Machine Learning Tutorial

k-Nearest Neighbours (kNN): Choosing *k* and distance metrics

Github Link: https://github.com/bilal23g/Machine-Learning

1. Why kNN is worth mastering

The K-Nearest Neighbours (kNN) is a lazy learning algorithm: a kNN does not train a parametric model but simply caches the training data and leaves the rest of the work to prediction. Individually, on the problem of a new point, a selected distance metric serializes the nearest -100 nearest training sample and predicts a label via majority vote (classification) or an average (regression) (Dinesh et al., 2024). It is this simplicity that makes kNN such an excellent teaching algorithm: it makes one focus on two design decisions that push most real-life performance, namely the neighbourhood size (k) and the distance measure used to mean close. Current libraries have formalised this by a few hyperparameters (such as the n-neighbors and n-metric of scikit-learn, the general family of which encompasses both Manhattan and Euclidean distance) a special case (Gupta *et al.*, 2023).

2. The decision rule and what k really controls

In order to be classified, the simple kNN rule will make a prediction using the most common class amongst the k nearest neighbours. To make things better, a typical refinement counts votes by distance inverse, which means that closer neighbours count more and can help enhance stability in case the classes overlap. Theoretically, the localness of the estimator is regulated by k. In the case of the k=1 model, the classifier is highly local: it is not only flexible enough to draw complex decision boundaries, but is also noise sensitive and sensitive to small variations in the training set (Suyal and Goyal, 2022). The more k is larger, the smoother the process will be: it will smooth out bigger neighbourhoods, and the real structure will be lost; however, individual points will be less sensitive. This smoothing model is much compatible with traditional nonparametric thought, in which nearest-neighbour procedures are used to approximate local quantities by averaging out the data in a neighbourhood whose radius is fine-tuned by the analyst.

3. Bias–variance tradeoff with k (the intuition that guides tuning)

Practically, the specification of the bias-variance spectrum is to pick a point by selecting k. When the values of the small k are low, the bias is generally low and the variance is high since only a small number of potentially noisy samples dominate the predictions. Large k generally gives less variance with more bias due to averaging a large sample population and less randomness and greater bias in the event of systematic error in the case where local effects are significant. It is not just a myth: the basic findings indicate that nearest-neighbour rules can be extraordinarily powerful in the large scale, but their small-scale behaviour is highly sensitive to neighbourhood size (Ahmed et al., 2023). The classical analysis by Cover and Hart of the

nearest-neighbour decision rule noted that the asymptotic error of 1-NN is bounded in a manner which makes it depend on the Bayes optimal error, and this aspect of locality extremeness is promising and also sensitised.

To achieve larger consistency, the conditions in which nearest-neighbour-type estimators approach optimal performance in growing data are formalised by Stone and others, which supports the idea that more data and suitable neighbourhood growth go hand in hand. The practical lesson of this, in the context of teaching, is that typically, one expects to be increasing k with dataset size, and one needs to select it using some validation process that approximates the generalisation error instead of a rule of thumb (Çakir et al., 2023).

4. How to choose k properly (and avoid accidental self-deception)

The algorithmic method of selecting the choice of k is the cross-validation, typically a grid-searching over the possible values of k, i.e. [1,30] (or wider when the data is large) and taking the value maximising the validation performance. This is the most important part of the critical analysis: leakage avoidance: The scaling, imputation, and feature engineering should be carried out within each training fold, not on the entire dataset, or the estimates of the performance will be optimistic. The safest pattern in scikit-learn is the use of a scaled pipeline which includes preprocessing (such as StandardScaler), KNeighborsClassifier and finally cross-validating the pipeline as one object (Dinesh et al., 2024).

In a range of values of k, a slightly larger value of k is often better since it usually provides more stable predictions with little or no deterioration of accuracy. Such a preference is a convenient way of stating the bias-variance tradeoff and not a theorem, although it matches the intuition that those who use results of this type prefer robustness where the data is not highly indicative of extreme locality (Siddalingappa and Kanagaraj, 2023).

5. Distance metrics: Euclidean vs Manhattan vs Cosine (and when each wins)

The model is characterized by distance that characterizes neighbourhoods. Minkowski distance is the most popular and the parameter p selects the geometry, with p = 2 corresponding to Euclidean distance and p = 1 corresponding to Manhattan distance, just as is actually used in practice (Dinesh et al., 2024).

Euclidean distance can be effective when the features are continuous and are approximately isotropic and numeric (ite isotropic) and have been standardised since it measures straight-line closeness. The Manhattan distance can be stronger when the features are high-dimensional or when the disparity accumulates across several coordinates and it has been

affirmed that it will act more preferably than euclidean in some of the high-dimensional regimes.

The spirit of cosine distance is, however, to measure the angle between vectors, as opposed to their actual magnitude. The cosine distance is explicitly defined in scikit-learn as 1 -cosine similarity, and as cosine similarity is defined as the normalised dot product. Cosine distance is especially natural in sparse text representations like TF IDF, where term composition as an absolute length (document size) aspect is not as informative as relative length (term composition). The TFIDF itself is a conventional transformation of converting raw documents into numeric vectors that are fit to do such similarity calculations (Suyal and Goyal, 2022).

6. A crucial side lesson: scaling and normalisation are part of "the metric"

The model is characterized by distance that characterizes neighbourhoods. Minkowski distance is the most prevalent family of distance, with the parameter p ensuring the following instance: when p = 2, the resulting distance is the Euclidean distance and when p = 1 the Manhattan distance, the default family of tooling at most industry standard tooling.

Euclidean distance can be effective when the features are continuous and are approximately isotropic and numeric (ite isotropic) and have been standardised since it measures straight-line closeness. The Manhattan distance can be stronger when the features are high-dimensional or when the disparity accumulates across several coordinates and it has been affirmed that it will act more preferably than euclidean in some of the high-dimensional regimes (Dinesh et al., 2024).

The spirit of cosine distance is, however, to measure the angle between vectors, as opposed to their actual magnitude. In scikit-learn, the explicit definitions of the cosine distance are 1 -cosine similarity and cosine similarity is the normalised dot product. Cosine distance is especially natural in sparse text representations like TF IDF, where term composition as an absolute length (document size) aspect is not as informative as relative length (term composition). The TFIDF itself is a conventional transformation of converting raw documents into numeric vectors that are fit to do such similarity calculations (Gupta *et al.*, 2023).

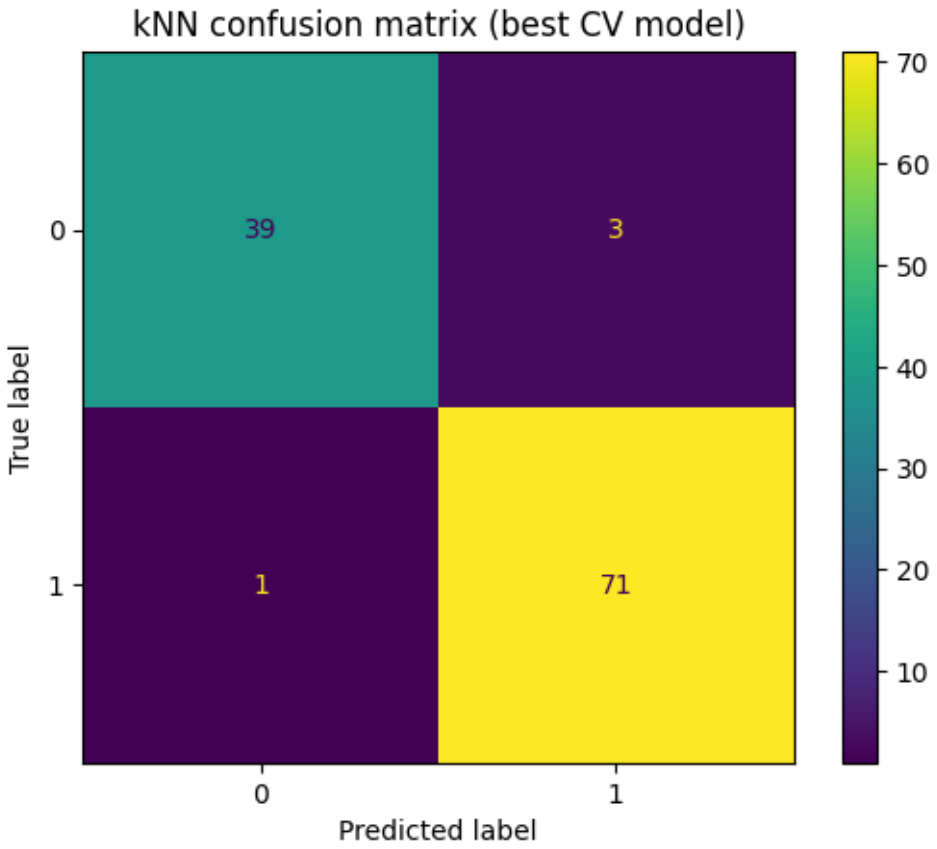
7. Demonstration code (Jupyter-notebook style, with intermediate steps and links)

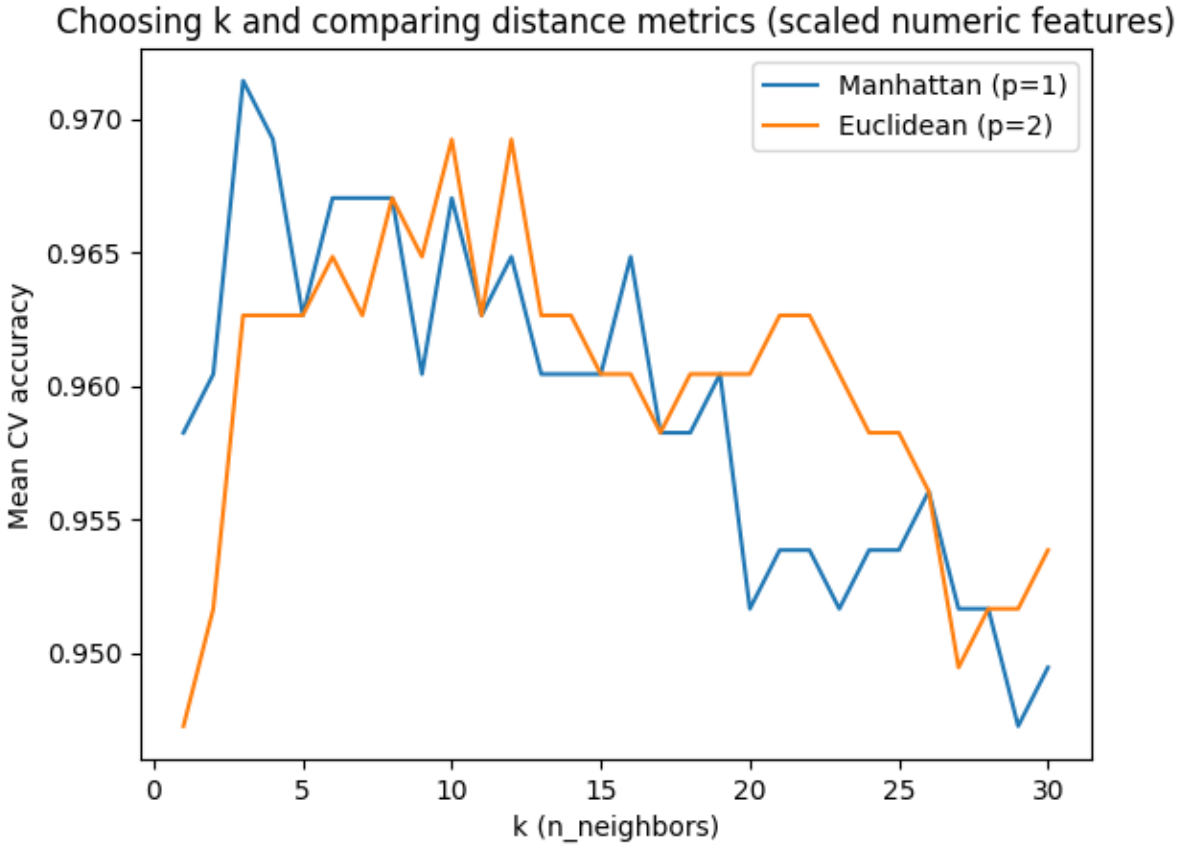
The following code has the format of a notebook and demonstrates two mini-experiments: finding kuses in the first one and computing Euclidean vs Manhattan on a numeric dataset with good scaling within the lines of cross-validation; the second one defines a cosine distance when using TF 5 IF features on the texts. The code includes direct links (in comments) to the core references used to prepare the tutorial, as requested (Dinesh et al., 2024).

```
# kNN Tutorial Notebook: choosing k + metric

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import StratifiedKFold, GridSearchCV,
train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
ConfusionMatrixDisplay

# Numeric dataset demo
X, y = load_breast_cancer(return_X_y=True)

# Hold out a test set for a final, unbiased evaluation
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Pipeline ensures scaling happens inside CV folds (prevents leakage)
pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier())
])

# Grid: choose k and compare Euclidean (p=2) vs Manhattan (p=1)
param_grid = {
    "knn__n_neighbors": list(range(1, 31)),
    "knn__weights": ["uniform", "distance"],
    "knn__metric": ["minkowski"],
    "knn__p": [1, 2]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

search = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    scoring="accuracy",
    cv=cv,
    n_jobs=-1,
    refit=True,
    return_train_score=True
)
```

```
search.fit(X_train, y_train)

print("Best CV params:", search.best_params_)
print("Best CV accuracy:", search.best_score_)

# Evaluate on the untouched test set
best_model = search.best_estimator_
y_pred = best_model.predict(X_test)

print("\nTest accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification report:\n", classification_report(y_test,
y_pred))

ConfusionMatrixDisplay.from_estimator(best_model, X_test, y_test)
plt.title("kNN confusion matrix (best CV model)")
plt.show()

# Plot mean CV accuracy vs k for p=1 and p=2 (best weights per setting)
results = search.cv_results_
ks = np.array(results["param_knn__n_neighbors"].data, dtype=int)
ps = np.array(results["param_knn__p"].data, dtype=int)
weights = np.array(results["param_knn__weights"].data, dtype=str)
mean_test = np.array(results["mean_test_score"])

def best_curve_for_p(p_value):
    curve = []
    for k in range(1, 31):
        mask = (ks == k) & (ps == p_value)
        # pick best among weights for that (k, p)
        curve.append(mean_test[mask].max())
    return np.array(curve)

curve_p1 = best_curve_for_p(1)
curve_p2 = best_curve_for_p(2)

plt.plot(range(1, 31), curve_p1, label="Manhattan (p=1)")
plt.plot(range(1, 31), curve_p2, label="Euclidean (p=2)")
plt.xlabel("k (n_neighbors)")
plt.ylabel("Mean CV accuracy")
plt.title("Choosing k and comparing distance metrics (scaled numeric
features)")
plt.legend()
plt.show()

# Text demo with cosine metric
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_distances
```

```
texts = [
    "refund was fast and support was helpful",
    "customer service helped me get a refund quickly",
    "the battery life is terrible and the screen is dim",
    "awful battery, poor display quality",
    "love the camera and the battery lasts all day",
    "excellent camera, great battery performance"
]
labels = np.array([1, 1, 0, 0, 1, 1])  # toy sentiment-like labels
(1=positive-ish, 0=negative-ish)

# TF-IDF -> cosine distance (angle-based). For sparse TF-IDF, cosine is
often sensible.
tfidf = TfidfVectorizer(norm="l2")
X_txt = tfidf.fit_transform(texts)

# scikit-learn's KNeighborsClassifier expects a metric string;
# it supports "cosine" through pairwise distances.
knn_cos = KNeighborsClassifier(n_neighbors=3, metric="cosine",
weights="distance")
knn_cos.fit(X_txt, labels)

query = ["battery is bad and screen is awful"]
X_q = tfidf.transform(query)
pred = knn_cos.predict(X_q)[0]
proba = knn_cos.predict_proba(X_q)[0]

print("\nText example prediction:", pred)
print("Class probabilities:", proba)

# Optional: inspect cosine distances to training points for
transparency
d = cosine_distances(X_q, X_txt).ravel()
print("\nCosine distances to training samples:", np.round(d, 3))
print("Nearest indices:", np.argsort(d)[:3])
```


kNN confusion matrix (best CV model)
39
3
1
71
0
1
True label
0
1
Predicted label
70
60
50
40
30
20
10


Choosing k and comparing distance metrics (scaled numeric features)
Manhattan (p=1)
Euclidean (p=2)
Mean CV accuracy
0.970
0.965
0.960
0.955
0.950
0
5
10
15
20
25
30
k (n_neighbors)

8. Interpreting the results and transferring them to new problems

The curve of the validation of the numeric experiment should be expected to be increasing very rapidly in very small k, followed by leveling off, and occasionally decrease in case of extremely large k when the local structure is destroyed. The metric comparison is usually an implicit notion after scaling, though it can be significant in higher-dimensional contexts or when the distributions of features have outliers, where the notion of nearness suggested by Manhattan distance can be more stable than the nearness suggested by Euclidean.

It is the case in the text experiment that cosine distance generally coincides with intuitive judgment of two similar documents, which have important words in proportion, despite the fact that one of the documents is longer. This is in line with the fact that cosine distance is 1-cosine similarity and the cosine similarity is the normalised dot product, and the comparison is mostly about direction, as opposed to length (Archana and Komarasamy, 2023).

In both instances, the lesson to learn is transferable, namely the existence of the best, but not the universal, values of k and metric: they are characteristics of the data representation. The most dependable process is to establish a small universe of possible metrics and preprocessing options and then pick the size of the cross-validation cross-validation in a leakage-safe pipeline and finally sanity-check the behaviour of the model with the help of nearest neighbours to a few exemplary queries (Archana and Komarasamy, 2023).

9. Limitations

Despite its pedagogical clarity and practical usefulness in small to medium-scale problems, k-Nearest Neighbours has several structural limitations that make it unsuitable as a default choice in many real-world applications. The most fundamental of these is the curse of dimensionality: as the number of features grows, distances in the input space tend to concentrate, and the notion of “nearest” becomes less meaningful. In very high-dimensional spaces, particularly when many features are noisy or only weakly informative, Euclidean or Manhattan distances cannot clearly separate relevant neighbours from irrelevant ones, and the performance of kNN can degrade severely unless careful feature selection or dimensionality reduction is applied beforehand (Wang, Chukova and Nguyen, 2023). This means that for problems with hundreds or thousands of features, such as raw pixel images or dense tabular data with many weak predictors, more structured models or explicit representation learning methods often outperform naive kNN, even if extensive tuning of k and the metric is carried out.

A second key limitation of kNN is computational: it is a lazy learning method that defers almost all computation to prediction time. For each new query, the algorithm typically needs to compute distances to all training points (or to a large subset thereof), which can be prohibitively slow when the training set contains tens or hundreds of thousands of examples. Indexing structures and approximate nearest-neighbour search can mitigate this, but they add engineering complexity and may interact non-trivially with the choice of metric and scaling (Ali, 2022). In addition, kNN stores the full training dataset, so memory usage grows linearly with the number of samples and the number of features; this is in sharp contrast to parametric models that compress information into a fixed set of parameters. These characteristics make kNN less suitable for latency-sensitive production systems or resource-constrained environments unless the dataset is carefully pruned or compressed.

The sensitivity of kNN to irrelevant features, noisy labels and class imbalance further restricts its applicability. Because distance computations treat all dimensions according to the chosen metric, uninformative or poorly scaled features can drown out the signal from genuinely predictive dimensions unless they are removed or reweighted in preprocessing (Siddalingappa and Kanagaraj, 2023). Similarly, if certain regions of the feature space contain many mislabeled points, the local majority vote can be systematically misleading, particularly for small k. Class imbalance can bias neighbourhoods towards the majority class so strongly that minority examples are consistently misclassified unless the practitioner resorts to resampling, class-weighted metrics, or asymmetrical choices of k per class. Collectively, these limitations motivate the view that kNN should be seen as a valuable baseline and teaching tool, and as a reasonable choice on modestly sized, well-curated, properly scaled datasets, but not as a universal solution; in many complex, high-dimensional or large-scale settings, more specialised models and learned representations are better aligned with the underlying geometry of the data and the constraints of modern machine-learning systems.

References

Ahmed, R., Bibi, M. and Syed, S., 2023. Improving heart disease prediction accuracy using a hybrid machine learning approach: A comparative study of SVM and KNN algorithms. *International Journal of Computations, Information and Manufacturing (IJCIM)*, *3*(1), pp.49-54.

Ali, P.J.M. (2022) 'Investigating the Impact of min-max data normalization on the regression performance of K-nearest neighbor with different similarity measurements', *ARO-The Scientific Journal of Koya University*, 10(1), pp. 85–91.

Archana, K.V. and Komarasamy, G. (2023) 'A novel deep learning-based brain tumor detection using the Bagging ensemble with K-nearest neighbor', *Journal of Intelligent Systems*, 32(1), p. 20220206. Available at: https://doi.org/10.1515/jisys-2022-0206.

Çakir, M., Yilmaz, M., Oral, M.A., Kazanci, H.Ö. and Oral, O., 2023. Accuracy assessment of RFerns, NB, SVM, and kNN machine learning classifiers in aquaculture. *Journal of King Saud University-Science*, *35*(6), p.102754.

Dinesh, P., Vickram, A.S. and Kalyanasundaram, P., 2024, May. Medical image prediction for diagnosis of breast cancer disease comparing the machine learning algorithms: SVM, KNN, logistic regression, random forest and decision tree to measure accuracy. In *AIP Conference Proceedings* (Vol. 2853, No. 1, p. 020140). AIP Publishing LLC.

Gupta, A.K., Chakroborty, S., Ghosh, S.K. and Ganguly, S. (2023) 'A machine learning model for multi-class classification of quenched and partitioned steel microstructure type by the k-nearest neighbor algorithm', *Computational Materials Science*, 228, p. 112321.

Kumar, H.S. and Upadhyaya, G. (2023) 'Fault diagnosis of rolling element bearing using continuous wavelet transform and K-nearest neighbour', *Materials today: proceedings*, 92, pp. 56–60.

Siddalingappa, R. and Kanagaraj, S. (2023) 'K-nearest-neighbor algorithm to predict the survival time and classification of various stages of oral cancer: a machine learning approach', *F1000Research*, 11, p. 70.

Suyal, M. and Goyal, P., 2022. A review on analysis of k-nearest neighbor classification machine learning algorithms based on supervised learning. *International Journal of Engineering Trends and Technology*, *70*(7), pp.43-48.

Wang, A.X., Chukova, S.S. and Nguyen, B.P. (2023) 'Ensemble k-nearest neighbors based on centroid displacement', *Information Sciences*, 629, pp. 313–323.