L03: KNN & K-Means

Mathematical Foundations and Implementation

Methods and Algorithms – MSc Data Science

KNN: The Lazy Learner

Key Insight

- No explicit model training (store all data)
- Classification by majority vote of K nearest neighbors
- "Lazy" because work is done at prediction time

The Algorithm

1. Store all training examples
2. For new query **x**: find K nearest training points
3. Return majority class among neighbors

Instance-based learning: the training data IS the model

Distance Metrics

02_distance_metrics/chart.pdf

Distance Metrics: Minkowski Family

Minkowski Distance

$$d_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

- $p = 1$: Manhattan (L1)
- $p = 2$: Euclidean (L2)
- $p = \infty$: Chebyshev (max absolute difference)

Choosing p

- $p = 2$: Default, works well in most cases
- $p = 1$: More robust to outliers
- Higher p: Sensitive to large single differences

In high dimensions, all distances become similar (curse of dimensionality)

KNN Decision Boundaries

01_knn_boundaries/chart.pdf

Choosing K

The Bias-Variance Trade-off

- $K = 1$: High variance, low bias (very flexible)
- $K = n$: High bias, low variance (always predicts majority class)

Practical Guidelines

- Start with $K = \sqrt{n}$ where n is training size
- Use odd K for binary classification (avoid ties)
- Cross-validation to find optimal K

Common Choices: $K \in \{3, 5, 7, 11\}$

Small K for complex patterns, larger K for noisy data

Weighted KNN

Problem with Equal Voting

- All K neighbors have equal influence
- A distant neighbor counts as much as closest neighbor

Solution: Distance Weighting

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^2}$$

- Closer neighbors get higher weight
- Reduces sensitivity to K choice

In scikit-learn

- `weights='uniform'`: equal weights (default)
- `weights='distance'`: inverse distance weighting

Distance weighting often improves performance

Feature Scaling for KNN

Why Scaling Matters

Without scaling:

- Income: ranges 20,000–200,000
- Age: ranges 20–80
- Distance dominated by income (larger scale)!

Scaling Methods

- **Standardization**: $z = \frac{x-\mu}{\sigma}$ (mean=0, std=1)
- **Min-Max**: $x' = \frac{x-x_{min}}{x_{max}-x_{min}}$ (range [0,1])

Rule: Always scale features for distance-based methods!

StandardScaler for Gaussian-like features, MinMaxScaler for bounded

KNN: Curse of Dimensionality

The Problem

In high dimensions:

- All points become approximately equidistant
- “Nearest neighbor” becomes meaningless
- Exponentially more data needed

Solutions

- **Dimensionality reduction**: PCA before KNN
- **Feature selection**: keep only relevant features
- **Use domain knowledge**: select meaningful features

KNN works best with moderate number of features (¡20)

KNN: Computational Considerations

Time Complexity

- Training: $O(1)$ – just store data!
- Prediction: $O(nd)$ for brute force (n samples, d features)

Acceleration Techniques

- **KD-Tree**: $O(d \log n)$ average for low d
- **Ball Tree**: Works better in higher dimensions
- **Approximate NN**: Trade accuracy for speed

In scikit-learn

- `algorithm='auto'`: automatically chooses best
- `algorithm='brute'`: force brute force

For large datasets: consider approximate methods or trees

KNN: scikit-learn Implementation

Classification

- `from sklearn.neighbors import KNeighborsClassifier`
- `knn = KNeighborsClassifier(n_neighbors=5)`
- `knn.fit(X_train, y_train)`
- `y_pred = knn.predict(X_test)`

Key Parameters

- `n_neighbors`: K value
- `weights`: 'uniform' or 'distance'
- `metric`: 'euclidean', 'manhattan', etc.
- `algorithm`: 'auto', 'ball_tree', 'kd_tree', 'brute'

Also available: KNeighborsRegressor for regression tasks

K-Means: The Idea

Goal: Partition n points into K clusters

Objective: Minimize within-cluster sum of squares (WCSS)

$$\sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \mu_k\|^2$$

where μ_k is the centroid of cluster C_k

Key Insight

- Each point assigned to nearest centroid
- Centroids are cluster means
- Iterative refinement until convergence

K-Means finds locally optimal solution (not guaranteed global)

K-Means Algorithm

```
Input: Data $\mathbf{X}$, number of clusters $K$
Initialize $K$ centroids randomly
repeat
    Assignment: assign each point to nearest centroid
    Update: recompute centroids as cluster means
until centroids don't change (or max iterations)
return cluster assignments, centroids
```

Convergence

- Guaranteed to converge (WCSS decreases each iteration)
- May converge to local optimum

Each iteration: $O(nKd)$ where n = samples, K = clusters, d = features

K-Means: Visualization

03_kmeans_iteration/chart.pdf

Initialization Strategies

Random Initialization

- Pick K random points as initial centroids
- Sensitive to choice, may get poor solution
- Run multiple times, keep best result

K-Means++ (Default in scikit-learn)

- Smart initialization: spread out initial centroids
- First centroid: random
- Next centroids: probability proportional to squared distance

Result: Much better starting point, fewer iterations

K-Means++ gives provably better initialization with theoretical guarantees

Choosing K: Elbow Method

04_elbow_method/chart.pdf

Choosing K: Silhouette Score

For each point i

- $a(i)$: average distance to points in same cluster
- $b(i)$: average distance to points in nearest other cluster

Silhouette score

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- Range: $[-1, 1]$
- $s \approx 1$: point is well-matched to cluster
- $s \approx 0$: point is on boundary
- $s < 0$: point may be in wrong cluster

Average silhouette score summarizes overall clustering quality

Silhouette Plot

05_silhouette/chart.pdf

K-Means: Decision Regions

06_voronoi/chart.pdf

K-Means Assumptions

What K-Means Assumes

- Clusters are spherical (isotropic)
- Clusters have similar sizes
- Clusters have similar densities

When K-Means Fails

- Non-convex shapes (e.g., crescents)
- Very different cluster sizes
- Different densities
- Outliers (pull centroids away)

Consider DBSCAN or Gaussian Mixture Models for these cases

K-Means Variants

Mini-Batch K-Means

- Uses random subsets for updates
- Much faster for large datasets
- Slightly worse results

K-Medoids

- Centroids must be actual data points
- More robust to outliers
- Slower than K-Means

K-Means for Mixed Data

- K-Modes: for categorical data
- K-Prototypes: mixed continuous and categorical

Mini-Batch K-Means: good for ¿10k samples

K-Means: scikit-learn

Basic Usage

- `from sklearn.cluster import KMeans`
- `kmeans = KMeans(n_clusters=3, random_state=42)`
- `labels = kmeans.fit_predict(X)`
- `centroids = kmeans.cluster_centers_`

Key Parameters

- `n_clusters`: K (required)
- `init`: 'k-means++' (default) or 'random'
- `n_init`: number of runs (default 10)
- `max_iter`: max iterations per run

inertia_ attribute gives WCSS after fitting

KNN vs K-Means: Key Differences

Aspect	KNN	K-Means
Task	Classification/Regression	Clustering
Learning	Supervised (needs labels)	Unsupervised
K meaning	Number of neighbors	Number of clusters
Training	None (lazy)	Iterative optimization
Prediction	Compute distances	Assign to centroid
Output	Class label	Cluster ID

The "K" in KNN and K-Means mean completely different things!

Finance Application: Customer Segmentation

Problem: Group customers for targeted marketing
Features

- Transaction frequency
- Average transaction amount
- Account balance
- Product holdings

K-Means Solution

- Cluster into segments (e.g., K=4)
- Profile each segment
- Tailor offerings to segment needs

Example segments: High-value frequent, Dormant, New/Growing, Price-sensitive

Finance Application: Fraud Detection

Problem: Flag suspicious transactions

KNN Approach

- Features: amount, time, location, merchant category
- Find K similar past transactions
- If most neighbors are fraud, flag as suspicious

K-Means Approach

- Cluster "normal" transactions
- New transaction far from all centroids = anomaly
- Combined with distance threshold

KNN needs labeled fraud examples, K-Means detects deviation from normal

When to Use What

Use KNN When

- You have labeled training data
- Local patterns matter (non-linear boundaries)
- Interpretability: “similar to these examples”
- Moderate dataset size

Use K-Means When

- No labels available
- Looking for natural groupings
- Clusters are roughly spherical
- Need fast clustering of large data

K-Means often used as preprocessing before supervised learning

Decision Framework

07_decision_flowchart/chart.pdf

Alternatives to Consider

Instead of KNN

- Large data: use ball tree or approximate NN
- Need probability: logistic regression
- Many features: random forest

Instead of K-Means

- Unknown K: DBSCAN (density-based)
- Hierarchical structure: agglomerative clustering
- Soft assignments: Gaussian Mixture Models
- Non-spherical: spectral clustering

DBSCAN automatically determines number of clusters

Key Takeaways

K-Nearest Neighbors

- Instance-based, lazy learner
- Scale features, choose K via cross-validation
- Works best with moderate features, moderate data size

K-Means

- Iterative: assign points, update centroids
- K-Means++ for initialization, elbow/silhouette for K
- Assumes spherical, similar-size clusters

Common Considerations

- Feature scaling is critical for both
- "K" means different things in each algorithm

Both are foundational algorithms: simple, interpretable, widely used

References

Textbooks

- James et al. (2021). *ISLR*, Chapters 2, 12
- Hastie et al. (2009). *ESL*, Chapters 13, 14

Key Papers

- Arthur & Vassilvitskii (2007). K-Means++
- Cover & Hart (1967). Nearest Neighbor Pattern Classification

Next Lecture

- L04: Random Forests
- Ensemble methods and feature importance