CENG 442 – Natural Language Processing

Assignment 1: Azerbaijani Text Preprocessing + Word Embeddings (Domain-Aware)

Deadline: 31.10.2025 23:45 | Group size: up to 4 students

Overview

Goal: Clean five Azerbaijani text datasets for sentiment analysis; keep all three polarities as numeric values (Negative=0.0, Neutral=0.5, Positive=1.0). Produce a simple two-column Excel per dataset. Train Word2Vec and FastText on the cleaned text. Make the pipeline lightly domain-aware (news/social/reviews/general). Keep it simple and reproducible.

1) Deliverables (what to submit)

• Five Excel files (one per dataset) with two columns: cleaned_text (string), sentiment_value (float in {0.0, 0.5, 1.0}).

• One plain-text file corpus_all.txt: all cleaned text as one sentence per line, lowercase, punctuation-free. Prepend a simple domain tag token (e.g., domnews, domsocial, domreviews, domgeneral) to each line.

• Two embedding models trained on the combined corpus: embeddings/word2vec.model and embeddings/fasttext.model.

• A short report as README.md in your GitHub repository (see Section 9).

2) Datasets & Label Mapping

Canonical file names and columns (adapt if your local names differ):

• labeled-sentiment.xlsx — 3-class (columns: text, sentiment)

• test__1_.xlsx — binary (columns: text, label)

• train__3_.xlsx — binary (columns: text, label)

• train-00000-of-00001.xlsx — 3-class (columns: text, labels)

• merged_dataset_CSV__1_.xlsx — binary (columns: text, labels; drop Unnamed: 0 if present)

Sentiment mapping to sentiment_value: Negative→0.0, Neutral→0.5, Positive→1.0.

3) Technologies (what & why)

• Python 3: Main scripting environment.

• pandas: Excel I/O and vectorized preprocessing.

• regex (re): Find/replace for URLs, emails, mentions, numbers, hashtags.

• unicodedata: Azerbaijani-aware casing (İ→i, I→ı) and Unicode normalization.

• ftfy (optional): Fix common encoding glitches.

• gensim: Train Word2Vec and FastText embeddings.

• openpyxl: Excel engine for .xlsx I/O.

Install essentials:

```
pip install pandas gensim openpyxl regex ftfy scikit-learn
```

4) Simple Cleaning Rules (Azerbaijani-aware)

• Lowercase with Azerbaijani rules (replace İ→i, I→ı before lower()).

• Replace URLs→URL, emails→EMAIL, phone numbers→PHONE, @mentions→USER.

• Strip HTML tags/entities; drop # but keep hashtag text (split camelCase hashtags).

• Collapse ≥3 repeated letters to 2 (cooool→coool).

• Replace digits with <NUM> (be consistent).

• Remove standalone punctuation and extra spaces; keep letters: ə, ğ, ı, ö, ü, ç, ş (plus x, q).

• Remove single-letter tokens except o, e.

• Drop exact duplicates and empty rows.

5) Mini Challenges (easy, programming practice)

• Hashtag split: #QarabagIsBack → 'qarabag is back'.

• Emoji mapping: build a tiny dict to map emojis to EMO_POS/EMO_NEG before tokenization.

• Stopword research: compare Azerbaijani with at least one other language (TR/EN/RU), propose 10 candidates (do not remove negations like yox, deyil, heç).

• Negation scope (toggle): mark next 3 tokens with _NEG after a negator (yox, deyil, heç, qətiyyən, yoxdur) and compare nearest neighbors qualitatively.

• Simple deasciify: apply small map (cox→çox, yaxsi→yaxşı) and report how many tokens changed.

6) Domain-Aware Additions (simple & useful)

Use a tiny 4-class domain scheme: news, social, reviews, general. Detect with simple rules; apply a small domain-specific normalization; and tag lines in corpus_all.txt.

```
# --- Domain detection ---
import re
```

```
NEWS_HINTS   = re.compile(r"\b(apa|trend|azertac|reuters|bloomberg|dha|aa)\b", re.I)
SOCIAL_HINTS = re.compile(r"\b(rt)\b|@|#|(?:😂|😍|😊|👍|👎|😡|🙂)")
REV_HINTS    = re.compile(r"\b(azn|manat|qiymət|aldım|ulduz|çox yaxşı|çox pis)\b",
re.I)

def detect_domain(text: str) -> str:
    s = text.lower()
    if NEWS_HINTS.search(s): return "news"
    if SOCIAL_HINTS.search(s): return "social"
    if REV_HINTS.search(s):   return "reviews"
    return "general"

# --- Domain-specific normalization (reviews) ---
PRICE_RE = re.compile(r"\b\d+\s*(azn|manat)\b", re.I)
STARS_RE = re.compile(r"\b([1-5])\s*ulduz\b", re.I)
POS_RATE = re.compile(r"\bçox yaxşı\b")
NEG_RATE = re.compile(r"\bçox pis\b")

def domain_specific_normalize(cleaned: str, domain: str) -> str:
    if domain == "reviews":
        s = PRICE_RE.sub(" <PRICE> ", cleaned)
        s = STARS_RE.sub(lambda m: f" <STARS_{m.group(1)}> ", cleaned)
        s = POS_RATE.sub(" <RATING_POS> ", s)
        s = NEG_RATE.sub(" <RATING_NEG> ", s)
        return " ".join(s.split())
    return cleaned

# --- Domain tag token for corpus (no punctuation) ---
def add_domain_tag(line: str, domain: str) -> str:
    return f"dom{domain} " + line  # e.g., 'domnews', 'domreviews'
```

7) Pipeline Code (two-column outputs + domain-tagged corpus)

```
# -*- coding: utf-8 -*-
import re, html, unicodedata
import pandas as pd
from pathlib import Path

try:
    from ftfy import fix_text
except Exception:
    def fix_text(s): return s

# Azerbaijani-aware lowercase
def lower_az(s: str) -> str:
    if not isinstance(s, str): return ""
    s = unicodedata.normalize("NFC", s)
    s = s.replace("I", "ı").replace("İ", "i")
    s = s.lower().replace("i","i")
    return s

HTML_TAG_RE = re.compile(r"<[^>]+>")
URL_RE      = re.compile(r"(https?://\S+|www\.\S+)", re.IGNORECASE)
EMAIL_RE    = re.compile(r"\b[\w\.-]+@[\w\.-]+\.\w+\b", re.IGNORECASE)
PHONE_RE    = re.compile(r"\+?\d[\d\-\s\(\)]{6,}\d")
USER_RE     = re.compile(r"@\w+")
MULTI_PUNCT = re.compile(r"([!?.,;:])\1{1,}")
MULTI_SPACE = re.compile(r"\s+")
REPEAT_CHARS= re.compile(r"(.)\1{2,}", flags=re.UNICODE)

TOKEN_RE = re.compile(
```

```
    r"[A-Za-zƏəĞğIıİiÖöÜüÇçŞşXxQq]+(?:'[A-Za-zƏəĞğIıİiÖöÜüÇçŞşXxQq]+)?"
    r"|<NUM>|URL|EMAIL|PHONE|USER|EMO_(?:POS|NEG)"
)

EMO_MAP = {"🙂":"EMO_POS","😀":"EMO_POS","😍":"EMO_POS","😊":"EMO_POS","👍":"EMO_POS",
           "☹":"EMO_NEG","🙁":"EMO_NEG","😠":"EMO_NEG","😡":"EMO_NEG","👎":"EMO_NEG"}

SLANG_MAP = {"slm":"salam","tmm":"tamam","sagol":"sağol","cox":"çox","yaxsi":"yaxşı"}
NEGATORS  = {"yox","deyil","heç","qətiyyən","yoxdur"}

# Domain helpers (paste from Section 6)
import re
NEWS_HINTS   = re.compile(r"\b(apa|trend|azertac|reuters|bloomberg|dha|aa)\b", re.I)
SOCIAL_HINTS = re.compile(r"\b(rt)\b|@|#|(?:😂|😍|😊|👍|👎|😡|🙂)")
REV_HINTS    = re.compile(r"\b(azn|manat|qiymət|aldım|ulduz|çox yaxşı|çox pis)\b",
re.I)
PRICE_RE     = re.compile(r"\b\d+\s*(azn|manat)\b", re.I)
STARS_RE     = re.compile(r"\b([1-5])\s*ulduz\b", re.I)
POS_RATE     = re.compile(r"\bçox yaxşı\b")
NEG_RATE     = re.compile(r"\bçox pis\b")

def detect_domain(text: str) -> str:
    s = text.lower()
    if NEWS_HINTS.search(s): return "news"
    if SOCIAL_HINTS.search(s): return "social"
    if REV_HINTS.search(s):  return "reviews"
    return "general"

def domain_specific_normalize(cleaned: str, domain: str) -> str:
    if domain == "reviews":
        s = PRICE_RE.sub(" <PRICE> ", cleaned)
        s = STARS_RE.sub(lambda m: f" <STARS_{m.group(1)}> ", cleaned)
        s = POS_RATE.sub(" <RATING_POS> ", s)
        s = NEG_RATE.sub(" <RATING_NEG> ", s)
        return " ".join(s.split())
    return cleaned

def add_domain_tag(line: str, domain: str) -> str:
    return f"dom{domain} " + line  # no punctuation

def normalize_text_az(s: str, numbers_to_token=True, keep_sentence_punct=False) -> str:
    if not isinstance(s, str): return ""
    # emoji map first
    for emo, tag in EMO_MAP.items():
        s = s.replace(emo, f" {tag} ")
    s = fix_text(s)
    s = html.unescape(s)
    s = HTML_TAG_RE.sub(" ", s)
    s = URL_RE.sub(" URL ", s)
    s = EMAIL_RE.sub(" EMAIL ", s)
    s = PHONE_RE.sub(" PHONE ", s)
    # Hashtag: keep text, split camelCase
    s = re.sub(r"#([A-Za-z0-9_]+)", lambda m: " " + re.sub('([a-z])([A-Z])', r'\1 \2',
m.group(1)) + " ", s)
    s = USER_RE.sub(" USER ", s)
    s = lower_az(s)
    s = MULTI_PUNCT.sub(r"\1", s)
    if numbers_to_token:
        s = re.sub(r"\d+", " <NUM> ", s)
    if keep_sentence_punct:
        s = re.sub(r"[^\w\s<>'əğıöşüçƏĞIİÖŞÜÇxqXQ.!?]", " ", s)
    else:
```

```
        s = re.sub(r"[^\w\s<>'əğıöşüçƏĞIİÖŞÜÇxqXQ]", " ", s)
    s = MULTI_SPACE.sub(" ", s).strip()
    toks = TOKEN_RE.findall(s)
    norm = []
    mark_neg = 0
    for t in toks:
        t = REPEAT_CHARS.sub(r"\1\1", t)
        t = SLANG_MAP.get(t, t)
        if t in NEGATORS:
            norm.append(t); mark_neg = 3; continue
        if mark_neg > 0 and t not in {"URL","EMAIL","PHONE","USER"}:
            norm.append(t + "_NEG"); mark_neg -= 1
        else:
            norm.append(t)
    norm = [t for t in norm if not (len(t) == 1 and t not in {"o","e"})]
    return " ".join(norm).strip()

def map_sentiment_value(v, scheme: str):
    if scheme == "binary":
        try: return 1.0 if int(v) == 1 else 0.0
        except Exception: return None
    s = str(v).strip().lower()
    if s in {"pos","positive","1","müsbət","good","pozitiv"}: return 1.0
    if s in {"neu","neutral","2","neytral"}: return 0.5
    if s in {"neg","negative","0","mənfi","bad","neqativ"}: return 0.0
    return None

def process_file(in_path, text_col, label_col, scheme, out_two_col_path,
remove_stopwords=False):
    df = pd.read_excel(in_path)
    for c in ["Unnamed: 0","index"]:
        if c in df.columns: df = df.drop(columns=[c])
    assert text_col in df.columns and label_col in df.columns, f"Missing columns in
{in_path}"
    # original text kept for domain detection
    df = df.dropna(subset=[text_col])
    df = df[df[text_col].astype(str).str.strip().str.len() > 0]
    df = df.drop_duplicates(subset=[text_col])

    # base clean
    df["cleaned_text"] = df[text_col].astype(str).apply(lambda s: normalize_text_az(s))
    # domain-aware tweak
    df["__domain__"] = df[text_col].astype(str).apply(detect_domain)
    df["cleaned_text"] = df.apply(lambda r:
domain_specific_normalize(r["cleaned_text"], r["__domain__"]), axis=1)

    # optional stopwords (kept minimal, sentiment words preserved)
    if remove_stopwords:
        sw =
set(["və","ilə","amma","ancaq","lakin","ya","həm","ki","bu","bir","o","biz","siz","mən"
,"sən","orada","burada","bütün","hər","artıq","çox","az","ən","də","da","üçün"])
        for keep in ["deyil","yox","heç","qətiyyən","yoxdur"]:
            sw.discard(keep)
        df["cleaned_text"] = df["cleaned_text"].apply(lambda s: " ".join([t for t in
s.split() if t not in sw]))

    # sentiment mapping (0.0 / 0.5 / 1.0)
    df["sentiment_value"] = df[label_col].apply(lambda v: map_sentiment_value(v,
scheme))
    df = df.dropna(subset=["sentiment_value"])
    df["sentiment_value"] = df["sentiment_value"].astype(float)
```

```
    # final two-column output
    out_df = df[["cleaned_text","sentiment_value"]].reset_index(drop=True)
    Path(out_two_col_path).parent.mkdir(parents=True, exist_ok=True)
    out_df.to_excel(out_two_col_path, index=False)
    print(f"Saved: {out_two_col_path} (rows={len(out_df)})")

def build_corpus_txt(input_files, text_cols, out_txt="corpus_all.txt"):
    """Create domain-tagged, lowercase, punctuation-free corpus (one sentence per
line)."""
    lines = []
    for (f, text_col) in zip(input_files, text_cols):
        df = pd.read_excel(f)
        for raw in df[text_col].dropna().astype(str):
            dom = detect_domain(raw)
            s = normalize_text_az(raw, keep_sentence_punct=True)
            parts = re.split(r"[.!?]+", s)
            for p in parts:
                p = p.strip()
                if not p: continue
                p = re.sub(r"[^\w\səğıöşüçƏĞIİÖŞÜÇxqXQ]", " ", p)  # remove punctuation
                p = " ".join(p.split()).lower()
                if p:
                    lines.append(f"dom{dom} " + p)
    with open(out_txt, "w", encoding="utf-8") as w:
        for ln in lines:
            w.write(ln + "\n")
    print(f"Wrote {out_txt} with {len(lines)} lines")

if __name__ == "__main__":
    CFG = [
        ("labeled-sentiment.xlsx",        "text", "sentiment", "tri"),
        ("test__1_.xlsx",                 "text", "label",     "binary"),
        ("train__3_.xlsx",                "text", "label",     "binary"),
        ("train-00000-of-00001.xlsx",     "text", "labels",    "tri"),
        ("merged_dataset_CSV__1_.xlsx",   "text", "labels",    "binary"),
    ]
    # two-column outputs
    for fname, tcol, lcol, scheme in CFG:
        out = f"{Path(fname).stem}_2col.xlsx"
        process_file(fname, tcol, lcol, scheme, out, remove_stopwords=False)
    # combined domain-tagged, punctuation-free corpus
    build_corpus_txt([c[0] for c in CFG], [c[1] for c in CFG],
out_txt="corpus_all.txt")
```

8) Train Word2Vec & FastText (combined cleaned_text)

```
from gensim.models import Word2Vec, FastText
import pandas as pd
from pathlib import Path

files = [
    "labeled-sentiment_2col.xlsx",
    "test__1__2col.xlsx",
    "train__3__2col.xlsx",
    "train-00000-of-00001_2col.xlsx",
    "merged_dataset_CSV__1__2col.xlsx",
]

sentences = []
for f in files:
    df = pd.read_excel(f, usecols=["cleaned_text"])
```

```
    sentences.extend(df["cleaned_text"].astype(str).str.split().tolist())

Path("embeddings").mkdir(exist_ok=True)
w2v = Word2Vec(sentences=sentences, vector_size=300, window=5, min_count=3, sg=1,
negative=10, epochs=10)
w2v.save("embeddings/word2vec.model")
ft  = FastText(sentences=sentences, vector_size=300, window=5, min_count=3, sg=1,
min_n=3, max_n=6, epochs=10)
ft.save("embeddings/fasttext.model")
print("Saved embeddings.")
```

9) Compare Word2Vec vs FastText (simple metrics)

Evaluate coverage, synonym/antonym similarity and nearest neighbors; report per domain if possible.

```
import pandas as pd
from gensim.models import Word2Vec, FastText
import re

w2v = Word2Vec.load("embeddings/word2vec.model")
ft  = FastText.load("embeddings/fasttext.model")

seed_words =
["yaxşı","pis","çox","bahalı","ucuz","mükəmməl","dəhşət","<PRICE>","<RATING_POS>"]
syn_pairs  = [("yaxşı","əla"), ("bahalı","qiymətli"), ("ucuz","sərfəli")]
ant_pairs  = [("yaxşı","pis"), ("bahalı","ucuz")]

def lexical_coverage(model, tokens):
    vocab = model.wv.key_to_index
    return sum(1 for t in tokens if t in vocab) / max(1,len(tokens))

files = [
    "labeled-sentiment_2col.xlsx",
    "test__1__2col.xlsx",
    "train__3__2col.xlsx",
    "train-00000-of-00001_2col.xlsx",
    "merged_dataset_CSV__1__2col.xlsx",
]

def read_tokens(f):
    df = pd.read_excel(f, usecols=["cleaned_text"])
    return [t for row in df["cleaned_text"].astype(str) for t in row.split()]

print("== Lexical coverage (per dataset) ==")
for f in files:
    toks = read_tokens(f)
    cov_w2v = lexical_coverage(w2v, toks)
    cov_ftv = lexical_coverage(ft, toks)  # FT still embeds OOV via subwords
    print(f"{f}: W2V={cov_w2v:.3f}, FT(vocab)={cov_ftv:.3f}")

from numpy import dot
from numpy.linalg import norm

def cos(a,b): return float(dot(a,b)/(norm(a)*norm(b)))

def pair_sim(model, pairs):
    vals = []
    for a,b in pairs:
        try: vals.append(model.wv.similarity(a,b))
```

```
        except KeyError: pass
    return sum(vals)/len(vals) if vals else float('nan')

syn_w2v = pair_sim(w2v, syn_pairs)
syn_ft  = pair_sim(ft,  syn_pairs)
ant_w2v = pair_sim(w2v, ant_pairs)
ant_ft  = pair_sim(ft,  ant_pairs)

print("\n== Similarity (higher better for synonyms; lower better for antonyms) ==")
print(f"Synonyms: W2V={syn_w2v:.3f}, FT={syn_ft:.3f}")
print(f"Antonyms: W2V={ant_w2v:.3f}, FT={ant_ft:.3f}")
print(f"Separation (Syn - Ant): W2V={(syn_w2v - ant_w2v):.3f}, FT={(syn_ft -
ant_ft):.3f}")

def neighbors(model, word, k=5):
    try: return [w for w,_ in model.wv.most_similar(word, topn=k)]
    except KeyError: return []

print("\n== Nearest neighbors (qualitative) ==")
for w in seed_words:
    print(f"  W2V NN for '{w}':", neighbors(w2v, w))
    print(f"  FT  NN for '{w}':", neighbors(ft,  w))

# (Optional) domain drift if you train domain-specific models separately:
# drift(word, model_a, model_b) = 1 - cos(vec_a, vec_b)
```

10) Report (README.md, ≤3 pages)

• 1) Data & Goal (short): datasets, why keep neutral=0.5.

• 2) Preprocessing: rules in 1 paragraph; before→after examples; duplicates/empties removed.

• 3) Mini Challenges: what you implemented and quick observations (1–2 bullets each).

• 4) Domain-Aware: detection rule(s), domain-specific normalization, how you added dom tags to corpus.

• 5) Embeddings: training settings (short table) and results (coverage, Syn/Ant similarities, NN samples; per domain if possible).

• 6) (Optional) Lemmatization: approach/effect if attempted.

• 7) Reproducibility: versions, seeds, machine; how to run.

• 8) Conclusions: which model worked better for your data and why; next steps.

11) Submission (GitHub + AYBUZEM)

All submissions must be via GitHub. AYBUZEM will only receive a text file with your repo link and group info.

• Create a public GitHub repository named: ceng442-assignment1-<groupname>

• Add your code, two-column outputs, corpus_all.txt, and embeddings/ (models). If models are large, provide a download link in the README.

• Your main report must be README.md (Markdown) at the repository root. It should follow Section 10 and render on the repo homepage.

• Ensure all scripts and notebooks are runnable from a clean clone (provide requirements.txt).

• Add group members (max 4) in README and in AYBUZEM text file.

On AYBUZEM, upload ONE text file named: CENG442_Assignment1_Submission.txt with the following content:

```
Repository: https://github.com/<org-or-user>/ceng442-assignment1-<groupname>
Group members:
- <Full Name 1>
- <Full Name 2>
- <Full Name 3>
- <Full Name 4>
```