Autonomous Fast Classifiers For Pharmaceutical Datasets

Paul Komarek and Ting Liu
Auton Lab
komarek@cmu.edu

School of Computer Science, Carnegie Mellon University
Papers and software available at http://autonlab.org
We will discuss two traditional classifiers, logistic regression (LR) and k nearest-neighbor (KNN).
We will discuss two traditional classifiers, logistic regression (LR) and k nearest-neighbor (KNN).

Both have been described as slow.
We will discuss two traditional classifiers, logistic regression (LR) and k nearest-neighbor (KNN).

Both have been described as slow.

They don’t have be slow.
We will discuss two traditional classifiers, logistic regression (LR) and k nearest-neighbor (KNN).

Both have been described as slow.

They don’t have to be slow.

I will start with my work on accelerating LR, and then discuss briefly Ting Liu’s work on KNN.
Introduction To Our Work

Automated telescopes, roboticized chemistry, and the computerization of business processes are generating immense volumes of data. Is this data useful?

- Humans cannot process this data without help.
- Many popular data analysis techniques do not scale to the size of modern datasets.
- Commonly-used, well-understood statistical tools make conclusions more trustworthy.
- Through computer science, statistics, mathematics, and computers, we can continue to use our favorite tools.

Our goal is to produce algorithms, data structures, and reference implementations that allow scientists and decision-makers to apply their core competencies to modern data volumes.
Data Mining, Classification

**Data mining:** Finding interesting records in data, or finding relations between attributes of the dataset.

- Which aspects our factory have the greatest effect on product quality?
- Which molecules from our library are most likely to function as catalysts for a certain reaction?

**Classification:** Deciding if a test data point belongs to $\text{class}_1, \text{class}_2, \ldots, \text{class}_n$, using information from previous data. We will only discuss binary classification in this talk.

- Will this molecule inhibit viral reproduction?
- Did this rock form in the presence of water?
Dataset Notation

Terms:
- Datasets have *inputs* and *outputs*
- Inputs are the data points or experiments
- Outputs are the results of the experiments; the class label

Dimensions:
- $M$ is the number of columns, a.k.a. attributes
- $R$ is the number of rows, a.k.a. records
- $F$ is the percentage of nonzero elements, a.k.a. sparsity

Matrix notation:
- $X$ is the matrix of inputs
- $y$ is the vector of outputs
Datasets: Just how big are they?

<table>
<thead>
<tr>
<th>Name</th>
<th>Attributes</th>
<th>Rows</th>
<th>Sparsity</th>
<th>Nonzero</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>citeseer</td>
<td>105,354</td>
<td>181,395</td>
<td>0.00002</td>
<td>512,267</td>
<td>299</td>
</tr>
<tr>
<td>imdb</td>
<td>685,569</td>
<td>167,773</td>
<td>0.00002</td>
<td>2,442,721</td>
<td>824</td>
</tr>
<tr>
<td>ds2</td>
<td>1,143,054</td>
<td>88,358</td>
<td>0.00029</td>
<td>29,861,146</td>
<td>423</td>
</tr>
<tr>
<td>ds1</td>
<td>6,348</td>
<td>26,733</td>
<td>0.02199</td>
<td>3,732,607</td>
<td>804</td>
</tr>
<tr>
<td>ds1.100pca</td>
<td>100</td>
<td>26,733</td>
<td>1.00000</td>
<td>2,673,300</td>
<td>804</td>
</tr>
<tr>
<td>ds1.10pca</td>
<td>10</td>
<td>26,733</td>
<td>1.00000</td>
<td>267,330</td>
<td>804</td>
</tr>
</tbody>
</table>

citeseer and imdb are link detection datasets.

ds1 and ds2 are life-sciences datasets, similar to those of the National Cancer Institute. ds1.100pca and ds1.10pca are projections onto the first 100 and 10 principal components of ds1.
Classifiers

Why not use one of the popular existing classifiers on this data?

- Bayes’ Classifier: Very fast, but scores poorly. Makes unrealistic assumptions and doesn’t directly approximate class probabilities.

- K-Nearest-Neighbor: Slow. Has trouble representing linear boundaries. Sensitive to small changes in data.

- Decision Trees: Medium speed. Has trouble representing linear boundaries. Binary partitioning leads to high variance.

- Support Vector Machines: Theoretically slow (requires solution to quadratic program), but heuristics make them fast in many cases. Generally good predictions, considered state of the art.

We will describe these classifiers in more detail later.
Why haven’t we been using logistic regression?
LR expectation function:

\[ \mu_i = \mu(x_i, \beta) = \frac{\exp(\beta^T x_i)}{1 + \exp(\beta^T x_i)} \]

LR model:

\[ y_i = \mu_i + \epsilon, \quad \epsilon \sim b(1, \mu_i) \]

LR likelihood:

\[ L(\beta) = \prod_{i=1}^{R} \mu_i^{y_i} \left( 1 - \mu_i^{1-y_i} \right) \]
Minimize Deviance

Unlike linear regression, the RSS minimizer is not the MLE.

Use *deviance* instead:

\[
\text{DEV} = -2 \sum_{i=1}^{R} y_i \ln(\mu(x_i, \beta)) + (1 - y_i) \ln(1 - \mu(x_i, \beta))
\]

(shown for binary LR)

Equivalent to maximizing the LR likelihood.

Nonlinear.
Strategy Tree

Use Sigmoid

↓

Min Deviance

↓

Nonlinear – SCARY!
Strategy Tree

Generic Nonlinear Solver

Use Sigmoid

Min Deviance

Nonlinear – SCARY!
Nonlinear Minimizers

Could use any of these methods to minimize the deviance:

- Steepest Descent
- Newton-Raphson, quasi-Newton, truncated-Newton
- Conjugate Gradient
Nonlinear Minimizers

Could use any of these methods to minimize the deviance:

- Steepest Descent
- Newton-Raphson, quasi-Newton, truncated-Newton
- Conjugate Gradient

Decisions:

- CG is not a great nonlinear solver
- Deviance is convex
- CG is safe, unlikely to spiral in or wander off
- CG is favored for LR MLE

We’ll come back to CG later.
Strategy Tree

Generic Nonlinear Solver

Use Sigmoid

Min Deviance

Nonlinear – SCARY!
Strategy Tree

Generic Nonlinear Solver

Use Sigmoid

Min Deviance

Nonlinear – SCARY!

CG

Common Statistical Method: IRLS
Iteratively Re-Weighted Least Squares

IRLS iteratively approximates LR using linear regression problems. Let $\beta_i$ be our current estimate of the LR model parameters.

Define the \textit{adjusted dependent covariates} $z_i$ by

$$ z_i = \beta_i^T x + (y_i - \mu_i)/\mu_i(1 - \mu_i) $$
Iteratively Re-Weighted Least Squares

IRLS iteratively approximates LR using linear regression problems. Let $\beta_i$ be our current estimate of the LR model parameters.

Define the *adjusted dependent covariates* $z_i$ by

$$z_i = \beta_i^T x + (y_i - \mu_i)/\mu_i(1 - \mu_i)$$

The variance of $z$ is unknown (unlike ordinary linear regression), and it varies with $\mu(x_i)$ (also unlike OLR).
Iteratively Re-Weighted Least Squares

- IRLS iteratively approximates LR using linear regression problems. Let $\beta_i$ be our current estimate of the LR model parameters.

- Define the *adjusted dependent covariates* $z_i$ by

$$z_i = \beta_i^T x + (y_i - \mu_i)/\mu_i(1 - \mu_i)$$

- The variance of $z$ is unknown (unlike ordinary linear regression), and it varies with $\mu(x_i)$ (also unlike OLR).

- This out-of-the-ordinary linear regression problem can be partially solved with *Weighted Least Squares*
Iteratively Re-Weighted Least Squares

IRLS iteratively approximates LR using linear regression problems. Let $\beta_i$ be our current estimate of the LR model parameters.

- Define the *adjusted dependent covariates* $z_i$ by

$$z_i = \beta_i^T x + (y_i - \mu_i)/\mu_i(1 - \mu_i)$$

- The variance of $z$ is unknown (unlike ordinary linear regression), and it varies with $\mu(x_i)$ (also unlike OLR).

- This out-of-the-ordinary linear regression problem can be partially solved with *Weighted Least Squares*

- This requires solving the weighted linear regression

$$(X^T WX)\beta_{i+1} = X^T Wz$$

where $W = \text{diag}(w_i)$, $w_i^2 = \mu_i(1 - \mu_i)$. 

Paul Komarek, MBSW 2004 – p. 16/59
Iteratively Re-Weighted Least Squares

This is Iteratively Re-Weighted Least Squares (IRLS):

Make initial guess $\beta_0$

\[ i = 0 \]

Compute weights $w_j = \mu(x_j, \beta_j)(1 - \mu(x_j, \beta_j))$

Compute adjusted dependent covariates $z_j$

Compute $\beta_{i+1}$ using WLS

\[ i = i + 1 \]

Stop when the relative difference of the deviance is small:

\[ \left| \frac{(DEV_i - DEV_{i+1})}{DEV_{i+1}} \right| < \text{lreps} \]
Iteratively Re-Weighted Least Squares

For LR, IRLS happens to be equivalent to Newton-Raphson applied to the derivative of the likelihood.
Iteratively Re-Weighted Least Squares

- For LR, IRLS happens to be equivalent to Newton-Raphson applied to the derivative of the likelihood.
- Uh-oh, Newton-Raphson is slow
- We have to solve a linear system for each IRLS iteration
Strategy Tree

- Use Sigmoid
  - Min Deviance
    - Nonlinear – SCARY!

Generic Nonlinear Solver

Common Statistical Method: IRLS

IRLS = Newton–Raphson
IRLS with Cholesky Backsub

- Try Cholesky decomposition with back-substitution
- Slow – roughly $O(M^3)$
- Dies when data has linear dependencies ($\approx$ always)
IRLS with Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- Slow – roughly $O(M^3)$
- Dies when data has linear dependencies ($\approx$ always)
IRLS with Modified Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- Modified Cholesky with embedded column reduction
  - Slow – roughly $O(M^3)$
  - Handles linear dependencies by discarding attributes
  - Numerical problems:
IRLS with Modified Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- Modified Cholesky with embedded column reduction
  - Slow – roughly $O(M^3)$
  - Handles linear dependencies by discarding attributes
- Numerical problems:
  - Saturation when computing $\mu_i$
  - Apply band-aid: force $\text{modelmin} \leq \mu_i \leq \text{modelmax}$

Paul Komarek, MBSW 2004 – p. 22/59
IRLS with Modified Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- Modified Cholesky with embedded column reduction
  - Slow – roughly $O(M^3)$
  - Handles linear dependencies by discarding attributes
- Numerical problems:
  - Saturation when computing $\mu_i$
  - Apply band-aid: force $\text{modelmin} \leq \mu_i \leq \text{modelmax}$
  - Over/underflows when computing $w_i$
  - Apply band-aid: force $\text{wmargin} \leq w_i \leq 1-\text{wmargin}$
IRLS with Modified Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- Modified Cholesky with embedded column reduction
  - Slow – roughly $O(M^3)$
  - Handles linear dependencies by discarding attributes
- Numerical problems:
  - Saturation when computing $\mu_i$
    - Apply band-aid: force $\text{modelmin} \leq \mu_i \leq \text{modelmax}$
  - Over/underflows when computing $w_i$
    - Apply band-aid: force $\text{wmargin} \leq w_i \leq 1-\text{wmargin}$
- Correlations cause runaway estimates
IRLS with Modified Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- Modified Cholesky with embedded column reduction
  - Slow – roughly $O(M^3)$
  - Handles linear dependencies by discarding attributes
- Numerical problems:
  - Saturation when computing $\mu_i$
  - Apply band-aid: force $\text{modelmin} \leq \mu_i \leq \text{modelmax}$
  - Over/underflows when computing $w_i$
  - Apply band-aid: force $\text{wmargin} \leq w_i \leq 1-\text{wmargin}$
- Correlations cause runaway estimates
  - If atts 1 and 2 are positively correlated, then increases in $\beta_1$ may be offset by decreasing $\beta_2$
  - This may allow arbitrarily large $\beta_1$ with virtually no change in the model predictions
IRLS with Modified Cholesky Backsub

- FAILED: Cholesky decomposition with back-substitution
- FAILED: Modified Cholesky with embedded column reduction
  - Slow – roughly $O(M^3)$
  - Handles linear dependencies by discarding attributes
  - Numerical problems:
    - Saturation when computing $\mu_i$
    - Apply band-aid: force $\text{modelmin} \leq \mu_i \leq \text{modelmax}$
    - Over/underflows when computing $w_i$
    - Apply band-aid: force $\text{wmargin} \leq \mu_i \leq 1-\text{wmargin}$
  - Correlations cause runaway estimates
    - If atts 1 and 2 are positively correlated, then increases in $\beta_1$ may be offset by decreasing $\beta_2$
    - This may allow arbitrarily large $\beta_1$ with virtually no change in the model predictions
IRLS with Stepwise Modified Cholesky

- FAILED: Cholesky decomposition with back-substitution
- FAILED: Modified Cholesky with embedded column reduction
- **Stepwise** Modified Cholesky
  - Add attributes to model one at a time
  - Not quite so slow while model is small
  - Many numerical and correlation problems are avoided (if the att causes trouble, don’t add it)
  - How do we choose the next att to add? Information gain?
IRLS with Stepwise Modified Cholesky

- FAILED: Cholesky decomposition with back-substitution
- FAILED: Modified Cholesky with embedded column reduction
- FAILED: Stepwise Modified Cholesky
  - Add attributes to model one at a time
  - Not quite so slow while model is small
  - Many numerical and correlation problems are avoided (if the att causes trouble, don’t add it)
  - How do we choose the next att to add? Information gain?
IRLS with Divide-and-Conquer

- FAILED: Cholesky decomposition with back-substitution
- FAILED: Modified Cholesky with embedded column reduction
- FAILED: Stepwise Modified Cholesky

Divide-and-Conquer

- Use binary tree to partition attributes, leaves have pairs
- Write $2 \times 2$ matrix inverse explicitly
- Solve one 2-attribute regression per tree node
- Never ran fast enough to justify pursuing optimal recombination
- (read: we found something better)
IRLS with Divide-and-Conquer

- FAILED: Cholesky decomposition with back-substitution
- FAILED: Modified Cholesky with embedded column reduction
- FAILED: Stepwise Modified Cholesky
- FAILED: Divide-and-Conquer
Strategy Tree

- Use Sigmoid
  - Min Deviance
    - Nonlinear – SCARY!
- Common Statistical Method: IRLS
  - IRLS = Newton–Raphson
    - Cholesky Backsub
    - Modified Cholesky
    - Stepwise Cholesky
    - Divide–and–Conquer
      - modmin, modmax, wmarg
      - SLOW!
IRLS: What are we solving again?

The WLS solution $\beta_{i+1}$ is an estimate of the real $\beta$. 

Paul Komarek, MBSW 2004 – p. 29/59
IRLS: What are we solving again?

- The WLS solution $\beta_{i+1}$ is an estimate of the real $\beta$
- Why compute this estimate exactly?
IRLS: What are we solving again?

- The WLS solution $\beta_{i+1}$ is an estimate of the real $\beta$
- Why compute this estimate exactly?
- Estimate the estimates!
IRLS: Approx WLS Using CG

Let $A = (X^T WX)$, $b = X^T Wz$. The WLS problem is

$$A \beta_{i+1} = b$$

This is equivalent to minimizing the quadratic form

$$\frac{1}{2} x^T Ax - b$$
IRLS: Approx WLS Using CG

Let \( A = (X^T WX) \), \( b = X^T Wz \). The WLS problem is

\[ A\beta_{i+1} = b \]

This is equivalent to minimizing the quadratic form

\[ \frac{1}{2} x^T Ax - b \]

CG has a special, simple, fast form for this problem.

We call this *linear* CG.

Handles linear dependencies auto-magically (minimizes \( A\)-norm). Also, sparse computations are easy.

Similar, but not identical, to *truncated-Newton* methods.
Strategy Tree

Generic Nonlinear Solver
- CG
- Use Sigmoid
  - Min Deviance
    - Nonlinear – SCARY!

Common Statistical Method: IRLS
- IRLS = Newton-Raphson
  - Estimate the estimates
    - CG, specialized
  - Cholesky Backsub
  - Modified Cholesky
  - Stepwise Cholesky
  - Divide-and-Conquer
    - $\text{modmin, modmax, wmax}$
    - SLOW!
## IRLS: Approx Using CG

<table>
<thead>
<tr>
<th>Linear CG</th>
<th>Nonlinear CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal search direction updates</td>
<td>Multiple search directions available, nothing is optimal</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## IRLS: Approx Using CG

<table>
<thead>
<tr>
<th>Linear CG</th>
<th>Nonlinear CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal search direction updates</td>
<td>Multiple search directions available, nothing is optimal</td>
</tr>
<tr>
<td>Location of minimum in search direction is known</td>
<td>Must use <em>line search</em> to find optimum in search direction</td>
</tr>
</tbody>
</table>
# IRLS: Approx Using CG

<table>
<thead>
<tr>
<th>Linear CG</th>
<th>Nonlinear CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal search direction updates</td>
<td>Multiple search directions available, nothing is optimal</td>
</tr>
<tr>
<td>Location of minimum in search direction is known</td>
<td>Must use line search to find optimum in search direction</td>
</tr>
<tr>
<td><em>Conjugacy</em> property of search directions guarantees exact solution in (M) steps (with perfect arithmetic)</td>
<td>Because Hessian changes with location, <em>conjugacy</em> of search directions decays. Must occasionally restart. Might never converge.</td>
</tr>
</tbody>
</table>
Strategy Tree

Generic Nonlinear Solver
- CG
  - * Directions
  - * Line Searches
  - * Restarts
  - OH MY!
- Use Sigmoid
- Min Deviance
- Nonlinear – SCARY!

Common Statistical Method: IRLS
- IRLS = Newton–Raphson
- Estimate the estimates
  - CG, specialized
- Cholesky Backsub
- Modified Cholesky
- Stepwise Cholesky
- Divide–and–Conquer
  - modmin, modmax, wmax
  - SLOW!
How many linear CG iterations are needed to approximate
solution to

$$(X^T W X) \beta_{i+1} = X^T W z$$

Far fewer than $M$

CG convergence depends on the eigenvalues of the matrix

Answers are usually good-enough after 5-100 iterations,
depending on dataset size
IRLS: Terminating CG

How many linear CG iterations are needed to approximate solution to

$$(X^T WX)\beta_i + 1 = X^T Wz$$

Far fewer than $M$

CG convergence depends on the eigenvalues of the matrix

Answers are usually good-enough after 5-100 iterations, depending on dataset size

Terminate according to $c\text{geps}$ or $c\text{gdeveps}$:

- $||b - A\beta_i||_2^2 < c\text{geps}$
- $||(\text{DEV}_i - \text{DEV}_{i+1})/\text{DEV}_{i+1}| < c\text{gdeveps}$
IRLS: Some CG Details

- Instability and overfitting are possible.
IRLS: Some CG Details

- Instability and overfitting are possible.
- Result of empirical study: Ridge Regression (coefficient shrinkage, i.e. large coefficient penalty, i.e. Bayesian prior on coefficients) and constant-improvement requirement.
IRLS: Some CG Details

- Instability and overfitting are possible.
- Result of empirical study: Ridge Regression (coefficient shrinkage, i.e. large coefficient penalty, i.e. Bayesian prior on coefficients) and constant-improvement requirement.
- Some small further accelerations available.
Strategy Tree

Generic Nonlinear Solver

- CG
  - * Directions
  - * Line Searches
  - * Restarts
  - OH MY!

  - modmin, modmax
  - cgeps
  - cgdecal, cgwindow
  - binitmean
  - margin
  - rrlambda

- Cholesky Backsub
- Modified Cholesky
- Stepwise Cholesky
- Divide–and–Conquer
- SLOW!

Common Statistical Method: IRLS

IRLS = Newton–Raphson

- Estimate the estimates
  - CG, specialized
    - cgeps
    - cgdeveps
    - cgdecal, cgwindow
      - * binitmean
      - * output shrink (margin)
      - * Ridge Regression (rnlambd)

- Use Sigmoid
  - Min Deviance
  - Nonlinear – SCARY!
Logistic Regression Results
Requirements For LR

- Autonomous
- Accurate
- Very fast
Scoring

Precision and recall are computed from the number of false positives and true positives. These measures treat classifications as “black and white”.

Another option is to examine how well a classifier can rank test points. This is what ROC curves do. The AUC (Area Under Curve) score summarizes the ROC curve.
Comparison To Other Classifiers

**Support Vector Machines (SVMs):** SVMs search for the maximum-margin separating hyperplane between classes. They are considered a state-of-the-art classifier. SVMs are routinely used with a nonlinear data transformation, to help accommodate nonlinear class boundaries. Radial Basis Functions (RBF) are a common basis for transformation. We use the popular SVM\(^{\text{light}}\), version 5.0.

**K-Nearest-Neighbor (KNN):** KNN makes a class assignment based on the classes of the K training points nearest to the test point. We use the KNS2 algorithm from Liu et al. [2003], which uses ball trees to accelerate predictions for data with skewed classes.

**Bayes’ Classifier (BC):** BC is a well-known, generative fast classifier which assumes the attributes are conditionally independent.
Comparison: Results (1 of 2)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>citeeseer Time</th>
<th>AUC</th>
<th>imdb Time</th>
<th>AUC</th>
<th>ds2 Time</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR-CGEPS</td>
<td>43</td>
<td>0.946±0.021</td>
<td>303</td>
<td>0.983±0.007</td>
<td>2983</td>
<td>0.720±0.030</td>
</tr>
<tr>
<td>LR-CGDVEPS</td>
<td>67</td>
<td>0.945±0.021</td>
<td>320</td>
<td>0.983±0.007</td>
<td>1647</td>
<td>0.722±0.032</td>
</tr>
<tr>
<td>CG-MLE</td>
<td>97</td>
<td>0.946±0.021</td>
<td>369</td>
<td>0.983±0.009</td>
<td>3198</td>
<td>0.724±0.030</td>
</tr>
<tr>
<td>SVM LINEAR</td>
<td>87</td>
<td>0.810±0.049</td>
<td>565</td>
<td>0.938±0.016</td>
<td>2536</td>
<td>0.693±0.034</td>
</tr>
<tr>
<td>SVM RBF</td>
<td>1456</td>
<td>0.854±0.045</td>
<td>6553</td>
<td>0.954±0.011</td>
<td>67117</td>
<td>0.700±0.034</td>
</tr>
<tr>
<td>KNN K=1</td>
<td>NA</td>
<td>NA±NA</td>
<td>NA</td>
<td>NA±NA</td>
<td>NA</td>
<td>NA±NA</td>
</tr>
<tr>
<td>KNN K=9</td>
<td>NA</td>
<td>NA±NA</td>
<td>NA</td>
<td>NA±NA</td>
<td>NA</td>
<td>NA±NA</td>
</tr>
<tr>
<td>KNN K=129</td>
<td>NA</td>
<td>NA±NA</td>
<td>NA</td>
<td>NA±NA</td>
<td>NA</td>
<td>NA±NA</td>
</tr>
<tr>
<td>BC</td>
<td>10</td>
<td>0.501±0.038</td>
<td>33</td>
<td>0.507±0.023</td>
<td>127</td>
<td>0.533±0.020</td>
</tr>
</tbody>
</table>
## Comparison: Results (2 of 2)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>ds1 Time</th>
<th>ds1 AUC</th>
<th>ds1.100pca Time</th>
<th>ds1.100pca AUC</th>
<th>ds1.10pca Time</th>
<th>ds1.10pca AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR-CGEPS</td>
<td>86</td>
<td>0.949±0.009</td>
<td>44</td>
<td>0.918±0.011</td>
<td>8</td>
<td>0.846±0.013</td>
</tr>
<tr>
<td>LR-CGDEVEPS</td>
<td>59</td>
<td>0.948±0.009</td>
<td>35</td>
<td>0.913±0.011</td>
<td>9</td>
<td>0.842±0.015</td>
</tr>
<tr>
<td>CG-MLE</td>
<td>151</td>
<td>0.946±0.008</td>
<td>364</td>
<td>0.916±0.012</td>
<td>48</td>
<td>0.844±0.014</td>
</tr>
<tr>
<td>SVM LINEAR</td>
<td>188</td>
<td>0.918±0.012</td>
<td>130</td>
<td>0.874±0.012</td>
<td>68</td>
<td>0.582±0.048</td>
</tr>
<tr>
<td>SVM RBF</td>
<td>1850</td>
<td>0.924±0.012</td>
<td>1036</td>
<td>0.897±0.010</td>
<td>490</td>
<td>0.856±0.017</td>
</tr>
<tr>
<td>KNN K=1</td>
<td>424</td>
<td>0.790±0.029</td>
<td>74</td>
<td>0.785±0.024</td>
<td>9</td>
<td>0.753±0.028</td>
</tr>
<tr>
<td>KNN K=9</td>
<td>782</td>
<td>0.909±0.016</td>
<td>166</td>
<td>0.894±0.016</td>
<td>14</td>
<td>0.859±0.019</td>
</tr>
<tr>
<td>KNN K=129</td>
<td>2381</td>
<td>0.938±0.010</td>
<td>819</td>
<td>0.938±0.010</td>
<td>89</td>
<td>0.909±0.013</td>
</tr>
<tr>
<td>BC</td>
<td>4</td>
<td>0.884±0.011</td>
<td>8</td>
<td>0.890±0.012</td>
<td>2</td>
<td>0.863±0.015</td>
</tr>
</tbody>
</table>
Comparison: ROC for ds1

ROC curves for dataset ds1, with a linear False positives axis.
Comparison: Log ROC for ds1

ROC curves for dataset ds1, with a logarithmic False positives axis.
Conclusions

- We have demonstrated that LR is a capable and fast tool for data mining and high-dimensional classification problems.

- Our novel IRLS fitting procedures outperforms the traditional combination of CG and LR maximum likelihood estimation, as well as state-of-the-art techniques such as Support Vector Machines.

- This superior performance was demonstrated on many synthetic and real-world datasets.
FKNN: Even Better Than KNS2.
K-Nearest Neighbor (KNN) Classification

- For a query point \( q \), find the neighbor set \( \mathcal{N} = \{n_1, \ldots, n_k\} \) from the training set \( \mathcal{T} = \{x_1, \ldots, x_R\} \).
- Classify \( q \) as positive if the majority of neighbors are positive.
- Intuition: data of the same class tend to form clusters, and \( q \) is likely to have the same class as its neighbors.
Why KNN?

- Why use KNN?
  - It is simple and mature.
  - It is wide application: text, vision, drug activity, . . .
  - Easy to implement and test.

Usual implementation of KNN is slow for large datasets.

O(MR) for each query point.

Overkill for linear boundaries in high dimensions.

Can be very sensitive to number of neighbors used, leading to high variance.

However, it is possible to accelerate KNN. We will discuss the use of ball trees to partition the training set.
Why KNN?

- Why use KNN?
  - It is simple and mature.
  - It is wide application: text, vision, drug activity, . . .
  - Easy to implement and test.

- Why not use KNN?
  - Usual implementation of KNN is slow for large datasets. $O(MR)$ for each query point.
  - Overkill for linear boundaries in high dimensions.
  - Can be very sensitive to number of neighbors used, leading to high variance.
Why KNN?

Why use KNN?
- It is simple and mature.
- It is wide application: text, vision, drug activity, ...
- Easy to implement and test.

Why not use KNN?
- Usual implementation of KNN is slow for large datasets. \( O(MR) \) for each query point.
- Overkill for linear boundaries in high dimensions.
- Can be very sensitive to number of neighbors used, leading to high variance.

However, it is possible to accelerate KNN. We will discuss the use of ball trees to partition the training set.
Training Set
Ball Tree
Ball Tree
Let $q$ be a query point, and let $x$ be any training set point inside a ball $B$.

\[ \|q - x\| \geq \|q - c\| - r \]
\[ \|q - x\| \leq \|q - c\| + r \]
KNS2: Accelerating KNN Search

- **Method Of KNS2:**
  - Use separate trees for positive and negative training points.
  - Prune ball tree nodes using triangle inequality, instead of examining all points in each node.

- **Results Of KNS2:**
  - Significant speed-up for small $M$: $O(M \log R)$.
  - Little speed-up for large $M$: $O(MR)$.
  - Reason: with many dimensions, training data “spreads out”.

- **Conclusions:** Ball trees do not help high-dimensional KNN search.
  - But wait! We want KNN classification – do we need a full KNN search? NO!
**KNS2: Accelerating KNN Search**

- **Method Of KNS2:**
  - Use separate trees for positive and negative training points.
  - Prune ball tree nodes using triangle inequality, instead of examining all points in each node.

- **Results Of KNS2:**
  - Significant speed-up for small $M$: $O(M \log(R))$.
  - Little speed-up for large $M$: $O(MR)$.
  - Reason: with many dimensions, training data “spreads out”.
KNS2: Accelerating KNN Search

- Method Of KNS2:
  - Use separate trees for positive and negative training points.
  - Prune ball tree nodes using triangle inequality, instead of examining all points in each node.

- Results Of KNS2:
  - Significant speed-up for small $M$: $O(M \log(R))$.
  - Little speed-up for large $M$: $O(MR)$.
  - Reason: with many dimensions, training data “spreads out”.

- Conclusions:
  - Ball trees do not help high-dimensional KNN search.
KNS2: Accelerating KNN Search

Method Of KNS2:
- Use separate trees for positive and negative training points.
- Prune ball tree nodes using triangle inequality, instead of examining all points in each node.

Results Of KNS2:
- Significant speed-up for small $M$: $O(M \log(R))$.
- Little speed-up for large $M$: $O(MR)$.
- Reason: with many dimensions, training data “spreads out”.

Conclusions:
- Ball trees do not help high-dimensional KNN search.
- But wait! We want KNN classification – do we need a full KNN search? NO!
For binary classification, we only need decide whether \( \lceil k/2 \rceil \) of the nearest neighbors are positive. We will describe FKNN, a Fast KNN classifier from the Auton Lab, described in Liu et al. [2003].
For binary classification, we only need decide whether $\lfloor k/2 \rfloor$ of the nearest neighbors are positive. We will describe FKNN, a Fast KNN classifier from the Auton Lab, described in Liu et al. [2003].

Let $t = \lfloor k/2 \rfloor$, and let $q$ be a query point.

Let $m = k - t + 1$. If $k$ is odd, then $m = k$. 

Accelerating KNN Classification: FKNN
For binary classification, we only need decide whether \( \lfloor k/2 \rfloor \) of the nearest neighbors are positive. We will describe FKNN, a Fast KNN classifier from the Auton Lab, described in Liu et al. [2003].

Let \( t = \lfloor k/2 \rfloor \), and let \( q \) be a query point.

Let \( m = k - t + 1 \). If \( k \) is odd, then \( m = k \).

Define \( D_{k}^{\text{pos}} \) as the distance of the \( k^{\text{th}} \) positive nearest neighbor of \( q \).

Define \( D_{k}^{\text{neg}} \) as the distance of the \( k^{\text{th}} \) negative nearest neighbor of \( q \).
FKNN Notation Diagram

- **positive**
- **negative**

\[
k = 5 \\
t = \lceil k/2 \rceil = 3 \\
m = k - t + 1 = 3
\]
FKNN Method

To decide if the test point $q$ is positive or negative, we use the distances $D^\text{pos}_i$ and $D^\text{neg}_m$. If $D^\text{pos}_i < D^\text{neg}_m$, then the positive class dominates the $k$ nearest neighbors and $q$ is positive (there are more positive points closer to $q$). If $D^\text{pos}_i > D^\text{neg}_m$, then the negative class dominates and $q$ is negative. If we apply triangle-inequality bounds from our ball tree to $D^\text{pos}_i$ and $D^\text{neg}_m$, we can avoid a complete KNN search. We only need to decide which distance is larger - fast approximate method is available. This was just a teaser. To find out how the approximations are made, see Liu et al. [2003].
FKNN Method

To decide if the test point $q$ is positive or negative, we use the distances $D_{i}^{\text{pos}}$ and $D_{m}^{\text{neg}}$.

- If $D_{i}^{\text{pos}} \leq D_{m}^{\text{neg}}$, then the positive class dominates the $k$ nearest neighbors and $q$ is positive (there are more positive points closer to $q$).

- If $D_{i}^{\text{pos}} > D_{m}^{\text{neg}}$, then the negative class dominates and $q$ is negative.
FKNN Method

To decide if the test point \( q \) is positive or negative, we use the distances \( D_{i}^{\text{pos}} \) and \( D_{m}^{\text{neg}} \).

If \( D_{i}^{\text{pos}} \leq D_{m}^{\text{neg}} \), then the positive class dominates the \( k \) nearest neighbors and \( q \) is positive (there are more positive points closer to \( q \)).

If \( D_{i}^{\text{pos}} > D_{m}^{\text{neg}} \), then the negative class dominates and \( q \) is negative.

If we apply triangle-inequality bounds from our ball tree to \( D_{i}^{\text{pos}} \) and \( D_{m}^{\text{neg}} \), we can avoid a complete KNN search.

We only need to decide which distance is larger - fast approximate method is available.
FKNN Method

- To decide if the test point $q$ is positive or negative, we use the distances $D_i^{\text{pos}}$ and $D_m^{\text{neg}}$.
- If $D_i^{\text{pos}} \leq D_m^{\text{neg}}$, then the positive class dominates the $k$ nearest neighbors and $q$ is positive (there are more positive points closer to $q$).
- If $D_i^{\text{pos}} > D_m^{\text{neg}}$, then the negative class dominates and $q$ is negative.
- If we apply triangle-inequality bounds from our ball tree to $D_i^{\text{pos}}$ and $D_m^{\text{neg}}$, we can avoid a complete KNN search.
- We only need to decide which distance is larger - fast approximate method is available.
- This was just a teaser. To find out how the approximations are made, see Liu et al. [2003].
## FKNN: Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dimension ($M$)</th>
<th>Rows ($R$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movie</td>
<td>62</td>
<td>38,943</td>
</tr>
<tr>
<td>J_Lee (citeseer) (projection)</td>
<td>100</td>
<td>181,395</td>
</tr>
<tr>
<td>Blanc_Mel (imdb) (projection)</td>
<td>100</td>
<td>186,414</td>
</tr>
<tr>
<td>ds1</td>
<td>6,348</td>
<td>26,733</td>
</tr>
<tr>
<td>ds2</td>
<td>1,143,054</td>
<td>88,358</td>
</tr>
</tbody>
</table>
FKNN: Speed-up over Naive KNN

FKNN is 10-50 times faster than Naive KNN.
Autonomous Fast Classifiers For Pharmaceutical Datasets

Paul Komarek and Ting Liu
Auton Lab
komarek@cmu.edu

School of Computer Science, Carnegie Mellon University
Papers and software available at http://autonlab.org
References
