

Numerical Optimization for Machine Learning

Group Sparsity, Structured Regularization, Kernel Methods

Mark Schmidt

University of British Columbia

Summer 2022 - Summer 2023

Outline

- 1 Group Sparsity
- 2 Structured Regularization
- 3 Kernel Trick
- 4 Valid Kernels and Representer Theorem
- 5 Large-Scale Kernel Methods

Motivation for Group Sparsity

- Recall that **multi-class logistic regression** uses

$$\hat{y}^i = \operatorname{argmax}_c \{w_c^\top x^i\},$$

where we have a **parameter vector** w_c for each class c .

- We typically use **softmax loss** and write our parameters as a matrix,

$$W = \begin{bmatrix} | & | & | & \cdots & | \\ w_1 & w_2 & w_3 & \cdots & w_k \\ | & | & | & & | \end{bmatrix}$$

- Suppose we want to use **L1-regularization for feature selection**,

$$\operatorname{argmin}_{W \in \mathbb{R}^{d \times k}} \underbrace{f(W)}_{\text{softmax loss}} + \lambda \underbrace{\sum_{c=1}^k \|w_c\|_1}_{\text{L1-regularization}} .$$

- Unfortunately, **setting elements of W to zero may not select features**.

Motivation for Group Sparsity

- Suppose L1-regularization gives a sparse W with a **non-zero in each row**:

$$W = \begin{bmatrix} -0.83 & 0 & 0 & 0 \\ 0 & 0 & 0.62 & 0 \\ 0 & 0 & 0 & -0.06 \\ 0 & 0.72 & 0 & 0 \end{bmatrix}.$$

- Even though it's very sparse, it uses **all features**.
 - Remember that classifier multiplies feature j by **each value in row j** .
 - Feature 1 is used in w_1 .
 - Feature 2 is used in w_3 .
 - Feature 3 is used in w_4 .
 - Feature 4 is used in w_2 .
- In order to remove a feature, we need its **entire row to be zero**.

Motivation for Group Sparsity


- What we want is **group sparsity**:

$$W = \begin{bmatrix} -0.77 & 0.04 & -0.03 & -0.09 \\ 0 & 0 & 0 & 0 \\ 0.04 & -0.08 & 0.01 & -0.06 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

- Each **row is a group**, and we want **groups (rows) of variables that have all zeroes**.
 - If row j is zero, then x_j is not used by the model.
- Pattern arises in other settings where each row gives parameters for one feature:
 - **Multiple regression**, **multi-label classification**, and **multi-task classification**.

Motivation for Group Sparsity

- **Categorical features** are another setting where **group sparsity** is needed.
- Consider categorical features encoded as **binary indicator** features (“1 of k ”):

City	Age		Vancouver	Burnaby	Surrey	Age ≤ 20	20 < Age ≤ 30	Age > 30
Vancouver	22		1	0	0	0	1	0
Burnaby	35		0	1	0	0	0	1
Vancouver	28		1	0	0	0	1	0

- A linear model would use

$$\hat{y}^i = w_1 x_{\text{van}} + w_2 x_{\text{bur}} + w_3 x_{\text{sur}} + w_4 x_{\leq 20} + w_5 x_{21-30} + w_6 x_{> 30}.$$

- If we want feature selection of **original categorical variables**, we have 2 groups:
 - $\{w_1, w_2, w_3\}$ correspond to “City” and $\{w_4, w_5, w_6\}$ correspond to “Age”.

Group L1-Regularization

- Consider a problem with a **set of disjoint groups** \mathcal{G} .
 - For example, $\mathcal{G} = \{\{1, 2\}, \{3, 4\}\}$.

- Minimizing a function f with **group L1-regularization**:

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda \sum_{g \in \mathcal{G}} \|w_g\|_p,$$

where g refers to individual group indices and $\|\cdot\|_p$ is some norm.

- For certain norms, it encourages **sparsity in terms of groups** g .
 - Variables x_1 and x_2 will either be **both zero or both non-zero**.
 - Variables x_3 and x_4 will either be **both zero or both non-zero**.

Group L1-Regularization

- Why is it called group **L1**-regularization?
- Consider $G = \{\{1, 2\}, \{3, 4\}\}$ and using L2-norm,

$$\sum_{g \in G} \|w_g\|_2 = \sqrt{w_1^2 + w_2^2} + \sqrt{w_3^2 + w_4^2}.$$

- If vector v contains the group norms, it's the **L1-norm of v** :

$$\text{If } v \triangleq \begin{bmatrix} \|w_{12}\|_2 \\ \|w_{34}\|_2 \end{bmatrix} \text{ then } \sum_{g \in G} \|w_g\|_2 = \|w_{12}\|_2 + \|w_{34}\|_2 = v_1 + v_2 = |v_1| + |v_2| = \|v\|_1.$$

- So group L1-regularization encourages **sparsity in the group norms**.
 - When the norm of the group is 0, all group elements are 0.

Group L1-Regularization: Choice of Norm

- The **group L1-regularizer** is sometimes written as a “mixed” norm,

$$\|w\|_{1,p} \triangleq \sum_{g \in \mathcal{G}} \|w_g\|_p.$$

- The most common choice for the norm is the **L2-norm**:
 - If $\mathcal{G} = \{\{1, 2\}, \{3, 4\}\}$ we obtain

$$\|w\|_{1,2} = \sqrt{w_1^2 + w_2^2} + \sqrt{w_3^2 + w_4^2}.$$

- Another common choice is the **L ∞ -norm**,

$$\|w\|_{1,\infty} = \max\{|w_1|, |w_2|\} + \max\{|w_3|, |w_4|\}.$$

- But note that the **L1-norm does not give group sparsity**,

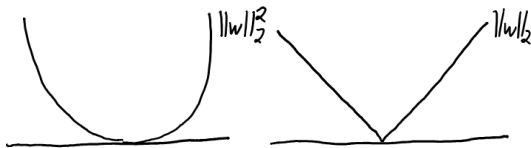
$$\|w\|_{1,1} = |w_1| + |w_2| + |w_3| + |w_4| = \|w\|_1,$$

as it's equivalent to non-group L1-regularization.

Sparsity from the L2-Norm?

- Didn't we say sparsity comes from the L1-norm and not the L2-norm?
 - Yes, but we were using the **squared L2-norm**.

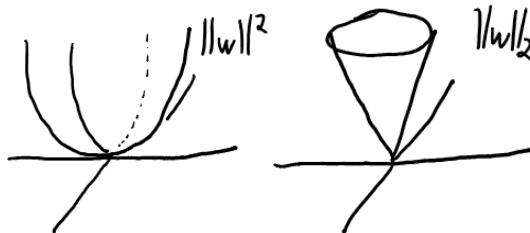
- Squared vs. non-squared L2-norm in 1D:



- **Non-squared L2-norm is absolute value.**
 - Non-squared L2-regularizer will set **all $w_j = 0$** for some finite λ .
- Squaring the L2-norm gives a smooth function but destroys sparsity.

Sparsity from the L2-Norm?

- Squared vs. non-squared L2-norm in 2D:



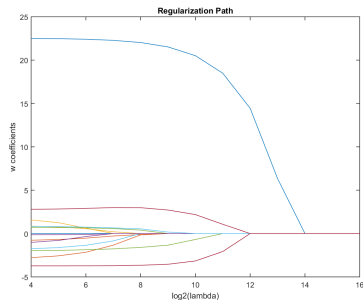
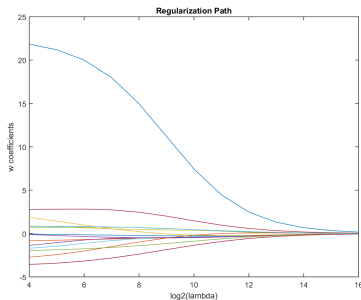
- The squared L2-norm is smooth and has no sparsity.
- Non-squared L2-norm is **non-smooth at the zero vector**.
 - It doesn't encourage us to set any $w_j = 0$ as long as one $w_{j'} \neq 0$.
 - But if λ is large enough it encourages all w_j to be set to 0.

L2 and L1 Regularization Paths

- The **regularization path** is the set of w values as λ varies,

$$w^\lambda = \operatorname{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda r(w),$$

- Squared L2-regularization path vs. L1-regularization path:



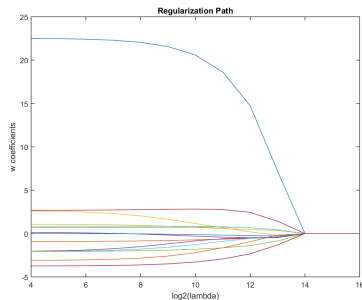
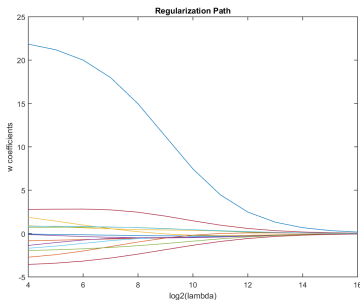
- With $r(w) = \|w\|^2$, each w_j gets close to 0 but is never exactly 0.
- With $r(w) = \|w\|_1$, each w_j gets set to exactly zero for a finite λ .

L² and L₂ Regularization Paths

- The **regularization path** is the set of w values as λ varies,

$$w^\lambda = \operatorname{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda r(w),$$

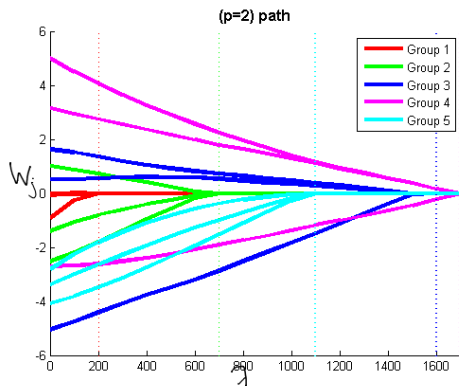
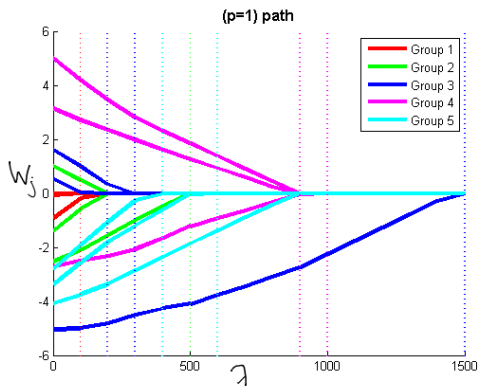
- Squared L₂-regularization path vs. **non-squared** path:



- With $r(w) = \|w\|^2$, each w_j gets close to 0 but is never exactly 0.
- With $r(w) = \|w\|_2$, **all** w_j get set to exactly zero for **same finite** λ .

Group L1-Regularization Paths

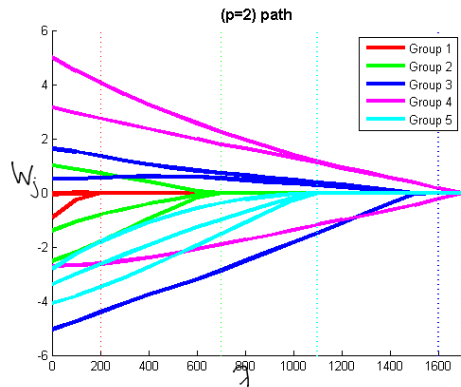
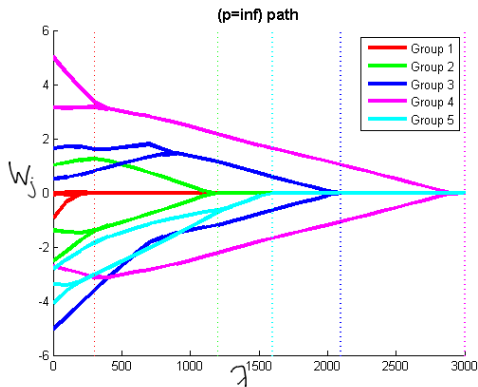
- The regularization path for group L1-regularization for different p values:



- With $p = 1$ there is **no grouping effect**.
- With $p = 2$ the **groups become zero at the same time**.

Group L1-Regularization Paths

- The regularization path for group L1-regularization for different p values:



- With $p = 1$ there is **no grouping effect**.
- With $p = 2$ the **groups become zero at the same time**.
- With $p = \infty$ the **groups converge to same magnitude which then goes to 0**.

Sub-differential of Group L1-Regularization

- For our **group L1-regularization** objective with the **2-norm**,

$$F(w) = f(w) + \lambda \sum_{g \in \mathcal{G}} \|w_g\|_2,$$

the indices g in the sub-differential are given by

$$\partial_g F(w) \equiv \nabla_g f(w) + \lambda \partial \|w_g\|_2.$$

- In order to have $0 \in \partial F(w)$, we thus need for each group that

$$0 \in \nabla_g f(w) + \lambda \partial \|w_g\|_2,$$

and subtracting $\nabla_g f(w)$ from both sides gives

$$-\nabla_g f(w) \in \lambda \partial \|w_g\|_2.$$

Sub-differential of Group L1-Regularization

- So at minimizer w^* we must have for all groups that

$$-\nabla_g f(w^*) \in \lambda \partial \|w_g^*\|_2.$$

- The **sub-differential of the scaled L2-norm** is given by the “signum” function,

$$\partial \|w\|_2 = \begin{cases} \left\{ \frac{w}{\|w\|_2} \right\} & w \neq 0 \\ \{v \mid \|v\|_2 \leq 1\} & w = 0. \end{cases}$$

- So at a solution w^* we have for each group that

$$\begin{cases} -\nabla_g f(w^*) = \lambda \frac{w_g^*}{\|w_g^*\|_2} & w_g \neq 0, \\ \|\nabla_g f(w^*)\| \leq \lambda & w_g^* = 0. \end{cases}$$

- For sufficiently-large λ we'll set the group to zero.

- With **squared group norms** we would need $\nabla_g f(w^*) = 0$ with $w_g^* = 0$ (unlikely).

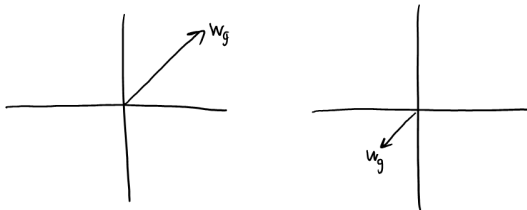
Proximal-Gradient for Group L1-Regularization

- The proximal operator for **group** L1-regularization,

$$\operatorname{argmin}_{v \in \mathbb{R}^d} \left\{ \frac{1}{2} \|v - w\|^2 + \alpha_k \lambda \sum_{g \in G} \|v\|_2 \right\},$$

applies a soft-threshold **group**-wise,

$$w_g \leftarrow \frac{w_g}{\|w_g\|_2} \max\{0, \|w_g\|_2 - \alpha_k \lambda\}.$$



- So we can **solve group L1-regularization problems as fast as smooth problems.**

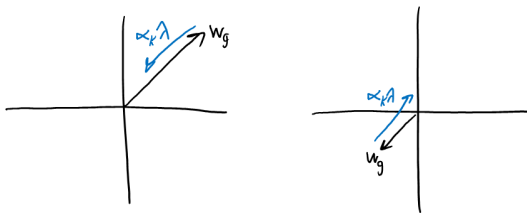
Proximal-Gradient for Group L1-Regularization

- The proximal operator for **group** L1-regularization,

$$\operatorname{argmin}_{v \in \mathbb{R}^d} \left\{ \frac{1}{2} \|v - w\|^2 + \alpha_k \lambda \sum_{g \in G} \|v\|_2 \right\},$$

applies a soft-threshold **group**-wise,

$$w_g \leftarrow \frac{w_g}{\|w_g\|_2} \max\{0, \|w_g\|_2 - \alpha_k \lambda\}.$$



- So we can **solve group L1-regularization problems as fast as smooth problems.**

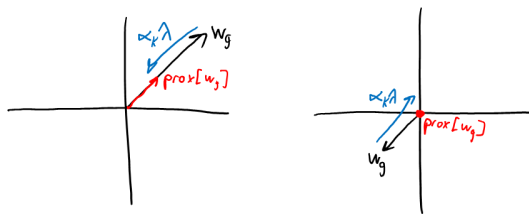
Proximal-Gradient for Group L1-Regularization

- The proximal operator for **group** L1-regularization,

$$\operatorname{argmin}_{v \in \mathbb{R}^d} \left\{ \frac{1}{2} \|v - w\|^2 + \alpha_k \lambda \sum_{g \in G} \|v\|_2 \right\},$$

applies a soft-threshold **group**-wise,

$$w_g \leftarrow \frac{w_g}{\|w_g\|_2} \max\{0, \|w_g\|_2 - \alpha_k \lambda\}.$$



- So we can **solve group L1-regularization problems as fast as smooth problems.**

Outline

- 1 Group Sparsity
- 2 Structured Regularization**
- 3 Kernel Trick
- 4 Valid Kernels and Representer Theorem
- 5 Large-Scale Kernel Methods

Structured Regularization

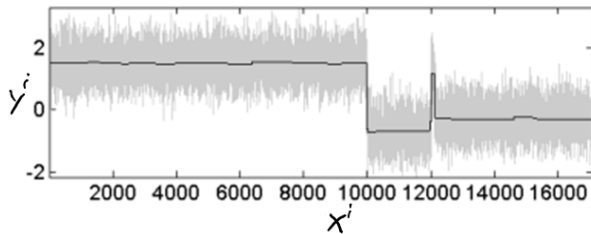
- There are many other patterns that regularization can encourage.
 - We'll call this structured regularization.
- The three most common cases:
 - Total-variation regularization encourages slow/sparse changes in w .
 - Nuclear-norm regularization encourages sparsity in rank of matrices.
 - Structured sparsity encourages sparsity in variable patterns.
- Cannot efficiently compute proximal operator, but can approximate them:
 - For total-variation and overlapping group-L1, we can use Dykstra's algorithm
 - Iterative method that computes proximal operator for sum of "simple" functions.
 - For nuclear-norm regularization, methods approximate top singular vectors.
 - Krylov subspace methods, randomized SVD approximations.

Total-Variation Regularization

- 1D **total-variation regularization** (“fused LASSO”) takes the form

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda \sum_{j=1}^{d-1} |w_j - w_{j+1}|.$$

- Encourages **consecutive parameters to have same value**.
- Often used for time-series or sequence data.



<http://statweb.stanford.edu/~bjk/regreg/examples/fusedlassoapprox.html>

Here we're trying to **estimate de-noised w_i** of y^i at each time x^i .

Total-Variation Regularization

- More generally, we could **penalize differences on general graph** between variables.
- An example is **social regularization** in recommender systems:
 - Penalizing the **difference between your parameters and your friends' parameters**.

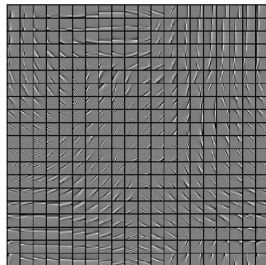
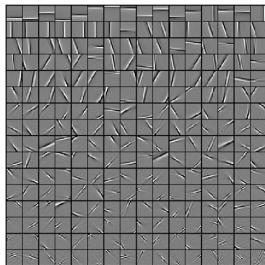
$$\operatorname{argmin}_{W \in \mathbb{R}^{d \times k}} f(W) + \lambda \sum_{(i,j) \in \text{Friends}} \|w_i - w_j\|^2.$$

- Typically use L2-regularization (we aren't aiming for identical parameters).



Total-Variation Regularization

- Consider applying **latent factor models** (from 340) on **image patches**.
 - Similar to learning first layer of convolutional neural networks.
- Latent-factors discovered on patches with/without TV regularization.
 - Encouraging **neighbours in a spatial grid to have similar filters**.

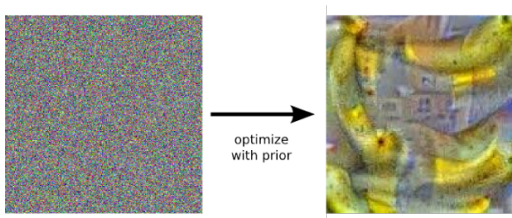


http://lear.inrialpes.fr/people/mairal/resources/pdf/review_sparse_arxiv.pdf

- Similar to “cortical columns” theory of visual cortex.

Total-Variation Regularization

- Another application is **inceptionism**.



<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

- Find **image x** that causes strongest activation of class c in neural network.

$$\operatorname{argmin}_x f(v_c^\top h(W^{(m)} h(W^{(m-1)} \dots h(W^{(1)} x))) + \lambda \sum_{(x_i, x_j) \in \text{neigh.}} (x_i - x_j)^2,$$

- Total variation based on neighbours in image (needed to get interpretable images).

Nuclear Norm Regularization

- With matrix parameters an alternative is **nuclear norm regularization**,

$$\operatorname{argmin}_{W \in \mathbb{R}^{d \times k}} f(W) + \lambda \|W\|_*,$$

where $\|W\|_*$ is the **sum of singular values**.

- “L1-regularization of the singular values”.
 - Encourages parameter **matrix to have low-rank**.
- Consider a multi-class logistic regression with a huge number of features/labels,

$$W = \begin{bmatrix} | & | & \cdots & | \\ w_1 & w_2 & \cdots & w_k \\ | & | & & | \end{bmatrix} = UV^\top, \quad \text{with} \quad U = \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix}, \quad V = \begin{bmatrix} | & | \\ v_1 & v_2 \\ | & | \end{bmatrix},$$

U and V can be much smaller, and $XW = (XU)V^\top$ can be computed faster:

- $O(ndk)$ cost reduced to $O(n dr + nkr)$ for rank r , much faster if $r < \min\{d, k\}$.

UV^\top Parameterization for Matrix Problems

- We discussed **nuclear norm regularization** problems,

$$\operatorname{argmin}_{W \in \mathbb{R}^{d \times k}} f(W) + \lambda \|W\|_*,$$

which gives a solution with a low rank representation $W = UV^\top$.

- But standard algorithms are **too costly** in many applications.
 - We often **can't store W** .
- Many recent approaches **directly minimize under UV^\top parameterization**,

$$\operatorname{argmin}_{U \in \mathbb{R}^{d \times R}, V \in \mathbb{R}^{k \times R}} f(UV^\top) + \lambda_U \|U\|_F^2 + \lambda_V \|V\|_F^2,$$

and just regularize U and V (here we're using the **Frobenius matrix norm**).

UV^T Parameterization for Matrix Problems

- We used this approach in 340 for **latent-factor models**,

$$f(W, Z) = \frac{1}{2} \|ZW - X\|_F^2 + \frac{\lambda_1}{2} \|Z\|_F^2 + \frac{\lambda_2}{2} \|W\|_F^2.$$

- We can sometimes prove these **non-convex** re-formulation give a global solution.
 - Includes **PCA**.
- In other cases, people are working hard on finding assumptions where this is true.
 - These assumptions are typically unrealistically strong.
 - But it works well enough in practice that practitioners don't seem to care.

Structured Sparsity

- **Structured sparsity** is variation on **group L1-regularization**,

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(w) + \sum_{g \in \mathcal{G}} \lambda_g \|w_g\|_p,$$

where now the **groups g can overlap**.

- Why is this interesting?

- Consider the case of two groups, $\{1\}$ and $\{1, 2\}$,

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda_1 |w_1| + \lambda_2 \sqrt{w_1^2 + w_2^2}.$$

- This encourages 3 non-zero “patterns”: $\{\}$, $\{w_2\}$, $\{w_1, w_2\}$.
 - “You can only take w_1 if you’ve already taken w_2 .”
- If $w_1 \neq 0$, the **third term is smooth** and doesn’t encourage w_2 to be zero.
- If $w_2 \neq 0$, we still pay a λ_1 penalty for making w_1 non-zero.
- We can use this type of “ordering” to **impose patterns on our sparsity**.

Structured Sparsity

- Consider a problem with matrix parameters W .
- We want W to be “band-limited”:
 - Non-zeros only on the main diagonals.

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & 0 & 0 & 0 & 0 \\ w_{21} & w_{22} & w_{23} & w_{24} & 0 & 0 & 0 \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & 0 & 0 \\ 0 & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & 0 \\ 0 & 0 & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \\ 0 & 0 & 0 & w_{64} & w_{65} & w_{66} & w_{67} \\ 0 & 0 & 0 & 0 & w_{75} & w_{76} & w_{77} \end{bmatrix}.$$

- This makes many computations much faster.
- We can enforce this with structured sparsity:
 - Only allow non-zeros on ± 1 diagonal if you are non-zero on main diagonal.
 - Only allow non-zeros on ± 2 diagonal if you are non-zero on ± 1 diagonal.
 - Only allow non-zeros on ± 3 diagonal if you are non-zero on ± 2 diagonal.

Structured Sparsity

- Consider a linear model with **higher-order terms**,

$$\hat{y}^i = w_0 + w_1 x_1^i + w_2 x_2^i + w_3 x_3^i + w_{12} x_1^i x_2^i + w_{13} x_1^i x_3^i + w_{23} x_2^i x_3^i + w_{123} x_1^i x_2^i x_3^i.$$

- If d is non-trivial, then the **number of higher-order terms is too large**.
- We can use **structured sparsity** to enforce a **hierarchy**.
 - We only allow $w_{12} \neq 0$ if $w_1 \neq 0$ and $w_2 \neq 0$.
 - You can enforce this using the groups $\{\{w_{12}\}, \{w_1, w_{12}\}, \{w_2, w_{12}\}\}$:

$$\operatorname{argmin}_w f(w) + \lambda_{12}|w_{12}| + \lambda_1 \sqrt{w_1^2 + w_{12}^2} + \lambda_2 \sqrt{w_2^2 + w_{12}^2}.$$

Structured Sparsity

- We can use **structured sparsity** to enforce a **hierarchy**.
 - We only allow $w_{12} \neq 0$ if $w_1 \neq 0$ and $w_2 \neq 0$.
 - We only allow $w_{123} \neq 0$ if $w_{12} \neq 0$, $w_{13} \neq 0$, and $w_{23} \neq 0$.
 - We only allow $w_{1234} \neq 0$ if all threeway interactions are present.

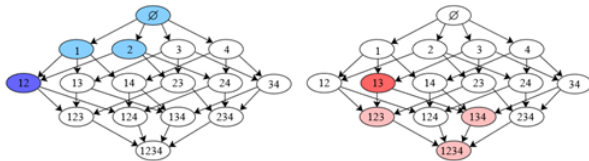


Fig 9: Power set of the set $\{1, \dots, 4\}$: in blue, an authorized set of selected subsets. In red, an example of a group used within the norm (a subset and all of its descendants in the DAG).

<http://arxiv.org/pdf/1109.2397v2.pdf>

- For certain bases, you can **work with the full hierarchy in polynomial time**.
 - Otherwise, a heuristic is to gradually “grow” the set of allowed bases.

Structured Sparsity

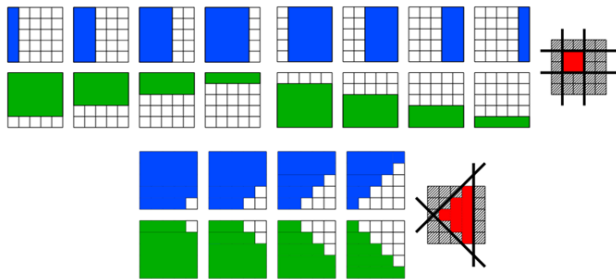
- Structured sparsity encourages zeroes to be **any intersections of groups**.
 - Possible non-zeroes are given by $\cap_{g \in \mathcal{G}'} g^c$ for all $\mathcal{G}' \subseteq \mathcal{G}$.
 - Equivalently, the set of zeroes is any $\cup_{g \in \mathcal{G}'} g$.
 - Our first example used $\{1\}$ and $\{1, 2\}$ so possible non-zeroes $\{\}, \{2\}$, or $\{1, 2\}$.
 - E.g., $\{2\}$ is $\{1, 2\} \cap \{1\}^c = \{1, 2\} \cap \{2\}$.
- Example is enforcing **convex non-zero patterns**:



Fig 3: (Left) The set of blue groups to penalize in order to select contiguous patterns in a sequence. (Right) In red, an example of such a nonzero pattern with its corresponding zero pattern (hatched area).

Structured Sparsity

- Structured sparsity encourages zeroes to be **any intersections of groups**.
 - Possible non-zeroes are given by $\cap_{g \in \mathcal{G}'} g^c$ for all $\mathcal{G}' \subseteq \mathcal{G}$.
 - Equivalently, the set of zeroes is any $\cup_{g \in \mathcal{G}'} g$.
 - Our first example used $\{1\}$ and $\{1, 2\}$ so possible non-zeroes $\{\}, \{2\}$, or $\{1, 2\}$.
 - E.g., $\{2\}$ is $\{1, 2\} \cap \{1\}^c = \{1, 2\} \cap \{2\}$.
- Example is enforcing **convex non-zero patterns**:



Structured Sparsity

- Structured sparsity encourages zeroes to be **any intersections of groups**.
 - Possible non-zeroes are given by $\cap_{g \in \mathcal{G}'} g^c$ for all $\mathcal{G}' \subseteq \mathcal{G}$.
 - Equivalently, the set of zeroes is any $\cup_{g \in \mathcal{G}'} g$.
 - Our first example used $\{1\}$ and $\{1, 2\}$ so possible non-zeroes $\{\}, \{2\}$, or $\{1, 2\}$.
 - E.g., $\{2\}$ is $\{1, 2\} \cap \{1\}^c = \{1, 2\} \cap \{2\}$.
- Example is enforcing **convex non-zero patterns**:



<https://arxiv.org/pdf/1109.2397v2.pdf>

- Left-to-right: data, NMF, sparse PCA, and PCA with structured sparsity.

Structured Sparsity

- Structured sparsity encourages zeroes to be **any intersections of groups**.
 - Possible non-zeroes are given by $\cap_{g \in \mathcal{G}'} g^c$ for all $\mathcal{G}' \subseteq \mathcal{G}$.
 - Equivalently, the set of zeroes is any $\cup_{g \in \mathcal{G}'} g$.
 - Our first example used $\{1\}$ and $\{1, 2\}$ so possible non-zeroes $\{\}, \{2\}$, or $\{1, 2\}$.
 - E.g., $\{2\}$ is $\{1, 2\} \cap \{1\}^c = \{1, 2\} \cap \{2\}$.
- Example is enforcing **convex non-zero patterns**:

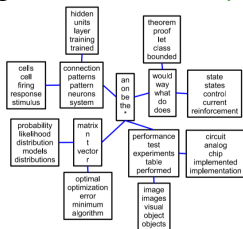


Figure 4. Example of a topic hierarchy estimated from 1714 NIPS proceedings papers (from 1988 through 1999). Each node corresponds to a topic whose 5 most important words are displayed. Single characters such as n, t, r are part of the vocabulary and often appear in NIPS papers, and their place in the hierarchy is semantically relevant to children topics.

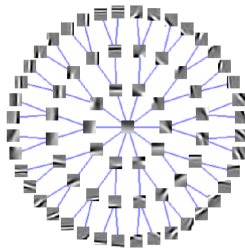


Figure 3. Learned dictionary with tree structure of depth 4. The root of the tree is in the middle of the figure. The branching factors are $p_1 = 10, p_2 = 2, p_3 = 2$. The dictionary is learned on 50,000 patches of size 16×16 pixels.

- There is also a variant (“over-LASSO”) that considers **unions of groups**.

Outline

- 1 Group Sparsity
- 2 Structured Regularization
- 3 Kernel Trick**
- 4 Valid Kernels and Representer Theorem
- 5 Large-Scale Kernel Methods

Motivation: Multi-Dimensional Polynomial Basis

- Consider quadratic **polynomial basis** with only have two features ($x^i \in \mathbb{R}^2$):

$$\hat{y}^i = w_0 + w_1 x_1^i + w_2 x_2^i + w_3 (x_1^i)^2 + w_4 (x_2^i)^2 + w_5 x_1^i x_2^i.$$

- In 340 we saw that we can fit this model using a **change of basis**:

$$X = \begin{bmatrix} 0.2 & 0.3 \\ 1 & 0.5 \\ -0.5 & -0.1 \end{bmatrix} \Rightarrow Z = \begin{bmatrix} 1 & 0.2 & 0.3 & (0.2)^2 & (0.3)^2 & 0.2 \cdot 0.3 \\ 1 & 1 & 0.5 & (1)^2 & (0.5)^2 & 1 \cdot 0.5 \\ 1 & -0.5 & -0.1 & (-0.5)^2 & (-0.1)^2 & -0.5 \cdot -0.1 \end{bmatrix}$$

- If you have $d = 100$ and $p = 5$, there are $O(100^5)$ **possible degree-5 terms**:

$$(x_1^i)^5, (x_1^i)^4 x_2^i, (x_1^i)^4 x_3^i, \dots, (x_1^i)^3 (x_2^i)^2, (x_1^i)^3 (x_3^i)^2, \dots, (x_1^i)^3 x_2^i x_3^i, \dots$$

- How can we do this when number of features k in basis is huge?**

The "Other" Normal Equations

- Recall the L2-regularized least squares model with basis Z ,

$$\operatorname{argmin}_{v \in \mathbb{R}^d} \frac{1}{2} \|Zv - y\|^2 + \frac{\lambda}{2} \|v\|^2.$$

- By solving for $\nabla f(v) = 0$ we get that

$$v = \underbrace{(Z^\top Z + \lambda I_d)}_{k \text{ by } k}^{-1} Z^\top y,$$

where I_d is the k by k identity matrix.

- An **equivalent way to write the solution is:**

$$v = Z^\top \underbrace{(ZZ^\top + \lambda I_n)}_{n \text{ by } n}^{-1} y,$$

by using a variant of the **matrix inversion lemma** (bonus slide).

- Computing v with this formula is **faster if $n \ll k$:**
 - ZZ^\top is n by n while $Z^\top Z$ is k by k .

Equivalent Form of Ridge Regression

Note that \hat{X} and Y are the same on the left and right side, so we only need to show that

$$(X^T X + \lambda I)^{-1} X^T = X^T (X X^T + \lambda I)^{-1}. \quad (1)$$

A version of the matrix inversion lemma (Equation 4.107 in MLAPP) is

$$(E - FH^{-1}G)^{-1}FH^{-1} = E^{-1}F(H - GE^{-1}F)^{-1}.$$

Since matrix addition is commutative and multiplying by the identity matrix does nothing, we can re-write the left side of (1) as

$$(X^T X + \lambda I)^{-1} X^T = (\lambda I + X^T X)^{-1} X^T = (\lambda I + X^T I X)^{-1} X^T = (\lambda I - X^T (-I) X)^{-1} X^T = -(\lambda I - X^T (-I) X)^{-1} X^T (-I)$$

Now apply the matrix inversion with $E = \lambda I$ (so $E^{-1} = (\frac{1}{\lambda}) I$), $F = X^T$, $H = -I$ (so $H^{-1} = -I$ too), and $G = X$:

$$-(\lambda I - X^T (-I) X)^{-1} X^T (-I) = -\left(\frac{1}{\lambda}\right) I X^T (-I - X \left(\frac{1}{\lambda}\right) X^T)^{-1}.$$

Now use that $(1/\alpha)A^{-1} = (\alpha A)^{-1}$, to push the $(-1/\lambda)$ inside the sum as $-\lambda$,

$$-\left(\frac{1}{\lambda}\right) I X^T (-I - X \left(\frac{1}{\lambda}\right) X^T)^{-1} = X^T (\lambda I + X X^T)^{-1} = X^T (X X^T + \lambda I)^{-1}.$$

Predictions using Equivalent Form

- Given test data \tilde{X} , we predict \hat{y} using:

$$\begin{aligned}\hat{y} &= \tilde{Z}v \\ &= \tilde{Z} \underbrace{Z^\top (ZZ^\top + \lambda I_n)^{-1} y}_{\text{"other" normal equations}}\end{aligned}$$

- If we define $K = ZZ^\top$ (**Gram matrix**) and $\tilde{K} = \tilde{Z}Z^\top$, then we have

$$\hat{y} = \tilde{K}(K + \lambda I_n)^{-1}y,$$

where K is $n \times n$ and \tilde{K} is $t \times n$.

- Key observation behind **kernel trick**:
 - If we can directly compute K and \tilde{K} , we don't need to form Z or \tilde{Z} .

Gram Matrix

- The **Gram matrix** K is defined by:

$$\begin{aligned}
 K = ZZ^\top &= \begin{bmatrix} \text{---} & (z^1)^\top & \text{---} \\ \text{---} & (z^2)^\top & \text{---} \\ & \vdots & \\ \text{---} & (z^n)^\top & \text{---} \end{bmatrix} \begin{bmatrix} | & | & | & \cdots & | \\ z^1 & z^2 & z^3 & \cdots & z^n \\ | & | & | & \cdots & | \end{bmatrix} \\
 &= \begin{bmatrix} \langle z^1, z^1 \rangle & \langle z^1, z^2 \rangle & \cdots & \langle z^1, z^n \rangle \\ \langle z^2, z^1 \rangle & \langle z^2, z^2 \rangle & \cdots & \langle z^2, z^n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle z^n, z^1 \rangle & \langle z^n, z^2 \rangle & \cdots & \langle z^n, z^n \rangle \end{bmatrix} = \begin{bmatrix} k(x^1, x^1) & k(x^1, x^2) & \cdots & k(x^1, x^n) \\ k(x^2, x^1) & k(x^2, x^2) & \cdots & k(x^2, x^n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x^n, x^1) & k(x^n, x^2) & \cdots & k(x^n, x^n) \end{bmatrix}
 \end{aligned}$$

- K contains the **inner products** between all training examples in basis z
- \tilde{K} contains the **inner products** between training and test examples.
 - Kernel trick**: if we can compute $k(x^i, x^j) = \langle z^i, z^j \rangle$, we **don't need** z^i and z^j .

Polynomial Kernel

- In 340 we saw the **polynomial kernel** of degree p ,

$$k(x^i, x^j) = (1 + \langle x^i, x^j \rangle)^p,$$

which corresponds to a **general degree- p polynomial** z^i .

- You can make predictions with these z^i using

$$\hat{y} = \tilde{K}(K + \lambda I)^{-1}y.$$

- **Total cost is only $O(n^2d + n^3)$** even though number of features is $O(d^p)$.

- **Kernel trick:**

- We have kernel function $k(x^i, x^j)$ that gives $\langle z^i, z^j \rangle$.
- **Skip forming Z and directly form K and \tilde{K} .**
- Size of K is n by n even if Z has exponential or infinite columns.

Gaussian-RBF Kernels

- The most common kernel is the **Gaussian-RBF** (or 'squared exponential') kernel,

$$k(x^i, x^j) = \exp\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right).$$

- What features z_i would lead to this as the inner-product?
 - To simplify, assume $d = 1$ and $\sigma = 1$,

$$k(x^i, x^j) = \exp\left(-\frac{1}{2}(x^i)^2 + x^i x^j - \frac{1}{2}(x^j)^2\right) = \exp\left(-\frac{1}{2}(x^i)^2\right) \exp(x^i x^j) \exp\left(-\frac{1}{2}(x^j)^2\right),$$

so we need $z_i = \exp(-\frac{1}{2}(x^i)^2)u_i$ where $u_i u_j = \exp(x^i x^j)$.

- For this to work for *all* x^i and x^j , z_i must be **infinite-dimensional**.
- If we use that

$$\exp(x^i x^j) = \sum_{k=0}^{\infty} \frac{(x^i)^k (x^j)^k}{k!},$$

then we obtain

$$z_i = \exp\left(-\frac{1}{2}(x^i)^2\right) \left[1 \quad \frac{1}{\sqrt{1!}}x^i \quad \frac{1}{\sqrt{2!}}(x^i)^2 \quad \frac{1}{\sqrt{3!}}(x^i)^3 \quad \dots \right].$$

Kernel Trick for Structured Data

- Kernel trick can be useful for **structured data**:
 - Consider data doesn't look like this:

$$X = \begin{bmatrix} 0.5377 & 0.3188 & 3.5784 \\ 1.8339 & -1.3077 & 2.7694 \\ -2.2588 & -0.4336 & -1.3499 \\ 0.8622 & 0.3426 & 3.0349 \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix},$$

but instead looks like this:

$$X = \begin{bmatrix} \text{Do you want to go for a drink sometime?} \\ \text{J'achète du pain tous les jours.} \\ \text{Fais ce que tu veux.} \\ \text{There are inner products between sentences?} \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.$$

- It might be **easier to define a “similarity”** between sentences than to define features.

Kernel Trick for Structured Data

- A classic “string kernel”:
 - We want to compute $k(\text{“cat”}, \text{“cart”})$.
 - Find common subsequences: ‘c’, ‘a’, ‘t’, ‘ca’, ‘at’, ‘ct’, ‘cat’.
 - Weight them by total length in original strings:
 - ‘c’ is has lengths (1,1), ‘ca’ has lengths (2,2), ‘ct’ has lengths (3,4), and son.
 - Add up the **weighted lengths of common subsequences** to get a similarity:

$$k(\text{“cat”}, \text{“cart”}) = \underbrace{\gamma^1 \gamma^1}_{\text{‘c’}} + \underbrace{\gamma^1 \gamma^1}_{\text{‘a’}} + \underbrace{\gamma^1 \gamma^1}_{\text{‘t’}} + \underbrace{\gamma^2 \gamma^2}_{\text{‘ca’}} + \underbrace{\gamma^2 \gamma^3}_{\text{‘at’}} + \underbrace{\gamma^3 \gamma^4}_{\text{‘ct’}} + \underbrace{\gamma^3 \gamma^4}_{\text{‘cat’}}$$

where γ is a hyper-parameter controlling influence of length.

- Corresponds to exponential feature set (counts/lengths of all subsequences).
 - But kernel can be computed in linear time by **dynamic programming**.
- Many variations exist. And there are “image kernels”, “graph kernels”, and so on.
 - You can turn **probabilities over examples** (second half of course) into kernels.
 - A survey on the topic is [here](#).

Valid Kernels

- Can we use any function k for our kernel/similarity function $k(x^i, x^j)$?
- We need to have kernel k be an inner product in some space:
 - There exists transformation $z^i = \phi(x^i)$ such that $k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle$.

We can decompose a (continuous or finite-domain) function k into

$$k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle,$$

iff it is symmetric and for any finite $\{x^1, x^2, \dots, x^n\}$ we have $K \succeq 0$.

- For finite domains you can show existence of ϕ using spectral theorem (bonus).
 - The general case is called Mercer's Theorem.

Constructing Feature Space (Finite Domain)

- Why is positive semi-definiteness important?
 - With finite domain we can define K over all points.
 - By symmetry of K it has a spectral decomposition

$$K = U^\top \Lambda U,$$

and $K \succeq 0$ means $\lambda_i \geq 0$ and so we have a real diagonal $\Lambda^{\frac{1}{2}}$.

- Thus we have $K = U^\top \Lambda^{\frac{1}{2}} \Lambda^{\frac{1}{2}} U = (\Lambda^{\frac{1}{2}} U)^\top (\Lambda^{\frac{1}{2}} U)$ and we could use

$$Z = \Lambda^{\frac{1}{2}} U, \text{ which means } z_i = \Lambda^{\frac{1}{2}} U_{:,i}.$$

- The above reasoning isn't quite right for continuous domains.
- The more careful generalization is known as "Mercer's theorem".

Valid Kernels

- Mercer's Theorem is nice in theory, what do we do in practice?
 - You could show explicitly that $k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle$ for some function ϕ .
 - You could show that K is positive semi-definite by construction.
 - Or you can show k is **constructed from other valid kernels**.

(If we use invalid kernel, lose feature-space interpretation but may work fine.)

Constructing Valid Kernels

- If $k_1(x^i, x^j)$ and $k_2(x^i, x^j)$ are valid kernels, then the following are **valid kernels**:
 - **Non-negative scaling**: $\alpha k_1(x^i, x^j)$ for $\alpha \geq 0$.
 - **Sum**: $k_1(x^i, x^j) + k_2(x^i, x^j)$.
 - **Product**: $k_1(x^i, x^j)k_2(x^i, x^j)$.
 - Special case: $\phi(x^i)k_1(x^i, x^j)\phi(x^j)$ for any function ϕ .
 - **Exponentiation**: $\exp(k_1(x^i, x^j))$.
 - **Recursion**: $k_1(\phi(x^i), \phi(x^j))$ for any function ϕ .
- Example: Gaussian-RBF kernel:

$$\begin{aligned}
 k(x^i, x^j) &= \exp\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\|x^i\|^2}{2\sigma^2} + \frac{1}{\sigma^2}\langle x^i, x^j \rangle - \frac{1}{2\sigma^2}\|x^j\|^2\right) \\
 &= \underbrace{\exp\left(-\frac{\|x^i\|^2}{2\sigma^2}\right)}_{\phi(x^i)} \underbrace{\exp\left(\frac{1}{\sigma^2}\langle x^i, x^j \rangle\right)}_{\substack{\alpha > 0 \\ \text{valid}}} \underbrace{\exp\left(-\frac{\|x^j\|^2}{2\sigma^2}\right)}_{\phi(x^j)}.
 \end{aligned}$$

Models allowing Kernel Trick

- Besides L2-regularized least squares, when can we apply the kernel trick?
 - **Distance-based** methods from CPSC 340:

$$\begin{aligned}\|z^i - z^j\|^2 &= \langle z^i, z^i \rangle - 2\langle z^i, z^j \rangle + \langle z^j, z^j \rangle \\ &= k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j).\end{aligned}$$

- k -nearest neighbours.
 - Clustering algorithms (k -means, density-based clustering, hierarchical clustering).
 - Distance-based outlier detection (KNN-based, outlier ratio)
 - “Amazon product recommendation”.
 - Multi-dimensional scaling (ISOMAP, t-SNE).
 - Label propagation.
- **L2-regularized linear models** (today).
- **Eigenvalue** methods:
 - Principle component analysis (need trick for centering in high-dimensional space).
 - Canonical correlation analysis.
 - Spectral clustering.

Representer Theorem

- Consider L2-regularized loss only depending on Xw ,

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(Xw) + \frac{\lambda}{2} \|w\|^2.$$

- Setting the gradient equal to zero we get

$$0 = X^\top r + \lambda w,$$

where $r = \nabla f(Aw)$.

- So any solution w^* can be written as a linear combination of features x^i ,

$$w^* = -\frac{1}{\lambda} X^\top r = X^\top v,$$

where $v = \frac{1}{\lambda} r$ (this means we can restrict to w satisfying $w = X^\top v$).

Representer Theorem

- Since we know $w^* = X^\top v$ for some v , let's optimize over v instead of w :

$$\begin{aligned} & \operatorname{argmin}_{w \in \mathbb{R}^d} f(Xw) + \frac{\lambda}{2} \|w\|^2 \\ &= \operatorname{argmin}_{v \in \mathbb{R}^n} f(XX^\top v) + \frac{\lambda}{2} \|X^\top v\|^2 \\ &= \operatorname{argmin}_{v \in \mathbb{R}^n} f(XX^\top v) + \frac{\lambda}{2} v^\top XX^\top v \\ &\equiv \operatorname{argmin}_{v \in \mathbb{R}^n} f(Kv) + \frac{\lambda}{2} v^\top K v. \end{aligned}$$

- Which is a kernelized version of the problem.

Representer Theorem

- Using $w = X^\top v$, at test time we use

$$\begin{aligned}\hat{y} &= \tilde{X}w \\ &= \tilde{X}X^\top v \\ &= \tilde{K}v,\end{aligned}$$

or that each $\hat{y}^i = \sum_{j=1}^n v_j k(\tilde{x}^i, x^j)$.

- That prediction is a linear combination of kernels is called representer theorem.
 - It holds under more general conditions, including non-smooth f_i like SVMs.

Multiple Kernel Learning

- We can **kernelize L2-regularized linear models**,

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(Xw, y) + \frac{\lambda}{2} \|w\|^2 \Leftrightarrow \operatorname{argmin}_{v \in \mathbb{R}^n} f(Kv, y) + \frac{\lambda}{2} \|v\|_K^2,$$

under fairly general conditions.

- What if we have **multiple potential kernels** and don't know which to use?
 - Obvious approach: cross-validation to choose the best one.
- What if we have **multiple potentially-relevant kernels**?
 - Multiple kernel learning**:

$$\operatorname{argmin}_{v_1 \in \mathbb{R}^n, v_2 \in \mathbb{R}^n, \dots, v_k \in \mathbb{R}^n} f \left(\sum_{c=1}^k K_c v_c, y \right) + \frac{1}{2} \sum_{c=1}^k \lambda_c \|v\|_{K_c}.$$

- Defines a **valid kernel** and is convex if f is convex (affine function).
- Group L1-regularization of parameters associated with each kernel.
 - Selects a **sparse** set of kernels.
- Hierarchical kernel learning**:
 - Use **structured sparsity** to search through exponential number of kernels.

Outline

- 1 Group Sparsity
- 2 Structured Regularization
- 3 Kernel Trick
- 4 Valid Kernels and Representer Theorem
- 5 Large-Scale Kernel Methods**

Large-Scale Kernel Methods

- Let's go back to the basic L2-regularized least squares setting,

$$\hat{y} = \hat{K}(K + \lambda I)^{-1}y.$$

- Obvious drawback of kernel methods: **we can't compute/store K** .
 - It has $O(n^2)$ elements.
- Standard general approaches:
 - ① Kernels with **special structure**.
 - ② **Subsampling** methods.
 - ③ **Explicit feature** construction.

Kernels with Special Structure

- The bottleneck in fitting the model is $O(n^3)$ cost of solving the linear system

$$(K + \lambda I)v = y.$$

- Consider using the “identity” kernel,

$$k(x^i, x^j) = \mathbb{I}[x^i = x^j].$$

- In this case K is diagonal so we can solve linear system in $O(n)$.
- More interesting special K structures that support fast linear algebra:
 - Band-diagonal matrices.
 - Sparse matrices (via conjugate gradient).
 - Diagonal plus low-rank, $D + UV^\top$.
 - Toeplitz matrices.
 - Kronecker product matrices.
 - Fast Gauss transform.

Subsampling Methods

- In **subsampling** methods we only use a subset of the kernels.
- For example, some loss functions have **support vectors**.
 - But this mainly helps at testing time, and some problems have $O(n)$ support vectors.
- **Nystrom approximation** chooses a random and fixed subset of training examples.
 - Many variations exist such as greedily choosing kernels.
- A common variation is the **subset of regressors** approach....

Subsampling Methods

- Consider partitioning our matrices as

$$K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} = [K_1 \quad K_2], \quad \hat{K} = [\hat{K}_1 \quad \hat{K}_2],$$

where K_{11} corresponds to a set of m training examples

- K is m by m , K_1 is n by m .
- In **subset of regressors** we use the approximation

$$K \approx K_1 K_{11}^{-1} K_1^\top, \quad \hat{K} \approx \hat{K}_1 K_{11}^{-1} K_1^\top.$$

- Which for L2-regularized least squares can be shown to give

$$\hat{y} = \hat{K}_1 \underbrace{(K_1^\top K_1 + \lambda K_{11})^{-1} K_1^\top}_{v} y.$$

- Given K_1 and K_{11} , computing v costs $O(m^2 n + m^3)$ which is cheap for small m .

Explicit Feature Construction

- In **explicit feature** methods, we form Z such that $Z^\top Z \approx K$.
 - But where Z has a small number of columns of m .

- We then use our non-kernelized approach with features Z ,

$$w = (Z^\top Z + \lambda I)^{-1}(Z^\top y).$$

- **Random kitchen sinks** approach does this for translation-invariant kernels,

$$k(x^i, x^j) = k(x^i - x^j, 0),$$

by sampling elements of inverse Fourier transform (not obvious).

- In the special case of the Gaussian RBF kernel this gives $Z = \exp(iXR)$.
 - R is a d by m matrix with elements sampled from the Gaussian (same variance).
 - i is $\sqrt{-1}$ and \exp is taken element-wise.

Summary

- **Group L1-regularization** encourages sparsity in variable groups.
- **Structured regularization** encourages more-general patterns in variables.
- **Total-variation** penalizes differences between variables.
- **Structured sparsity** can enforce sparsity hierarchies.
- **Kernel trick**: allows working with “similarity” instead of features.
 - Also allows exponential- or infinite-sized feature spaces.
- **Valid kernels** are typically constructed from other valid kernels.
- **Representer theorem** allows kernel trick for L2-regularized linear models.
- **Fenchel dual** re-writes sum of convex functions with convex conjugates:
 - Dual may have nice structure: differentiable, sparse, coordinate optimization.
- **Large-scale kernel methods** is an active research area.
 - Special K structures, subsampling methods, explicit feature construction.