# K Nearest Neighbors (KNN) Cheat Sheet

*Mastering Classification & Regression: Concepts, Math, and Implementation*

## 1 What is K Nearest Neighbors

**Definition**: A non-parametric, supervised learning algorithm that classifies or predicts a value for a new data point based on the $K$ closest training data points.

**How it Works**:

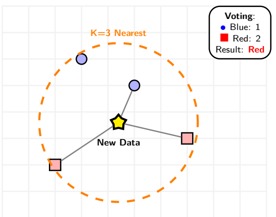
1. Store all training data.
2. For a new input, calculate distance to all training points.
3. Select the $K$ nearest points.
4. **Vote** (Classification) or **Average** (Regression).

**Lazy Learning**: KNN does not "learn" a model (weights/biases) during training. It simply stores data. All computation happens at prediction time (Eager evaluation).

**Use Cases**: Recommendation systems, pattern recognition, data imputation.

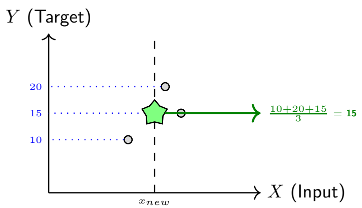
## 2 KNN Classification vs Regression

**Classification (Voting)**:




- **Output**: Discrete Class Label.
- **Mechanism**: Majority Vote (Mode).
- **Example**: Is this email Spam or Not?

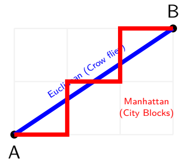
**Regression (Averaging)**:




- **Output**: Continuous Value.
- **Mechanism**: Average (Mean) of neighbors' values.
- **Example**: Estimate house price based on 3 neighbors.

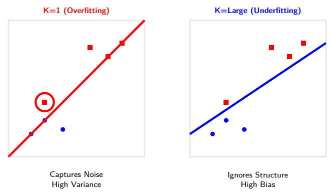
## 3 Distance Metrics

How "closeness" is measured drastically affects performance.




- **Euclidean (L2)**: Straight line. Standard for continuous data. Sensitive to scale.
- **Manhattan (L1)**: Grid path. Good for high dimensions or sparse data.
- **Minkowski**: Generalization. $p = 1$ is Manhattan, $p = 2$ is Euclidean.
- **Cosine**: Angle between vectors. Best for text/NLP where magnitude matters less than direction.

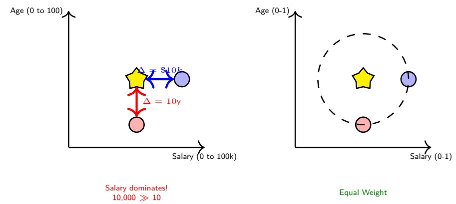
## 4 Choosing the Value of K




- **Small K (e.g., 1)**: Low Bias, High Variance. Model captures noise. **Overfitting**. Decision boundary is jagged.
- **Large K**: High Bias, Low Variance. Model smooths over details. **Underfitting**. Computationally cheaper.
- **Selection**: Use **Cross-Validation** (e.g., Elbow Method) to find the K that minimizes validation error.

## 5 Data Preprocessing

**Feature Scaling (CRITICAL)**: KNN calculates distances. If Feature A ranges [0-1] and Feature B [0-1000], Feature B will dominate the distance calculation.




- **Standardization**: $\frac{x-\mu}{\sigma}$ (Mean 0, Var 1). Good for outliers.
- **Normalization**: $\frac{x-min}{max-min}$ (Range 0-1). Good for fixed bounds.

**Handling Categorical**: Must convert to numeric (One-Hot Encoding) as distance functions require numbers.

## 6 KNN Algorithm Step-by-Step

1. **Load Data**: Initialize training set $D$. 2. **Input**: Receive query point $q$. 3. **Distance**: For every point $x_i$ in $D$, calculate $dist(q, x_i)$. 4. **Sort**: Order points by increasing distance. 5. **Select**: Pick top $K$ points. 6. **Aggregate**: - *Class*: Return Mode(Labels). - *Reg*: Return Mean(Values).

## 7 End-to-End Classification

**Dataset**: Iris (Predict species).

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 1. Load & Split
data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, test_size=0.2
)

# 2. Scale (Mandatory)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 3. Train
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# 4. Predict & Evaluate
preds = knn.predict(X_test)
print(f"Acc: {accuracy_score(y_test, preds)}")
```

## 8 End-to-End Regression

**Dataset**: California Housing (Predict price).

```python
from sklearn.datasets import fetch_california_housing
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# 1. Load & Split
data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, test_size=0.2
)

# 2. Scale
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 3. Train
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train, y_train)

# 4. Predict & Evaluate
preds = knn_reg.predict(X_test)
mse = mean_squared_error(y_test, preds)
print(f"RMSE: {mse**0.5:.2f}")
```

## 9 Performance Metrics

**Classification**:

- **Accuracy**: Correct / Total.
- **Precision/Recall**: Quality of positive predictions.
- **Confusion Matrix**: Shows Type I/II errors.

**Regression**:

- **MSE**: Mean Squared Error (Penalizes large errors).
- **RMSE**: Root MSE (Same units as target).
- **MAE**: Mean Absolute Error (Robust to outliers).

## 10 Hyperparameter Tuning

Using `GridSearchCV` to find the best model.

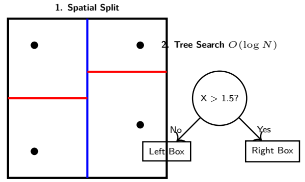
- **n_neighbors**: Range [1, 30]. Odd numbers preferred to avoid ties.
- **weights**: 'uniform' (all neighbors equal) vs 'distance' (closer neighbors count more).
- **metric**: 'euclidean', 'manhattan'.

## 11 Complexity & Scalability

**Time Complexity**:

- Training: $O(1)$ (Just storage).
- Prediction: $O(N \times D)$ where $N$ is samples, $D$ is dimensions. **Very Slow** for large $N$.

**Scalability Issues**: KNN requires calculating distance to *every* point. **Solution**: Use Approximate Nearest Neighbors (ANN) or Tree structures:




## 12 Pitfalls & Best Practices

**Curse of Dimensionality**: As features $(D)$ increase, data becomes sparse. Distances between "nearest" and "farthest" neighbors converge, making distance meaningless. *Fix*: Use Dimensionality Reduction (PCA) before KNN.
**Irrelevant Features**: Noise features dilutes distance. *Fix*: Feature Selection.
**Imbalanced Data**: Majority class dominates voting. *Fix*: Use ‘weights='distance'‘.

## 13 Summary

| Concept | Key Takeaway |
|---|---|
| Type | Instance-based, Lazy |
| Metric | Euclidean (Standard), Cosine (Text) |
| Scaling | **Mandatory** |
| K | Hyperparameter (Tune it!) |
| Speed | Fast Train, Slow Predict |