Fowlkes_Gandy_Project_CSE590_51

December 7, 2025

1 CSE-590 Project Playbook - Application of Natural Language Processing Against

Enterprise System Chat Logs will be parsed using class provided lessons. This is to find themes or pattern recognition needed in an enterprise system where AI is not allowed to be incororated.

Scope. Rules > silver labels > supervised classifiers > sequence logic > trends. Everything runs inside the course Docker/Jupyter container. Each logic block lists the Sessionion source used.

Sessionion sources: Session 2 TextProcessing; Session 3 Preprocessing/Feature Extraction; Session 4 Logistic Regression; Session 5 - 6 Probability/Naïve Bayes; Session 7 Vector Space; Session 8 - 9 Vectors background; Session 10 - 13 NN/cost background; Session 15 - 16 LSTM/GRU optional; Session 17 - 18 Siamese optional; Session19 - 20 Seq2Seq/Attention background; Session 21 BLEU/ROUGE; Session 22 Teacher Forcing; Session 23 - 26 Transformers optional.

1.1 1. Prerequisites and Environment (Env)

1. Open a terminal in the project folder with `docker-compose.yaml`, `environment.yml`, and your CSV.
2. Run `docker-compose up -d --build`.
3. Browse to `http://127.0.0.1:8888` and use the configured token.
4. Place `teams_messages.csv` in the mapped project folder.

Container keeps versions and paths consistent for all notebooks.

1.2 2. Imports and Global Setup (Session 2: Text Processing)

[1]:
```
import pandas as pd, numpy as np, re
from datetime import time
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪precision_recall_fscore_support
import matplotlib.pyplot as plt

import nltk; nltk.download('punkt')

TZ = "America/New_York"
STEM = PorterStemmer()
TEXT_COL = "text"       # message text
TS_COL   = "timestamp" # parse-able timestamp
THREAD_COL = "thread_id"
CSV_PATH = "stackexchange_style_devops_chat_regenerated.csv"
```

```
[nltk_data] Downloading package punkt to /home/jovyan/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[2]:
```
import nltk
nltk.download("punkt")
nltk.download("punkt_tab")
```

```
[nltk_data] Downloading package punkt to /home/jovyan/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /home/jovyan/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
```

[2]:
```
True
```

2 3. Normalization Helpers (Session 2: Tokenization, Normalization, Stemming)

[3]:
```
def normalize_rules_text(s: str) -> str:
    s = re.sub(r"https?://\S+|\S+@\S+", " ", str(s).lower())
    s = re.sub(r"[^a-z0-9\s]", " ", s)
    return re.sub(r"\s+", " ", s).strip()

def normalize_model_text(s: str) -> str:
    toks = word_tokenize(str(s).lower())
    toks = [re.sub(r"[^a-z0-9]", "", t) for t in toks]
    toks = [t for t in toks if t]
    toks = [STEM.stem(t) for t in toks]
    return " ".join(toks)

def add_time_flags(df: pd.DataFrame, ts_col: str = TS_COL) -> pd.DataFrame:
    ts = pd.to_datetime(df[ts_col], errors="coerce")
    if ts.dt.tz is None:
        ts = ts.dt.tz_localize(TZ, nonexistent="NaT", ambiguous="NaT")
    else:
        ts = ts.dt.tz_convert(TZ)
    df = df.copy()
    df["_ts"], df["_dow"], df["_hour"] = ts, ts.dt.weekday, ts.dt.hour
    df["_is_after_5"] = df["_hour"] >= 17
    df["_is_fri_after_3"] = (df["_dow"] == 4) & (df["_hour"] >= 15)
    df["_is_weekend"] = df["_dow"].isin([5, 6])
    df["_is_after_hours"] = df["_is_after_5"] | df["_is_fri_after_3"] |␣
 ↪df["_is_weekend"]
    return df
```

2.1 4. Load CSV and Build Base Columns (Session 2)

[4]:
```
TEXT_COL = "text"
TS_COL = "created_at"

raw = pd.read_csv(CSV_PATH).rename(columns={
    "content": TEXT_COL,
    "timestamp": TS_COL
})[[TEXT_COL, TS_COL]]

assert TEXT_COL in raw.columns and TS_COL in raw.columns

if THREAD_COL not in raw.columns:
    THREAD_COL = "thread_id_fallback"
    raw[THREAD_COL] = raw[TEXT_COL].str.extract(r"(#[A-Z]{1,5}-\d+)",␣
 ↪expand=False).fillna("NA")

raw["rules_text"] = raw[TEXT_COL].apply(normalize_rules_text)
raw["model_text"] = raw[TEXT_COL].apply(normalize_model_text)
raw = add_time_flags(raw, ts_col=TS_COL)

raw.head()
```

[4]:
```
                                                text  \
0  We have inconsistent checksum results in the a…
1          Switching runners to isolate the failure.
2          CI runner is stuck in queued state again.
3  Pre-flight validation failed due to missing en…
4    Adding more debug output to the pipeline steps.

                    created_at thread_id_fallback  \
0  2025-11-25T12:16:40.908741Z                 NA
1  2025-10-24T05:17:36.908741Z                 NA
2  2025-11-09T22:25:07.908741Z                 NA
3  2025-11-02T09:57:18.908741Z                 NA
```

```
4  2025-11-21T15:10:24.908741Z                 NA

                                        rules_text  \
0  we have inconsistent checksum results in the a…
1          switching runners to isolate the failure
2          ci runner is stuck in queued state again
3  pre flight validation failed due to missing en…
4     adding more debug output to the pipeline steps

                                        model_text  \
0  we have inconsist checksum result in the artif…
1                  switch runner to isol the failur
2               ci runner is stuck in queu state again
3          preflight valid fail due to miss env var
4          ad more debug output to the pipelin step

                               _ts  _dow  _hour  _is_after_5  _is_fri_after_3  \
0 2025-11-25 07:16:40.908741-05:00     1      7        False            False
1 2025-10-24 01:17:36.908741-04:00     4      1        False            False
2 2025-11-09 17:25:07.908741-05:00     6     17         True            False
3 2025-11-02 04:57:18.908741-05:00     6      4        False            False
4 2025-11-21 10:10:24.908741-05:00     4     10        False            False

   _is_weekend  _is_after_hours
0        False            False
1        False            False
2         True             True
3         True             True
4        False            False
```

2.2 5. Silver Labels: Behavior Rules (Session 2; Session 5 – 6 framing)

[5]:
```python
import re
import pandas as pd

DELIVERY_KW = re.compile(
    r"\b(?:i|we)\s+(?:just\s+)?(?:
 ↪approved|did|completed|finished|pushed|deployed|delivered|fixed|resolved|submitted)\b|"
    r"\b(?:this|it)\s+is\s+ready\b|"
    r"\bready\s+for\s+(?:review|deployment|prod|production|testing)\b|"
    r"\bmoved\s+to\s+(?:the\s+)?(?:correct\s+)?status\b|"
    r"\b(?:recipe|runbook|artifact|package)\s+is\s+ready\b|"
    r"\bmarked\s+as\s+(?:done|complete|resolved)\b|"
    r"\bupdated\s+the\s+(?:ticket|story|task)\b",
    re.I,
)
```

```
APOLOGY_KW = re.compile(r"\bsorry\b.*\blate\b|\blast\s+minute\b", re.I)
READY_KW = re.compile(r"\bready\s+for\s+(?:review|test|prod|production)\b|\b(?:
↪this|it)\s+is\s+ready\b", re.I)
STATUS_KW = re.compile(r"\bmoved\s+to\s+(?:the\s+)?(?:correct\s+)?
↪status\b|\bmarked\s+as\s+(?:done|complete|resolved)\b", re.I)
REWORK_KW = re.compile(r"\b(?:rework|fix(?:ed)?
↪\s+again|again\s+fix|redo|do\s+over|second\s+pass)\b", re.I)
RESUB_KW = re.compile(r"\b(?:re-?submit(?:ted|ting)?|resubmission|updated\s+(?:
↪pr|pull\s+request|ticket|story)|reopen(?:ed)?)\b", re.I)
MISSED_KW = re.compile(r"\b(?:missed|forgot|overlooked|didn'?
↪t\s+include|left\s+out|not\s+covered)\b", re.I)

def label_silver(df: pd.DataFrame) -> pd.DataFrame:
    out = df.copy()

    # Always ensure text is string (avoids errors on NaN)
    txt = out["rules_text"].astype(str)

    out["DeliveryAfterHours"]        = (txt.str.contains(DELIVERY_KW, na=False)␣
  ↪& out["_is_after_hours"]).astype(int)
    out["StatusChangeAfterHours"]    = (txt.str.contains(STATUS_KW,   na=False)␣
  ↪& out["_is_after_hours"]).astype(int)
    out["ReadyForReviewAfterHours"]  = (txt.str.contains(READY_KW,    na=False)␣
  ↪& out["_is_after_hours"]).astype(int)
    out["FridayEarlyCutoffDelivery"] = (txt.str.contains(DELIVERY_KW, na=False)␣
  ↪& (out["_dow"] == 4) & (out["_hour"] >= 15)).astype(int)
    out["WeekendDelivery"]           = (txt.str.contains(DELIVERY_KW, na=False)␣
  ↪& out["_is_weekend"]).astype(int)

    out["ApologyRush"]               = txt.str.contains(APOLOGY_KW, na=False).
  ↪astype(int)
    out["ReworkPhrase"]              = txt.str.contains(REWORK_KW,  na=False).
  ↪astype(int)
    out["ResubmissionPhrase"]        = txt.str.contains(RESUB_KW,   na=False).
  ↪astype(int)
    out["MissedRequirementPhrase"]   = txt.str.contains(MISSED_KW,  na=False).
  ↪astype(int)

    return out

silver = label_silver(raw)
silver.head()
```

[5]:
```
                                              text  \
0  We have inconsistent checksum results in the a…
1          Switching runners to isolate the failure.
```

```
2          CI runner is stuck in queued state again.
3  Pre-flight validation failed due to missing en…
4    Adding more debug output to the pipeline steps.

                   created_at thread_id_fallback  \
0  2025-11-25T12:16:40.908741Z                 NA
1  2025-10-24T05:17:36.908741Z                 NA
2  2025-11-09T22:25:07.908741Z                 NA
3  2025-11-02T09:57:18.908741Z                 NA
4  2025-11-21T15:10:24.908741Z                 NA

                                       rules_text  \
0  we have inconsistent checksum results in the a…
1           switching runners to isolate the failure
2           ci runner is stuck in queued state again
3  pre flight validation failed due to missing en…
4     adding more debug output to the pipeline steps

                                       model_text  \
0  we have inconsist checksum result in the artif…
1                  switch runner to isol the failur
2             ci runner is stuck in queu state again
3           preflight valid fail due to miss env var
4           ad more debug output to the pipelin step

                                 _ts  _dow  _hour  _is_after_5  _is_fri_after_3  \
0 2025-11-25 07:16:40.908741-05:00     1      7        False            False
1 2025-10-24 01:17:36.908741-04:00     4      1        False            False
2 2025-11-09 17:25:07.908741-05:00     6     17         True            False
3 2025-11-02 04:57:18.908741-05:00     6      4        False            False
4 2025-11-21 10:10:24.908741-05:00     4     10        False            False

   …  _is_after_hours  DeliveryAfterHours  StatusChangeAfterHours  \
0  …            False                   0                       0
1  …            False                   0                       0
2  …             True                   0                       0
3  …             True                   0                       0
4  …            False                   0                       0

   ReadyForReviewAfterHours  FridayEarlyCutoffDelivery  WeekendDelivery  \
0                         0                          0                0
1                         0                          0                0
2                         0                          0                0
3                         0                          0                0
4                         0                          0                0

   ApologyRush  ReworkPhrase  ResubmissionPhrase  MissedRequirementPhrase
```

```
0              0              0                   0                          0
1              0              0                   0                          0
2              0              0                   0                          0
3              0              0                   0                          0
4              0              0                   0                          0

[5 rows x 21 columns]
```

2.3 6. Sequence-Aware Behavior: NotRightFirstTime (Session 2; Session 7)

[6]:
```python
seq = silver.sort_values([THREAD_COL, "_ts"]).copy()

seq["delivery_like"] = (
    seq["rules_text"].str.contains(DELIVERY_KW, na=False)
    | seq["StatusChangeAfterHours"].eq(1)
    | seq["ReadyForReviewAfterHours"].eq(1)
)

# convert timestamps
seq["_ts_unix"] = seq["_ts"].astype("int64") // 10**9
WIN_SECONDS = 72 * 3600

# initialize result array
prior_counts = np.zeros(len(seq), dtype=int)

# compute per-thread rolling window counts
for thread, sub in seq.groupby(THREAD_COL):
    idx = sub.index
    ts = sub["_ts_unix"].values
    flags = sub["delivery_like"].values

    # sliding two-pointer window
    left = 0
    for right in range(len(sub)):
        while ts[right] - ts[left] > WIN_SECONDS:
            left += 1
        # count TRUEs in window excluding current
        prior_counts[idx[right]] = flags[left:right].sum()

seq["prior_delivery_like_72h"] = prior_counts

seq["NotRightFirstTime"] = (
    seq["delivery_like"] & (seq["prior_delivery_like_72h"] >= 1)
).astype(int)

# merge back
silver = silver.merge(
    seq[["NotRightFirstTime"]],
    left_index=True,
    right_index=True,
    how="left"
).fillna({"NotRightFirstTime": 0})

silver["NotRightFirstTime"] = silver["NotRightFirstTime"].astype(int)
```

2.4 7. Supervised Classifiers (Session 4; Session 5 – 6)

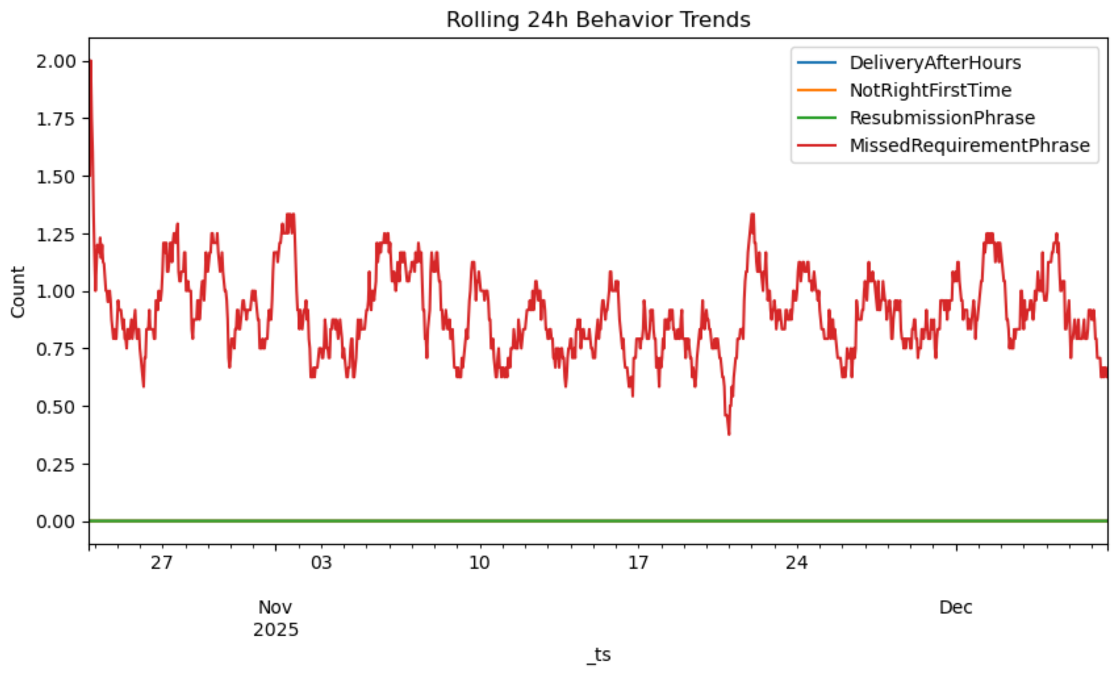
[7]:
```
BEHAVIORS = [
    "DeliveryAfterHours","StatusChangeAfterHours","ReadyForReviewAfterHours",
    "FridayEarlyCutoffDelivery","WeekendDelivery","ApologyRush",
    ␣
 ↪"ReworkPhrase","ResubmissionPhrase","MissedRequirementPhrase","NotRightFirstTime"
]

vec = TfidfVectorizer(ngram_range=(1,2), min_df=5)
X_all = vec.fit_transform(silver["model_text"])

MODELS = {}
for b in BEHAVIORS:
    y = silver[b]
    if y.sum() == 0:
        print(f"[Skip] No positive labels for {b}.")
        continue
    Xtr, Xte, ytr, yte = train_test_split(X_all, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

    lr = LogisticRegression(max_iter=300)
    lr.fit(Xtr, ytr)
    print(f"\n== {b} :: Logistic Regression ==\n", classification_report(yte,␣
 ↪lr.predict(Xte), digits=3))

    nb = MultinomialNB()
    nb.fit(Xtr, ytr)
    print(f"\n== {b} :: Naive Bayes ==\n", classification_report(yte, nb.
 ↪predict(Xte), digits=3))

    MODELS[b] = {"vec": vec, "lr": lr, "nb": nb}
```

```
[Skip] No positive labels for DeliveryAfterHours.
[Skip] No positive labels for StatusChangeAfterHours.
[Skip] No positive labels for ReadyForReviewAfterHours.
[Skip] No positive labels for FridayEarlyCutoffDelivery.
[Skip] No positive labels for WeekendDelivery.
[Skip] No positive labels for ApologyRush.
```

```
[Skip] No positive labels for ReworkPhrase.
[Skip] No positive labels for ResubmissionPhrase.

== MissedRequirementPhrase :: Logistic Regression ==
              precision    recall  f1-score   support

           0      1.000     1.000     1.000     19806
           1      1.000     1.000     1.000       194

    accuracy                          1.000     20000
   macro avg      1.000     1.000     1.000     20000
weighted avg      1.000     1.000     1.000     20000


== MissedRequirementPhrase :: Naive Bayes ==
              precision    recall  f1-score   support

           0      1.000     1.000     1.000     19806
           1      1.000     1.000     1.000       194

    accuracy                          1.000     20000
   macro avg      1.000     1.000     1.000     20000
weighted avg      1.000     1.000     1.000     20000

[Skip] No positive labels for NotRightFirstTime.
```

2.5 8. Trend Charts (Session 3 evaluation framing)

[8]:
```
def behavior_counts(df: pd.DataFrame, cols, freq="1H") -> pd.DataFrame:
    g = df.set_index("_ts")[cols].resample(freq).sum().fillna(0)
    return g

SELECTED =␣
 ↪["DeliveryAfterHours","NotRightFirstTime","ResubmissionPhrase","MissedRequirementPhrase"]
trend = behavior_counts(silver, SELECTED, "1H")

ax = trend.rolling(24, min_periods=1).mean().plot(figsize=(10,5))
ax.set_title("Rolling 24h Behavior Trends")
ax.set_ylabel("Count")
plt.show()
```



2.6 9. Near Real - Time Batch Refresh (Session 2; Session 7)

[ ]:

```
```

[9]:

```
def score_behaviors(df_new: pd.DataFrame) -> pd.DataFrame:
    df_new = df_new.copy()
    df_new["rules_text"] = df_new[TEXT_COL].apply(normalize_rules_text)
    df_new["model_text"] = df_new[TEXT_COL].apply(normalize_model_text)
    df_new = add_time_flags(df_new, ts_col=TS_COL)
    df_new = label_silver(df_new)
    for b, pack in MODELS.items():
        v, lr = pack["vec"], pack["lr"]
        df_new[f"{b}_pred"] = lr.predict(v.transform(df_new["model_text"]))
    return df_new

base = silver.copy()

def refresh(csv_path: str):
    df_all = pd.read_csv(csv_path)
    if len(df_all) <= len(base):
        print("No new rows."); return None
    new = df_all.iloc[len(base):].copy()
    if THREAD_COL not in new.columns:
        new[THREAD_COL] = new[TEXT_COL].str.extract(r"(#[A-Z]{1,5}-\d+)",␣
 ↪expand=False).fillna("NA")
```

```
    new_scored = score_behaviors(new)
    out = pd.concat([base, new_scored], ignore_index=True)
    g = behavior_counts(out, SELECTED, "1H").rolling(24, min_periods=1).mean()
    ax = g.plot(figsize=(10,5))
    ax.set_title("Rolling 24h Behavior Trends - Updated")
    ax.set_ylabel("Count")
    plt.show()
    return out
```

2.7 10. Sentiment (Session 4 – 6)

[10]:
```
POS_KW = re.compile(r"\b(?:thanks|great|appreciate|nice|good)\b", re.I)
NEG_KW = re.compile(r"\b(?:bad|frustrat|angry|upset|issue|broken)\b", re.I)

sent = silver.copy()
sent["sent_rule"] = 0
sent.loc[sent["rules_text"].str.contains(POS_KW, na=False), "sent_rule"] = 1
sent.loc[sent["rules_text"].str.contains(NEG_KW, na=False), "sent_rule"] = -1

# Re-use X_all from TF-IDF of model_text
Xtr, Xte, ytr, yte = train_test_split(X_all, sent["sent_rule"], test_size=0.2,␣
 ↪random_state=42, stratify=sent["sent_rule"])
logreg_sent = LogisticRegression(max_iter=300)
logreg_sent.fit(Xtr, ytr)
print(classification_report(yte, logreg_sent.predict(Xte), digits=3))
```

```
              precision    recall  f1-score   support

          -1      1.000     1.000     1.000       393
           0      1.000     1.000     1.000     19607

    accuracy                          1.000     20000
   macro avg      1.000     1.000     1.000     20000
weighted avg      1.000     1.000     1.000     20000
```

2.8 11. Export Artifacts (Reproducibility)

[11]:
```
silver.to_csv("processed_behaviors.csv", index=False)
silver.sample(1000, random_state=7).to_csv("sample_behaviors_1k.csv",␣
 ↪index=False)
from joblib import dump
for b, pack in MODELS.items():
    dump(pack["lr"], f"model_lr_{b}.joblib")
    dump(pack["vec"], f"vec_{b}.joblib")
print("Artifacts saved.")
```

```
Artifacts saved.
```