Large-Scale Nearest Neighbor Classification with Statistical Guarantees

Guang Cheng

Big Data Theory Lab
Department of Statistics
Purdue University

Joint Work with Xingye Qiao and Jiexin Duan

July 3, 2018
SUSY Data Set (UCI Machine Learning Repository)

- Size: 5,000,000 particles from the accelerator
- Predictor variables: 18 (properties of the particles)
- Goal – Classification:
  distinguish between a signal process and a background process
The $k$NN classifier predicts the class of $x \in \mathbb{R}^d$ to be the most frequent class of its $k$ nearest neighbors (Euclidean distance).
Computational/Space Challenges for kNN Classifiers

- Time complexity: $O(dN + N\log(N))$
  - $dN$: computing distances from the query point to all $N$ observations in $\mathbb{R}^d$
  - $N\log(N)$: sorting $N$ distances and selecting the $k$ nearest distances
- Space complexity: $O(dN)$
- Here, $N$ is the size of the entire dataset
How to Conquer "Big Data?"
Train the total data at one time: oracle $k$NN

**Figure:** I’m Pac-Superman!
To build a Pac-superman for oracle-$k$NN, we need:

- Expensive super-computer
To build a Pac-superman for oracle-$k$NN, we need:

- Expensive super-computer
- Operation system, e.g. Linux
To build a Pac-superman for oracle-\(k\)-NN, we need:

- Expensive super-computer
- Operation system, e.g. Linux
- Statistical software designed for big data, e.g. Spark, Hadoop
To build a Pac-superman for oracle-\(k\)NN, we need:

- Expensive super-computer
- Operation system, e.g. Linux
- Statistical software designed for big data, e.g. Spark, Hadoop
- Write complicated algorithms (e.g. MapReduce) with limited extendibility to other classification methods
In machine learning area, there are some nearest neighbor algorithms for big data already

- Muja&Lowe(2014) designed an algorithm to search nearest neighbors for big data
- Comments: complicated; without statistical guarantee; lack of extendibility

We hope to design an algorithm framework:

- Easy to implement
- Strong statistical guarantee
- More generic and expandable (e.g. easy to be applied to other classifiers (decision trees, SVM, etc))
If We Don’t Have A Supercomputer…

What can we do?
What about splitting the data?
Goal: deliver *different* insights from regression problems.
Big-$k$NN for SUSY Data

- **Oracle-$k$NN**: $k$NN trained by the entire data
- **Number of subsets**: $s = N^\gamma$, $\gamma = 0.1, 0.2, \ldots, 0.8$

![Graph showing risk and speedup for Big-$k$NN](image)

**Figure**: Risk and Speedup for Big-$k$NN.

- **Risk of classifier** $\phi$: $R(\phi) = \mathbb{P}(\phi(X) \neq Y)$
- **Speedup**: running time ratio between Oracle-$k$NN & Big-$k$NN
Construction of Pac-Man (Big Data)
We need a *statistical* understanding of Big-$k$NN....
In a dataset with size $n$, the WNN classifier assigns a weight $w_{ni}$ on the $i$-th neighbor of $x$:

$$
\hat{\phi}_n^{wn}(x) = 1 \left\{ \sum_{i=1}^{n} w_{ni} \mathbb{1}\{Y(i) = 1\} \geq \frac{1}{2} \right\} \text{ s.t. } \sum_{i=1}^{n} w_{ni} = 1,
$$

When $w_{ni} = k^{-1} \mathbb{1}\{1 \leq i \leq k\}$, WNN reduces to $k$NN.
In a big data set with size $N = sn$, the Big-WNN is constructed as:

$$\hat{\phi}_{n,s}^{\text{Big}}(x) = 1 \left\{ s^{-1} \sum_{j=1}^{s} \hat{\phi}_n^{(j)}(x) \geq 1/2 \right\},$$

where $\hat{\phi}_n^{(j)}(x)$ is the local WNN in $j$-th subset.
Primary criterion: accuracy

- Regret $=$ Expected Risk $-$ Bayes Risk $= \mathbb{E}_D \left[ R(\hat{\phi}_n) \right] - R(\phi^{\text{Bayes}})$
- A small value of regret is preferred
Primary criterion: accuracy
- Regret = Expected Risk – Bayes Risk = $\mathbb{E}_{D} \left[ R(\hat{\phi}_n) \right] - R(\phi^{\text{Bayes}})$
- A small value of regret is preferred

Secondary criterion: stability (Sun et al, 2016, JASA)

Definition: Classification Instability (CIS)
Define classification instability of a classification procedure $\Psi$ as

$$\text{CIS}(\Psi) = \mathbb{E}_{D_1, D_2} \left[ \mathbb{P}_X \left( \hat{\phi}_{n_1}(X) \neq \hat{\phi}_{n_2}(X) \right) \right],$$

where $\hat{\phi}_{n_1}$ and $\hat{\phi}_{n_2}$ are the classifiers trained from the same classification procedure $\Psi$ based on $D_1$ and $D_2$ with the same distribution.

- A small value of CIS is preferred
Theorem (Regret of Big-WNN)

Under regularity assumptions, with $s$ upper bounded by the subset size $n$, we have as $n, s \to \infty$,

$$\text{Regret (Big-WNN)} \approx B_1 s^{-1} \sum_{i=1}^{n} w_{ni}^2 + B_2 \left( \sum_{i=1}^{n} \frac{\alpha_i w_{ni}}{n^2/d} \right)^2,$$

where $\alpha_i = i^{1+\frac{2}{d}} - (i-1)^{1+\frac{2}{d}}$, $w_{ni}$ are the local weights. The constants $B_1$ and $B_2$ are distribution-dependent quantities.
Asymptotic Regret of Big-WNN

**Theorem (Regret of Big-WNN)**

Under regularity assumptions, with $s$ upper bounded by the subset size $n$, we have as $n, s \to \infty$,

$$\text{Regret (Big-WNN)} \approx B_1 s^{-1} \sum_{i=1}^{n} w_{ni}^2 + B_2 \left( \sum_{i=1}^{n} \frac{\alpha_i w_{ni}}{n^2/d} \right)^2,$$

where $\alpha_i = i^{1+\frac{2}{d}} - (i - 1)^{1+\frac{2}{d}}$, $w_{ni}$ are the local weights. The constants $B_1$ and $B_2$ are distribution-dependent quantities.
Asymptotic Regret of Big-WNN

Theorem (Regret of Big-WNN)

Under regularity assumptions, with $s$ upper bounded by the subset size $n$, we have as $n, s \to \infty$,

$$\text{Regret (Big-WNN)} \approx B_1 s^{-1} \sum_{i=1}^{n} w_{ni}^2 + B_2 \left( \sum_{i=1}^{n} \frac{\alpha_i w_{ni}}{n^2/d} \right)^2,$$

where $\alpha_i = i^{1+\frac{2}{d}} - (i-1)^{1+\frac{2}{d}}$, $w_{ni}$ are the local weights. The constants $B_1$ and $B_2$ are distribution-dependent quantities.

Variance part

Bias part
Asymptotic Regret Comparison

Round 1

Big-WNN

Oracle-WNN

Regret

Q
Theorem (Regret Ratio of Big-WNN and Oracle-WNN)

Given an Oracle-WNN classifier, we can always design a Big-WNN by adjusting its weights according to those in the oracle version s.t.

\[
\frac{\text{Regret (Big-WNN)}}{\text{Regret (Oracle-WNN)}} \to Q,
\]

where \( Q = \left( \frac{\pi}{2} \right)^{\frac{4}{d+4}} \) and \( 1 < Q < 2 \).
Asymptotic Regret Comparison

Theorem (Regret Ratio of Big-WNN and Oracle-WNN)

Given an Oracle-WNN classifier, we can always design a Big-WNN by adjusting its weights according to those in the oracle version s.t.

\[
\frac{\text{Regret (Big-WNN)}}{\text{Regret (Oracle-WNN)}} \to Q,
\]

where \( Q = \left(\frac{\pi}{2}\right)^{\frac{4}{d+4}} \) and \( 1 < Q < 2 \).

- E.g., in Big-\(k\)NN case, we set \( k = \lceil \left(\frac{\pi}{2}\right)^{\frac{d}{d+4}} \frac{k^O}{s} \rceil \) given the oracle \( k^O \). We need a multiplicative constant \( \left(\frac{\pi}{2}\right)^{\frac{d}{d+4}} \)!
Theorem (Regret Ratio of Big-WNN and Oracle-WNN)

Given an Oracle-WNN classifier, we can always design a Big-WNN by adjusting its weights according to those in the oracle version s.t.

$$\frac{\text{Regret (Big-WNN)}}{\text{Regret (Oracle-WNN)}} \rightarrow Q,$$

where $Q = \left(\frac{\pi}{2}\right)^{\frac{4}{d+4}}$ and $1 < Q < 2$.

- E.g., in Big-kNN case, we set $k = \left\lfloor \left(\frac{\pi}{2}\right)^{\frac{d}{d+4}} \frac{k^O}{s} \right\rfloor$ given the oracle $k^O$. We need a multiplicative constant $\left(\frac{\pi}{2}\right)^{\frac{d}{d+4}}$!
- We name $Q$ the Majority Voting (MV) Constant.
MV constant monotonically decreases to one as \( d \) increases.

For example:
- \( d=1, \quad Q=1.44 \)
- \( d=2, \quad Q=1.35 \)
- \( d=5, \quad Q=1.22 \)
- \( d=10, \quad Q=1.14 \)
- \( d=20, \quad Q=1.08 \)
- \( d=50, \quad Q=1.03 \)
- \( d=100, \quad Q=1.02 \)
Why is there statistical accuracy loss?
Accuracy loss due to the transformation from (continuous) percentage to (discrete) 0-1 label
Majority Voting/Accuracy Loss Once in Oracle Classifier

Only one majority voting in oracle classifier

Figure: I’m Pac-Superman!
Majority Voting/Accuracy Loss Twice in D&C Framework

1st loss

2nd loss
Is it possible to apply majority voting once in D&C?
Only Loss Once in the Continuous Version

Guang Cheng
Big Data Theory Lab@Purdue
Construction of Pac-Batman (Continuous Version)
Definition: C-Big-WNN

In a data set with size $N = sn$, the C-Big-WNN is constructed as:

$$
\hat{\phi}_{n,s}^{CBig}(x) = 1 \left\{ \frac{1}{s} \sum_{j=1}^{s} \hat{S}_n^{(j)}(x) \geq 1/2 \right\},
$$

where $\hat{S}_n^{(j)}(x) = \sum_{i=1}^{n} w_{ni,j} Y_{(i),j}(x)$ is the weighted average for nearest neighbors of query point $x$ in $j$-th subset.
Continuous Version of Big-WNN (C-Big-WNN)

Definition: C-Big-WNN

In a data set with size \( N = sn \), the C-Big-WNN is constructed as:

\[
\hat{\phi}_{n,s}^{CBig}(x) = 1 \left\{ \frac{1}{s} \sum_{j=1}^{s} \hat{S}_n^{(j)}(x) \geq 1/2 \right\},
\]

where \( \hat{S}_n^{(j)}(x) = \sum_{i=1}^{n} w_{ni,j} Y_{(i),j}(x) \) is the weighted average for nearest neighbors of query point \( x \) in \( j \)-th subset.

Step 1:
Percentage calculation
**Definition: C-Big-WNN**

In a data set with size $N = sn$, the C-Big-WNN is constructed as:

$$
\hat{\phi}_{CBig}^{n,s}(x) = 1 \left\{ \frac{1}{s} \sum_{j=1}^{s} \hat{S}_n^{(j)}(x) \geq 1/2 \right\},
$$

where $\hat{S}_n^{(j)}(x) = \sum_{i=1}^{n} w_{ni,j} Y_{(i),j}(x)$ is the weighted average for nearest neighbors of query point $x$ in $j$-th subset.

**Step 1:** Percentage calculation

**Step 2:** Majority voting
Figure: Risk and Speedup for Big-$k$NN and C-Big-$k$NN.
Asymptotic Regret Comparison

Round 2

C-BigWNN

Oracle-WNN

Regret

Regret

1
Theorem (Regret Ratio between C-Big-WNN and Oracle-WNN)

Given an Oracle-WNN, we can always design a C-Big-WNN by adjusting its weights according to those in the oracle version s.t.

\[
\frac{\text{Regret (C-Big-WNN)}}{\text{Regret (Oracle-WNN)}} \rightarrow 1,
\]
Theorem (Regret Ratio between C-Big-WNN and Oracle-WNN)

Given an Oracle-WNN, we can always design a C-Big-WNN by adjusting its weights according to those in the oracle version s.t.

\[
\frac{\text{Regret (C-Big-WNN)}}{\text{Regret (Oracle-WNN)}} \rightarrow 1,
\]

E.g., in the C-Big-kNN case, we can set \( k = \left\lfloor \frac{k^O}{s} \right\rfloor \) given the oracle \( k^O \). There is no multiplicative constant!
Theorem (Regret Ratio between C-Big-WNN and Oracle-WNN)

Given an Oracle-WNN, we can always design a C-Big-WNN by adjusting its weights according to those in the oracle version s.t.

\[
\frac{\text{Regret (C-Big-WNN)}}{\text{Regret (Oracle-WNN)}} \rightarrow 1,
\]

- E.g., in the C-Big-kNN case, we can set \( k = \lfloor \frac{k^O}{s} \rfloor \) given the oracle \( k^O \). There is no multiplicative constant!
- Note that this theorem can be directly applied to the optimally weighted NN (Samworth, 2012, AoS), i.e., OWNN, whose weights are optimized to lead to the minimal regret.
It is time consuming to apply OWNN in each subset although it leads to the minimal regret.
Practical Consideration: Speed vs Accuracy

- It is time consuming to apply OWNN in each subset although it leads to the minimal regret.
- Rather, it is more easier and faster to implement $k$NN in each subset (with accuracy loss, though).
Practical Consideration: Speed vs Accuracy

- It is time consuming to apply OWNN in each subset although it leads to the minimal regret.
- Rather, it is more easier and faster to implement $k$NN in each subset (with accuracy loss, though).
- What is the statistical price (in terms of accuracy and stability) in order to trade accuracy with speed?
Theorem (Statistical Price)

We can design an optimal C-Big-kNN by setting its weights as

\[ k = \left\lfloor \frac{k^{O, \text{opt}}}{s} \right\rfloor \text{ s.t.} \]

\[ \frac{\text{Regret(Optimal C-Big-kNN)}}{\text{Regret(Oracle-OWNN)}} \rightarrow Q', \]

\[ \frac{\text{CIS(Optimal C-Big-kNN)}}{\text{CIS(Oracle-OWNN)}} \rightarrow \sqrt{Q'}, \]

where \( Q' = 2^{-4/d+4} \left( \frac{d+4}{d+2} \right) (2d+4)/(d+4) \) and \( 1 < Q' < 2 \). Here, \( k^{O, \text{opt}} \) minimizes the regret of Oracle-kNN.

Interestingly, both \( Q \) and \( Q' \) only depend on data dimension!
$Q'$ converges to one as $d$ grows in a unimodal way

For example:

- $d=1$, $Q=1.06$
- $d=2$, $Q=1.08$
- $d=5$, $Q=1.08$
- $d=10$, $Q=1.07$
- $d=20$, $Q=1.04$
- $d=50$, $Q=1.02$
- $d=100$, $Q=1.01$

The worst dimension is $d^* = 4 \implies Q^* = 1.089$. 
Consider the classification problem for Big-$k$NN and C-Big-$k$NN:

- Sample size: $N = 27,000$
- Dimensions: $d = 6,8$
- $P_0 \sim N(0_d, I_d)$ and $P_1 \sim N(\frac{2}{\sqrt{d}}1_d, I_d)$
- Prior class probability: $\pi_1 = Pr(Y = 1) = 1/3$
- Number of neighbors in Oracle-$k$NN: $k^O = N^{0.7}$
- Number of subsamples in D&C: $s = N^{\gamma}$, $\gamma = 0.1, 0.2, \ldots 0.8$
- Number of neighbors in Big-$k$NN: $k^d = \lceil (\frac{\pi}{2})^{\frac{d}{2d+4}} \frac{k^O}{s} \rceil$
- Number of neighbors in C-Big-$k$NN: $k^c = \lceil \frac{k^O}{s} \rceil$
Figure: Empirical Risk (Testing Error). Left/Right: $d = 6/8$. 
Figure: Running Time. Left/Right: $d = 6/8$. 
Simulation Analysis – Empirical Regret Ratio

Figure: Empirical Ratio of Regret. Left/Right: $d = 6/8$.

- **Q**: $\text{Regret(Big-kNN)}/\text{Regret(Oracle-kNN)}$ or $\text{Regret(C-Big-kNN)}/\text{Regret(Oracle-kNN)}$
**Figure:** Empirical CIS. Left/Right: $d = 6/8$. 
<table>
<thead>
<tr>
<th>Data</th>
<th>Size</th>
<th>Dim</th>
<th>Big-kNN</th>
<th>C-Big-kNN</th>
<th>Oracle-kNN</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>htru2</td>
<td>17898</td>
<td>8</td>
<td>3.72</td>
<td>3.36</td>
<td><strong>3.34</strong></td>
<td>21.27</td>
</tr>
<tr>
<td>gisette</td>
<td>6000</td>
<td>5000</td>
<td>12.86</td>
<td><strong>10.77</strong></td>
<td>10.78</td>
<td>12.83</td>
</tr>
<tr>
<td>musk1</td>
<td>476</td>
<td>166</td>
<td>36.43</td>
<td><strong>33.87</strong></td>
<td>35.71</td>
<td>3.6</td>
</tr>
<tr>
<td>musk2</td>
<td>6598</td>
<td>166</td>
<td>10.43</td>
<td><strong>9.98</strong></td>
<td>10.14</td>
<td>14.68</td>
</tr>
<tr>
<td>occup</td>
<td>20560</td>
<td>6</td>
<td>5.09</td>
<td><strong>4.87</strong></td>
<td>5.19</td>
<td>20.31</td>
</tr>
<tr>
<td>credit</td>
<td>30000</td>
<td>24</td>
<td>21.85</td>
<td><strong>21.37</strong></td>
<td>21.5</td>
<td>22.44</td>
</tr>
<tr>
<td>SUSY</td>
<td>5000000</td>
<td>18</td>
<td>30.66</td>
<td>30.08</td>
<td><strong>30.01</strong></td>
<td>68.45</td>
</tr>
</tbody>
</table>

**Table:** Test error (Risk): Big-$k$NN compared to Oracle-$k$NN in real datasets. Best performance is shown in bold-face. The speedup factor is defined as computing time of Oracle-$k$NN divided by the time of the slower Big-$k$NN method. Oracle $k = N^{0.7}$, number of subsets $s = N^{0.3}$. 
Thank you!