Nearest Neighbor-V

M. R. Hasan
Readings

• Bishop: 2.5
• Murphy: 1.4.1, 1.4.2, 1.4.3
• Alpaydin: 8.1, 8.2, 8.3, 8.4
What We Will Cover

• Time-complexity of K-NN Algorithms
• Various K-NN Algorithms
• Overcoming K-NN Limitations: Neighbourhood Components Analysis (NCA)
• K-NN: How Effective is It for Image Classification?
• K-NN: How do make K-NN as an Effective Technique for Image Classification?
• K-NN: Versatile Model (Classification & Regression)
• K-NN: Scikit-Learn Implementation
Practical Issues

- Distance metric (Euclidean, Manhattan, etc.)
- How many neighbors?
- Variance of the features are significantly different
- Non-zero covariance among some features
- Data is high-dimensional
- Complexity of K-NN algorithm & efficient solutions
K-NN: Time-Complexity Analysis
K-NN Time-Complexity

- We will discuss the **time-complexity of the K-NN model**.
- There are **various algorithms** to implement the K-NN model.
- We will discuss **3 variants of the K-NN algorithm** that are frequently used:
  - Brute-Force
  - k-d Tree
  - Ball Tree
K-NN Time-Complexity: Brute-Force

- So far our discussion was based on the basic K-NN algorithm that is **conceptually trivial**.
- Given a set of N examples and k, **calculate the L_p norm** (p = 1 or 2) from point of interest $x_i$ (of whose class we need to determine) to all the points $z_i$ in training set.
- **Sort the distances** and keep the best (closest) K.
- Then for $x_i$ choose the class with **majority from K points**.

$$L_p(\hat{x}, \bar{z}) := d(\hat{x}, \bar{z}) = \left[ \sum_{i=1}^{d} |x_i - z_i|^p \right]^{1/p}$$

This is called the **brute force** algorithm.
K-NN Time-Complexity: Brute-Force

• For N samples in d-dimensional feature the time-complexity turns out to be $O[dN^2]$. 

---

8
K-NN Time-Complexity: Brute-Force

• If we have small number of dimensions and small training set, brute-force algorithm would run in a reasonable time.
• But as the training set size increases, the running time grows quickly.
• Try running the brute-fore K-NN on the MNIST dataset (70,000 instances):
  http://deeplearning.net/tutorial/gettingstarted.html

Brute force algorithm performs worst when there are large dimensions and large training set.
K-NN Time-Complexity: Brute-Force

• Instance-based methods are designed for large data sets.
• So we would like an algorithm with sublinear run time.

• How do we improve the brute-force search time?
• What is the most efficient search algorithm paradigm?
• Divide and conquer!

• Recall that a binary tree provides O(logN) search time.
K-NN Time-Complexity: k-d Tree

• A **balanced binary tree** over data with an arbitrary number of dimensions is called a **k-d** tree, for **k-dimensional** tree.

• In our notation, the number of dimensions is **N**, so they would be **N-d** or **N-dimensional** trees.

• The construction of a k-d tree is similar to the construction of a one-dimensional balanced binary tree.
The time-complexity of k-d tree based K-NN is:

\[ O[d \, N * \log(N)] \]

This is a significant improvement over brute-force for large \( N \).

The k-d tree performs well enough when \( d < 20 \).

With larger \( d \), it again takes longer time ("curse of dimensionality").
The k-d trees are appropriate only when there are many more examples than dimensions.

Thus, k-d trees work well with up to 10 dimensions with thousands of examples or up to 20 dimensions with millions of examples.

If we don’t have enough examples, lookup is no faster than a linear scan of the entire data set.
K-NN Time-Complexity: Ball Tree

- The **Ball Tree** algorithm improves the complexity of the k-d tree algorithm.
- Where k-d trees partition data along Cartesian axes, ball trees partition data in a series of nesting hyper-spheres.
- This makes tree construction more costly than that of the k-d tree, but results in a data structure which can be very efficient on highly structured data, even in very high dimensions.
- Time-complexity is $O[d\log(N)]$. 
K-NN: Choosing Algorithms

• Which K-NN algorithm should we use?
• It depends on the number of dimensions and the size of training set.
• For small sample size and small dimensions, brute force performs well.
• Sparsity of data: If data is sparse (i.e., dimension is high so that all neighbors are far away) with small dimensions (< 20) k-d tree will perform better than Ball Tree algorithm.
• Value of K (neighbors): As the K increases, query time of both k-d tree and Ball tree increases.
K-NN: Overcoming Limitations
K-NN: Overcoming Limitations

- We present a technique to overcome two key limitations of K-NN.
  - Computational: K-NN needs to store the entire training dataset and compute distance with all training points.
  - The absence of an optimal distance metric in K-NN.

Neighbourhood Components Analysis (NCA) method for learning a Mahalanobis distance measure to be used in the K-NN classification

K-NN: Neighbourhood Component Analysis (NCA)

• The NCA method addresses the two limitations of K-NN by “learning” a distance metric for a given problem.

• NCA finds a linear transformation of input data such that the average leave-one-out (LOO) classification performance is maximized in the transformed space.

9 samples from 3 classes

Samples in the embedded space
K-NN: Neighbourhood Component Analysis (NCA)

- The learned Mahalanobis distance measure of NCA is used by a K-NN classifier model.
- The learned distance metric of NCA is low rank, thus substantially reduces storage and search costs at test time.

![Diagram showing 9 samples from 3 classes in the original points and the embedded space using NCA.](image)
K-NN: Neighbourhood Component Analysis (NCA)

- See the following GitHub notebook for an empirical understanding of the comparison of the K-NN on the transformed feature space of NCA and the original feature space.
K-NN: How Effective is It for Image Classification?
K-NN: Image Classification

- The analogy based learning models such as K-Nearest Neighbors (K-NN) use **distance similarity metric** to classify images.

Image pixels are considered as **features**.
K-NN: Image Classification

- Two images are “similar” (share the same semantic identity or class label) if their pixel-wise Minkowski distance is small.
K-NN: Image Classification

• **MNIST**: the inter-class distance is larger than the intra-class distance.

• However, this is not always true (i.e., for other datasets).
K-NN: Image Classification

- **CIFAR-10**: There is no significant difference between the inter-class distance and intra-class distance.
K-NN: Image Classification

- The distance-based metrics (e.g., Minkowski) are **not effective**, when applied at **pixel level**, for determining similarity between the images.
K-NN: Image Classification

- See the following GitHub notebook for an empirical understanding of **when K-NN is effective for image classification** and when it’s not, and why.
K-NN: How Do We use K-NN as an Effective Technique for Image Classification?
K-NN: Is Analogy based Learning Effective?

- The distance metric does not provide a reliable measure, when applied at the pixel level, to determine semantic identity of an image.
- It doesn’t mean that the distance based technique or analogy based reasoning (e.g., K-NN) in general is flawed/weak.
K-NN: Is Analogy based Learning Effective?

- The similarity based approach is effective when more expressive and powerful high-level features are extracted from raw pixels.
- In other words, while distance measures at the raw pixel level produce spurious results, similarity calculation on the high-level features may reveal semantic identity effectively.
• How do we create more expressive and high-level features from raw pixels?

• A **Neural Network** model can be used to learn high-level features from the raw pixels.

• The high-level features are **informative** to determine the semantic identity.
K-NN: Is Analogy based Learning Effective?

- The following papers achieve state-of-the-art results on image classification by applying the K-NN analogy based approach on the learned features.
K-NN: Versatile ML Model (Classification & Regression)
K-NN: Versatile ML Model

- K-NN is a **versatile** ML model.
- We can use K-NN for both classification and **regression** problems.
- See the following GitHub notebook for an empirical understanding of how to use K-NN for solving regression problems.
K-NN: Scikit-Learn Implementation
K-NN: How to Use Scikit-Learn

- Scikit-Learn provides implementation for K-Nearest Neighbors Classifier.

```python
class sklearn.neighbors. KNeighborsClassifier (n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

- **n_neighbors**: (default = 5)
  Number of neighbors to use

- **weights**: (default = ‘uniform’)
  - ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
  - ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
K-NN: How to Use Scikit-Learn

```python
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

**algorithm**: {'auto', 'ball_tree', 'kd_tree', 'brute'}
Algorithm used to compute the nearest neighbors:
- ‘ball_tree’ will use BallTree
- ‘kd_tree’ will use KDTree
- ‘brute’ will use a brute-force search.
- ‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method.

**p**: (default = 2)
Power parameter for the Minkowski metric.
- p = 1 manhattan_distance (l1)
- p = 2 euclidean_distance (l2)
- For arbitrary p, minkowski_distance (l_p)
K-NN: How to Use Scikit-Learn

```python
class sklearn.neighbors.KNeighborsClassifier (n_neighbors=5, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

**metric**: (default ‘minkowski’)
- the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric.
K-NN: How to Use Scikit-Learn

```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(features_matrix, labels)
predicted_values = knn.predict(test_matrix)
```

- **n_neighbors**: default = 5
- **weights**: default = ‘uniform’
- **algorithm**: optional
- **p**: default = 2
- **metric**: default ‘minkowski’
What We Need to Remember

- Classification Problem
- Supervised Algorithm for Classification
- Nearest Neighbor
- K-NN
- Limitations of K-NN
- Weighted K-NN
- How to use K-NN effectively
- K-NN Regressor

K-NN is not a “learning” algorithm

There is no function or model

Store the entire data set and compare each new point against it for classification