Parallel Spatial Boosting

Carlotta Domeniconi
Dept. of Computer Science
George Mason University

Joint work with
Uday Kamath and Kenneth De Jong

Outline

- The problem and motivation
- Proposed solution: Parallel Boosting Algorithm (PSBML)
  - The overall idea
  - The algorithm
  - Theoretical analysis of PSBML
  - Empirical analysis
    - Meta learning
    - Scalability
    - Robustness to noise

The Problem

Current scenario in ML Algorithms:

- Need entire training data in memory for modeling
- Training time and memory requirements
  - E.g. SVMs: $O(n^3)$ time - $O(n^2)$ space

Current solutions:

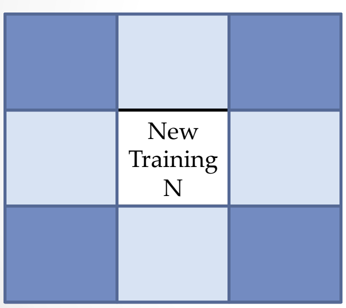
- Sampling data
- Algorithm-specific customization for parallelization

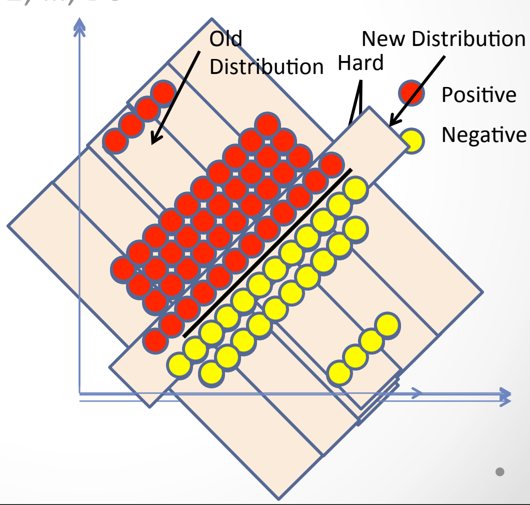
Parallel Boosting Algorithm

- Main ingredients:
  - Meta-Learning: can use any classifier
  - Parallelization using a grid structure and neighborhoods
  - Ensemble and Boosting methodologies

The Overall Idea

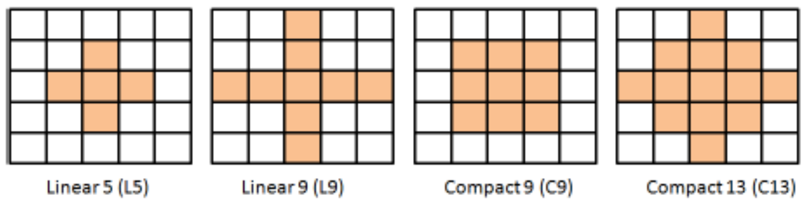
- Classifiers: C1, C2, …, C9
- Training Data: D1, D2, …, D9





Algorithm Details

- Parallelism:



Neighborhood structures

Algorithm Details

- Ensemble assessment of confidence:

$$cs_i = min_{n \in N_i} c_{ni}$$ Confidence of instance *i*

$$cs_i^{norm} = \frac{cs_i - cs_{min}}{cs_{max} - cs_{min}}$$

$$w_i = 1 - cs_i^{norm}$$ Weight of instance *i*

Algorithm Details

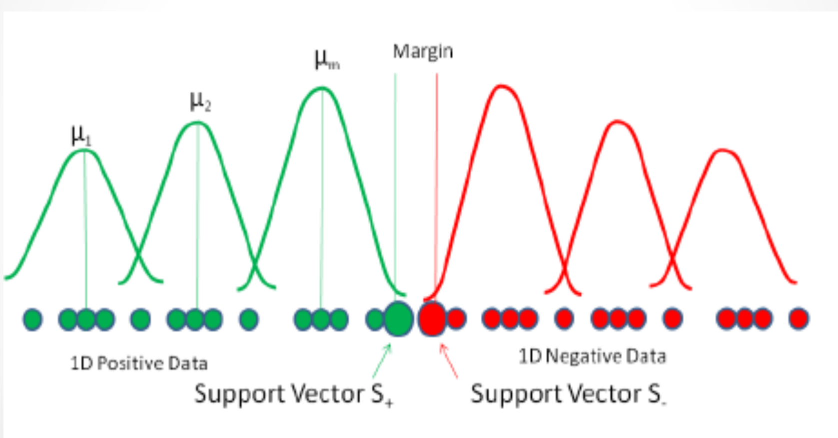
- Boosting-like behavior:
  - Weighted Sampling using $w_i = 1 - cs_i^{norm}$
  - The weights define a distribution over the data
  - The smaller the confidence credited to an instance, the larger the probability to be selected

```
INITIALIZEGRID(Train, GridParam)
comment: distribute the instances over the nodes in grid
currentMin ← 100
Pr ← GridParam.pr
comment: Probability of replacement
for i ← 0 to GridParam.iter
  do
marginData ← Train
comment: marginData initialized to all training data
  { TRAINNODES(GridParam)
  { comment: Train all nodes
  { TESTANDWEIGHNODES(GridParam);
  { comment: Test using neighborhood and assign weight
  { PrunedData ← {}
  { for j ← 0 to GridParam.nodes
  {   do
  {   { NeighborData ← COLLECTNEIGHBORDATA(j);
  {   { NodeData ← NodeData + NeighborData
  {   { ReplaceData ← ROULETTEWHEELSEL(NodeData, Pr)
  {   { PrunedData ← UNIQUE(PrunedData, ReplaceData)
  {   { comment: Unique keeps 1 copy of instances in set
  { error ← TESTCLASSIFIER(PrunedData, Validation)
  { comment: Use Validation set to track model learning
  { if error < currentMin
  {          { currentMin ← error
  {   then   { bestClassifier ← SAVECLASSIFIER(PrunedData)
  {          { marginData ← PrunedData
  {          { comment: marginData set reduced
return (bestClassifier, marginData)
```

Theoretical Analysis

- We use Gaussian Mixtures (GMMs) and mean-shift to model the behavior of PSBML
- We show: PSBML converges to a data distribution whose modes are centered around the margin, i.e. around the hardest points to classify

Theoretical Analysis

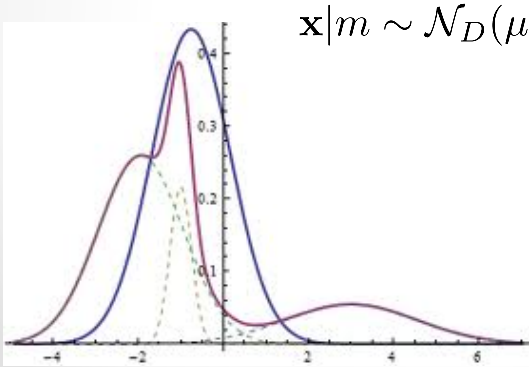


Theoretical Analysis - Tools

Gaussian Mixture Model

$$p(\mathbf{x}) = \sum_{m=1}^{M} p(m)p(\mathbf{x}|m) \; \forall \mathbf{x} \in \mathbb{R}^{\mathrm{D}}$$

$$\mathbf{x}|m \sim \mathcal{N}_D(\mu_m, \Sigma_m)$$

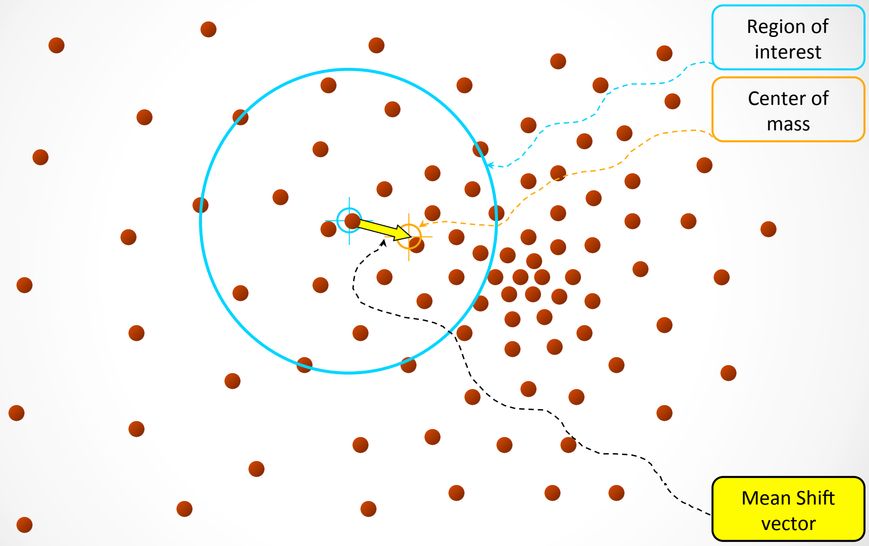


Theoretical Analysis - Tools

Mean-Shift Algorithm

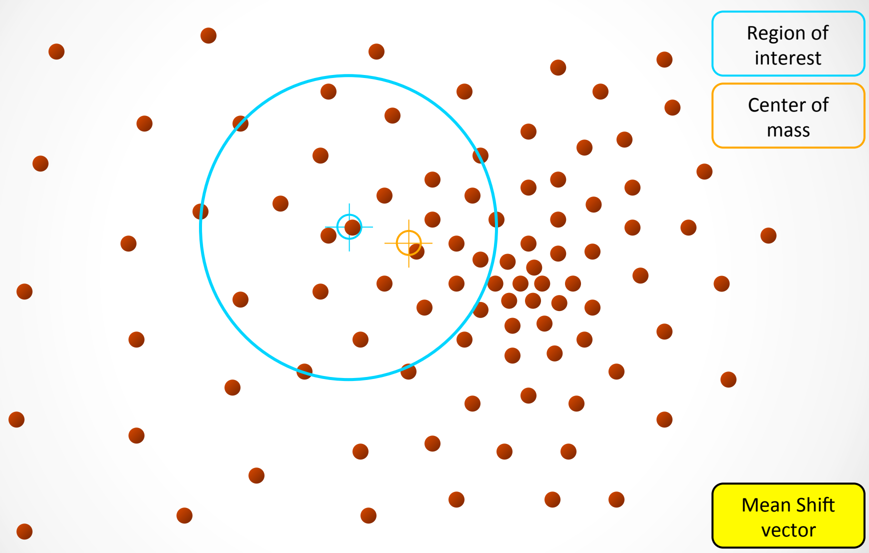
- Finds modes in a set of data samples, manifesting an underlying distribution
- Non-Parametric Model
- Distribution changes along the gradient

Mean Shift-Intuitive Description

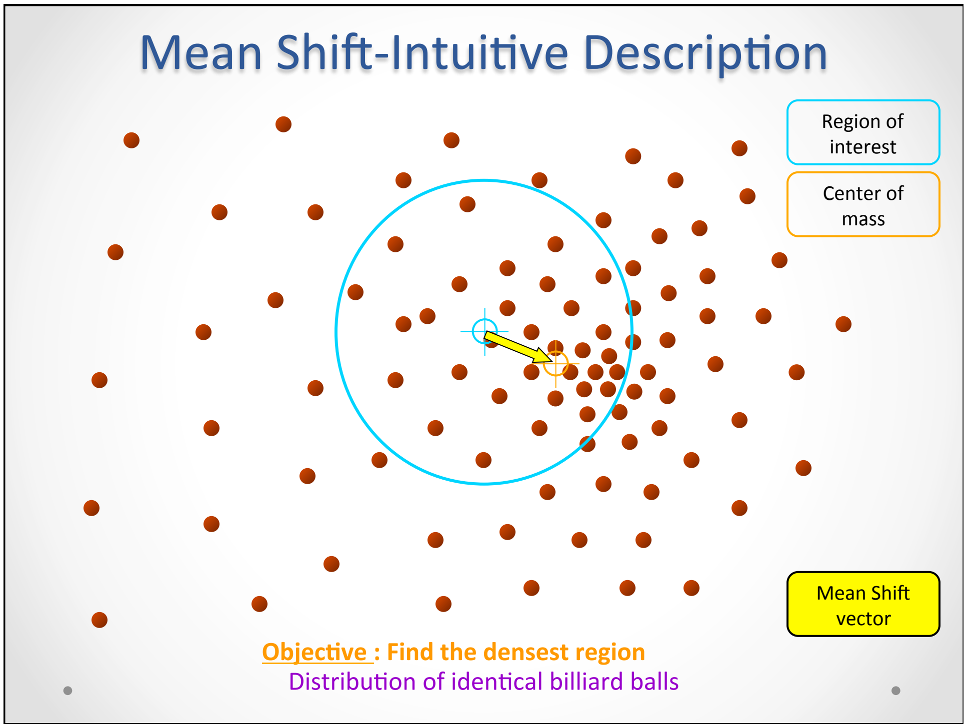


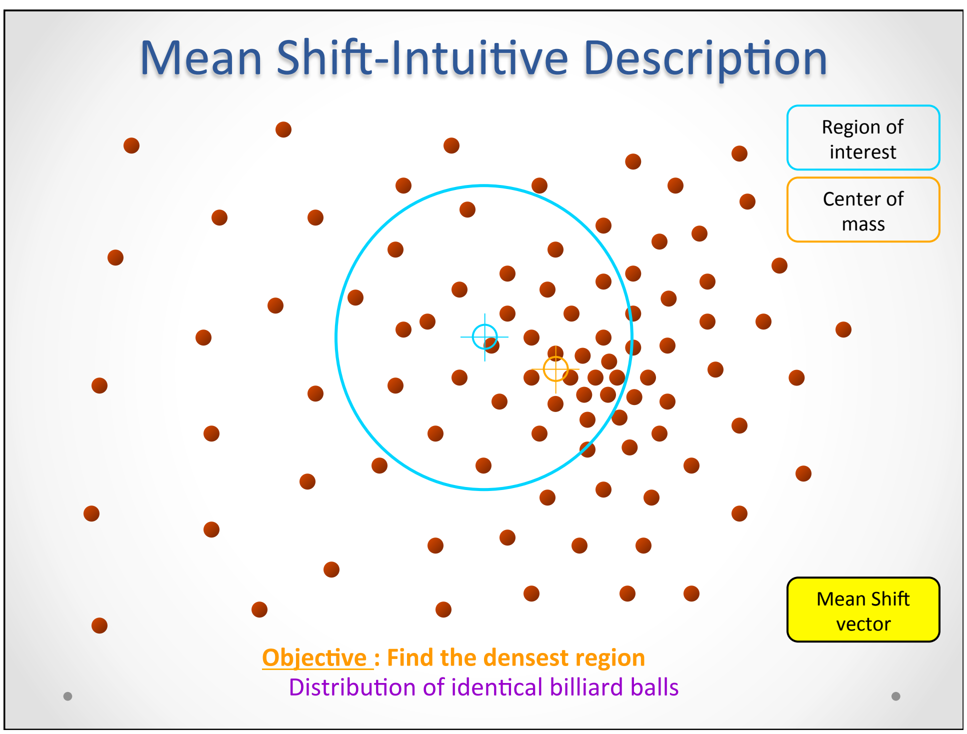
Objective : Find the densest region
Distribution of identical billiard balls

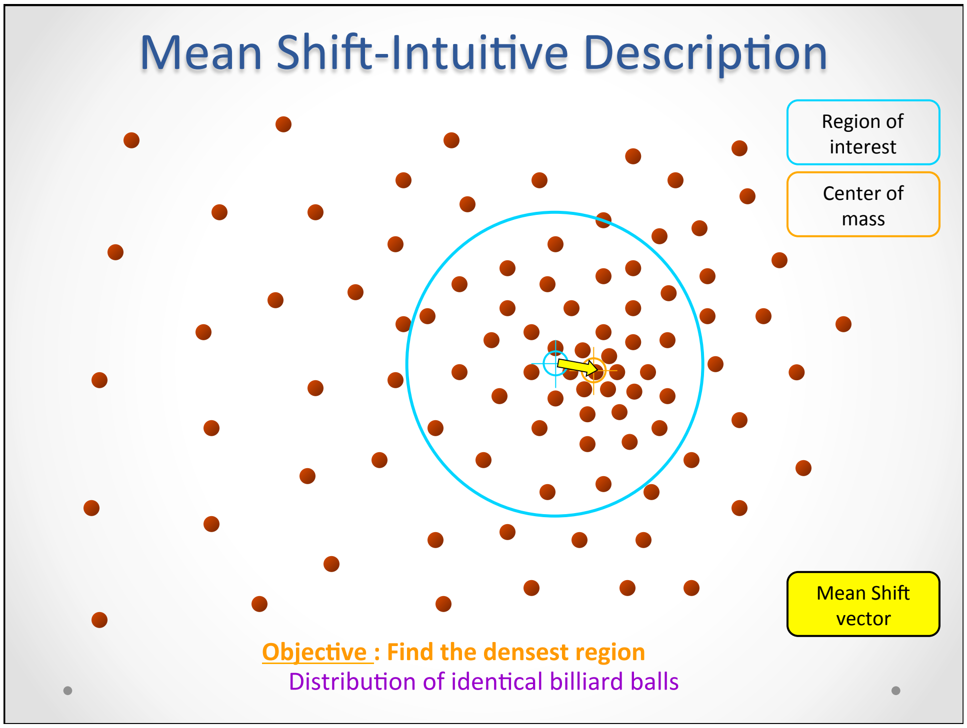
Mean Shift-Intuitive Description

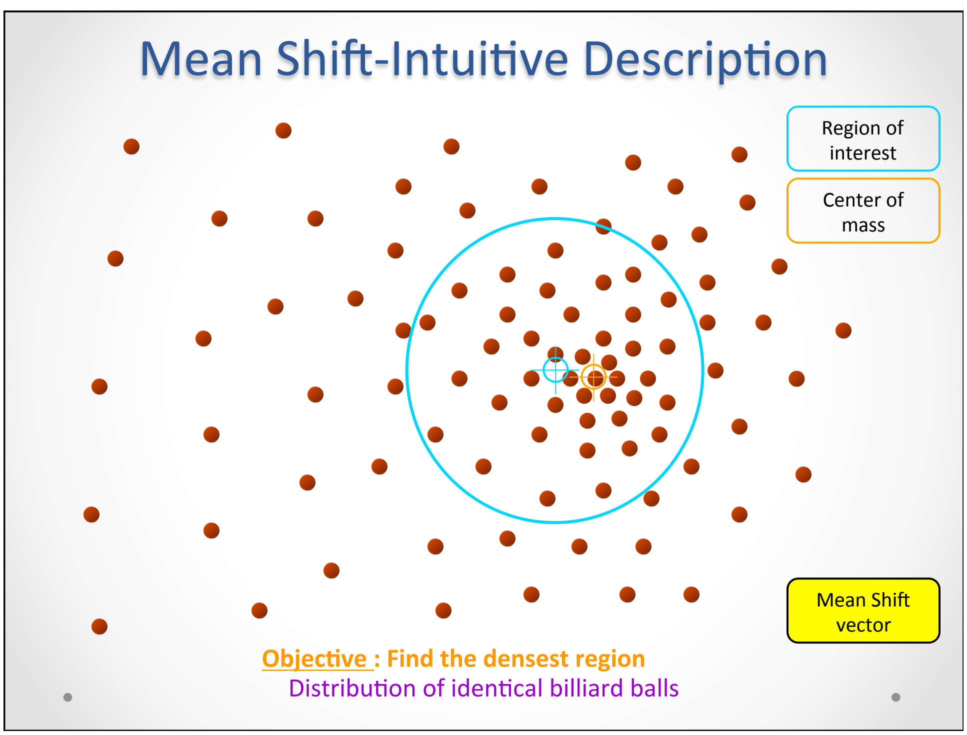


Objective : Find the densest region
Distribution of identical billiard balls

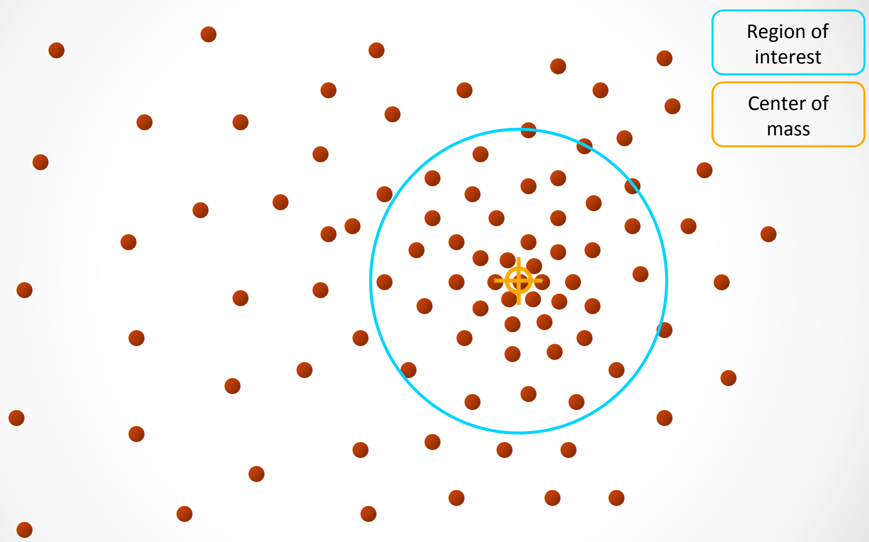

Mean Shift-Intuitive Description
Region of interest
Center of mass
Mean Shift vector
Objective : Find the densest region
Distribution of identical billiard balls


Mean Shift-Intuitive Description
Region of interest
Center of mass
Mean Shift vector
Objective : Find the densest region
Distribution of identical billiard balls


Mean Shift-Intuitive Description
Region of interest
Center of mass
Mean Shift vector
Objective : Find the densest region
Distribution of identical billiard balls


Mean Shift-Intuitive Description
Region of interest
Center of mass
Mean Shift vector
Objective : Find the densest region
Distribution of identical billiard balls

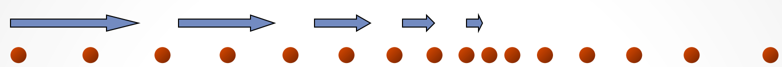
Mean Shift-Intuitive Description



Objective : Find the densest region

Distribution of identical billiard balls

Mean Shift Properties



- Automatic convergence speed – the mean shift vector size depends on the gradient itself
- Near maxima, the steps are small and refined

} **Adaptive** Gradient Ascent

- Convergence is guaranteed for infinitesimal steps only → infinitely convergent, (therefore set a lower bound)
- For Uniform Kernel, convergence is achieved in a finite number of steps
- Normal Kernel, exhibits a smooth trajectory, but is slower than Uniform Kernel

Mean Shift Properties

Mean shift and GMMs

- Fixed-point iterative method solution:

$$\mathbf{x}^{(t+1)} = \mathbf{f}(\mathbf{x}^{(t)})$$

$$\mathbf{x} = \mathbf{f}(\mathbf{x}) = (\sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_m^{-1})^{-1} \sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_m^{-1}\mu_m$$

Theoretical Analysis

- We model the sample data assigned to a node and to its neighbors using a GMM
- We need to model the weighted sampling process within a GMM

$$p'(\mathbf{x}|m) = w(\mathbf{x}) * p(\mathbf{x}|m)$$

$$w(\mathbf{x}) = (2\pi)^{-D/2}|\Sigma_{\mathbf{s}}|^{-1/2}e^{-1/2(\mathbf{x}-\mathbf{s})^T\Sigma_{\mathbf{s}}^{-1}(\mathbf{x}-\mathbf{s})}$$

Theoretical Analysis

- Our fixed-point solution:

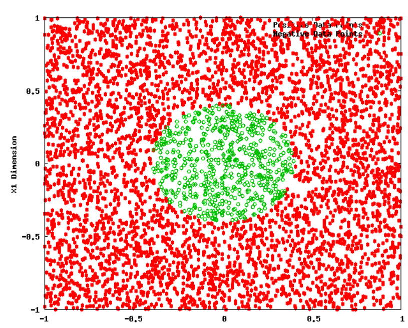
$$\mathbf{x} = \left(\sum_{m=1}^{M} p(m|\mathbf{x}) {\Sigma_s}^{-1} + \sum_{m=1}^{M} p(m|\mathbf{x}) {\Sigma_m}^{-1} \right)^{-1} \times$$
$$\left(\sum_{m=1}^{M} p(m|\mathbf{x}) {\Sigma_s}^{-1} \mathbf{s} + \sum_{m=1}^{M} p(m|\mathbf{x}) {\Sigma_m}^{-1} \mu_m \right)$$

Compare with:

$$\mathbf{x} = \mathbf{f}(\mathbf{x}) = \left(\sum_{m=1}^{M} p(m|\mathbf{x}) \Sigma_m^{-1} \right)^{-1} \sum_{m=1}^{M} p(m|\mathbf{x}) \Sigma_m^{-1} \mu_m$$

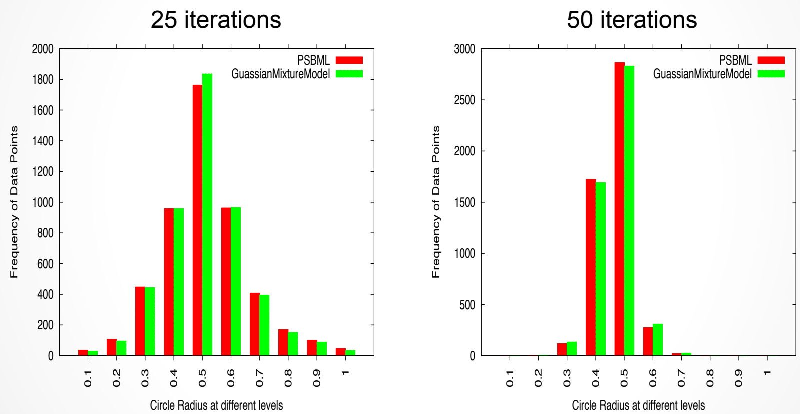
Theoretical Analysis - Experiment 1

Circle dataset



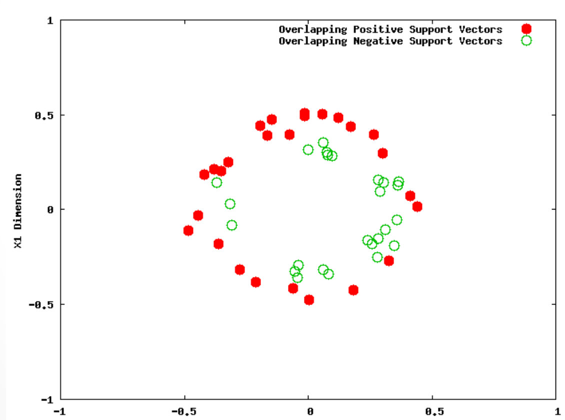
5 x 5 grid with large margin classifiers vs.
5 x 5 grid with GMM + mean shift

Theoretical Analysis - Experiment 1

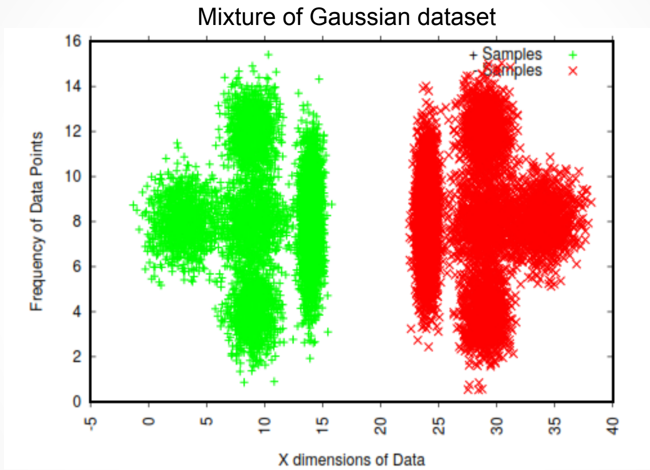


Theoretical Analysis - Experiment 1

Support vectors and PSBML hard instances overlap (90%)



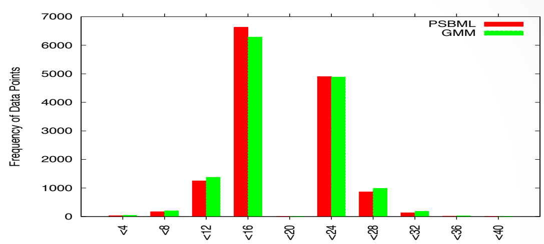
Theoretical Analysis – Experiment 2



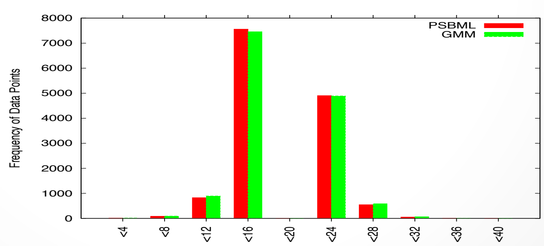
5 x 5 grid of large margin classifiers vs.
5 x 5 grid of GMM + mean shift

Theoretical Analysis – Experiment 2

25 iterations



50 iterations



Meta-learning – Experiment 3

Goal: Does PSBML provide a general framework for meta-learning?
Is PSBML effective as a parallel algorithm?

Datasets

	Adult	**W8A**	**ICJNN1**	**Cod**	**Cover**
# *Train*	32560	49749	49990	331617	581012
# *Test*	16279	14951	91701	59535	58102
# *Features*	123	300	22	8	54

Meta-learning – Experiment 3

AUC Results

	Adult	**W8A**	**ICJNN1**	**Cod**	**Cover**
NB	90.1	94.30	81.60	87.20	84.90
PSBML	**90.69**	**96.10**	**81.79**	**91.79**	**87.01**
C4.5	89.88	**87.80**	94.60	95.90	99.50
PSBML	89.78	84.80	**97.30**	**97.24**	97.44
Linear SVM	54.60	80.20	64.60	88.80	72.20
PSBML	**60.01**	80.70	64.80	**95.10**	**79.10**

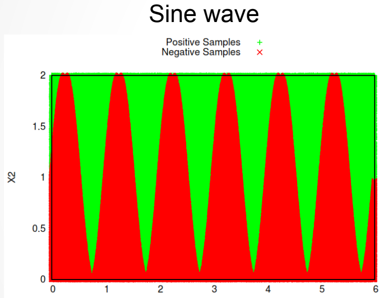
Scalability – Experiment 4

- Goal: Is PSBML competitive against custom optimized learning algorithms?
- Machine used: dual, 3.33 GHz, 6 core Intel Xeon 5670 processor
- Comparisons:
  - Various customized SVMs
  - Parallel AdaBoost
- Synthetic data: Sine wave and Checkerboard
- Real data: KDD Cup 1999

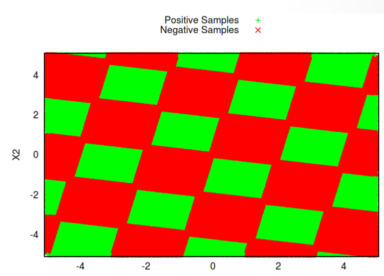
Scalability – Experiment 4

Synthetic data

Sine wave



Checkerboard



Scalability – Experiment 4

Synthetic data: Results

Algorithm	*Checkerboard* Speed	Acc	*Sine Wave* Speed	Acc
SVM				
LP-SVM (Linear)	44.20	50.23	33.20	68.80
LP-SVM (RBF)	33.20	57.11	105.56	70.11
LibLinear	133.20	50.08	203.12	68.60
SGDT (10 iterations)	4.20	54.49	4.20	54.89
SVM-PERF (Linear)	1.10	51.01	2.01	61.90
BVM (RBF)	1.80	50.03	1.20	49.03
LibSVM (RBF, 0.1% data)	136.20	98.20	423.23	70.80
Boosting				
AdaBoostM1	38.21	51.25	30.71	74.25
ParalleAdalBoost (9 threads,10 iterations)	17.90	51.22	13.90	78.30
PSBML				
PSBML (C4.5)	123.10	99.49	193.10	99.56

Scalability – Experiment 4

Real data: Results

Algorithm	Training Time (secs)	MisClass
SVM		
LibLinear	80.20	25447.3
LibSVM (RBF, 1% data)	90.20	25517.8
LibSVM (RBF, 10% data)	1495.20	25366.1
SGDT (10 iterations)	211.10	121301
SVM-PERF (Linear)	2.90	25877.1
BVM (RBF)	3.20	25451.3
Boosting		
AdaBoostM1	13296.42	190103.3
ParallelAdaBoost (9 threads, 10 iterations)	202.30	26170.2
PSBML		
PSBML(C4.5)	2913.10	21089.8

Impact of Noise – Experiment 5

- Goal: test robustness in presence of noise

- Comparisons:
  - AdaBoost
    - decision stumps
    - Naïve Bayes

Impact of Noise – Experiment 5

AUC Results

	Adult	W8A	ICJNN1	Cod	Cover
No Noise					
AdaBoostM1/DS	87.10	77.80	**93.40**	92.80	75.70
AdaBoostM1/NB	87.20	93.30	84.30	**95.70**	85.30
PSBML/NB	**90.69**	**96.10**	81.79	91.79	**87.31**
10% Noise					
AdaBoostM1/DS	85.70	58.90	**92.82**	92.20	75.10
AdaBoostM1/NB	85.80	83.40	79.80	**95.10**	85.10
PSBML/NB	**90.46**	**96.01**	77.46	88.06	**87.14**
20% Noise					
AdaBoostM1/DS	85.10	57.10	**92.30**	92.10	75.10
AdaBoostM1/NB	84.88	79.01	79.70	**94.90**	84.20
PSBML/NB	**90.10**	**95.97**	77.42	86.98	**87.11**

Impact of Noise – Experiment 5

$$impact = \frac{1}{N}\sum_{i=1}^{N}(auc^{i}_{no-noise} - auc^{i}_{noise})$$

10% noise	
AdaBoostM1/DS:	4.41
AdaBoostM1/NB:	3.32
PSBML/NB:	**1.71**
20% noise	
AdaBoostM1/DS:	5.02
AdaBoostM1/NB:	4.62
PSBML/NB:	**2.02**

Conclusion

- A parallel boosting framework was introduced
- Behavior modeled in terms of well-known statistical methods
- Extensive results on accuracy, scalability, and resilience to noise
- Future work
  - Extension to semi-supervised setting
  - Extension to unsupervised setting
  - Implementation using a distributed architecture in combination with Map-reduce

Reference

- U. Kamath, C. Domeniconi, and K. De Jong, An analysis of a spatial EA parallel boosting algorithm, GECCO 2013.