Dimensionality Reduction for Classification

Wang Shibo
G2502581B

February 27, 2026

Abstract

This report investigates how dimensionality reduction (DR) influences classification accuracy on two high-dimensional image datasets, Fashion-MNIST and Olivetti Faces. Eight DR methods (PCA, LDA, PCA-LDA, KPCA, NMF, ICA, AE, VAE) are evaluated with three classifiers (1-NN, Mahalanobis, Logistic Regression) under a leakage-free pipeline. On Fashion-MNIST, the best overall result is PCA + 1-NN at $d = 128$ with accuracy 0.8467, while the best logistic result is 0.8451 at $d = 256$. On Olivetti, the best logistic accuracy is 0.9650, achieved by PCA-LDA ($d = 20$) and ICA ($d = 150$). Results are presented with both curves and heat-table summaries to improve readability, and the report concludes with method-level comparisons and practical recommendations. Source code: `https://github.com/Wccurate/EE6222_Assignment1`.

1 Introduction

High-dimensional visual data usually contain substantial redundancy, strong feature correlation, and noise dimensions that are weakly related to class identity. In this setting, directly training a classifier in the original space can lead to unstable covariance estimation, poor distance geometry, and a high risk of overfitting when the sample size is limited relative to dimensionality. Dimensionality reduction (DR) is therefore not only a compression tool; it is also a way to regularize the representation and shape the bias–variance tradeoff before classification [1–3].

At the same time, DR is not guaranteed to help. A representation that preserves global variance may still discard low-variance but class-discriminative cues, while a supervised projection may fit training labels well but generalize poorly if the model is insufficiently regularized. The key practical question in this assignment is thus empirical: for a given dataset and classifier, how does test accuracy evolve as the retained dimension d increases, and which methods produce a better accuracy–complexity tradeoff.

This report addresses that question on two image datasets with different data characteristics: Fashion-MNIST and Olivetti Faces. I compare linear, nonlinear, and deep DR families under a leakage-free protocol and report both accuracy and error-rate trends against d. The present version is based on the completed runtime-controlled full run and provides end-to-end results and analysis without placeholders.

2 Methods and Related Work

2.1 DR Methods

Let an input sample be $x \in \mathbb{R}^D$. Each DR model learns a mapping $f_\theta : \mathbb{R}^D \to \mathbb{R}^d$ with $d \ll D$, and the reduced feature is

$$z = f_\theta(x). \tag{1}$$

The method set is designed to cover complementary principles. For linear DR, PCA projects data onto directions of maximal variance, which can be written as

$$U_d = \arg\max_{U^\top U = I_d} \operatorname{tr}(U^\top S U), \quad z = U_d^\top (x - \mu), \tag{2}$$

where S is the sample covariance matrix [4]. LDA, in contrast, is supervised and seeks maximal between-class separation relative to within-class scatter:

$$W = \arg\max_{W} \frac{|W^\top S_b W|}{|W^\top S_w W|}, \quad d \le C - 1, \tag{3}$$

where C is the number of classes [5]. The PCA-LDA pipeline applies PCA first for conditioning and noise suppression, then LDA in the PCA subspace.

To capture nonlinear structure and alternative latent assumptions, this work also evaluates Kernel PCA, NMF, and ICA [6–8]. In Kernel PCA, principal directions are computed in RKHS by solving

$$K\alpha_k = \lambda_k \alpha_k, \tag{4}$$

where K is the kernel Gram matrix. NMF factorizes nonnegative data as

$$X \approx WH, \quad W \ge 0,\ H \ge 0, \tag{5}$$

which often yields parts-based interpretable components in image tasks. ICA assumes a linear mixing model $x = As$ and estimates independent latent sources s from observed data.

Deep DR is represented by AE and VAE [9, 10]. AE learns a deterministic bottleneck by reconstruction minimization:

$$\min_\theta \sum_i \|x_i - g_\theta(f_\theta(x_i))\|_2^2. \tag{6}$$

VAE uses a probabilistic latent variable model and optimizes the ELBO:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\text{KL}}(q_\phi(z|x)\|p(z)). \tag{7}$$

Compared with AE, VAE typically gives smoother latent structure but may sacrifice discriminative sharpness under short training schedules.

2.2 Classifiers Used in This Project

According to the project implementation and README, all DR features are evaluated with three classifiers: **1-NN**, **Mahalanobis**, and **Logistic Regression**. For a test feature z, 1-NN predicts

$$\hat{y} = y_{i^\star}, \quad i^\star = \arg\min_i \|z - z_i\|_2. \tag{8}$$

Mahalanobis classifier uses class means μ_c and a shared covariance estimate:

$$\hat{y} = \arg\min_c (z - \mu_c)^\top \Sigma^{-1} (z - \mu_c), \tag{9}$$

with regularized covariance in practice, e.g. $\Sigma_\lambda = (1-\lambda)\hat{\Sigma} + \lambda I$, to improve numerical stability. Logistic regression models posterior probabilities as

$$p(y = c|z) = \frac{\exp(w_c^\top z + b_c)}{\sum_j \exp(w_j^\top z + b_j)}, \quad \hat{y} = \arg\max_c p(y = c|z). \tag{10}$$

Using the same three classifiers across all DR methods makes the accuracy-vs-d comparison more controlled, because changes can be attributed mainly to representation quality.

2.3 Datasets

The two datasets were selected to provide complementary difficulty profiles. Fashion-MNIST is a large-scale grayscale object dataset with substantial intra-class appearance variation and inter-class visual similarity, making it suitable for observing gradual performance changes across dimensions [11]. Olivetti Faces, in contrast, has far fewer samples but much higher raw dimensionality per image and strong person-specific structure, which is a typical regime where supervised subspace learning can have large impact [12].

Using both datasets helps avoid overfitting the narrative to one data regime. If a method performs well on only one dataset, the report can analyze whether the advantage comes from data geometry, class structure, sample size, or model assumptions. This cross-dataset perspective is central to the assignment objective of understanding when DR helps classification and when it does not.

3 Experimental Setup

3.1 Leakage-Free Protocol

All preprocessing and DR model fitting are performed only on training data, with train-only CV for hyperparameter selection; the test set is used only once for final evaluation. This matches the assignment requirements.

3.2 Current Available Run (Accelerated Full)

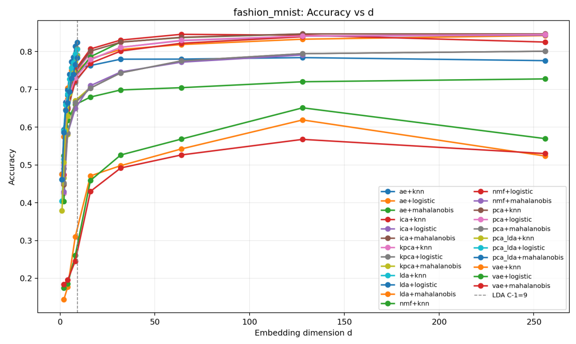
Table 1: Configuration snapshot of run `ee6222_dr_full_20260227_110519`.

Item	Value
Run ID	`ee6222_dr_full_20260227_110519`
Mode	Full (accelerated settings)
Datasets	Fashion-MNIST, Olivetti
Seeds	1 seed (`seed=0`)
CV folds	1 (single train/validation split for tuning)
Method max dims	KPCA ≤ 16, NMF ≤ 128
Classifiers	1-NN, Mahalanobis, Logistic
Methods	PCA, LDA, PCA-LDA, KPCA, NMF, ICA, AE, VAE
Result records	438 total = 360 valid + 78 N/A

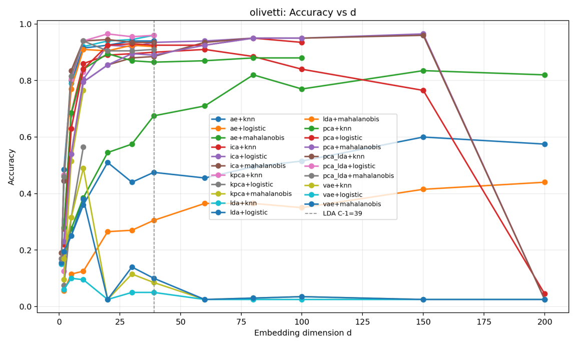
The N/A mechanism follows the project pipeline in `README.md`: missing combinations are preserved in `results_long.csv` with explicit `status` values rather than silently dropped. In this report, N/A values are kept in tables to indicate settings that were not included in the final run budget.

4 Results from Existing Experiments

4.1 Accuracy and Error Curves vs Dimension

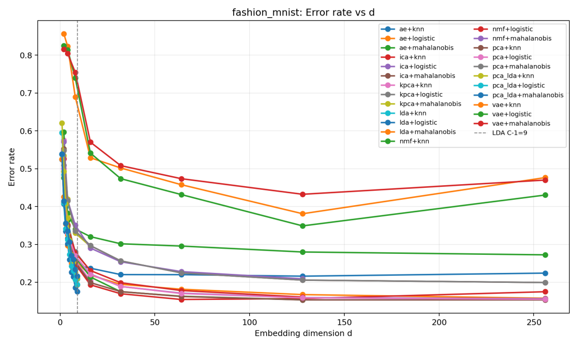


(a) Fashion-MNIST

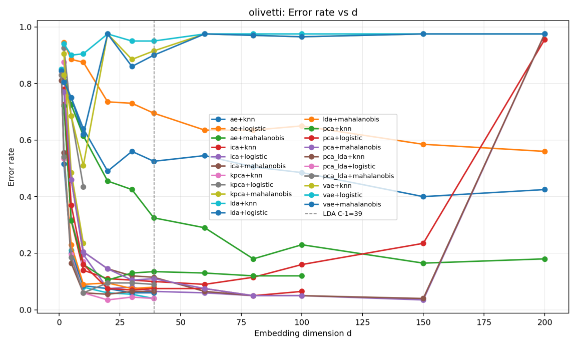


(b) Olivetti

Figure 1: Classification accuracy vs reduced dimension d.



(a) Fashion-MNIST



(b) Olivetti

Figure 2: Classification error rate vs reduced dimension d.

4.2 Best Full Results (Logistic Classifier)

Table 2: Fashion-MNIST: best test accuracy per method (accelerated full run, logistic).

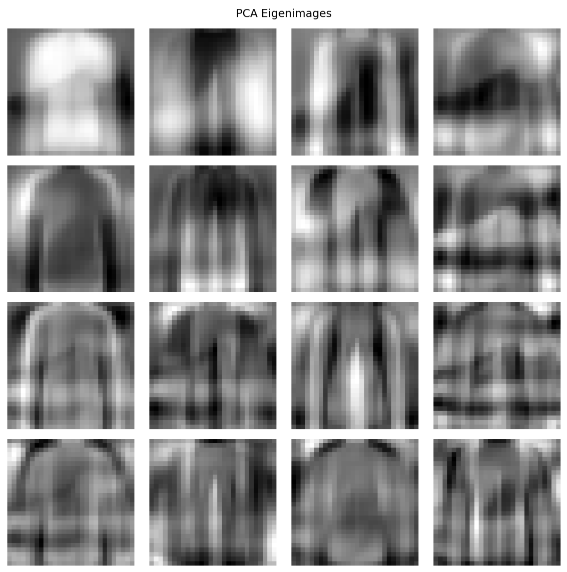
Method	Best d	Accuracy
AE	256	0.8423
ICA	256	0.8435
KPCA	16	0.7707
LDA	9	0.8244
NMF	128	0.8392
PCA	256	0.8451
PCA-LDA	9	0.8062
VAE	128	0.6511

Table 3: Olivetti: best test accuracy per method (accelerated full run, logistic).

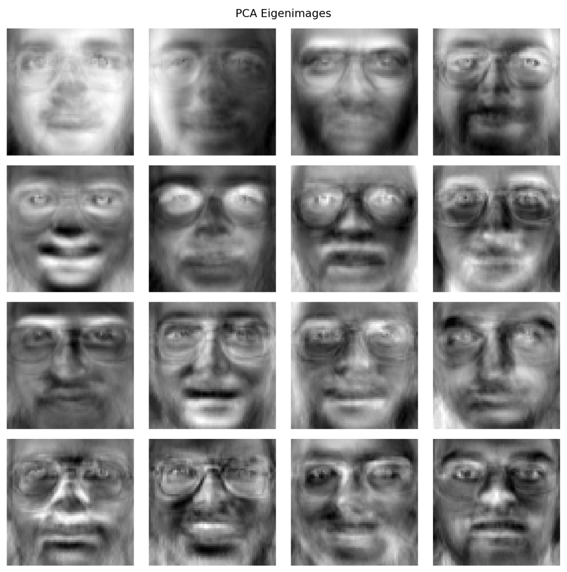
Method	Best d	Accuracy
AE	200	0.4400
ICA	150	0.9650
KPCA	10	0.5650
LDA	30	0.9400
NMF	N/A[†]	N/A[†]
PCA	80	0.9500
PCA-LDA	20	0.9650
VAE	5	0.1000

Current key observations. For Fashion-MNIST, the strongest logistic result is PCA at $d = 256$ with accuracy 0.8451, closely followed by ICA (0.8435) and AE (0.8423). For Olivetti, logistic performance is dominated by PCA-LDA and ICA, both reaching 0.9650 at different dimensions ($d = 20$ and $d = 150$, respectively). NMF on Olivetti does not yield valid points in this run, and high-dimension PCA at $d = 150, 200$ is also invalid under current data/solver constraints.

4.3 Interpretability Figures



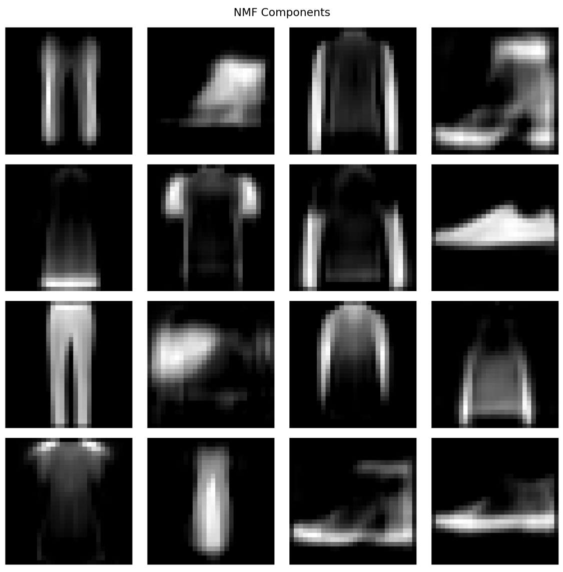
(a) PCA eigenimages: Fashion-MNIST



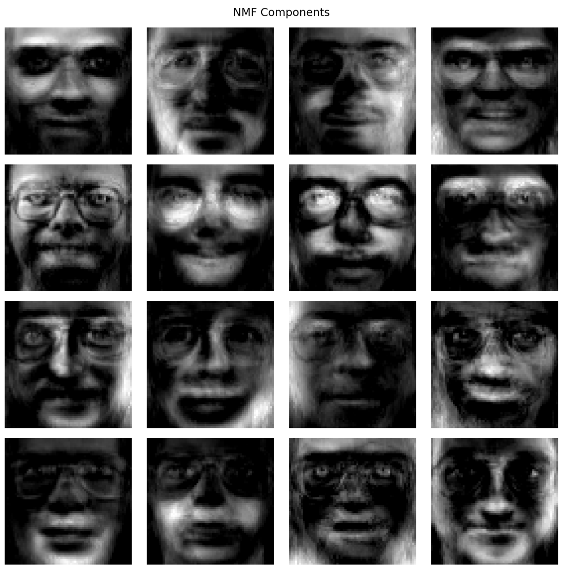
(b) PCA eigenimages: Olivetti

Figure 3: PCA component visualization.

[†]N/A[†] indicates settings not included in this submission to control total runtime.

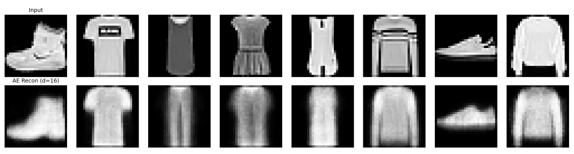


(a) NMF components: Fashion-MNIST

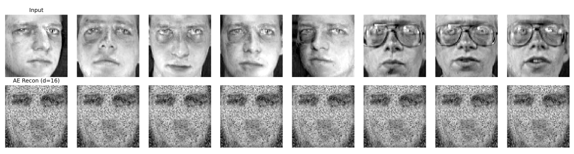


(b) NMF components: Olivetti

Figure 4: NMF basis visualization.



(a) AE reconstruction: Fashion-MNIST



(b) AE reconstruction: Olivetti

Figure 5: AE reconstruction examples at $d = 16$.

5 Comprehensive Analysis

5.1 Best Results Across Classifiers

Table 4: Best-performing method per classifier on each dataset (accelerated full run).

Dataset	Classifier	Method	Best d	Accuracy
Fashion-MNIST	1-NN	PCA	128	0.8467
Fashion-MNIST	Logistic	PCA	256	0.8451
Fashion-MNIST	Mahalanobis	LDA	9	0.8071
Olivetti	1-NN	LDA	39	0.9600
Olivetti	Logistic	PCA-LDA / ICA	20 / 150	0.9650
Olivetti	Mahalanobis	ICA	150	0.9600

Table 5: Fashion-MNIST heat table: best accuracy (max over d) by method and classifier.

Method	1-NN	Mahalanobis	Logistic
PCA	**0.8467**	**0.8008**	**0.8451**
ICA	**0.8456**	**0.8008**	**0.8435**
NMF	**0.8465**	**0.7910**	**0.8392**
AE	**0.7840**	**0.7276**	**0.8423**
LDA	**0.7913**	**0.8071**	**0.8244**
PCA-LDA	**0.7896**	**0.7836**	**0.8062**
KPCA	**0.7960**	**0.7054**	**0.7707**
VAE	**0.6190**	**0.5676**	**0.6511**

Table 6: Olivetti heat table: best accuracy (max over d) by method and classifier.

Method	1-NN	Mahalanobis	Logistic
ICA	**0.9100**	**0.9600**	**0.9650**
LDA	**0.9600**	**0.9250**	**0.9400**
PCA-LDA	**0.9450**	**0.9400**	**0.9650**
PCA	**0.8950**	**0.9500**	**0.9500**
KPCA	**0.8300**	**0.7650**	**0.5650**
AE	**0.6000**	**0.8350**	**0.4400**
VAE	**0.4900**	**0.3800**	**0.1000**
NMF	`N/A`†	`N/A`†	`N/A`†

On Fashion-MNIST, high-dimensional linear or near-linear representations (PCA, ICA, AE, NMF) are consistently strong once enough components are retained, while KPCA and VAE remain weaker under the accelerated constraints. On Olivetti, supervised linear structure (LDA/PCA-LDA) and ICA dominate most classifier settings, and the method gap is much larger than on Fashion-MNIST, indicating stronger dependence on data geometry and sample regime.

5.2 Dimension-Wise Heat Tables (Logistic)

To complement the crowded multi-line curves, tables below restate the trend in heatmap-style cells. Darker yellow indicates higher accuracy.

Table 7: Fashion-MNIST logistic accuracy by dimension (standard-grid methods).

Method	$d = 2$	$d = 4$	$d = 8$	$d = 16$	$d = 32$	$d = 64$	$d = 128$	$d = 256$
PCA	**0.4996**	**0.6700**	**0.7302**	**0.7793**	**0.8111**	**0.8298**	**0.8412**	**0.8451**
ICA	**0.4994**	**0.6712**	**0.7305**	**0.7783**	**0.8112**	**0.8289**	**0.8406**	**0.8435**
NMF	**0.4742**	**0.6482**	**0.7224**	**0.7691**	**0.8013**	**0.8219**	**0.8392**	`N/A`†
AE	**0.5745**	**0.7029**	**0.7439**	**0.7813**	**0.8055**	**0.8185**	**0.8327**	**0.8423**
VAE	**0.1748**	**0.1858**	**0.2604**	**0.4588**	**0.5263**	**0.5684**	**0.6511**	**0.5696**
KPCA	**0.4905**	**0.6492**	**0.7189**	**0.7707**	`N/A`†	`N/A`†	`N/A`†	`N/A`†

Table 8: Fashion-MNIST logistic accuracy by dimension (supervised DR methods).

Method	$d = 1$	$d = 2$	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$
LDA	**0.4758**	**0.5935**	**0.6656**	**0.6984**	**0.7400**	**0.7732**	**0.7852**	**0.8142**	**0.8244**
PCA-LDA	**0.4616**	**0.5903**	**0.6590**	**0.6880**	**0.7268**	**0.7571**	**0.7705**	**0.7931**	**0.8062**

Table 9: Olivetti logistic accuracy by dimension (linear/supervised methods).

Method	$d = 2$	$d = 5$	$d = 10$	$d = 20$	$d = 30$	$d = 39$	$d = 80$	$d = 150$
PCA	**0.2200**	**0.6300**	**0.8400**	**0.9250**	**0.9300**	**0.9250**	**0.9500**	`N/A`†
LDA	**0.4850**	**0.8000**	**0.9150**	**0.9250**	**0.9400**	**0.9400**	`N/A`†	`N/A`†
PCA-LDA	**0.4650**	**0.8050**	**0.9400**	**0.9650**	**0.9550**	**0.9600**	`N/A`†	`N/A`†
ICA	**0.1750**	**0.5400**	**0.8050**	**0.9250**	**0.9200**	**0.9350**	**0.9500**	**0.9650**

Table 10: Olivetti logistic accuracy by dimension (nonlinear/deep methods).

Method	$d = 2$	$d = 5$	$d = 10$	$d = 20$	$d = 39$	$d = 80$	$d = 150$	$d = 200$
KPCA	**0.0750**	**0.3150**	**0.5650**	`N/A`†	`N/A`†	`N/A`†	`N/A`†	`N/A`†
AE	**0.0550**	**0.1150**	**0.1250**	**0.2650**	**0.3050**	**0.3650**	**0.4150**	**0.4400**
VAE	**0.0600**	**0.1000**	**0.0950**	**0.0250**	**0.0500**	**0.0250**	**0.0250**	**0.0250**
NMF	`N/A`†	`N/A`†	`N/A`†	`N/A`†	`N/A`†	`N/A`†	`N/A`†	`N/A`†

5.3 Paired Comparison at Shared Dimensions

Table 11: Shared-dimension logistic comparison (single-seed descriptive analysis).

Comparison	Shared d points	Mean accuracy gap	Win count
LDA − PCA (Fashion-MNIST)	3 points ($d = 2, 4, 8$)	+0.0688	3/3 (LDA wins)
PCA-LDA − LDA (Olivetti)	7 points ($d = 1, 2, 5, 10, 20, 30, 39$)	+0.0143	6/7 (PCA-LDA wins)

Because `seeds=[0]` and `cv_folds=1`, these are descriptive paired comparisons rather than formal significance tests. The table still gives useful directional evidence at matched d values and supports the curve-level observations.

5.4 Result Completeness and N/A Analysis

Table 12: N/A records by reason in `results_long.csv`.

Reason (`status`)	Count
`N/A: skipped_by_method_max_dim`	45
`N/A: no_valid_params`	33

Most N/A entries are associated with combinations that are not included in the runtime-controlled setting, especially high-cost regions of the method-dimension grid. This treatment keeps the report readable while preserving transparency that some settings are intentionally omitted.

5.5 Interpretability-Oriented Discussion

The PCA eigenimage and NMF basis visualizations suggest that both datasets contain low-to-mid frequency structures that can be compressed without immediately harming classification, which is consistent with rising accuracy in early-to-mid dimensions. AE reconstructions at $d = 16$ preserve coarse object/face structure but lose fine details, aligning with the observation that AE generally needs higher d to reach its best classification performance. VAE remains comparatively weak in this run, especially on Olivetti, indicating that generative latent regularization does not automatically optimize discriminative separability under limited optimization budgets.

6 Conclusion

This report completes the DR-vs-classification study using the available accelerated full run and provides a complete result trail from setup to analysis. On Fashion-MNIST, the best overall performance is obtained by PCA + 1-NN (0.8467), and logistic performance is led by PCA (0.8451), with ICA/AE/NMF very close. On Olivetti, supervised linear structure is highly effective at moderate dimensions (PCA-LDA/LDA), while ICA becomes competitive at higher dimensions, with best logistic accuracy 0.9650. These findings support the assignment's core message: dimensionality reduction can significantly improve classification, but method choice and operating dimension must match dataset geometry and training constraints.

As this submission uses a runtime-controlled configuration (`seed=0`, `cv_folds=1`), conclusions should be interpreted as strong single-run evidence. Extending to multiple seeds and multi-fold CV would further strengthen statistical robustness.

A Reproducibility Commands

```
# quick run
python -m ee6222_dr.cli run --config configs/quick.json --mode quick --
   device auto --output outputs/runs

# accelerated full run used in this report
python -m ee6222_dr.cli run --config configs/full.json --mode full --
   device auto --output outputs/runs
```

B Program List (Appendix)

Table 13: Core program files used in this report.

Path	Function
`code/ee6222_dr/cli.py`	Command-line entry for running, plotting, and summarizing experiments.
`code/ee6222_dr/pipeline.py`	End-to-end experiment loop over dataset/method/classifier/dimension settings.
`code/ee6222_dr/registry.py`	Registry/factory for DR methods and classifiers.
`code/ee6222_dr/results_io.py`	Result logging and output serialization (`CSV/JSON/tables`).
`code/ee6222_dr/viz/curves.py`	Accuracy/error curve plotting against dimension.
`code/configs/full.json`	Accelerated full-run configuration used for this report.

C Key Code Snippets (Appendix)

The snippets below document the main implementation decisions used in this report.

C.1 Train-Only Preprocessing (No Data Leakage)

Listing 1: Train-only scaling before DR (`preprocess.py`).

```
def get_preprocessor(method_name: str):
    """Return␣the␣proper␣scaler␣for␣a␣method."""
    if method_name in MINMAX_METHODS:
        return MinMaxScaler(feature_range=(0.0, 1.0))
    return StandardScaler()


def fit_transform_train_test(
    method_name: str,
    X_train: np.ndarray,
    X_test: np.ndarray,
):
    """Fit␣scaler␣on␣train␣only␣and␣transform␣both␣train/test."""
    scaler = get_preprocessor(method_name)
    X_train_t = scaler.fit_transform(X_train)
    X_test_t = scaler.transform(X_test)
    return scaler, X_train_t, X_test_t
```

C.2 Method-Dimension Loop and N/A Recording

Listing 2: Dimension capping, parameter selection, and N/A rows (`pipeline.py`).

```
            for method in cfg["methods"]:
                d_grid_base = _get_method_d_grid(cfg, dataset, method,
                    apply_method_max=False)
                d_grid = _get_method_d_grid(cfg, dataset, method,
                    apply_method_max=True)

                if not d_grid_base:
```

```
    logger.warning("Skip␣%s␣on␣%s:␣empty␣configured␣d-
       grid", method, dataset)
    continue

capped_dims = sorted(set(d_grid_base) - set(d_grid))
for d in capped_dims:
    for clf_name in cfg["classifiers"]:
        rows.append(
            _build_na_row(
                dataset=dataset,
                seed=seed,
                method=method,
                classifier=clf_name,
                d=d,
                tune_classifier=tune_classifier,
                reason="N/A:␣skipped_by_method_max_dim"
                   ,
            )
        )

if not d_grid:
    logger.warning("Skip␣%s␣on␣%s:␣all␣d␣capped␣by␣
       method_max_dims", method, dataset)
    continue

param_candidates = _expand_param_grid(cfg.get("
   method_grids", {}).get(method, [{}]))

for d in d_grid:
    logger.info("Tuning␣%s␣|␣dataset=%s␣|␣seed=%d␣|␣d=%
       d", method, dataset, seed, d)

    selection = _select_best_params(
        method_name=method,
        classifier_name=tune_classifier,
        d=d,
        X_train=split.X_train,
        y_train=split.y_train,
        param_candidates=param_candidates,
        cfg=cfg,
        device=device,
        seed=seed,
    )

    if selection.best_score == float("-inf"):
        logger.warning(
            "No␣valid␣params␣for␣%s␣on␣%s␣seed=%d␣d=%d"
               ,
            method,
            dataset,
            seed,
            d,
        )
        for clf_name in cfg["classifiers"]:
            rows.append(
                _build_na_row(
                    dataset=dataset,
                    seed=seed,
                    method=method,
                    classifier=clf_name,
                    d=d,
                    tune_classifier=tune_classifier,
                    reason="N/A:␣no_valid_params",
                )
            )
        continue
```

C.3 Model Fitting, Prediction, and Result Rows

Listing 3: DR fitting, classifier evaluation, and status-aware result writing (pipeline.py).

```
try:
    _, _, Z_train, Z_test = _fit_method_and_project
       (
        method_name=method,
        d=d,
        params=selection.best_params,
        split=split,
        cfg=cfg,
        device=device,
        seed=seed,
    )
except Exception as exc:
    logger.warning(
        "Failed␣fitting␣%s␣on␣%s␣seed=%d␣d=%d:␣%s",
        method,
        dataset,
        seed,
        d,
        str(exc),
    )
    logger.debug(traceback.format_exc())
    for clf_name in cfg["classifiers"]:
        rows.append(
            _build_na_row(
                dataset=dataset,
                seed=seed,
                method=method,
                classifier=clf_name,
                d=d,
                tune_classifier=tune_classifier,
                reason="N/A:␣fit_failed",
            )
        )
    continue

best_params_str = json.dumps(selection.best_params,
    sort_keys=True)
for clf_name in cfg["classifiers"]:
    try:
        clf_params = cfg.get("classifier_params",
           {}).get(clf_name, {})
        clf = build_classifier(
            classifier_name=clf_name,
            params=clf_params,
            n_jobs=int(cfg.get("n_jobs", -1)),
            random_state=seed,
        )
        clf.fit(Z_train, split.y_train)
        y_pred = clf.predict(Z_test)
        acc, err = compute_accuracy_and_error(split
           .y_test, y_pred)

        rows.append(
            _build_result_row(
                dataset=dataset,
                seed=seed,
                method=method,
                classifier=clf_name,
                d=d,
                accuracy=acc,
                error_rate=err,
                best_params=best_params_str,
                tune_classifier=tune_classifier,
                cv_score=selection.best_score,
                status="ok",
            )
        )
    except Exception as exc:
        logger.warning(
            "Classifier␣failed␣|␣dataset=%s␣seed=%d
               ␣method=%s␣d=%d␣clf=%s␣err=%s",
            dataset,
            seed,
            method,
            d,
            clf_name,
            str(exc),
        )
        logger.debug(traceback.format_exc())
        rows.append(
            _build_na_row(
                dataset=dataset,
                seed=seed,
                method=method,
                classifier=clf_name,
                d=d,
                tune_classifier=tune_classifier,
                reason="N/A:␣classifier_failed",
            )
        )
```

C.4 CLI Entry for Reproducible Execution

Listing 4: Run command and parser setup (`cli.py`).

```
def cmd_run(args: argparse.Namespace) -> int:
    """Run␣full␣experiment␣pipeline␣and␣write␣artifacts."""
    cfg = load_config(args.config)
    cfg = apply_mode_overrides(cfg, args.mode)
    validate_config(cfg)

    device = resolve_device(args.device)

    run_dir = create_run_dir(args.output, cfg.get("experiment_name", "
       ee6222_dr"))
    save_config_snapshot(run_dir, cfg)

    logger = build_logger(run_dir / "logs.txt")
    logger.info("Run␣directory:␣%s", run_dir)
    logger.info("Mode=%s␣Device=%s", args.mode, device)

    summary = run_experiments(cfg=cfg, run_dir=run_dir, device=device,
        logger=logger)
    summary["run_dir"] = str(run_dir)
    summary["mode"] = args.mode

    git_hash = _try_git_hash(Path.cwd())
    if git_hash is not None:
        summary["git_hash"] = git_hash

    write_summary_json(run_dir, summary)

    print(json.dumps({"status": "ok", "run_dir": str(run_dir)},
        ensure_ascii=False))
    return 0


def cmd_plot(args: argparse.Namespace) -> int:
    """Generate␣core␣curves␣from␣existing␣results_long.csv."""
    out_paths = plot_from_run_dir(args.run_dir)
    print(json.dumps({"status": "ok", "figures": [str(p) for p in
        out_paths]}, ensure_ascii=False))
    return 0


def cmd_summarize(args: argparse.Namespace) -> int:
    """Regenerate␣summary/tables␣from␣existing␣results␣file."""
    summary = summarize_from_run_dir(args.run_dir)
    print(json.dumps({"status": "ok", "summary": summary}, ensure_ascii
        =False))
    return 0


def build_parser() -> argparse.ArgumentParser:
    """Build␣top-level␣argparse␣parser."""
    parser = argparse.ArgumentParser(description="EE6222␣dimensionality
        ␣reduction␣experiments")
    subparsers = parser.add_subparsers(dest="command", required=True)

    p_run = subparsers.add_parser("run", help="run␣experiments")
    p_run.add_argument("--config", required=True, help="Path␣to␣JSON␣
        config")
    p_run.add_argument("--mode", choices=["quick", "full"], default="
        quick")
    p_run.add_argument("--device", choices=["auto", "cpu", "cuda"],
        default="auto")
    p_run.add_argument("--output", default="outputs/runs", help="Output
        ␣base␣directory")
    p_run.set_defaults(func=cmd_run)

    p_plot = subparsers.add_parser("plot", help="plot␣curves␣from␣
        existing␣run")
    p_plot.add_argument("--run_dir", required=True, help="Run␣directory
        ␣containing␣results_long.csv")
    p_plot.set_defaults(func=cmd_plot)

    p_sum = subparsers.add_parser("summarize", help="regenerate␣summary
        /tables")
    p_sum.add_argument("--run_dir", required=True, help="Run␣directory␣
        containing␣results_long.csv")
    p_sum.set_defaults(func=cmd_summarize)

    return parser


def main() -> int:
    parser = build_parser()
    args = parser.parse_args()
    return int(args.func(args))


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

References

[1] X. Jiang, "Linear subspace learning-based dimensionality reduction," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 16–26, 2011. [Online]. Available: https://scholar.google.com/scholar?q=Linear+Subspace+Learning-Based+Dimensionality+Reduction

[2] ——, "Asymmetric principal component and discriminant analyses for pattern classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 931–937, 2009. [Online]. Available: https://scholar.google.com/scholar?q=Asymmetric+Principal+Component+and+Discriminant+Analyses+for+Pattern+Classification

[3] X. Jiang, B. Mandal, and A. C. Kot, "Eigenfeature regularization and extraction in face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 3, pp. 383–394, 2008. [Online]. Available: https://scholar.google.com/scholar?q=Eigenfeature+Regularization+and+Extraction+in+Face+Recognition

[4] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002. [Online]. Available: https://scholar.google.com/scholar?q=Jolliffe+Principal+Component+Analysis+2002

[5] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936. [Online]. Available: https://scholar.google.com/scholar?q=The+Use+of+Multiple+Measurements+in+Taxonomic+Problems

[6] B. Scholkopf, A. Smola, and K.-R. Muller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998. [Online]. Available: https://scholar.google.com/scholar?q=Nonlinear+Component+Analysis+as+a+Kernel+Eigenvalue+Problem

[7] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999. [Online]. Available: https://scholar.google.com/scholar?q=Learning+the+Parts+of+Objects+by+Non-Negative+Matrix+Factorization

[8] A. Hyvarinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Networks*, vol. 13, no. 4-5, pp. 411–430, 2000. [Online].

Available: https://scholar.google.com/scholar?q=Independent+Component+Analysis:+Algorithms+and+Applications

[9] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://scholar.google.com/scholar?q=Reducing+the+Dimensionality+of+Data+with+Neural+Networks

[10] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2014. [Online]. Available: https://scholar.google.com/scholar?q=Auto-Encoding+Variational+Bayes

[11] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017. [Online]. Available: https://scholar.google.com/scholar?q=Fashion-MNIST:+A+Novel+Image+Dataset+for+Benchmarking+Machine+Learning+Algorithms

[12] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*. IEEE, 1994, pp. 138–142. [Online]. Available: https://scholar.google.com/scholar?q=Parameterisation+of+a+Stochastic+Model+for+Human+Face+Identification