# When Close Isn't Close Enough: Exploring k-NN Under the Curse of Dimensionality

*On the Wine and Digits datasets, how do k, the distance metric, and the number of retained principal components affect k-NN accuracy, and what does this reveal about the curse of dimensionality?*

## 1 Abstract

This project asks how three design choices govern the accuracy of a k-Nearest Neighbours (k-NN) classifier: the number of neighbours k, the distance metric, and the number of principal components retained before classification. I implemented k-NN from scratch in NumPy, together with the evaluation metrics, and used scikit-learn only for principal component analysis (PCA), feature scaling and cross-validation splitting. The study runs on two contrasting datasets, Wine (178 samples, 13 features) and Digits (1797 samples, 64 features). Therefore, the same questions can be examined in a low-dimensional and a high-dimensional setting. Cross-validated sweeps show that k reproduces the taught bias-variance curve, that the three metrics separate by margins smaller than the fold-to-fold spread, and that dimensionality reduction matters most on Digits, where accuracy climbs steeply over the first fifteen components then plateaus. Standardisation proved decisive on Wine, lifting accuracy by about 0.24. The models selected by cross-validation reached 0.975 test accuracy on Wine and 0.967 on Digits, with macro F1 close to those figures.

## 2 Introduction

k-Nearest Neighbours is one of the oldest and most studied classifiers in machine learning. The idea of labelling a query point by the labels of its closest examples was formalised by Fix and Hodges (1951) and given its familiar theoretical footing by Cover and Hart (1967); it demonstrated that, as the sample size increases, the error rate of the one-nearest-neighbour rule is no more than twice the Bayes error rate. The method carries no training stage beyond storing the data, makes no assumption about the shape of the class distributions, and remains a common baseline against which more elaborate models are measured. These properties make it a natural subject for a study of how its behaviour depends on a small number of design choices.

This project aims to examine three of those choices together: the number of neighbours k, the distance metric used to compare points, and the number of principal components retained by PCA before classification. Each connects to a theme from the taught material. The value of k sets the bias-variance trade-off, with small k giving a flexible boundary of high variance and large k smoothing that boundary at the cost of bias. The distance metric determines what near means, which becomes delicate as the number of features grows. The number of retained components links directly to the curse of dimensionality, the observation that distance-based methods weaken as dimensionality rises because pairwise distances tend to concentrate (Beyer et al., 1999).

To study these effects in two methodologies, I use two standard datasets supplied with scikit-learn. Wine contains 178 samples described by 13 chemical measurements across three cultivars. Its features sit on very different numerical scales, which motivates standardisation, and its feature space is low-dimensional, giving a well-behaved baseline. Digits contains 1797 samples, each an eight-by-eight greyscale image of a handwritten numeral flattened to 64 pixels across ten classes. Its feature space is high-dimensional, and its pixels are strongly correlated, which makes it a suitable

place to look for the curse of dimensionality and to test whether PCA helps. Both datasets are close to class-balanced, so accuracy is a reasonable headline metric, and I also report macro-averaged precision, recall and F1 to weight every class equally.

Figure 1 shows one example image per class, which grounds the description of each sample as an eight-by-eight grid of pixels.

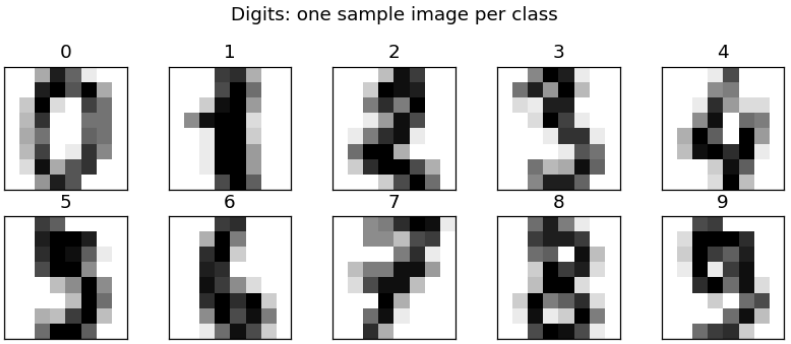


*Figure 1. Digits: one sample image per class.*

A known caution with Digits is that its intrinsic dimensionality is well below 64, since neighbouring pixels vary together, so the effective difficulty is lower than the raw feature count suggests. A caution with Wine is its small size, which makes single train-test splits noisy and motivates cross-validation throughout. Figure 2 shows a two-dimensional PCA projection of each dataset, which previews the class structure the classifier has to recover and the difference in separability between the two problems.

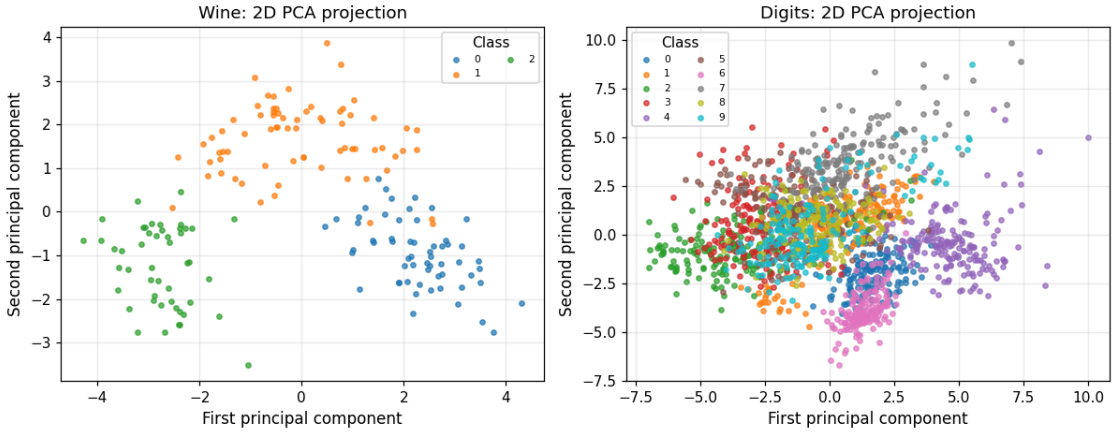


*Figure 2. Two-dimensional PCA projection of each dataset after standardisation, coloured by class. Fitted on the full data for illustration only and not used for any reported accuracy.*

# 3 Background

## k-Nearest Neighbours

The classifier stores the training set and defers all work to prediction time, which makes it a lazy learner. To classify a query point, it computes the distance from that point to every training point, ranks the training points from nearest to furthest, takes the k closest, and assigns the class that holds the majority among them. With k set to one, the prediction is the label of the single closest point, which traces a highly irregular boundary that follows individual examples. As k grows, the vote is taken over a wider neighbourhood, so the boundary smooths and the prediction becomes more stable

across resamples of the data. This is the bias-variance trade-off in a concrete form: low k gives low bias and high variance; high k gives higher bias and lower variance. A practical detail is tie handling. When two classes receive equal votes, the outcome must repeat across runs, so I break ties by the class whose nearest member among the k neighbours is closest, and settle any residual tie towards the smaller class label. Figure 3 pictures the rule and shows how a larger k can change the vote.

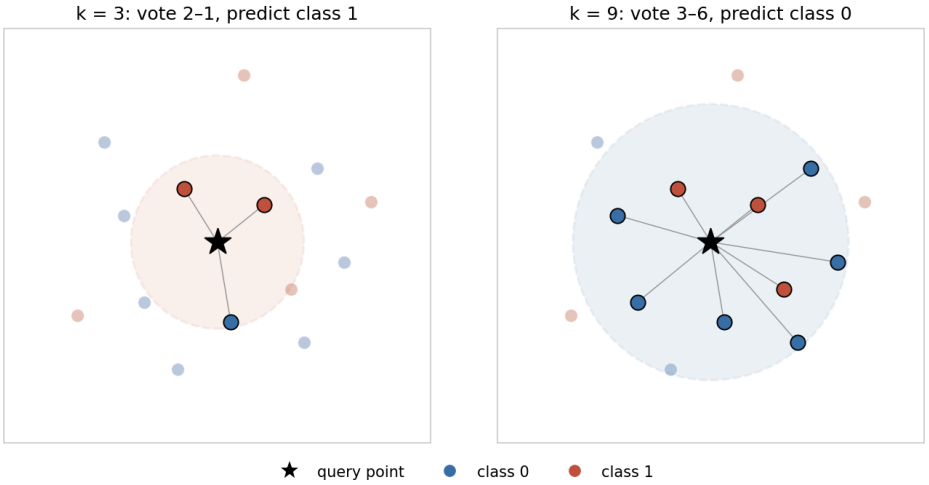


*Figure 3. The k-NN decision rule for a query point (star). The dashed ring encloses the k nearest training points that cast the vote. A small k follows the closest points, while a larger k smooths the decision over a wider neighbourhood, which can change the predicted class.*

## Distance metrics

The idea of nearest depends on how distance is measured, so I support three metrics. Euclidean (L2) distance is the straight-line distance and treats all directions equally. Manhattan (L1) distance sums the absolute differences per feature and is less sensitive to a large deviation in a single coordinate. Cosine distance measures one minus the cosine of the angle between two feature vectors, so it compares their direction and ignores their magnitude. Cosine is often suited to high-dimensional data, where the length of a vector can carry less information than its orientation, which is why it is worth testing on Digits.

## Principal component analysis

PCA is an unsupervised linear method that re-expresses the data in a set of orthogonal axes ordered by the variance they capture (Jolliffe, 2002). The first principal component is the direction of greatest variance in the data, the second is the direction of greatest remaining variance orthogonal to the first, and so on. Keeping the first m components projects the data into an m-dimensional subspace that retains as much variance as any m-dimensional linear projection can. For a distance-based classifier, this serves two purposes: it reduces the number of features, which shortens distance computation, and it can discard low-variance directions that carry mostly noise. PCA is fitted without reference to the class labels, so it maximises retained variance, which need not coincide with class separation, a distinction that matters when reading the results.

## The curse of dimensionality

As the number of features grows, the volume of the feature space grows so quickly that data become sparse and the contrast between the nearest and furthest neighbour of a point shrinks. Beyer et al. (1999) showed that under broad conditions the ratio of the furthest to the nearest distance approaches one as dimensionality rises, which erodes the meaning of a nearest neighbour.

Dimensionality reduction is one response, since projecting onto a smaller set of informative directions can restore useful distance contrast. Digits is the setting where this effect is expected, though its strong pixel correlations mean its intrinsic dimensionality is modest, so the effect should appear in a mild form. Figure 4 illustrates the underlying mechanism on random data.

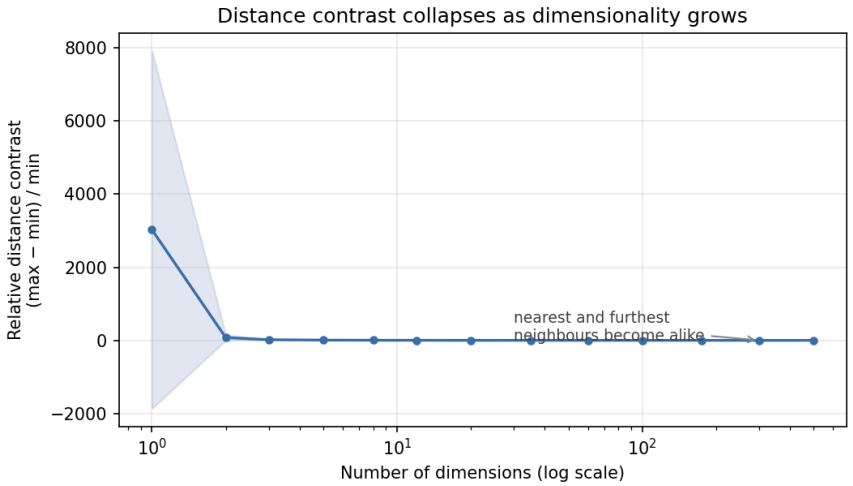


*Figure 4. Relative distance contrast, the gap between the furthest and nearest neighbour divided by the nearest, for uniform random points as the number of dimensions grows. The contrast shrinks toward zero, so nearest and furthest neighbours become almost equally distant. The band shows one standard deviation over repeated draws.*

## Standardisation, cross-validation and metrics

Distances add contributions from every feature, so a feature measured on a wide numerical range dominates the distance unless the features are placed on a common scale. Standardisation subtracts the mean and divides by the standard deviation of each feature, giving every feature zero mean and unit variance. For Wine, whose chemical measurements span very different ranges; this is expected to be important, and the scaling study tests exactly that. Figure 5 shows the raw feature ranges that make the point.

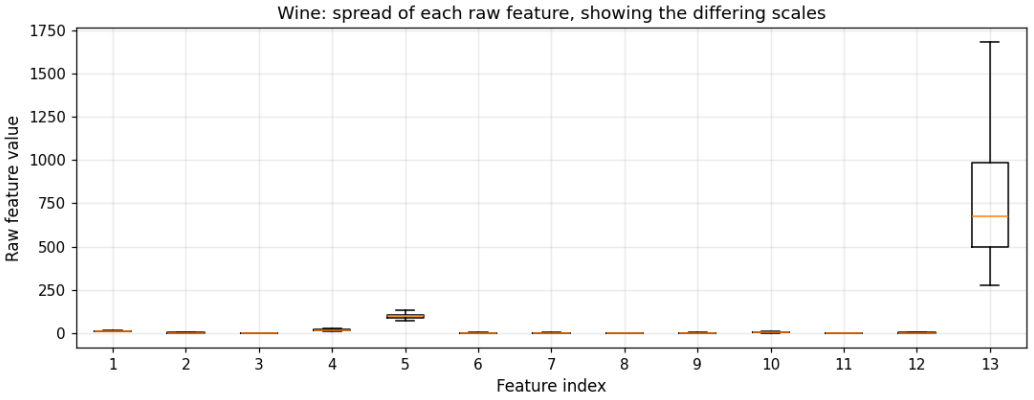


*Figure 5. Wine: spread of each raw feature, showing the differing numerical scales. Without standardisation, the widest-ranging feature would dominate the distance.*

A single train-test split gives one noisy estimate of accuracy, which is unreliable on a small dataset such as Wine. k-fold cross-validation partitions the training data into k folds, trains on all but one of them and evaluates on the held-out fold, then rotates so every fold is held out once, and averages the results. I use stratified folds so each fold preserves the class proportions, and I report the mean and the spread across folds to give both a central estimate and a sense of its stability. Accuracy is the proportion of correct predictions and is a fair headline for these near-balanced datasets. To weight

every class equally, I also compute macro-averaged precision, recall and F1 from the confusion matrix. Precision for a class is the share of its correct predictions, recall is the share of its true members that are recovered, and F1 is their harmonic mean. Macro-averaging takes the unweighted mean of the per-class figures, so a small class counts as much as a large one.

# 4 Methodology

### Reproducibility and the held-out test set

All work uses a single random seed (42) set once, so every split, fold and PCA fit repeats exactly. I set aside a stratified test set of 22 per cent of each dataset at the start (Wine 40 samples, Digits 396) and did not touch it until the final evaluation. Every sweep, tuning step and cross-validation ran on the training split alone (Wine 138, Digits 1401), which keeps the test estimate honest.

### k-NN implemented from scratch

The classifier is implemented in NumPy with no machine learning library inside the class. Fitting stores the training points and their labels. Prediction builds a full distance matrix between query and training points for the chosen metric, sorts each row with a stable sort so ties order reproducibly, takes the k nearest labels, and votes. The vote is vectorised: votes are counted per class, then broken first by the nearest member of a tied class and finally by the smaller label, using a single ranking key so the whole batch is decided at once.

```
class KNN:
    # Simple k-NN with euclidean, manhattan and cosine distances
    def __init__(self, k=5, metric="euclidean"):
        self.k = k
        self.metric = metric

    def fit(self, X, y):
        # remember the training points and their labels
        self.X_train = np.asarray(X, dtype=float)
        self.y_train = np.asarray(y)
        self.classes_ = np.unique(self.y_train)
        # store labels as 0-based indices so voting can use array positions
        self._class_index = np.searchsorted(self.classes_, self.y_train)
        return self

    def _distances(self, X):
        # distance from every query point to every training point
        query_points = np.asarray(X, dtype=float)
        train_points = self.X_train
        if self.metric == "euclidean":
            diff = query_points[:, None, :] - train_points[None, :, :]
            return np.sqrt(np.sum(diff ** 2, axis=2))
        if self.metric == "manhattan":
            diff = query_points[:, None, :] - train_points[None, :, :]
            return np.sum(np.abs(diff), axis=2)
        if self.metric == "cosine":
            # normalise each row to unit length, then use 1 minus the cosine similarity
            query_norm = query_points / np.linalg.norm(query_points, axis=1, keepdims=True)
            train_norm = train_points / np.linalg.norm(train_points, axis=1, keepdims=True)
            return 1.0 - query_norm @ train_norm.T
        raise ValueError("unknown metric: " + str(self.metric))

    def _sorted_neighbours(self, X):
        # for each query point, the training labels ordered from nearest to furthest
        distance_matrix = self._distances(X)
        order = np.argsort(distance_matrix, axis=1, kind="stable")
        return self._class_index[order]

    def _vote_k(self, sorted_classes, k):
        # majority vote over the k nearest neighbours.
        # if classes tie on votes, the one whose nearest neighbour is closest wins,
        # and any remaining tie goes to the smaller class label.
        top_k = sorted_classes[:, :k]
        n_queries = top_k.shape[0]
        n_classes = len(self.classes_)
```

```
        vote_counts = np.zeros((n_queries, n_classes), dtype=int)
        np.add.at(vote_counts, (np.arange(n_queries)[:, None], top_k), 1)
        # position of each class's nearest neighbour among the k (smaller is nearer)
        first_position = np.full((n_queries, n_classes), k, dtype=int)
        for position in range(k - 1, -1, -1):
            first_position[np.arange(n_queries), top_k[:, position]] = position
        # one number that ranks by votes, then by nearer neighbour, then by smaller label
        key = (vote_counts * (k + 1) * n_classes
               + (k - first_position) * n_classes
               + (n_classes - 1 - np.arange(n_classes)))
        return self.classes_[np.argmax(key, axis=1)]

    def predict(self, X):
        # classify each query point using the chosen k
        return self._vote_k(self._sorted_neighbours(X), self.k)
```

## Evaluation metrics from scratch

Accuracy, the confusion matrix, per-class recall and the macro precision, recall and F1 are coded directly from their definitions. The macro figures are derived from the confusion matrix by counting true positives on the diagonal and false positives and negatives from the column and row sums. Macro-averaging is the only technique used beyond the lecture material.

```
# Evaluation metrics, implemented from scratch.
# Macro-averaging of precision, recall and F1 weights every class equally, which suits the
# multiclass setting. This averaging is the only technique used beyond the lecture material.

def accuracy(y_true, y_pred):
    return float(np.mean(y_true == y_pred))

def confusion_matrix(y_true, y_pred, labels):
    # Rows are the true class, columns are the predicted class
    label_to_row = {label: i for i, label in enumerate(labels)}
    matrix = np.zeros((len(labels), len(labels)), dtype=int)
    for true_label, pred_label in zip(y_true, y_pred):
        matrix[label_to_row[true_label], label_to_row[pred_label]] += 1
    return matrix

def per_class_recall(conf_matrix):
    # Per-class accuracy is the diagonal divided by the true count of each class
    return np.diag(conf_matrix) / np.maximum(conf_matrix.sum(axis=1), 1)

def precision_recall_f1_macro(conf_matrix):
    # Derive the per-class figures from the confusion matrix, then average them
    true_pos = np.diag(conf_matrix).astype(float)
    false_pos = conf_matrix.sum(axis=0) - true_pos
    false_neg = conf_matrix.sum(axis=1) - true_pos
    precision = true_pos / np.maximum(true_pos + false_pos, 1e-12)
    recall = true_pos / np.maximum(true_pos + false_neg, 1e-12)
    f1 = 2 * precision * recall / np.maximum(precision + recall, 1e-12)
    return precision.mean(), recall.mean(), f1.mean(), precision, recall, f1
```

## Fold-safe preprocessing

Scaling and PCA are fitted on the training rows of each fold and then applied to the held-out rows, so no information from the held-out data enters the fitted transforms (Hastie, Tibshirani and Friedman, 2009). This ordering, standardise then PCA then k-NN, runs identically inside every cross-validation routine and in the final test.

```
def fit_transform_fold(X_train, X_test, n_components=None, standardise=True):
    # fit the scaler and PCA on the training rows only, then apply them to both
    if standardise:
        scaler = StandardScaler().fit(X_train)
        X_train, X_test = scaler.transform(X_train), scaler.transform(X_test)
    if n_components is not None:
        pca = PCA(n_components=n_components, random_state=MASTER_SEED).fit(X_train)
        X_train, X_test = pca.transform(X_train), pca.transform(X_test)
    return X_train, X_test
```

Figure 6 sets out this pipeline for one cross-validation fold and shows where the outer test split sits.

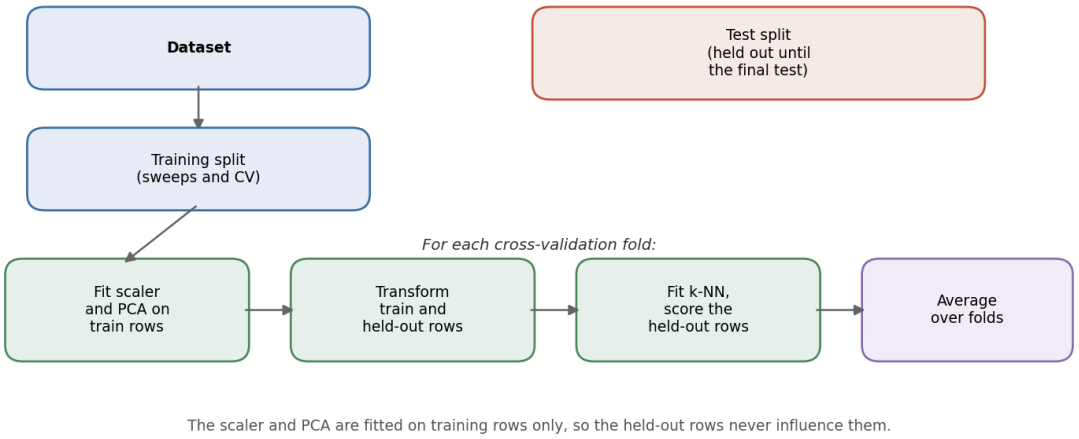


*Figure 6. The fold-safe cross-validation pipeline. Within each fold, the scaler and PCA are fitted on the training rows, then applied to the held-out rows before k-NN is scored, and the fold scores are averaged. The outer test split stays untouched until the final test.*

### Experiment design

I organised the study as eight experiments, each answering one question. E1 (correctness) compares the from-scratch classifier against scikit-learn's KNeighborsClassifier in brute-force mode across k in {1, 5, 11} and all three metrics on both datasets, using an internal split of the training data. Agreement and both accuracies are reported side by side, since the two implementations resolve ties by different documented conventions. E2 (exploratory analysis) uses class-distribution bars, Wine feature-scale boxplots, sample digit images and the two-dimensional PCA projection of Figure 2 to describe the data. E3 (effect of k) sweeps accuracy over k for each metric with five-fold stratified cross-validation and no PCA, so the effect of k is isolated. E4 (effect of metric) ranks the metrics by their best mean accuracy over k, reusing the E3 fold accuracies. E5 (effect of components) fixes k and metric at their E3, and E4 best and sweeps the number of PCA components, with PCA fitted inside each fold, alongside a cumulative explained-variance curve for illustration. E6 (joint selection) runs a grid search over k, metric and components, scoring every combination by five-fold cross-validation and keeping the best mean accuracy, preferring a smaller k to break ties. E7 (scaling study) compares standardised against raw features for k-NN on Wine across k. E8 (final test) trains the chosen configuration once on the full training split and evaluates it a single time on the untouched test set, reporting accuracy, macro precision, recall and F1, per-class recall and a confusion matrix.

## 5 Results

The from-scratch classifier matched scikit-learn closely in E1, with a mean prediction agreement of 0.9987 across all settings. The two agreed identically on Wine and on all k equal to one; the small disagreements occurred on Digits at k equal to 5 and 11, where a handful of query points tie and the two implementations break those ties by different conventions. The measured accuracies of the two implementations were equal to within rounding, which supports the correctness of the from-scratch code.

E3 and E4 characterise k and the metric. Both datasets reproduce the bias-variance pattern of Figure 7: accuracy at k equal to one is lower and more variable across folds, rises to a peak at a moderate k, then drifts down as larger k oversmooths the boundary.

Table 1 ranks the metrics by their best mean cross-validation accuracy over k. On Wine, the leading two, Euclidean and Manhattan, are level to four decimal places, so I read this as empirical model selection and not evidence that one metric is genuinely better. Cosine was worth testing on the higher-dimensional Digits and did not lead there either. Figure 8 presents the same comparison as a heatmap over k.

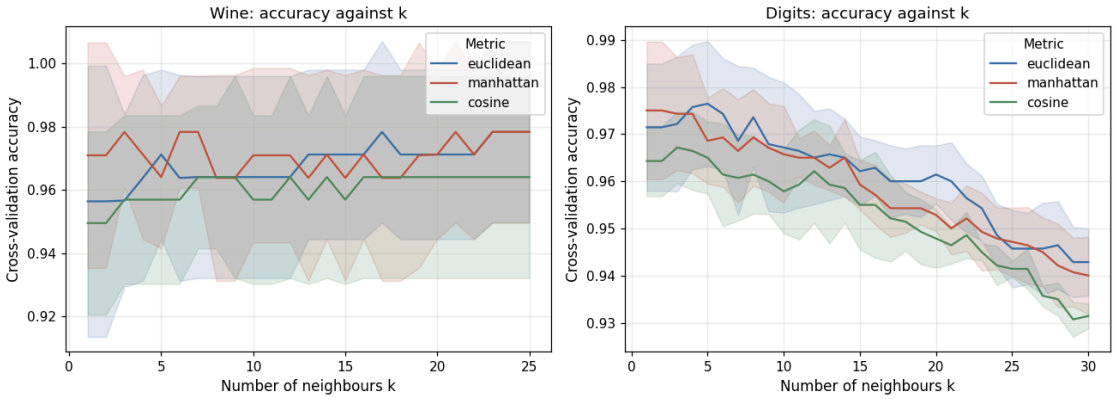


*Figure 7. Cross-validation accuracy against k for each metric, with plus or minus one standard deviation bands across the five folds. Left: Wine. Right: Digits.*

*Table 1. Metrics ranked by best mean cross-validation accuracy over k, with the k at which the best value occurs and the standard deviation across folds (E3, E4).*

| Dataset | Metric | Best k | Mean accuracy | Std (folds) |
|---|---|---|---|---|
| Wine | Euclidean | 17 | 0.9783 | 0.0287 |
| Wine | Manhattan | 3 | 0.9783 | 0.0177 |
| Wine | Cosine | 7 | 0.9640 | 0.0226 |
| Digits | Euclidean | 5 | 0.9765 | 0.0133 |
| Digits | Manhattan | 1 | 0.9750 | 0.0146 |
| Digits | Cosine | 3 | 0.9672 | 0.0076 |

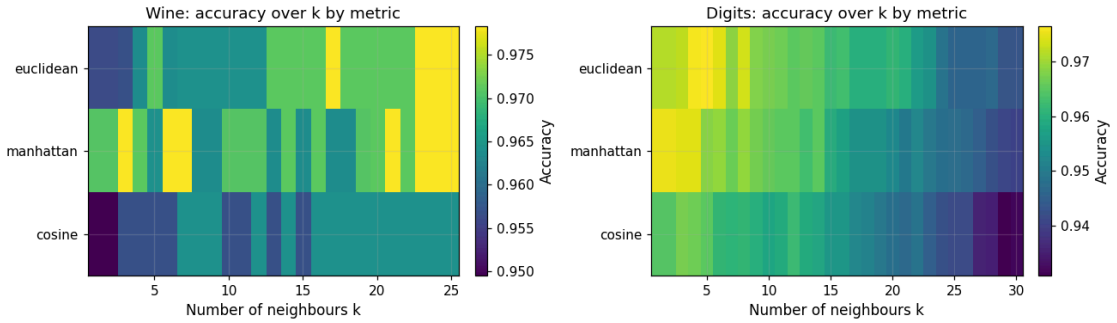


*Figure 8. Mean cross-validation accuracy over k by metric, one panel per dataset. The near-uniform shading within each panel reflects how small the differences between the metrics are.*

E5 isolates the number of PCA components, shown in Figure 9. On Wine, a low-dimensional set, accuracy is near its best with only a few components and moves little afterwards. On Digits, it climbs steeply over the first components, from 0.535 at two components to 0.927 at ten and 0.969 at fifteen, then settles into a plateau near its peak of 0.977 at 64 components. The expected curse-of-dimensionality fall is therefore mild here: the pixels are highly correlated, so the intrinsic dimensionality is well below 64, and PCA adds the noisiest directions last with a small magnitude.

The cumulative explained-variance curve keeps rising while accuracy flattens, a reminder that variance retained and classification accuracy are different objectives.

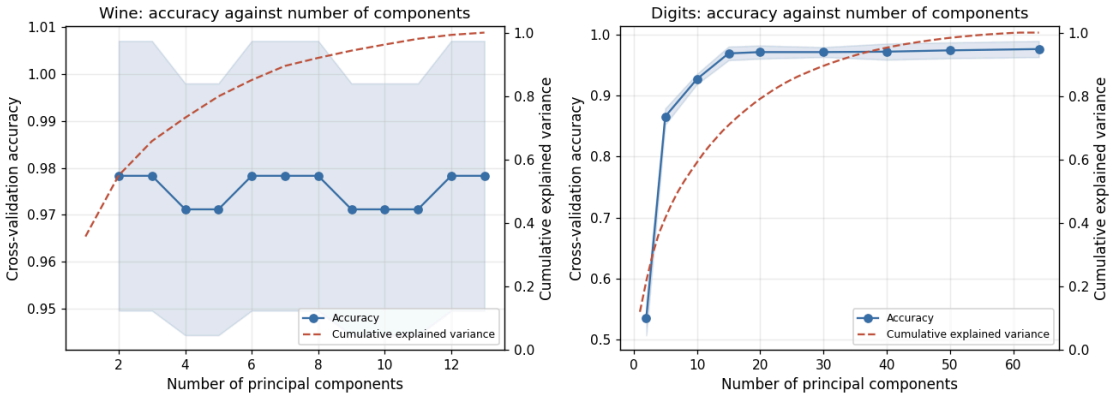


*Figure 9. Cross-validation accuracy against the number of PCA components (left axis, with plus or minus one standard deviation band) and cumulative explained variance (right axis). Left: Wine. Right: Digits.*

E6 selects k, the metric and the number of components together. The grid search picked Euclidean with a moderate k on both datasets, and the leading configurations were separated by small margins. On Wine, several configurations tied at a cross-validation accuracy of 0.9854, so the tie-break preferred the smaller k, giving Euclidean with k equal to 5 and 6 components. On Digits, the search kept all 64 components at a cross-validation accuracy of 0.9765, since reducing them did not improve on the full feature set. Figure 10 maps the accuracy over the grid, and Figure 11 shows the decision regions the chosen configuration draws in a two-component projection.

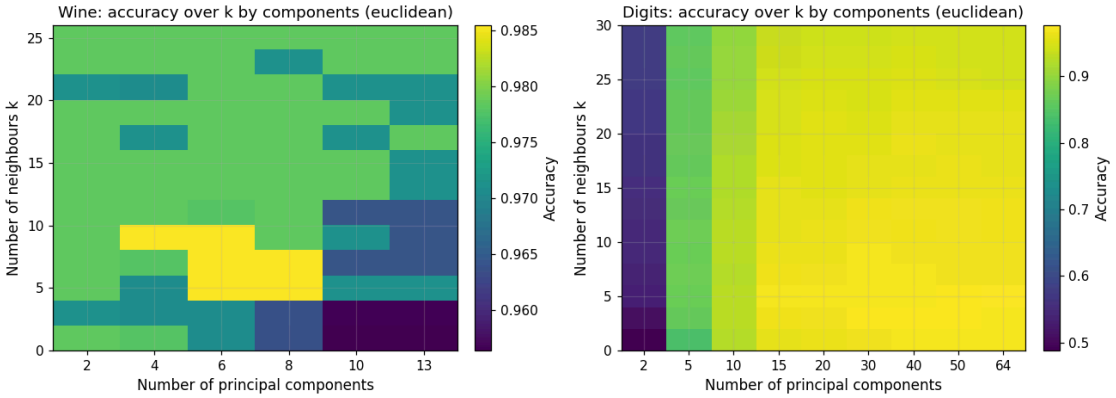


*Figure 10. Mean cross-validation accuracy over k by number of components for the best metric on each dataset. Left: Wine. Right: Digits.*

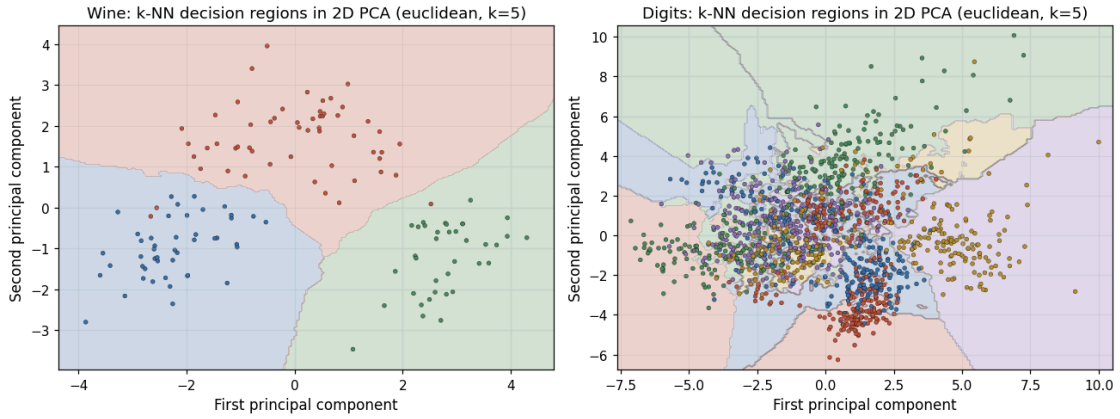


*Figure 11. k-NN decision regions in a two-component PCA projection of each training split, coloured by class. This two-dimensional view is illustrative and is not the space used for the reported accuracies. Left: Wine. Right: Digits.*

E7 measures the effect of scaling on Wine, shown in Figure 12. Standardising the features raised the best cross-validation accuracy from 0.7389 on raw features to 0.9783, a difference of 0.2394. This confirms how strongly a distance-based method depends on feature scaling when the raw features span very different ranges.

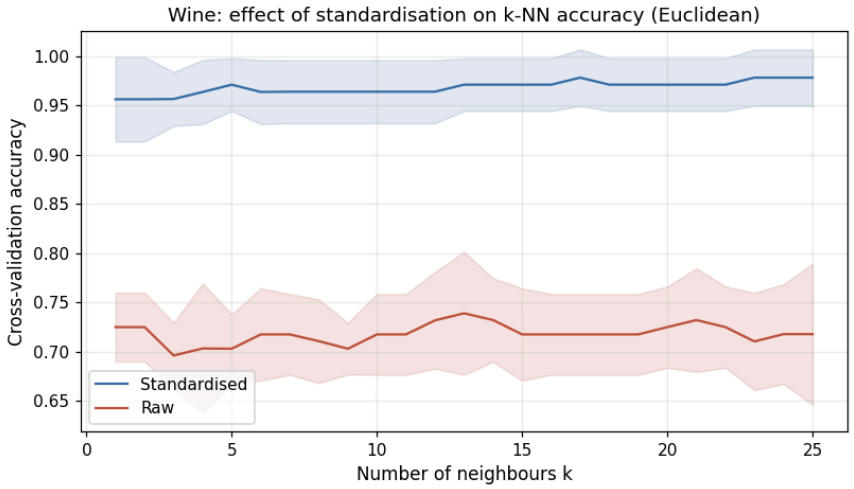


*Figure 12. Effect of standardisation on Wine k-NN accuracy across k, with the Euclidean metric. Bands show plus or minus one standard deviation across folds.*

E8 gives each chosen model its single test on the untouched test set. Table 2 collects the chosen configurations and the resulting figures, and Figure 13 shows the confusion matrices. Both models generalise well, with the errors on Digits concentrated on the visually similar numerals, most of the residual confusion falling on the classes eight and nine.

*Table 2. Configuration selected by cross-validation on the training split and its performance on the untouched test set (E6, E8).*

| Dataset | Metric | k | PCA comp. | CV acc. | Test acc. | Macro F1 |
|---|---|---|---|---|---|---|
| Wine | Euclidean | 5 | 6 | 0.985 | 0.975 | 0.975 |
| Digits | Euclidean | 5 | 64 | 0.977 | 0.967 | 0.967 |

On Wine, the final model reached a macro precision of 0.972 and a macro recall of 0.979, with per-class recall of 1.00, 0.94 and 1.00 across the three cultivars.

On Digits, it reached a macro precision of 0.968 and a macro recall of 0.967, with per-class recall at or above 0.90 for every numeral and the lowest values on nine (0.90) and eight (0.92).

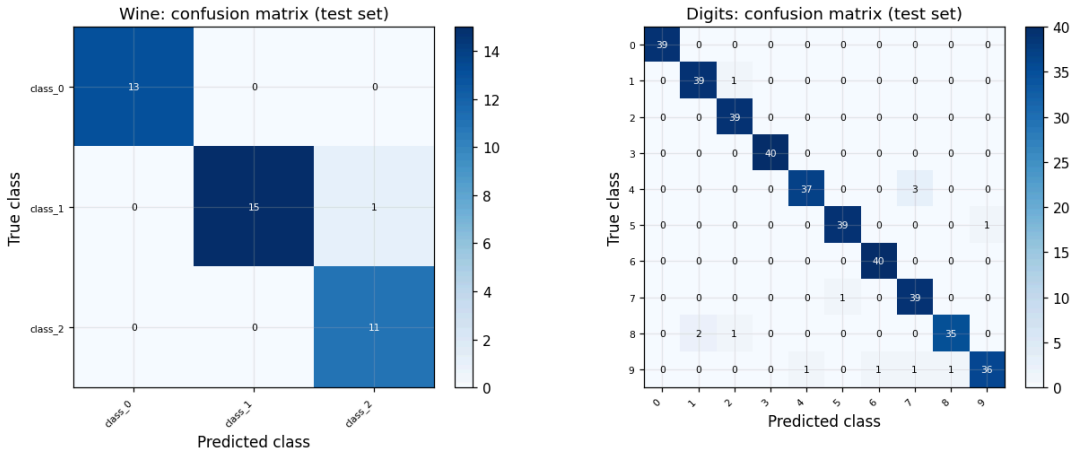


*Figure 13. Confusion matrices for the final models on the untouched test set. Left: Wine. Right: Digits.*

# 6 Evaluation

Judged against its stated aim, the project succeeded in characterising all three effects on both datasets. k reproduced the bias-variance curve, the metric differences fell within the fold-to-fold spread, and PCA helped where dimensionality was genuinely high, while the curse of dimensionality appeared in the mild form the data structure predicts. The conclusions are modest, and each is supported by the cross-validation evidence.

Several design decisions strengthen the estimates. Scaling and PCA are fitted inside each fold, so no held-out information leaks into the transforms, which is the most common source of optimistic bias in a pipeline of this kind. The from-scratch classifier was validated against a trusted library at 0.9987 agreement before it was used for any conclusion. The test set was set aside at the outset and touched once, so the final figures are an honest estimate of generalisation. Cross-validation with reported fold spread was used throughout, which matters on a small dataset. The metrics were coded from their definitions, and the tie-breaking was made deterministic, so the whole notebook repeats exactly.

The results also carry honest limitations. On the metric question, Euclidean and Manhattan tie to four decimal places on Wine, so the choice of Euclidean is an empirical selection and not a claim that it is the better metric; a fair statement is that the metric barely matters on these two problems. Cosine, expected to suit high-dimensional Digits, did not lead there, a negative result worth recording. On PCA, the joint search retained all 64 components on Digits, so dimensionality reduction confirmed the accuracy plateau while leaving the ceiling unchanged and shortening computation. That PCA maximises variance without regard to labels is visible in the widening gap between the explained-variance curve and accuracy in Figure 9, and it caps how much the technique can help a classifier.

Some threats to validity remain. Wine is small, so its cross-validation estimates carry meaningful variance and its 40-point test set moves by 0.025 for every single misclassification, which is why the Wine figures should be read at that granularity. The metric comparison in E4 fixed each metric at its own best k drawn from the same folds used to rank it, a mild optimistic bias, though one applied equally to all three metrics. The exploratory projection and the decision-region plots are fitted on the full data and are illustrative, so I kept them separate from the reported accuracies.

The brute-force distance computation is order n-train times n-query times d, which is comfortable at these sizes and would need a spatial index such as a k-d tree to scale to larger data, a step outside the aim. The E1 disagreements on Digits, while explained by tie-breaking, would be checked more strictly by enumerating the tied points.

A design judgement worth noting is the choice to fix k and the metric before the component sweep in E5, which isolates each effect cleanly but assumes the best k does not shift much with the number of components. The joint grid search in E6 relaxes that assumption and returned consistent choices, which gives some reassurance that the staged sweeps did not mislead. Taken together, the project is deliberately narrow, and a more ambitious aim, such as beating a strong tuned baseline on Digits, was not attempted, which keeps the claims within what the evidence supports.

# 7 Conclusions

The study set out to measure how k, the distance metric and the number of retained principal components affect k-NN accuracy across two datasets of contrasting dimensionality, and what this reveals about the curse of dimensionality. The value of k reproduced the bias-variance curve on both datasets, with accuracy lowest and most variable at k equal to one, a peak at a moderate k, and a gentle decline thereafter. The three metrics separated by margins smaller than the fold-to-fold spread; Euclidean was selected on both datasets, with Manhattan level on Wine, and cosine did not lead on the high-dimensional Digits.

Dimensionality reduction mattered most where dimensionality was genuinely high. Digits accuracy climbed steeply over the first fifteen components, then plateaued, and the joint search kept all 64 components, so reduction confirmed the plateau while leaving the peak unchanged; Wine reached its best with a handful of components. Standardisation was decisive on Wine, worth about 0.24 in accuracy, which confirms how strongly a distance-based method depends on feature scaling. The selected models reached 0.975 test accuracy on Wine and 0.967 on Digits, with macro F1 close to those figures. The headline is that the results follow the taught theory: k controls the bias-variance trade-off, scaling is essential for a distance-based method, and dimensionality reduction matters most where dimensionality is high. The main caution is Wine's small size, which limits the precision of its estimates and the strength of any claim drawn from them.

# 8 References


Beyer, K., Goldstein, J., Ramakrishnan, R. and Shaft, U. (1999). When is “nearest neighbor” meaningful? In Proceedings of the 7th International Conference on Database Theory (ICDT 1999), pp. 217 to 235. Springer.

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21 to 27.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis, nonparametric discrimination: consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas.

Hastie, T., Tibshirani, R. and Friedman, J. (2009). The Elements of Statistical Learning. Second edition. Springer.

Jolliffe, I. T. (2002). Principal Component Analysis. Second edition. Springer.