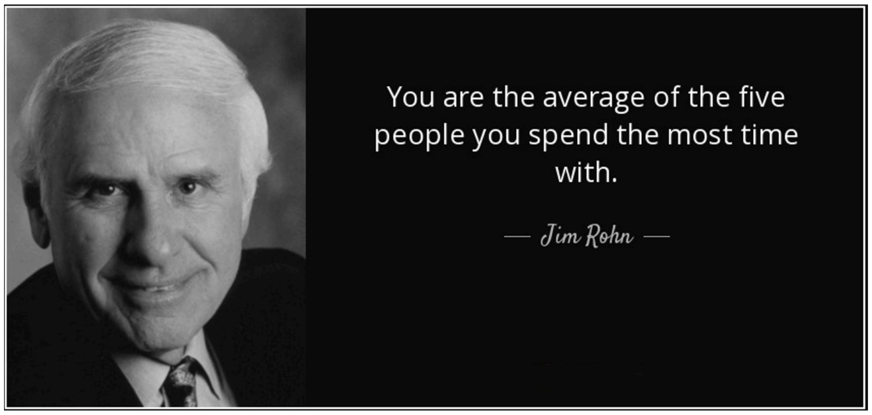
K Nearest Neighbors

`from` `sklearn.neighbors import KNeighborsClassifier`


You are the average of the five people you spend the most time with.
— Jim Rohn —

- **Type**: Supervised learning algorithm for **classification** and **regression**.
- **Key Idea**: Predictions are based on the similarity (distance) to the nearest training examples.
- **Instance-Based**: Stores the entire dataset (no explicit training phase; "lazy learner").
- **Non-Parametric**: Makes no assumptions about data distribution.

💡 **Scale your data**: Always normalize/standardize.

How It Works:

- **For Classification:**When predicting the class of a new sample, KNN finds the **k closest samples** (neighbors) in the feature space and assigns the class that appears most frequently among them.

- **For Regression:**It predicts the value by averaging the values of the **k nearest neighbors**.

Step-by-Step Process:

1. **Choose Hyperparameters**:
    - **k**: Number of neighbors to consider.
    - **Distance Metric**: How to measure similarity (e.g., Euclidean, Manhattan).
2. **Calculate Distances**:
    - Compute the distance between the new data point and all training examples.
3. **Find k-Nearest Neighbors**:
    - Select the `k` training points closest to the new point.
4. **Make Prediction**:
    - **Classification**: Majority vote among neighbors.
    - **Regression**: Average/median of neighbors' values.

💡 **A small `k` can be sensitive to noise, while a large k can smooth out the predictions.**

Example:

- **Task**: Classify a new flower as "Iris-setosa" or "Iris-versicolor".
- **k=3**: Check the 3 closest flowers in the training data. If 2 are "setosa", predict "setosa".

Distance Metrics

Metric	Formula	Use Case
Euclidean	$\sqrt{(\Sigma(x_i - y_i)^2}$	Continuous features (default choice).
Manhattan	$\Sigma\|x_i - y_i\|$	High-dimensional/categorical data.

Minkowski	$(\Sigma\|x_i - y_i\|^p)^(1/p)$	Generalizes Euclidean (p=2) and Manhattan (p=1).
Cosine	$(x\cdot y)/(\|x\|\|y\|)$	Text/data with directionality (e.g., TF-IDF vectors).

Choosing `k`

- **Small k (e.g., 1)**:
  - High variance (overfitting): Sensitive to noise.
  - Example: A single outlier can misclassify a point.
- **Large k (e.g., 20)**:
  - High bias (underfitting): Smoothens decision boundaries.
  - Example: May ignore small/local patterns.

Rule of Thumb: Start with `k = √n` (where `n` is the number of samples) and tune via cross-validation.

Handling Challenges

- **Curse of Dimensionality**:
  - Reduce features via PCA or feature selection.
- **Imbalanced Data**:
  - Use class weights or resampling (SMOTE).
- **Missing Values**:
  - Impute missing data before applying KNN.

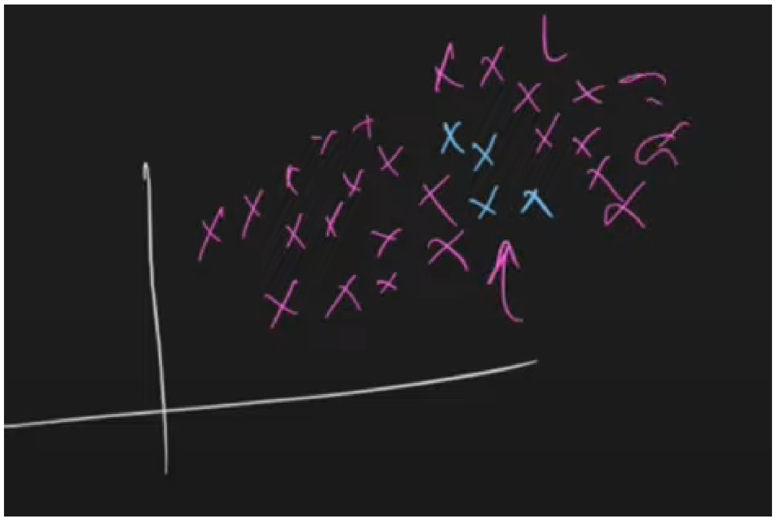
Key Takeaways

- **k is critical**: Tune it using cross-validation.
- **Scale your data**: Always normalize/standardize.

- **Avoid high dimensions**: Use KNN with < 20 features for best results.

Limitations of KNN

- Large datasets
  - KNN is lazy learning technique
  - You do nothing in training phase
  - All the work is done in prediction phase
  - Therefore, training is fast but prediction is slow if dataset is large.
- Curse of dimensionality
  - Distances aren't reliable in higher dimensions
- Doesn't work well with outliers
- Non-homogenous scale
  - Needs scaling
- Imbalanced dataset 👇



- Fails for inferences

k Nearest Neighbour (KNN) Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
e=42)

# Create a KNN classifier with k=5 neighbors
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = knn.predict(X_test)

# Calculate and print the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:

```
Accuracy: 1.0
```

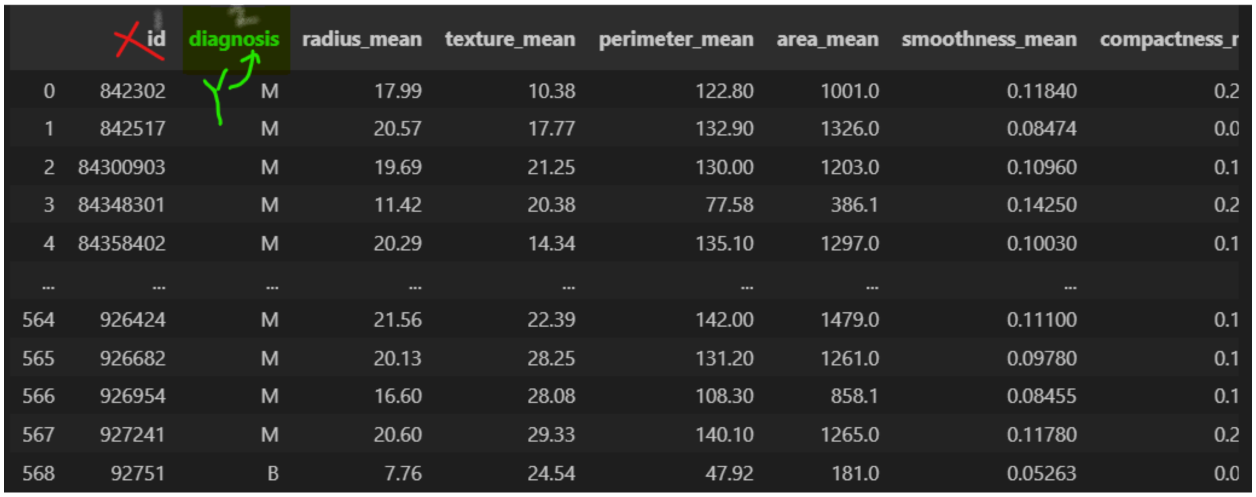
Another example:

- Dataset: breast cancer dataset

```
df = pd.read_csv(r'https://raw.githubusercontent.com/G1Codes/Datasets/refs/
heads/main/Breast%20Cancer%20Wisconsin.csv')
df.head()
df=df.iloc[:, :-1] # removed the last NaN column
df.shape

Output:
(569, 32)
```

`df=df.iloc[:, :-1]` → removed the last **NaN** column

```
df
```

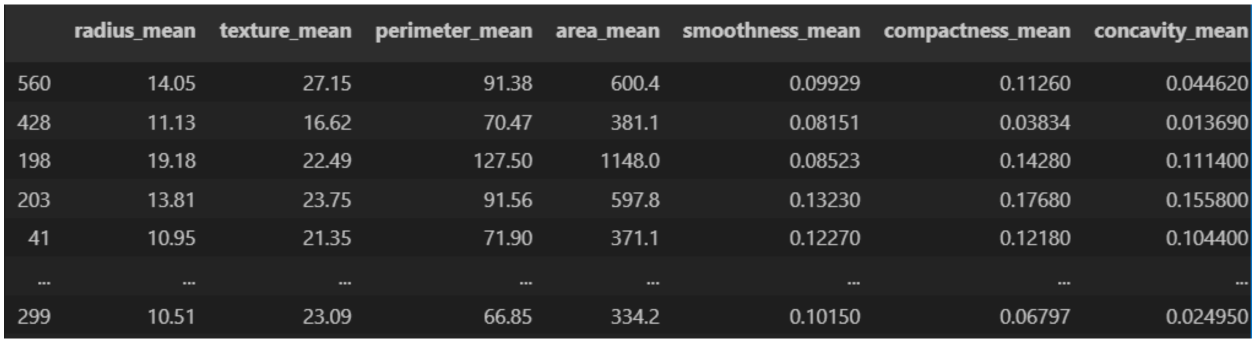


	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_r
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.2
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.2
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.1
...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.1
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.1
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.1
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.2
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.0

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,2:], df.iloc[:,1],test_size
=0.2, random_state=2)
```

`df.iloc[:,2:]` → Dropped 1st 2 columns: id & diagnosis

```
X_train
```



	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
560	14.05	27.15	91.38	600.4	0.09929	0.11260	0.044620
428	11.13	16.62	70.47	381.1	0.08151	0.03834	0.013690
198	19.18	22.49	127.50	1148.0	0.08523	0.14280	0.111400
203	13.81	23.75	91.56	597.8	0.13230	0.17680	0.155800
41	10.95	21.35	71.90	371.1	0.12270	0.12180	0.104400
...	...	...	...	...	...	...	...
299	10.51	23.09	66.85	334.2	0.10150	0.06797	0.024950

```
X_train.shape

Output:
(455, 30)
```

Scale (VVIMP Step):

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Apply KNN:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
```

Fit:

```
knn.fit(X_train,y_train)
```

```
from sklearn.metrics import accuracy_score

y_pred = knn.predict(X_test)

accuracy_score(y_test, y_pred)

Output:
0.9736842105263158
```

- In classification model, we use accuracy score.

For `n_neighbors=3`**, Accuracy Score= 0.9912280701754386**

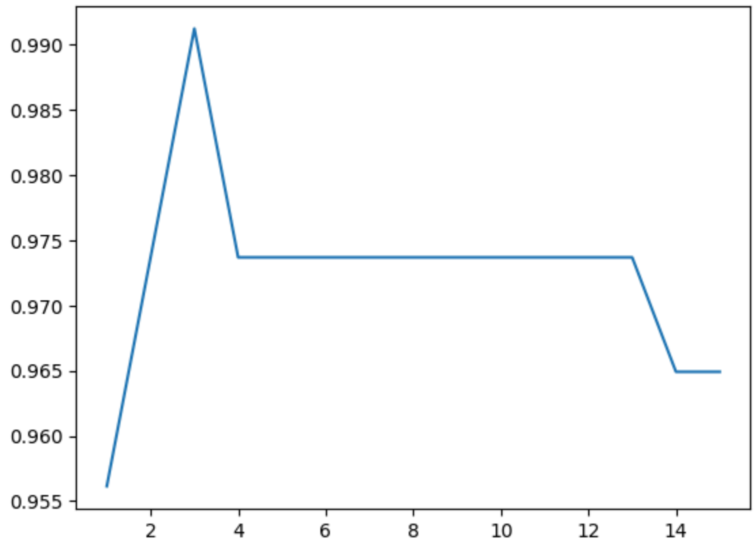
How to select K?

- Heuristic Approach → $\sqrt{n}$
  - n is number of observations
- Experimentation → Try different values
- **Avoid even values**

Test different values of k:

```
scores = []

for i in range(1,16):
    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train,y_train)

    y_pred = knn.predict(X_test)

    scores.append(accuracy_score(y_test, y_pred))
```

```
import matplotlib.pyplot as plt

plt.plot(range(1,16),scores)
```



Find Best K using from GridSearchCV

`sklearn.model_selection import GridSearchCV`

```
param_grid = {'n_neighbors': np.arange(1, 21)}
```

```
# Set up GridSearchCV with KNeighborsClassifier
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scorin
g='accuracy')
grid_search.fit(X_train, y_train)

# Retrieve the best k value and best cross-validation score
best_k = grid_search.best_params_['n_neighbors']
best_cv_accuracy = grid_search.best_score_

print(f"Best k value: {best_k}")
print(f"Best cross-validated accuracy: {best_cv_accuracy:.4f}")

# Evaluate on the test set
y_pred = grid_search.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test set accuracy: {test_accuracy:.4f}")

Output:
Best k value: 3
Best cross-validated accuracy: 0.9670
Test set accuracy: 0.9912
```

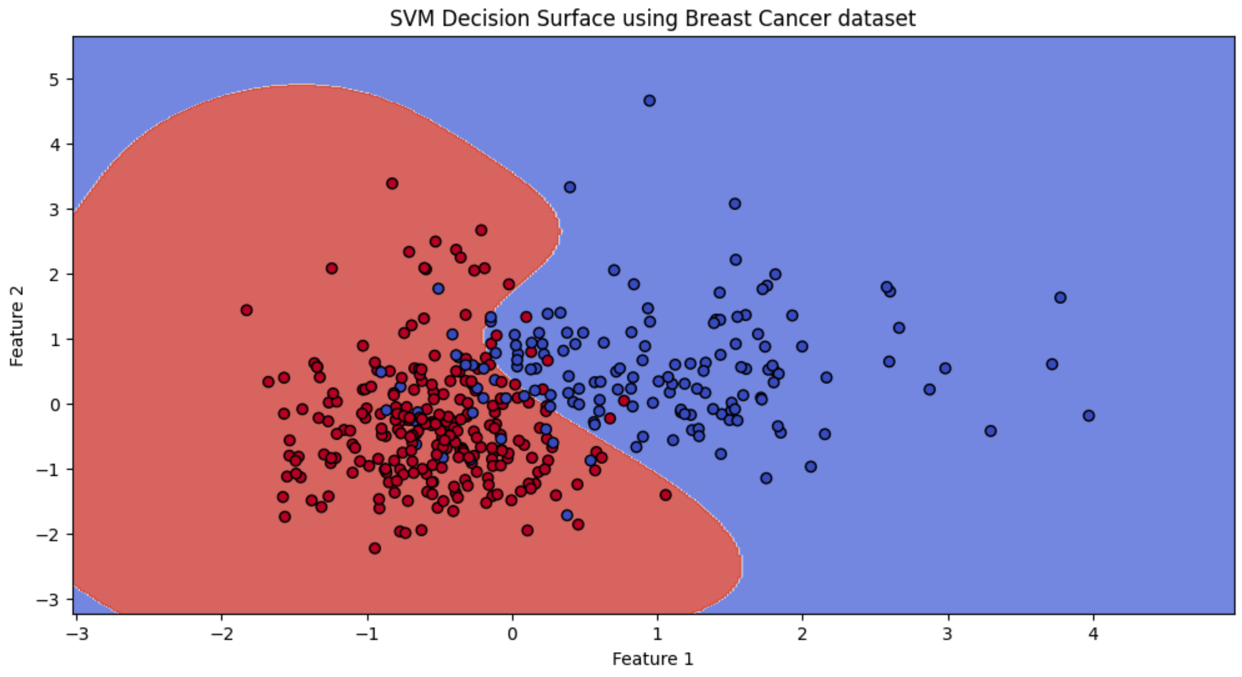
- **`best_params_`:**
  - **What It Is:** A dictionary containing the parameter value(s) that yielded the best performance during cross-validation.
  - **How It Works:**GridSearchCV tests multiple parameter combinations and selects the one with the highest average score across all folds.
  - **Example:** `{'n_neighbors': 7}` means that 7 neighbors gave the best results.
- **`best_score_`:**
  - **What It Is:** The best (highest) average cross-validation score achieved using the best parameters.
  - **How It Works:**During cross-validation, the model's performance is measured on several folds; `best_score_` is the mean score for the best

parameter setting.

  - **Example:** A value of 0.95 indicates a 95% accuracy (if using accuracy scoring) on average across folds.

- `test_accuracy`:
  - **What It Is:** The performance metric (e.g., accuracy) computed on a separate test set, using the best model found during GridSearchCV.
  - **How It Works:**After selecting the best parameters, you predict on the test set and calculate the accuracy (or other metric) using functions like `accuracy_score`.
  - **Example:** If `test_accuracy` is 0.93, then the model correctly predicts 93% of test cases.

Decision Surface/Boundary



- **Decision Boundary**: The hypersurface that partitions the feature space into regions assigned to different classes.

- **Decision Surface**: A higher-dimensional generalization of the decision boundary (e.g., in 3D space).

Key Characteristics

- **Separates Classes**: Points on one side of the boundary are classified differently from those on the other side.
- **Shape Depends on the Model**:
  - **Linear Models**: Straight lines/planes (e.g., logistic regression, linear SVM).
  - **Non-Linear Models**: Curved or complex boundaries (e.g., SVM with RBF kernel, decision trees, neural networks).

How Models Create Decision Boundaries

Model	Boundary Type	Mechanism
Logistic Regression	Linear	Sigmoid threshold at 0.5.
SVM (Linear Kernel)	Linear	Maximizes margin between classes.
SVM (Non-Linear Kernel)	Non-Linear	Maps data to higher dimensions for linear separation.
Decision Trees	Axis-aligned piecewise	Splits feature space recursively using feature thresholds.
k-Nearest Neighbors	Data-dependent	Majority class of the k*k* nearest neighbors defines local boundaries.
Neural Networks	Highly non-linear	Layers of neurons learn hierarchical feature transformations.