Session 1: Statistical and Machine Learning Regression

2021 July 12

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
http://rmc.link/

A quick overview of the course

Goals: Day 1

> All about econometrics

1. Traditional econometrics on panel data in python
   - Tying back to using Pandas
   - Linear and logistic (among many others)
2. Machine learning approaches to econometrics
   - LASSO
   - Elastic Net
   - SVM
   - XGBoost
   - Combining the above

Goals: Day 2

All about text data

1. Working with text in python
   - Importing
   - Pattern matching (regular expressions)
2. Using Parsers
   - Natural language using NLTK and spaCy
   - Web pages using Beautiful Soup
3. Text classification
   - Supervised using textbooks
   - Embedding methods
   - Unsupervised using LDA
4. Dimensionality reduction
   - t-SNE and UMAP

Goals: Day 3

> More advanced/modern concepts

1. Bias in algorithms or data
   - Using Shapley additive explanations (SHAP)
2. Causal ML
   - Double/debiased/Neyman ML
3. Neural networks
   - Various network structures
   - Introduction to Keras
   - Leveraging pre-built models

Main applications

Application 1: Linear problem

- Idea: Discussion of risks, such as as foreign currency risks, operating risks, or legal risks should provide insight on the volatility of future outcomes for the firm.
- Testing: Predicting future stock return volatility based on 10-K filing discussion

Dependent Variable

- Future stock return volatility

Independent Variables

- A set of 31 measures of what was discussed in a firm's annual report

> This test mirrors Bao and Datta (2014 MS)

Application 2: Binary problem

- Idea: Using the same data as in Application 1, can we predict instances of intentional misreporting?
- Testing: Predicting 10-K/A irregularities using finance, textual style, and topics

Dependent Variable

Intentional misreporting as stated in 10-K/A filings

Independent Variables

- 17 Financial measures
- 20 Style characteristics
- 31 10-K discussion topics

> This test mirrors a subset of Brown, Crowley and Elliott (2020 JAR)

Preparation

Importing data in Pandas

- We can use `pandas` to import the data set
- Notes:
  1. `pandas` is traditionally imported as `pd` using `import pandas as pd`
  2. `pd.read_csv()` is able to read csv files *as well as compressed csv files
     - This is very useful!
     - Compressing a csv file can save 50-90% of the storage space of the file

```python
df = pd.read_csv('../../Data/S1_data.csv.gz')
```

- Note:
  - SAS, python pandas, and R can all handle `.csv.gz` and `.csv.zip` files
  - Stata is a bit tedious here, requiring uncompressing first
    - Either use your file manager or using Stata's `unzipfile` command

Examining the data

```
df.shape
```

```
## (14301, 198)
```

```
df.describe().to_html()
```

	gvkey	Firm	sic	year	logtotasset	rsst_acc	chg_re
count	14301.000000	1.430100e+04	14301.000000	14301.000000	14301.000000	14301.000000	14301.(
mean	38272.730159	7.100841e+05	4628.199636	2001.717362	5.507901	0.014126	0.00637
std	39101.761060	3.745443e+05	1973.464631	1.729618	1.905595	0.386033	0.0711
min	1004.000000	2.000000e+01	100.000000	1999.000000	-0.796288	-27.752728	-0.9328
25%	9225.000000	3.546550e+05	3330.000000	2000.000000	4.115454	-0.053155	-0.0129
50%	24708.000000	8.686110e+05	3841.000000	2002.000000	5.370675	0.021280	0.0051
75%	62811.000000	1.002531e+06	5900.000000	2003.000000	6.729078	0.091943	0.0301
max	230796.000000	1.261482e+06	9997.000000	2004.000000	12.397614	22.244062	0.8337

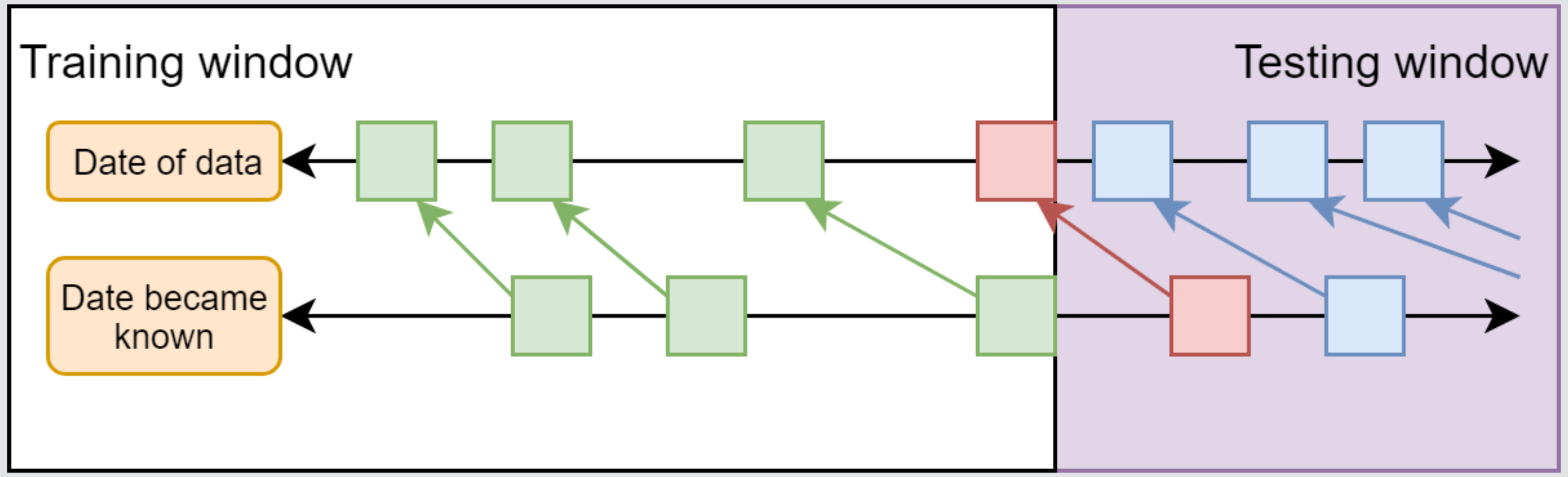
Other preparation

- For convenience later, we can store the variable names we will use for regressions into lists
  - Note the use of a list comprehension for the topic measures
    - There are 31 measures in the data, but the name is all of the form `Topic_#_n_oI`

```python
vars_financial = ['logtotasset', 'rsst_acc', 'chg_recv', 'chg_inv', 'soft_assets', 'pct_chg_cashsales', 'chg_roa',
                  'issuance', 'oplease_dum', 'book_mkt', 'lag_sdvol', 'merger', 'bigNaudit', 'midNaudit', 'cffin',
                  'exfin', 'restruct']

vars_style = ['bullets', 'headerlen', 'newlines', 'alltags', 'processedsize', 'sentlen_u', 'wordlen_s', 'paralen_s',
              'repetitious_p', 'sentlen_s', 'typetoken', 'clindex', 'fog', 'active_p', 'passive_p', 'lm_negative_p',
              'lm_positive_p', 'allcaps', 'exclamationpoints', 'questionmarks']

vars_topic = ['Topic_' + str(i+1) + '_n_oI' for i in range(0,31)]
```

Validating predictive analyses

- Ideal:
  - Withhold the last year (or a few) of data when building the model
  - Check performance on *hold out sample*
  - This is *out of sample* testing
  - Ensure that the data is independent across time!



- Sometimes acceptable:
  - Withhold a random sample of data when building the model
  - Check performance on *hold out sample*
  - Potential problems with correlations between hold out sample and training sample

Training vs. testing split

- A simple approach is to split by time
- Check which years are in the data using `.unique()`

```python
# Check the years in the data
df['year'].unique()
```

```
## array([2002, 2003, 2004, 1999, 2000, 2001], dtype=int64)
```

- Split out the last year as the testing sample
  - This can be done using a simple conditional
  - Final year is 2004, so…
    - Testing: `df.year == 2004`
    - Training: `df.year < 2004`

```python
# Subset the final year to be the testing year
train = df[df.year < 2004]
test = df[df.year == 2004]
print(df.shape, train.shape, test.shape)
```

```
## (14301, 198) (11478, 198) (2823, 198)
```

- Note that the number of rows in `df` is the same as the sum of rows in `train` and `test`

Aside: Random testing sample

- Scikit-learn (`sklearn`) can handle this robustly
  - Scikit-learn is a package focused on simple machine learning methods
- Since random sampling is common in ML, Scikit-learn provides multiple ways to handle this.
  - The function is `sklearn.model_selection.train_test_split()`

```python
Y1 = df['sdvol1']
X1 = df.drop(columns=['sdvol1'])

# test_size specifies the percent of the files to hold for testing
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X1, Y1, test_size=0.2)

print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
## (11440, 197) (2861, 197) (11440,) (2861,)
```

- Optionally you can stratify across classes in your data using the `stratify=` parameter

Running simple regressions in Python

Package: Statsmodels

- The `statsmodels` package provides a suit of basic regression functions
- It supports most standard statistical approaches
  - OLS, Logit, GLM, Probit, Poisson, ARIMA, etc.
- It includes some other interesting functions as well, such as:
  - Imputation methods (e.g., MICE), GAMs, Quantile regression, Markov switching, etc.
- There are 2 interfaces to the package:
  1. `statsmodels.formula.api` (usually imported as `smf`) – pandas-friendly
  2. `statsmodels.api` (usually imported as `sm`) – requires data to be formatted differently

Linear regression (OLS)

- Unlike most statistical software, regressions in `statsmodels` require multiple steps.

Step 1: specify the regression structure

```
model = smf.ols(formula='sdvol1 ~ logtotasset + fog', data=train)
```

- Note the use of ~ as the equals sign in the equation

Step 2: Run the regression

```
fit1 = model.fit()
```

Linear regression (OLS)

Step 3: Output the results (optional)

```
fit1.summary()
```

OLS Regression Results

Dep. Variable:	sdvol1	**R-squared:**	0.201
Model:	OLS	**Adj. R-squared:**	0.201
Method:	Least Squares	**F-statistic:**	1445.
Date:	Mon, 12 Jul 2021	**Prob (F-statistic):**	0.00
Time:	02:31:18	**Log-Likelihood:**	24787.
No. Observations:	11478	**AIC:**	-4.957e+04
Df Residuals:	11475	**BIC:**	-4.955e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.0523	0.004	14.869	0.000	0.045	0.059
logtotasset	-0.0073	0.000	-52.769	0.000	-0.008	-0.007
fog	0.0019	0.000	9.627	0.000	0.002	0.002

Omnibus:	8713.393	**Durbin-Watson:**	1.394

Tricks with statsmodels

#1. Using a function in an equation

```python
model = smf.ols(formula='sdvol1 ~ np.log(asset) + fog', data=train)
fit1 = model.fit()
```

#2. Defining your function in a variable

```python
formula = 'sdvol1 ~ logtotasset + fog'

model = smf.ols(formula=formula, data=train)
fit1 = model.fit()
```

Full model

```
formula = 'sdvol1 ~ ' + ' + '.join(vars_topic[0:-1])

model = smf.ols(formula=formula, data=train)
fit_ols = model.fit()

fit_ols.summary()
```

OLS Regression Results

Dep. Variable:	sdvol1	**R-squared:**	0.161
Model:	OLS	**Adj. R-squared:**	0.159
Method:	Least Squares	**F-statistic:**	73.45
Date:	Mon, 12 Jul 2021	**Prob (F-statistic):**	0.00
Time:	02:31:19	**Log-Likelihood:**	24508.
No. Observations:	11478	**AIC:**	-4.895e+04
Df Residuals:	11447	**BIC:**	-4.873e+04
Df Model:	30		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0458	0.000	171.114	0.000	0.045	0.046
Topic_1_n_ol	1.1709	0.340	3.440	0.001	0.504	1.838

Estout/Outreg2 style tables in Python

- To combine multiple regressions into one using statsmodels, you can use the stargazer package

```
Stargazer([fit1, fit_ols])
```

	Dependent variable:sdvol1	
	(1)	(2)
Intercept	0.052***	0.046***
	(0.004)	(0.000)
Topic_10_n_ol		0.672***
		(0.207)
Topic_11_n_ol		-1.218***
		(0.259)
Topic_12_n_ol		-0.031
		(0.295)
Topic_13_n_ol		0.537
		(0.811)
Topic_14_n_ol		-1.982***

Logit

- Same idea as with OLS, replacing `smf.ols()` with `smf.logit()`

```
formula = 'Restate_Int ~ ' + ' + '.join(vars_topic[0:-1])  # Drop the final value to avoid multicollinearity

model = smf.logit(formula=formula, data=train)
fit_logit = model.fit()
```

```
## Optimization terminated successfully.
##          Current function value: 0.060121
##          Iterations 16
```

```
fit_logit.summary()
```

Logit Regression Results

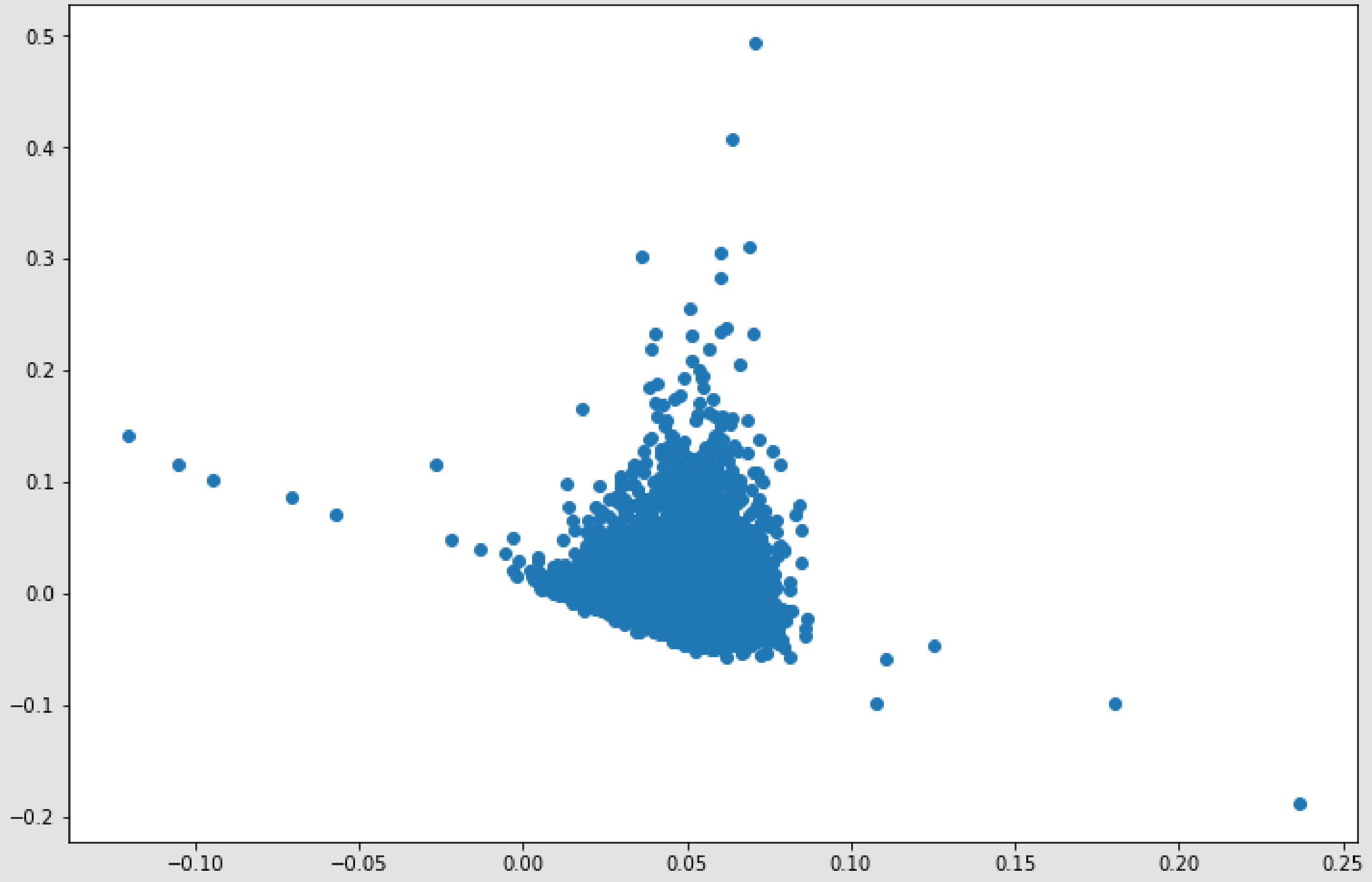
Dep. Variable:	Restate_Int	**No. Observations:**	11478
Model:	Logit	**Df Residuals:**	11447
Method:	MLE	**Df Model:**	30
Date:	Mon, 12 Jul 2021	**Pseudo R-squ.:**	0.02432
Time:	02:31:20	**Log-Likelihood:**	-690.07
converged:	True	**LL-Null:**	-707.27
Covariance Type:	nonrobust	**LLR p-value:**	0.2651

Measuring predictive performance

Getting predictions

- Most regression structures in python provide a `.predict()` method for predicting in or out of sample

```
Y_hat_train = fit_ols.predict(train)

Residual_train = train.sdvol1 - Y_hat_train
```



Linear predictive power

- 2 methods that are often used are:
  - RMSE: Root Mean Squared Error
  - MAE: Mean Absolute Error

```
rmse = metrics.mean_squared_error(train.sdvol1, Y_hat_tra
                                  squared=False)

print('RMSE: {:.4f}'.format(rmse))
```

```
## RMSE: 0.0286
```

```
mae = metrics.mean_absolute_error(train.sdvol1, Y_hat_tra

print('MAE: {:.4f}'.format(mae))
```

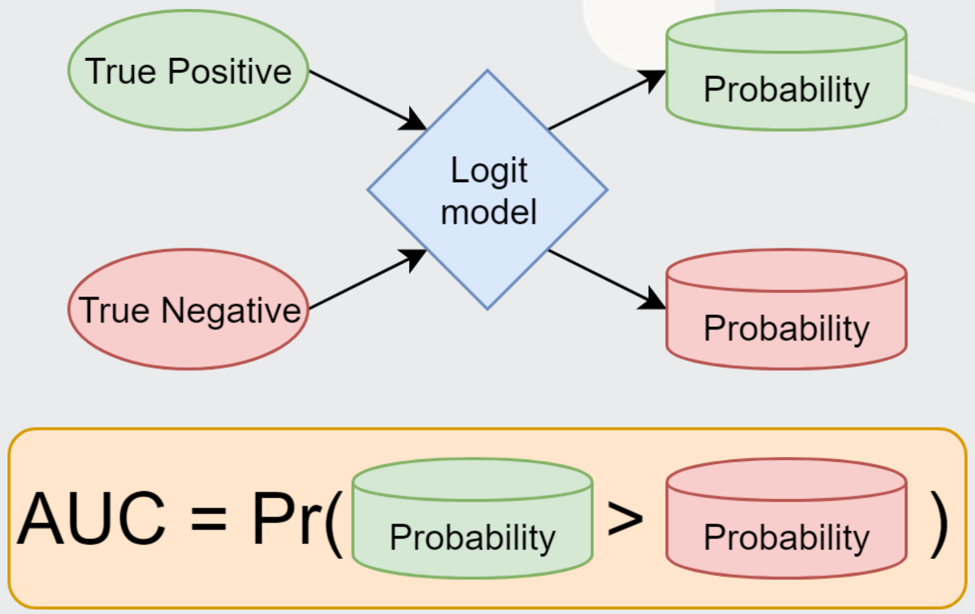
```
## MAE: 0.0191
```

Logistic predictive power

- For logistic regression, ROC AUC is a good measure

```python
Y_hat_train = fit_logit.predict(train)
auc = metrics.roc_auc_score(train.Restate_Int, Y_hat_trai
print('ROC AUC: {:.4f}'.format(auc))
```

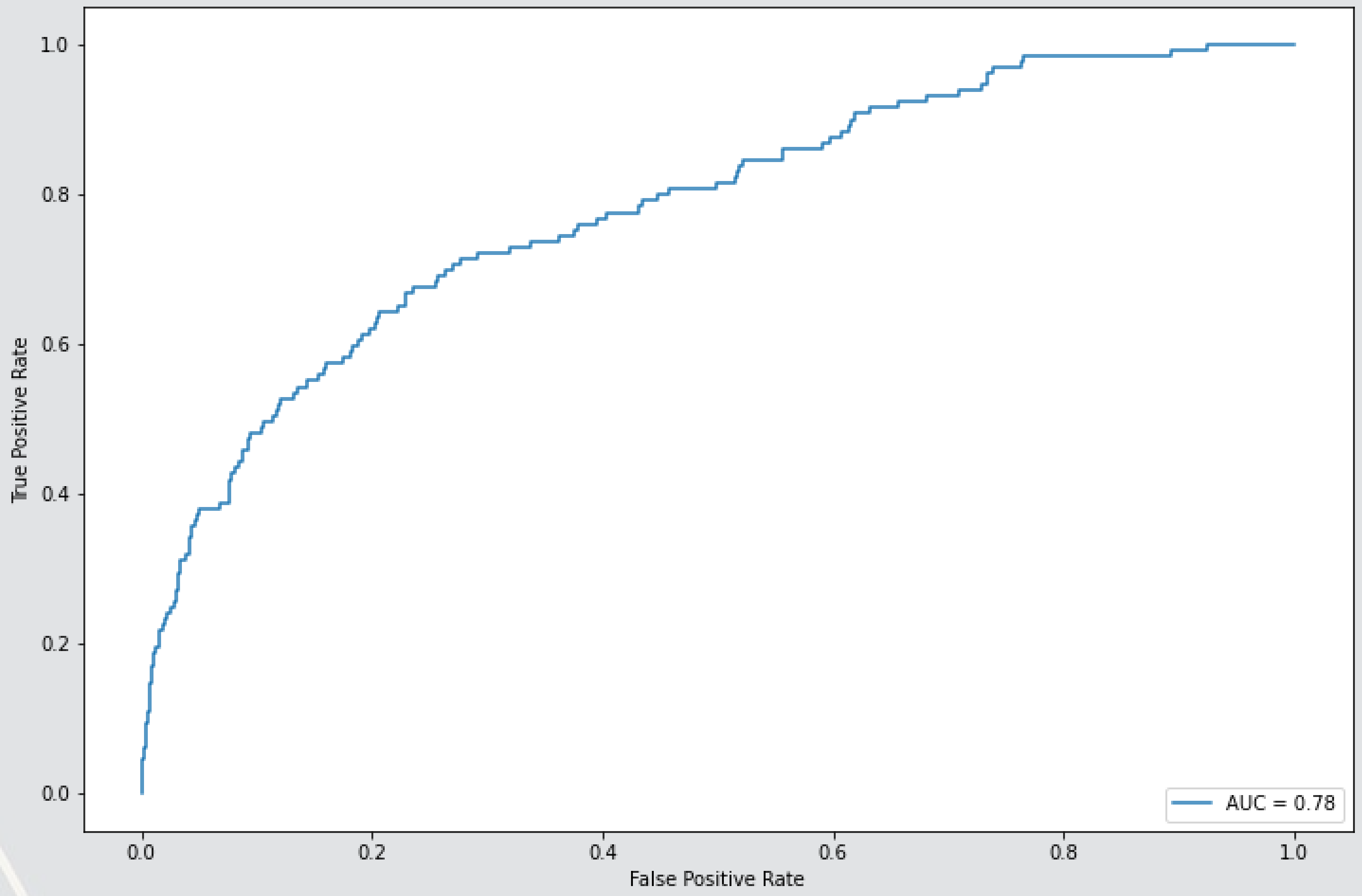
```
## ROC AUC: 0.6538
```



Visualizing AUC with the ROC curve

- `sklearn` makes it easy to output a ROC curve as well

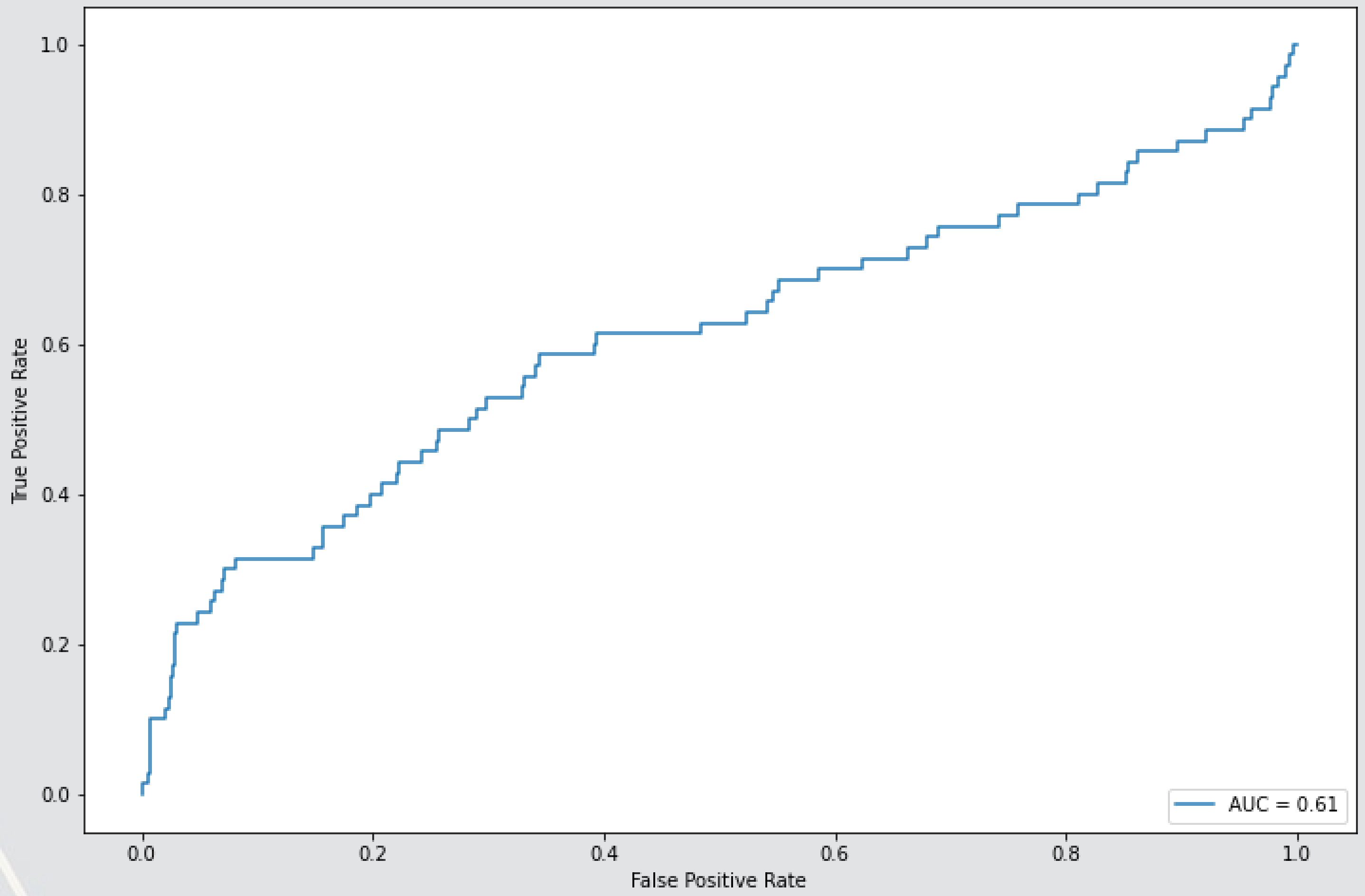
```python
# Full code to replicate -- first two lines are same as prior slide
Y_hat_train = fit_logit.predict(train)
auc = metrics.roc_auc_score(train.Restate_Int, Y_hat_train)

fpr, tpr, thresholds = metrics.roc_curve(train.Restate_Int, Y_hat_train)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=auc)
display.plot()
```



Out of sample AUC

- All we need to do is swap in `test` for `train`!

```python
# Logit, out-of-sample
Y_hat_test = fit_logit.predict(test)
auc = metrics.roc_auc_score(test.Restate_Int, Y_hat_test)

fpr, tpr, thresholds = metrics.roc_curve(test.Restate_Int, Y_hat_test)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=auc)
display.plot()
```



Fixed effects

1 or 2 fixed effect

- `statsmodels` doesn't support fixed effects, but you can add variables as categorical using `C()`

```python
# Defining the function in a variable
formula = 'sdvol1 ~ logtotasset + fog + C(year)'
model = smf.ols(formula=formula, data=train)
fit1_fe = model.fit()
fit1_fe.summary()
```

OLS Regression Results

Dep. Variable:	sdvol1	**R-squared:**	0.288
Model:	OLS	**Adj. R-squared:**	0.288
Method:	Least Squares	**F-statistic:**	774.0
Date:	Mon, 12 Jul 2021	**Prob (F-statistic):**	0.00
Time:	02:31:22	**Log-Likelihood:**	25449.
No. Observations:	11478	**AIC:**	-5.088e+04
Df Residuals:	11471	**BIC:**	-5.083e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0406	0.003	12.097	0.000	0.034	0.047

3 or more fixed effects

- `statsmodels` cannot handle HDFE
  - This has been an open issue since 2015…
- Use the `linearmodels` package instead!

What can linearmodels do?

Can do

- Anything OLS
  - Fixed effects
  - Random effects
  - HDFE/Absorbing
  - Fama-MacBeth
  - 2SLS, GM, etc.
  - 3SLS, SUR, GMM system

Cannot do

- Anything that isn't explicitly linear

Adding in HDFE

- Use `linearmodels.iv.absorbing.AbsorbingLS()` to include HDFE
- Syntax is a bit difficult – need to supply data as 3 data frames or matrices

```python
x = train[["logtotasset", "fog"]]
y = train["sdvol1"]
absorb = train[["year", "gvkey"]].copy()  # include as many FEs as needed here
absorb['year'] = absorb['year'].astype('category')
absorb['gvkey'] = absorb['gvkey'].astype('category')
model = linearmodels.iv.absorbing.AbsorbingLS(y, x, absorb=absorb)
model.fit()
```

Absorbing LS Estimation Summary

Dep. Variable:	sdvol1	**R-squared:**	0.8268
Estimator:	Absorbing LS	**Adj. R-squared:**	0.7290
No. Observations:	11478	**F-statistic:**	95.219
Date:	Mon, Jul 12 2021	**P-value (F-stat):**	0.0000
Time:	02:31:23	**Distribution:**	chi2(2)
Cov. Estimator:	robust	**R-squared (No Effects):**	0.0168
		Varaibles Absorbed:	4142.0

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
logtotasset	-0.0062	0.0007	-8.8599	0.0000	-0.0076	-0.0048
fog	0.0007	0.0002	3.8611	0.0001	0.0003	0.0010

Caveats

- `stargazer` doesn't play nicely with `linearmodels`
- `linearmodels` *only* handles linear cases – it can't handle other GLM structures
  - E.g., you can't do Logit, Poisson, or Cox with it
    - So Stata is more flexible for HDFE models

Addendum: Using R

- In R, HDFE regression is handled quite well by `fixest`
  - Supports many structural forms (OLS, Poisson, Logit, Negative binomial)
  - **Fast** – in some case completing in less than 1% of the time needed by Stata
  - Also supports clustering of standard errors
  - Has a summarization method, `etable()`, that parallels `estout` and `outreg2`
  - Supports IV/2SLS
  - Supports interactions between fixed effects and other fixed effects or IVs.
  - Supports *unbiased* staggered DID (following Sun and Abraham (2020 JE))

> If you need complicated econometrics, R or Stata is better

What about ML for panel data?

Problems of the prior approach

- For both linear and logistic regression:
  - Too many covariates
    - Probably high VIFs
    - Multicollinearity is quite high
- For logit:
  - Convergence is iffy when using sparse datasets or DVs

> How can machine learning help?

1. Some methods directly adress the issues of multicollinearity or having too many covariates (via model selection)
2. Some methods address sparsity well, being robust to binary DVs with sub 10% classes

What is LASSO?

- **L**east **A**bsolute **S**hrinkage and **S**election **O**perator
  - Least absolute: uses an error term like $|\varepsilon|$
  - Shrinkage: it will make coefficients smaller
    - Less sensitive → less overfitting issues
  - Selection: it will completely remove some variables
    - Less variables → less overfitting issues
- Sometimes called L_1 regularization
  - L_1 means 1 dimensional distance, i.e., $|\varepsilon|$

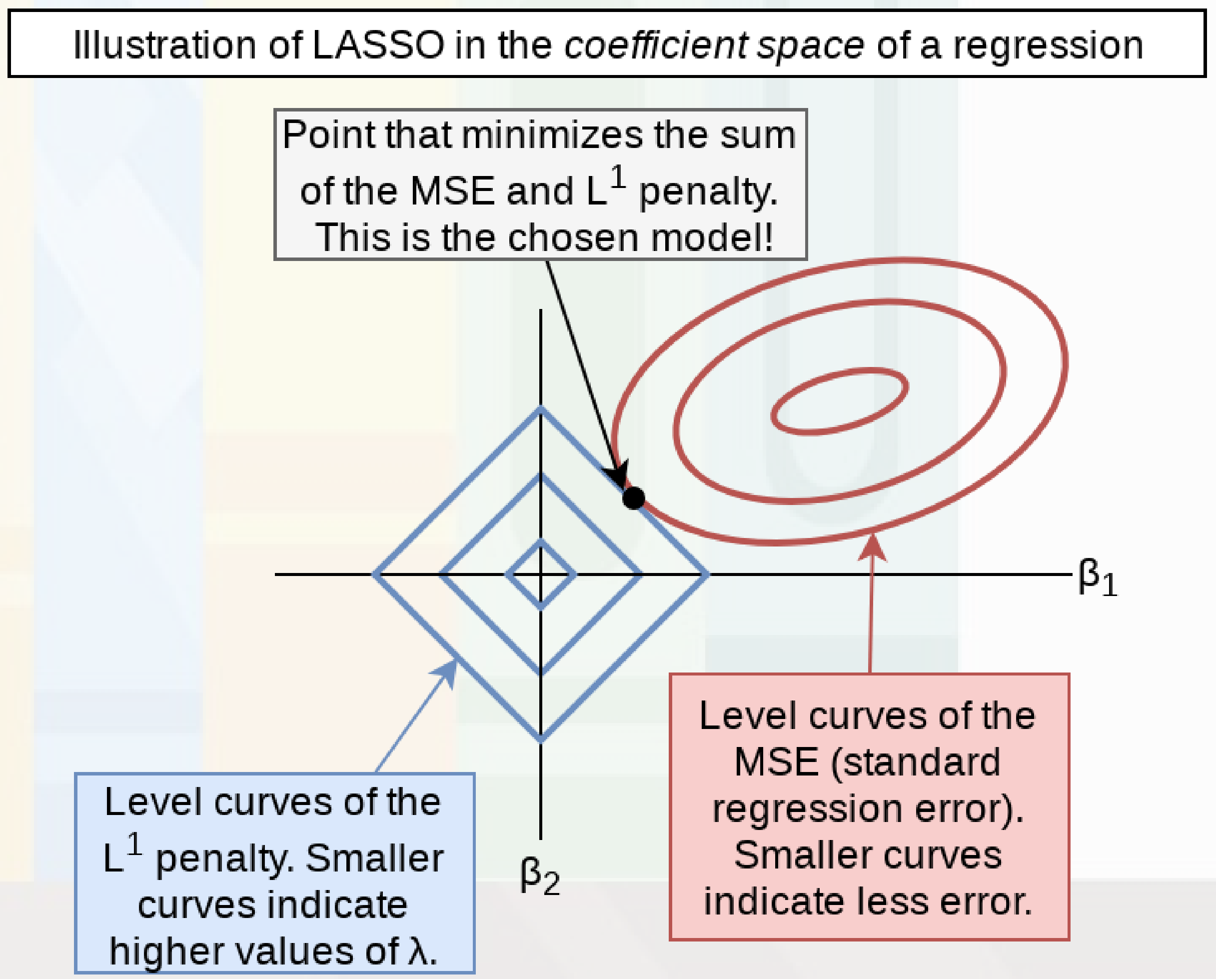
> Great if you have way too many inputs in your model or high multicollinearity

- Note that L^1 regularization is a standard approach to dealing with inflated VIFs as well!

How does it work?

$$\min_{\beta \in \mathbb{R}} \left\{ \frac{1}{N} |\varepsilon|_2^2 + \lambda |\beta|_1 \right\}$$

- Add an additional penalty term that is increasing in the absolute value of each β
  - Incentivizes lower βs, *shrinking* them
- The selection is part is explainable geometrically in 2D
  - If the MSE level curves hit a corner of the diamond shaped penalty curve, then a coefficient is set to 0 and dropped



LASSO example: Restaurant pricing

From Chahuneau et al. (2012 EMNLP)

- The paper uses a large data set on menu information from `www.allmenus.com` to predict:
  1. Menu item prices
  2. Price range for a restaurant (categorical)
  3. Median price and sentiment for a restaurant.
- Uses L_1 regularization
- Optimizes MAE and MRE (Mean Relative Error – MAE where each observation's error is scaled by y_i)

City	# Restaurants			# Menu Items			# Reviews		
	train	dev.	test	train	dev.	test	train	dev.	test
Boston	930	107	113	63,422	8,426	8,409	80,309	10,976	11,511
Chicago	804	98	100	51,480	6,633	6,939	73,251	9,582	10,965
Los Angeles	624	80	68	17,980	2,938	1,592	75,455	13,227	5,716
New York	3,965	473	499	365,518	42,315	45,728	326,801	35,529	37,795
Philadelphia	1,015	129	117	83,818	11,777	9,295	52,275	7,347	5,790
San Francisco	1,908	255	234	103,954	12,871	12,510	499,984	59,378	67,010
Washington, D.C.	773	110	121	47,188	5,957	7,224	71,179	11,852	14,129
Total	10,019	1,252	1,252	733,360	90,917	91,697	1,179,254	147,891	152,916

Table 1: Dataset statistics.

Menu pricing

$$\begin{aligned} log(price) = \quad & \alpha + \beta \cdot MENU\ NAMES + \gamma \cdot MENU\ DESC + \delta \cdot METADATA + \\ & \zeta \cdot MENTIONS + \eta \hat{PR} + \varepsilon \end{aligned}$$

- $MENU\ NAMES$: n-grams (1, 2, 3) of the name of the item on the menu
- $MENU\ DESC$: n-grams of item descriptions
- $METADATA$: “location (city, neighborhood, transit stop), services available (take-out, delivery), wifi, parking, etc.), and ambience (good for groups, noise level, attire, etc.).” Also included was food type and user rating (1-5 stars). All of these are one-hot encoded (i.e., turned into indicator variables)
- $MENTIONS$: n-grams from reviews where the menu item matched best
- $\hat{PR}$: The prediction from a model without menu or mention text included

Menu pricing

- The full model has 4,959,488 variables
- There are only 733,360 observations in the data set

> How is it possible to run this regression?

- This is another advantage of LASSO
  - It's a bit like running a simulation for variable selection, and thus it can optimize the included coefficients down to a feasible set
  - The LASSO model output retains only 458,462 features – less than 10%!

Final result?

- The final algorithm using LASSO is off by $3.06 USD on average of the actual price (~34%)
- The best non-LASSO algorithm in the paper is off by $3.67 USD on average (~43%)

> Some interesting findings by measure category

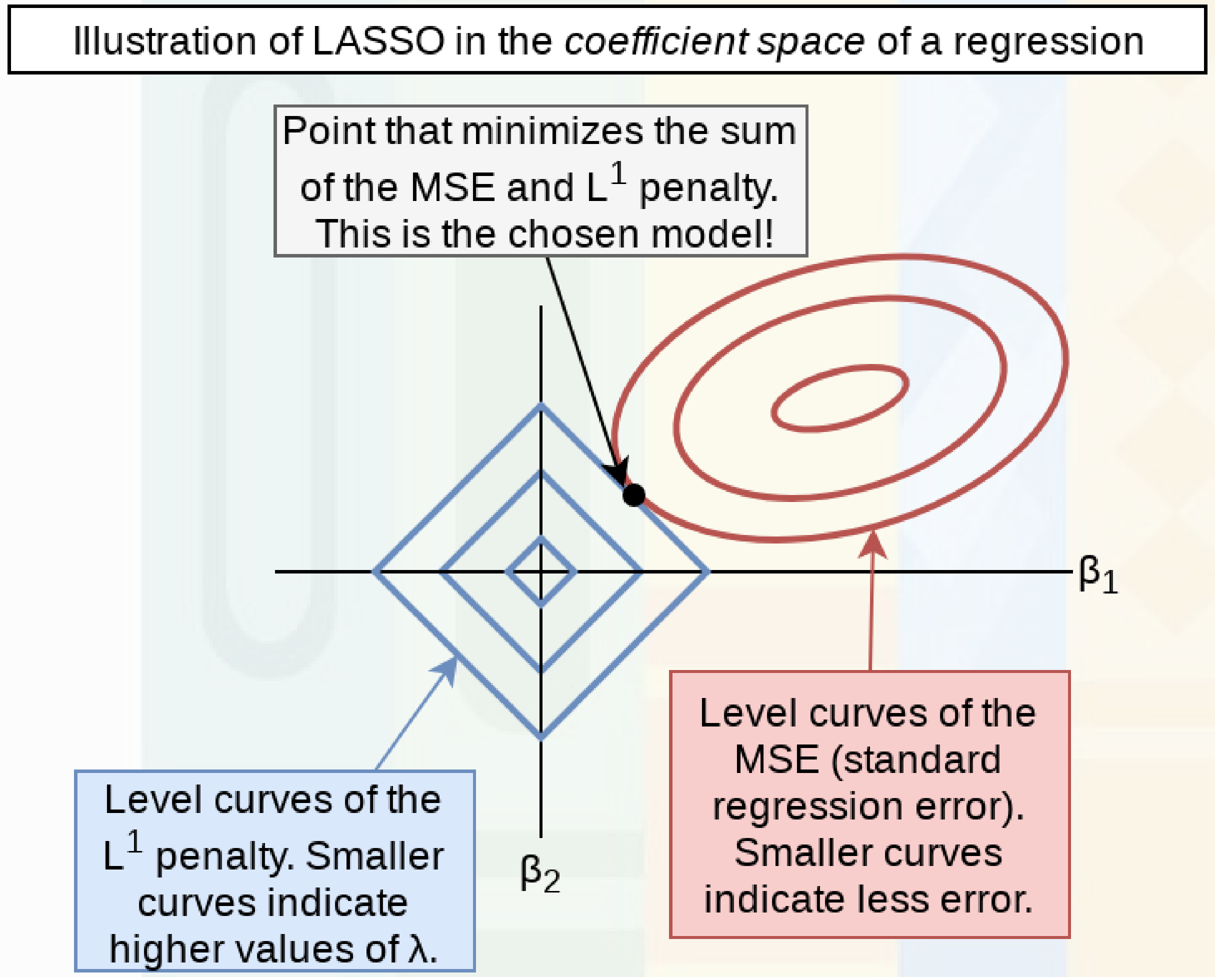
category	Cheapest	Most.expensive
Metadata, ambience	dive-y	upscale; touristy
Menu Desc, cooking	panfried; chargrilled	flamebroiled
Menu Desc, descriptors	old time favorite	farmhouse
Menu Desc, “of chicken”	slices of chicken	cuts of chicken
Menu Desc, “potatoes”	real mashed potatoes	smooth mached potatoes
Menu Desc, “roast” and “roasted”	roasted chicken	roast salmon

Restaurant pricing prediction

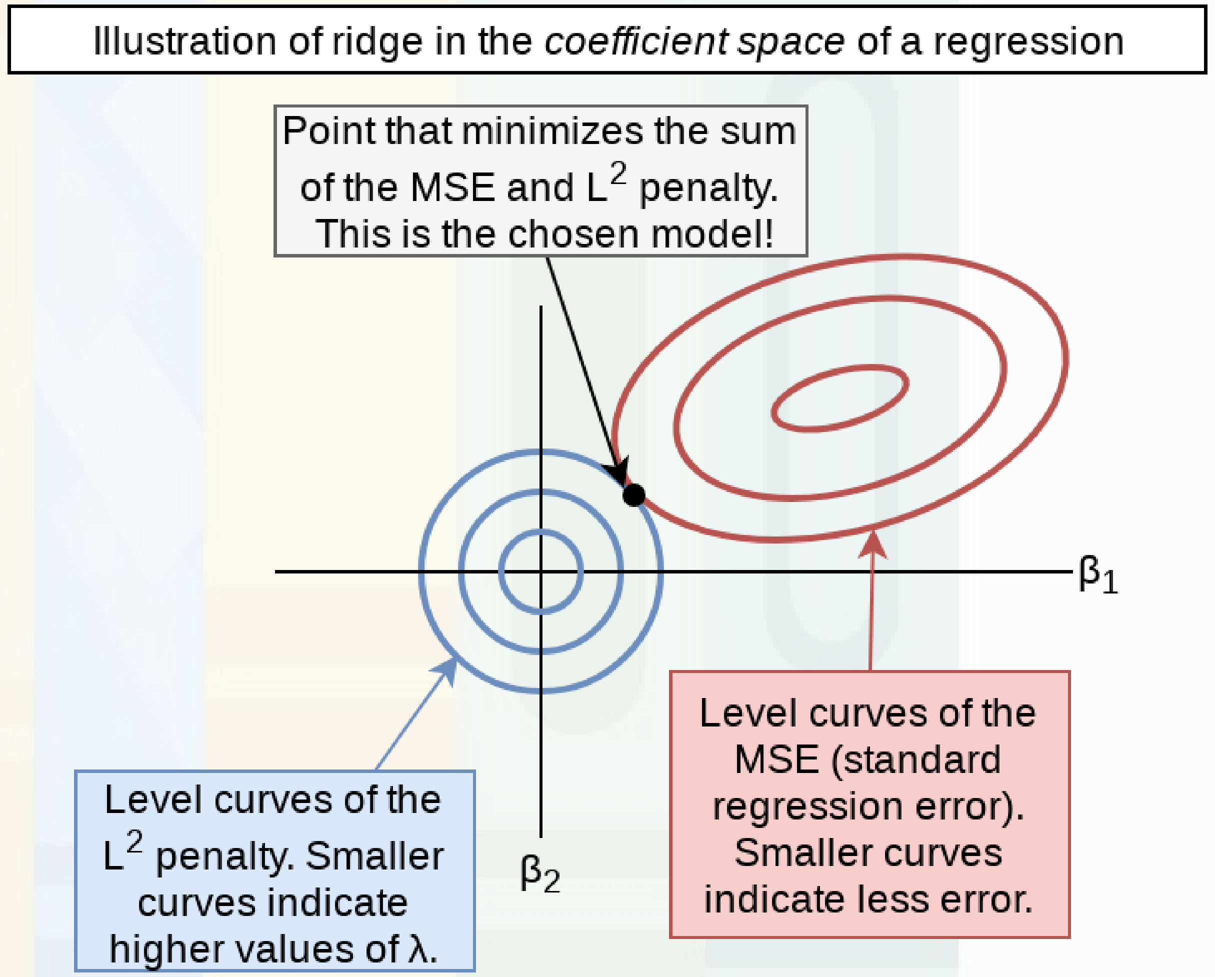
- This uses the same data, but tries to predict the restaurant's category ('$' through '$$$$')
- The simple, univariate model achieves only 48.22% accuracy
- A LASSO model including Reviews and restaurant metadata (3,027,943 features, 1,376 retained) achieves 80.36% accuracy

What about other penalty types?

LASSO (L_1)



Ridge (L_2)



LASSO:

- Decreases coefficient values
  - Makes many of them 0
  - Increases prediction stability

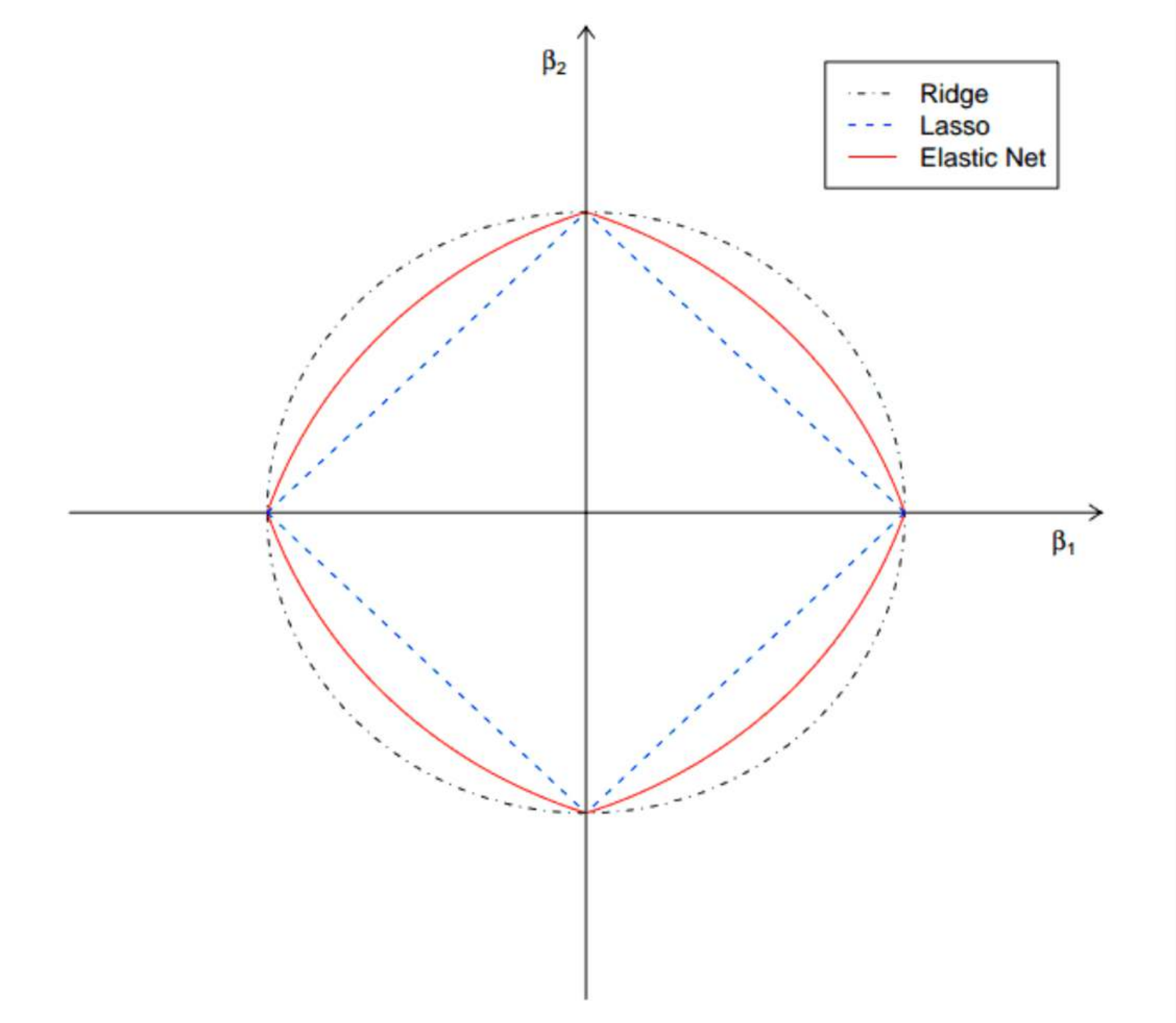
Ridge:

- Decreases coefficient values
  - Increases prediction stability more
  - Less sensitive to outliers

Combining LASSO and Ridge: Elastic Net

- Elastic Net has both L_1 *and* L_2 penalties!
- Allows you to optimize the amount of selection effect you want from LASSO and the amount of shrinkage from Ridge
- A generalization of LASSO and Ridge

$$\min_{\beta\in\mathbb{R}}\left\{\frac{1}{N}|\varepsilon|_2^2+\lambda_1|\beta|_1+\lambda_2||\beta||^2\right\}$$



Implementing LASSO in Python

Setting up to use Scikit-Learn

- Scikit-learn, like many machine learning packages, expects separate data sets or matrices for DVs and IVs
  - We saw this earlier with `linearmodels` as well
- LASSO, Ridge, and Elastic net are also particular about data format:

> Every input should be normalized to a Z-score!

- Scikit-learn has this all built in, so it will be easy

```python
vars = vars_topic
scaler_X = preprocessing.StandardScaler()
scaler_X.fit(train[vars])
```

```python
train_X_linear = scaler_X.transform(train[vars])
test_X_linear = scaler_X.transform(test[vars])
```

- `sklearn.preprocessing.StandardScaler()` defaults to transforming to Z-scores
- Applying `.fit()` with data makes it calculate the mean and standard deviation of each column
- Applying `.transform()` with data applies the Z-score based on the fitted parameters
  - Avoids any look-ahead bias in our testing sample!

Setting up to use Scikit-Learn

```python
scaler_Y = preprocessing.StandardScaler()
scaler_Y.fit(np.array(train.sdvol1).reshape(-1, 1))
```

```python
train_Y_linear = scaler_Y.transform(np.array(train.sdvol1).reshape(-1, 1))
test_Y_linear = scaler_Y.transform(np.array(test.sdvol1).reshape(-1, 1))
```

- Inputs are required to be 2D matrices by `sklearn`
- The `np.array(_____).reshape(-1, 1)` bit is to cast the Pandas series back into a 2D matrix - `np.array()` casts the pandas series object to an array (matrix), but it is only 1D
  - `.reshape(-1,1)` forces the matrix to be a column (and thus 2D) instead of a 1D row matrix

Simple LASSO, linear

- Fitting a LASSO with a pre-specified penalty is quite easy

```
reg_lasso = linear_model.Lasso(alpha=0.1)
reg_lasso.fit(train_X_linear, train_Y_linear)
```

```
## Lasso(alpha=0.1)
```

- Seeing the result is not

Coerce the data

```
print('\n'.join([str(i) for i in
                 zip(vars, list(reg_lasso.coef_))]))
```

```
## ('Topic_1_n_oI', 0.0)
## ('Topic_2_n_oI', -0.0)
## ('Topic_3_n_oI', -0.0)
## ('Topic_4_n_oI', 0.0)
## ('Topic_5_n_oI', 0.0)
## ('Topic_6_n_oI', -0.0)
## ('Topic_7_n_oI', -0.024652670717254)
## ('Topic_8_n_oI', 0.0)
## ('Topic_9_n_oI', 0.0025216975893077123)
## ('Topic_10_n_oI', -0.0)
```

Custom coefficient plot function

```
coefplot(vars, reg_lasso.coef_)
```

Simple LASSO, logistic

- Instead of using `sklearn.linear_model.Lasso()`…
  - Use `sklearn.linear_model.LogisticRegression()`
- This function has options for L_1, L_2, or both penalties together
  - Thus, it supports LASSO, Ridge, and Elastic net, respectively

Prep the data

```
vars = vars_topic + vars_financial + vars_style
scaler_X = preprocessing.StandardScaler()
scaler_X.fit(train[vars])
```

```
## StandardScaler()
```

```
train_X_logistic = scaler_X.transform(train[vars])
test_X_logistic = scaler_X.transform(test[vars])

train_Y_logistic = train.Restate_Int
test_Y_logistic = test.Restate_Int
```

Simple LASSO, logistic

```
reg_lasso = linear_model.LogisticRegression(penalty='l1', solver='saga', C=0.1)
reg_lasso.fit(train_X_logistic, train_Y_logistic)
```

```
## LogisticRegression(C=0.1, penalty='l1', solver='saga')
```

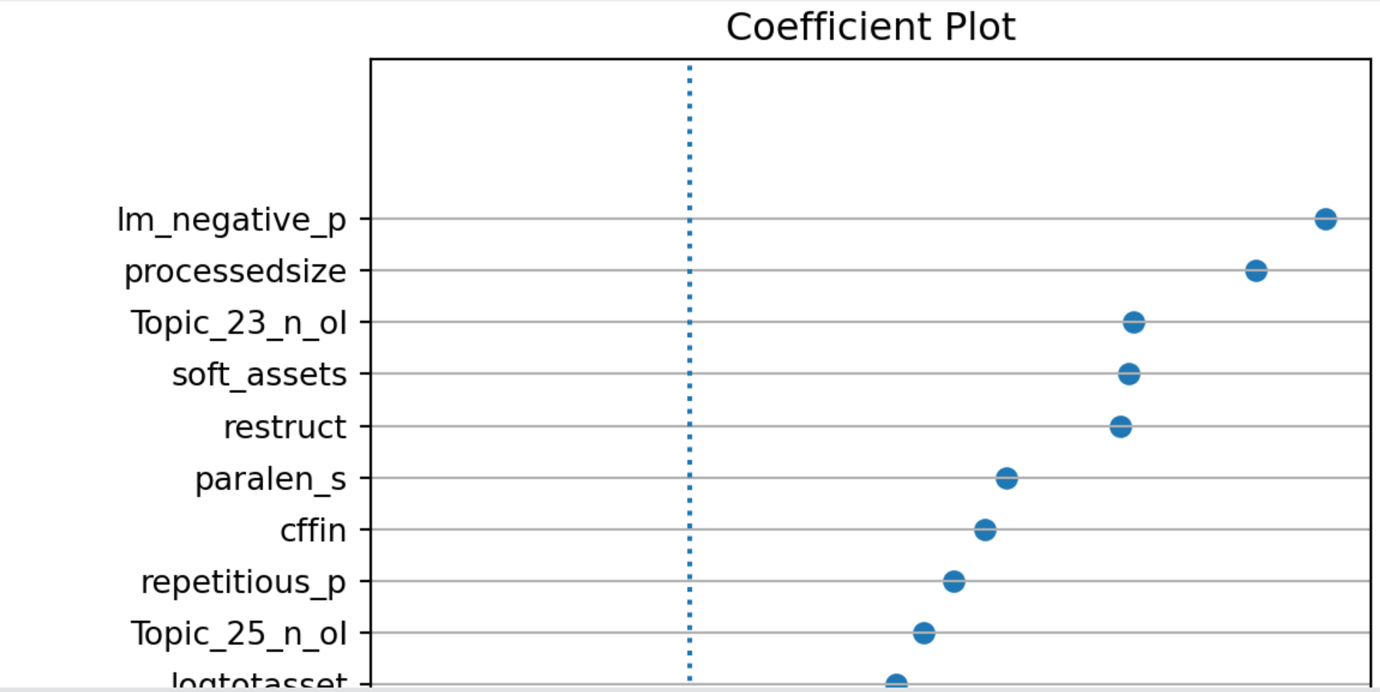
Coerce the data

```
print('\n'.join([str(i) for i in
                 zip(vars, list(reg_lasso.coef_[0]))]))
```

```
## ('Topic_1_n_oI', -0.025128726785553102)
## ('Topic_2_n_oI', 0.0)
## ('Topic_3_n_oI', 0.0)
## ('Topic_4_n_oI', 0.0)
## ('Topic_5_n_oI', 0.0)
## ('Topic_6_n_oI', 0.0)
## ('Topic_7_n_oI', 0.0)
## ('Topic_8_n_oI', -0.0001299021019774882)
## ('Topic_9_n_oI', 0.0)
## ('Topic_10_n_oI', 0.0)
## ('Topic_11_n_oI', -0.042083026063157634)
## ('Topic_12_n_oI', -0.017223345281073166)
## ('Topic_13_n_oI', 0.0)
## ('Topic_14_n_oI', 0.0)
```

Custom coefficient plot function

```
coefplot(vars, reg_lasso.coef_)
```



Cross Validation

What is cross validation?

- Validation is where you keep part of the training sample as a hold out sample to evaluate and improve your algorithm against
  - This prevents biasing towards the real hold out sample (the testing sample)
- Cross validation takes this further by making a bunch of validation samples,
- An example of 10-fold cross validation:
  1. Randomly splits the data into 10 groups
  2. Runs the algorithm on 90% of the data ($10 - 1 = 9$ groups)
  3. Determines the best model based on the performance of the group that was left out
  4. Repeat steps 2 and 3 $10 - 1 = 9$ more times
  5. Uses the best overall model across all 10 hold out samples

> Scikit-learn has this built in!

10-fold CV LASSO, linear

```python
reg_lasso = linear_model.LassoCV(cv=10)
reg_lasso.fit(train_X_linear, np.ravel(train_Y_linear))
```

```
## LassoCV(cv=10)
```

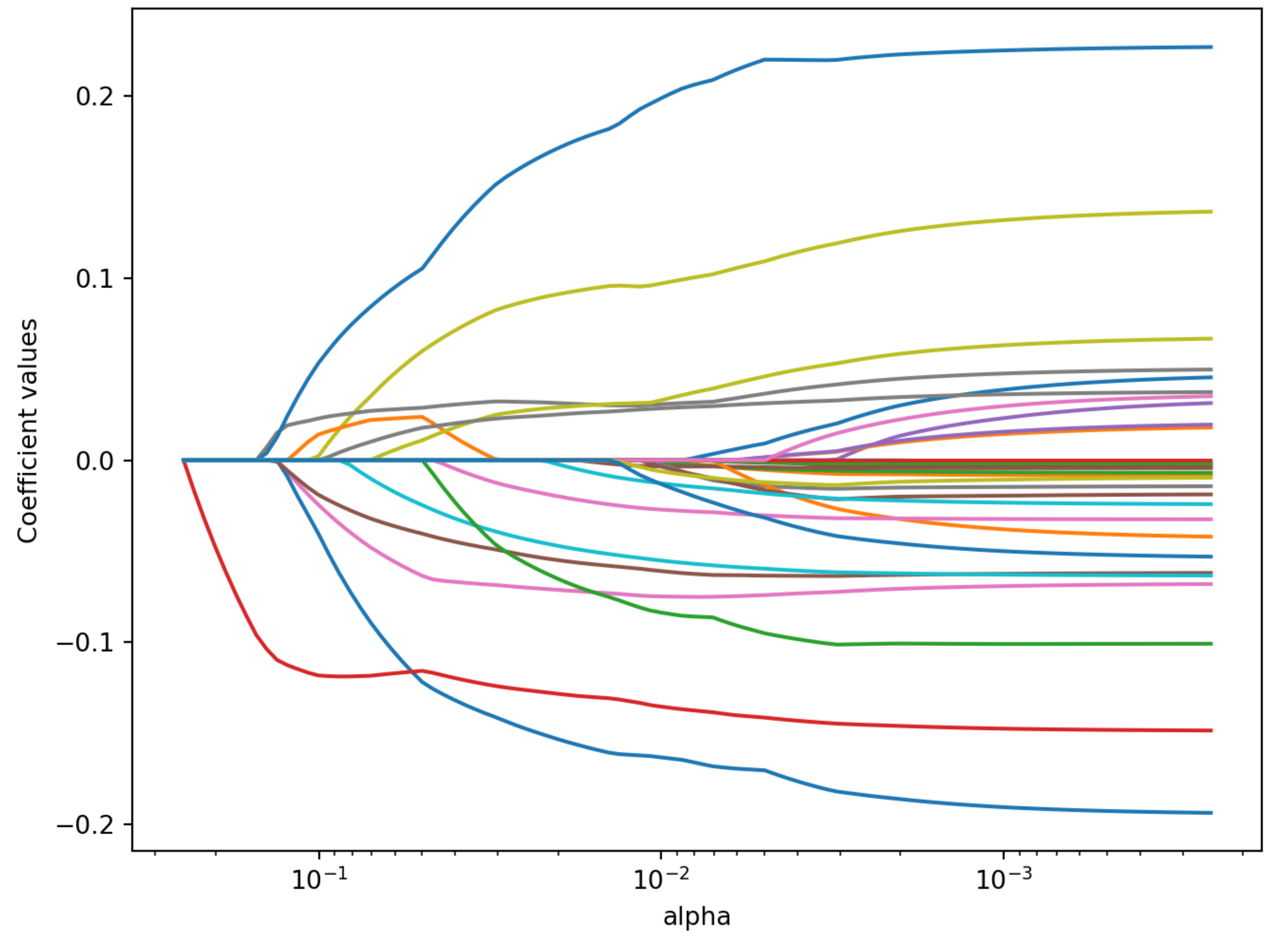
```python
print('The alpha that optimizes R^2 is: {}'.format(reg_lasso.alpha_))
```

```
## The alpha that optimizes R^2 is: 0.018778122679424136
```

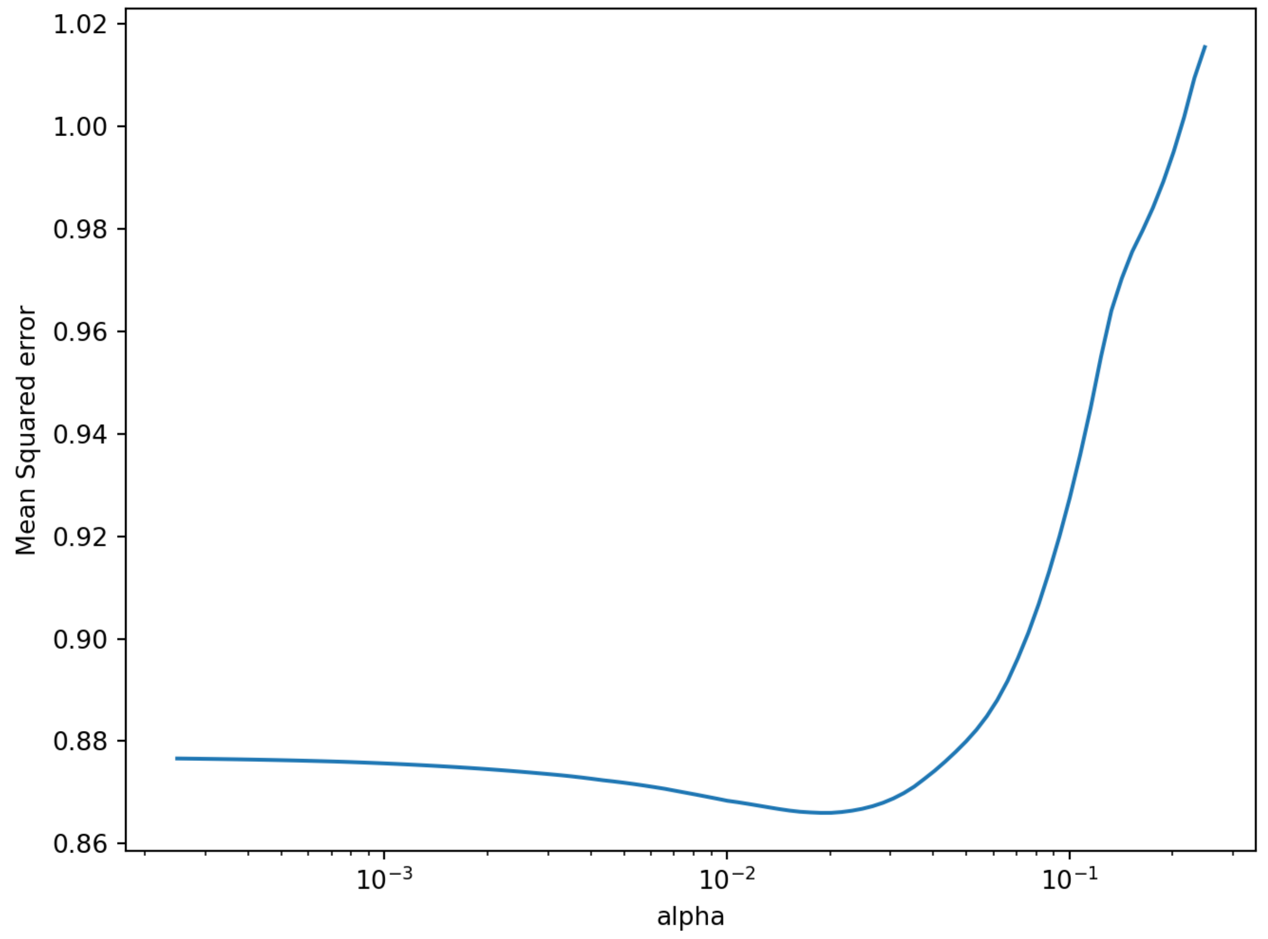
```python
coefplot(vars, reg_lasso.coef_)
```

How did the optimization work?

```
lasso_coefpath(reg_lasso, train_X_linear, train_Y_linear)
```




```
lasso_scorepath(reg_lasso, errorbars=False)
```



5-fold CV LASSO, logistic

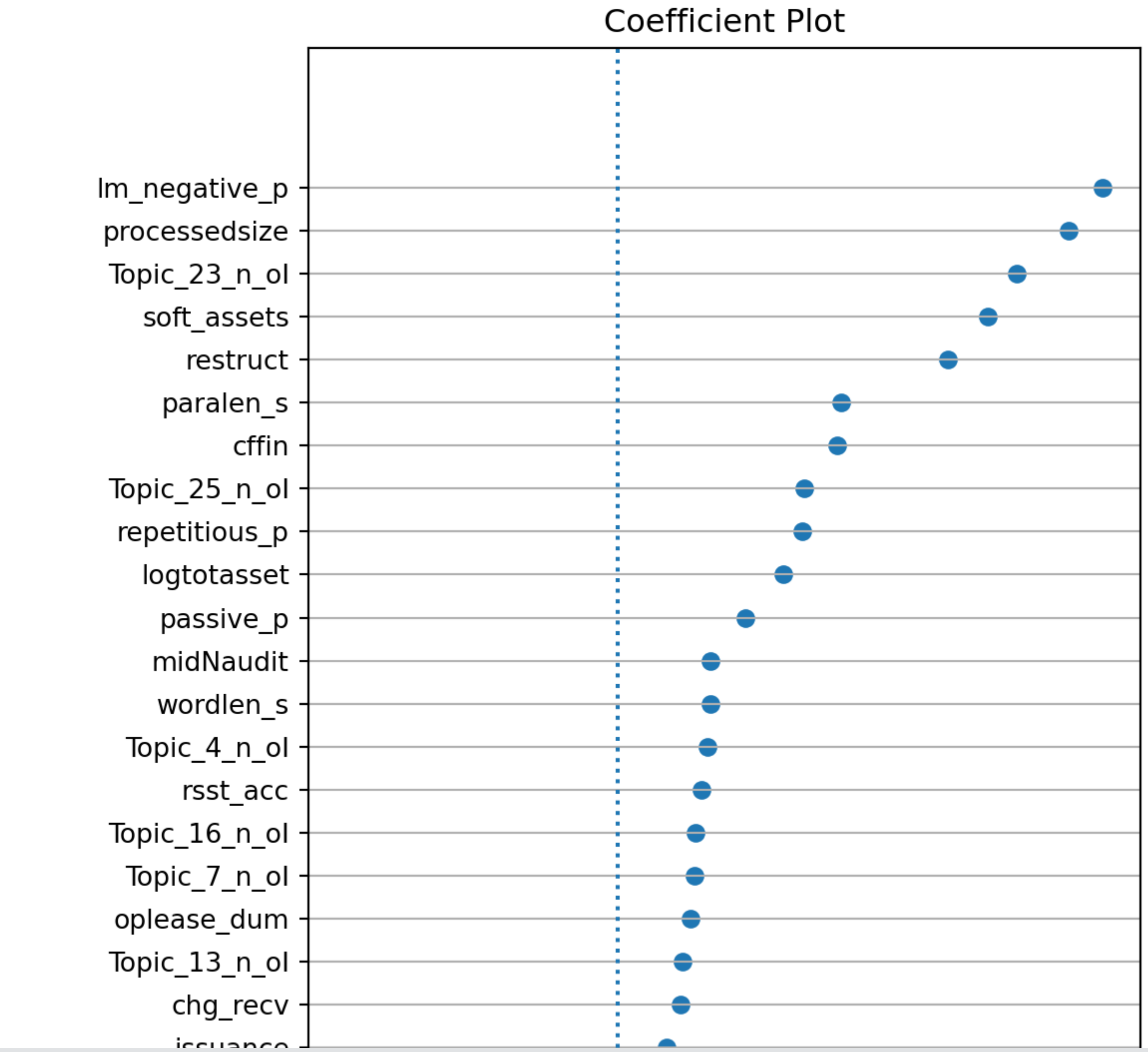
```python
reg_lasso = linear_model.LogisticRegressionCV(
  penalty='l1', solver='saga', Cs=10, cv=5, scoring="roc_a

reg_lasso.fit(train_X_logistic, train_Y_logistic)
```

```
## LogisticRegressionCV(cv=5, penalty='l1', scoring='roc_a
```

```python
print('The C that optimizes ROC AUC is: {}'.format(reg_la
```

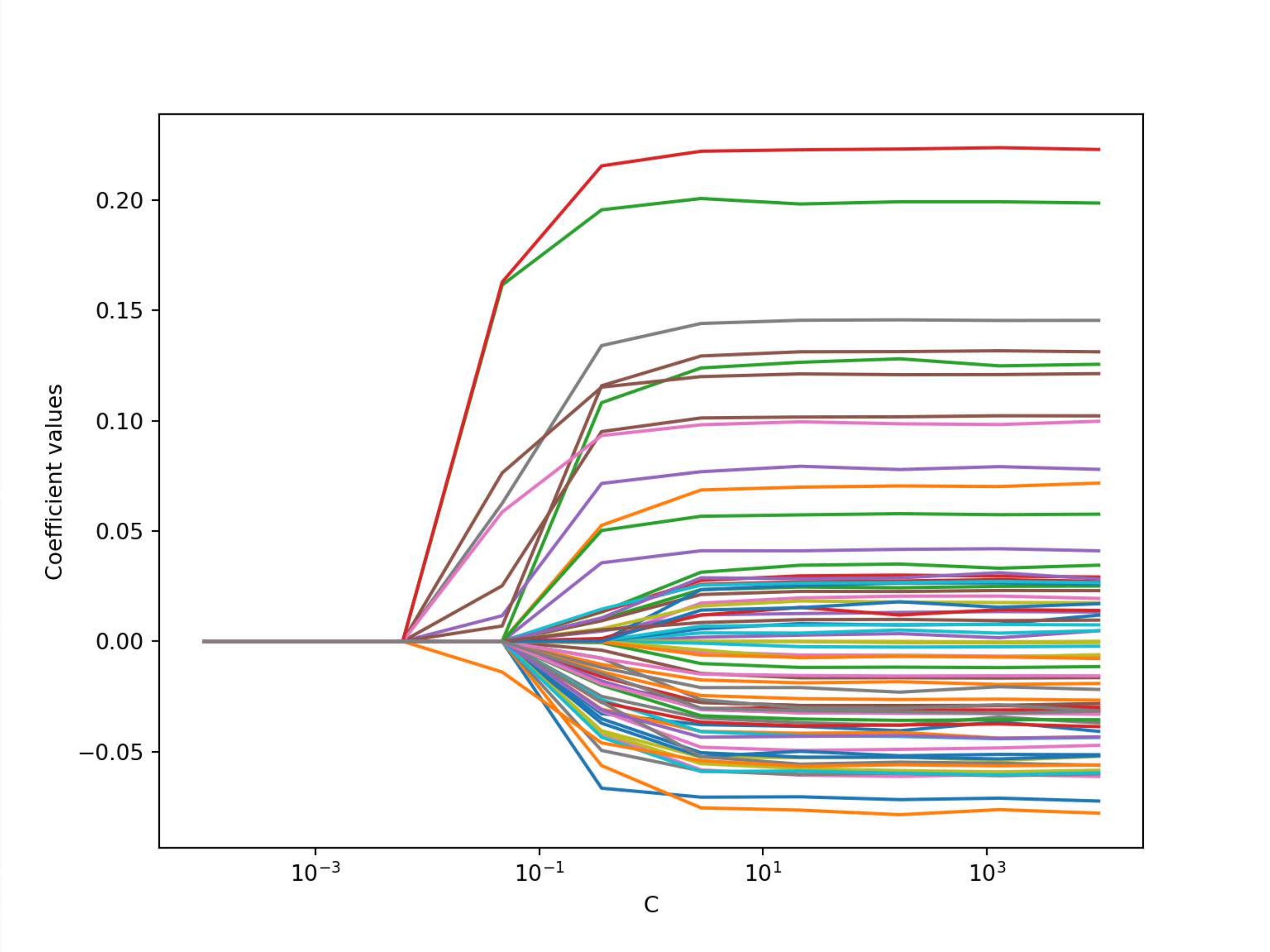
```
## The C that optimizes ROC AUC is: [2.7825594]
```

```python
coefplot(vars, reg_lasso.coef_)
```

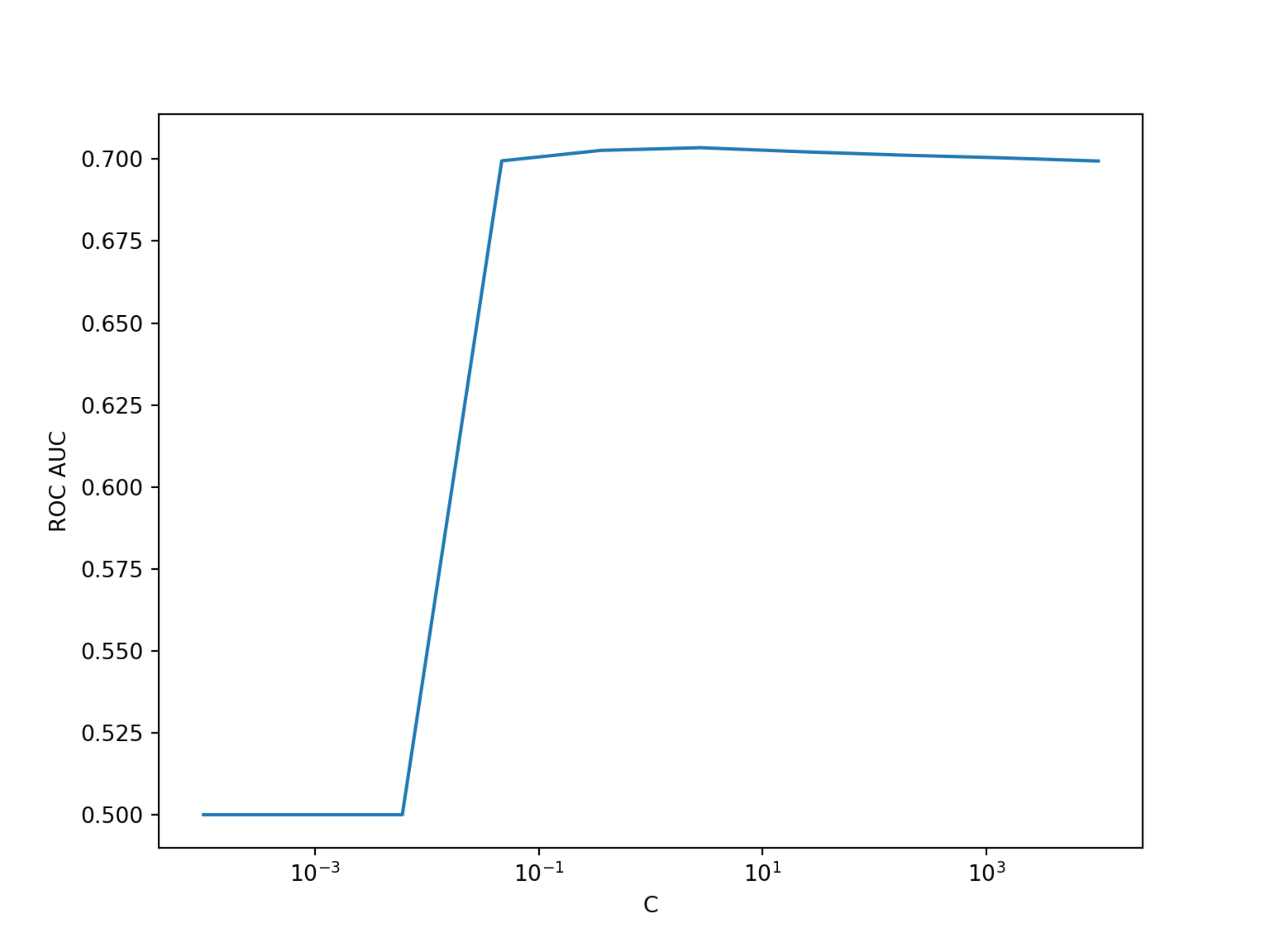


How did the optimization work?

```
lasso_coefpath(reg_lasso, train_X_logistic, train_Y_logis
```




```
lasso_scorepath(reg_lasso, errorbars=False)
```



Addendum: Using R

- In R, `glmnet` can do everything presented in this section and more!
  - It is also faster in terms of computation time
  - It can fit any base GLM family in R
- To replicate our linear LASSO:

```r
cvfit <- cv.glmnet.fit(train_X_linear, train_Y_linear, k=10, lambda=1)
plot(cvfit)
coefplot(cvfit, lambda='lambda.min', sort='magnitude')
```

- To replicate our logistic LASSO:

```r
cvfit <- cv.glmnet.fit(train_X_logistic, train_Y_logistic, k=10, lambda=1,
                       family='binomial', type.measure="auc")
plot(cvfit)
coefplot(cvfit, lambda='lambda.min', sort='magnitude')
```

Implementing Elastic net in Python

10-fold CV elastic net, linear

- Need to specify values to examine for the ratio between L_1 and L_2 penalty
  - `l1_ratio=1` is a LASSO, `l1_ratio=0` is Ridge, in between is elastic net

```
reg_EN = linear_model.ElasticNetCV(cv=10, l1_ratio=[.1, .5, .7, .9, .95, .99, 1])
reg_EN.fit(train_X_linear, np.ravel(train_Y_linear))
```

```
## ElasticNetCV(cv=10, l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1])
```

```
print('Optimal R^2 at l1_ratio of {} and alpha of {:.4f}'.format(reg_EN.l1_ratio_,reg_EN.alpha_))
```

```
## Optimal R^2 at l1_ratio of 0.5 and alpha of 0.0376
```

```
coefplot(vars, reg_EN.coef_)
```

5-fold CV elastic net, logistic

```
reg_EN = linear_model.LogisticRegressionCV(
  penalty='elasticnet', solver='saga', Cs=5, cv=5,
  scoring="roc_auc", l1_ratios=[.96, .97, .98, .99, 1])
reg_EN.fit(train_X_logistic, train_Y_logistic)
```

```
## LogisticRegressionCV(Cs=5, cv=5, l1_ratios=[0.96, 0.97
##                      penalty='elasticnet', scoring='ro
```

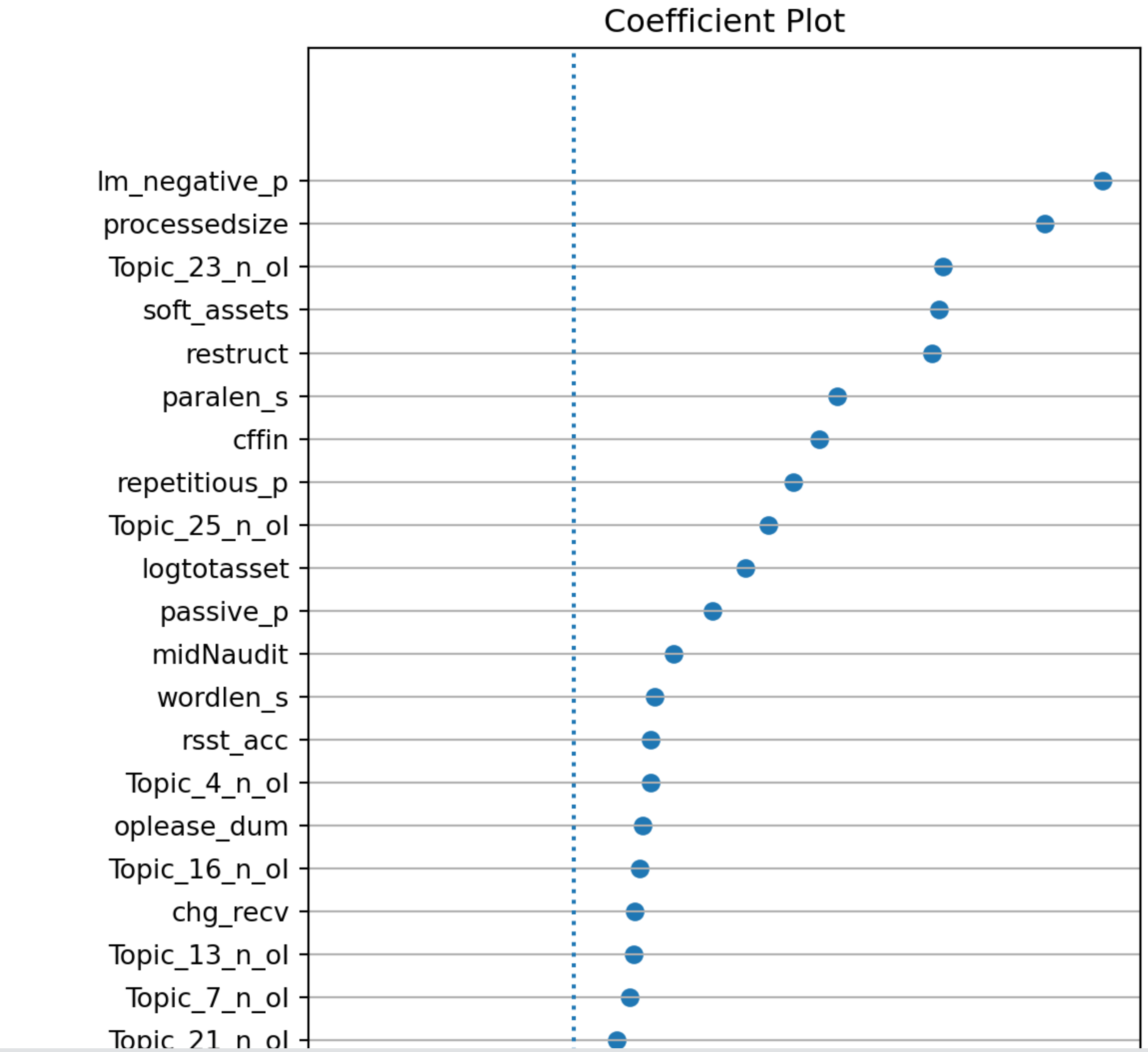
```
print('The l1_ratio that optimizes ROC AUC is {}'.format(
  reg_EN.l1_ratio_[0]))
```

```
## The l1_ratio that optimizes ROC AUC is 0.96
```

```
print('The C that optimizes ROC AUC is {:.4f}'.format(
  reg_EN.C_[0]))
```

```
## The C that optimizes ROC AUC is 1.0000
```

```
coefplot(vars, reg_EN.coef_)
```



Addendum: Using R

- In R, `glmnet` can do this too
  - `lambda=1` is LASSO
  - `lambda=0` is Ridge
  - If `lambda` is set between 0 and 1, it's an elastic net!
- To replicate our linear LASSO:

```r
cvfit <- cv.glmnet.fit(train_X_linear, train_Y_linear, k=10, lambda=?)
plot(cvfit)
coefplot(cvfit, lambda='lambda.min', sort='magnitude')
```

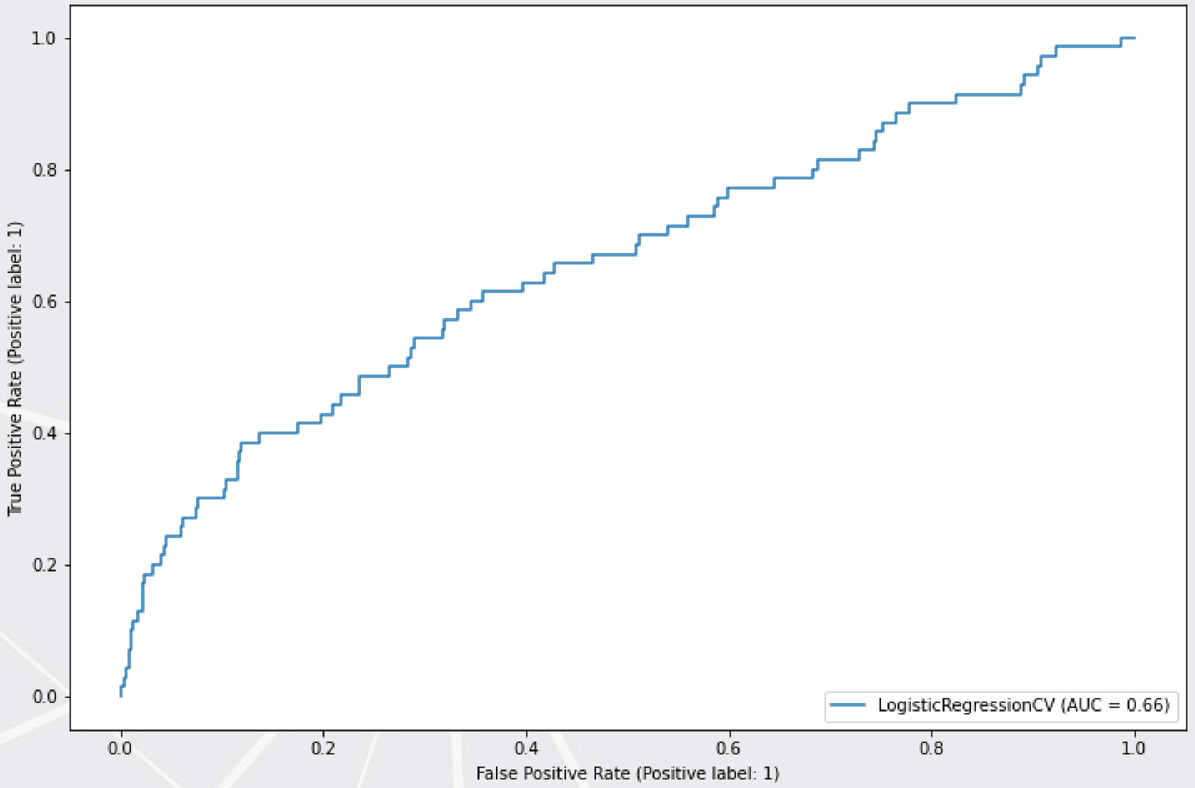
- To replicate our logistic LASSO:

```r
cvfit <- cv.glmnet.fit(train_X_logistic, train_Y_logistic, k=10, lambda=?,
                       family='binomial', type.measure="auc")
plot(cvfit)
coefplot(cvfit, lambda='lambda.min', sort='magnitude')
```

Comparing logistic model performance

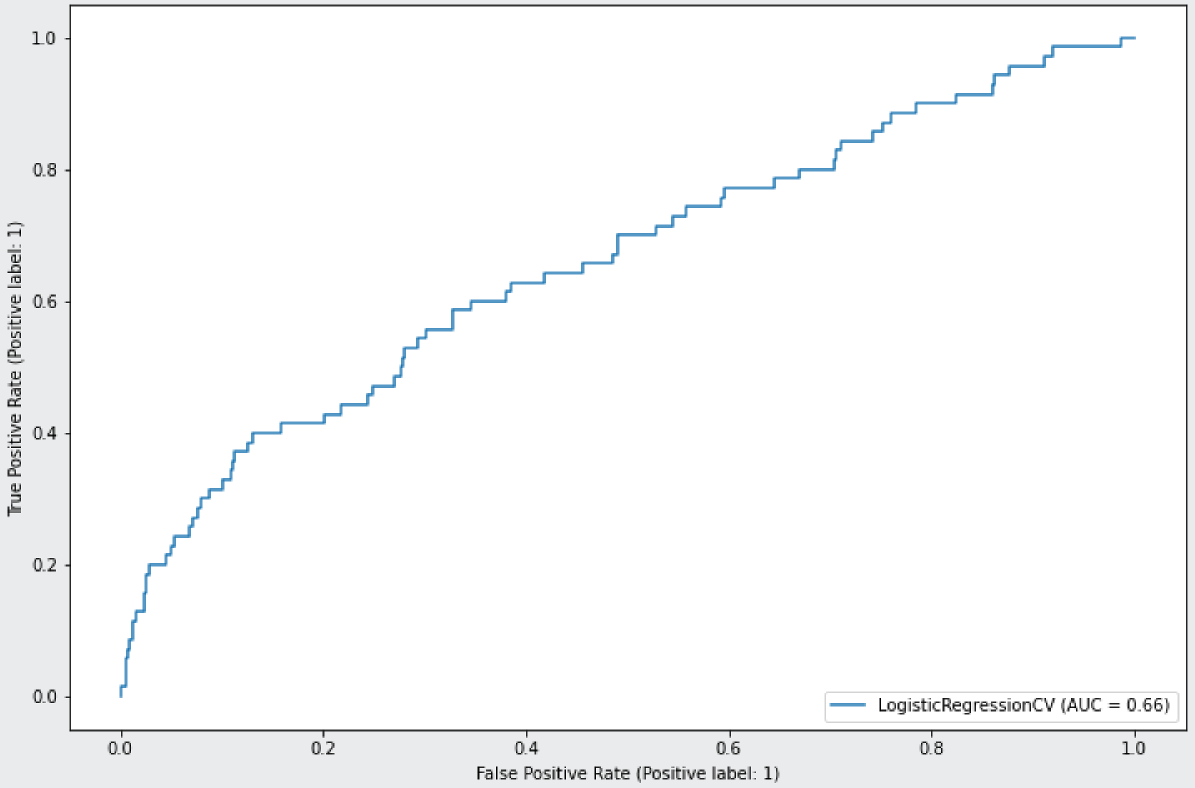
LASSO

```python
metrics.plot_roc_curve(reg_lasso, test_X_logistic,
                       test_Y_logistic)
```



Elastic net

```python
metrics.plot_roc_curve(reg_EN, test_X_logistic,
                       test_Y_logistic)
```



Conclusion

Wrap-up

Econometrics in python

- Feasible, though perhaps not the most efficient
  - R and Stata are both better for this

Machine learning regression in python (Elastic net family)

- Python is better at this
- In some circumstances, these techniques are
  - More econometrically defensible
  - More robust
  - More accurate
- R is still better for this

We will see more of these methods where python will be the best choice

Packages used for these slides

Python

- linearmodels
- matplotlib
- numpy
- pandas
- scikit-learn
- stargazer
- statsmodels

R

- kableExtra
- knitr
- reticulate
- revealjs

References

- Bao, Yang, and Anindya Datta. "Simultaneously discovering and quantifying risk types from textual risk disclosures." Management Science 60, no. 6 (2014): 1371-1391.
- Brown, Nerissa C., Richard M. Crowley, and W. Brooke Elliott. "What are you saying? Using topic to detect financial misreporting." Journal of Accounting Research 58, no. 1 (2020): 237-291.
- Chahuneau, Victor, Kevin Gimpel, Bryan R. Routledge, Lily Scherlis, and Noah A. Smith. "Word salad: Relating food prices and descriptions." In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 1357-1367. 2012.
- Sun, Liyang, and Sarah Abraham. "Estimating dynamic treatment effects in event studies with heterogeneous treatment effects." Journal of Econometrics (2020).

Custom code

```python
# Replication of R's coefplot function for use with sklearn's linear and logistic LASSO

def coefplot(names, coef, title=None):
    # Make sure coef is list, cast to list if needed.
    if isinstance(coef, np.ndarray):
        if len(coef.shape) > 1:
            coef = list(coef[0])
        else:
            coef = list(coef)

    # Drop unneeded vars
    data = []
    for i in range(0, len(coef)):
        if coef[i] != 0:
            data.append([names[i], coef[i]])
    data.sort(key=lambda x: x[1])

    # Add in a key for the plot axis
    data = [data[i] + [i+1] for i in range(0,len(data))]

    fig, ax = plt.subplots(figsize=(4,0.25*len(data)))

    ax.scatter([i[1] for i in data], [i[2] for i in data])

    ax.grid(axis='y')
    ax.set(xlabel="Fitted value", ylabel="Residual", title=(title if title is not None else "Coefficient Plot"))

    ax.axvline(x=0, linestyle='dotted')
    ax.set_yticks([i[2] for i in data])
    ax.set_yticklabels([i[0] for i in data])

    return ax
```

Custom code

```python
# Replication of R's glmnet's function plotting coefficient paths for use with sklearn's linear and logistic LASSO

def lasso_coefpath(model, X, Y):
    if 'alphas_' in dir(model):
        alphas = reg_lasso.alphas_
        coefs = []
        for a in alphas:
            temp_lasso = linear_model.Lasso(alpha=a, warm_start=True)
            temp_lasso.fit(X, Y)
            coefs.append(temp_lasso.coef_)

        fig, ax = plt.subplots()

        ax.plot(alphas, coefs)
        ax.set_xscale('log')
        ax.set_xlim(ax.get_xlim()[::-1])
        ax.set_xlabel("alpha")
        ax.set_ylabel("Coefficient values")

        return ax
    elif 'Cs_' in dir(model):
        Cs = reg_lasso.Cs_
        coefs = []
        for c in Cs:
            temp_lasso = linear_model.LogisticRegression(penalty='l1', solver='saga', C=c, warm_start=True)
            temp_lasso.fit(X, Y)
            coefs.append(temp_lasso.coef_[0])

        fig, ax = plt.subplots()

        ax.plot(Cs, coefs)
        ax.set_xscale('log')
        ax.set_xlabel("C")
        ax.set_ylabel("Coefficient values")

        return ax
    else:
        print("Does not match linear_model.LassoCV or linear_model.LogisticRegressionCV")
        return False
```

Custom code

```python
# Replication of R's glmnet's function plotting metric paths for use with sklearn's linear and logistic LASSO

def lasso_scorepath(model, errorbars=True):
    if 'alphas_' in dir(model):
        alphas = reg_lasso.alphas_
        mean = np.mean(reg_lasso.mse_path_, axis=1)
        std = np.std(reg_lasso.mse_path_, axis=1)*1.96

        fig, ax = plt.subplots()

        if errorbars:
            ax.errorbar(alphas, mean, yerr=std, ecolor="lightgray", elinewidth=2, capsize=4, capthick=2)
        else:
            ax.plot(alphas, mean)
        ax.set_xscale('log')
        ax.set_xlabel("alpha")
        ax.set_ylabel("Mean Squared error")

        return ax
    elif 'Cs_' in dir(model):
        Cs = reg_lasso.Cs_
        mean = np.mean(reg_lasso.scores_[1], axis=0)
        std = np.std(reg_lasso.scores_[1], axis=0)*1.96

        fig, ax = plt.subplots()

        if errorbars:
            ax.errorbar(Cs, mean, yerr=std, ecolor="lightgray", elinewidth=2, capsize=4, capthick=2)
        else:
            ax.plot(Cs, mean)
        ax.set_xscale('log')
        ax.set_xlabel("C")
        ax.set_ylabel("ROC AUC")

        return ax
    else:
        print("Does not match linear_model.LassoCV or linear_model.LogisticRegressionCV")
        return False
```