Quantitative Applications for Data Analysis: feature selection

Erik Spence

SciNet HPC Consortium

21 March 2024

SciNet compute • calcul CANADA

Today's slides

Today's slides can be found here. Go to the "Quantitative Applications for Data Analysis" page, under Lectures, "Feature selection".

https://scinet.courses/1346

SciNet compute • calcul CANADA

Today's class

Today we are going to explore the problem of picking the set of features to include in your model.

- p-values, for features,
- feature selection techniques,
- Various measures of model efficacy.
- Lasso, Ridge, Elastic Net regression.

As with all classes, please ask questions.

Multivariate linear regression

Let's take a look at linear regression again, using the statsmodels package and a wider data set.

The star98 data set is a set of standardized student test scores taken in California in 1998. The target represents the 9th graders that scored above the national median on the mathematics exam.

```
In [1]:
In [1]: import statsmodels.api as sm
In [2]:
In [2]: data = sm.datasets.star98.load()
In [3]:
In [3]: x = data.exog
In [4]: y = data.endog
In [5]: x = sm.add_constant(x)
In [6]:
In [6]: model = sm.OLS(y.iloc[:,0], x)
In [7]: results = model.fit()
In [8]:
In [8]: print(results.summary())
In [9]:
```

Multivariate Linear Regression, continued

Model:	OLS	Adj. R-squared:	0.507
[...]			
Time:	13:01:10	Log-Likelihood:	-2279.5
No. Observations:	303	AIC:	4601.
Df Residuals:	282	BIC:	4679.
[...]			

	coef	std err	t	$P > \|t\|$	[95.0% Conf.	Int.]
const	-1.014e+04	5209.878	-1.946	0.053	-2.04e+04	115.090
x1	0.2139	2.230	0.096	0.924	-4.175	4.603
x2	15.3423	3.505	4.377	0.000	8.443	22.242
x3	5.9944	4.140	1.448	0.149	-2.155	14.144
x4	-3.0512	2.138	-1.427	0.155	-7.261	1.158
x5	1597.6665	155.557	10.271	0.000	1291.467	1903.866
x6	1176.8157	249.155	4.723	0.000	686.377	1667.255

Multivariate Linear Regression, continued more

	coef	std err	t	$P > \|t\|$	[95.0% Conf. Int.]	
x7	340.5269	62.086	5.485	0.000	218.317	462.737
x8	-1714.4834	904.058	-1.896	0.059	-3494.042	65.075
x9	-405.7498	195.183	-2.079	0.039	-789.951	-21.549
x10	-228.3569	99.588	-2.293	0.023	-424.388	-32.326
x11	7.1600	4.862	1.473	0.142	-2.411	16.731
x12	0.4313	1.333	0.324	0.747	-2.193	3.056
x13	-109.8054	10.718	-10.245	0.000	-130.903	-88.708
x14	-28.1694	2.587	-10.888	0.000	-33.262	-23.077
x15	-21.4271	4.318	-4.963	0.000	-29.926	-12.928
x16	84.6002	43.967	1.924	0.055	-1.945	171.146
x17	47.8686	20.668	2.316	0.021	7.185	88.552
x18	11.9591	4.679	2.556	0.011	2.750	21.168
x19	1.9302	0.175	11.049	0.000	1.586	2.274
x20	-2.5449	1.018	-2.499	0.013	-4.549	-0.540

Multivariate Linear Regression, continued even more

What do we make of all of this?

- We can compute a linear regression with all available features.
- Is this meaningful?
- Naively, it looks like some of these features aren't very important.
- We can construct a better-motivated, more robust model if we reduce the number of features.
- Some regression/clustering/classification algorithms have difficulties on problems with many extraneous features.
- Sometimes features are highly colinear and will break some algorithms.
- The information "only these 5 (say) features are really important" is itself worth having.

We should strive to produce the simplest model that produces sufficiently good answers: feature selection.

SciNet compute • calcul CANADA

How not to do Feature Selection

"Features 3, 5, 6, 13, 14, 15, and 19 all have really low p values - let's just use those."

- No.

A super-low p-value means, "in this model, the probability of committing a Type I error is low, under the null hypothesis that the coefficient is zero".

- It says nothing about its contribution to the hypothetical model in which you've removed other features.
- It says even less about how important the contribution of the feature is.
- If there are correlations between features then the p-values may move drastically if features are removed.

Do not just get rid of features based soley on the p value.

SciNet compute • calcul CANADA

How to do a little feature selection

There are some filtering steps one can take to judiciously weed out some clearly irrelevant features:

- Low variance filtering (sklearn.feature_selection.VarianceThreshold). If the feature is essentially constant, it can't matter much to any model (there's no information in that feature).
- Univariate tests (sklearn.feature_selection.SelectKBest): for supervised learning, *if* your data is sampled densely enough, you can do univariate tests on each parameter and remove ones that are sufficiently uncorrelated with the target.
  - ANOVA F-test can be used for numerical features and categorical data (sklearn.feature_selection.f_classif).
  - Mutual information, from information theory. Can be used for categorical or numerical input and categorical output (sklearn.feature_selection.mutual_info_classif and mutual_info_regression).

Filtering your features for those which are irrelevant is a good first step.

How, maybe, to do feature selection

How have we done model selection in the past?

- Cross-validation!

But this is a huge cross-validation (CV) problem:

- Try all 2^p possible combinations of features, where p is the number of features.
- Regress on them.
- Of the possibilities, find best one, based on some metric, or combination of metrics.
- This actually has a name: "Best Subset Selection" (or sometimes, Best Subset Regression).
- This will work for modest p.
- (But for modest p we don't really need to do much feature selection.)

SciNet compute • calcul CANADA

Aside: Danger, Danger!

In general, flags should be going off inside you when facing the possibility of doing tens of thousands of tests on your data.

For feature selection this is ok. But for small variations on this theme things can quickly go horribly, inexorably, off the rails.

- "Let's look through all pairwise combinations of my 100 features, looking for statistically significant correlations!"
- This can come up in cases where it's not necessarily obvious - looking for correlations between pixels in images.
- If you are doing 10,000 hypothesis tests, using as your significance $p < 0.05$, you should expect 500 significant results — even if the data is just random noise.

Huge numbers of tests will naturally result in false positives. Be careful.

SciNet compute • calcul CANADA

Forward and backward selection

Exhaustive search is safe, but infeasible in the most urgent cases. ($p = 100$ implies 10^{30} model tests, and 100 isn't a huge number of features.) What other options have we?

Three greedy methods are in common use, but can get caught in local minima:

- Forward selection: starting from nothing,
  - For each remaining feature, include it, and calculate CV error, for some metric.
  - If for the best feature, the error drops enough, select it and continue,
  - Else terminate and report the selected features.
  - sklearn.feature_selection.f_regression.
- Backward selection: same, but start with a model with all features, and drop until error rises too much.
  - sklearn.feature_selection.RFE, sklearn.feature_selection.RFECV (Recursive Feature Elimination with CV).
- Stepwise selection: combination of Forward and Backward selection.

SciNet compute • calcul CANADA

Feature selection criteria, continuous targets

Those techniques are all well and good, but what measure should we be using to determine if one combination of features is better than another?

- We always expect the more-complex model to fit a little better,
- is the extra complexity (the extra features) sufficiently better to justify using it?
- $\boldsymbol{R^2}$ will always go up with increasing complexity of the model (and thus is not a good criterion).
- Adjusted $\boldsymbol{R^2}$ is better, as it penalizes bigger models.

There are a few more measures which can be used to determine if one combination of features is better than another:

- ANOVA,
- Mallows' $\boldsymbol{C_p}$,
- Information criteria methods (AIC, BIC).

Feature selection criteria, categorical targets

If our target, $\boldsymbol{y}$, is not continuous, but is instead categorical, then we have different measures of performance:

- Accuracy,
- ROC curve, area under the ROC curve.

These, in conjunction backward/forward selection and cross-validation, can be used to find your best model. We've seen these in previous classes.

R^2

Recall that R^2 is one minus the ratio of the variation in the data explained by the model divided by the total variation in the data.

$$\begin{aligned} R^2 &= 1 - \frac{SS_{\text{model}}}{SS_{\text{total}}} \\ &= 1 - \frac{\sum_i (y_i - f(\vec{x}_i))^2}{\sum_i (y_i - \langle y \rangle)^2} \end{aligned}$$

where $f(\vec{x}_i)$ is the value of the model for input $\vec{x}_i$, $\langle y \rangle$ is the mean value of y, SS stands for "Sum of Squares", and the sum is over all data points.

This is not a good measure to use, since it will always be higher for a more-complex model.

Adjusted R^2

Because more-complex models will always have a higher R^2, since it does not punish models for complexity, it's better to use Adjusted R^2 as a measure of how well the model is doing.

$$
\begin{aligned}
R^2_{\text{adj}} &= 1 - \frac{SS_{\text{model}}/df_{\text{model}}}{SS_{\text{total}}/df_{\text{total}}} \\
&= 1 - \frac{SS_{\text{model}}/(n-1-p)}{SS_{\text{total}}/(n-1)} \\
&= 1 - \frac{(n-1)\sum_i (y_i - f(\vec{x}_i))^2}{(n-1-p)\sum_i (y_i - \langle y \rangle)^2}
\end{aligned}
$$

where df_{model} is the number of degrees of freedom of the model, n is the number of data points, and p is the number of features in the model. As the number of features goes up the values of R^2_{adj} goes down.

Mallows' C_p

Mallows' C_p statistic is defined as

$$C_p = \frac{SS_{\text{model}}}{MSS_{\text{all}}} - n + 2p$$

where MSS_{all} is the mean sum of squares of the model including all features, SS_{model} refers to a model including a subset of all the possible features, and p now includes the intercept as one of the fittable features.

We won't derive where this comes from, since it's related to bias-variance tradeoff, a topic we haven't covered in this course. Let it be sufficient to say, an ideal model will have

$$C_p \lesssim p$$

This is a good criteria for selecting which features to include in your model.

ANOVA model comparison

ANOVA can also be used to compare different regression models to each other. How does that work?

- ANOVA is usually used to compare the means of different groups to each other.
- Here the ANOVA will determine if one model is significantly better at fitting the data, given the number of free parameters in the model.
- How? It calculates the F statistic to compare these two models.
- The F statistic is used to calculate the p-value for this hypothesis test.
- The Null Hypothesis here is that the more-complex model does not add any predictive value.

We can compare multiple models to each other, in order of increasing complexity.

SciNet compute • calcul CANADA

ANOVA model comparison, example

ANOVA can be used to compare different regression models to each other.

Let's begin by recreating our noisy data from the resampling class.

The data need to be cast into a dictionary for the statsmodels package to handle.

```
In [9]:
In [9]: import numpy as np
In [10]: import numpy.random as npr
In [11]:
In [11]: n = 40
In [12]:
In [12]: x = np.linspace(-1, 1, n)
In [13]: x += 0.1 * npr.rand(n)
In [14]:
In [14]: y = np.tanh(8 * x) - x
In [15]: y += 0.1 * npr.rand(n)
In [16]:
In [16]: data = {'x': x, 'y': y}
In [17]:
```

ANOVA model comparison, example, continued

As you can see, the statsmodels package wants a formula using the R syntax.

We use the "I(x**2)" function to create polynomial terms, as the "poly" function is not available here.

Also note that the models must be "nested" (each model contain the previous model) for this approach to make statistical sense.

```
In [17]: import statsmodels.formula.api as smf
In [18]: import statsmodels.stats.anova as smsa
In [19]:
In [19]: m1 = smf.ols("y ~ x", data = data).fit()
In [20]: m2 = smf.ols("y ~ x + I(x**2)", data = data).fit()
In [21]: m3 = smf.ols("y ~ x + I(x**2) + I(x**3)",
...:                  data = data).fit()
In [22]:
In [22]: smsa.anova_lm(m1, m2, m3)
Out[22]:
   df_resid       ssr df_diff   ss_diff          F        Pr(>F)
0      38.0  6.033968     0.0       NaN        NaN           NaN
1      37.0  5.890577     1.0  0.143391   2.504684  1.220198e-01
2      36.0  2.060963     1.0  3.829614  66.894011  9.971036e-10
In [23]:
```

AIC, BIC

There are two Information Criterion methods you can also use to determine your optimal model:

- Akaike Information Criteria (AIC), $\mathbf{AIC} = \boldsymbol{2k - 2\log(L)}$
- Bayesian Information Criteria (BIC), $\mathbf{BIC} = \boldsymbol{k\log(n) - 2\log(L)}$

where

- $\boldsymbol{k}$ is the number of model parameters.
- $\boldsymbol{L}$ is the maximum value of the model likelihood function, $\mathbf{P}(\vec{\boldsymbol{x}}|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the model parameters.
- $\boldsymbol{n}$ is the number of data points.

AIC is based on the KL divergence. The lower the values of AIC or BIC, the better the model, taking into account the complexity of the model.

AIC model selection, regression example

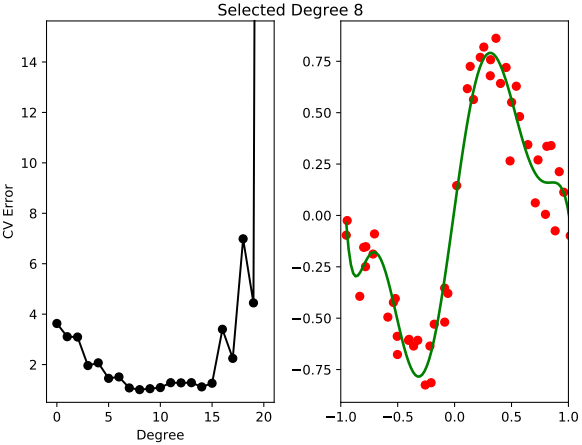
Unfortunately, sklearn does not have a built-in function for calculating AIC.

- This is because, in general, you cannot calculate $\log(L)$ without knowing the form of the model in question.
- However, for linear regression models, we can use the squared residuals of the model to estimate the maximum of the model likelihood function.
- This results in a slightly different expression for AIC, but is equivalent.

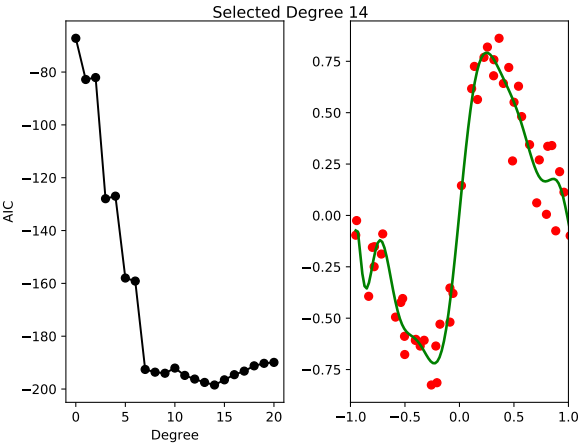
```
# AICdemo.py
import numpy as np

def calcAIC(x, y, d):

  p = numpy.polyfit(x, y, d)
  fit = numpy.poly1d(p)
  err = sum((y - fit(x))**2)

  n = len(x)

  return 2 * (d + 1) + n * np.log(err / n)
```

You'll notice that the output of our first statsmodels model included AIC and BIC.

AIC model selection, regression example, continued



Cross-validation



AIC

SciNet compute • calcul CANADA

Lasso, Ridge and Elastic Net regression

Recall that for generic least-squares linear regression, we are minimizing the MSE:

$$\begin{aligned}
\mathbf{MSE} &= \frac{1}{n}\sum_i \left(y_i - f(\vec{x}_i)\right)^2 \\
&= \frac{1}{n}\sum_i \left(y_i - x_{i0}\beta_0 - x_{i1}\beta_1 - \cdots - x_{ip}\beta_p\right)^2 \\
&= \frac{1}{n}\sum_i \left(y_i - \vec{x}_i \cdot \vec{\beta}\right)^2
\end{aligned}$$

resulting in

$$\hat{\beta} = \mathbf{argmin}_\beta \sum_i \left(y_i - \vec{x}_i \cdot \vec{\beta}\right)^2$$

We'd like to keep the problem as simple as possible - one way to do that in regression is forcing the fit parameters to stay small; to constrain $|\vec{\beta}|$ in some way.

Lasso, Ridge and Elastic Net regression, continued

Lasso, Ridge and Elastic Net regressions perform a similar minimization as on the previous slide, but with a penalty term (regularization) for the coefficients:

Ridge:

$$\hat{\beta} = \mathbf{argmin}_{\beta} \sum_{i} \left(y_i - \vec{x}_i \cdot \vec{\beta} \right)^2 + \alpha |\vec{\beta}|_2^2$$

Lasso:

$$\hat{\beta} = \mathbf{argmin}_{\beta} \sum_{i} \left(y_i - \vec{x}_i \cdot \vec{\beta} \right)^2 + \alpha |\vec{\beta}|_1$$

Elastic Net:

$$\hat{\beta} = \mathbf{argmin}_{\beta} \sum_{i} \left(y_i - \vec{x}_i \cdot \vec{\beta} \right)^2 + \alpha_1 |\vec{\beta}|_1 + \alpha_2 |\vec{\beta}|_2^2$$

Feature selection using Lasso regression

Why would you do that?

- The effect of the regularization (the extra term) is to keep any particular $\boldsymbol{\beta_j}$ from getting too big.
- This acts as a type of smoothing, and helps to prevent overfitting.
- Because of the smoother penalty term in Ridge regression, features are not zeroed out ($\boldsymbol{\beta_j = 0}$),
- Lasso will zero out coefficients, and thus can do feature selection.
- Lasso has a shortcoming: in small-$\boldsymbol{n}$-large-$\boldsymbol{p}$ situations Lasso will select at most $\boldsymbol{n}$ features before it saturates.
- Elastic Net overcomes this shortcoming.

SciNet compute • calcul CANADA

Lasso regression, example

In scikit-learn, these are all available through sklearn.linear_model. They all have cross-validation versions as well.

```
In [23]: import sklearn.model_selection as skms
In [24]: import sklearn.linear_model as sklm
In [25]: import sklearn.datasets as skd
In [26]:
In [26]: data = skd.load_diabetes()
In [27]:
In [27]: train_x, test_x, train_y, test_y = \
...:         skms.train_test_split(data.data,
...:                               data.target,
...:                               test_size = 0.2)
In [28]:
```

```
In [28]:
In [28]: model = sklm.LassoCV(cv = 10)
In [29]: model = model.fit(train_x, train_y)
In [30]:
In [30]: model.coef_
Out[30]:
array([ 16.14392371, -243.94964858,
557.4299411, 293.81352676, -661.40382688,
360.91035424, 0., 160.83046309, 693.25160897,
91.10262666])
In [31]:
In [31]: model.alpha_
Out[31]: 0.006553032886916717
In [32]:
```

Other types of feature selection

Note that some algorithms naturally perform feature selection, either implicitly or explicitly.

- Decision Trees pick out the most important features for you, by choosing them for splits. The "feature importances" can also be calculated by the splitting algorithm.
- Dimensionality reduction techniques, such as PCA, implicitly remove features from the data which are not important.

These can also be used to do feature selection.

SciNet compute • calcul CANADA

Summary

Today we discussed techniques you can use to do feature selection.

- Do not rely on the p-values to do feature selection.
- Forward and backward selection are options, though a metric for selecting the best model must be chosen.
- Adjusted R^2, Mallows' C_p, AIC and BIC are all options.
- ANOVA can be used to compare nested models to each other.
- Lasso and Elastic Net regression can also be used to do feature selection.

SciNet compute • calcul CANADA