Model Selection:

Regularization, Cross-Validation, Information Criteria

Fantastic models and ~~where~~ how to find them

From Lectures 4/5

So far we have mostly considered simple regression scenarios

- *linear models can be solved exactly in the least squares sense*
- *linear models can also be alternatively optimized using simple algorithms (e.g., gradient descent)*
- *fancier optimization methods are usually just smarter versions of gradient descent*
- *from an algorithmic standpoint, optimization of nonlinear models is not much different than linear ones (success may vary)*
- *from an algorithmic standpoint, univariate to multivariate does not change matters*

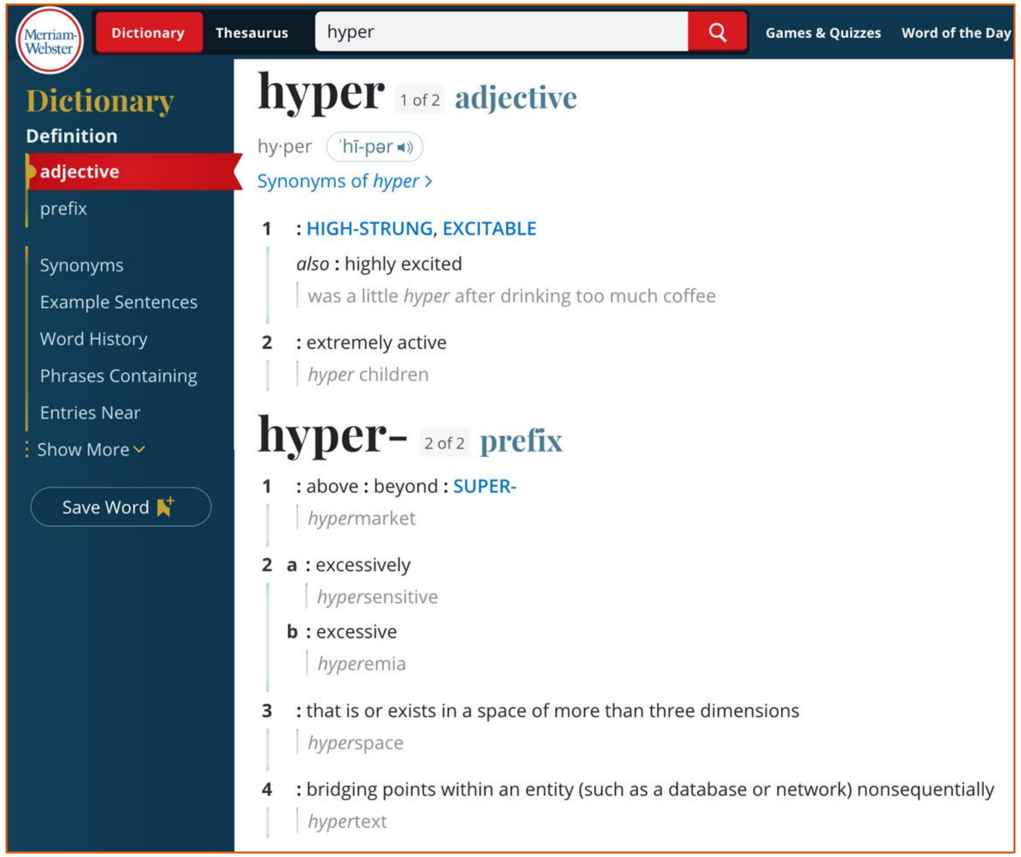
Feature scaling

- *This is a common step in preprocessing data pipelines*
- *Typically, most algorithms work best when they are handling variables of order 1*
- *Having consistent scales on variables may be important to avoid unintentional bias*
- *There are numerous common scaling methods (min-max, standard scaling, non-linear transforms) that you can consider depending on your representation or modeling goals*
- *Implementations are readily found in and facilitated by scikit-learn*

Models revisited

Labeled Data	**Model**	**Predictions**	**Loss**
$\{(\boldsymbol{x}, y)_i\}$	$f(\boldsymbol{x};\boldsymbol{\theta})$	$\hat{y}_k = f(\boldsymbol{x}_k;\boldsymbol{\theta})$	$\mathcal{E}(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}, \ldots)$

optimization allows for selection of model parameters according to the loss


Merriam-Webster | Dictionary | Thesaurus | hyper | Games & Quizzes | Word of the Day

Dictionary
Definition
adjective
prefix
Synonyms
Example Sentences
Word History
Phrases Containing
Entries Near
Show More
Save Word

hyper 1 of 2 *adjective*

hy·per ˈhī-pər

Synonyms of *hyper* ›

1 : HIGH-STRUNG, EXCITABLE
also : highly excited
was a little *hyper* after drinking too much coffee

2 : extremely active
hyper children

hyper- 2 of 2 *prefix*

1 : above : beyond : SUPER-
*hyper*market

2 a : excessively
*hyper*sensitive
b : excessive
*hyper*emia

3 : that is or exists in a space of more than three dimensions
*hyper*space

4 : bridging points within an entity (such as a database or network) nonsequentially
*hyper*text

options that we have/choices that we make *external* to the process of finding model parameters are often referenced as **hyperparameters**

$\boldsymbol{\lambda}$

in our prior examples:
order of our polynomial, the step-size in gradient descent,…

hyperparameter optimization – a process of that should facilitate our selection of such things

Which model is best?

So far… **machine learning ~→ *optimizing models of data without human intervention***

optimization ~→ *minimizing some relevant loss/error metric on example data*

Let's momentarily reconsider optimization of a <u>linear model</u>

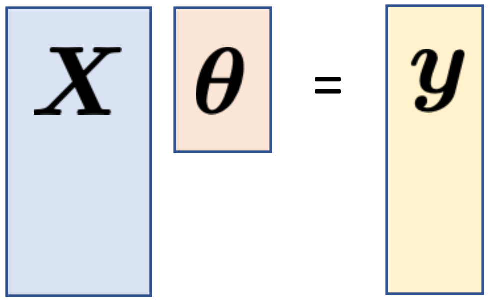
$$\boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{y}$$

(features)(parameters) = (labels)

Recall that there may be either

- **no solution**
- one solution
- **infinite solutions**

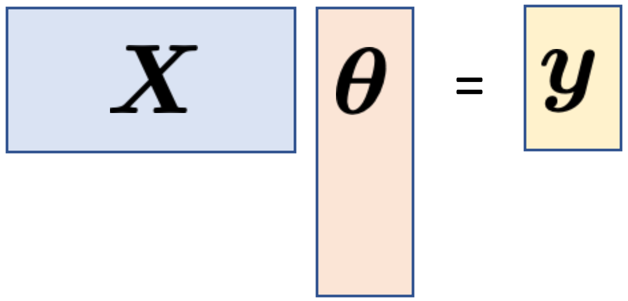
Over-Determined System



we have many more equations (examples) than free parameters.

How do we choose the "best" model in these different scenarios?

Under-Determined System



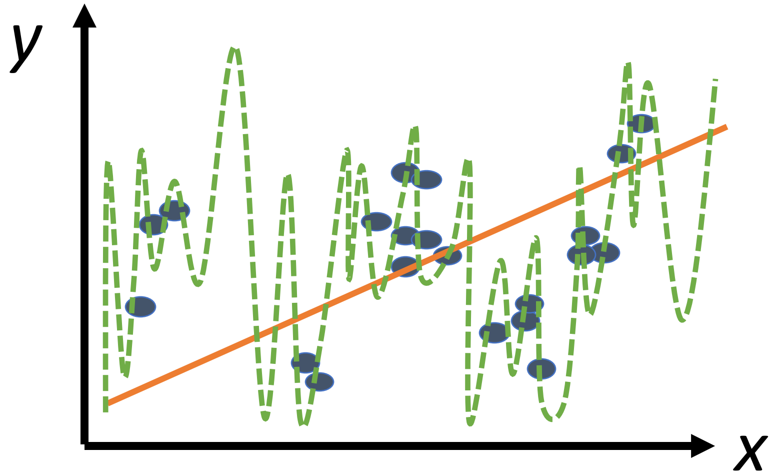
we have more parameters than examples → infinite solution set.

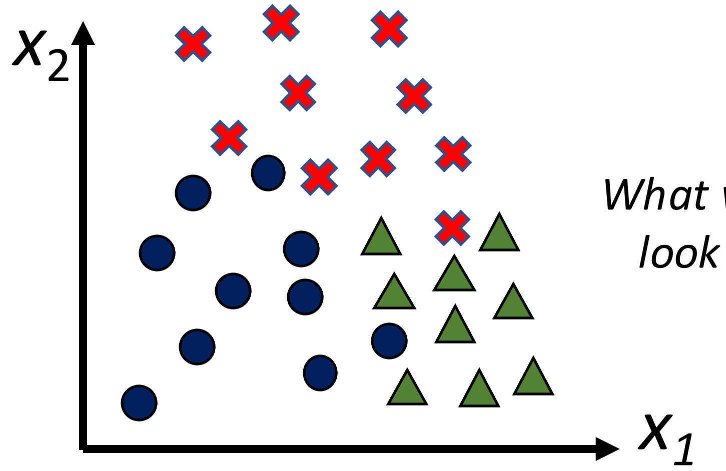
Underfitting vs. Overfitting; Bias and Variance

Without any additional direction, the conditions set forth on the previous slide can lead to some nefarious situations; this is related to **bias-variance** tradeoff

Underfitting arises when our model lacks adequate complexity to describe the underlying system behavior. In such a scenario, the model exhibits high ***bias***
(it will yield consistent future predictions, albeit consistently wrong)

Overfitting arises when we have too many parameters given the complexity of the underlying data; the result here is a model with high ***variance***
(predictions will be inconsistent as model represents noise in training examples)





What would under/overfitting look like for classification?

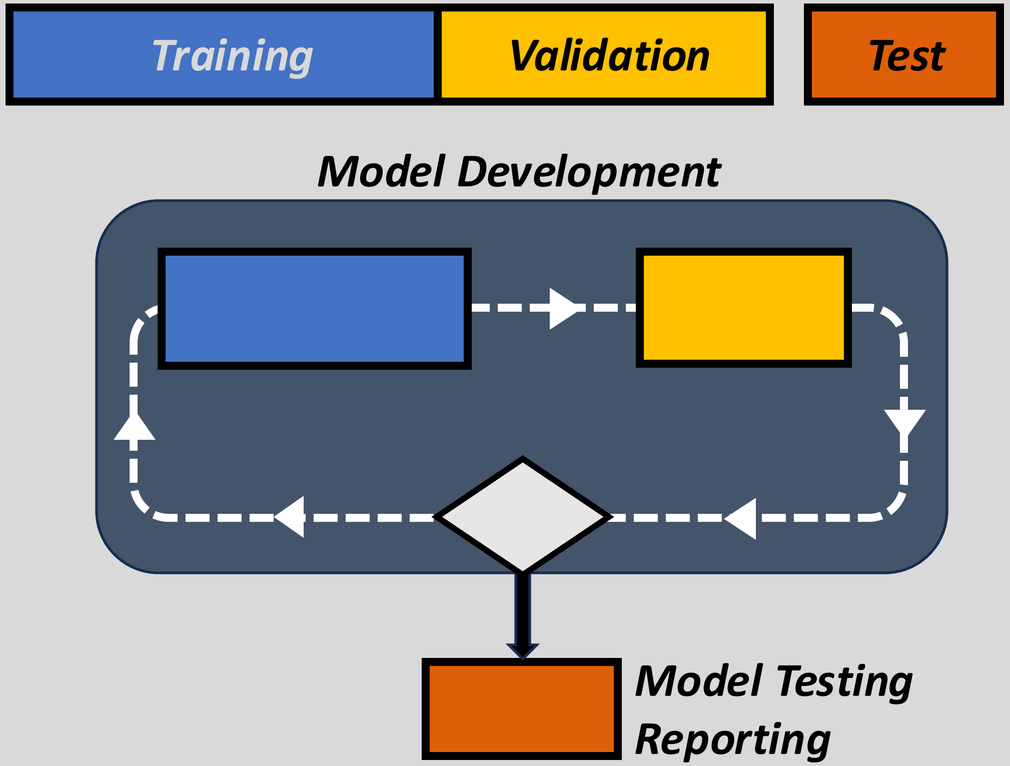
Cross-Validation and Train/Test Splits

A primary goal of many machine learning regression problems is to achieve models that permit ***generalization*** (evaluation outside the domain of our dataset)

Cross-validation and/or the use of **train/test splits** are two common strategies to mitigate overfitting a model to example data/inhibiting generalization

Basic premise of train-(validation)-test split

1. Shuffle dataset *randomly*
2. *Partition* your data into three groups, which we refer to as a **training** set, **validation** set, and a **test** set
3. How do we use these splits?
   - i. **training** –model is allowed to see this data and adjust its parameters accordingly
   - ii. **validation** – model does not adjust its parameters in response to this data but evaluation on the data provides feedback on generalizability for the given model type and mode of optimization
   - iii. **test** – mode neither adjusts its parameters, model type, or mode of optimization – this is data that allows for critical assessment



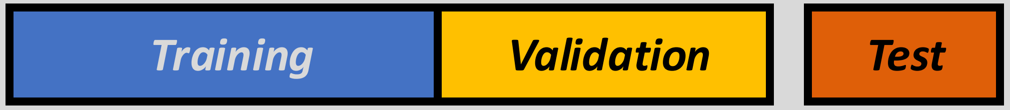
Cross-Validation and Train/Test Splits

A primary goal of many machine learning regression problems is to achieve models that permit ***generalization*** (evaluation outside the domain of our dataset)

Cross-validation and/or the use of **train/test splits** are two common strategies to mitigate overfitting a model to example data/inhibiting generalization

Basic premise of train-(validation)-test split

1. Shuffle dataset *randomly*
2. *Partition* your data into three groups, which we refer to as a **training** set, **validation** set, and a **test** set
3. How do we use these splits?
   i. **training** –model is allowed to see this data and adjust its parameters accordingly
   ii. **validation** – model does not adjust its parameters in response to this data but evaluation on the data provides feedback on generalizability for the given model type and mode of optimization
   iii. **test** – mode neither adjusts its parameters, model type, or mode of optimization – this is data that allows for critical assessment



What is the right ratio of train : validation : test?

There is no single answer. Keep in mind…

- ***how much data do you have?***
  if data is precious, you may need to lean on more training data- without it, the model may just be garbage
- ***how many hyperparameters do you have?***
  you many need more on the validation side to practically guide model selection
- ***what are you using your model for?***
  either large or small test might be appropriate

Cross-Validation and Train/Test Splits

A primary goal of many machine learning regression problems is to achieve models that permit ***generalization*** (evaluation outside the domain of our dataset)

Cross-validation and/or the use of **train/test splits** are two common strategies to mitigate overfitting a model to example data/inhibiting generalization

Basic premise of train-(validation)-test split

1. Shuffle dataset *randomly*
2. *Partition* your data into three groups, which we refer to as a **training** set, **validation** set, and a **test** set
3. How do we use these splits?
   i. **training** –model is allowed to see this data and adjust its parameters accordingly
   ii. **validation** – model does not adjust its parameters in response to this data but evaluation on the data provides feedback on generalizability for the given model type and mode of optimization
   iii. **test** – mode neither adjusts its parameters, model type, or mode of optimization – this is data that allows for critical assessment

Training
Validation
Test

What is the right ratio of train : validation : test?

As a rough guide, some standards are usually

- 50-80% for **training** data
- 10-30% for **validation** data
- 10-30% for **test** data

You will thus see

- We performed an 80/10/10 split
- … 70/20/10
- … 60/20/20
- etc.

Cross-Validation and Train/Test Splits

A primary goal of many machine learning regression problems is to achieve models that permit ***generalization*** (evaluation outside the domain of our dataset)

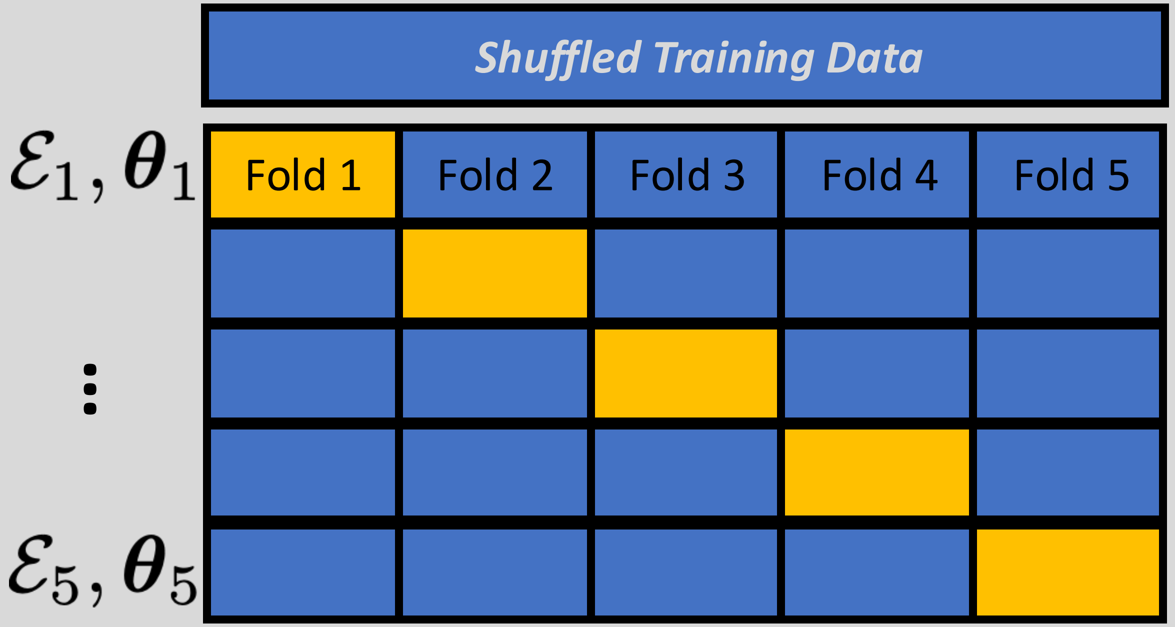
Cross-validation and/or the use of **train/test splits** are two common strategies to mitigate overfitting a model to example data/inhibiting generalization

***cross validation** provides a more structured way to guide model selection*

k-fold Cross-Validation

1. Shuffle dataset randomly
2. Split dataset into *k* groups
3. For each unique group
   - *i. Hold out the group for evaluation as "test" data*
   - *ii. Assign remaining groups as training data*
   - *iii. Optimize a model on the training data*
   - *iv. Evaluate model performance*
   - *v. Store model parameters and evaluation*
4. Summarize model: average evaluations

   … and perhaps parameters



Shuffled Training Data
$\mathcal{E}_1, \boldsymbol{\theta}_1$
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
$\vdots$
$\mathcal{E}_5, \boldsymbol{\theta}_5$

$$\boldsymbol{\theta}^* = \frac{1}{k}\sum_{i=1}^{k}\boldsymbol{\theta}_i$$

or maybe

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}_i}{\operatorname{argmin}}\, \mathcal{E}_i(\boldsymbol{\theta}_i)$$

Cross-Validation and Train/Test Splits

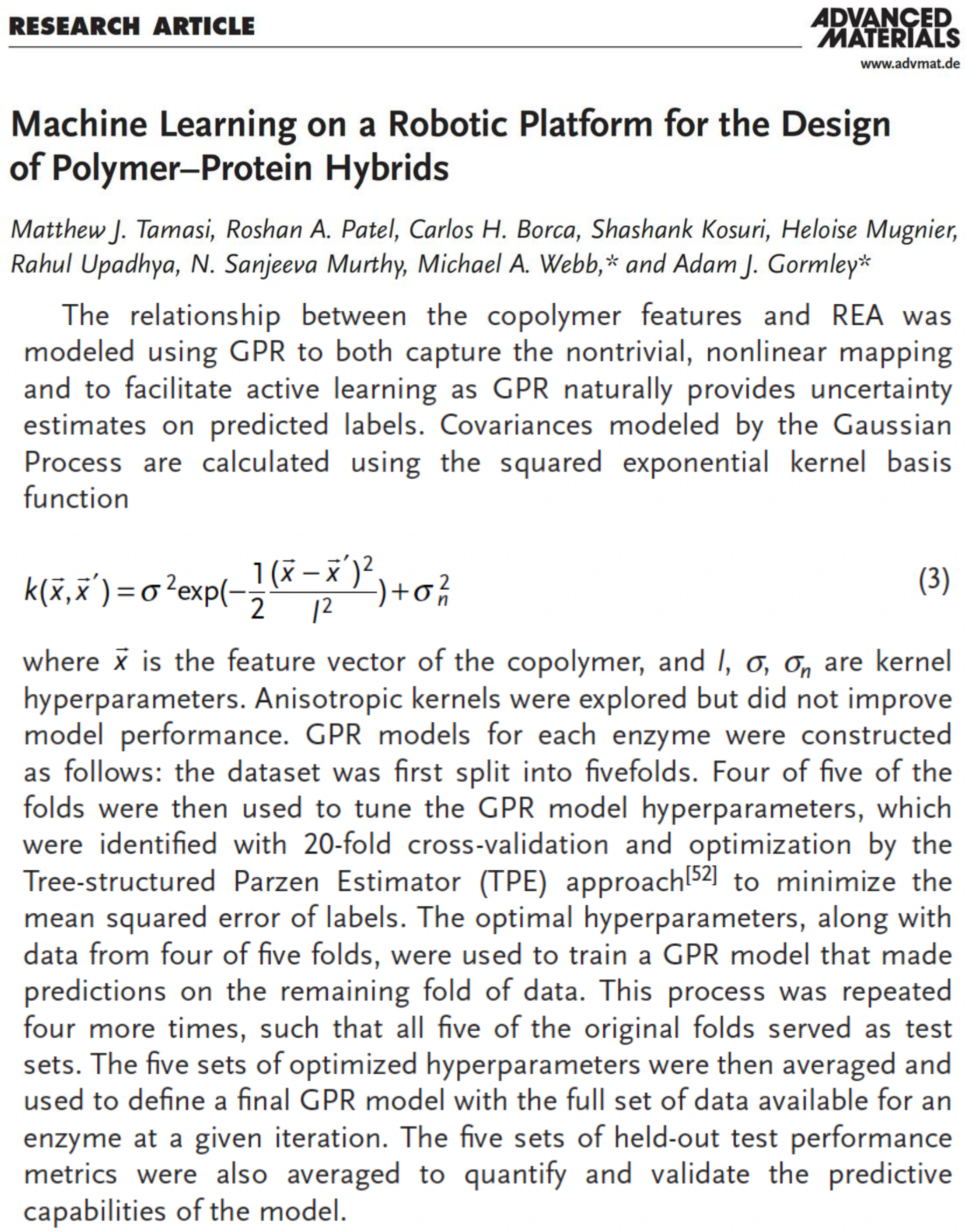
A primary goal of many machine learning regression problems is to achieve models that permit ***generalization*** (evaluation outside the domain of our dataset)

Cross-validation and/or the use of **train/test splits** are two common strategies to mitigate overfitting a model to example data/inhibiting generalization

Some Flavors of Cross-Validation

- ***k-fold*** – (split data into k groups, test on each one while training on the rest)
  - *How big should folds be?*
- ***Leave p Out (LpOCV)*** – remove *p* examples, train on balance, test on *p*, repeat until all examples within training data have been used in "tests"
  - *Common in small-data scenarios*
- ***Stratified k-fold*** – perturbation on k-fold that attempts to preserve proportional representation of *something* across all folds
- ***Repeated*** or ***Nested*** – Repeat the procedure with different shuffles or perform cross-validation within each fold of cross-validation

Example of Nested k-fold for model selection



RESEARCH ARTICLE

ADVANCED MATERIALS
www.advmat.de

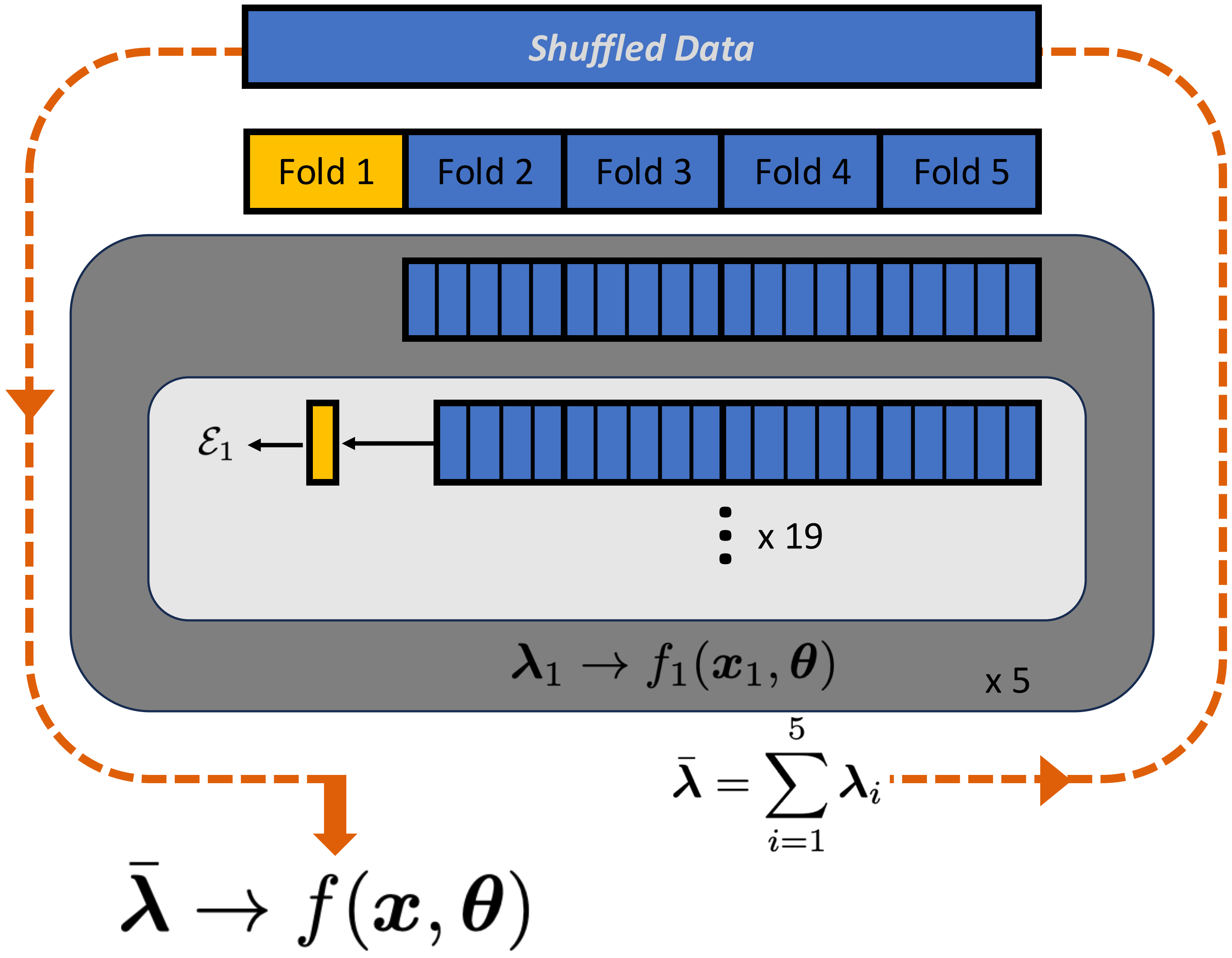
Machine Learning on a Robotic Platform for the Design of Polymer–Protein Hybrids

Matthew J. Tamasi, Roshan A. Patel, Carlos H. Borca, Shashank Kosuri, Heloise Mugnier, Rahul Upadhya, N. Sanjeeva Murthy, Michael A. Webb, and Adam J. Gormley**

The relationship between the copolymer features and REA was modeled using GPR to both capture the nontrivial, nonlinear mapping and to facilitate active learning as GPR naturally provides uncertainty estimates on predicted labels. Covariances modeled by the Gaussian Process are calculated using the squared exponential kernel basis function

$$k(\vec{x},\vec{x}\,') = \sigma^2 \exp\left(-\frac{1}{2}\frac{(\vec{x}-\vec{x}\,')^2}{l^2}\right) + \sigma_n^2 \qquad (3)$$

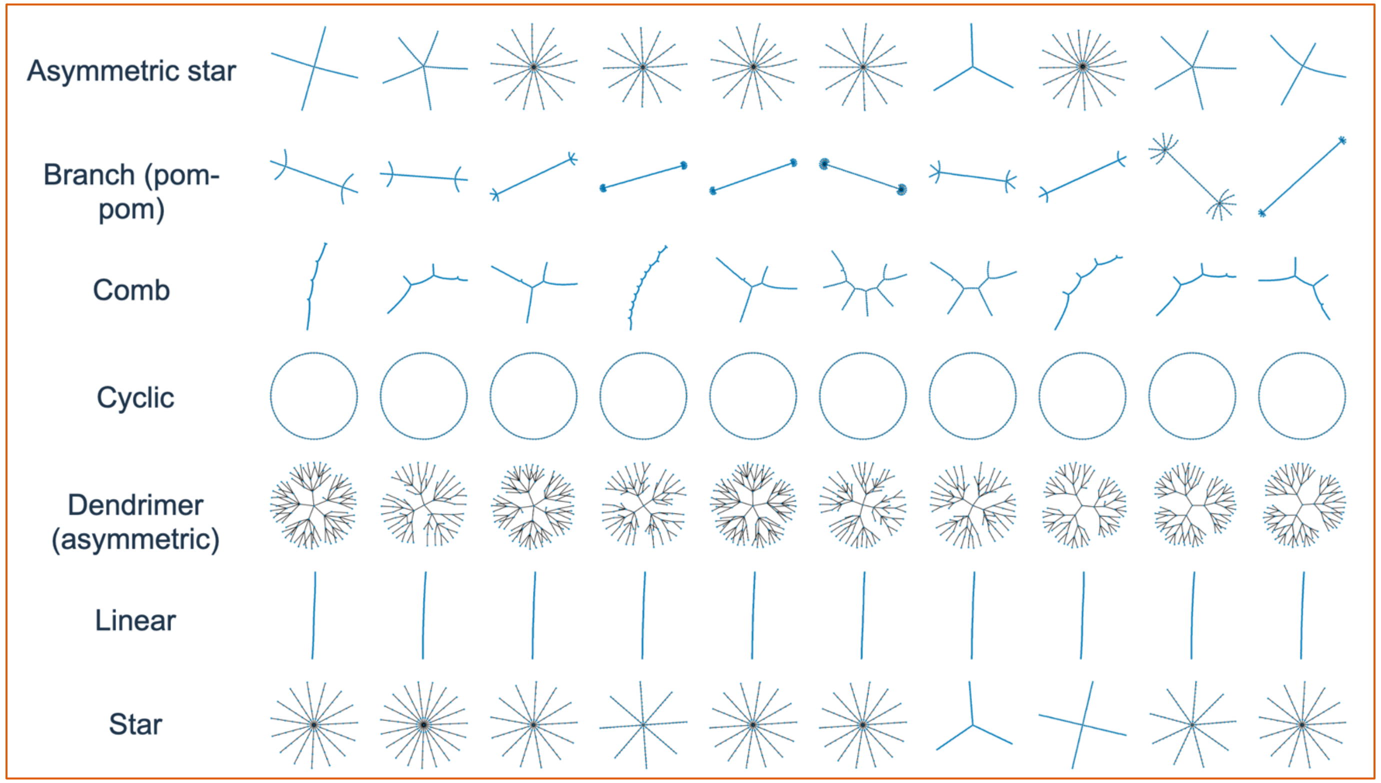
where $\vec{x}$ is the feature vector of the copolymer, and l, σ, σ_n are kernel hyperparameters. Anisotropic kernels were explored but did not improve model performance. GPR models for each enzyme were constructed as follows: the dataset was first split into fivefolds. Four of five of the folds were then used to tune the GPR model hyperparameters, which were identified with 20-fold cross-validation and optimization by the Tree-structured Parzen Estimator (TPE) approach[52] to minimize the mean squared error of labels. The optimal hyperparameters, along with data from four of five folds, were used to train a GPR model that made predictions on the remaining fold of data. This process was repeated four more times, such that all five of the original folds served as test sets. The five sets of optimized hyperparameters were then averaged and used to define a final GPR model with the full set of data available for an enzyme at a given iteration. The five sets of held-out test performance metrics were also averaged to quantify and validate the predictive capabilities of the model.

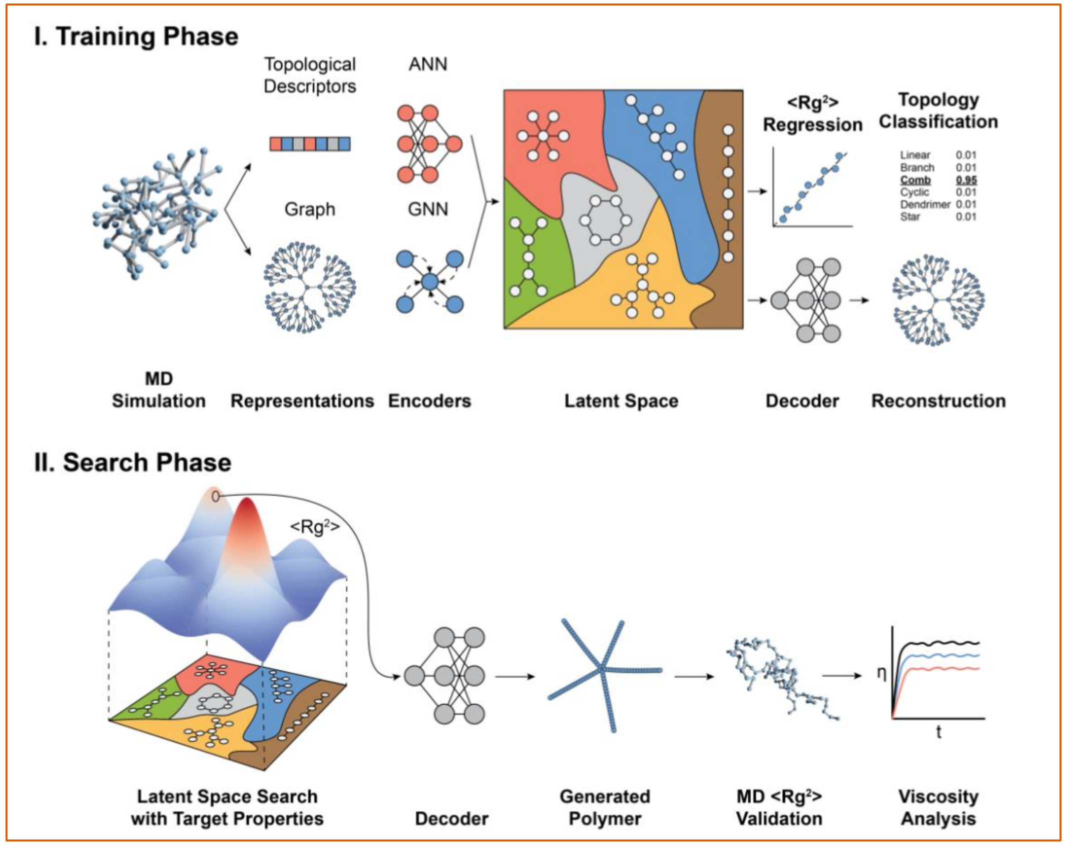


Shuffled Data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
$\mathcal{E}_1$
x 19
$\boldsymbol{\lambda}_1 \rightarrow f_1(\boldsymbol{x}_1, \boldsymbol{\theta})$
x 5
$\bar{\boldsymbol{\lambda}} = \sum_{i=1}^{5} \boldsymbol{\lambda}_i$
$\bar{\boldsymbol{\lambda}} \rightarrow f(\boldsymbol{x}, \boldsymbol{\theta})$

Example of Stratified Selection

polymer architectures with mass constraints





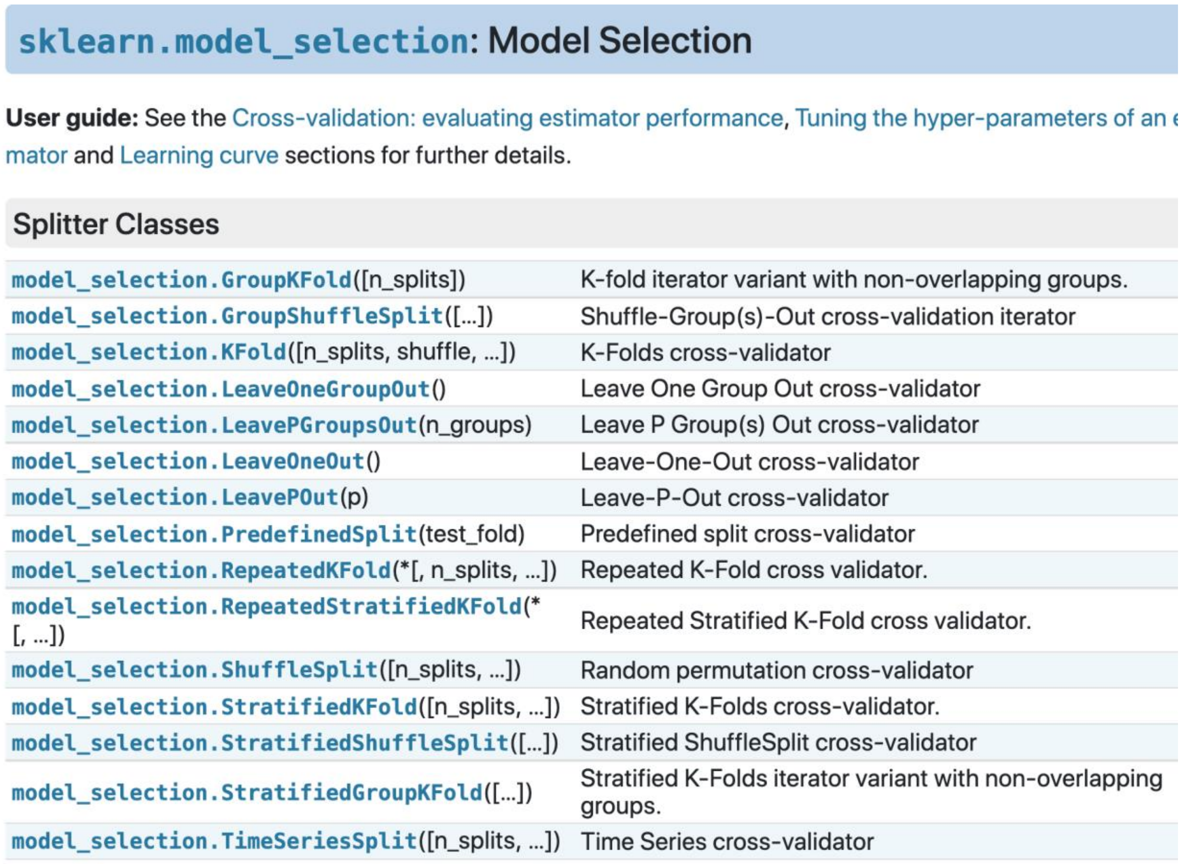
Topology	Number
Linear	11
Cyclic	11
Star/asymmetric star	24/306
Branch (pom-pom)	330
Comb	330
Dendrimer	330

significant class imbalance

What is prospective problem with class or representation imbalance?

- This is a source of bias during model training
- Poor representation may lead to poor generalizability
- Large errors may be irrelevant (drop in the bucket)

Cross-Validation and Train/Test Splits

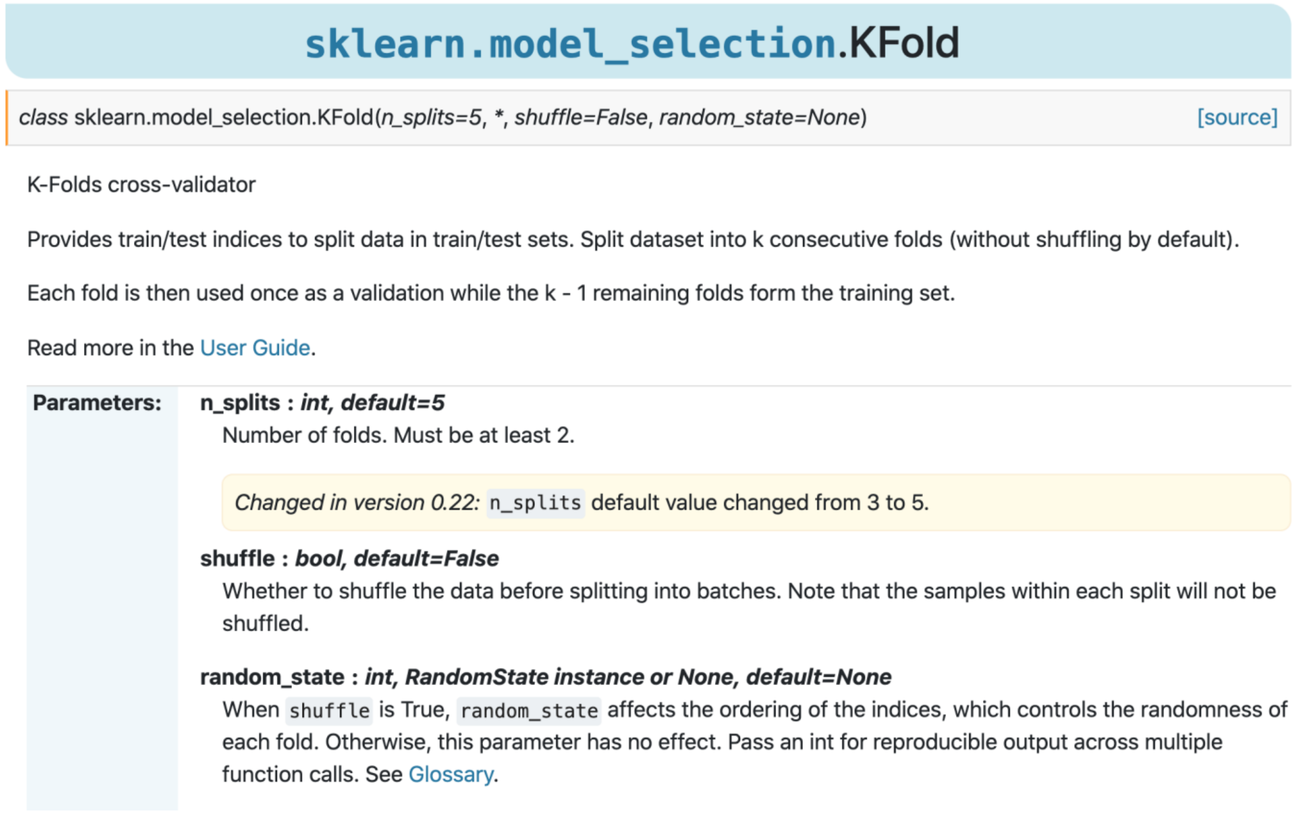


sklearn.model_selection: Model Selection

User guide: See the Cross-validation: evaluating estimator performance, Tuning the hyper-parameters of an e mator and Learning curve sections for further details.

Splitter Classes

`model_selection.GroupKFold`([n_splits])	K-fold iterator variant with non-overlapping groups.
`model_selection.GroupShuffleSplit`([...])	Shuffle-Group(s)-Out cross-validation iterator
`model_selection.KFold`([n_splits, shuffle, ...])	K-Folds cross-validator
`model_selection.LeaveOneGroupOut`()	Leave One Group Out cross-validator
`model_selection.LeavePGroupsOut`(n_groups)	Leave P Group(s) Out cross-validator
`model_selection.LeaveOneOut`()	Leave-One-Out cross-validator
`model_selection.LeavePOut`(p)	Leave-P-Out cross-validator
`model_selection.PredefinedSplit`(test_fold)	Predefined split cross-validator
`model_selection.RepeatedKFold`(*[, n_splits, ...])	Repeated K-Fold cross validator.
`model_selection.RepeatedStratifiedKFold`(*[, ...])	Repeated Stratified K-Fold cross validator.
`model_selection.ShuffleSplit`([n_splits, ...])	Random permutation cross-validator
`model_selection.StratifiedKFold`([n_splits, ...])	Stratified K-Folds cross-validator.
`model_selection.StratifiedShuffleSplit`([...])	Stratified ShuffleSplit cross-validator
`model_selection.StratifiedGroupKFold`([...])	Stratified K-Folds iterator variant with non-overlapping groups.
`model_selection.TimeSeriesSplit`([n_splits, ...])	Time Series cross-validator



sklearn.model_selection.KFold

class sklearn.model_selection.KFold(*n_splits=5*, ***, *shuffle=False*, *random_state=None*) [source]

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

Parameters:

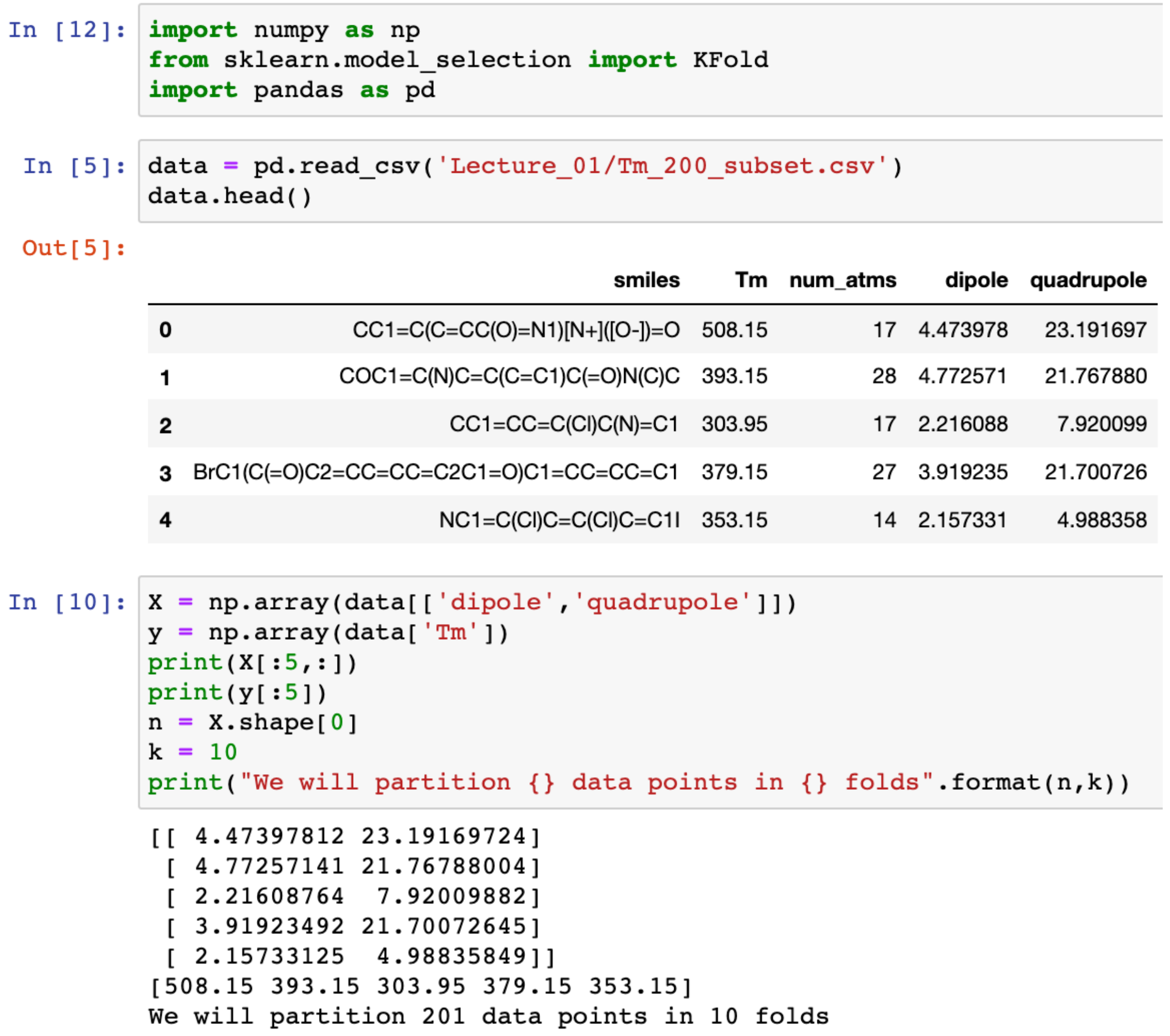
n_splits : *int, default=5*
Number of folds. Must be at least 2.

Changed in version 0.22: `n_splits` default value changed from 3 to 5.

shuffle : *bool, default=False*
Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

random_state : *int, RandomState instance or None, default=None*
When `shuffle` is True, `random_state` affects the ordering of the indices, which controls the randomness of each fold. Otherwise, this parameter has no effect. Pass an int for reproducible output across multiple function calls. See Glossary.

Cross-Validation and Train/Test Splits



```
In [12]: import numpy as np
         from sklearn.model_selection import KFold
         import pandas as pd
```

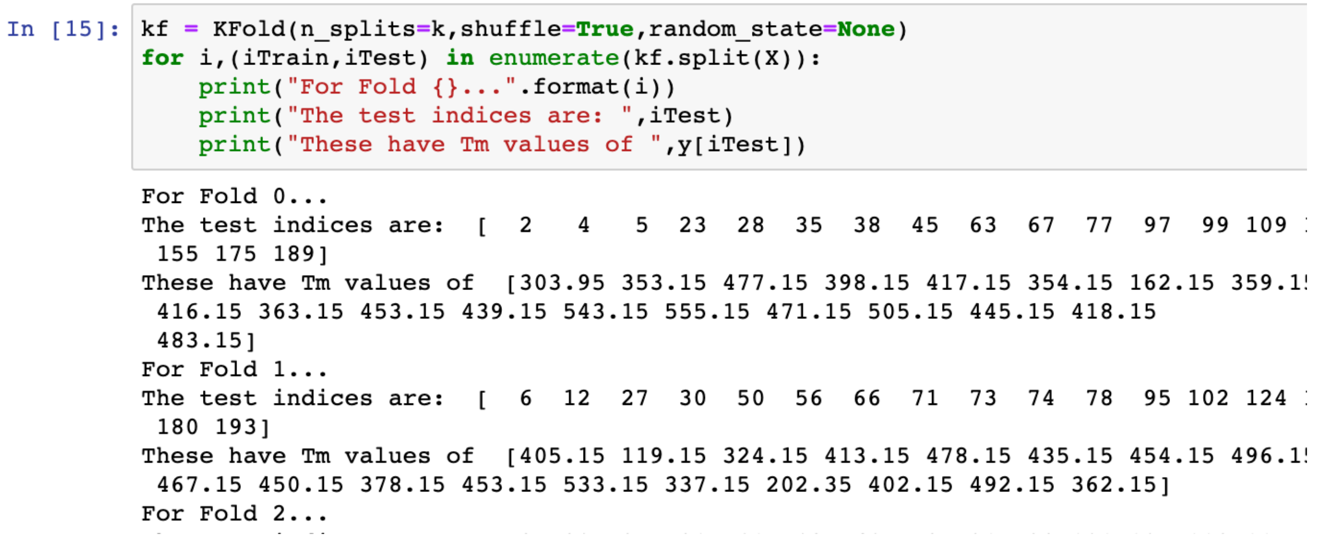
```
In [5]:  data = pd.read_csv('Lecture_01/Tm_200_subset.csv')
         data.head()
```

Out[5]:

	smiles	Tm	num_atms	dipole	quadrupole
0	CC1=C(C=CC(O)=N1)[N+]([O-])=O	508.15	17	4.473978	23.191697
1	COC1=C(N)C=C(C=C1)C(=O)N(C)C	393.15	28	4.772571	21.767880
2	CC1=CC=C(Cl)C(N)=C1	303.95	17	2.216088	7.920099
3	BrC1(C(=O)C2=CC=CC=C2C1=O)C1=CC=CC=C1	379.15	27	3.919235	21.700726
4	NC1=C(Cl)C=C(Cl)C=C1I	353.15	14	2.157331	4.988358

```
In [10]: X = np.array(data[['dipole','quadrupole']])
         y = np.array(data['Tm'])
         print(X[:5,:])
         print(y[:5])
         n = X.shape[0]
         k = 10
         print("We will partition {} data points in {} folds".format(n,k))
```

```
[[ 4.47397812 23.19169724]
 [ 4.77257141 21.76788004]
 [ 2.21608764  7.92009882]
 [ 3.91923492 21.70072645]
 [ 2.15733125  4.98835849]]
[508.15 393.15 303.95 379.15 353.15]
We will partition 201 data points in 10 folds
```



```
In [15]: kf = KFold(n_splits=k,shuffle=True,random_state=None)
         for i,(iTrain,iTest) in enumerate(kf.split(X)):
             print("For Fold {}...".format(i))
             print("The test indices are: ",iTest)
             print("These have Tm values of ",y[iTest])
```

```
For Fold 0...
The test indices are:  [  2   4   5  23  28  35  38  45  63  67  77  97  99 109
 155 175 189]
These have Tm values of  [303.95 353.15 477.15 398.15 417.15 354.15 162.15 359.1
 416.15 363.15 453.15 439.15 543.15 555.15 471.15 505.15 445.15 418.15
 483.15]
For Fold 1...
The test indices are:  [  6  12  27  30  50  56  66  71  73  74  78  95 102 124
 180 193]
These have Tm values of  [405.15 119.15 324.15 413.15 478.15 435.15 454.15 496.1
 467.15 450.15 378.15 453.15 533.15 337.15 202.35 402.15 492.15 362.15]
For Fold 2...
```

Evaluating Model Complexity

In general, the model complexity should be justified by the data

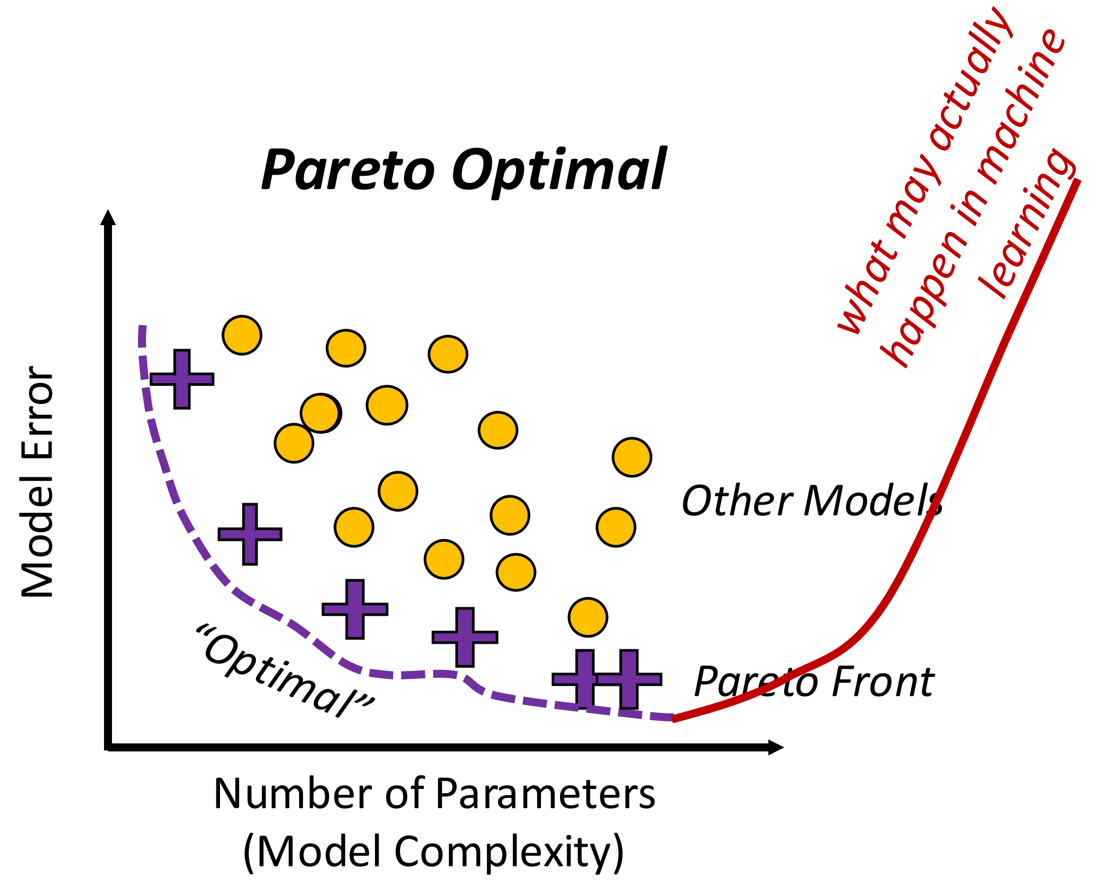
Lex Parsimonae

"Law of Parsimony"

Occam's Razor

"Given two machine learning models of (approximately) equal accuracy, it's better to choose the simpler one."

Pareto Optimal

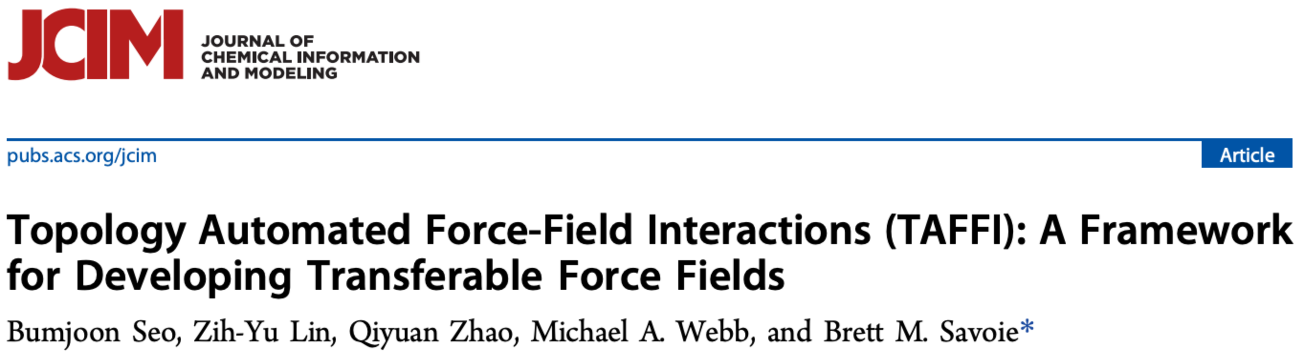


features *can* be treated as a hyperparameter

Too many parameters for too little data can cause problems;
one way to reduce the number of parameters is to reduce the number of features

Regularization

Another way to guide model selection in over/underdetermined problems is to employ ***regularization,*** which adds penalty terms to the loss function based on parameter values

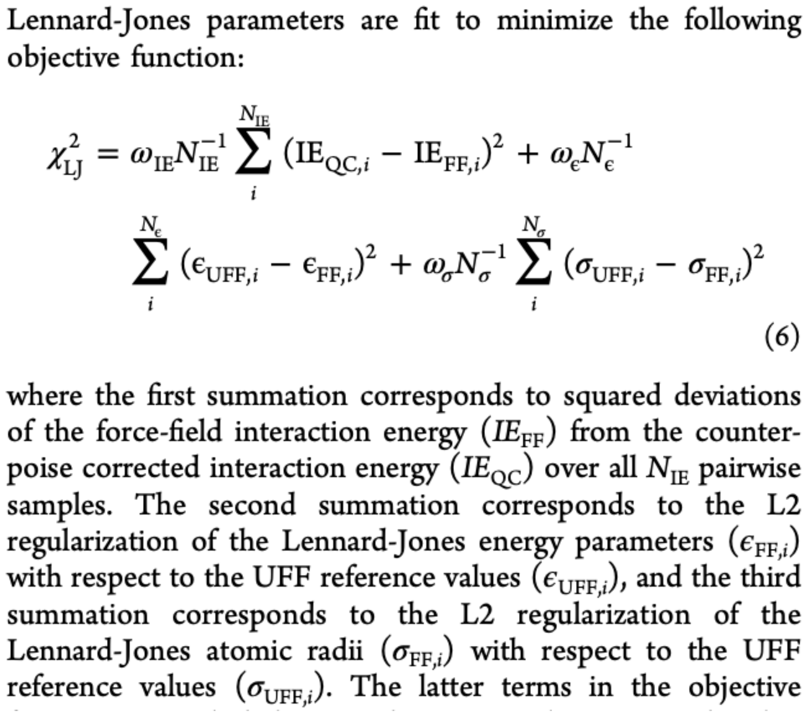


JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim Article

Topology Automated Force-Field Interactions (TAFFI): A Framework for Developing Transferable Force Fields

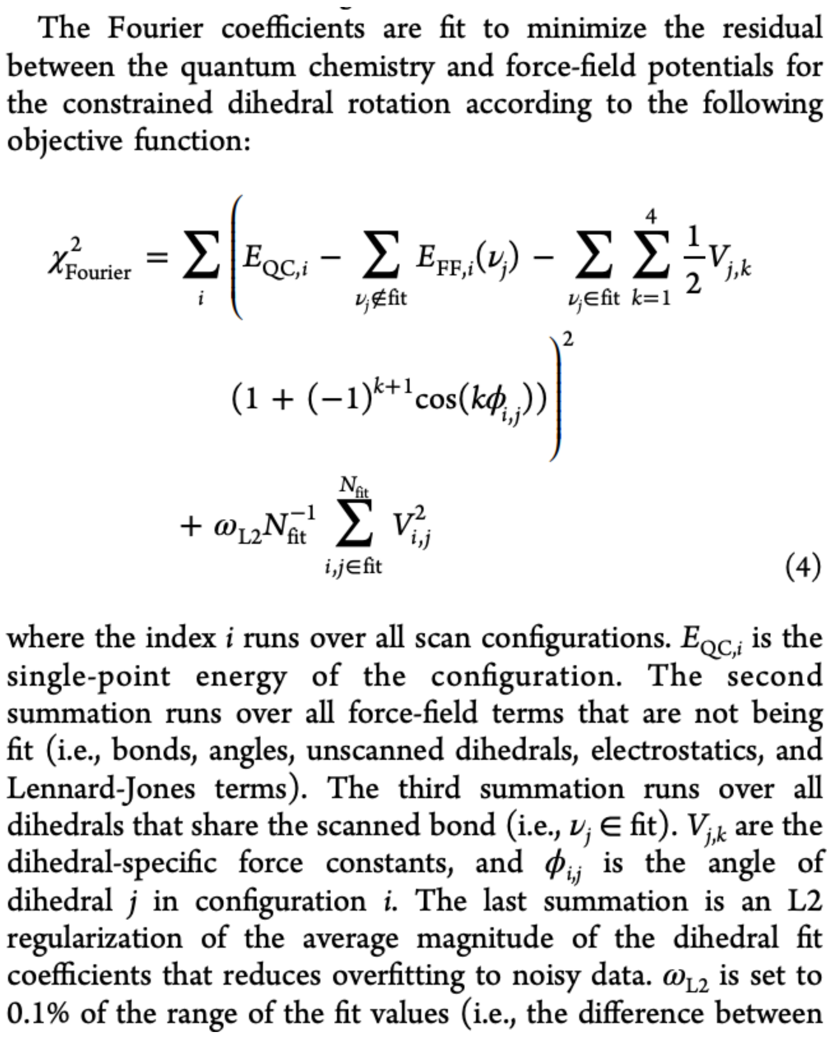
Bumjoon Seo, Zih-Yu Lin, Qiyuan Zhao, Michael A. Webb, and Brett M. Savoie*



Lennard-Jones parameters are fit to minimize the following objective function:

$$\chi^2_{\mathrm{LJ}} = \omega_{\mathrm{IE}} N_{\mathrm{IE}}^{-1} \sum_i^{N_{\mathrm{IE}}} (\mathrm{IE}_{\mathrm{QC},i} - \mathrm{IE}_{\mathrm{FF},i})^2 + \omega_\epsilon N_\epsilon^{-1} \sum_i^{N_\epsilon} (\epsilon_{\mathrm{UFF},i} - \epsilon_{\mathrm{FF},i})^2 + \omega_\sigma N_\sigma^{-1} \sum_i^{N_\sigma} (\sigma_{\mathrm{UFF},i} - \sigma_{\mathrm{FF},i})^2 \tag{6}$$

where the first summation corresponds to squared deviations of the force-field interaction energy (IE_{FF}) from the counterpoise corrected interaction energy (IE_{QC}) over all N_{IE} pairwise samples. The second summation corresponds to the L2 regularization of the Lennard-Jones energy parameters ($\epsilon_{\mathrm{FF},i}$) with respect to the UFF reference values ($\epsilon_{\mathrm{UFF},i}$), and the third summation corresponds to the L2 regularization of the Lennard-Jones atomic radii ($\sigma_{\mathrm{FF},i}$) with respect to the UFF reference values ($\sigma_{\mathrm{UFF},i}$). The latter terms in the objective



The Fourier coefficients are fit to minimize the residual between the quantum chemistry and force-field potentials for the constrained dihedral rotation according to the following objective function:

$$\chi^2_{\mathrm{Fourier}} = \sum_i \left(E_{\mathrm{QC},i} - \sum_{\nu_j \notin \mathrm{fit}} E_{\mathrm{FF},i}(\nu_j) - \sum_{\nu_j \in \mathrm{fit}} \sum_{k=1}^{4} \frac{1}{2} V_{j,k} (1 + (-1)^{k+1} \cos(k\phi_{i,j})) \right)^2 + \omega_{\mathrm{L2}} N_{\mathrm{fit}}^{-1} \sum_{i,j \in \mathrm{fit}}^{N_{\mathrm{fit}}} V_{i,j}^2 \tag{4}$$

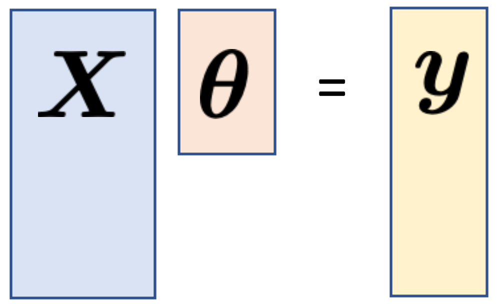
where the index i runs over all scan configurations. $E_{\mathrm{QC},i}$ is the single-point energy of the configuration. The second summation runs over all force-field terms that are not being fit (i.e., bonds, angles, unscanned dihedrals, electrostatics, and Lennard-Jones terms). The third summation runs over all dihedrals that share the scanned bond (i.e., $\nu_j \in$ fit). $V_{j,k}$ are the dihedral-specific force constants, and $\phi_{i,j}$ is the angle of dihedral j in configuration i. The last summation is an L2 regularization of the average magnitude of the dihedral fit coefficients that reduces overfitting to noisy data. ω_{L2} is set to 0.1% of the range of the fit values (i.e., the difference between

usually regularization will penalize parameters from taking on large values or can be used to penalize deviations from reference values

Regularization

Another way to guide model selection in over/underdetermined problems is to employ ***regularization,*** which adds penalty terms to the loss function based on parameter values

Over-Determined System



e.g.,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\hat{\boldsymbol{y}}(\boldsymbol{\theta}) - \boldsymbol{y}\|_2$$

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\hat{\boldsymbol{y}}(\boldsymbol{\theta}) - \boldsymbol{y}\|_2 + \omega_1 \|\boldsymbol{\theta}\|_1 + \omega_2 \|\boldsymbol{\theta}\|_2$$

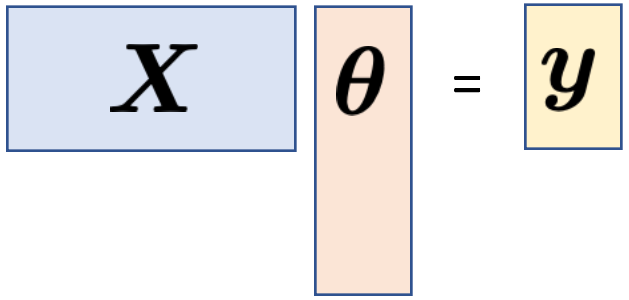
constraint on error *constraints on solution vector*

- constraints on solution vector can be *designed* (there are no rules!)
- nonetheless, penalizing magnitudes via norms is common

Regularization

Another way to guide model selection in over/underdetermined problems is to employ ***regularization,*** which adds penalty terms to the loss function based on parameter values

Under-Determined System



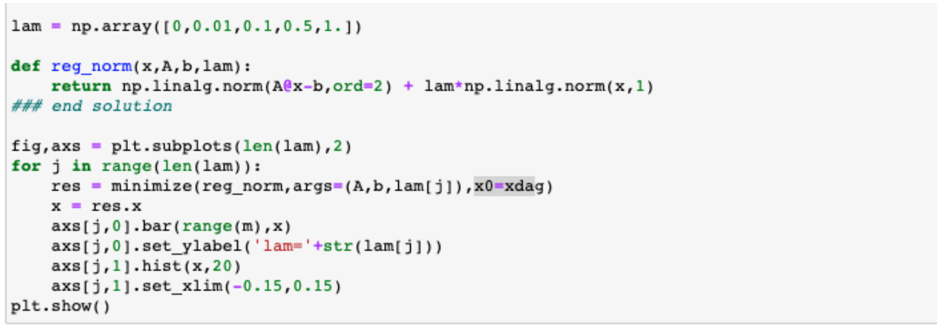
e.g.,

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\|\boldsymbol{\theta}\right\|_p ; \boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{y}$$

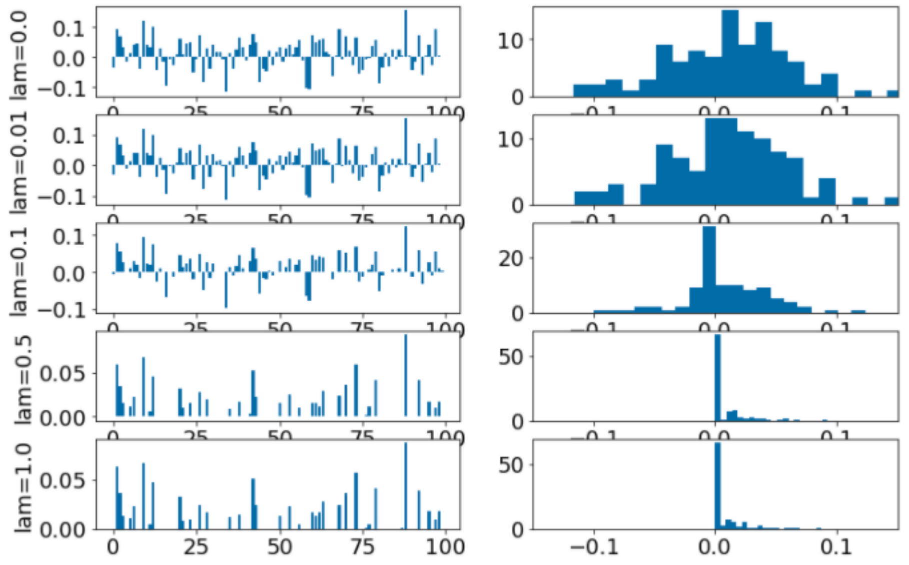
- constraints on solution vector can be *designed* (there are no rules!)
- nonetheless, penalizing magnitudes via norms is common

Regularization in optimization

L1 Regularization promotes solution sparsity

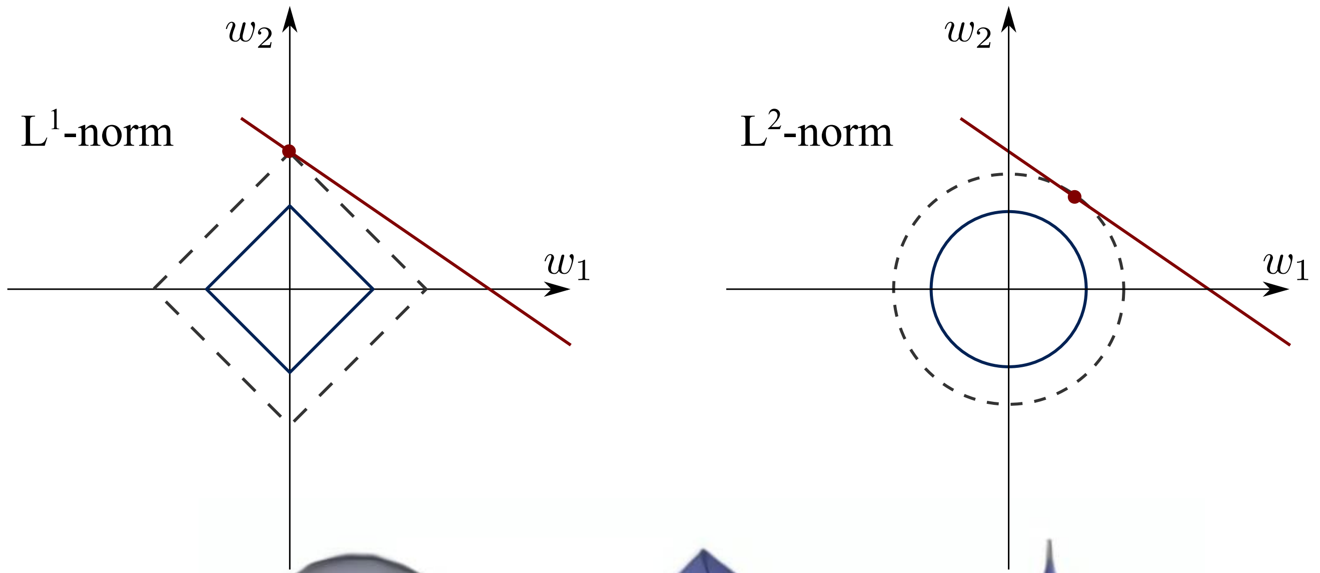


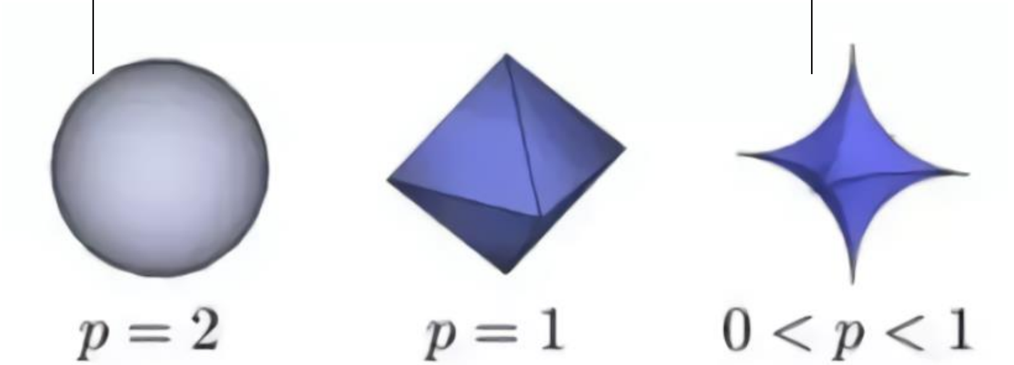
```
lam = np.array([0,0.01,0.1,0.5,1.])

def reg_norm(x,A,b,lam):
    return np.linalg.norm(A@x-b,ord=2) + lam*np.linalg.norm(x,1)
### end solution

fig,axs = plt.subplots(len(lam),2)
for j in range(len(lam)):
    res = minimize(reg_norm,args=(A,b,lam[j]),x0=xdag)
    x = res.x
    axs[j,0].bar(range(m),x)
    axs[j,0].set_ylabel('lam='+str(lam[j]))
    axs[j,1].hist(x,20)
    axs[j,1].set_xlim(-0.15,0.15)
plt.show()
```



This type of regularization is more or less the premise of LASSO (least absolute shrinkage and selection operator) regression

Why? We can understand based on shapes of norm surfaces…





this would be even more sparse but is not as easy to calculate

Information Criteria

A primary goal of many machine learning regression problems is to achieve models that permit ***generalization*** (evaluation outside the domain of our dataset)

Another way to evaluate/discriminate against model complexity is to consider ***Information Criteria;*** typically these will penalize models that are unnecessarily complex given the evidence (data)

Some Flavors of Information Criteria

- ***Kullback-Leibler divergence***

$$I(p,q) = \int p(\boldsymbol{x},\boldsymbol{\theta}) \log\left[\frac{p(\boldsymbol{x},\boldsymbol{\theta})}{q(\boldsymbol{x},\boldsymbol{\theta})}\right] d\boldsymbol{x}$$

does not really address model complexity

- ***Akaike Information Criterion***

$$\mathrm{AIC} = 2m - 2\log\left[\mathcal{L}(\boldsymbol{\theta}^*|\boldsymbol{y})\right]$$

number of parameters

likelihood function

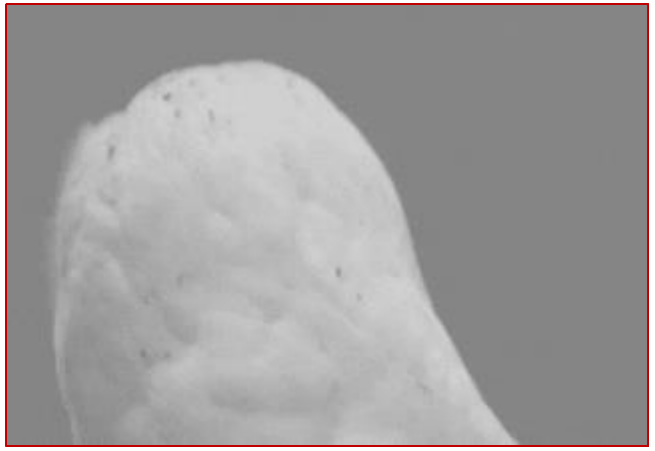
- ***Bayesian Information Criterion***

$$\mathrm{BIC} = m\log(n) - 2\log\left[\mathcal{L}(\boldsymbol{\theta}^*|\boldsymbol{y})\right]$$

number of examples

Least Squares and Maximum Likelihood Estimation

Under a particular set of assumptions… least squares estimation is equivalent to maximum likelihood estimation.



- linear model
- homoscedastic, normal, and independent errors

$$y_i \middle| (x_{i1}, \ldots, x_{ik}) \sim \mathcal{N}(\boldsymbol{x}^T\boldsymbol{\theta}, \sigma^2) \; \forall \; \boldsymbol{x} \in \mathbb{R}^k$$

$$\mathcal{L}(\boldsymbol{\theta}) = \log(\phi(\boldsymbol{y}; \boldsymbol{X}\boldsymbol{\theta}, \sigma^2\boldsymbol{I})) = \sum_{i=1}^{n} \log(\phi(y_i; \boldsymbol{x}_i^T\boldsymbol{\theta}, \sigma^2))$$

$$= -\log((2\pi)^{n/2}\sigma^n) - \frac{1}{2\sigma^2}\underbrace{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})}_{\text{This is MSE}}$$

Note: *to maximize the likelihood, you would differentiate with respect to the parameters → you don't even need to the noise/variance*