Supplementary Information

Machine learning for small-data in aquatic environments: a review of challenges, methods, and optimization approaches

Running title: Machine learning for small-data in aquatic environments

Yulin Chen [1, 2, 3, #], Lin Liu [3, #, †], Dawen Gao [1, 2, †]

[1]Centre for Urban Environmental Remediation, Beijing University of Civil Engineering and Architecture, Beijing, 100044, China;

[2]Beijing Energy Conservation & Sustainable Urban and Rural Development Provincial and Ministry Co-construction Collaboration Innovation Center, Beijing University of Civil Engineering and Architecture, Beijing, 100044, China;

[3]State Key Laboratory of Advanced Environmental Technology, Institute of Urban Environment, Chinese Academy of Sciences, Xiamen, 361021, China;

[†] Corresponding author: Dawen Gao, gaodawen@bucea.edu.cn; Lin Liu, lliu@iue.ac.cn

[#] Yulin Chen and Lin Liu contributed equally to this work

Table of Contents

Supplementary Table S1. Search query used in Web of Science to retrieve 9746 publications for co-occurrence network analysis.

Database	Time span	Search query	Subject categories
Web of Science Core Collection	2010–2025	("aquatic environment" OR "water environment" OR river OR lake OR reservoir OR groundwater OR wastewater OR sediment) AND ("machine learning" OR "deep learning" OR "artificial intelligence")	Environmental Sciences; Water Resources; Remote Sensing; Engineering Environmental; Computer Science Interdisciplinary Applications; Computer Science Artificial Intelligence

This table provides the complete Boolean search strategy and subject categories applied in Web of Science for initial literature retrieval.

Supplementary Table S2. Python code for merging Web of Science records (10 batches, 9746 publications) into a single .txt file.

all_9746.py

```
import glob

input_folder = r"D:\Drawing"
output_file = r"D:\Drawing\all_9746.txt"

file_list = sorted(glob.glob(input_folder + r"\*.txt"))

with open(output_file, "w", encoding="utf-8") as outfile:
    for i, fname in enumerate(file_list):
        with open(fname, "r", encoding="utf-8", errors="ignore") as infile:
            lines = infile.readlines()

            if i == 0:
                outfile.writelines(lines)
            else:
                outfile.writelines(lines[1:])

print(f"Merge completed! Output file: {output_file}")
```

This script concatenates multiple export files into one unified dataset for downstream analysis.

Supplementary Table S3. Python code for keyword extraction and co-occurrence network generation.

Co-occurrence Network.py

```
import re
import csv
import argparse
from pathlib import Path
from collections import Counter
from itertools import combinations

def parse_args():
    ap = argparse.ArgumentParser()
    ap.add_argument("--in", dest="infile", required=True, help="WOS TXT path")
    ap.add_argument("--min_occ", type=int, default=5, help="Minimum keyword occurrence t
hreshold (default 5)")
    return ap.parse_args()

NORMALIZE_MAP = {
    r"\bneural\s*network(s)?\b": "neural networks",
    r"\bsupport\s*vector\s*machine(s)?\b": "svm",
    r"\brandom\s*forest(s)?\b": "random forest",
    r"\blong[-\s]*short[-\s]*term\s*memory\b": "lstm",
    r"\bdeep\s*neural\s*network(s)?\b": "dnn",
    r"\bconvolutional\s*neural\s*network(s)?\b": "cnn",
    r"\bwater\s*quality\b": "water quality",
    r"\bclimate\s*change\b": "climate change",
}

STOPWORDS = {
    "model","models","method","methods","approach","approaches","system","systems",
    "performance","classification","regression","prediction","predictions","analysis",
    "dataset","datasets","review","study","studies"
}

def iter_records(lines):
    buf = []
    for ln in lines:
        if ln.startswith("PT "):
            if buf:
                yield buf
                buf = []
        buf.append(ln.rstrip("\n"))
        if ln.startswith("ER"):
            if buf:
                yield buf
                buf = []
    if buf:
        yield buf

def _flush_kw(buf):
    if not buf: return []
    text = " ".join(buf).strip()
    kws = [t.strip(" .;").lower() for t in text.split(";") if t.strip()]
    return kws
```

```
def extract_keywords(rec):
    kws, cur, buf = [], None, []
    target = {"DE"}
    for line in rec:
        if re.match(r"^[A-Z]{2}\s", line[:3]):
            if cur in target and buf:
                kws += _flush_kw(buf)
            cur = line[:2]
            buf = [line[3:].strip()] if cur in target else []
        else:
            if cur in target:
                buf.append(line.strip())
    if cur in target and buf:
        kws += _flush_kw(buf)
    return kws

def normalize_kw(kw: str) -> str:
    s = kw.lower().strip()
    s = re.sub(r"[\"\'\(\)\[\]\{\},/\\]+", " ", s)
    s = re.sub(r"\s+", " ", s).strip()
    for pat, rep in NORMALIZE_MAP.items():
        s = re.sub(pat, rep, s)
    return s

def main():
    args = parse_args()
    infile = Path(args.infile)

    lines = infile.read_text(encoding="utf-8", errors="ignore").splitlines()

    rec_kws = []
    total = 0
    for rec in iter_records(lines):
        total += 1
        kws = extract_keywords(rec)
        if kws:
            norm = {normalize_kw(w) for w in kws}
            norm = {w for w in norm if w and w not in STOPWORDS}
            if norm:
                rec_kws.append(sorted(norm))

    print(f"[INFO] Parsed records: {total}; records with author keywords: {len(rec_kws)}")

    counter = Counter()
    for kwlist in rec_kws:
        counter.update(kwlist)

    vocab = {w for w, c in counter.items() if c >= args.min_occ}
    print(f"[INFO] Total unique author keywords: {len(counter)}; retained after threshold ≥{a
rgs.min_occ}: {len(vocab)}")

    edge_counter = Counter()
    for kwlist in rec_kws:
        filtered = [w for w in kwlist if w in vocab]
        for a, b in combinations(filtered, 2):
            if a != b:
```

```
                    x, y = sorted((a, b))
                    edge_counter[(x, y)] += 1

    print(f"[INFO] Number of edges: {len(edge_counter)}")

    with open("nodes.csv", "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(["id", "label", "occurrences"])
        for kw in sorted(vocab):
            w.writerow([kw, kw.title(), counter[kw]])

    with open("edges.csv", "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(["source", "target", "weight"])
        for (a, b), wt in edge_counter.items():
            if a in vocab and b in vocab:
                w.writerow([a, b, wt])

    print(f"[DONE] Output: {Path('nodes.csv').resolve()}")
    print(f"[DONE] Output: {Path('edges.csv').resolve()}")
    print("→ Open in Gephi: File > Open select edges.csv (undirected), nodes.csv will auto-link; then run layout/clustering/labeling.")

if __name__ == "__main__":
    main()
```

This script normalizes keywords, applies filtering rules, and generates nodes.csv and edges.csv for network visualization in Gephi.

Supplementary Table S4. Structure of the nodes.csv file exported by Co-occurrence Network.py.

Column name	Description	Example value
id	Unique keyword identifier	machine learning
label	Capitalized keyword label	Machine Learning
occurrences	Number of times the keyword appeared	3035

This table illustrates the structure of the nodes.csv file exported through the Python script “Co-occurrence Network.py,” including field names, descriptions, and representative example values.

As the complete dataset is too large to be presented here, only selected examples are provided to indicate the format, while the full dataset is available from the corresponding author upon reasonable request.

Supplementary Table S5. Structure of the edges.csv file exported by Co-occurrence Network.py.

Column name	Description	Example value
source	Source keyword id	machine learning
target	Target keyword id	random forest
weight	Co-occurrence frequency between source and target	147

This table illustrates the structure of the edges.csv file exported through the Python script "Co-occurrence Network.py," including field names, descriptions, and representative example values for keyword co-occurrence relationships.

As the complete dataset is too large to be presented here, only selected examples are provided to indicate the format, while the full dataset is available from the corresponding author upon reasonable request.

Supplementary Table S6. Python code for extracting publication years and classifying records into four aquatic environment themes.

Publications.py

```
import re, sys
from pathlib import Path
from collections import defaultdict, Counter
from typing import List, Dict, Optional, Set
import pandas as pd

INPUT_PATH = Path(sys.argv[1]) if len(sys.argv) > 1 else Path("all_9746.txt")
OUT_MAIN    = Path("ae_ml_year_theme.csv")
OUT_CHECK   = Path("ae_ml_year_theme_with_total.csv")


def read_text(p: Path) -> str:
    if not p.exists():
        raise FileNotFoundError(f"Input file not found: {p.resolve()}")
    try:
        return p.read_text(encoding="utf-8", errors="ignore")
    except Exception:
        return p.read_text(errors="ignore")


def split_records(raw: str) -> List[str]:
    raw = raw.replace("\r\n", "\n").strip()
    parts = re.split(r"\nER\s*\n", raw + "\n")
    parts = [re.sub(r"\nEF\s*\n?$", "\n", p).strip() for p in parts]
    return [p for p in parts if p]


def unwrap_to_dict(block: str) -> Dict[str, List[str]]:
    out: Dict[str, List[str]] = defaultdict(list)
    cur = None
    for line in block.splitlines():
        if not line.strip():
            continue
        m = re.match(r"^([A-Z0-9]{2})\s(.*)$", line)
        if m:
            cur = m.group(1)
            out[cur].append(m.group(2).rstrip())
        else:
            if cur is not None:
                out[cur][-1] += " " + line.strip()
    return out


def norm(s: str) -> str:
    s = s.lower()
    s = s.replace("-", " ")
    s = re.sub(r"\s+", " ", s).strip()
    return s


def get_year(rec: Dict[str, List[str]]) -> Optional[int]:
    for tag in ("PY", "YR"):
        if tag in rec and rec[tag]:
            txt = " ".join(rec[tag])
```

```
                m = re.search(r"\b(19|20)\d{2}\b", txt)
                if m:
                    y = int(m.group(0))
                    if 1900 <= y <= 2100:
                        return y
    return None


def get_blob(rec: Dict[str, List[str]]) -> str:
    parts = []
    for tag in ("DE", "ID", "TI", "AB"):
        if tag in rec and rec[tag]:
            parts.extend(rec[tag])
    return norm("; ".join(parts)) if parts else ""


THEME_PATTERNS = {
    "SurfaceWater": [
        r"\briver(s|ine)?\b", r"\briver\s*basin(s)?\b", r"\briver\s*water\b",
        r"\bstream(s)?\b", r"\bcreek(s)?\b",
        r"\blake(s)?\b", r"\blake\s*water\b", r"\bpond(s)?\b",
        r"\breservoir(s)?\b", r"\bimpoundment(s)?\b",
        r"\bchannel(s)?\b", r"\brunoff\b",
        r"\bstorm\s*water\b", r"\bstormwater\b"
    ],
    "Groundwater": [
        r"\bground\s*water\b", r"\bgroundwater\b",
        r"\baquifer(s)?\b", r"\bwell(s)?\b", r"\bspring(s)?\b",
        r"\bkarst(ic)?\b", r"\bsubsurface\s+water\b", r"\bborehole(s)?\b", r"\bpiezometer(s)?\b"
    ],
    "Wastewater": [
        r"\bwaste\s*water\b", r"\bwastewater\b",
        r"\bsewage\b", r"\bsewer(age)?\b", r"\bwwtp(s)?\b", r"\bstp\b",
        r"\bsewage\s*treatment\s*plant(s)?\b",
        r"\beffluent(s)?\b", r"\binfluent(s)?\b",
        r"\bsewage\s*sludge\b", r"\burban\s*runoff\b", r"\bstorm\s*water\b", r"\bstormwater\b"
    ],
    "Sediment": [
        r"\bsediment(s|ary)?\b", r"\bed\s*sediment(s)?\b",
        r"\bsuspended\s*sediment(s)?\b",
        r"\bsludge\b", r"\benthic\b", r"\bsediment\s*load\b"
    ]
}
THEME_COMPILED = {k: [re.compile(p) for p in v] for k, v in THEME_PATTERNS.items
()}


def classify(blob: str) -> Set[str]:
    labs: Set[str] = set()
    if not blob:
        return labs
    for theme, regs in THEME_COMPILED.items():
        if any(r.search(blob) for r in regs):
            labs.add(theme)
    return labs
```

```
def main():
    raw = read_text(INPUT_PATH)
    records = split_records(raw)
    parsed = [unwrap_to_dict(r) for r in records]

    per_year_total = Counter()
    per_year_theme = defaultdict(lambda: Counter())

    for rec in parsed:
        y = get_year(rec)
        if y is None:
            continue
        per_year_total[y] += 1

        blob = get_blob(rec)
        labs = classify(blob)

        if labs:
            for t in labs:
                per_year_theme[y][t] += 1

    years = sorted(per_year_total.keys())
    rows_main, rows_check = [], []
    for y in years:
        sw = per_year_theme[y]["SurfaceWater"]
        gw = per_year_theme[y]["Groundwater"]
        ww = per_year_theme[y]["Wastewater"]
        sd = per_year_theme[y]["Sediment"]
        total = per_year_total[y]
        other = total - (sw + gw + ww + sd)
        rows_main.append({"Year": y, "SurfaceWater": sw, "Groundwater": gw, "Wastewater":
 ww, "Sediment": sd})
        rows_check.append({
            "Year": y, "SurfaceWater": sw, "Groundwater": gw, "Wastewater": ww, "Sedime
nt": sd,
            "Other": other, "Total": total
        })

    df_main  = pd.DataFrame(rows_main).sort_values("Year")
    df_check = pd.DataFrame(rows_check).sort_values("Year")

    df_main.to_csv(OUT_MAIN, index=False, encoding="utf-8-sig")
    df_check.to_csv(OUT_CHECK, index=False, encoding="utf-8-sig")

    print(f"[OK] Theme summary table: {OUT_MAIN.resolve()}")
    print(f"[OK] Reconciliation table: {OUT_CHECK.resolve()}")
    print("Coverage check:")
    covered = df_check[["SurfaceWater","Groundwater","Wastewater","Sediment"]].sum().sum()
    print(f"  - Total records (with year): {sum(per_year_total.values())}")
    print(f"  - Four themes total:         {covered}")
    print(f"  - Coverage rate:             {covered / max(1,sum(per_year_total.values())):.
2%}")

if __name__ == "__main__":
    main()
```

This script identifies yearly publication counts and categorizes them into SurfaceWater, Gr oundwater, Wastewater, and Sediment themes.

Supplementary Table S7. Annual publication volume by theme extracted from Web of Science records (2010–2025).

Year	Surface Water	Groundwater	Wastewater	Sediment
2010	17	8	3	4
2011	13	4	2	1
2012	22	13	4	6
2013	34	17	4	6
2014	33	13	1	5
2015	60	19	2	11
2016	52	18	8	16
2017	88	56	11	16
2018	160	72	24	30
2019	265	128	25	61
2020	478	228	51	89
2021	727	366	115	158
2022	1079	534	148	172
2023	1287	609	188	200
2024	1584	744	354	299
2025	1452	625	319	253

This table shows the yearly distribution of publications across four aquatic environment themes.

Supplementary Table S8. Web of Science search query with small-data constraints for literature retrieval (2010–2025).

Database	Time span	Search query	Subject categories
Web of Science Core Collection	2010–2025	("aquatic environment" OR "water environment" OR river OR lake OR reservoir OR groundwater OR wastewater OR sediment) AND ("machine learning" OR "deep learning" OR "artificial intelligence") AND ("small data" OR "limited sample*" OR "low sample size" OR "few-shot" OR "data scarcity" OR "scarce data")	Environmental Sciences; Water Resources; Remote Sensing; Engineering Environmental; Computer Science Interdisciplinary Applications; Computer Science Artificial Intelligence

This table documents the refined search strategy including small-data related keyw ords to identify limited-sample studies.