

Feature Induction in Machine Learning

M.Tech Seminar Presentation

Amrita Saha

under the guidance of
Prof. Ganesh Ramakrishnan

Nov. 04, 2011



- 1 Introduction
- 2 Features and Representation
- 3 Strategic Search in Inductive Logic Programming
 - Optimal Search for Inductive Logic Programming
 - Future Work-More Efficient search
- 4 Searching for globally Optimum set of features
 - Hierarchical Kernel Learning on structured output spaces
- 5 Future Works
 - Applications of Hierarchical Kernel Learning
 - Problems with practical applicability
 - clustered HKL
 - Randomized HKL

1

Introduction

Feature Induction

- Learning features for any predictive learning task
 - identifying the key attributes which characterise the domain
 - and have good generalization performance
 - but yet are human-interpretable.
- to induce a compact set of relevant features
- understand how they interact with the strengths and limitations of the predictive learner

Feature Induction

- Learning features for any predictive learning task
 - identifying the key attributes which characterise the domain
 - and have good generalization performance
 - but yet are human-interpretable.
- to induce a compact set of relevant features
- understand how they interact with the strengths and limitations of the predictive learner

Feature Induction

- Learning features for any predictive learning task
 - identifying the key attributes which characterise the domain
 - and have good generalization performance
 - but yet are human-interpretable.
- to induce a compact set of relevant features
- understand how they interact with the strengths and limitations of the predictive learner

Feature Induction

- Learning features for any predictive learning task
 - identifying the key attributes which characterise the domain
 - and have good generalization performance
 - but yet are human-interpretable.
- to induce a compact set of relevant features
- understand how they interact with the strengths and limitations of the predictive learner

Feature Induction

- Learning features for any predictive learning task
 - identifying the key attributes which characterise the domain
 - and have good generalization performance
 - but yet are human-interpretable.
- to induce a compact set of relevant features
- understand how they interact with the strengths and limitations of the predictive learner

Feature Induction

- Learning features for any predictive learning task
 - identifying the key attributes which characterise the domain
 - and have good generalization performance
 - but yet are human-interpretable.
- to induce a compact set of relevant features
- understand how they interact with the strengths and limitations of the predictive learner

My Classification of Composite Features

- **Non-Linear Features**

- Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - usually not compact and human-interpretable.
- Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

My Classification of Composite Features

- Non-Linear Features
 - Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - usually not compact and human-interpretable.
 - Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

My Classification of Composite Features

- Non-Linear Features
 - Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - ususally not compact and human-interpretable.
 - Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

My Classification of Composite Features

- Non-Linear Features
 - Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - ususally not compact and human-interpretable.
 - Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

My Classification of Composite Features

- Non-Linear Features
 - Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - ususally not compact and human-interpretable.
 - Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

My Classification of Composite Features

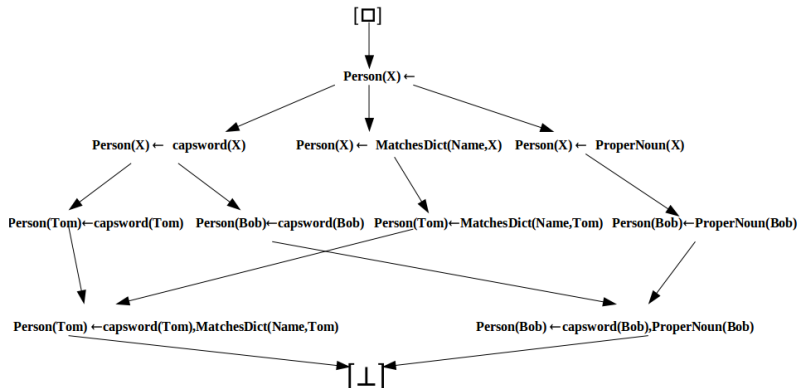
- Non-Linear Features
 - Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - ususally not compact and human-interpretable.
 - Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

My Classification of Composite Features

- Non-Linear Features
 - Continuous function: for e.g. represented by a kernel (polynomial / gaussian kernel)
 - usually not compact and human-interpretable.
 - Logical Features, i.e.
 - Features having arbitrary FOL representation
 - Propositional Features (Conjunction, Disjunction)
 - Aggregation of other features, say AVERAGE or MAX of other features.

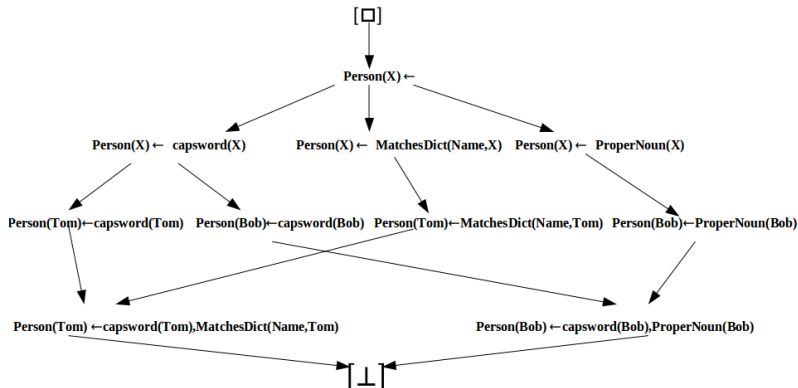
Features as Hypothesis

- the search space of features as lattice structure, defined on *covers* relation
- upward & downward refinement operators on the lattice structure



Features as Hypothesis

- the search space of features as lattice structure, defined on *covers* relation
- upward & downward refinement operators on the lattice structure



2.1

Strategic Search in Inductive Logic Programming

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

- B : background knowledge of clauses = $\{C_1, C_2, \dots\}$
- E : set of examples = $E^+ \cup E^-$ where:
- H : output of the algorithm given B and E
 - *Prior Satisfiability.* $B \not\models E^-$
 - *Posterior Satisfiability.* $(B \wedge H) \not\models E^-$
 - *Prior Necessity.* $B \not\models E^+$
 - *Posterior Sufficiency.* $B \cup H \models e_1 \wedge e_2 \wedge \dots$

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
 - begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
 - end
 - return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square\}$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square\}$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
and $B \wedge h_j \wedge E^- \not\models \square\}$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

Predictive Inductive Logic Programming

random_greedy_ilp(B, E, \mathcal{L}, f, n)

- while $E_i^+ \neq \emptyset$
- begin
 - $S := \text{randomsample}(n, E_{i-1}^+)$
 - for $e_j \in S$
 - Find $h_{best} = \max_{h \in H_S} f(h, B, E_{i-1} \cup E^-); h_j \in \mathcal{L} \text{ and } B \wedge h_j \models e_j$
 and $B \wedge h_j \wedge E^- \not\models \square$
 - $H_i = H_{i-1} \cup \{h_{best}\}$
 - $E_i = \{e : e \in E_{i-1}^+ \text{ and } B \wedge h_i \models e\}$
 - $E_i^+ = E_{i-1}^+ - E_i$
- end
- return H_i

Figure:

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices

- rlgg lattice
- bottom clause lattice of an example

most specialized clause covering at least that example forms the bottom lattice

each node in the lattice is a more general clause, with the most general at the top

relations in this lattice are generalization and specialization

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices
 - rlgg lattice
 - bottom clause lattice of an example
 - most specialised clause covering at least that example forms the lattice bottom
 - each node in the lattice is a more general clause, with the most general at the top
 - relations in the lattice are generalisation and specialization

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices
 - rlgg lattice
 - bottom clause lattice of an example
 - most specialised clause covering at least that example forms the lattice bottom
 - each node in the lattice is a more general clause, with the most general at the top
 - relations in the lattice are generalisation and specialization

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices
 - rlgg lattice
 - bottom clause lattice of an example
 - most specialised clause covering at least that example forms the lattice bottom
 - each node in the lattice is a more general clause, with the most general at the top
 - relations in the lattice are generalisation and specialization

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices
 - rlgg lattice
 - bottom clause lattice of an example
 - most specialised clause covering at least that example forms the lattice bottom
 - each node in the lattice is a more general clause, with the most general at the top
 - relations in the lattice are generalisation and specialization

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices
 - rlgg lattice
 - bottom clause lattice of an example
 - most specialised clause covering at least that example forms the lattice bottom
 - each node in the lattice is a more general clause, with the most general at the top
 - relations in the lattice are generalisation and specialization

SearchSpace for Clauses

- Search Space is a lattice
- there can be two types of lattices
 - rlgg lattice
 - bottom clause lattice of an example
 - most specialised clause covering at least that example forms the lattice bottom
 - each node in the lattice is a more general clause, with the most general at the top
 - relations in the lattice are generalisation and specialization

Bottom Clause Lattice

B:

grandfather(X,Y) \leftarrow father(X,Z),parent(Z,Y)

father(henry, jane) \leftarrow

mother(jane, john) \leftarrow

mother(jane, alice) \leftarrow

E:

grandfather(henry,john).

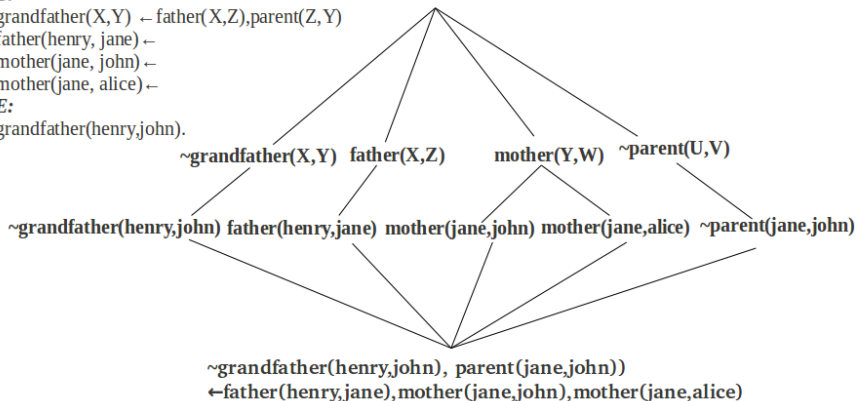


Figure: Bottom Clause lattice: Built on top of Most specialised clause covering that example

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Theory of Optimal Search

- **Problem:** Searching for a single target ball from 'n' bins; j 'th containing ' n_j ' balls
- With a fixed total resource, how to allocate optimally resources for the actual search to the different bins?
- How to understand when optimal resource allocation will be profitable over uniform allocation?
- $Bin_j \equiv \perp_j$
- balls \equiv clauses
- target \equiv best (Highest Utility) clause
- optimal allocation \equiv effort f_j expended on \perp_j search space

Parameters of Optimization Problem

- Solution to optimized allocation problem

Notation	Meaning
$p(j)$	Prob(target is in bin j)
$f(j)$	Effort allocated to bin j
\mathbf{f}	Vector of allocations $[f(1), f(2), \dots, f(n)]$
$P[\mathbf{f}]$	Prob(detecting target with allocation vector \mathbf{f})
$b(j, f(j))$	Detecting the target in j^{th} bin with effort $f(j)$ given target is in the j^{th} bin
$c(j, f(j))$	Cost of applying effort $f(j)$ in bin j
$C[\mathbf{f}]$	Total cost with allocation function \mathbf{f}

- Given, $p(j)$ and $c(j, f(j)), b(j, f(j))$ for each bin,

$$\begin{aligned}
 & \text{maximize}_{\mathbf{f}} P[\mathbf{f}] \\
 \text{s.t.} \quad & C[\mathbf{f}] \leq nK \\
 & \sum_{j=1}^n p(j) = 1 \\
 \text{and } \forall_j \quad & f(j) \geq 0
 \end{aligned}$$

Parameters of Optimization Problem

- Solution to optimized allocation problem

Notation	Meaning
$p(j)$	Prob(target is in bin j)
$f(j)$	Effort allocated to bin j
\mathbf{f}	Vector of allocations $[f(1), f(2), \dots, f(n)]$
$P[\mathbf{f}]$	Prob(detecting target with allocation vector \mathbf{f})
$b(j, f(j))$	Detecting the target in j^{th} bin with effort $f(j)$ given target is in the j^{th} bin
$c(j, f(j))$	Cost of applying effort $f(j)$ in bin j
$C[\mathbf{f}]$	Total cost with allocation function \mathbf{f}

- Given, $p(j)$ and $c(j, f(j)), b(j, f(j))$ for each bin,

$$\begin{aligned}
 & \text{maximize}_{\mathbf{f}} P[\mathbf{f}] \\
 & \text{s.t.} \quad C[\mathbf{f}] \leq nK \\
 & \quad \sum_{j=1}^n p(j) = 1 \\
 & \text{and } \forall_j \quad f(j) \geq 0
 \end{aligned}$$

Estimating parameters of the Optimization Problem

- **Prior Distribution($p(j)$)**
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- **Form of conditional probability of finding target $b(j, f(j))$**
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- **Cost $c(j, f(j))$ is taken as $f(j)$**

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{|\perp_j|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{|\perp_j|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Estimating parameters of the Optimization Problem

- Prior Distribution($p(j)$)
 - Giving importance to rare events of getting a 'good' clause
 - Fitting Gumbel max-order distribution over each \perp_i
- Form of conditional probability of finding target $b(j, f(j))$
 - desirably, smooth function
 - decreasing function of search space \perp_j size
 - increasing function of effort $f(j)$ allocated
 - combining to a heuristic form $b(j, f(j)) = 1 - e^{-\frac{f(j)}{\|\perp_j\|}}$
- Cost $c(j, f(j))$ is taken as $f(j)$

Optimization Problem

- say X_j is event of (finding target in \perp_j with effort $f(j)$)
- $P(X_j) = P(\text{target preset in } \perp_j) \cdot P(X_j | \text{target preset in } \perp_j)$
- Block-Coordinate Descent, solving over 2 variables $f(i), f(j)$, at a time
- $f^* = \underset{f(i), f(j)}{\operatorname{argmin}} \sum_{k=1}^n -p(k) \cdot (1 - e^{\frac{-f(k)}{|\perp_k|}})$
- s.t. $f(i) + f(j) \leq K'$
- $f(i), f(j) \geq 0$
where $K' = nK - \sum_{k=1, k \neq i, j}^n f(k)$

Optimization Problem

- say X_j is event of (finding target in \perp_j with effort $f(j)$)
- $P(X_j) = P(\text{target preset in } \perp_j) \cdot P(X_j | \text{target preset in } \perp_j)$
- Block-Coordinate Descent, solving over 2 variables $f(i), f(j)$, at a time
- $f^* = \underset{f(i), f(j)}{\operatorname{argmin}} \sum_{k=1}^n -p(k) \cdot (1 - e^{\frac{-f(k)}{|\perp_k|}})$
- s.t. $f(i) + f(j) \leq K'$
- $f(i), f(j) \geq 0$
where $K' = nK - \sum_{k=1, k \neq i, j}^n f(k)$

Optimization Problem

- say X_j is event of (finding target in \perp_j with effort $f(j)$)
- $P(X_j) = P(\text{target preset in } \perp_j) \cdot P(X_j | \text{target preset in } \perp_j)$
- Block-Coordinate Descent, solving over 2 variables $f(i), f(j)$, at a time
- $f^* = \underset{f(i), f(j)}{\operatorname{argmin}} \sum_{k=1}^n -p(k) \cdot (1 - e^{\frac{-f(k)}{|\perp_k|}})$
- s.t. $f(i) + f(j) \leq K'$
- $f(i), f(j) \geq 0$
where $K' = nK - \sum_{k=1, k \neq i, j}^n f(k)$

Optimization Problem

- say X_j is event of (finding target in \perp_j with effort $f(j)$)
- $P(X_j) = P(\text{target preset in } \perp_j) \cdot P(X_j | \text{target preset in } \perp_j)$
- Block-Coordinate Descent, solving over 2 variables $f(i), f(j)$, at a time
- $f^* = \underset{f(i), f(j)}{\operatorname{argmin}} \sum_{k=1}^n -p(k) \cdot (1 - e^{\frac{-f(k)}{|\perp_k|}})$
- s.t. $f(i) + f(j) \leq K'$
- $f(i), f(j) \geq 0$
where $K' = nK - \sum_{k=1, k \neq i, j}^n f(k)$

Optimization Problem

- say X_j is event of (finding target in \perp_j with effort $f(j)$)
- $P(X_j) = P(\text{target preset in } \perp_j) \cdot P(X_j | \text{target preset in } \perp_j)$
- Block-Coordinate Descent, solving over 2 variables $f(i), f(j)$, at a time
- $f^* = \underset{f(i), f(j)}{\operatorname{argmin}} \sum_{k=1}^n -p(k) \cdot (1 - e^{\frac{-f(k)}{|\perp_k|}})$
- s.t. $f(i) + f(j) \leq K'$
- $f(i), f(j) \geq 0$
where $K' = nK - \sum_{k=1, k \neq i, j}^n f(k)$

Optimization Problem

- say X_j is event of (finding target in \perp_j with effort $f(j)$)
- $P(X_j) = P(\text{target preset in } \perp_j) \cdot P(X_j | \text{target preset in } \perp_j)$
- Block-Coordinate Descent, solving over 2 variables $f(i), f(j)$, at a time
- $f^* = \underset{f(i), f(j)}{\operatorname{argmin}} \sum_{k=1}^n -p(k) \cdot (1 - e^{\frac{-f(k)}{|\perp_k|}})$
- s.t. $f(i) + f(j) \leq K'$
- $f(i), f(j) \geq 0$
where $K' = nK - \sum_{k=1, k \neq i, j}^n f(k)$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$

- Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$

- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$

- Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$

- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

- $f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

- $f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$

- Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$

- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$

- Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$

- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

- $f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

- $f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$
 - Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$
- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$
 - Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$
- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

$$f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$$

$$f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$
 - Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$
- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$
 - Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$
- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

$$f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$$

$$f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$
 - Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$
- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$
 - Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$
- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

$$\bullet f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$$

$$\bullet f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$

- Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$

- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$

- Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$

- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

- $f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

- $f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

Feasible Solutions : KKT

- **Case 1:** When $\lambda_{ij} > 0$, $\lambda_i > 0$ & $\lambda_j = 0$

- Condition $K' < |\perp_j| \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)$

- **Case 2:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j > 0$

- Condition: $K' < |\perp_i| \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)$

- **Case 3:** When $\lambda_{ij} > 0$, $\lambda_i = 0$ & $\lambda_j = 0$

- $f(i) = \frac{\frac{K'}{|\perp_j|} + \log \left(\frac{p(i) \cdot |\perp_j|}{p(j) \cdot |\perp_i|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

- $f(j) = \frac{\frac{K'}{|\perp_i|} + \log \left(\frac{p(j) \cdot |\perp_i|}{p(i) \cdot |\perp_j|} \right)}{\frac{1}{|\perp_i|} + \frac{1}{|\perp_j|}}$

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment

- Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
- Bin Mean-utility
- Low/High resource bounds

- Types of Experiments

- Synthetic Experiments
- Real Experiments
- Experiments on Phase Transition datasets

- Performance Evaluation

- take 2 bins i and j at a time with total resource $2K$
- Sample to find the goodness of the bins as p
- use p to find f
- Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
- If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Empirical Evaluation

Optimal-Vs-Uniform Search Performance

- Parameters of Experiment
 - Location of the target
 - Case 1: in 1st bin
 - Case 2: beyond the 1st bin
 - Bin Mean-utility
 - Low/High resource bounds
- Types of Experiments
 - Synthetic Experiments
 - Real Experiments
 - Experiments on Phase Transition datasets
- Performance Evaluation
 - take 2 bins i and j at a time with total resource $2K$
 - Sample to find the goodness of the bins as p
 - use p to find f
 - Given $(f(i), f(j))$ and (K, K) , the one with higher utility wins
 - If both give equal utility, the one constructing lesser clauses wins

Results of Synthetic Experiments

		Bin Means							
		Substantially Different				Not Substantially Different			
Small K	Case	Optimal Search			Case	Optimal Search			
		W	L	T		W	L	T	
	1	1	0	99	1	0	0	100	
	2	71	0	29	2	9	0	91	
Large K	Case	Optimal Search			Case	Optimal Search			
		W	L	T		W	L	T	
	1	0	0	100	1	0	10	90	
	2	27	0	73	2	11	0	89	

Table: For target location being uniformly distributed in Target bin

Results of Synthetic Experiments

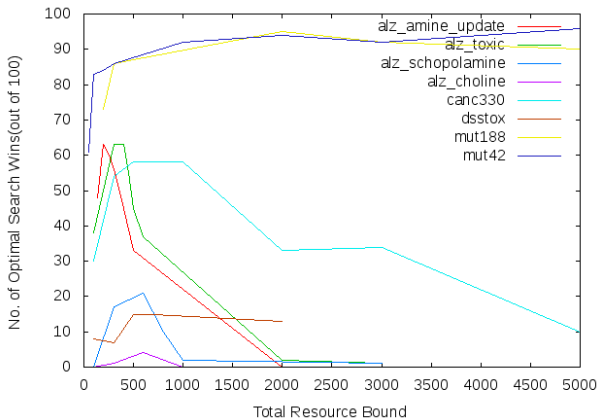
		Bin Means						
		Substantially Different			Not Substantially Different			
Small K		Optimal Search				Optimal Search		
	Case	W	L	T	Case	W	L	T
	1	0	0	100	1	0	0	100
	2	98	0	2	2	10	0	90
Large K		Optimal Search				Optimal Search		
	Case	W	L	T	Case	W	L	T
	1	0	0	100	1	0	0	100
	2	99	0	1	2	0	0	100

Table: For target location being binomially distributed in Target bin

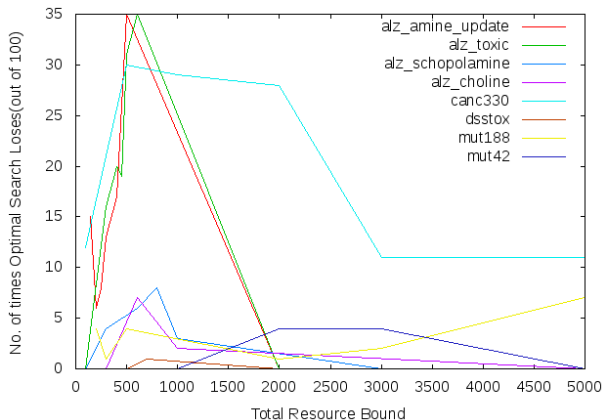
Experiments on Real Datasets

Performance Evaluation on Real Datasets				
Datasets	Resource	W	L	T
alz_amine_update	140	48	15	37
	200	63	6	31
	400	45	17	38
	500	33	35	32
	1000	0	0	100
alz_toxic	100	38	0	62
	300	63	16	21
	600	37	35	28
	2000	2	0	98
mut188	200	73	4	23
	500	87	4	9
	2000	95	1	4
mut42	50	61	0	39
	300	86	0	14
	500	78	6	16
	700	82	0	18
	5000	96	0	5

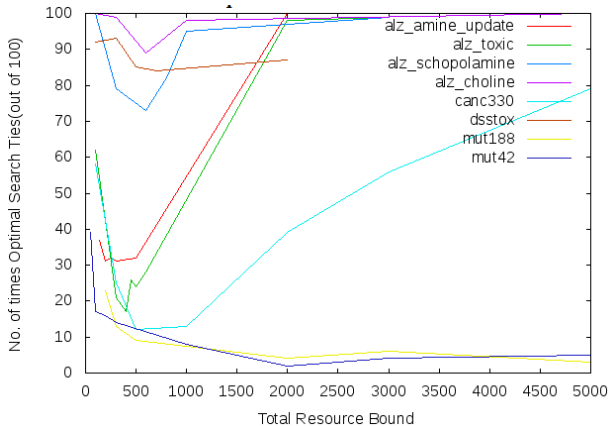
Wins of optimal search over uniform search



Losses of optimal search over uniform search



Ties of optimal search with uniform search



Observations

- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

Observations

- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

Observations

- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

Observations

- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

Observations

- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

Observations

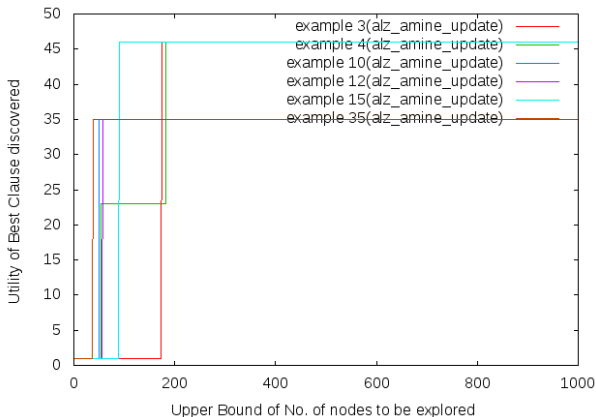
- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

Observations

- graph of Wins:nonWins of optimal search increases with total resource bound till a peak and falls
- no of losses fairly low for most datasets
- margin of optimal search's win
(for alz_amine_update dataset; $\|E^+\| = 100, \|E^-\| = 300,$)
 - Max difference in (Pos-Neg) utility: 89
 - Mean difference (Pos-Neg) utility: 55
 - Max clause difference: 29.67% of Total Resource
 - Mean clause difference: 1.94% of Total Resource

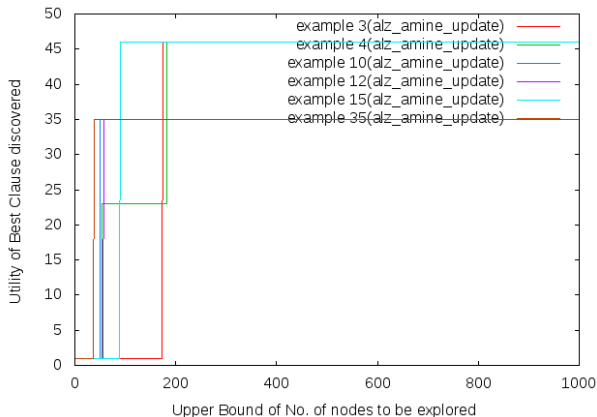
Characteristics of Datasets

- some datasets have large number of 'useless' examples (e.g. alz_schopolamine)
- some datasets show 'needle-in-a-haystack' characteristics



Characteristics of Datasets

- some datasets have large number of 'useless' examples (e.g. alz_schopolamine)
- some datasets show 'needle-in-a-haystack' characteristics



Phase Transition Datasets

Performance Evaluation on Phase Transition Datasets				
Datasets	'm' value	W	L	T
Botta Datasets	m=4	37	0	63
	m=5	39	0	61
	m=6	58	0	42
	m=7	55	0	45

Table: Results of wins and losses for low resource bound on the phase transition datasets of different sizes.

m = No. of literals in dataset (all ' m ' literals are in true hypothesis)

Observations on Phase transition Datasets

- For $m \leq 7$, for small resources optimal search wins over uniform search substantial of times
- there is no case of optimal search losing
- as m increases notion of small and large resource increases exponentially
- as the resource bound increases from 10^4 to 10^7 , then ratio of Wins:NonWins increases from 37:63 to 26:33 ($m=5$)
- One reason of large number of ties can be: every example is generated from a single hypothesis

Observations on Phase transition Datasets

- For $m \leq 7$, for small resources optimal search wins over uniform search substantial of times
- there is no case of optimal search losing
- as m increases notion of small and large resource increases exponentially
- as the resource bound increases from 10^4 to 10^7 , then ratio of Wins:NonWins increases from 37:63 to 26:33 ($m=5$)
- One reason of large number of ties can be: every example is generated from a single hypothesis

Observations on Phase transition Datasets

- For $m \leq 7$, for small resources optimal search wins over uniform search substantial of times
- there is no case of optimal search losing
- as m increases notion of small and large resource increases exponentially
- as the resource bound increases from 10^4 to 10^7 , then ratio of Wins:NonWins increases from 37:63 to 26:33 ($m=5$)
- One reason of large number of ties can be: every example is generated from a single hypothesis

Observations on Phase transition Datasets

- For $m \leq 7$, for small resources optimal search wins over uniform search substantial of times
- there is no case of optimal search losing
- as m increases notion of small and large resource increases exponentially
- as the resource bound increases from 10^4 to 10^7 , then ratio of Wins:NonWins increases from 37:63 to 26:33 ($m=5$)
- One reason of large number of ties can be: every example is generated from a single hypothesis

Observations on Phase transition Datasets

- For $m \leq 7$, for small resources optimal search wins over uniform search substantial of times
- there is no case of optimal search losing
- as m increases notion of small and large resource increases exponentially
- as the resource bound increases from 10^4 to 10^7 , then ratio of Wins:NonWins increases from 37:63 to 26:33 ($m=5$)
- One reason of large number of ties can be: every example is generated from a single hypothesis

Identifying Some Synthetic Parameters for real Datasets

- Resource Bound (from Real Experiments)
 - Difference in Bin Means(from Synthetic Experiments)
- Target location can be ignored as Case-1 is a degenerate case of no-gain no loss
- For resource bounds; only low resource bounds are interesting
 - Is resource sufficient for exhaustive search?

Identifying Some Synthetic Parameters for real Datasets

- Resource Bound (from Real Experiments)
- Difference in Bin Means(from Synthetic Experiments)
- Target location can be ignored as Case-1 is a degenerate case of no-gain no loss
- For resource bounds; only low resource bounds are interesting
 - Is resource sufficient for exhaustive search?

Identifying Some Synthetic Parameters for real Datasets

- Resource Bound (from Real Experiments)
- Difference in Bin Means(from Synthetic Experiments)
- Target location can be ignored as Case-1 is a degenerate case of no-gain no loss
- For resource bounds; only low resource bounds are interesting
 - Is resource sufficient for exhaustive search?

Identifying Some Synthetic Parameters for real Datasets

- Resource Bound (from Real Experiments)
- Difference in Bin Means(from Synthetic Experiments)
- Target location can be ignored as Case-1 is a degenerate case of no-gain no loss
- For resource bounds; only low resource bounds are interesting
 - Is resource sufficient for exhaustive search?

Future Works

Future Works

- Problem of useless examples:

- Preprocessing: coarse-grained pruning of search-spaces
- clustering of bottom clauses of examples, based on some similarity metric
- estimating degree of overlap between bottom clause lattices of two examples-
- without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?

- Problem of Selecting Next Search Space

- the actual search chooses the next bottom-clause randomly
- greedy method of searching for bottom-clause lattice which maximises utility
- Submodular function?

Future Works

- Problem of useless examples:

- Preprocessing: coarse-grained pruning of search-spaces
- clustering of bottom clauses of examples, based on some similarity metric
- estimating degree of overlap between bottom clause lattices of two examples-
- without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?

- Problem of Selecting Next Search Space

- the actual search chooses the next bottom-clause randomly
- greedy method of searching for bottom-clause lattice which maximises utility
- Submodular function?

Future Works

- Problem of useless examples:

- Preprocessing: coarse-grained pruning of search-spaces
- clustering of bottom clauses of examples, based on some similarity metric
- estimating degree of overlap between bottom clause lattices of two examples-
- without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?

- Problem of Selecting Next Search Space

- the actual search chooses the next bottom-clause randomly
- greedy method of searching for bottom-clause lattice which maximises utility
- Submodular function?

Future Works

- Problem of useless examples:

- Preprocessing: coarse-grained pruning of search-spaces
- clustering of bottom clauses of examples, based on some similarity metric
- estimating degree of overlap between bottom clause lattices of two examples-
- without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?

- Problem of Selecting Next Search Space

- the actual search chooses the next bottom-clause randomly
- greedy method of searching for bottom-clause lattice which maximises utility
- Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, → higher overlap
 - LGG of the two examples → bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, → higher overlap
 - LGG of the two examples → bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, → higher overlap
 - LGG of the two examples → bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

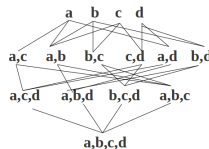
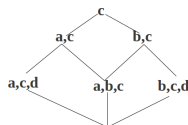
Future Works

- Problem of useless examples:
 - Preprocessing: coarse-grained pruning of search-spaces
 - clustering of bottom clauses of examples, based on some similarity metric
 - estimating degree of overlap between bottom clause lattices of two examples-
 - without **actually constructing** bottom clause lattice
 - If bottom-point of overlap-lattice is 'close' to bottom of both lattice, \rightarrow higher overlap
 - LGG of the two examples \rightarrow bottom point of overlap lattice
 - 'how close' is good enough for such inferencing?
- Problem of Selecting Next Search Space
 - the actual search chooses the next bottom-clause randomly
 - greedy method of searching for bottom-clause lattice which maximises utility
 - Submodular function?

Future Works

● Problem of needle-in-a-haystack

- Search-Space too limited & fine-grained
- searching is very slow

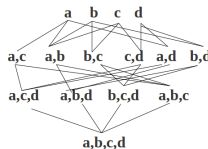
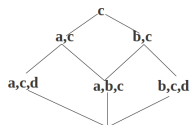


● Searching in rlgg lattice

- Search bottom-up; using active-set approach
- parallelize rlgg construction
- in every iteration select good rlgs using L1 SVM
- Add them to active-set
- Problem: rlgs are too large to handle
- no. of rlgs constructed → exponential of no. of examples
- Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

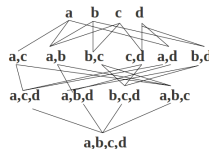
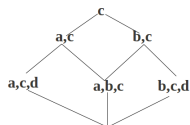
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed → exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

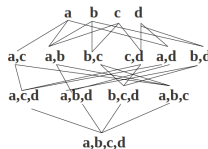
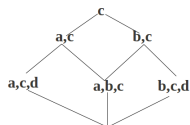
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgg's are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

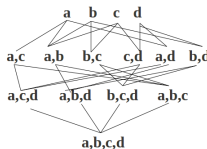
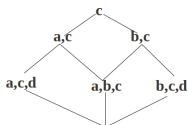
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

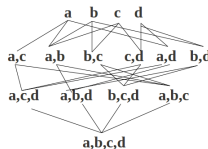
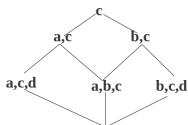
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

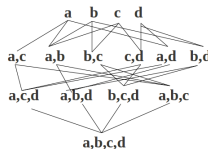
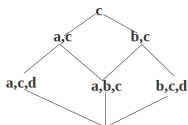
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

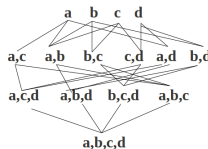
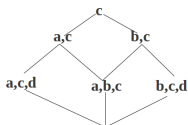
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

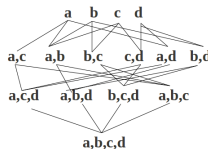
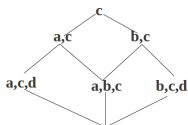
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

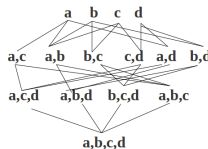
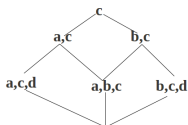
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

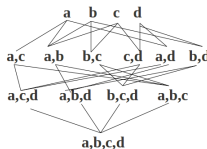
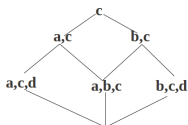
- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgg's are too large to handle
 - no. of rlgs constructed → exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Future Works

- Problem of needle-in-a-haystack
 - Search-Space too limited & fine-grained
 - searching is very slow



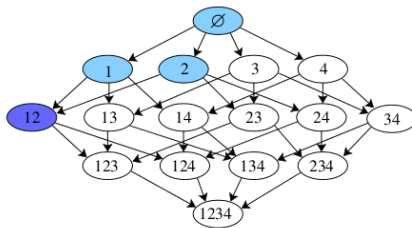
- Searching in rlgg lattice
 - Search bottom-up; using active-set approach
 - parallelize rlgg construction
 - in every iteration select good rlgs using L1 SVM
 - Add them to active-set
 - Problem: rlgs are too large to handle
 - no. of rlgs constructed \rightarrow exponential of no. of examples
 - Solution: Clustering of similar examples and building rlgg lattice on the clusters

Searching for globally Optimum set of features

Hierarchical Kernel Learning

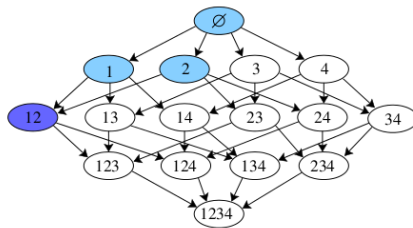
- Exploiting

- structure over exponential sized feature space
- sparsity in feature space



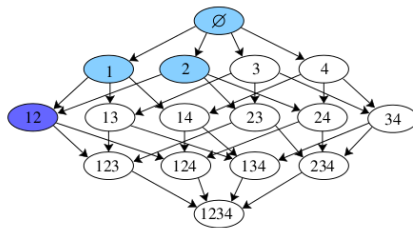
Hierarchical Kernel Learning

- Exploiting
 - structure over exponential sized feature space
 - sparsity in feature space



Hierarchical Kernel Learning

- Exploiting
 - structure over exponential sized feature space
 - sparsity in feature space



Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(poly(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(poly(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(poly(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(poly(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(poly(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(\text{poly}(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(\text{poly}(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Hierarchical Kernel Learning(HKL)

- Multiple Kernel learning (MKL) approach allows the system to learn good kernels
- Hierarchical kernel learning;
 - MKL with exponential number of base kernels
 - kernels form hierarchy
- Choice of ancestor norm or descendant norm - for top-down method like HKL
- Active set algorithm- time complexity $O(\text{poly}(\text{No of features selected}))$
- Stopping condition- Sufficiency Condition Reached
- Boolean-features case reduces to computationally efficient, $\sum_{v \in V} K(x_i, x_j) = \prod_{f \in F} (1 + \phi(x_i)\phi(x_j))$

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

- **Application - Activity Recognition using sensors**
- HMM structure on output space Y
- conjunctive feature space $\Phi(X)$
- feature vector $\Psi(X, Y)$ depends on emission and transition features
- hierarchical kernel learning in conjunctive feature space (emission lattice)
- feature space is sparse

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

- Application - Activity Recognition using sensors
- HMM structure on output space Y
- conjunctive feature space $\Phi(X)$
- feature vector $\Psi(X, Y)$ depends on emission and transition features
- hierarchical kernel learning in conjunctive feature space (emission lattice)
- feature space is sparse

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

- Application - Activity Recognition using sensors
- HMM structure on output space Y
- conjunctive feature space $\Phi(X)$
- feature vector $\Psi(X, Y)$ depends on emission and transition features
- hierarchical kernel learning in conjunctive feature space (emission lattice)
- feature space is sparse

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

- Application - Activity Recognition using sensors
- HMM structure on output space Y
- conjunctive feature space $\Phi(X)$
- feature vector $\Psi(X, Y)$ depends on emission and transition features
- hierarchical kernel learning in conjunctive feature space (emission lattice)
- feature space is sparse

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

- Application - Activity Recognition using sensors
- HMM structure on output space Y
- conjunctive feature space $\Phi(X)$
- feature vector $\Psi(X, Y)$ depends on emission and transition features
- hierarchical kernel learning in conjunctive feature space (emission lattice)
- feature space is sparse

Rule Ensemble Learning using HKL on structured output spaces (RelHklActivity)

- Application - Activity Recognition using sensors
- HMM structure on output space Y
- conjunctive feature space $\Phi(X)$
- feature vector $\Psi(X, Y)$ depends on emission and transition features
- hierarchical kernel learning in conjunctive feature space (emission lattice)
- feature space is sparse

Activity Recognition Lattice

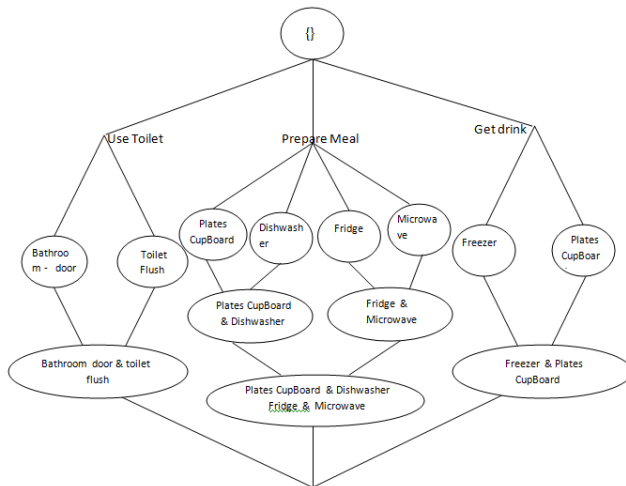


Figure: Activity Recognition Lattice

Formulation

- SVM formulation

$$\min_{\mathbf{f}, \xi} \frac{1}{2} \Omega_E(\mathbf{f}_E)^2 + \frac{1}{2} \Omega_T(\mathbf{f}_T)^2 + \frac{C}{m} \sum_{i=1}^m \xi_i,$$

$$\forall i, \forall Y \in \mathcal{Y} \setminus Y_i : \langle \mathbf{f}, \psi_i^\delta(Y) \rangle \geq 1 - \frac{\xi_i}{\Delta(Y_i, Y)}$$
$$\forall i : \xi_i \geq 0$$

- Decision function for prediction

$$\mathcal{F}(X; \mathbf{f}) = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \langle \mathbf{f}, \psi(X, Y) \rangle \quad (1)$$

Formulation

- SVM formulation

$$\min_{\mathbf{f}, \xi} \frac{1}{2} \Omega_E(\mathbf{f}_E)^2 + \frac{1}{2} \Omega_T(\mathbf{f}_T)^2 + \frac{C}{m} \sum_{i=1}^m \xi_i,$$
$$\forall i, \forall Y \in \mathcal{Y} \setminus Y_i : \langle \mathbf{f}, \psi_i^\delta(Y) \rangle \geq 1 - \frac{\xi_i}{\Delta(Y_i, Y)}$$
$$\forall i : \xi_i \geq 0$$

- Decision function for prediction

$$\mathcal{F}(X; \mathbf{f}) = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \langle \mathbf{f}, \psi(X, Y) \rangle \quad (1)$$

SVM formulation for RelHklActivity

- $\Omega_E(\mathbf{f}_E) = \sum_{v \in \mathcal{V}_E} d_v \|\mathbf{f}_{ED(v)}\|_\rho, \rho \in (1, 2]$
- $\Omega_T(\mathbf{f}_T) = \left(\sum_i f_{Ti}^2\right)^{\frac{1}{2}}$
- SVM formulation

$$\min_{\mathbf{f}, \xi} \frac{1}{2} \left(\sum_{v \in \mathcal{V}} d_v \|\mathbf{f}_{ED(v) \cap \mathcal{V}}\|_\rho \right)^2 + \frac{1}{2} \|\mathbf{f}_T\|_2^2 + \frac{C}{m} \sum_{i=1}^m \xi_i,$$

$$\forall i, \forall Y \in \mathcal{Y} \setminus Y_i :$$

$$- \left(\sum_{v \in \mathcal{V}} \langle f_{Ev}, \psi_{Evi}^\delta(Y) \rangle + \sum_{v \in \mathcal{V}_T} \langle f_{Tv}, \psi_{Tvi}^\delta(Y) \rangle + \frac{\xi_i}{\Delta(Y_i, Y)} - 1 \right) \leq 0$$

$$\forall i : -\xi_i \leq 0$$

where ρ in $(1, 2]$

SVM formulation for RelHklActivity

- $\Omega_E(\mathbf{f}_E) = \sum_{v \in \mathcal{V}_E} d_v \|\mathbf{f}_{ED(v)}\|_\rho, \rho \in (1, 2]$
- $\Omega_T(\mathbf{f}_T) = \left(\sum_i f_{Ti}^2\right)^{\frac{1}{2}}$
- SVM formulation

$$\min_{\mathbf{f}, \xi} \frac{1}{2} \left(\sum_{v \in \mathcal{V}} d_v \|\mathbf{f}_{ED(v) \cap \mathcal{V}}\|_\rho \right)^2 + \frac{1}{2} \|\mathbf{f}_T\|_2^2 + \frac{C}{m} \sum_{i=1}^m \xi_i,$$

$$\forall i, \forall Y \in \mathcal{Y} \setminus Y_i :$$

$$- \left(\sum_{v \in \mathcal{V}} \langle f_{Ev}, \psi_{Evi}^\delta(Y) \rangle + \sum_{v \in \mathcal{V}_T} \langle f_{Tv}, \psi_{Tvi}^\delta(Y) \rangle + \frac{\xi_i}{\Delta(Y_i, Y)} - 1 \right) \leq 0$$

$$\forall i : -\xi_i \leq 0$$

where ρ in $(1, 2]$

SVM formulation for RelHklActivity

- $\Omega_E(\mathbf{f}_E) = \sum_{v \in \mathcal{V}_E} d_v \|\mathbf{f}_{ED(v)}\|_\rho, \rho \in (1, 2]$
- $\Omega_T(\mathbf{f}_T) = \left(\sum_i f_{Ti}^2\right)^{\frac{1}{2}}$
- SVM formulation

$$\min_{\mathbf{f}, \xi} \frac{1}{2} \left(\sum_{v \in \mathcal{V}} d_v \|\mathbf{f}_{ED(v) \cap \mathcal{V}}\|_\rho \right)^2 + \frac{1}{2} \|\mathbf{f}_T\|_2^2 + \frac{C}{m} \sum_{i=1}^m \xi_i,$$

$$\forall i, \forall Y \in \mathcal{Y} \setminus Y_i :$$

$$- \left(\sum_{v \in \mathcal{V}} \langle f_{Ev}, \psi_{Evi}^\delta(Y) \rangle + \sum_{v \in \mathcal{V}_T} \langle f_{Tv}, \psi_{Tvi}^\delta(Y) \rangle + \frac{\xi_i}{\Delta(Y_i, Y)} - 1 \right) \leq 0$$

$$\forall i : -\xi_i \leq 0$$

where ρ in $(1, 2]$

Regularization Term

- Applying variational characterization, regularization term becomes

$$= \min_{\gamma \in \Delta_{|V|}, 1} \min_{\lambda_v \in \Delta_{|D(v)|}, \hat{\rho}} \sum_{\forall v \in V_e, w \in V_e} \delta_w(\gamma, \lambda)^{-1} \|f_{ew}\|_2^2 \text{ where}$$

$$\delta_w(\gamma, \lambda)^{-1} = \sum_{v \in A(w)} \frac{d_v^2}{\lambda_{wv} \gamma_v}$$

Dual Formulation

- Partial dual got after solving

$$\begin{aligned}
 & \max_{\alpha} \min_{\gamma \in \Delta_{|V_e|}, 1} \min_{\lambda_{v \in \Delta_{|D(v)|}}, \rho_{v \in V_e}} \\
 & - \frac{1}{2} \sum_{v \in V_e} \delta_{ev}(\gamma, \lambda) \sum_{i, y \neq y_i} \sum_{j, y' \neq y_j} \alpha_{iy} \alpha_{jy'} \delta \psi_{evj}(y') \delta \psi_{evi}(y) \\
 & - \frac{1}{2} \sum_{i, y \neq y_i} \sum_{j, y' \neq y_j} \alpha_{iy} \alpha_{jy'} \delta \psi_{ii}(y) \delta \psi_{jj}(y') \\
 & + \sum_{i, y \neq y_i} \alpha_{iy} \\
 & \sum_{y \neq y_i} \frac{\alpha_{iy}}{\Delta(y, y_i)} \leq C \forall i \\
 & \alpha_{iy} \geq 0 \forall i, y
 \end{aligned}$$

Kernel Formulation

- Joint Feature Map [4] $\Psi_e(x, y) = \Phi_e(x) \otimes \Lambda^c(y)$
- where $\Lambda^c(y) = (\delta(y, 1), \delta(y, 2), \dots, \delta(y, k))' \in \{0, 1\}^k$.
- and $\langle \Lambda^c(y), \Lambda^c(y') \rangle = \delta(y, y')$.

Kernel Formulation

- Joint Feature Map [4] $\Psi_e(x, y) = \Phi_e(x) \otimes \Lambda^c(y)$
- where $\Lambda^c(y) = (\delta(y, 1), \delta(y, 2), \dots, \delta(y, k))' \in \{0, 1\}^k$.
- and $\langle \Lambda^c(y), \Lambda^c(y') \rangle = \delta(y, y')$.

Kernel Formulation

- Joint Feature Map [4] $\Psi_e(x, y) = \Phi_e(x) \otimes \Lambda^c(y)$
- where $\Lambda^c(y) = (\delta(y, 1), \delta(y, 2), \dots, \delta(y, k))' \in \{0, 1\}^k$.
- and $\langle \Lambda^c(y), \Lambda^c(y') \rangle = \delta(y, y')$.

Emission Kernel

- for the emission kernel $K_{ew}((x_i, y_i), (x_j, y_j)) =$

$$K_{2ew}(y_i, y_j) K_{1ew}(x_i, x_j) = \sum_{p=1}^{l_{x_i}} \sum_{q=1}^{l_{x_j}} K_{ew}(x_i^p, x_j^q) \delta(y_i^p, y_j^q)$$

- Regularizer term for emission features

$$\begin{aligned} \Omega_e(f_e)^2 &= \min_{\gamma \in \Delta_{|v|, 1}} \min_{\lambda_v \in \Delta_{|D(v)|, \hat{\rho}} \forall v \in V_e} \sum_{w \in V_e} \delta_{ew}(\gamma, \lambda) \sum_{i,j} \sum_{y \neq y_i, y' \neq y_j} \alpha_{iy} \alpha_{jy'} \\ &\quad \left(\sum_{p=1}^{l_{x_i}} \sum_{q=1}^{l_{x_j}} K_{2ew}(x_i^p, x_j^q) (\delta(y_i^p, y_j^q) + \delta(y^p, y'^q) - \delta(y_i^p, y'^q) - \delta(y^p, y_j^q)) \right) \\ &= \min_{\gamma \in \Delta_{|v|, 1}} \min_{\lambda_v \in \Delta_{|D(v)|, \hat{\rho}} \forall v \in V_e} \alpha^T \left(\sum_{w \in V_e} \delta_{ew}(\gamma, \lambda) \kappa_{ew} \right) \alpha \\ &= \left(\sum_{w \in V} (\alpha^T \kappa'_{ew} \alpha)^{\bar{\rho}} \right)^{\frac{1}{\bar{\rho}}} ; \text{ where } \bar{\rho} = \frac{\rho}{2(\rho-1)} \end{aligned}$$

Emission Kernel

- for the emission kernel $K_{ew}((x_i, y_i), (x_j, y_j)) =$

$$K_{2ew}(y_i, y_j)K_{1ew}(x_i, x_j) = \sum_{p=1}^{l_{x_i}} \sum_{q=1}^{l_{x_j}} K_{ew}(x_i^p, x_j^q) \delta(y_i^p, y_j^q)$$

- Regularizer term for emission features

$$\Omega_e(f_e)^2 = \min_{\gamma \in \Delta_{|v|,1}} \min_{\lambda_v \in \Delta_{|D(v)|, \hat{\rho}} \forall v \in V_e} \sum_{w \in V_e} \delta_{ew}(\gamma, \lambda) \sum_{i,j} \sum_{y \neq y_i, y' \neq y_j} \alpha_{iy} \alpha_{jy'}$$

$$\left(\sum_{p=1}^{l_{x_i}} \sum_{q=1}^{l_{x_j}} K_{2ew}(x_i^p, x_j^q) (\delta(y_i^p, y_j^q) + \delta(y^p, y'^q) - \delta(y_i^p, y'^q) - \delta(y^p, y_j^q)) \right)$$

$$= \min_{\gamma \in \Delta_{|v|,1}} \min_{\lambda_v \in \Delta_{|D(v)|, \hat{\rho}} \forall v \in V_e} \alpha^T \left(\sum_{w \in V_e} \delta_{ew}(\gamma, \lambda) \kappa_{ew} \right) \alpha$$

$$= \left(\sum_{w \in V} (\alpha^T \kappa'_{ew} \alpha)^{\bar{\rho}} \right)^{\frac{1}{\bar{\rho}}}; \text{ where } \bar{\rho} = \frac{\rho}{2(\rho-1)}$$

Transition Kernel

- for the transition kernel

$$K_t(y_i, y_j) = \sum_{p=1}^{l_{x_i}-1} \sum_{q=1}^{l_{x_j}-1} \delta(y_i^p, y_j^q) \delta(y_i^{p+1}, y_j^{q+1})$$

- regularization term for transition features $\Omega_t(f_t)^2 =$

$$\sum_{i,j} \sum_{y \neq y_i, y' \neq y_j} \alpha_{iy} \alpha_{jy'} (K_t(y_i, y_j) + K_t(y, y') - K_t(y_i, y') - K_t(y, y_j))$$

$$= \alpha^T \kappa_t \alpha$$

Transition Kernel

- for the transition kernel

$$K_t(y_i, y_j) = \sum_{p=1}^{l_{x_i}-1} \sum_{q=1}^{l_{x_j}-1} \delta(y_i^p, y_j^q) \delta(y_i^{p+1}, y_j^{q+1})$$

- regularization term for transition features $\Omega_t(f_t)^2 =$

$$\sum_{i,j} \sum_{y \neq y_i, y' \neq y_j} \alpha_{iy} \alpha_{jy'} (K_t(y_i, y_j) + K_t(y, y') - K_t(y_i, y') - K_t(y, y_j))$$

$$= \alpha^T \kappa_t \alpha$$

Sufficiency Condition

- Taking the primal-dual gap and using $\|\cdot\|_\rho \leq \|\cdot\|_1$, stricter sufficiency condition becomes

$$(\Omega_e(f_{eW})^2 + \Omega_t(f_{tW})^2) + 2.(e_W + \varepsilon) \geq \max_{u \in \text{sources}(\mathcal{W}^c)} \sum_{i, Y \neq Y_i} \sum_{j, Y' \neq Y_j} \alpha_{\mathcal{W}}^\top_{iY} Q(u)_{i,j} \alpha_{\mathcal{W}}_{jY'}$$

- where $Q(u)_{i,j} = \sum_{p=1}^{l_i} \sum_{q=1}^{l_j} 2 \sum_{w \in D(u)} \frac{\kappa_{E_W}(\mathbf{x}_i^p, \mathbf{x}_j^q)}{(\sum_{v \in A(w) \cap D(u)} d_v)^2}$
- $d_v = 1$ for the topmost node of the lattice
 $d_v = b^i$ for node having i basic features

Sufficiency Condition

- Taking the primal-dual gap and using $\|\cdot\|_\rho \leq \|\cdot\|_1$, stricter sufficiency condition becomes

$$\max_{u \in \text{sources}(\mathcal{W}^c)} (\Omega_e(f_{eW})^2 + \Omega_t(f_{tW})^2) + 2 \cdot (e_W + \varepsilon) \geq \sum_{i, Y \neq Y_i} \sum_{j, Y' \neq Y_j} \alpha_{\mathcal{W}}^\top_{iY} Q(u)_{i,j} \alpha_{\mathcal{W}}_{jY'}$$

- where $Q(u)_{i,j} = \sum_{p=1}^{l_i} \sum_{q=1}^{l_j} 2 \sum_{w \in D(u)} \frac{\kappa_{Ew}(\mathbf{x}_i^p, \mathbf{x}_j^q)}{(\sum_{v \in A(w) \cap D(u)} d_v)^2}$
- $d_v = 1$ for the topmost node of the lattice
- $d_v = b^i$ for node having i basic features

Sufficiency Condition

- Taking the primal-dual gap and using $\|\cdot\|_\rho \leq \|\cdot\|_1$, stricter sufficiency condition becomes

$$(\Omega_e(f_{eW})^2 + \Omega_t(f_{tW})^2) + 2.(e_W + \varepsilon) \geq \max_{u \in \text{sources}(\mathcal{W}^c)} \sum_{i, Y \neq Y_i} \sum_{j, Y' \neq Y_j} \alpha_{\mathcal{W}^c i Y}^\top Q(u)_{i,j} \alpha_{\mathcal{W}^c j Y'}$$

- where $Q(u)_{i,j} = \sum_{p=1}^{l_i} \sum_{q=1}^{l_j} 2 \sum_{w \in D(u)} \frac{\kappa_{Ew}(\mathbf{x}_i^p, \mathbf{x}_j^q)}{(\sum_{v \in A(w) \cap D(u)} d_v)^2}$
- $d_v = 1$ for the topmost node of the lattice
 $d_v = b^i$ for node having i basic features

Sufficiency Condition

- For computational efficiency, employing

$$\sum_{v \in V_e} K_{ev}(x_i, x_j) = \prod_{k \in F} (1 + \Phi_e(x_i) \Phi_e(x_j)) \text{ sufficiency condition}$$

becomes,

$$(\Omega_e(f_{eW})^2 + \Omega_t(f_{tW})^2) + 2 \cdot (e_W + \varepsilon)$$

\geq

$$\max_{t \in \text{sources}(W^c)} \left(\alpha_W^T \sum_{p=1}^{l_{x_i}} \sum_{q=1}^{l_{x_j}} 2 \left(\prod_{k \in t} \left(\frac{\Phi_{ek}(x_i^p) \Phi_{ek}(x_j^q)}{(b^2)} \right) \prod_{k \notin t} \left(1 + \frac{\Phi_{ek}(x_i^p) \Phi_{ek}(x_j^q)}{(1+b)^2} \right) \right) \alpha_W \right)$$

Reduced Optimization Problem

- Complete dual got from partial dual:

$$\min_{\eta \in \Delta_{|D(v)|,1}} \left(\max_{\alpha \in S(\mathcal{Y}, C)} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \alpha^\top \mathbf{K}_T \alpha \right. \\ \left. - \frac{1}{2} \left(\sum_{w \in \mathcal{V}} \zeta_w(\eta) (\alpha^\top \mathbf{K}_{E_w} \alpha)^{\bar{\rho}} \right)^{\frac{1}{\bar{\rho}}} \right)$$

where $\zeta_w(\eta) = \left(\sum_{v \in A(w)} d_v^\rho \eta_v^{1-\rho} \right)^{\frac{1}{1-\rho}}$, $\bar{\rho} = \frac{\rho}{2(\rho-1)}$.

- Mirror Descent Algorithm to solve for α and η iteratively
- Cutting Plane Algorithm Inside Mirror Descent to handle exponential constraints in solving α .

Reduced Optimization Problem

- Complete dual got from partial dual:

$$\min_{\eta \in \Delta_{|D(v)|,1}} \left(\max_{\alpha \in S(\mathcal{Y}, C)} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \alpha^\top \mathbf{K}_T \alpha \right. \\ \left. - \frac{1}{2} \left(\sum_{w \in \mathcal{V}} \zeta_w(\eta) (\alpha^\top \mathbf{K}_{E_w} \alpha)^{\bar{\rho}} \right)^{\frac{1}{\bar{\rho}}} \right)$$

where $\zeta_w(\eta) = \left(\sum_{v \in A(w)} d_v^\rho \eta_v^{1-\rho} \right)^{\frac{1}{1-\rho}}$, $\bar{\rho} = \frac{\rho}{2(\rho-1)}$.

- Mirror Descent Algorithm to solve for α and η iteratively
- Cutting Plane Algorithm Inside Mirror Descent to handle exponential constraints in solving α .

Reduced Optimization Problem

- Complete dual got from partial dual:

$$\min_{\eta \in \Delta_{|D(v)|,1}} \left(\max_{\alpha \in S(\mathcal{Y}, C)} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \alpha^\top \mathbf{K}_T \alpha \right. \\ \left. - \frac{1}{2} \left(\sum_{w \in \mathcal{V}} \zeta_w(\eta) (\alpha^\top \mathbf{K}_{E_w} \alpha)^{\bar{\rho}} \right)^{\frac{1}{\bar{\rho}}} \right)$$

where $\zeta_w(\eta) = \left(\sum_{v \in A(w)} d_v^\rho \eta_v^{1-\rho} \right)^{\frac{1}{1-\rho}}$, $\bar{\rho} = \frac{\rho}{2(\rho-1)}$.

- Mirror Descent Algorithm to solve for α and η iteratively
- Cutting Plane Algorithm Inside Mirror Descent to handle exponential constraints in solving α .

Cutting Plane Algorithm for solving Reduced Optimization Problem

- to solve the reduced optimization problem over α

$$\begin{aligned} \max_{\alpha \in S(\mathcal{Y}, C)} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa_{TYjY'} \\ - \frac{1}{2} \left(\sum_{w \in \mathcal{Y}} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa'_{EwiYjY'} \right)^{\bar{p}} \right)^{\frac{1}{\bar{p}}} \end{aligned}$$

under $\alpha_{i,Y} \geq 0$ and $m \sum_{Y \neq Y_i} \frac{\alpha_{iY}}{\Delta(Y, Y_i)} \leq C$

Cutting Plane Algorithm

- **Input:** kernels, C , ϵ_{margin} (allowed violation of margin)

$S_i \leftarrow \phi \quad \forall i = 1, \dots, m$

- **repeat**

- **for** $i = 1, \dots, m$ **do**

- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_T^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$

- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.

- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.

- **if** $H(\hat{Y}) > \xi_i + \epsilon_{margin}$, **then**

- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.

- compute α using $S = \bigcup_i S_i$

- **end if**

- **end for**

- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ϵ_{margin} (allowed violation of margin)

$$S_i \leftarrow \phi \quad \forall i = 1, \dots, m$$

- **repeat**

- **for** $i = 1, \dots, m$ **do**

- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_T^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$

- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.

- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.

- **if** $H(\hat{Y}) > \xi_i + \epsilon_{margin}$, **then**

- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.

- compute α using $S = \bigcup_i S_i$

- **end if**

- **end for**

- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_T^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max_{Y \in S_i} \{0, \max H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max\{0, \max_{Y \in S_i} H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ε_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{argmax} H(Y)$.
- compute $\xi_i = \max_{Y \in S_i} \{0, \max H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \varepsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ϵ_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{argmax} H(Y)$.
- compute $\xi_i = \max_{Y \in S_i} \{0, H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \epsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Cutting Plane Algorithm

- **Input:** kernels, C , ϵ_{margin} (allowed violation of margin)
 $S_i \leftarrow \phi \quad \forall i = 1, \dots, m$
- **repeat**
- **for** $i = 1, \dots, m$ **do**
- $\forall Y: H(Y)$ is computed using

$$H(Y) \equiv \left[1 - \langle \mathbf{f}_E, \psi_{E_i}^\delta(Y) \rangle - \langle \mathbf{f}_T, \psi_{T_i}^\delta(Y) \rangle \right] \Delta(Y_i, Y)$$
- compute $\hat{Y} = \underset{Y}{\operatorname{argmax}} H(Y)$.
- compute $\xi_i = \max_{Y \in S_i} \{0, \max H(Y)\}$.
- **if** $H(\hat{Y}) > \xi_i + \epsilon_{margin}$, **then**
- $S_i \leftarrow S_i \cup \{\hat{Y}\}$.
- compute α using $S = \bigcup_i S_i$
- **end if**
- **end for**
- **until** no S_i has changed during the iteration.

Solving for α in reduced optimization Problem

- Projected Gradient Descent method \Rightarrow sub-optimal result

- Another method by [5] using lp-MKL approach is

$$\min_{\theta} \max_{\alpha \in \mathcal{S}(\mathcal{Y}, \mathcal{C})} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa_{TiYjY'} \right. \\ \left. + \left(\sum_{w \in \mathcal{V}} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \theta \kappa'_{EwiYjY'} \right) \right) \right) \\ \alpha_{i, Y} \geq 0, m \sum_{Y \neq Y_i} \frac{\alpha_{iY}}{\Delta(Y, Y_i)} \leq C \|\theta\|_{\hat{\rho}}^{\hat{\rho}} \leq 1 (\hat{\rho} = \frac{\rho}{2-\rho} \text{ \& } \theta \geq 0)$$

Solving for α in reduced optimization Problem

- Projected Gradient Descent method \Rightarrow sub-optimal result

- Another method by [5] using lp-MKL approach is

$$\min_{\theta} \max_{\alpha \in \mathcal{S}(\mathcal{Y}, \mathcal{C})} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa_{TiYjY'} \right. \\ \left. + \left(\sum_{w \in \mathcal{V}} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \theta \kappa'_{EwiYjY'} \right) \right) \right) \\ \alpha_{i, Y} \geq 0, m \sum_{Y \neq Y_i} \frac{\alpha_{iY}}{\Delta(Y, Y_i)} \leq C \|\theta\|_{\hat{\rho}}^{\hat{\rho}} \leq 1 (\hat{\rho} = \frac{\rho}{2-\rho} \text{ \& } \theta \geq 0)$$

Solving for α in reduced optimization Problem

- Projected Gradient Descent method \Rightarrow sub-optimal result
- Another method by [5] using lp-MKL approach is

$$\min_{\theta} \max_{\alpha \in \mathcal{S}(\mathcal{Y}, C)} \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa_{TiYjY'} \right. \\ \left. + \left(\sum_{w \in \mathcal{V}} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \theta \kappa'_{EwiYjY'} \right) \right) \right) \\ \alpha_{i, Y} \geq 0, m \sum_{Y \neq Y_i} \frac{\alpha_{iY}}{\Delta(Y, Y_i)} \leq C \|\theta\|_{\hat{\rho}}^{\hat{\rho}} \leq 1 \left(\hat{\rho} = \frac{\rho}{2-\rho} \text{ \& } \theta \geq 0 \right)$$

Solving for α in reduced optimization Problem

- solve quadratic approximation of $\|\cdot\|_\rho$ over θ
- apply cutting plane ?

- Objective becomes $\min_{\eta, \theta} \eta$ s.t.

$$\eta \geq \sum_{i,Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \left(\sum_{i,Y} \sum_{j,Y'} \alpha_{iY} \alpha_{jY'} \kappa_{YY'} + \sum_{w \in \mathcal{Y}} \left(\sum_{i,Y} \sum_{j,Y'} \alpha_{iY} \alpha_{jY'} \theta \kappa'_{Ew|YY'} \right) \right)$$

Solving for α in reduced optimization Problem

- solve quadratic approximation of $\| \cdot \|_\rho$ over θ
- apply cutting plane ?

- Objective becomes $\min_{\eta, \theta} \eta$ s.t.

$$\eta \geq \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa_{TYjY'} + \left(\sum_{w \in \mathcal{Y}} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \theta \kappa'_{EwiYjY'} \right) \right) \right)$$

Solving for α in reduced optimization Problem

- solve quadratic approximation of $\| \cdot \|_\rho$ over θ
- apply cutting plane ?
 - Objective becomes $\min_{\eta, \theta} \eta$ s.t.

$$\eta \geq \sum_{i, Y \neq Y_i} \alpha_{iY} - \frac{1}{2} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \kappa_{TiYjY'} + \left(\sum_{w \in \mathcal{Y}} \left(\sum_{i, Y} \sum_{j, Y'} \alpha_{iY} \alpha_{jY'} \theta \kappa'_{EwiYjY'} \right) \right) \right)$$

Future Works

Applications of Hierarchical Kernel learning For Propositional Features

Applications of Hierarchical Kernel learning For Propositional Features

Learning rule ensembles

- Conjunctive propositional features [1]
- Disjunctive propositional features

Applications of Hierarchical Kernel learning For Propositional Features

Learning rule ensembles

- Conjunctive propositional features [1]
- Disjunctive propositional features

Disjunctive propositional features

- Since HKL follows a top-down approach \rightarrow descendant norm is more suitable
- Top node in lattice is the most general, i.e. disjunction of all

basic features $\bigvee_{n=1}^N \phi_n$

- descendant of node is a more specialized node; got by removing one of the features of its parent.
- Only sufficiency condition changes; everything else remains same.

Disjunctive propositional features

- Since HKL follows a top-down approach \rightarrow descendant norm is more suitable
- Top node in lattice is the most general, i.e. disjunction of all

basic features $\bigvee_{n=1}^N \phi_n$

- descendant of node is a more specialized node; got by removing one of the features of its parent.
- Only sufficiency condition changes; everything else remains same.

Disjunctive propositional features

- Since HKL follows a top-down approach \rightarrow descendant norm is more suitable
- Top node in lattice is the most general, i.e. disjunction of all basic features $\bigvee_{n=1}^N \phi_n$
- descendant of node is a more specialized node; got by removing one of the features of its parent.
- Only sufficiency condition changes; everything else remains same.

Disjunctive propositional features

- Since HKL follows a top-down approach \rightarrow descendant norm is more suitable
- Top node in lattice is the most general, i.e. disjunction of all

basic features $\bigvee_{n=1}^N \phi_n$

- descendant of node is a more specialized node; got by removing one of the features of its parent.
- Only sufficiency condition changes; everything else remains same.

Disjunctive propositional features

- feature map $\phi(x)$ as $(1 - \overline{\phi}(x))$ ($\overline{\phi}(x)$ is boolean complement of $\phi(x)$).

- a disjunctive feature corresponding to

$$\bigvee_{n=1}^N \phi_n(x_i) = (1 - \prod_{n=1}^N \overline{\phi}_n(x_i))$$

- kernel corresponding to the disjunctive feature is $(1 - \prod_{n=1}^N \overline{\phi}_n(x_i))(1 - \prod_{n=1}^N \overline{\phi}_n(x_j))$

Disjunctive propositional features

- feature map $\phi(x)$ as $(1 - \overline{\phi}(x))$ ($\overline{\phi}(x)$ is boolean complement of $\phi(x)$).

- a disjunctive feature corresponding to

$$\bigvee_{n=1}^N \phi_n(x_i) = (1 - \prod_{n=1}^N \overline{\phi}_n(x_i))$$

- kernel corresponding to the disjunctive feature is $(1 - \prod_{n=1}^N \overline{\phi}_n(x_i))(1 - \prod_{n=1}^N \overline{\phi}_n(x_j))$

Disjunctive propositional features

- feature map $\phi(x)$ as $(1 - \overline{\phi}(x))$ ($\overline{\phi}(x)$ is boolean complement of $\phi(x)$).

- a disjunctive feature corresponding to

$$\bigvee_{n=1}^N \phi_n(x_i) = (1 - \prod_{n=1}^N \overline{\phi}_n(x_i))$$

- kernel corresponding to the disjunctive feature is $(1 - \prod_{n=1}^N \overline{\phi}_n(x_i))(1 - \prod_{n=1}^N \overline{\phi}_n(x_j))$

Applications of Hierarchical Kernel learning For Disjunctive Features

- sum of exponential kernels of the entire lattice:

$$\sum_{v \in V} K_v(x_i, x_j) =$$
$$1 + 2^N + \prod_{n=1}^N (1 + \bar{\phi}_n(x_i) \bar{\phi}_n(x_j)) - \prod_n (1 + \bar{\phi}_n(x_i)) - \prod_n (1 + \bar{\phi}_n(x_j))$$

Applications of Hierarchical Kernel learning For Disjunctive Features

- sufficiency condition: $\max_{t \in \text{sources}(W^C)} \sum_{i,j} \alpha_{Wi} Q(t)_{ij} \alpha_{Wj} \leq \Omega_s(f)^2 + \varepsilon$

where

$$Q(t)_{ij} = \frac{1}{(1+b)^{2|t|}} \left(\left(1 + \left(\frac{1+b}{b}\right)^2\right)^{|t|} - \prod_{k \in t} \left(1 + \frac{\bar{\phi}_k(x_i)}{\left(\frac{b}{b+1}\right)^2}\right) - \prod_{k \in t} \left(1 + \frac{\bar{\phi}_k(x_j)}{\left(\frac{b}{b+1}\right)^2}\right) \right. \\ \left. + \prod_{k \in t} \left(1 + \frac{\bar{\phi}_k(x_i)}{\frac{b}{1+b}} \frac{\bar{\phi}_k(x_j)}{\frac{b}{1+b}}\right) \right)$$

Applications of Hierarchical Kernel learning For Learning Taxonomies

- **inherent hierarchical structure exploited**
- vocabulary consisting of important sense tagged words
- every sense of every word becomes a basic feature of HKL
- Syntagmatic Information (context-co-occurrence): conjunctive lattice
- Paradigmatic Information (synonymous words): disjunctive lattice

Applications of Hierarchical Kernel learning For Learning Taxonomies

- inherent hierarchical structure exploited
- vocabulary consisting of important sense tagged words
- every sense of every word becomes a basic feature of HKL
- Syntagmatic Information (context-co-occurrence): conjunctive lattice
- Paradigmatic Information (synonymous words): disjunctive lattice

Applications of Hierarchical Kernel learning For Learning Taxonomies

- inherent hierarchical structure exploited
- vocabulary consisting of important sense tagged words
- every sense of every word becomes a basic feature of HKL
- Syntagmatic Information (context-co-occurrence): conjunctive lattice
- Paradigmatic Information (synonymous words): disjunctive lattice

Applications of Hierarchical Kernel learning For Learning Taxonomies

- inherent hierarchical structure exploited
- vocabulary consisting of important sense tagged words
- every sense of every word becomes a basic feature of HKL
- Syntagmatic Information (context-co-occurrence): conjunctive lattice
- Paradigmatic Information (synonymous words): disjunctive lattice

Applications of Hierarchical Kernel learning For Learning Taxonomies

- inherent hierarchical structure exploited
- vocabulary consisting of important sense tagged words
- every sense of every word becomes a basic feature of HKL
- Syntagmatic Information (context-co-occurrence): conjunctive lattice
- Paradigmatic Information (synonymous words): disjunctive lattice

HKL for learning First Order Logic Features

HKL for learning First Order Logic Features

- Simple Clauses (subparts of clauses) [2]

For $teen_aged_boy(A) \leftarrow male(A), age(A, B), B \geq 12, B \leq 20$
the simple clauses are

$teen_aged_boy(A) \leftarrow male(A)$

$teen_aged_boy(A) \leftarrow age(A, B), B \geq 12$

$teen_aged_boy(A) \leftarrow age(A, B), B \leq 20$

- independant of the variables used
- determinate clauses

HKL for learning First Order Logic Features

- Simple Clauses (subparts of clauses) [2]

For $teen_aged_boy(A) \leftarrow male(A), age(A, B), B \geq 12, B \leq 20$
the simple clauses are

$teen_aged_boy(A) \leftarrow male(A)$

$teen_aged_boy(A) \leftarrow age(A, B), B \geq 12$

$teen_aged_boy(A) \leftarrow age(A, B), B \leq 20$

- independant of the variables used
- determinate clauses

HKL for learning First Order Logic Features

- Simple Clauses (subparts of clauses) [2]

For $teen_aged_boy(A) \leftarrow male(A), age(A, B), B \geq 12, B \leq 20$
the simple clauses are

$teen_aged_boy(A) \leftarrow male(A)$

$teen_aged_boy(A) \leftarrow age(A, B), B \geq 12$

$teen_aged_boy(A) \leftarrow age(A, B), B \leq 20$

- independant of the variables used
- determinate clauses

HKL for learning First Order Logic Features

- Simple Clauses (subparts of clauses) [2]

For $teen_aged_boy(A) \leftarrow male(A), age(A, B), B \geq 12, B \leq 20$
the simple clauses are

$teen_aged_boy(A) \leftarrow male(A)$

$teen_aged_boy(A) \leftarrow age(A, B), B \geq 12$

$teen_aged_boy(A) \leftarrow age(A, B), B \leq 20$

- independant of the variables used
- determinate clauses

Simple Clauses

- construct 'meaningful variable bindings' inside the simple clause
- conjunctions of simple clauses with 'meaningful variable mapping' constructs a clause
- only a few of simple clauses will be useful
- maintain variable mapping while combining simple clauses using HKL

Simple Clauses

- construct 'meaningful variable bindings' inside the simple clause
- conjunctions of simple clauses with 'meaningful variable mapping' constructs a clause
- only a few of simple clauses will be useful
- maintain variable mapping while combining simple clauses using HKL

Simple Clauses

- construct 'meaningful variable bindings' inside the simple clause
- conjunctions of simple clauses with 'meaningful variable mapping' constructs a clause
- only a few of simple clauses will be useful
- maintain variable mapping while combining simple clauses using HKL

Simple Clauses

- construct 'meaningful variable bindings' inside the simple clause
- conjunctions of simple clauses with 'meaningful variable mapping' constructs a clause
- only a few of simple clauses will be useful
- maintain variable mapping while combining simple clauses using HKL

Simple Clauses

- construct 'meaningful variable bindings' inside the simple clause
- conjunctions of simple clauses with 'meaningful variable mapping' constructs a clause
- only a few of simple clauses will be useful
- maintain variable mapping while combining simple clauses using HKL

Construction of Simple Clauses

- variable dependency graph
(acyclic for determinate clauses)

`teen_age_boy(A) ← male(A) , age(A,B) , B>12 , 20>B.`



Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Construction of Simple Clauses

- For each sink node, path from head to that node-potential simple clause (simple chain)
- Nodes in subgraph are literals
- Edges can have exponential configurations (variable bindings)
- For each combination of edge-configurations, total utility of simple clause = net potential
- utility as precision & recall; simple clauses should have high enough recall
- Selecting the good combination of edge configurations to maximise potential of chain graph
 - Randomized Algorithm, e.g. Simulated Annealing

Problems with practical applicability

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality(with high probability)

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality (with high probability)

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality (with high probability)

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality (with high probability)

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality(with high probability)

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality(with high probability)

Problems with practical applicability

- Large scale required
- Systematic Approach of HKL; starting from most general feature - slow
- maintaining hull structure for sufficiency condition
- Divide and Conquer Approach: clustered HKL
- Randomized / Approximation Algorithm
 - scales linearly with dimension of Feature Space
 - attain a parameterized bound on global optimality(with high probability)

Clustered HKL

Divide and Conquer Approach: clustered HKL

- in some datasets, groups of features occurring together; cluster them
- put the remaining features together
- for each cluster run HKL to get Active Set for each cluster
- clustering not rigid
- Combine the active set lattices levelwise
- Can reduce the overall complexity and response time?

Divide and Conquer Approach: clustered HKL

- in some datasets, groups of features occurring together; cluster them
- put the remaining features together
- for each cluster run HKL to get Active Set for each cluster
- clustering not rigid
- Combine the active set lattices levelwise
- Can reduce the overall complexity and response time?

Divide and Conquer Approach: clustered HKL

- in some datasets, groups of features occurring together; cluster them
- put the remaining features together
- for each cluster run HKL to get Active Set for each cluster
- clustering not rigid
- Combine the active set lattices levelwise
- Can reduce the overall complexity and response time?

Divide and Conquer Approach: clustered HKL

- in some datasets, groups of features occurring together; cluster them
- put the remaining features together
- for each cluster run HKL to get Active Set for each cluster
- clustering not rigid
- Combine the active set lattices levelwise
- Can reduce the overall complexity and response time?

Divide and Conquer Approach: clustered HKL

- in some datasets, groups of features occurring together; cluster them
- put the remaining features together
- for each cluster run HKL to get Active Set for each cluster
- clustering not rigid
- Combine the active set lattices levelwise
- Can reduce the overall complexity and response time?

Divide and Conquer Approach: clustered HKL

- in some datasets, groups of features occurring together; cluster them
- put the remaining features together
- for each cluster run HKL to get Active Set for each cluster
- clustering not rigid
- Combine the active set lattices levelwise
- Can reduce the overall complexity and response time?

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
 - For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of $Consolidated_Active_Set$ iff $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of $Consolidated_Active_Set$ iff $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists \text{ a basic feature } f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

clustered HKL Algorithm

- **Input:** Clusters of features F_1, F_2, \dots, F_n
- For each $i = 1, \dots, n$
 - Construct active set
 - $S(i)$ using HKL/Randomized HKL
 - $Consolidated_S = S(1)$
- For each $i = 1, \dots, n$
 - level $l = 0$
 - do {
 - combine a node from level l of $Consolidated_S$ and
 - $S(i)$ to get a new node v
 - if $\|v\|$ is j , add v to the correct level $\leq l+j$ of
 - $Consolidated_Active_Set$ iff
 - $v \notin \{S\} \Rightarrow \forall F_k; k = 1..n, \exists$ a basic feature $f \in v; f \notin F_k$
 - increment l
 - } till no more levels left in $S(i)$ and $Consolidated_S$

Randomized HKL

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm: Estimating size of Active Set

- Violators can only cause objective value to change- 'free variables'
- Number of violators = Combinatorial Dimension of the Optimization Problem in HKL
- Randomized algorithm for estimating combinatorial dimension
- Can help in deciding resource bound required for the feature induction problem
- Can be seen as a 0-normed constraint on the weights
 $\|f\|_0 = K$
- 0-norm is NP hard; use some approximation
- Can this constraint be used to find the good 'C' for the SVM

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

Pure Adaptive Search(PAS) [6] for optimizing a Lipschitz continuous function over convex set

- initialises from a random starting point
 - In HKL: starting point can be clusters of features occurring together
 - finding simple dependencies (domain-based) from dataset
- Next, PAS randomly samples points from a 'direction' that monotonically decreases the objective
 - In HKL, every addition of violator node reduces the objective and a non-violator node keeps objective unchanged
- No of iterations required by **PAS:PRS**(Pure random search) is exponential
- Complexity of PAS is linear in number of dimensions of the search space

Randomized Algorithm for HKL

- Can we do better by using some heuristic or distribution to sample points?
 - i.e. nodes sampled should be stronger violator than any random node
 - then PAS complexity would be an upper bound

Randomized Algorithm for HKL

- Can we do better by using some heuristic or distribution to sample points?
 - i.e. nodes sampled should be stronger violator than any random node
 - then PAS complexity would be an upper bound

Randomized Algorithm for HKL

- Can we do better by using some heuristic or distribution to sample points?
 - i.e. nodes sampled should be stronger violator than any random node
 - then PAS complexity would be an upper bound

A* type approach for Randomized Algorithm

- Determining 'step-size' and 'direction' of movement
- A* type approach?
- every iteration of active set algorithm is a state in A*
- Knowledge of goal node \Rightarrow cost from current state to goal state
- degree of violation is monotonic
$$\text{cost}(x) \leq \text{cost}(y) + \text{distance}(x, y)$$
- can monotonicity help reach global optima?

A* type approach for Randomized Algorithm

- Determining 'step-size' and 'direction' of movement
- A* type approach?
- every iteration of active set algorithm is a state in A*
- Knowledge of goal node \Rightarrow cost from current state to goal state
- degree of violation is monotonic
 $cost(x) \leq cost(y) + distance(x, y)$
- can monotonicity help reach global optima?

A* type approach for Randomized Algorithm

- Determining 'step-size' and 'direction' of movement
- A* type approach?
- every iteration of active set algorithm is a state in A*
- Knowledge of goal node \Rightarrow cost from current state to goal state
- degree of violation is monotonic
 $cost(x) \leq cost(y) + distance(x, y)$
- can monotonicity help reach global optima?

A* type approach for Randomized Algorithm

- Determining 'step-size' and 'direction' of movement
- A* type approach?
- every iteration of active set algorithm is a state in A*
- Knowledge of goal node \Rightarrow cost from current state to goal state
- degree of violation is monotonic
 $cost(x) \leq cost(y) + distance(x, y)$
- can monotonicity help reach global optima?

A* type approach for Randomized Algorithm

- Determining 'step-size' and 'direction' of movement
- A* type approach?
- every iteration of active set algorithm is a state in A*
- Knowledge of goal node \Rightarrow cost from current state to goal state
- degree of violation is monotonic
$$cost(x) \leq cost(y) + distance(x, y)$$
- can monotonicity help reach global optima?

A* type approach for Randomized Algorithm

- Determining 'step-size' and 'direction' of movement
- A* type approach?
- every iteration of active set algorithm is a state in A*
- Knowledge of goal node \Rightarrow cost from current state to goal state
- degree of violation is monotonic
$$cost(x) \leq cost(y) + distance(x, y)$$
- can monotonicity help reach global optima?

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$

- Hull structure ??

If descendant is visited, then include all its descendants into active set

- Sufficiency Condition ??

Let \mathcal{A} be the active set, \mathcal{A}^* be the optimal set, $\mathcal{A} \cap \mathcal{A}^* = \emptyset$ then sufficiency condition is satisfied

Depth 1, 2, 3 and 4, 13 of sufficiency condition is satisfied, but 1, 2, 3, 4, 13 are not optimal

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$

- Hull structure ??

How many nodes in the hull structure are considered for the next step?

- Sufficiency Condition ??

How many nodes in the hull structure are considered for the next step? (e.g., 10% of the hull structure)

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$
 - Hull structure ??
 - if descendant is violator, then include all its ancestors into active set
 - Sufficiency Condition ??
 - $$\sum_{u \in \text{sources}(W^c)} \alpha_W^T \sum_{w \in D(u)} \frac{K_w}{(\sum_{v \in A(w) \cup D(u)} d_v)^2} \alpha_W \leq \Omega(f)^2 + 2\epsilon$$
 - Both LHS and RHS of sufficiency condition knows nothing about the intermediate nodes that might be added later

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$
 - Hull structure ??
 - if descendant is violator, then include all its ancestors into active set
 - Sufficiency Condition ??
 - $$\sum_{u \in \text{sources}(W^c)} \alpha_W^T \sum_{w \in D(u)} \frac{K_w}{(\sum_{v \in A(w) \cup D(u)} d_v)^2} \alpha_W \leq \Omega(f)^2 + 2\epsilon$$
 - Both LHS and RHS of sufficiency condition knows nothing about the intermediate nodes that might be added later

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$
 - Hull structure ??
 - if descendant is violator, then include all its ancestors into active set
 - Sufficiency Condition ??
 - $$\sum_{u \in \text{sources}(W^c)} \alpha_u^T \sum_{w \in D(u)} \frac{K_w}{(\sum_{v \in A(w) \cup D(u)} d_v)^2} \alpha_w \leq \Omega(f)^2 + 2\epsilon$$
 - Both LHS and RHS of sufficiency condition knows nothing about the intermediate nodes that might be added later

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$
 - Hull structure ??
 - if descendant is violator, then include all its ancestors into active set
 - Sufficiency Condition ??
 - $$\sum_{u \in \text{sources}(W^c)} \alpha_W^T \sum_{w \in D(u)} \frac{K_w}{(\sum_{v \in A(w) \cup D(u)} d_v)^2} \alpha_W \leq \Omega(f)^2 + 2\varepsilon$$
 - Both LHS and RHS of sufficiency condition knows nothing about the intermediate nodes that might be added later

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$
 - Hull structure ??
 - if descendant is violator, then include all its ancestors into active set
 - Sufficiency Condition ??
 - $$\sum_{u \in \text{sources}(W^c)} \alpha_W^T \sum_{w \in D(u)} \frac{K_w}{(\sum_{v \in A(w) \cup D(u)} d_v)^2} \alpha_W \leq \Omega(f)^2 + 2\epsilon$$
 - Both LHS and RHS of sufficiency condition knows nothing about the intermediate nodes that might be added later

Step Size & Direction of Randomized Algorithm

- heuristic / distribution to choose direction
- step size 's' \Rightarrow descendants of downward-fringe of Active Set at that depth considered
- If $s > 1$
 - Hull structure ??
 - if descendant is violator, then include all its ancestors into active set
 - Sufficiency Condition ??
 - $$\sum_{u \in \text{sources}(W^c)} \alpha_W^T \sum_{w \in D(u)} \frac{K_w}{(\sum_{v \in A(w) \cup D(u)} d_v)^2} \alpha_W \leq \Omega(f)^2 + 2\epsilon$$
 - Both LHS and RHS of sufficiency condition knows nothing about the intermediate nodes that might be added later

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \varepsilon)) \geq 1 - \delta$
- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$
- and estimation of k is
 $E[Candidates(i)] = No_sources_W^C(i)$
 $= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$
 (when hull structure is maintained)

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \varepsilon)) \geq 1 - \delta$
- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$
- and estimation of k is
 $E[Candidates(i)] = No_sources_W^C(i)$
 $= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$
 (when hull structure is maintained)

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \epsilon)) \geq 1 - \delta$
- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$
- and estimation of k is
 $E[Candidates(i)] = No_sources_W^C(i)$
 $= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$
 (when hull structure is maintained)

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \varepsilon)) \geq 1 - \delta$
- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$
- and estimation of k is
 $E[Candidates(i)] = No_sources_W^C(i)$
 $= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$
 (when hull structure is maintained)

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \epsilon)) \geq 1 - \delta$

- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$

- and estimation of k is

$$E[Candidates(i)] = No_sources_W^C(i)$$

$$= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$$

(when hull structure is maintained)

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \epsilon)) \geq 1 - \delta$

- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$

- and estimation of k is

$$E[Candidates(i)] = No_sources_W^C(i)$$

$$= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$$

(when hull structure is maintained)

Sample Size for Randomized Algorithm

- How many sources to sample to get most violators?
- say nodes sampled in direction d is represented by boolean vector R
- say, k candidate nodes
- k' nodes to be sampled i.e. $\|R\|_0 = k'$ or $\|R\|_1 = k'$
- need to estimate k' s.t. (where $t \in \{Candidates(i)\}$),
 $Pr(\|(1_{k \times 1} - R) * [\beta_t]_t\|_\infty \leq (\Omega_s(f)^2 + \epsilon)) \geq 1 - \delta$
- where $\beta_t = \sum_{w \in D(t)} \frac{\alpha^T K_w \alpha}{(\sum_{v \in A(w) \cup D(t)} d_v)^2}$
- and estimation of k is
 $E[Candidates(i)] = No_sources_W^C(i)$
 $= \sum_{v \in violators(i-1)} \sum_{n \in F-v} Prob(\text{all subsets of set } (v, n) \text{ are in } W)$
 (when hull structure is maintained)

Stopping condition for Randomized Algorithm

- Sufficiency Condition relying on the nodes selected
- $Pr[\|\beta_t\|_\rho \leq \Omega_s(f)^2 + \varepsilon; \forall t \in \text{Selected}(i)] \geq 1 - \delta$
- A stricter bound on the sufficiency condition :
 $Pr[\|\beta_t\|_1 \leq \Omega_s(f)^2 + \varepsilon; \forall t \in \text{Selected}(i)] \geq 1 - \delta$
- Number of violators exceed the combinatorial dimension of the optimization problem of HKL

Stopping condition for Randomized Algorithm

- Sufficiency Condition relying on the nodes selected
- $Pr[\|\beta_t\|_\rho \leq \Omega_s(f)^2 + \varepsilon; \forall t \in \text{Selected}(i)] \geq 1 - \delta$
- A stricter bound on the sufficiency condition :
 $Pr[\|\beta_t\|_1 \leq \Omega_s(f)^2 + \varepsilon; \forall t \in \text{Selected}(i)] \geq 1 - \delta$
- Number of violators exceed the combinatorial dimension of the optimization problem of HKL

Stopping condition for Randomized Algorithm

- Sufficiency Condition relying on the nodes selected
- $Pr[\|\beta_t\|_\rho \leq \Omega_s(f)^2 + \varepsilon; \forall t \in \text{Selected}(i)] \geq 1 - \delta$
- A stricter bound on the sufficiency condition :
 $Pr[\|\beta_t\|_1 \leq \Omega_s(f)^2 + \varepsilon; \forall t \in \text{Selected}(i)] \geq 1 - \delta$
- Number of violators exceed the combinatorial dimension of the optimization problem of HKL

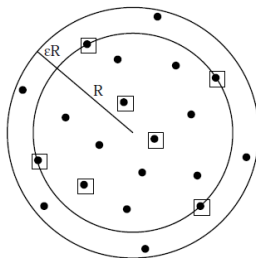
Stopping condition for Randomized Algorithm

- Sufficiency Condition relying on the nodes selected
- $Pr[\|\beta_t\|_\rho \leq \Omega_s(f)^2 + \varepsilon; \forall t \in Selected(i)] \geq 1 - \delta$
- A stricter bound on the sufficiency condition :
 $Pr[\|\beta_t\|_1 \leq \Omega_s(f)^2 + \varepsilon; \forall t \in Selected(i)] \geq 1 - \delta$
- Number of violators exceed the combinatorial dimension of the optimization problem of HKL

Randomized HKL as Minimum Enclosing Ball

Randomized HKL as Minimum Enclosing Ball

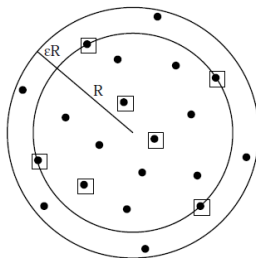
- Core Vector Machine [3] \rightarrow learning problem as finding minimum Ball $B(c, R)$ enclosing all m data points in $D \in \mathbb{R}^d$
- Approximate SVM training based on finding core-set



- Objective \rightarrow find an $(1 + \epsilon)$ approximation of core set circle to enclose all data points
- Center keeps shifting, final value of center gives the learning weights
- Core-set size depends only ϵ and not on m and d

Randomized HKL as Minimum Enclosing Ball

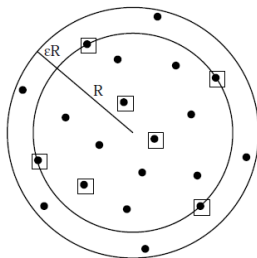
- Core Vector Machine [3] \rightarrow learning problem as finding minimum Ball $B(c, R)$ enclosing all m data points in $D \in \mathbb{R}^d$
- Approximate SVM training based on finding core-set



- Objective \rightarrow find an $(1 + \epsilon)$ approximation of core set circle to enclose all data points
- Center keeps shifting, final value of center gives the learning weights
- Core-set size depends only ϵ and not on m and d

Randomized HKL as Minimum Enclosing Ball

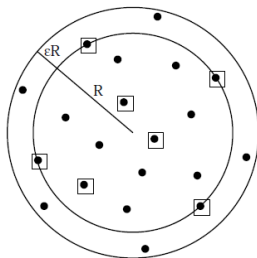
- Core Vector Machine [3] \rightarrow learning problem as finding minimum Ball $B(c, R)$ enclosing all m data points in $D \in \mathbb{R}^d$
- Approximate SVM training based on finding core-set



- Objective \rightarrow find an $(1 + \epsilon)$ approximation of core set circle to enclose all data points
- Center keeps shifting, final value of center gives the learning weights
- Core-set size depends only ϵ and not on m and d

Randomized HKL as Minimum Enclosing Ball

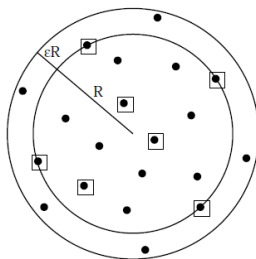
- Core Vector Machine [3] \rightarrow learning problem as finding minimum Ball $B(c, R)$ enclosing all m data points in $D \in \mathbb{R}^d$
- Approximate SVM training based on finding core-set



- Objective \rightarrow find an $(1 + \varepsilon)$ approximation of core set circle to enclose all data points
- Center keeps shifting, final value of center gives the learning weights
- Core-set size depends only ε and not on m and d

Randomized HKL as Minimum Enclosing Ball

- Core Vector Machine [3] \rightarrow learning problem as finding minimum Ball $B(c, R)$ enclosing all m data points in $D \in \mathbb{R}^d$
- Approximate SVM training based on finding core-set



- Objective \rightarrow find an $(1 + \epsilon)$ approximation of core set circle to enclose all data points
- Center keeps shifting, final value of center gives the learning weights
- Core-set size depends only ϵ and not on m and d

Randomized HKL as Minimum Enclosing Ball

- HKL starts with most general node - most violating
- finding top- k violating nodes in every iteration of active-set of HKL
- the maximum violation decreases exponentially in every iteration of active set
- in some cases, the descendant of the most violating node is the most violating node in later iterations

Randomized HKL as Minimum Enclosing Ball

- HKL starts with most general node - most violating
- finding top- k violating nodes in every iteration of active-set of HKL
- the maximum violation decreases exponentially in every iteration of active set
- in some cases, the descendant of the most violating node is the most violating node in later iterations

Randomized HKL as Minimum Enclosing Ball

- HKL starts with most general node - most violating
- finding top- k violating nodes in every iteration of active-set of HKL
- the maximum violation decreases exponentially in every iteration of active set
- in some cases, the descendant of the most violating node is the most violating node in later iterations

Randomized HKL as Minimum Enclosing Ball

- HKL starts with most general node - most violating
- finding top- k violating nodes in every iteration of active-set of HKL
- the maximum violation decreases exponentially in every iteration of active set
- in some cases, the descendant of the most violating node is the most violating node in later iterations

Analysis of violators in HKL

Analysis in Hierarchical Kernel Learning			
Datasets	Iteration1	Iteration2	Iteration3
hepatitis	2664:2347	46136:24265	-
bcancer	2812:2063	37636:23330	-
transfusion	420:149	570:4	-
hebarman	760:347	2347:624	733:8
vote	480:261	1060:15	-
tictactoe	1404:1302	19205:7013	32066:37
tictactoe3	1404:1336	20348:8116	40471:65

Table: No. of Sources : No. of violators in every iteration of Active Set algorithm

Analysis of violators in HKL

Analysis in Hierarchical Kernel Learning			
Datasets	Iteration1	Iteration2	Iteration3
hepatitis	(33,9)(50.097)	(3,9,30)(7.9119)	-
bcancer	(30,58)(65.831)	(20,30,42)(11.22)	-
transfusion	(20,29)(3.2441)	(2,20,29) (0.2155)	-
hebarman	(17,21)(13.205)	(20,27,36)(3.1304)	(20,27,35,36)(0.45)
vote	(6,26)(4.35)	(6,26,29)(0.7341)	-
tictactoe	(29,53)(25.8)	(29,49,53)(5.372)	(29,49,45,53)(0.4337)
tictactoe3	(38,43)(23.45)	(38,43,45)(5.842)	(38,43,45,48)(0.510689)

Table: Maximum Violator (Maximum Violation) in every iteration of Active Set algorithm

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

- Total number of points (\equiv nodes) exponential
- Take center of ball as the point of maximum violation : start of Active Set
- Take inverse of degree of violation of a node as the distance of point from center
- violators-all points lying in $\frac{1}{\epsilon}$ radius ball(All Core-sets??)
- violators in each iteration of active set constitute a shell
- Points outside it dont matter
- in every iteration if top -k most violating nodes are taken \Rightarrow the increment of ball is minimised
- Objective is finding the minimum ball $B(c, R)$ which approximates global optimality conditions

Randomized HKL as Minimum Enclosing Ball

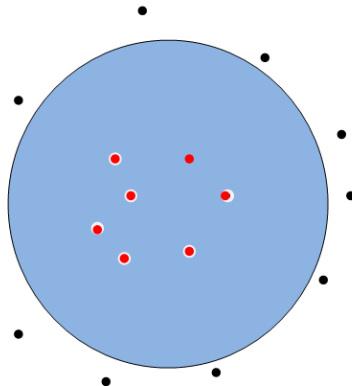


Figure: Active Set Algo Iteration 1

Randomized HKL as Minimum Enclosing Ball

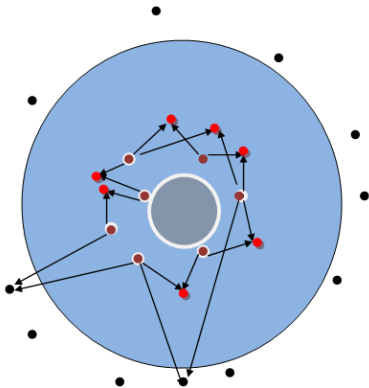


Figure: Active Set Algo Iteration 2

Randomized HKL as Minimum Enclosing Ball

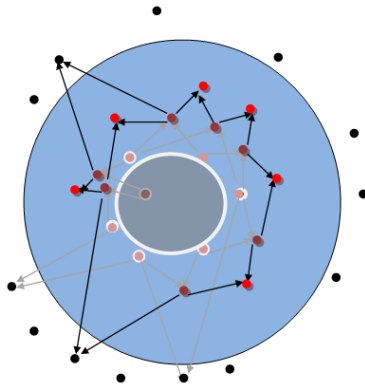


Figure: Active Set Algo Iteration 3

THANK YOU



Saketh Nath J, Pratik J, Ganesh Ramakrishnan. *Efficient Rule Ensemble Learning using Hierarchical Kernels*, ICML 2011



Eric McCreath, Arun Sharma. *LIME: A System for Learning Relations*,



I.W. Tsang, J.T. Kwok, P.M. Cheung. *Core Vector Machines: Fast SVM Training on Very Large Data Sets*, Journal of Machine Learning 2005



Y. Altun, T. Hoffman, I. Tsochantaridis. *SVM Learning for Interdependent and Structured Output Spaces*, Journal of Machine Learning 2006



M. Kloft, S. Sonnenberg, *Efficient and accurate l_p -Norm Multiple Kernel Learning*



Zelda Zabinsky, *Pure Adaptive Search in Global Optimization*, Technical Report, 1989