CSE 417T: Introduction to Machine Learning

Lecture 19: $k$-Nearest Neighbors

Henry Chai
11/06/18
Recall

- Classify a point as the label of the “most similar” training point

- Euclidean distance: \( d(\vec{x}, \vec{x}') = ||\vec{x} - \vec{x}'|| = \sum_{j=1}^{D} (x_j - x'_j)^2 \)

- Given \( \mathcal{D} = \{(x_1^1, y_1), (x_2^2, y_2), \ldots, (x_n^n, y_n)\} \) and a point \( \vec{x} \), let \( \vec{x}_{[i]}(\vec{x}) \) be the \( i^{th} \) closest point to \( \vec{x} \) in \( \mathcal{D} \)
The Nearest Neighbor Hypothesis

\[ g(\hat{x}) = y_{[1]}(\hat{x}) \]
Generalization of Nearest Neighbor

- Claim: $E_{out}$ for the nearest neighbor hypothesis is not much worse than the best possible $E_{out}$!
- Formally: under certain conditions, with high probability, $E_{out}(g) \leq 2E_{out}(g^*)$ as $n \to \infty$
- Interpretation: half of the data’s predictive power is in the nearest neighbor!
- Classify a point as the most common label among the labels of the $k$ nearest training points
- If we have a binary classification problem and $k$ is odd:
  \[ g(\bar{x}) = \text{sign} \left( \sum_{i=1}^{k} y_{[i]}(\bar{x}) \right) \]
- $k$ controls the complexity of the hypothesis set $\implies k$ affects how well the learned hypothesis will generalize
  - $k = 3$
  - $k = \lfloor \sqrt{n} \rfloor$
  - Cross-validation
**kNN Pros and Cons**

**Pros:**
- Intuitive / explainable
- No training / retraining
- Self-regularizes
- Provably near-optimal in terms of $E_{out}$

**Cons:**
- Computationally expensive
  - Always needs to store all data: $O(nD)$
  - Computing $g(\tilde{x})$ requires computing $d(\tilde{x}, \tilde{x}') \forall \tilde{x}' \in \mathcal{D}$ and finding the $k$ closest points: $O(nD + n \log(k))$
- Suffers from the “curse of dimensionality”
Curse of Dimensionality

- The fundamental assumption of kNN is that “similar” points or points close to one another should have the same label.

- The closer two points are, the more confident we can be that they will have the same label.

- As the number of dimensions the input has grows, the less likely it is that two random points will be close.

- As the number of dimensions the input has grows, it takes more points to “cover” the input space.
Curing the Curse of Dimensionality

- More data
- Fewer dimensions
- Blessing of non-uniformity: data from the real world is rarely uniformly distributed across the input space
Computational Cost of $k$NN

- No training required!
- Memory: $O(nD)$
- Computing $g(\bar{x})$: $O(nD + n \log(k))$

Idea: preprocess inputs in order to speed up predictions
- Reduce the number of inputs held in memory by eliminating redundancies
- Organize inputs in data structures that make searching for nearest neighbors more efficient
Data Condensing

- Reduce the number of inputs while maintaining the same decision boundary
- Let $g_D$ be the kNN hypothesis when trained on $\mathcal{D}$
- $S \subseteq \mathcal{D}$ is decision-boundary consistent if:
  $$g_S(\vec{x}) = g_D(\vec{x}) \forall \vec{x} \in \mathcal{X}$$
- Decision-boundary consistent subsets are computationally expensive to find
Data Condensing

- Reduce the number of inputs while maintaining the same predictions on all inputs
- Let $g_D$ be the kNN hypothesis when trained on $\mathcal{D}$
- $S \subseteq \mathcal{D}$ is training-set consistent if:
  $$g_S(x_i) = g_D(x_i) \forall x_i \in \mathcal{D}$$
- Training-set consistent is a much weaker constraint than decision-boundary consistent

From the point of view of the training data, these rules are the same
In the worst case, this algorithm will terminate after \( n \) iterations and the returned set \( S \) will just be the entire dataset \( \mathcal{D} \).

Finding a minimum training-set consistent set is NP-hard.
If at some iteration of the CNN algorithm, the circled point is chosen as the point where $g_S(\vec{x}_j) \neq g_D(\vec{x}_j)$, then the CNN algorithm will choose to add that point to the set $S$. 

$k = 3$

$g_S(\vec{x}) = -1, g_D(\vec{x}) = +1$
If at some iteration of the CNN algorithm, the circled point is chosen as the point where $g_S(x_i) \neq g_D(x_i)$, then the CNN algorithm will choose to add the green plus to the right of the circled point to the set $S$. 

$$g_S(x) = -1, g_D(x) = +1$$
Organizing the Inputs

- Intuition: split the inputs into clusters, groups of points that are close to one another but far from other groups.
- If an input point is really close to one group of points and really far from the other groups ...
- ... then we can skip searching through the other groups and just look for nearest neighbors in the close group!
- Questions:
  - What does it mean for a point to be close to a group?
  - How can we split the input into clusters?
Organizing the Inputs

Let \( \hat{x}^i_{[1]}(\hat{x}) \) be the \( i \)th closest point to \( \hat{x} \) in the cluster \( S_j \)

- \( \| \hat{x} - \hat{\mu}_j \| - r_j \leq \| \hat{x} - \hat{x}^i_{[1]}(\hat{x}) \| \)
- If \( \exists S_i \) s.t. \( \| \hat{x} - \hat{x}^i_{[1]}(\hat{x}) \| \leq \| \hat{x} - \hat{\mu}_j \| - r_j \) ...
- ... then we don’t need to search \( S_j \) for the nearest neighbor
Organizing the Inputs

Let $\hat{x}_{[i]}^j(\bar{x})$ be the $i^{th}$ closest point to $\bar{x}$ in the cluster $S_j$.

- $\|\bar{x} - \hat{x}_j\| - r_j \leq \|\bar{x} - \hat{x}_{[1]}^j(\bar{x})\| \leq ... \leq \|\bar{x} - \hat{x}_{[k]}^j(\bar{x})\|$
- If $\exists S_i$ s.t. $\|\bar{x} - \hat{x}_{[k]}^i(\bar{x})\| \leq \|\bar{x} - \hat{x}_j\| - r_j$ ...
- ... then we don’t need to search $S_j$ for the $k$ nearest neighbors.
Considering the case when $\bar{x} \approx \bar{\mu}_i$ is important because we hope that our sufficiency condition is met in these situations.
We're going to use the intuition about what we want out of the clusters as determined by the previous slide and try and build a simple clustering algorithm. There are a lot of clustering algorithms, some more complicated than others (see 517A)

\( M \) will be a set that holds the temporary cluster centers

The distance between a point and the set \( M \) is simply the distance between that point and its nearest neighbor in \( M \)

The first for loop tries to make the cluster centers as far as possible while the last for loop tries to minimize the radius
This procedure only makes sense if you have lots of query points that you have to make predictions about.
Decision trees are non-parametric

In nonparametric models, the complexity of the model grows with the number of training data; in parametric models, we have a fixed number of parameters.