src/classifier.py

```
\
        """Toxicity classifier with rule-based baseline and optional Hugging Face integration.

        Design:
        - If transformers are installed and HF_MODEL env var set, use a HF pipeline (text-classification
        - Otherwise use a simple rule-based scorer using a small bad-words list and heuristics.
        """
        from typing import Dict, Any, Optional
        import os
        import re

        HF_MODEL = os.getenv("HF_MODEL")

        # Attempt to import transformers lazily
        try:
            from transformers import pipeline
            _TRANSFORMERS_AVAILABLE = True
        except Exception:
            _TRANSFORMERS_AVAILABLE = False

        class ToxicityClassifier:
            def __init__(self, hf_model_name: Optional[str] = None):
                self.hf_model_name = hf_model_name or HF_MODEL
                self.hf_pipeline = None
                if _TRANSFORMERS_AVAILABLE and self.hf_model_name:
                    try:
                        # return_all_scores=True to get label distributions
                        self.hf_pipeline = pipeline("text-classification", model=self.hf_model_name, ret
                    except Exception as e:
                        print(f"[classifier] couldn't load HF model '{self.hf_model_name}': {e}")
                        self.hf_pipeline = None

                # small illustrative list — expand for production use
                self.bad_words = {
                    "insults": ["idiot", "stupid", "dumb", "moron", "trash", "hateyou", "loser"],
                    # avoid storing real hateful slurs in public projects — add domain-appropriate entri
                }

                # compile regex patterns for word boundaries
                self._bad_word_patterns = [re.compile(rf"\b{re.escape(w)}\b", re.IGNORECASE)
                                           for cat in self.bad_words.values() for w in cat]

            def rule_based_score(self, text: str) -> Dict[str, Any]:
                text = text or ""
                hits = []
                for pat in self._bad_word_patterns:
                    for m in pat.finditer(text):
                        hits.append(m.group(0))

                # heuristics: more hits -> higher toxicity; insult words count as 0.2 each
                score = min(1.0, 0.15 * len(hits))
                labels = []
                if hits:
                    labels.append("abusive")

                # stronger heuristics for threats or explicit harmful verbs
                if re.search(r"\bfuck\b|\bkill\b|\bdie\b|\bslap\b", text, re.IGNORECASE):
                    score = max(score, 0.6)
                    labels.append("threatening")

                # allcaps shouting heuristic
                words = text.split()
                if len([w for w in words if w.isupper() and len(w) > 1]) >= 2:
                    score = max(score, 0.45)
                    labels.append("shouting")

                return {
                    "engine": "rule-based",
                    "score": round(score, 3),
            "labels": list(set(labels)),
            "matches": hits,
        }

    def hf_score(self, text: str) -> Optional[Dict[str, Any]]:
        if not self.hf_pipeline:
            return None
        try:
            # pipeline returns list[list[{'label':..., 'score':...}]]
            res = self.hf_pipeline(text)
            # Many HF toxic models use label names like TOXIC, SEVERE_TOXICITY, etc.
            # We'll compute a simple toxicity score as the max score among non-negative labels.
            max_score = 0.0
            toxic_labels = []
            for group in res:
                for item in group:
                    label = item.get("label", "").upper()
                    scr = float(item.get("score", 0.0))
                    # treat labels that look toxic/not-ok as toxic-ish
                    if any(tok in label for tok in ["TOXIC", "ABUSE", "INSULT", "SEVERE", "THREA
                        toxic_labels.append(label)
                        max_score = max(max_score, scr)
                    else:
                        # If model uses POSITIVE/NEGATIVE or LABEL_0 style, treat non-positive a
                        if label not in {"POSITIVE", "NEGATIVE", "LABEL_0", "OK", "NOT_TOXIC", "
                            max_score = max(max_score, scr)

            return {
                "engine": "hf-model",
                "model": self.hf_model_name,
                "score": round(max_score, 3),
                "labels": list(set(toxic_labels)),
                "raw": res,
            }
        except Exception as e:
            print(f"[classifier] HF scoring failed: {e}")
            return None

    def analyze(self, text: str) -> Dict[str, Any]:
        # prefer HF if available
        if self.hf_pipeline:
            hf = self.hf_score(text)
            if hf:
                return hf

        # fallback to rule-based
        return self.rule_based_score(text)

# convenience default instance
_default = ToxicityClassifier()

def analyze_text(text: str) -> Dict[str, Any]:
    return _default.analyze(text)
```