if unstable, run twice and note variance.
- **SHAP speed:** Lower `max_evals` or background size; if still slow, include text output (top tokens) instead of the plot and explain the constraint.
- **Attention ≠ explanation:** Treat it as a **lens**, not ground truth; corroborate with faithfulness checks.

Close

You now have a minimal, repeatable **explainability audit** for a single prediction: three views (attention/LIME/SHAP), one **faithfulness test**, and a **plain-English** narrative that drives a concrete engineering decision.