Structured Sparsity in Machine Learning:
Models, Algorithms, and Applications

André F. T. Martins

Joint work with:

Mário A. T. Figueiredo, Instituto de Telecomunicações, Lisboa, Portugal
Noah A. Smith, Language Technologies Institute, Carnegie Mellon University, USA

BCS/AG DANK 2013, November 2013
Outline

1. Sparsity and Feature Selection

2. Structured Sparsity

3. Algorithms
   - Batch Algorithms
   - Online Algorithms

4. Applications

5. Conclusions
Our Setup

- Input set $\mathcal{X}$, output set $\mathcal{Y}$
- Linear model:

$$\hat{y} := \arg \max_{y \in \mathcal{Y}} w^\top f(x, y)$$

where $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ is a feature map

- Learning the model parameters from data $\{(x_n, y_n)\}_{n=1}^N \subseteq \mathcal{X} \times \mathcal{Y}$:

$$\hat{w} = \arg \min_w \frac{1}{N} \sum_{n=1}^N L(w; x_n, y_n) + \Omega(w)$$

where $\Omega(w)$ is the regularizer

This talk: we focus on the regularizer $\Omega(w)$

AndrÉ F. T. Martins (Priberam/IT)
Our Setup

- Input set $\mathcal{X}$, output set $\mathcal{Y}$
- Linear model:
  \[ \hat{y} := \arg \max_{y \in \mathcal{Y}} w^\top f(x, y) \]
  where $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ is a feature map
- Learning the model parameters from data $\{(x_n, y_n)\}_{n=1}^N \subseteq \mathcal{X} \times \mathcal{Y}$:
  \[ \hat{w} = \arg \min_w \frac{1}{N} \sum_{n=1}^N L(w; x_n, y_n) + \Omega(w) \]

This talk: we focus on the regularizer $\Omega$
The Bet On Sparsity (Friedman et al., 2004)

**Sparsity hypothesis:** not all dimensions of $f$ are needed (many features are irrelevant)

Setting the corresponding weights to zero leads to a sparse $w$

Models with just a few features:

- are easier to explain/interpret
- have a smaller memory footprint
- are faster to run (less features need to be evaluated)
- generalize better
(Automatic) Feature Selection

Domain experts are often good at engineering features. Can we automate the process of selecting which ones to keep?

Three main classes of methods (Guyon and Elisseeff, 2003):

1. filters
2. wrappers
3. embedded methods
Domain experts are often good at engineering features. Can we automate the process of selecting which ones to keep?

Three main classes of methods (Guyon and Elisseeff, 2003):

1. filters (inexpensive and simple, but very suboptimal)
2. wrappers
3. embedded methods
(Automatic) Feature Selection

Domain experts are often good at engineering features. Can we automate the process of selecting which ones to keep?

Three main classes of methods (Guyon and Elisseeff, 2003):

1. filters (inexpensive and simple, but very suboptimal)
2. wrappers (better, but very expensive)
3. embedded methods
(Automatic) Feature Selection

Domain experts are often good at engineering features. Can we automate the process of selecting which ones to keep?

Three main classes of methods (Guyon and Elisseeff, 2003):

1. filters (inexpensive and simple, but very suboptimal)
2. wrappers (better, but very expensive)
3. embedded methods (this talk)
Formulate the learning problem as a trade-off between

- minimizing loss (fitting the training data, achieving good accuracy on the training data, etc.)
- choosing a desirable model (e.g., with no more features than needed)

\[
\min_w \frac{1}{N} \sum_{n=1}^{N} L(w; x_n, y_n) + \Omega(w)
\]
Embedded Methods for Feature Selection

Formulate the learning problem as a trade-off between

- minimizing loss (fitting the training data, achieving good accuracy on the training data, etc.)
- choosing a desirable model (e.g., with no more features than needed)

\[
\min_w \frac{1}{N} \sum_{n=1}^{N} L(w; x_n, y_n) + \Omega(w)
\]

Design \( \Omega \) to select relevant features (\textit{sparsity-inducing regularization})
Embedded Methods for Feature Selection

Formulate the learning problem as a trade-off between

- minimizing loss (fitting the training data, achieving good accuracy on the training data, etc.)
- choosing a desirable model (e.g., with no more features than needed)

\[
\min_w \frac{1}{N} \sum_{n=1}^{N} L(w; x_n, y_n) + \Omega(w)
\]

Design \( \Omega \) to select relevant features (sparsity-inducing regularization)

Key advantage: declarative statements of model “desirability” often lead to well-understood, convex optimization problems.
Convex Loss Functions

**Squared** (linear regression) \[ \frac{1}{2} \left( y - w^\top f(x) \right)^2 \]

**Log-linear** (MaxEnt, CRF, logistic) \[ -w^\top f(x, y) + \log \sum_{y' \in Y} \exp(w^\top f(x, y')) \]

**Hinge** (SVMs) \[ -w^\top f(x, y) + \max_{y' \in Y} (w^\top f(x, y') + c(y, y')) \]

**Perceptron** \[ -w^\top f(x, y) + \max_{y' \in Y} w^\top f(x, y') \]
Regularization Formulations

- Tikhonov regularization: 
  \[ \hat{w} = \arg \min_w \lambda \Omega(w) + \sum_{n=1}^N L(w; x_n, y_n) \]

- Ivanov regularization
  \[ \hat{w} = \arg \min_w \sum_{n=1}^N L(w; x_n, y_n) \]
  subject to \( \Omega(w) \leq \tau \)

- Morozov regularization
  \[ \hat{w} = \arg \min_w \Omega(w) \]
  subject to \( \sum_{n=1}^N L(w; x_n, y_n) \leq \delta \)
Regularization Formulations

- Tikhonov regularization: 
  \[ \hat{w} = \arg \min_w \lambda \Omega(w) + \sum_{n=1}^{N} L(w; x_n, y_n) \]

- Ivanov regularization
  \[ \hat{w} = \arg \min_w \sum_{n=1}^{N} L(w; x_n, y_n) \]
  subject to \( \Omega(w) \leq \tau \)

- Morozov regularization
  \[ \hat{w} = \arg \min_w \Omega(w) \]
  subject to \( \sum_{n=1}^{N} L(w; x_n, y_n) \leq \delta \)

Equivalent, under mild conditions (namely convexity).
Norms: a Quick Review

- Any norm is a convex function (follows from triangle inequality)
Norms: a Quick Review

- Any norm is a convex function (follows from triangle inequality)

- $\ell_p$-norms ($p \geq 1$): $\|w\|_p = (\sum_i |w_i|^p)^{1/p}$

\[\begin{align*}
\|w\|_1 &= \sum_i |w_i|, \\
\|w\|_2 &= \sum_i w_i^2, \\
\|w\|_\infty &= \max_i |w_i|
\end{align*}\]
Norms: a Quick Review

- Any norm is a convex function (follows from triangle inequality)

- $\ell_p$-norms ($p \geq 1$): $\|w\|_p = (\sum_i |w_i|^p)^{1/p}$

- Side note: the infamous $\ell_0$ “norm” (non-convex, not a norm):

$$\|w\|_0 = \lim_{p \to 0} \|w\|_p = |\{i : w_i \neq 0\}|$$
Ridge and Lasso Regularizers

**Ridge** or $\ell_2$ regularization: \[ \Omega(w) = \frac{\lambda}{2} \|w\|_2^2 \]

- goes back to Tikhonov (1943) and Wiener (1949)
- corresponds to a zero-mean Gaussian prior
- **Pros**: smooth and convex, thus benign for optimization.
- **Cons**: doesn’t promote sparsity (no explicit feature selection)
Ridge and Lasso Regularizers

**Ridge** or $\ell_2$ regularization: \[
\Omega(w) = \frac{1}{2} \|w\|^2_2
\]

- Goes back to Tikhonov (1943) and Wiener (1949)
- Corresponds to a zero-mean Gaussian prior
- **Pros**: smooth and convex, thus benign for optimization.
- **Cons**: doesn’t promote sparsity (no explicit feature selection)

**Lasso** or $\ell_1$ regularization: \[
\Omega(w) = \lambda \|w\|_1
\]

- Goes back to Claerbout and Muir (1973); Taylor et al. (1979); Tibshirani (1996)
- Corresponds to zero-mean Laplacian prior
- **Pros**: encourages sparsity: embedded feature selection.
- **Cons**: convex, but non-smooth: more challenging optimization.
The Lasso and Sparsity

Why does the Lasso yield sparsity?
The Lasso and Sparsity

Why does the Lasso yield sparsity?

\[
\hat{w} = \text{arg min}_w \|Aw - y\|^2_2 \\
\text{subject to } \|w\|_1 \leq \tau
\]

\[
\hat{w} = \text{arg min}_w \|Aw - y\|^2_2 \\
\text{subject to } \|w\|^2_2 \leq \tau
\]
Take-Home Messages

- Sparsity is desirable for interpretability, computational savings, and generalization
- $\ell_1$-regularization gives an embedded method for feature selection
- Another view of $\ell_1$: a convex surrogate for direct penalization of cardinality ($\ell_0$)
- Under some conditions, $\ell_1$ guarantees exact support recovery (Candès et al., 2006; Donoho, 2006)
- However: the currently known sufficient conditions are too strong and not met in typical ML problems
Outline

1. Sparsity and Feature Selection

2. Structured Sparsity

3. Algorithms
   - Batch Algorithms
   - Online Algorithms

4. Applications

5. Conclusions
$\ell_1$ regularization promotes **sparse models**

A very simple sparsity pattern: **small cardinality**
$\ell_1$ regularization promotes **sparse models**

A very simple sparsity pattern: **small cardinality**

**Main question:** how to promote less trivial sparsity patterns?
Structured Sparsity and Groups

Main goal: promote **structural patterns**, not just penalize cardinality
Structured Sparsity and Groups

Main goal: promote **structural patterns**, not just penalize cardinality

**Group sparsity:** discard entire *groups* of features

- **density** inside each group
- **sparsity** with respect to the groups which are selected
- choice of groups: prior knowledge about the intended *sparsity patterns*

Yields statistical gains if prior assumptions are correct (Stojnic et al., 2009)
Structured Sparsity and Groups

Main goal: promote \textit{structural patterns}, not just penalize cardinality

\textbf{Group sparsity:} discard entire \textit{groups} of features

- \textbf{density} inside each group
- \textbf{sparsity} with respect to the groups which are selected
- choice of groups: prior knowledge about the intended \textit{sparsity patterns}

Yields statistical gains if prior assumptions are correct (Stojnic et al., 2009)
Example: Sparsity in a Grid

Assume the feature map decomposes as $f(x, y) = f(x) \otimes e_y$

In words: we’re conjoining each input feature with each output class

---

Input features

Labels

Dense

Sparse

"Standard" sparsity is wasteful—we may still need all the input features.

What we want:

discard some input features

Solution:

one group per input feature (conjoined with each of the labels)

Similar structure: multi-task learning (Caruana, 1997; Obozinski et al., 2010), multiple kernel learning (Lanckriet et al., 2004)
Example: Sparsity in a Grid

Assume the feature map decomposes as $f(x, y) = f(x) \otimes e_y$

In words: we’re conjoining each input feature with each output class

"Standard" sparsity is wasteful—we may still need all the input features

What we want: discard some input features
Example: Sparsity in a Grid

Assume the feature map decomposes as $f(x, y) = f(x) \otimes e_y$

In words: we’re conjoining each input feature with each output class

“Standard” sparsity is wasteful—we may still need all the input features

What we want: discard some input features

Solution: one group per input feature (conjoined with each of the labels)
Example: Sparsity in a Grid

Assume the feature map decomposes as $f(x, y) = f(x) \otimes e_y$

In words: we’re conjoining each input feature with each output class

“Standard” sparsity is wasteful—we may still need all the input features

What we want: discard some input features

Solution: one group per input feature (conjoined with each of the labels)
Example: Sparsity in a Grid

Assume the feature map decomposes as $f(x, y) = f(x) \otimes e_y$

In words: we’re conjoining each input feature with each output class

“Standard” sparsity is wasteful—we may still need all the input features

What we want: discard some input features

Solution: one group per input feature (conjoined with each of the labels)

Similar structure: multi-task learning (Caruana, 1997; Obozinski et al., 2010), multiple kernel learning (Lanckriet et al., 2004)
Group Sparsity

- $D$ features

Group-Lasso (Bakin, 1999; Yuan and Lin, 2006):

$$\Omega(w) = \sum_{m=1}^{M} \|w_m\|_2$$

Intuitively: the $\ell_1$ norm of the $\ell_2$ norms

Technically, still a norm (called a mixed norm, denoted $\ell_{2,1}$)

$\lambda_m$: prior weight for group $G_m$ (different groups have different sizes)

Statisticians call these composite absolute penalties (Zhao et al., 2009)
Group Sparsity

- $D$ features
- $M$ groups $G_1, \ldots, G_M$, each $G_m \subseteq \{1, \ldots, D\}$
- parameter subvectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$

Group-Lasso (Bakin, 1999; Yuan and Lin, 2006):

$$\Omega(\mathbf{w}) = \sum_{m=1}^{M} \| \mathbf{w}_m \|_2^2$$

Intuitively: the $\ell_1$ norm of the $\ell_2$ norms

Technically, still a norm (called a mixed norm, denoted $\ell_{2,1}$)

$\lambda_m$: prior weight for group $G_m$ (different groups have different sizes)

Statisticians call these composite absolute penalties (Zhao et al., 2009)
Group Sparsity

- $D$ features
- $M$ groups $G_1, \ldots, G_M$, each $G_m \subseteq \{1, \ldots, D\}$
- parameter subvectors $w_1, \ldots, w_M$

**Group-Lasso** (Bakin, 1999; Yuan and Lin, 2006):

\[
\Omega(w) = \sum_{m=1}^{M} \| w_m \|_2
\]
Group Sparsity

- $D$ features
- $M$ groups $G_1, \ldots, G_M$, each $G_m \subseteq \{1, \ldots, D\}$
- parameter subvectors $w_1, \ldots, w_M$

**Group-Lasso** (Bakin, 1999; Yuan and Lin, 2006):

$$\Omega(w) = \sum_{m=1}^{M} \|w_m\|_2$$

- Intuitively: the $\ell_1$ norm of the $\ell_2$ norms
- Technically, still a norm (called a *mixed* norm, denoted $\ell_{2,1}$)
Group Sparsity

- $D$ features
- $M$ groups $G_1, \ldots, G_M$, each $G_m \subseteq \{1, \ldots, D\}$
- parameter subvectors $w_1, \ldots, w_M$

Group-Lasso (Bakin, 1999; Yuan and Lin, 2006):

$$\Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2$$

- Intuitively: the $\ell_1$ norm of the $\ell_2$ norms
- Technically, still a norm (called a mixed norm, denoted $\ell_{2,1}$)
- $\lambda_m$: prior weight for group $G_m$ (different groups have different sizes)
Group Sparsity

- $D$ features
- $M$ groups $G_1, \ldots, G_M$, each $G_m \subseteq \{1, \ldots, D\}$
- parameter subvectors $w_1, \ldots, w_M$

**Group-Lasso** (Bakin, 1999; Yuan and Lin, 2006):

$$\Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2$$

- Intuitively: the $\ell_1$ norm of the $\ell_2$ norms
- Technically, still a norm (called a *mixed* norm, denoted $\ell_{2,1}$)
- $\lambda_m$: prior weight for group $G_m$ (different groups have different sizes)
- Statisticians call these **composite absolute penalties** (Zhao et al., 2009)
Lasso versus group-Lasso

\[ \Omega(w) = |w_1| + |w_2| + |w_3| \]
Lasso versus group-Lasso

$\Omega(w) = |w_1| + |w_2| + |w_3|$
Three Scenarios

- Non-overlapping groups
- Tree-structured groups
- Arbitrary groups
Three Scenarios

- Non-overlapping groups
- Tree-structured groups
- Arbitrary groups
Non-overlapping Groups

Assume $G_1, \ldots, G_M$ are disjoint

$\Rightarrow$ Each feature belongs to exactly one group
Non-overlapping Groups

Assume $G_1, \ldots, G_M$ are disjoint

$\Rightarrow$ Each feature belongs to exactly one group

\[
\Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2
\]

Trivial choices of groups recover unstructured regularizers:

- **$\ell_2$-regularization**: one large group $G_1 = \{1, \ldots, D\}$
- **$\ell_1$-regularization**: $D$ singleton groups $G_d = \{d\}$
Non-overlapping Groups

Assume $G_1, \ldots, G_M$ are disjoint

$\Rightarrow$ Each feature belongs to exactly one group

$$\Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2$$

Trivial choices of groups recover unstructured regularizers:

- $\ell_2$-regularization: one large group $G_1 = \{1, \ldots, D\}$
- $\ell_1$-regularization: $D$ singleton groups $G_d = \{d\}$
Non-overlapping Groups

Assume $G_1, \ldots, G_M$ are disjoint

$\Rightarrow$ Each feature belongs to exactly one group

$$\Omega(w) = \sum_{m=1}^{M} \lambda_m \| w_m \|_2$$

Trivial choices of groups recover unstructured regularizers:

- $\ell_2$-regularization: one large group $G_1 = \{1, \ldots, D\}$
- $\ell_1$-regularization: $D$ singleton groups $G_d = \{d\}$

Examples of non-trivial groups:

- label-based groups (groups are columns of a matrix)
- template-based groups (next)
Example: Feature Template Selection

Input:
We want to explore the feature space

Output:
(NP) (VP VP VP) (NP NP NP)

Goal: Select relevant feature templates
⇒ Make each group correspond to a feature template
Example: Feature Template Selection

Input: We want to explore the feature space

Output: (NP) (VP VP VP) (NP NP NP)
Example: Feature Template Selection

**Input:** We want to explore the feature space

PRP  VBP  TO  VB  DT  NN  NN

**Output:** (NP) (VP VP VP) (NP NP NP)

- Goal: Select relevant **feature templates**
Example: Feature Template Selection

Input: We want to explore the feature space
Output: (NP) (VP VP VP) (NP NP NP)

Goal: Select relevant feature templates
Example: Feature Template Selection

Input: We want to explore the feature space

Output: (NP) (VP VP VP) (NP NP NP)

Goal: Select relevant feature templates
Example: Feature Template Selection

Input: We want to explore the feature space

Output: (NP) (VP VP VP) (NP NP NP)

Goal: Select relevant feature templates
**Example: Feature Template Selection**

**Input:** We want to explore the feature space

**Output:** (NP) (VP VP VP) (NP NP NP)

- Goal: Select relevant *feature templates*
Example: Feature Template Selection

Input: We want to explore the feature space
       PRP VBP TO VB DT NN NN

Output: (NP) (VP VP VP) (NP NP NP)

Goal: Select relevant feature templates

"explore the"
"the feature"
Example: Feature Template Selection

Input: We want to explore the feature space

Output: (NP) (VP VP VP) (NP NP NP)

Goal: Select relevant feature templates
Example: Feature Template Selection

Input: We want to explore the feature space

Output: (NP) (VP VP VP) (NP NP NP)

- Goal: Select relevant feature templates
  - Make each group correspond to a feature template

"explore the"
"the feature"
"DT NN NN"
"VB DT NN"
**Example: Feature Template Selection**

**Input:**

```
We want to explore the feature space
```

**Output:**

```
(NP) (VP VP VP) (NP NP NP)
```

- **Goal:** Select relevant **feature templates**
  - Make each group correspond to a feature template
Example: Feature Template Selection

**Input:** We want to explore the feature space

PRP VBP TO VB DT NN NN

**Output:** (NP) (VP VP VP) (NP NP NP)

- **Goal:** Select relevant **feature templates**
  - Make each group correspond to a feature template
Example: Feature Template Selection

**Input:**

```
We  want  to  explore  the  feature  space
```

**Output:**

```
(NP)  (VP  VP  VP)  (NP  NP  NP)
```

- **Goal:** Select relevant feature templates
  - Make each group correspond to a feature template

![Diagram](image-url)
Example: Feature Template Selection

**Input:**

We want to explore the feature space

**Output:**

(NP) (VP VP VP) (NP NP NP)

- Goal: Select relevant **feature templates**
  - Make each group correspond to a feature template
Three Scenarios

- Non-overlapping groups
- Tree-structured groups
- Arbitrary groups
Three Scenarios

- Non-overlapping groups
- Tree-structured groups
- Arbitrary groups
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other

⇒ **hierarchical** structure (Kim and Xing, 2010; Mairal et al., 2010)
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other
⇒ *hierarchical* structure (Kim and Xing, 2010; Mairal et al., 2010)
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other

⇒ hierarchical structure (Kim and Xing, 2010; Mairal et al., 2010)
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other

⇒ hierarchical structure (Kim and Xing, 2010; Mairal et al., 2010)
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other
⇒ hierarchical structure (Kim and Xing, 2010; Mairal et al., 2010)
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other

⇒ **hierarchical** structure (Kim and Xing, 2010; Mairal et al., 2010)

What is the **sparsity pattern**?
Tree-Structured Groups

Assumption: if two groups overlap, one contains the other
⇒ hierarchical structure (Kim and Xing, 2010; Mairal et al., 2010)

- What is the sparsity pattern?
- If a group is discarded, all its descendants are also discarded
Three Scenarios

- Non-overlapping groups
- Tree-structured groups
- Arbitary Groups
Three Scenarios

- Non-overlapping groups
- Tree-structured groups
- Arbitrary Groups
Arbitrary Groups

In general: groups can be represented as a directed acyclic graph

- set inclusion induces a partial order on groups (Jenatton et al., 2009)
- feature space becomes a poset
- sparsity patterns: given by this poset
Example: Coarse-to-Fine Regularization

1. Define a partial order between basic feature templates (e.g., $p_0 \preceq w_0$)
2. Extend this partial order to all templates by lexicographic closure:
   $$p_0 \preceq p_0 p_1 \preceq w_0 w_1$$

**Goal:** only include \textit{finer} features if \textit{coarser} ones are also in the model
Things to Keep in Mind

- **Structured sparsity** cares about the *structure* of the feature space
- **Group-Lasso regularization** generalizes $\ell_1$ and it’s still convex
Things to Keep in Mind

- **Structured sparsity** cares about the *structure* of the feature space.
- **Group-Lasso regularization** generalizes $\ell_1$ and it’s still convex.
- **Choice of groups:** problem dependent, opportunity to use prior knowledge to favour certain structural patterns.
Things to Keep in Mind

- **Structured sparsity** cares about the *structure* of the feature space
- **Group-Lasso regularization** generalizes $\ell_1$ and it’s still convex
- **Choice of groups:** problem dependent, opportunity to use prior knowledge to favour certain structural patterns
- **Next:** algorithms
- We’ll see that optimization is easier with non-overlapping or tree-structured groups than with arbitrary overlaps
Outline

1. Sparsity and Feature Selection
2. Structured Sparsity
3. Algorithms
   - Batch Algorithms
   - Online Algorithms
4. Applications
5. Conclusions
Learning the Model

Recall that learning involves solving

\[
\min_w \Omega(w) + \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i),
\]

Two kinds of optimization algorithms:
- Batch algorithms (attack the complete problem)
- Online algorithms (use the training examples one by one)

We'll focus on proximal gradient algorithms (both batch and online)
Learning the Model

Recall that learning involves solving

$$\min_w \underbrace{\Omega(w)}_{\text{regularizer}} + \frac{1}{N} \sum_{i=1}^N L(w, x_i, y_i),$$

Two kinds of optimization algorithms:

- batch algorithms (attacks the complete problem);
- online algorithms (use the training examples one by one)
Learning the Model

Recall that learning involves solving

$$\min_w \Omega(w) + \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i),$$

Two kinds of optimization algorithms:

- batch algorithms (attacks the complete problem);
- online algorithms (use the training examples one by one)

*We’ll focus on proximal gradient algorithms (both batch and online)*
A Key Ingredient: Proximity Operator

The $\Omega$-proximity operator is the following $\mathbb{R}^D \to \mathbb{R}^D$ map:

$$w \mapsto \text{prox}_\Omega(w) = \arg \min_u \frac{1}{2} \|u - w\|^2 + \Omega(u)$$

(A generalization of Euclidean projection)
A Key Ingredient: Proximity Operator

The $\Omega$-proximity operator is the following $\mathbb{R}^D \rightarrow \mathbb{R}^D$ map:

$$w \mapsto \text{prox}_\Omega(w) = \arg \min_u \frac{1}{2} \|u - w\|^2 + \Omega(u)$$

(A generalization of Euclidean projection)

- $\ell_2$ regularization $\Omega(w) = \frac{\lambda}{2} \|w\|^2_2 \Rightarrow$ scaling operation
A Key Ingredient: Proximity Operator

The Ω-proximity operator is the following $\mathbb{R}^D \to \mathbb{R}^D$ map:

$$w \mapsto \text{prox}_\Omega(w) = \arg \min_u \frac{1}{2} \|u - w\|^2 + \Omega(u)$$

(A generalization of Euclidean projection)

- $\ell_2$ regularization $\Omega(w) = \frac{\lambda}{2} \|w\|_2^2 \Rightarrow$ scaling operation
A Key Ingredient: Proximity Operator

The **Ω-proximity operator** is the following \( \mathbb{R}^D \to \mathbb{R}^D \) map:

\[
\mathbf{w} \mapsto \text{prox}_\Omega(\mathbf{w}) = \arg \min_{\mathbf{u}} \frac{1}{2}\|\mathbf{u} - \mathbf{w}\|^2 + \Omega(\mathbf{u})
\]

(A generalization of Euclidean projection)

- **\( \ell_2 \) regularization** \( \Omega(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \) \( \Rightarrow \) **scaling operation**

- **\( \ell_1 \) regularization** \( \Omega(\mathbf{w}) = \lambda\|\mathbf{w}\|_1 \) \( \Rightarrow \) **soft-thresholding**:

\[
[\text{prox}_\Omega(\mathbf{w})]_d = \begin{cases} 
  w_d - \lambda & \text{if } w_d > \lambda \\
  0 & \text{if } |w_d| \leq \lambda \\
  w_d + \lambda & \text{if } w_d < -\lambda.
\end{cases}
\]
Proximity Operators for Structured Sparsity

\[ \Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2 \]

- Non-overlapping \( \Rightarrow \) vector soft-thresholding:

\[
[\text{prox}_\Omega(w)]_m = \begin{cases} 
0 & \text{if } \|w_m\|_2 \leq \lambda_m \\
\frac{\|w_m\|_2 - \lambda_m}{\|w_m\|_2} w_m & \text{otherwise.}
\end{cases}
\]
Proximity Operators for Structured Sparsity

\[ \Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2 \]

- **Non-overlapping**  \( \Rightarrow \) **vector soft-thresholding**:

\[
[\text{prox}_\Omega(w)]_m = \begin{cases} 
0 & \text{if } \|w_m\|_2 \leq \lambda_m \\
\frac{\|w_m\|_2 - \lambda_m}{\|w_m\|_2} w_m & \text{otherwise}.
\end{cases}
\]

- **Tree-structured**: can be computed recursively (Jenatton et al., 2010)
Proximity Operators for Structured Sparsity

\[ \Omega(w) = \sum_{m=1}^{M} \lambda_m \|w_m\|_2 \]

- **Non-overlapping** ⇒ **vector soft-thresholding**:

\[
[\text{prox}_\Omega(w)]_m = \begin{cases} 
0 & \text{if } \|w_m\|_2 \leq \lambda_m \\
\frac{\|w_m\|_2 - \lambda_m}{\|w_m\|_2} w_m & \text{otherwise.}
\end{cases}
\]

- **Tree-structured**: can be computed recursively (Jenatton et al., 2010)

- **Arbitrary groups**: no efficient procedure is known
Proximity Operators for Structured Sparsity

\[ \Omega(w) = \sum_{m=1}^{M} \lambda_m \| w_m \|_2 \]

- **Non-overlapping** \( \Rightarrow \) vector soft-thresholding:

  \[
  \text{[prox}_\Omega(w)]_m = \begin{cases} 
  0 & \text{if } \| w_m \|_2 \leq \lambda_m \\
  \frac{\| w_m \|_2 - \lambda_m}{\| w_m \|_2} w_m & \text{otherwise.}
  \end{cases}
  \]

- **Tree-structured**: can be computed recursively (Jenatton et al., 2010)

- **Arbitrary groups**: no efficient procedure is known

  The problem can be sidestepped with sequential proximity steps (Martins et al., 2011a) (more later).
Outline

1 Sparsity and Feature Selection

2 Structured Sparsity

3 Algorithms
   - Batch Algorithms
   - Online Algorithms

4 Applications

5 Conclusions
Iterative Shrinkage-Thresholding (IST)

\[
\min_{\mathbf{w}} \Omega(\mathbf{w}) + \Lambda(\mathbf{w}) , \quad \text{where} \quad \Lambda(\mathbf{w}) := \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{w}, x_i, y_i)
\]
Iterative Shrinkage-Thresholding (IST)

\[
\min_w \Omega(w) + \Lambda(w), \quad \text{where } \Lambda(w) := \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i)
\]

Building blocks:
- loss gradient/subgradient \( \nabla \Lambda \), proximity operator \( \text{prox}_\Omega \)
Iterative Shrinkage-Thresholding (IST)

\[
\min_w \Omega(w) + \Lambda(w), \quad \text{where } \Lambda(w) := \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i)
\]

Building blocks:

- loss gradient/subgradient \(\nabla \Lambda\), proximity operator \(\text{prox}_{\Omega}\)

\[
w_{t+1} \leftarrow \text{prox}_{\eta_t \Omega} (w_t - \eta_t \nabla \Lambda(w_t))
\]
Iterative Shrinkage-Thresholding (IST)

\[
\min_w \Omega(w) + \Lambda(w), \quad \text{where } \Lambda(w) := \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i)
\]

Building blocks:

- loss gradient/subgradient \( \nabla \Lambda \), proximity operator \( \text{prox}_{\Omega} \)

\[
w_{t+1} \leftarrow \text{prox}_{\eta_t \Omega} \left( w_t - \eta_t \nabla \Lambda(w_t) \right)
\]

Can be derived with different tools:

- expectation-maximization (EM) (Figueiredo and Nowak, 2003);
- majorization-minimization (Daubechies et al., 2004);
- forward-backward splitting (Combettes and Wajs, 2006);
- separable approximation (Wright et al., 2009).
Iterative Shrinkage-Thresholding (IST)

\[
\min_w \Omega(w) + \Lambda(w), \quad \text{where } \Lambda(w) := \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i)
\]

Building blocks:

- loss gradient/subgradient \(\nabla \Lambda\), proximity operator \(\text{prox}_\Omega\)

\[
w_{t+1} \leftarrow \text{prox}_{\eta t \Omega}(w_t - \eta t \nabla \Lambda(w_t))
\]

Can be derived with different tools:

- expectation-maximization (EM) (Figueiredo and Nowak, 2003);
- majorization-minimization (Daubechies et al., 2004);
- forward-backward splitting (Combettes and Wajs, 2006);
- separable approximation (Wright et al., 2009).

**Convergence:** requires \(O(1/\epsilon)\) iterations for \(\epsilon\)-accurate objective.
Other Proximal-Gradient Variants

**SpaRSA** (Wright et al., 2009): the same IST update scheme, but setting $\eta_t$ to mimic a Newton step (Barzilai and Borwein, 1988):

$$\eta_t^{-1}I \approx H(w_t) \quad \text{(Hessian)}$$

- Works very well in practice!
Other Proximal-Gradient Variants

**SpaRSA** (Wright et al., 2009): the same IST update scheme, but setting $\eta_t$ to mimic a Newton step (Barzilai and Borwein, 1988):

$$\eta_t^{-1} I \approx H(w_t) \quad \text{(Hessian)}$$

- Works very well in practice!

**FISTA** (Beck and Teboulle, 2009): compute $w_{t+1}$ based, not only on $w_t$, but also on $w_{t-1}$ (Nesterov, 1983):

\[
\begin{align*}
    b_{t+1} &= \frac{1 + \sqrt{1 + 4 b_t^2}}{2} \\
    z &= w_t + \frac{b_t - 1}{b_{t+1}} (w_t - w_{t-1}) \\
    w_{t+1} &= \text{prox}_{\eta \Omega} (z - \eta \nabla \Lambda(z))
\end{align*}
\]
Other Proximal-Gradient Variants

**SpaRSA** (Wright et al., 2009): the same IST update scheme, but setting $\eta_t$ to mimic a Newton step (Barzilai and Borwein, 1988):

$$\eta_t^{-1} I \approx H(w_t) \quad \text{(Hessian)}$$

- Works very well in practice!

**FISTA** (Beck and Teboulle, 2009): compute $w_{t+1}$ based, not only on $w_t$, but also on $w_{t-1}$ (Nesterov, 1983):

$$
\begin{align*}
b_{t+1} &= \frac{1 + \sqrt{1 + 4 b_t^2}}{2} \\
z &= w_t + \frac{b_{t-1}}{b_{t+1}} (w_t - w_{t-1}) \\
w_{t+1} &= \text{prox}_{\eta \Omega} (z - \eta \nabla \Lambda(z))
\end{align*}
$$

- **Iteration bound**: $O(1/\sqrt{\epsilon})$ as opposed to $O(1/\epsilon)$.
Many Other Batch Algorithms

- coordinate descent (Shevade and Keerthi, 2003; Genkin et al., 2007; Krishnapuram et al., 2005; Tseng and Yun, 2009)
- Least Angle Regression (LARS) and homotopy/continuation methods (Efron et al., 2004; Osborne et al., 2000; Figueiredo et al., 2007)
- shooting method (Fu, 1998)
- grafting (Perkins et al., 2003) and grafting-light (Zhu et al., 2010)
- orthant-wise limited-memory quasi-Newton (OWL-QN) (Andrew and Gao, 2007; Gao et al., 2007)
- alternating direction method of multipliers (ADMM) (Afonso et al., 2010; Figueiredo and Bioucas-Dias, 2011).

...several more; this is an active research area!
Outline

1. Sparsity and Feature Selection
2. Structured Sparsity
3. Algorithms
   - Batch Algorithms
   - Online Algorithms
4. Applications
5. Conclusions
Why Online?

1. Suitable for large datasets
Why Online?

1. Suitable for large datasets
2. Suitable for structured prediction
Why Online?

1. Suitable for large datasets
2. Suitable for structured prediction
3. Faster to approach a near-optimal region
Why Online?

1. Suitable for large datasets
2. Suitable for structured prediction
3. Faster to approach a near-optimal region
4. Slower convergence, but this is fine in machine learning ("the tradeoffs of large scale learning" by Bottou and Bousquet (2007))
Plain Stochastic (Sub-)Gradient Descent

\[
\min_w \left( \Omega(w) + \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i) \right),
\]

initialize \( w = 0 \)

\textbf{for} \( t = 1, 2, \ldots \) \textbf{do}

- take training pair \((x_t, y_t)\)
- (sub-)gradient step: \( w \leftarrow w - \eta_t \left( \nabla \Omega(w) + \nabla L(w; x_t, y_t) \right) \)

\textbf{end for}
Plain Stochastic (Sub-)Gradient Descent

\[
\min_w \Omega(w) + \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i),
\]

initialize \( w = 0 \)

for \( t = 1, 2, \ldots \) do
  take training pair \((x_t, y_t)\)
  (sub-)gradient step: \( w \leftarrow w - \eta_t \left( \tilde{\nabla} \Omega(w) + \tilde{\nabla} L(w; x_t, y_t) \right) \)
end for

- \( \ell_1 \)-regularization: \( \Omega(w) = \lambda \|w\|_1 \) \( \Rightarrow \tilde{\nabla} \Omega(w) = \lambda \text{sign}(w) \)

\[
w \leftarrow \underbrace{w - \eta_t \lambda \text{sign}(w)}_{\text{constant penalty}} - \eta_t \tilde{\nabla} L(w; x_t, y_t)
\]
Plain Stochastic (Sub-)Gradient Descent

\[
\min_w \Omega(w) + \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i),
\]

initialize \( w = 0 \)

for \( t = 1, 2, \ldots \) do
  take training pair \((x_t, y_t)\)
  (sub-)gradient step: \( w \leftarrow w - \eta_t (\tilde{\nabla} \Omega(w) + \tilde{\nabla} L(w; x_t, y_t)) \)
end for

- \( \ell_1 \)-regularization \( \Omega(w) = \lambda \|w\|_1 \) \( \implies \tilde{\nabla} \Omega(w) = \lambda \text{sign}(w) \)

\[
w \leftarrow w - \eta_t \lambda \text{sign}(w) - \eta_t \tilde{\nabla} L(w; x_t, y_t)
\]

\( \eta_t \) constant penalty
Plain Stochastic (Sub-)Gradient Descent

\[
\min_w \Omega(w) + \frac{1}{N} \sum_{i=1}^{N} L(w, x_i, y_i),
\]

initialize \( w = 0 \)

for \( t = 1, 2, \ldots \) do

- take training pair \((x_t, y_t)\)
- (sub-)gradient step: \( w \leftarrow w - \eta_t (\tilde{\nabla} \Omega(w) + \tilde{\nabla} L(w; x_t, y_t)) \)

end for

- \( \ell_1 \)-regularization \( \Omega(w) = \lambda \|w\|_1 \) \( \implies \tilde{\nabla} \Omega(w) = \lambda \text{sign}(w) \)

\[
\begin{align*}
\text{w} & \leftarrow \text{w} - \eta_t \lambda \text{sign(w)} - \eta_t \tilde{\nabla} L(w; x_t, y_t) \\
\text{constant penalty}
\end{align*}
\]

- Problem: iterates are never sparse!
Plain SGD with $\ell_1$-regularization

- loss gradient step
- regularizer gradient step
Plain SGD with $\ell_1$-regularization

- loss gradient step
- regularizer gradient step
Plain SGD with $\ell_1$-regularization

loss gradient step
regularizer gradient step
Plain SGD with $\ell_1$-regularization
Plain SGD with $\ell_1$-regularization

- Blue arrow: loss gradient step
- Red arrow: regularizer gradient step
Plain SGD with $\ell_1$-regularization
Plain SGD with $\ell_1$-regularization

![Diagram showing the effect of $\ell_1$-regularization in plain SGD. The diagram illustrates the movement of parameters during training, with blue arrows representing the loss gradient step and red arrows representing the regularizer gradient step.](image-url)
Plain SGD with $\ell_1$-regularization

loss gradient step
regularizer gradient step
Plain SGD with $\ell_1$-regularization
Plain SGD with $\ell_1$-regularization
“Sparse” Online Algorithms

- Truncated Gradient (Langford et al., 2009)
- Online Forward-Backward Splitting (Duchi and Singer, 2009)
- Regularized Dual Averaging (Xiao, 2010)
- Online Proximal Gradient (Martins et al., 2011a)
Truncated Gradient (Langford et al., 2009)

- take gradients-step only with respect to the loss
- apply soft-thresholding
- converges to $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step **only with respect to the loss**
- apply soft-thresholding
- **converges to** $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step **only with respect to the loss**
- apply soft-thresholding
- **converges to** $\epsilon$-**accurate objective after** $O(1/\epsilon^2)$ **iterations**

![Diagram showing gradient step and soft thresholding step](image)

- gradient step
- soft thresholding step
Truncated Gradient (Langford et al., 2009)

- take gradients-step **only with respect to the loss**
- apply soft-thresholding
- converges to $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step only with respect to the loss
- apply soft-thresholding
- converges to $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step **only with respect to the loss**
- apply soft-thresholding
- **converges to** $\epsilon$-**accurate objective after** $O(1/\epsilon^2)$ **iterations**
Truncated Gradient (Langford et al., 2009)

- take gradients-step **only with respect to the loss**
- apply soft-thresholding
- **converges to** $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step only with respect to the loss
- apply soft-thresholding
- converges to $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step only with respect to the loss
- apply soft-thresholding
- converges to $\epsilon$-accurate objective after $O(1/\epsilon^2)$ iterations
Truncated Gradient (Langford et al., 2009)

- take gradients-step **only with respect to the loss**
- apply soft-thresholding
- **converges to** $\epsilon$-**accurate objective after** $O(1/\epsilon^2)$ **iterations**
“Sparse” Online Algorithms

- Truncated Gradient (Langford et al., 2009)
- Online Forward-Backward Splitting (Duchi and Singer, 2009)
- Regularized Dual Averaging (Xiao, 2010)
- Online Proximal Gradient (Martins et al., 2011a)
Online Forward-Backward Splitting (Duchi and Singer, 2009)

initialize \( w = 0 \)

for \( t = 1, 2, \ldots \) do
  take training pair \( (x_t, y_t) \)
  gradient step: \( w \leftarrow w - \eta_t \nabla L(w; x_t, y_t) \)
  proximal step: \( w \leftarrow \text{prox}_{\eta_t \Omega}(w) \)
end for

- generalizes truncated gradient to arbitrary regularizers \( \Omega \)
  - can tackle non-overlapping or hierarchical group-Lasso, but arbitrary
    overlaps are difficult to handle (more later)
- converges to \( \epsilon \)-accurate objective after \( O(1/\epsilon^2) \) iterations
“Sparse” Online Algorithms

- Truncated Gradient (Langford et al., 2009)
- Online Forward-Backward Splitting (Duchi and Singer, 2009)
- Regularized Dual Averaging (Xiao, 2010)
- Online Proximal Gradient (Martins et al., 2011a)
Prox-Grad with Overlaps (Martins et al., 2011a)

**Key idea:** decompose $\Omega(w) = \sum_{j=1}^{J} \Omega_j(w)$, where each $\Omega_j$ is non-overlapping, and apply **sequential proximal steps**:

- **Gradient step:** $w \leftarrow w - \eta_t \nabla L(\theta; x_t, y_t)$
- **Proximal steps:** $w \leftarrow \text{prox}_{\eta_t \Omega_j} \left( \text{prox}_{\eta_t \Omega_{j-1}} \left( \ldots \text{prox}_{\eta_t \Omega_1}(w) \right) \right)$
Prox-Grad with Overlaps (Martins et al., 2011a)

Key idea: decompose $\Omega(w) = \sum_{j=1}^{J} \Omega_j(w)$, where each $\Omega_j$ is non-overlapping, and apply sequential proximal steps:

\[
\text{gradient step: } w \leftarrow w - \eta_t \nabla L(\theta; x_t, y_t) \\
\text{proximal steps: } w \leftarrow \text{prox}_{\eta_t \Omega_J} \left( \text{prox}_{\eta_t \Omega_{J-1}} \left( \ldots \text{prox}_{\eta_t \Omega_1}(w) \right) \right)
\]

- still convergent, same $O(1/\epsilon^2)$ iteration bound
- gradient step: linear in $\#$ of features that fire, independent of $D$.
- proximal steps: linear in $\#$ of groups $M$.
- other implementation tricks (debiasing, budget-driven shrinkage, etc.)
Memory Footprint

- 5 epochs for identifying relevant groups, 10 epochs for debiasing
## Summary of Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Converges</th>
<th>Rate</th>
<th>Sparse</th>
<th>Groups</th>
<th>Overlaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coord. desc.</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>Maybe</td>
<td>No</td>
</tr>
<tr>
<td>Prox-grad</td>
<td>✓</td>
<td>$O(1/\epsilon)$</td>
<td>Yes/No</td>
<td>✓</td>
<td>Not easy</td>
</tr>
<tr>
<td>OWL-QN</td>
<td>✓</td>
<td>?</td>
<td>Yes/No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SpaRSA</td>
<td>✓</td>
<td>$O(1/\epsilon)$ or better</td>
<td>Yes/No</td>
<td>✓</td>
<td>Not easy</td>
</tr>
<tr>
<td>FISTA</td>
<td>✓</td>
<td>$O(1/\sqrt{\epsilon})$</td>
<td>Yes/No</td>
<td>✓</td>
<td>Not easy</td>
</tr>
<tr>
<td>ADMM</td>
<td>✓</td>
<td>$O(1/\epsilon)$</td>
<td>No</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Online subgrad.</td>
<td>✓</td>
<td>$O(1/\epsilon^2)$</td>
<td>No</td>
<td>✓</td>
<td>No</td>
</tr>
<tr>
<td>Truncated grad.</td>
<td>✓</td>
<td>$O(1/\epsilon^2)$</td>
<td>✓</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>FOBOS</td>
<td>✓</td>
<td>$O(1/\epsilon^2)$</td>
<td>Sort of</td>
<td>✓</td>
<td>Not easy</td>
</tr>
<tr>
<td>Online prox-grad</td>
<td>✓</td>
<td>$O(1/\epsilon^2)$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Outline

1. Sparsity and Feature Selection

2. Structured Sparsity

3. Algorithms
   - Batch Algorithms
   - Online Algorithms

4. Applications

5. Conclusions
Applications of Structured Sparsity in ML

We will focus on two recent NLP applications (Martins et al., 2011b):

- Named entity recognition
- Dependency parsing

We use feature templates as groups.
Only France and Britain backed Fischler’s proposal.
Only France and Britain backed Fischler’s proposal.

[Tokenization and Named Entity Recognition]

- Only: RB
- France: NNP
- and: CC
- Britain: NNP
- backed: VBD
- Fischler: NNP
- ’s: POS
- proposal: NN
Only France and Britain backed Fischler’s proposal.

Spanish, Dutch, and English CoNLL datasets

- 452 feature templates using POS tags, words, shapes, affixes, with various context sizes
Named Entity Recognition

Only France and Britain backed Fischler’s proposal.

Spanish, Dutch, and English CoNLL datasets

452 feature templates using POS tags, words, shapes, affixes, with various context sizes

Comparison between:

- $\ell_2$-regularization (MIRA), best $\lambda$ on dev-set, all features
- $\ell_1$-regularization (Lasso), varying $\lambda$
- $\ell_{2,1}$-regularization (Group Lasso), varying the template budget
Named entity models: number of features. ($\text{Lasso } C = 1/\lambda N.$)
Named entity models: $F_1$ accuracy on the test set. (Lasso $C = 1/\lambda N$.)
Logic plays a minimal role here.
Logic plays a minimal role here.

- Arabic, Danish, Dutch, Japanese, Slovene, Spanish CoNLL datasets
- 684 feature templates (using words, lemmas, POS, contextual POS, arc length and direction)
Dependency Parsing

- Logic plays a minimal role here

- Arabic, Danish, Dutch, Japanese, Slovene, Spanish CoNLL datasets
- 684 feature templates (using words, lemmas, POS, contextual POS, arc length and direction)

Comparison between:

- $\ell_2$-regularization (MIRA), all features
- filter-based template selection (information gain)
- $\ell_1$-regularization (Lasso)
- $\ell_{2,1}$-regularization (Group Lasso, coarse-to-fine regularization)
Template-based group lasso is better at selecting feature templates than the IG criterion, and slightly better than coarse-to-fine.
Outline

1. Sparsity and Feature Selection

2. Structured Sparsity

3. Algorithms
   - Batch Algorithms
   - Online Algorithms

4. Applications

5. Conclusions
Summary

- Sparsity is desirable in machine learning: feature selection, runtime, memory footprint, interpretability
- Beyond plain sparsity: **structured sparsity** can be promoted through group-Lasso regularization
- Choice of groups reflects prior knowledge about the desired sparsity patterns.
- Small/medium scale: many batch algorithms available, with fast convergence (IST, FISTA, SpaRSA, ...)
- Large scale: online proximal-gradient algorithms suitable to explore large feature spaces
Thank you!

- Questions?
Acknowledgments

- National Science Foundation (USA), CAREER grant IIS-1054319
- Fundação para a Ciência e Tecnologia (Portugal), grant PEst-OE/EEI/LA0008/2011.
- Fundação para a Ciência e Tecnologia and Information and Communication Technologies Institute (Portugal/USA), through the CMU-Portugal Program.
- Priberam: QREN/POR Lisboa (Portugal), EU/FEDER programme, Discooperio project, contract 2011/18501.


