Variants of Linear Regression

Statistics and Data Science

Spring 2025

Goals for today: you should be able to...

- **lecture 24/25 notebook:**
  - Perform robust linear regression
- **lecture 25/26 notebook:**
  - Use scikit-learn to apply regression methods with regularization:
    - Ridge regression
    - LASSO
    - SGD
  - Optimizing the alpha parameter with cross-validation
- Introduction to MCMC methods

Reminder: you will need to turn in your notebooks...

- A portion of your grade comes from class activities, graded based upon your Jupyter notebooks (graded for completeness)
  - Upload a zip file containing them, or point me to a git repository I can clone
- I have activated the assignment for this on Canvas
- Due Monday, April 21

Presentations and writeups

- The presentation and writeup should describe the project you have done. Standard format for a paper:
  - Introduction: explaining project goals and why they are relevant/interesting/important: background of the kind of measurements/model you will make
  - Data: explaining the dataset/observations and key quantities employed
  - Methods: explaining how the methods you will apply work
  - Results: presenting the quantities you measured from the data
  - Conclusions: describing what conclusions you draw from the results and what future work is necessary

Structure for presentation & writeup

- Your target audience for presentations is your fellow students. Talks should be 12 minutes/presenter (plus a little time for questions), with five presenters each day (we will need to meet on Friday). Please try to split things up so that each person is providing at least some introductory/general material and some results/conclusions.
- Writeups should ideally be 2-3 pages; 5 pages **MAXIMUM** including text and figures, in 11-point or larger font and manuscript (**not** two-column) format. Note that this is a double-spaced format. You should include references as in any scientific paper, which will not count toward the length limit. Again, it should be written so that it would motivate and describe what you have done to a fellow graduate student, not an expert in the field/your project.
- **Write-ups should be done independently.** Focus your writeup on the portions of the project you led (though you will still need to introduce it, etc.).

Rubric for the presentations

Oral Presentation Assessment Rubric

Exemplary	*Proficient*	*Basic*	*Below Basic*
10-9 points	8-6 points	5-4 points	3-0 points
Content: The content is accurate and complete. The speaker explains all key concepts, theories, and/or data, drawing upon relevant literature. There is evidence of extensive and valid research with multiple and varied sources. **Organization:** The presentation is extremely well organized. The speaker introduces the purpose of the presentation clearly and creatively. S/he presents key points in a logical, interesting sequence, and ends with accurate conclusions and/or ideas for future work, which show thoughtful evaluation of the information and evidence presented. **Sensitivity to audience:** The level of the presentation is appropriate for the audience. Speaker is poised, uses clear articulation, proper and varied volume, and exudes enthusiasm and confidence. Sentences are complete and flow together easily. Speaker maintains eye contact with the audience. Speaker demonstrates extensive knowledge of the topic by encouraging and then fielding audience questions confidently, accurately, and appropriately. The speaker finishes on time. **Visual aids:** The visual aids are stimulating. There are no spelling or grammatical errors. Details are minimized and main points stand out. The layout is very well chosen and engaging. The writing and the color choices make the presentation easy to see. Graphics reinforce the presentation and maximize audience understanding.	**Content:** The content is accurate and extensive. The speaker explains several key concepts, theories, and/or data, drawing upon relevant literature. There is evidence of valid research with multiple sources. **Organization:** The presentation is generally well organized. The speaker introduces the purpose of the presentation. S/he presents key points in a logical sequence, and ends with a summary of the main points showing some evaluation of the information and evidence presented. **Sensitivity to audience:** The level of the presentation is generally appropriate for the audience. The speaker uses clear articulation, and proper volume, yet could have practiced the presentation a little more for greater polish. Sentences are complete and flow together easily. Speaker maintains eye contact with the audience. S/he demonstrates knowledge of the topic by fielding audience questions confidently, accurately, and appropriately. The speaker finishes within minutes of the allotted time. **Visual aids:** The visual aids are appropriate. There are a few spelling or grammatical errors. Details are minimized and main points stand out. The layout is well chosen. The font size and the color choices allow most of the audience to see the visual aids. Graphics are well chosen and relevant to the presentation.	**Content:** The content has minor inaccuracies. The speaker makes minor mistakes in explaining key concepts, theories, and/or data. Enough errors are made to distract a knowledgeable listener, but most information is accurate and there is evidence of valid research with an identifiable source. **Organization:** The presentation is somewhat organized. The speaker introduces the purpose of the presentation. S/he presents several points, but some of them are minor, irrelevant, or confusing. S/he ends with a summary, but the listener has difficulty connecting it to the content or the purpose of the presentation. **Sensitivity to audience:** Portions of the presentation are either too elementary or too sophisticated for the audience. Speaker has difficulty pronouncing some of the technical terms. The audience occasionally has trouble hearing the speaker, or maintaining interest in the presentation. Speaker frequently reads off the visual aids, and does not maintain eye contact with the audience. The speaker seems reluctant to respond to audience questions. The presentation was clearly too short or too long. **Visual aids:** The visual aids contain several spelling or grammatical errors. The layout distracts from the presentation. Some slides are too crowded, or difficult to see for the audience. Some graphics are poorly chosen or displayed and do not support the presentation.	**Content:** The content has several inaccuracies. The speaker makes major mistakes in explaining key concepts, theories, and/or data. The mistakes are evident to the listeners, to the extent that they cannot rely on the presentation as an accurate source of information. Little or no reference is made to the literature; there is no evidence of valid research. **Organization:** The presentation is poorly organized. The speaker does not clearly introduce the purpose of the presentation; it is disjointed with no apparent logical order, and it ends without a summary or conclusions. **Sensitivity to audience:** The presentation is inappropriate for the audience. The presenter cannot be heard or speaks in a monotone voice with little or no expression. S/he mumbles and pronounces technical terms incorrectly. The presenters back is to the audience at all times. The audience looses interest completely. After the presentation ends, there are no questions from the audience. **Visual aids:** The visual aids contain several spelling or grammatical errors. The words on the slides are notes for the speaker, rather than succinct points that support the presentation. The layout distracts from the presentation. Much of what is on the slides cannot be read or seen by the audience. Presenter uses superfluous graphics, no graphics, or graphics that are that are so poorly prepared that they detract from the presentation.

Rubric for the presentations

Presentation Evaluation of: _____

Content: The content is accurate and complete. The speaker explains all key concepts, theories, and/or data, drawing upon relevant literature. There is evidence of extensive and valid research with multiple and varied sources.

1 (poor) … 2 … 3 … 4 … 5 (excellent)

Organization: The presentation is extremely well organized. The speaker introduces the purpose of the presentation clearly and creatively. S/he presents key points in a logical, interesting sequence, and ends with accurate conclusions and/or ideas for future work, which show thoughtful evaluation of the information and evidence presented.

1 … 2 … 3 … 4 … 5

Sensitivity to audience: The level of the presentation is appropriate for the audience. Speaker is poised, uses clear articulation, proper and varied volume, and exudes enthusiasm and confidence. Sentences are complete and flow together easily. Speaker maintains eye contact with the audience. Speaker demonstrates extensive knowledge of the topic by encouraging and then fielding audience questions confidently, accurately, and appropriately. The speaker finishes on time.

1 … 2 … 3 … 4 … 5

Visual aids: The visual aids are stimulating. There are no spelling or grammatical errors. Details are minimized and main points stand out. The layout is very well chosen and engaging. The writing and the color choices make the presentation easy to see. Graphics reinforce the presentation and maximize audience understanding.

1 … 2 … 3 … 4 … 5 **Total:** /20

What did the speaker do particularly well?

_____

Review

- Suppose we have a set of measurements y_i that should be expressible as a function of some other set of variables, $\boldsymbol{x_i}$, with parameters α,β,γ etc. (so the ith y will depend on the *ith* vector of variables $\boldsymbol{x_i}$) with random scatter:

  $$y_i = f(\boldsymbol{x_i};\ \alpha,\beta,\gamma\ldots) + \varepsilon_i$$

  where ε_i is a random variable of mean 0, independent of all the parameters

- If ε_i is Gaussian, $L(data \mid params) = \prod (2\pi\,\sigma_i^2)^{1/2} \exp\left(-\,(y_i - f(\boldsymbol{x_i};\ \alpha,\beta,\gamma\ldots))^2/2\sigma_i^2\right)$
- So maximizing likelihood is minimizing $\chi^2 = \Sigma\,(y_i - f(\boldsymbol{x_i};\ \alpha,\beta,\gamma\ldots))^2\ /\sigma_i^2$
- If the error estimates are all correct and f has the correct functional form the value of χ^2 at the minimum will be drawn from a chi-squared distribution with (number of data - number of parameters) degrees of freedom

Robust regression methods

- The `statsmodels` package provides a variety of options for robust, multi-linear regression. Basic linear regression tools in `statsmodels` are:

`statsmodels.OLS`: ordinary least squares, no weights

`statsmodels.WLS`: least-squares with user-provided (e.g., 1/sigma^2) weights

`statsmodels.GLS`: least-squares with user-provided covariance matrix

`statsmodels.RLM`: robust linear regression with outliers deweighted

- See https://www.statsmodels.org/stable/examples/notebooks/generated/robust_models_1.html for the deweighting options provided (they include trimming and the Tukey biweight that we've seen before).
- Let's try it!!

Note: see `http://en.wikipedia.org/wiki/Robust_regression` for a good discussion of robust techniques.

Robust regression methods

- The `statsmodels` way to do a fit looks a bit different from what we have done before. First, let's see how ordinary regression works in `statsmodels`. We begin by doing imports and setting up an array of our independent variable:

```
import statsmodels.api as sm
X = logv
```

- We need to add a column to fit for a constant too:

```
X = sm.add_constant(X)
```

- Now, we do the fit and look at the results:

```
results = sm.WLS(mu, X, weights=weight).fit()
results.summary()
```

- Note: we could have done this in two lines as:

```
model = sm.WLS(mu, X, weights=weight)
results = model.fit()
```

Robust regression methods

`results.summary()`

- See https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.RegressionResults.html for documentation
- The F test tests compares the variance of residuals about the model to the variance of the values with no model subtracted.
- Durbin-Watson tests for covariance in successive residuals
- Jarque-Bera tests whether residuals have the skewness + kurtosis of a Gaussian.

[31]:

WLS Regression Results

Dep. Variable:	mu	**R-squared:**	0.970
Model:	WLS	**Adj. R-squared:**	0.969
Method:	Least Squares	**F-statistic:**	5161.
Date:	Wed, 21 Apr 2021	**Prob (F-statistic):**	9.05e-125
Time:	00:45:52	**Log-Likelihood:**	50.730
No. Observations:	164	**AIC:**	-97.46
Df Residuals:	162	**BIC:**	-91.26
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	15.7770	0.280	56.441	0.000	15.225	16.329
cz	5.0540	0.070	71.838	0.000	4.915	5.193

Omnibus:	4.919	**Durbin-Watson:**	2.223
Prob(Omnibus):	0.085	**Jarque-Bera (JB):**	5.038
Skew:	0.261	**Prob(JB):**	0.0806
Kurtosis:	3.682	**Cond. No.**	86.0

Robust regression methods

- You can choose a variety of 'M-estimators' (some sum of a symmetric function of the residuals to minimize)
- First, we use a t-distribution based deweighting:

```
rlm_model = sm.RLM(mu, X, M=sm.robust.norms.HuberT())
rlm_results = rlm_model.fit()
```

- Next, deweighting by trimming values far from the mean entirely:

```
rlm_model = sm.RLM(mu, X, M=sm.robust.norms.TrimmedMean())
rlm_results = rlm_model.fit()
```

- Finally, the Tukey biweight:

```
rlm_model = sm.RLM(mu, X,M=sm.robust.norms.TukeyBiweight())
rlm_results = rlm_model.fit()
```

- As you can see, it's easy to explore different methods with `statsmodels`.

Guidelines for regression

- Key things to think about:
  - Most regression algorithms assume that errors in the independent variables are negligible. Is this true for your data?
  - Does the function you're fitting with make physical sense?
  - Do you expect Gaussian residuals, or might there be outliers?
  - Beware of cases where you have a few data points with long lever arms

Guidelines for regression

- Be sure to look at the fit results, and especially the residuals vs. x. Check for patterns in the residuals, which indicate you have the wrong function (or not enough polynomial terms).
- Be very careful about extrapolating regression results (especially with polynomial fits).
- Remember least-squares may not be an appropriate loss function for your problem.
- Never believe the error estimates from a regression routine, unless you're absolutely sure you know your errors and they're Gaussian. Estimated parameter uncertainties can often be off by factors of 2 or more, especially if you underestimate your errors.

Regression in scikit-learn

- We will start with simple linear regression, the same problem we've dealt with before.
- Reviewing some jargon:
  - ***training*** data is the data used to develop an instance of a machine-learning algorithm: e.g. the data we fit in a regression
  - that algorithm can then provide ***predictions*** for other data
  - the ***features*** of the training data are the quantities that will be used to make predictions (e.g., the x vector in regression as we described it before)

Regression in scikit-learn

- In scikit-learn, `sklearn.linear_model.LinearRegression` defines an object used to perform linear regression
- We'll find that many algorithms can be run with a similar interface.
- Note: some robust regression techniques are also available; check out the documentation on RANSAC and Huber Regression at http://scikit-learn.org/stable/modules/linear_model.html). Also see http://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model for detailed documentation about these routines).

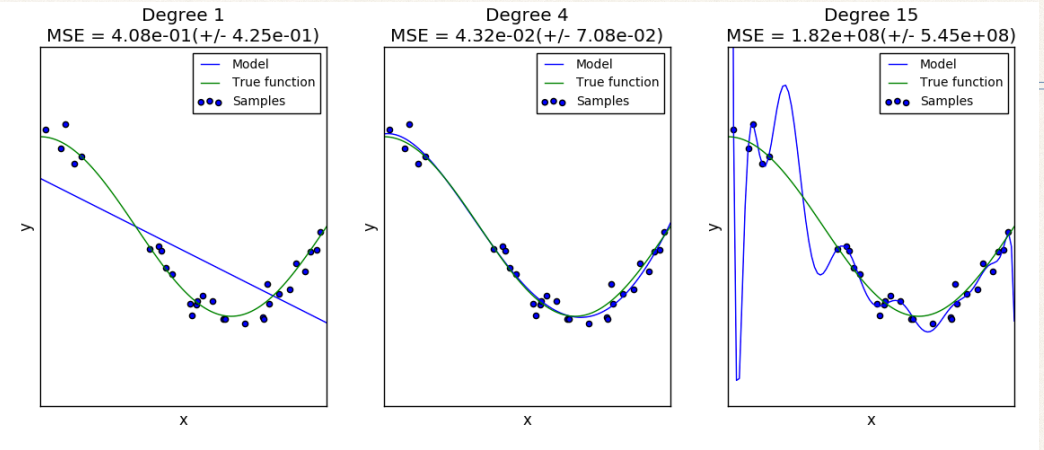
Regression in scikit-learn

```
import sklearn
import sklearn.linear_model as lm
# fit for mu as a function of logv, with weights of 1/sigma_mu^2

# X variable needs to be an (nsamples x nfeatures) array:
# need to reshape logv to an Nx1 array instead of just an N-element array
Xsl=logv.reshape(len(logv),1)

# create simple regression model object
simple_model = lm.LinearRegression()

# fit the model
simple_model.fit(Xsl, mu, weight)

# print out the slope & intercept from the model.  Note that coef_ is an
# array, but we only have one feature so it has one element.
print(simple_model.coef_, simple_model.intercept_)
```

Regression in scikit-learn

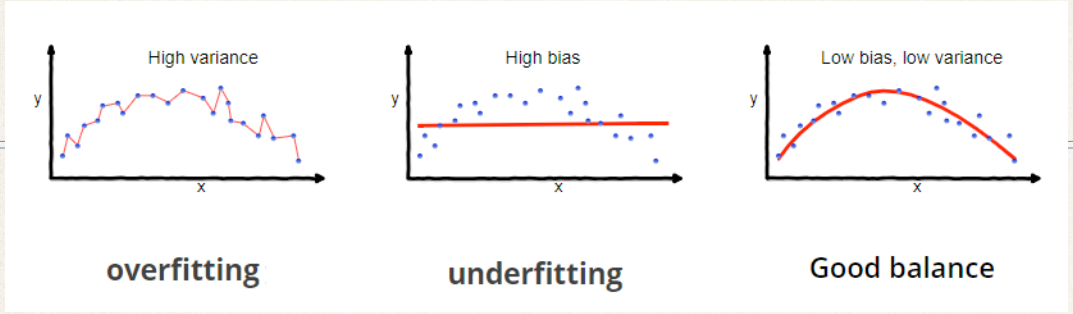
```
if 0:
    print(f'slope: {simple_model.coef_[0]:.3f}')
    print(f'intercept: {simple_model.intercept_:.3f}')
    print(f'H0: {(10**(-0.2*simple_model.intercept_-np.log10(1E-5))):.3f} km/sec/Mpc')

# we can make points semi-transparent by setting alpha < 1
    plt.errorbar(logv,mu,yerr=sigma_mu,fmt='g.',alpha=0.2)
    plt.ylim(33,39)
    plt.xlabel('log v')
    plt.ylabel(r'$\mu$')

# to get predictions from our model for arbitrary new data,
# we just use the .predict() method.
# Note that we still need an Nx1 array rather than a
# simple N-element array.
    plt.plot(logv,simple_model.predict(Xsl),'r-')
```

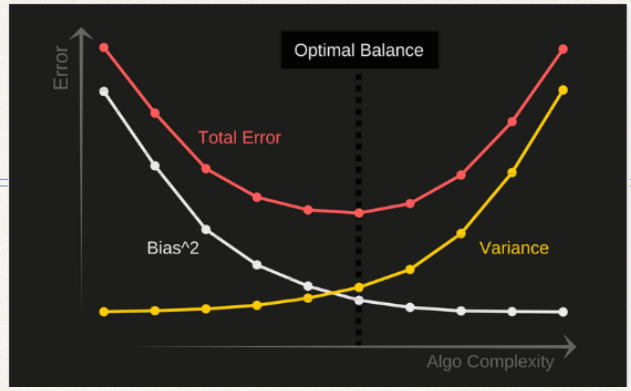
Review: The problem of over-fitting



- Plot from http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html
- Scikit-learn provides a number of variants of linear regression that minimize different losses (not just chi-squared) in such a way that we are less likely to get multiple nearly-cancelling coefficients as part of our fit, which may fit noise in our particular dataset well but will extrapolate poorly.
- We can explore this by trying to fit quartic functions again, with these regression variants.

The bias/variance trade-off





- Figures from https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229
- Bias: $< (y_{predicted} - y_{true}) >$
- Variance: $< (y_{predicted} - <y_{predicted}>)^2 >$
- A common problem: if we want to reduce bias in our statistics, increased variance is often the price we pay
  - E.g.: the mean vs. the median; cf. Homework 3
- We can instead minimize the risk = bias^2 + variance
  - Equivalent to the mean squared error on a test set, $< (y_{predicted} - y_{true})^2 >$

Results from overfitting: a quartic model

```
# Make an X array containing logv, logv^2, logv^3, and logv^4:
X_quart = np.stack([logv,logv**2,logv**3,logv**4],axis=1)

# initialize quartic model
quart_model = lm.LinearRegression()

# fit the model
quart_model.fit(X_quart, mu, weight)

# print out the coefficients & intercept from the model.
print(quart_model.coef_, quart_model.intercept_ )

# quart_model.predict(X_quart) would be used to get the predicted
# values from the fit
```

Ridge Regression

- Ridge regression adds robustness to overfitting due to covariant features by minimizing, not the sum of the square of the residuals, but that PLUS the sum of the squares of all the coefficients from the fit (their "L2 norm").
- To put all the features on the same rough scale, we will remap all features to have mean 0 and variance 1 (otherwise some features effectively get penalized more than others).
- Techniques for reducing overfitting are generally referred to as 'regularization' methods. By reducing overfitting, we reduce the variance we'd get from our fit when applied to data outside its training set, at the cost of potentially having a somewhat biased estimator (as we no longer are using an 'optimal', Maximum Likelihood-based least-squares method).
-

Fitting quartic model with ridge regression

```
# initialize ridge regression model.  The alpha parameter
# controls how strongly we penalize coefficient values.
ridge_model = lm.Ridge(alpha=1)
# We need to normalize variables to have mean 0 and variance 1
# before fitting.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# even the rescaling requires a fit step...
scaler.fit(X_quart)
# then we apply the scalar to create a new array of features
X_quart_norm = scaler.transform(X_quart)
# fit the model using Ridge Regression
ridge_model.fit(X_quart_norm, mu, weight)
```

- **Change the value of alpha over several orders of magnitude and observe how the coefficients (especially for the higher-order terms) and fit quality change. Alpha=0 corresponds to ordinary linear regression.**

LASSO Regression

- The LASSO algorithm penalizes the sum of the **absolute values** of the coefficients (the "L1 norm"), rather than the sum of the squares. This tends to drive coefficients all the way to zero -- i.e., eliminating features from the fit. It is implemented as `linear_model.Lasso` in Scikit-learn.
  - The `alpha` parameter determines how large the penalty for nonzero values is. `alpha=0` corresponds to ordinary regression.
  - `Scikit-learn` also implements ElasticNet in `linear_model.ElasticNet`; this combines an L2-based penalty like Ridge with an L1-based penalty like LASSO. It is run the same way as `linear_model.Lasso` except with an extra parameter to determine how much weight to give to L2 vs. L1.

Fitting quartic model with LASSO regression

```
# initialize LASSO model.  The alpha parameter controls
# how strongly we penalize for nonzero values of coefficients.
lasso_model = lm.Lasso(alpha=5E-2)

# fit the model (with weights, in this example)
lasso_model.fit(X_quart_norm,mu, sample_weight = weight)

# print out the coefficients & intercept from the model.
print(lasso_model.coef_, lasso_model.intercept_)
```

- **Change the value of alpha over several orders of magnitude and observe how the coefficients (especially for the higher-order terms) and fit quality change.**
  - alpha=0 corresponds to ordinary linear regression.

SGD Regression

- The Stochastic Gradient Descent algorithm can be useful when there are extremely large numbers of features (e.g. 10^5).
  - As implemented in Python, it can do robust regression, LASSO or Elastic Net with weights, etc.; very general, but can be slow.

Fitting quartic model with SGD regression

```
# initialize SGD model.  The alpha parameter controls how
# strongly we penalize for nonzero values of coefficients.
# We'll use the Huber t robust regression loss,
# and an L1 penalty ==LASSO ).
# A rule of thumb is to use a number of iterations
# equal to 10**6 / (size of dataset being fit), though below
# we'll need more for convergence.
# Set verbose=True if you want to see diagnostic output.

sgd_model = sgd_model = lm.SGDRegressor(loss='huber', penalty='l1', alpha=1E-2,verbose=False, tol=1E-3)


# fit the model.
sgd_model.fit(X_quart_norm, mu, sample_weight=weight)

# print out the coefficients & intercept from the model.
print(sgd_model.coef_, sgd_model.intercept_ )
```

Optimizing the value of the alpha parameter

- **Change the value of alpha over several orders of magnitude and observe how the coefficients (especially for the higher-order terms) and fit quality change.** As before, `alpha=0` corresponds to ordinary linear regression.
  - As we increase alpha, we are trading off increasing the potential bias of our estimated coefficients with reducing the variance of predictions outside our dataset.
  - We can assess the net effect by fitting our model ("training") with one subset of the data, and testing with another.
  - As before, we can assess errors in the presence of overfitting using `sklearn.model_selection.train_test_split` to split into separate, matched training and test sets.

Splitting into training and test sets

```
from sklearn.model_selection import train_test_split

# To better assess the quality of the fitting,
# we randomly split the data into Training (50%) and Test (50%) sets.
# The code below performs this task on the X_quart_norm, mu,
#   and sigma_mu arrays:


X_train, X_test, mu_train, mu_test, weight_train, weight_test = \
            train_test_split(X_quart_norm, mu, weight, \
            test_size = 0.5, train_size = 0.5)

from sklearn.metrics import mean_squared_error
```

Evaluating RMS error as we change alpha

```
# create an array of test values of alpha, from 1E-3 to 10**0.5,
# logarithmically spaced
alpha_test = np.logspace(-3,0.5,14)
mse=alpha_test*0.
#for each test value of alpha, evaluate the mean squared error
# from the regression fit, using the test set only
for i,a in enumerate(alpha_test):
    sgd_model = lm.SGDRegressor(loss='squared_error',penalty='l1',
     alpha=a, tol=1E-3, verbose=False)

# fit the model.
    sgd_model.fit(X_train,mu_train,sample_weight=weight_train)
# Calculate the weighted mean squared error (MSE)
    mse[i]=mean_squared_error(mu_test, sgd_model.predict(X_test),
     sample_weight=weight_test)
# Note: RMS error is square root of MSE
```

Review: Cross-validation

- In k-fold cross-validation, we split the data into k subsets. We loop over the subsets, training with all but one and testing with the other; in the end, we get the performance of training with a fraction *(k-1)/k* of the data, but are able to get test statistics based on the **entire** dataset.
  - This is easy to do in scikit-learn, but does mean running the training *k* times on more data than before...so slows things down a lot.

Applying cross-validation

```
# cross_val_predict will do training and prediction for a
# scikit-learn object, rotating the dataset between
# training and prediction via cross-validation
from sklearn.model_selection import cross_val_predict
mse=copy(alpha_test)*0.
for i,a in enumerate(alpha_test):
# define the model object with our choice of alpha
   sgd_model = lm.SGDRegressor(loss='squared_loss',penalty='l1',
                        alpha=a,tol=1E-3,verbose=False)
# do training AND prediction on the whole sample using
# cross-validation
    predicted = cross_val_predict(sgd_model,X_quart_norm,mu, cv=5)
#calculate mean squared error
    mse[i]=mean_squared_error(mu,predicted,sample_weight=weight)
```

- **What value of alpha would you choose?**

Applying cross-validation

```
# let's do the fit with the alpha value that minimized mean squared
error

whbest = mse.argmin()
print(f'best alpha: {alpha_test[whbest]:.3f}')

sgd_model = lm.SGDRegressor(loss='huber',penalty='l1',\
          alpha=alpha_test[whbest],verbose=False, tol=1E-3)
```

- **For this value of alpha, how many coefficients are being set to 0?**

MCMC and related methods

Review: Recipe for Maximum Likelihood

1. Write down $L=p(data \mid \boldsymbol{\theta})$, where $\boldsymbol{\theta}=[\theta_1, \theta_2,...]$; let $LL = \ln L$. Then $LL = \ln p(data \mid \boldsymbol{\theta})$
2. Either:
   A. Figure out the analytic derivatives, $\delta LL / \delta \theta_i$. Find the point(s) where all of these derivatives = 0, which indicates that they must be minima, maxima, or saddle points.
   B. Write a Python function that returns -*LL* for your actual dataset, given the value of the parameters. You can then use a variety of Python routines to find the minimum of that function (most can be applied via `scipy.optimize.minimize`)
   C. Set up a grid of possible values for $\boldsymbol{\theta}$. Calculate *LL* at each point of the grid; choose the maximum value; or
   D. apply Markov Chain Monte Carlo methods to explore $p(data \mid \boldsymbol{\theta})$
3. Check that your solution is actually a maximum of LL.

As the number of dimensions/free parameters gets large, these methods all break down... the "curse of dimensionality"

A. Figure out the analytic derivatives, $\delta LL / \delta \theta_i$. Find the point(s) where all of these derivatives = 0, which indicates that they must be minima, maxima, or saddle points.
B. Write a Python function that returns -*LL* for your actual dataset, given the value of the parameters. You can then use a variety of Python routines to find the minimum of that function (most can be applied via `scipy.optimize.minimize`)
C. Set up a grid of possible values for $\boldsymbol{\theta}$. Calculate *LL* at each point of the grid; choose the maximum value; or

Markov Chain Monte Carlo analysis provides an alternative

- **Markov Chain** -- a system for which the next step it takes is determined only by its current state, not its past history
- **Monte Carlo** - a calculation that proceeds via random sampling
  - MCMC is a family of methods that generate values distributed as a desired probability distribution -- generally, the posterior probability distribution function for a set of parameters
  - The points are generated by evolving states according to a rule which ensures the probability of going from state A to state B is proportional to the posterior value at state B divided by the posterior value at state A
  - As the number of steps goes to infinity (if things are well-behaved), the distribution of points generated should then be proportional to the posterior when following such a rule

The classic MCMC algorithm: Metropolis-Hastings

Algorithm 1 The procedure for a single Metropolis-Hastings MCMC step.

1: Draw a proposal $Y \sim Q(Y; X(t))$
2: $q \leftarrow [p(Y)\, Q(X(t); Y)]/[p(X(t))\, Q(Y; X(t))]$ *// This line is generally expensive*
3: $r \leftarrow R \sim [0, 1]$
4: **if** $r \leq q$ **then**
5: $\quad X(t+1) \leftarrow Y$
6: **else**
7: $\quad X(t+1) \leftarrow X(t)$
8: **end if**

- Pseudocode from Foreman-Mackey et al. 2013
  - *X(t)* is the state of the system (=set of parameter values) at step t
  - *Y* is a **potential** new state
  - *p(X)* represents the posterior value at a point in parameter space X
  - *Q(X(t);Y)* is the probability of transitioning from point Y to X(t), and *Q(Y;X(t))* is the probability of transitioning from point X(t) to Y; we can choose ~any rule we want for how to generate the possible next step
  - Most commonly, Q is a multivariate Gaussian distribution, in which case Q(Y; X) = Q(X; Y) by construction.

The classic MCMC algorithm: Metropolis-Hastings

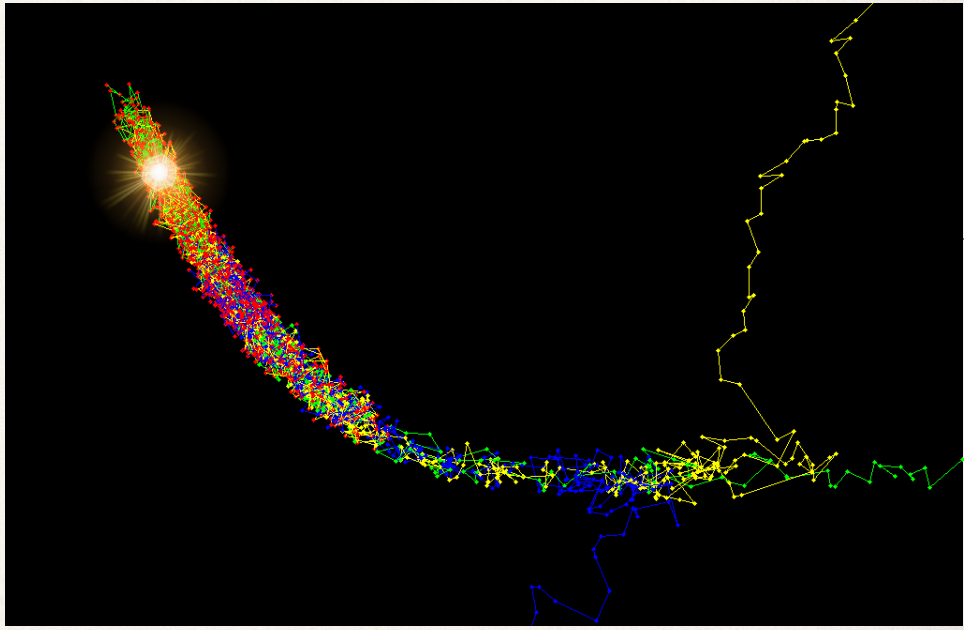
Algorithm 1 The procedure for a single Metropolis-Hastings MCMC step.

1: Draw a proposal $Y \sim Q(Y; X(t))$
2: $q \leftarrow [p(Y)\, Q(X(t); Y)]/[p(X(t))\, Q(Y; X(t))]$ *// This line is generally expensive*
3: $r \leftarrow R \sim [0, 1]$
4: **if** $r \leq q$ **then**
5: $\quad X(t+1) \leftarrow Y$
6: **else**
7: $\quad X(t+1) \leftarrow X(t)$
8: **end if**

- Pseudocode from Foreman-Mackey et al. 2013
  - *q* is the 'acceptance probability'
    - the way this is set up, if we change state from *X(t)* to *Y* with probability *q*, the distribution of state values will eventually converge to be proportional to *p(X)*
  - We generate a random number and compare to the acceptance probability *q*
  - *If r < q* we accept the step, so *X(t+1)* = Y; otherwise the chain stays in its current position

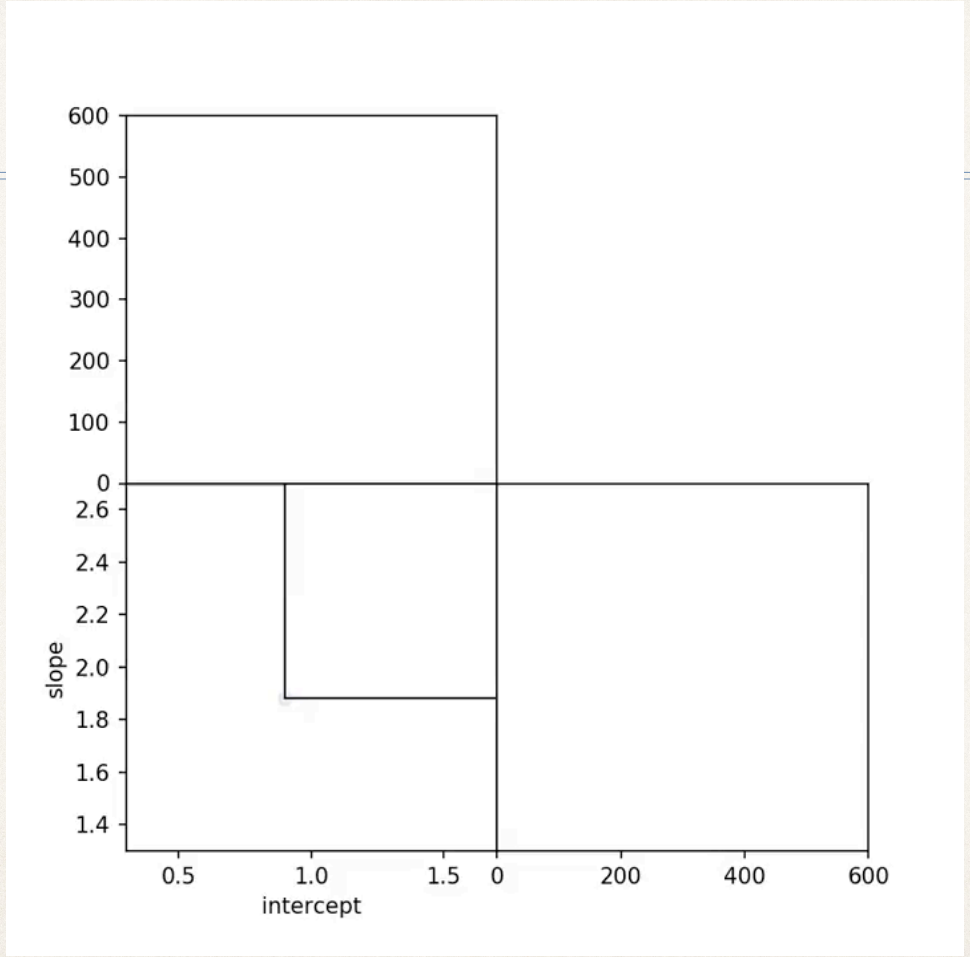
The net result...

- Transitions to states with higher posterior probability at Y than X are always accepted, but those with lower posterior probability are only accepted sometimes
- As a result the points in the chain tend to spend most of their steps in high-probability regions
  - After a 'burn-in' period that retains memory of whatever initial guess was used to initialize the chain, the distribution of points should match random draws from the posterior



- Figure from wikimedia.org . Red points are the ones after burn-in.

Animation of Metropolis sampling for fitting a line



Source: https://twiecki.io/blog/2014/01/02/visualizing-mcmc/