ML for SS: Workflow and ML regression

Dr. Richard M. Crowley

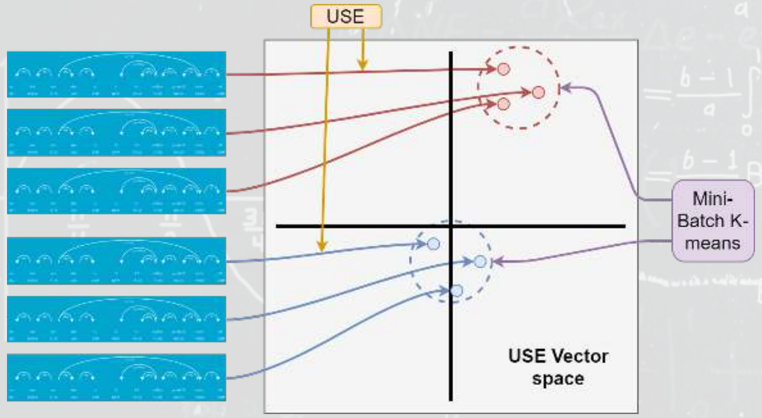
rcrowley@smu.edu.sg

https://rmc.link/

About me

Research

- Accounting disclosure: What companies say, and why it matters
  - Focus on social media and regulatory filings
- Approach this using AI/ML techniques



Research highlights

1. Detecting financial misreporting using the topic modeling of annual report text
2. Multiple projects on Twitter showcasing:
    1. How companies strategically disseminate financial information on Twitter
    2. That CSR disclosure on Twitter is not credible
    3. That executives' disclosures are as important on Twitter as their firms' disclosures
3. Newer work on
    - COVID-19 reactions worldwide
    - Sentiment and understandability in accounting text
    - Misinformation laws (e.g., POFMA)

What is the common thread?

All of the above use text-based data paired with AI/ML algorithms. A secondary thread is the importance of *content*, while some papers also push for better *causality* in research.

A quick overview of the course

A typical class session

2-3 papers to discuss

- Each paper will usually use a different method
- Almost all papers are applied
- Student led
  - 2 students per paper

Method overview

- Walk through methods' technical aspects
- Discuss how and where the method is useful
- Showcase a coded up example
  - When feasible, I will show this for both R and python
- Professor led

Expectations

Already

1. Have a working knowledge of python or R
    - If you don't, I can provide access to training materials
2. Have some understanding of statistics (e.g., regression)
3. Prior understanding of ML is *not required*
    - You will learn a lot in the next 12 weeks!

In class

1. Read *all* the required papers
2. Be ready to discuss them!
    - Ask questions
    - Answer questions
3. Paper presentations should balance presentation and discussion (critical thinking)
    - Have a really good point to discuss? Ask it to the class
        - A great presentation will make us all think more

What we will cover

> Regression and analysis with ML

1. Working with data and ML regression
2. Tree-based ML algorithms
3. Clustering algorithms

> Not far removed from traditional econometrics, but more flexible

- Drop-in replacements for regression
- Non-linear and non-parametric methods
- Dimensionality reduction

What we will cover

> Working with textual data

4. Text processing (NLP)
5. Linguistics
6. Embedding and topic models
7. Inferring traits from text

> These methods are often useful in measuring phenomenon

- What is being discussed (content)
- People’s sentiment or emotion toward something

What we will cover

> Economics approaches to ML

8. Causal machine learning
9. Policy prediction
10. Bias

> Useful for measuring impact or effects

- Understanding policy impacts
- Understanding processes

What we will cover

> Neural networks

11. Text processing
12. Image processing

> These methods can offer powerful methods for measuring phenomenon

- Better understanding message content
- Picking apart images
- Building better classifiers

Overview

Papers

> Paper 1: Mullainathan and Spiess 2017 JEP

- A fairly approachable overview of ML methods in economics
- The points the paper makes are applicable broadly in any archival/empirical discipline

> Paper 2: Chahuneau et al 2012

- An application of LASSO to a context most should be familiar with: restaurant menus
- Easy to motivate LASSO in this paper – more variables than observations!

Technical discussion: Implementing LASSO

1. Sample splitting
2. Cross validation
3. What are LASSO and Elastic Net (i.e., regularized regression)
4. Implementing them

Python

- Using `sklearn`
- Can be done using built-in CV methods

R

- Using glmnet
- Fast and easy to use
- Nice CV methods built-in

> Both Python and R are good for this. Stata is also OK with `lassopack`.

> A worked out solution for each language on my website, data is on eLearn.

Main application: A 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)

Secondary application: A 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)

Paper 1: An overview of applied ML

Paper 2: ML for panel data

Problems of the usual approach

- For both linear and logistic regression:
  - Easy to have too many covariates
    - Which can lead to high VIFs and multicollinearity
- 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., $|\varepsilon|$

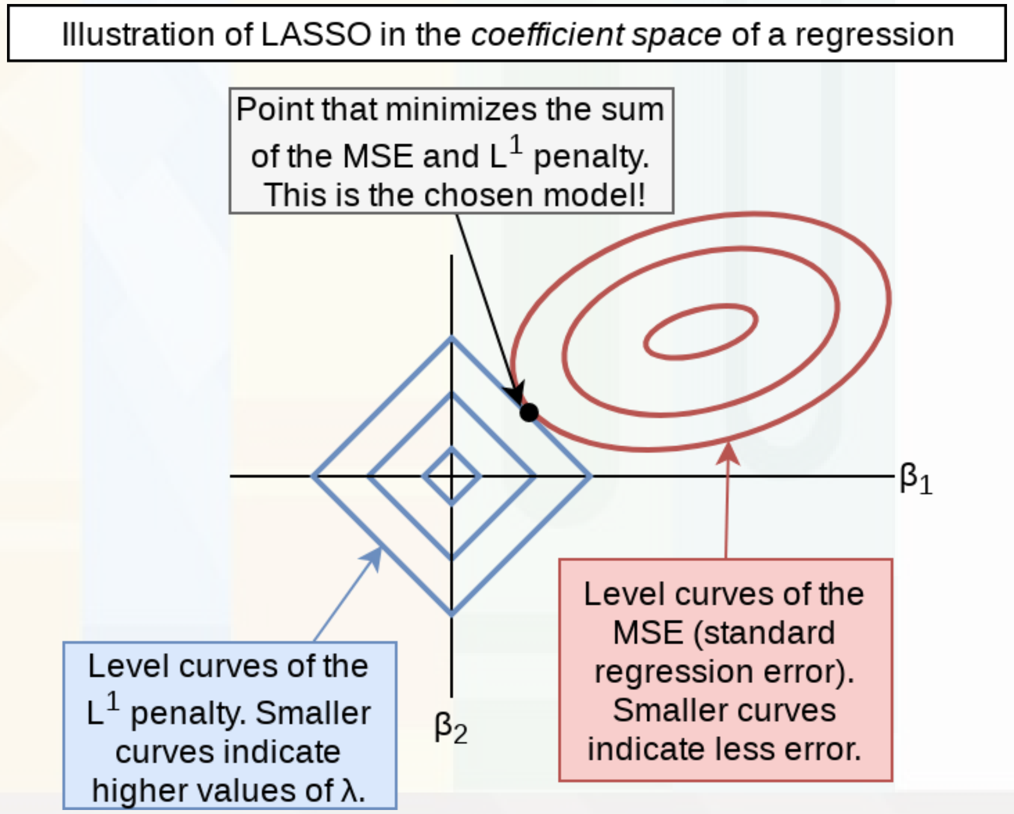
> Great if you have way too many inputs in your model or high multicollinearity

- Note that L_1 regularization is a standard approach to dealing with inflated VIFs as well!

How does it work?

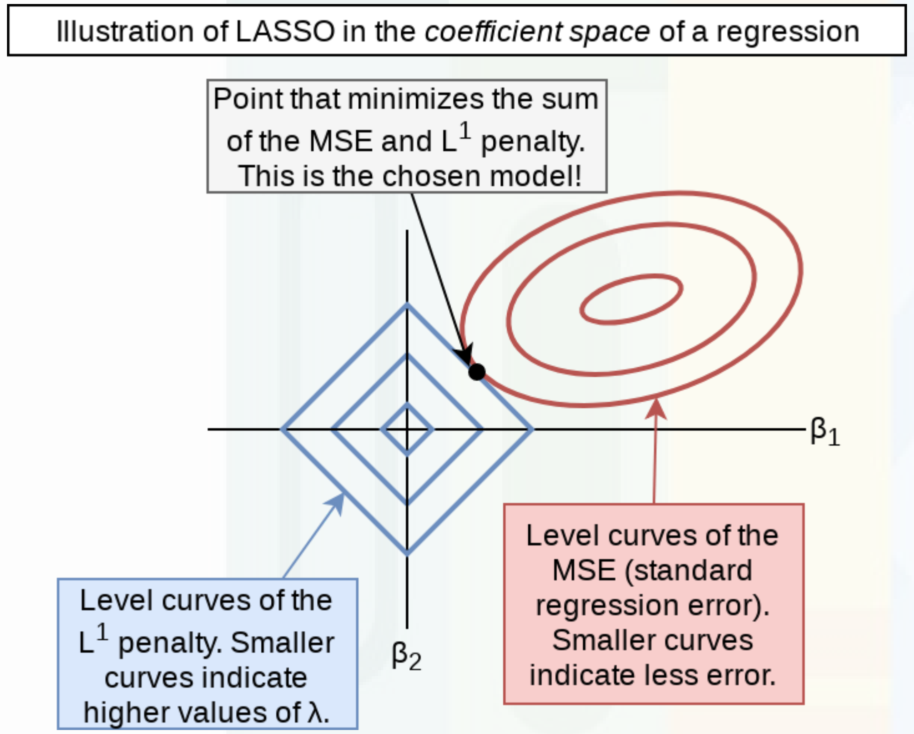
$$\min_{\beta \in \mathbb{R}} \left\{ \frac{1}{N} |\varepsilon|_2^2 + \lambda |\beta|_1 \right\}$$

- 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



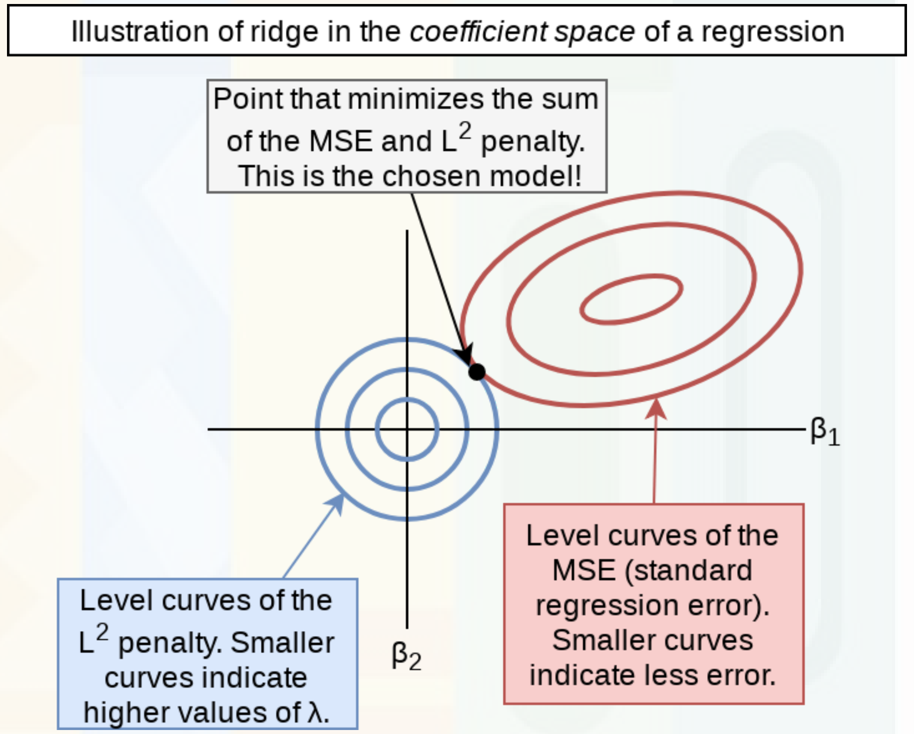
What about other penalty types?

LASSO (L_1)



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

Ridge (L_2)

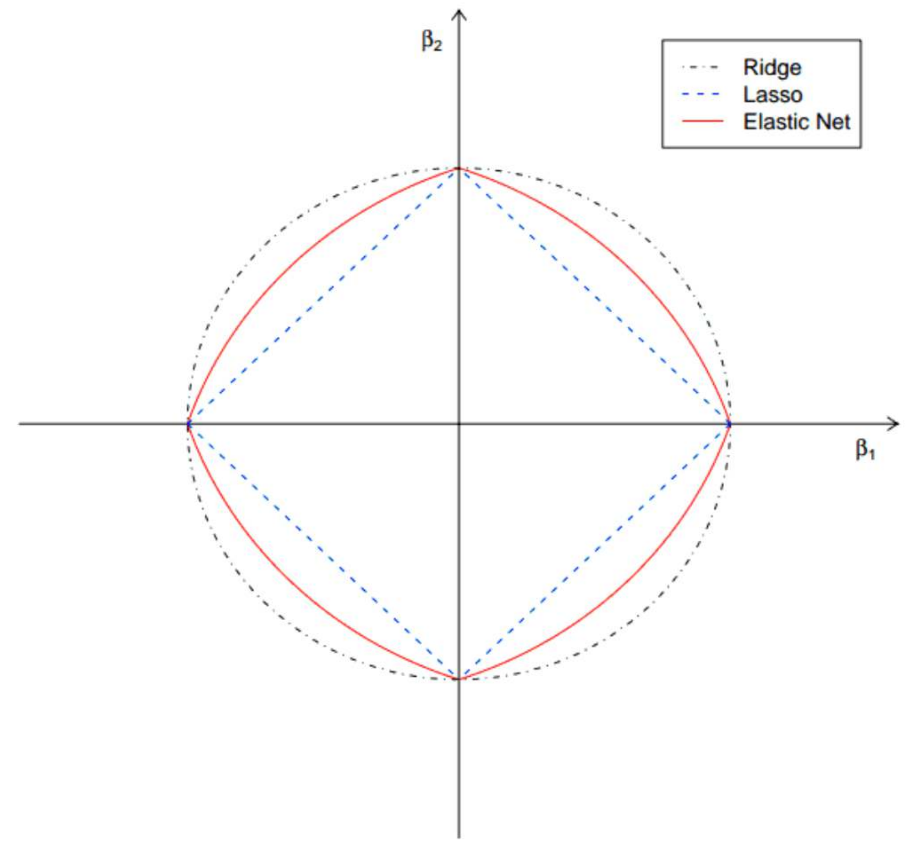


- Decreases coefficient values
  - Increases prediction stability more
  - Less sensitive to outliers

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

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



Technical: Preparation

Importing data

- Python: We can use pandas to import the data set
- R: We can use tidyverse to import the data set
- Compressing a csv file can save 50-90% of the storage space of the file

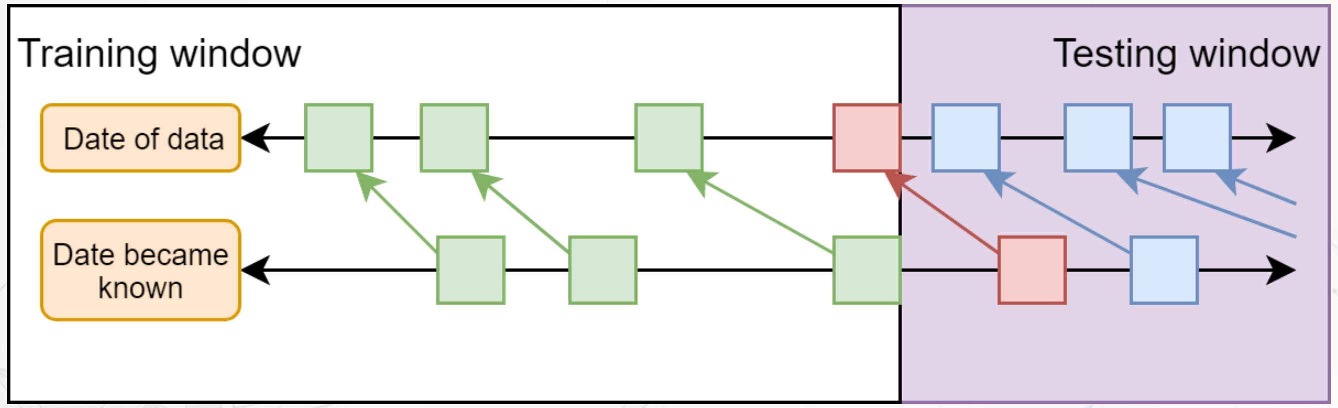
```python
df = pd.read_csv('../../Data/S1_data.csv.gz')
```

```r
df = 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

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)
```

```r
# Check the years in the data
unique(df$year)
```

```
[1] 2002 2003 2004 1999 2000 2001
```

- 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`

Splitting the sample

```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)
```

```r
# Subset the final year to be the testing year
train <- df %>% filter(year < 2004)
test  <- df %>% filter(year == 2004)
print(c(nrow(df), nrow(train), nrow(test)))
```

```
[1] 14301 11478  2823
```

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

Aside: Random testing sample

- In python, 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()`
  - Optionally you can stratify across classes in your data using the `stratify=` parameter
- In R, caret can handle this well using the `createDataPartition()` function

Technical: Running simple regressions

Using Statsmodels in Python

- 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

```python
formula = 'sdvol1 ~ ' + ' + '.join(vars_topic[0:-1])
model = smf.ols(formula=formula, data=train)
```

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

> Step 2: Run the regression

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

Linear regression (OLS)

> Step 3: Output the results (optional)

```python
fit1.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:	Sun, 20 Aug 2023	Prob (F-statistic):	0.00
Time:	17:23:20	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

Base R

- Fitting regressions is straightforward in R

```r
BD_eq <- as.formula(paste("sdvol1 ~ ", paste(paste0("Topic_",1:30,"_n_oI"), collapse=" + "), collapse=""))
model <- lm(BD_eq, train)
summary(model)
```

```

Call:
lm(formula = BD_eq, data = train)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.18799 -0.01707 -0.00646  0.00904  0.49410 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)    
(Intercept)    0.0457521  0.0002674 171.114  < 2e-16 ***
Topic_1_n_oI   1.1709484  0.3404372   3.440 0.000585 ***
Topic_2_n_oI   0.5367261  0.2615383   2.052 0.040174 *  
Topic_3_n_oI   0.4004462  0.4160324   0.963 0.335801    
Topic_4_n_oI   0.6475066  0.2386256   2.713 0.006668 ** 
```

Logistic regression in Python

```python
formula = 'Restate_Int ~ ' + \
          ' + '.join(vars_financial) + ' + ' +\
          ' + '.join(vars_style) + ' + ' +\
          ' + '.join(vars_topic[0:-1])  # Drop the final value to avoid multicollinearity
model = smf.logit(formula=formula, data=train)
fit_logit = model.fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.054196
         Iterations: 35
```

```python
fit_logit.summary()
```

Logit Regression Results

Dep. Variable:	Restate_Int	No. Observations:	11478
Model:	Logit	Df Residuals:	11410
Method:	MLE	Df Model:	67
Date:	Sun, 20 Aug 2023	Pseudo R-squ.:	0.1205
Time:	17:23:21	Log-Likelihood:	-622.06
converged:	False	LL-Null:	-707.27
Covariance Type:	nonrobust	LLR p-value:	5.753e-11

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-6.6337	5.591	-1.187	0.235	-17.592	4.324

Logistic regression in R

```r
BCE_eq <- as.formula(paste("Restate_Int ~ 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 + 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 + ",
  paste(paste0("Topic_",1:30,"_n_oI"), collapse=" + "), collapse=""))

model_logit <- glm(BCE_eq, train, family="binomial")

summary(model_logit)
```

```
Call:
glm(formula = BCE_eq, family = "binomial", data = train)

Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)       -6.634e+00  5.591e+00  -1.187  0.23541
logtotasset        9.363e-02  6.442e-02   1.454  0.14607
rsst_acc           3.269e-01  3.226e-01   1.013  0.31095
chg_recv           6.838e-01  1.307e+00   0.523  0.60085
chg_inv           -1.428e+00  1.509e+00  -0.947  0.34378
soft_assets        1.451e+00  4.698e-01   3.088  0.00201 **
pct_chg_cashsales -1.230e-03  8.480e-03  -0.145  0.88472
chg_roa           -2.584e-01  2.635e-01  -0.981  0.32666
issuance           2.336e-01  4.218e-01   0.554  0.57971
oplease_dum        1.529e-01  3.136e-01   0.488  0.62572
book_mkt           7.977e-03  4.436e-02   0.180  0.85731
lag_sdvol         -4.005e-02  1.003e-01  -0.399  0.68984
merger            -2.662e-01  2.563e-01  -1.039  0.29903
bigNaudit         -1.544e-01  4.452e-01  -0.347  0.72877
midNaudit          3.926e-01  5.218e-01   0.752  0.45180
```

Technical: Measuring predictive performance

Linear predictive power

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

RMSE

```python
sklearn.metrics.mean_squared_error()
```

```r
apply_rmse <- function(v1, v2) {
  sqrt(mean((v1 - v2)^2, na.rm=T))
}
```

MAE

```python
sklearn.metrics.mean_absolute_error()
```

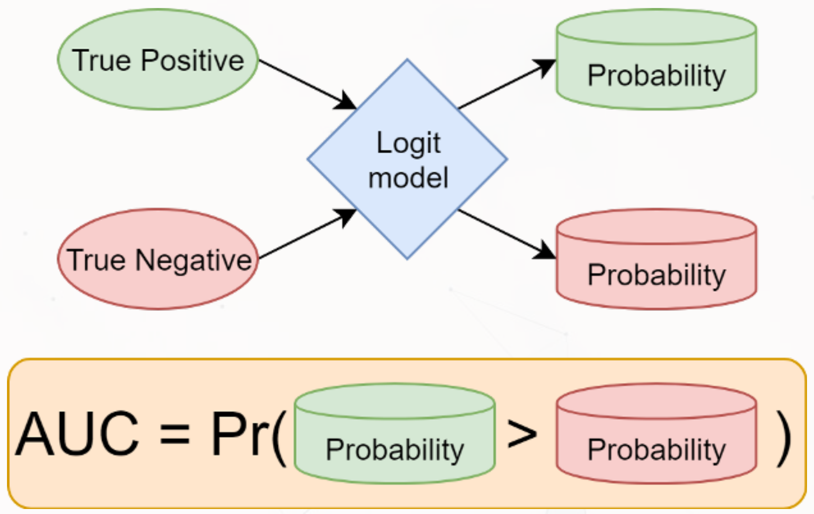
```r
apply_mae <- function(v1, v2) {
  mean(abs(v1-v2), na.rm=T)
}
```

Logistic predictive power

- For logistic regression, ROC AUC is a good measure
- Use `sklearn` in python or yardstick in R

```python
Y_hat_test = fit_logit.predict(test)
auc = metrics.roc_auc_score(test.Restate_In
                            Y_hat_test)
```

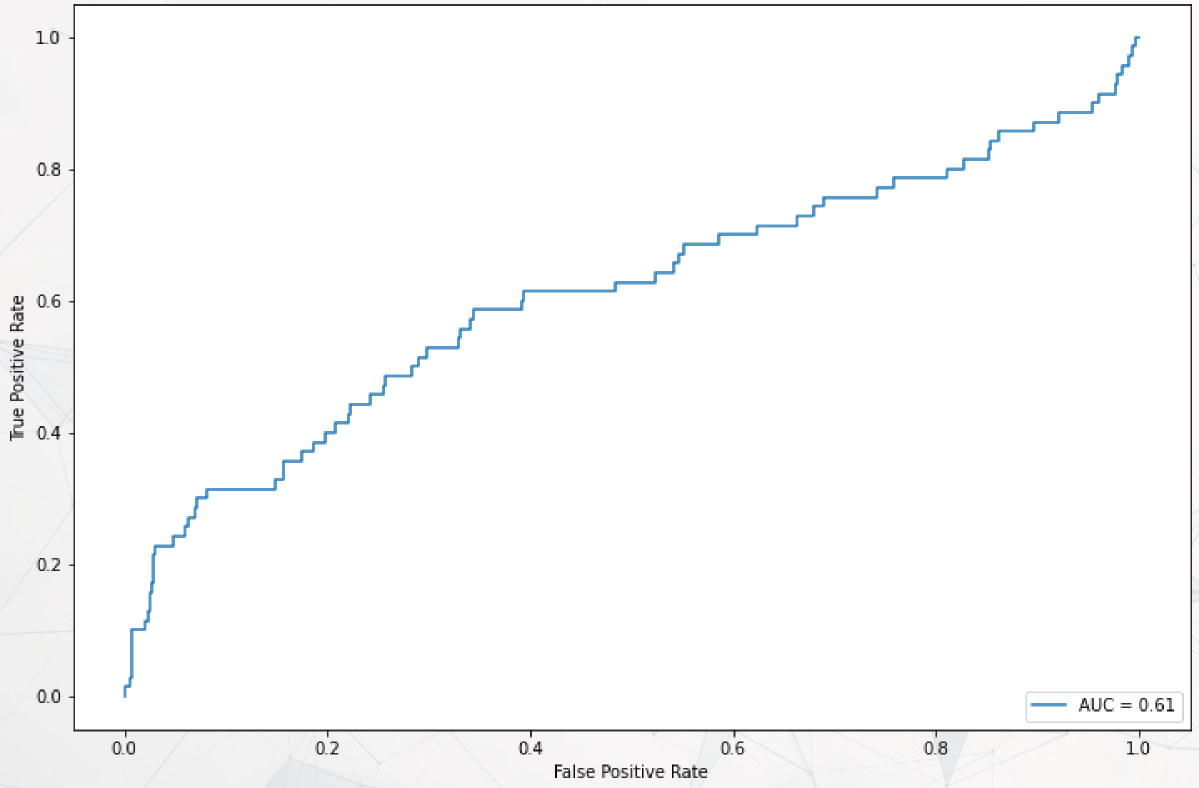
```r
test$Y_hat_test <- predict(model_logit,test
                           type="response")
auc_out <- test %>%
  roc_auc(Restate_Int_f,  # must be a facto
          Y_hat_test,
          event_level='second')
```



Visualizing AUC with the ROC curve

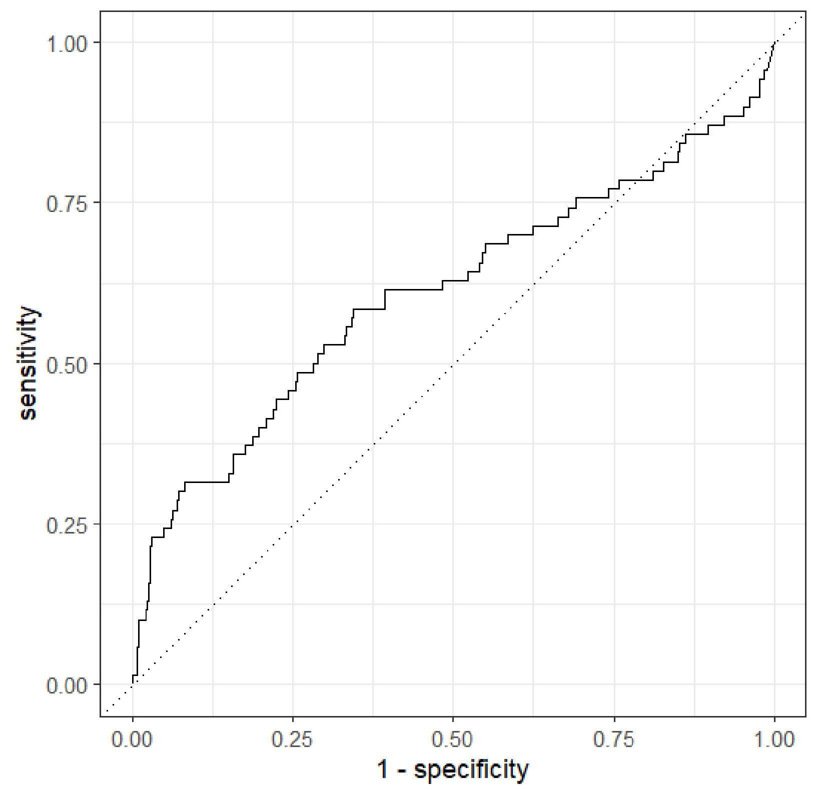
- `sklearn` makes it easy to output a ROC curve as well

```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()
```



Using yardstick in R

```r
test$Y_hat_test <- predict(model_logit, test, type="response")
test %>%
  roc_curve(Restate_Int_f, Y_hat_test, event_level='second') %>%
  autoplot()
```



Technical: Implementing LASSO (linear)

Using python: Setting up to use Scikit-Learn

- Scikit-learn, like many machine learning packages, expects separate data sets or matrices for DVs and IVs
- LASSO, Ridge, and Elastic net are also particular about data format:

> Every input should be normalized to a Z-score! (python-specific requirement)

- 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])
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 SD 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!

Using Python: Setting up to use Scikit-Learn

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

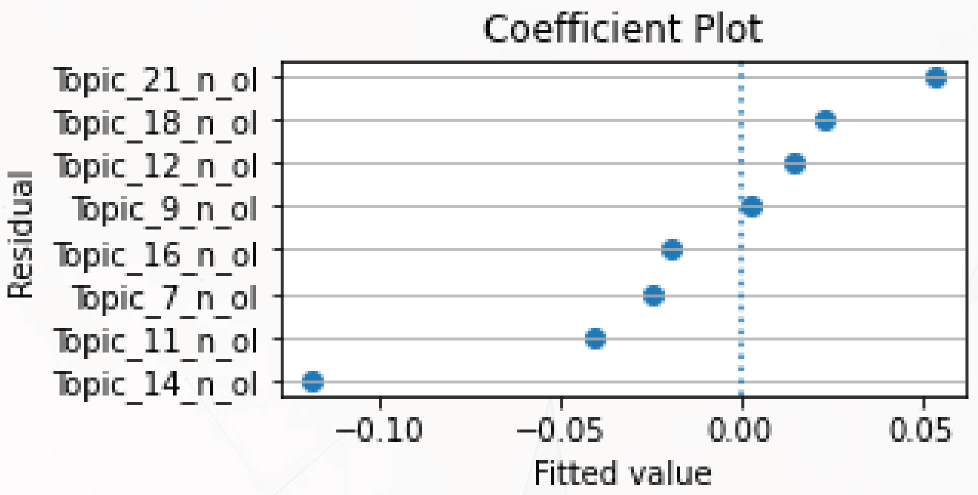
Using Python: Simple LASSO, linear

Fitting a LASSO with a pre-specified penalty

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

Custom coefficient plot function

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



> Not too difficult, but the coefplot function is custom (see Jupyter notebook for it)

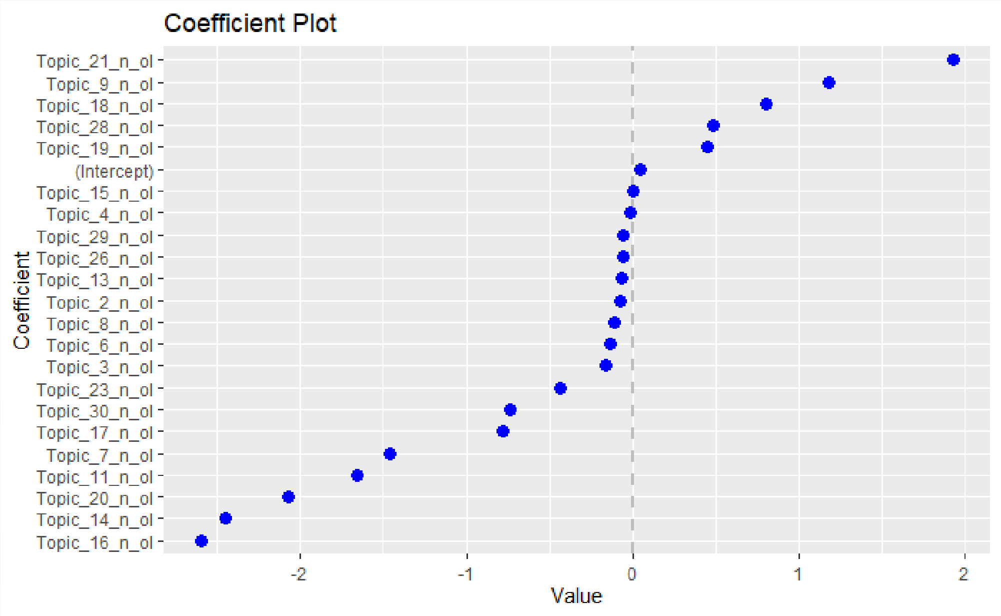
Using R: Setting up to use glmnet

- The glmnet package expects data as separate matrices for X and Y measures
- It does not require data to be Z-scores – it is invariant to this
  - This is a `{glmnet}'-specific nicety, other R packages may require scaling
- The `model.matrix()` and `model.frame()` commands from Base R make this easy

```r
x_lm <- model.matrix(BD_eq, data=train)[,-1]  # [,-1] to remove intercept
y_lm <- model.frame(BD_eq, data=train)[,"sdvol1"]
```

Using R: Running glmnet

```r
fit_LASSO_lm <- glmnet(x=x_lm, y=y_lm,
                       family = "gaussian",
                       alpha = 1  # Specifies LASSO.  alpha = 0 is ridge
                      )

coefplot(fit_LASSO_lm, sort='magnitude')
```



> In this case, `coefplot` is available from CRAN

Cross validation (linear)

What is cross validation?

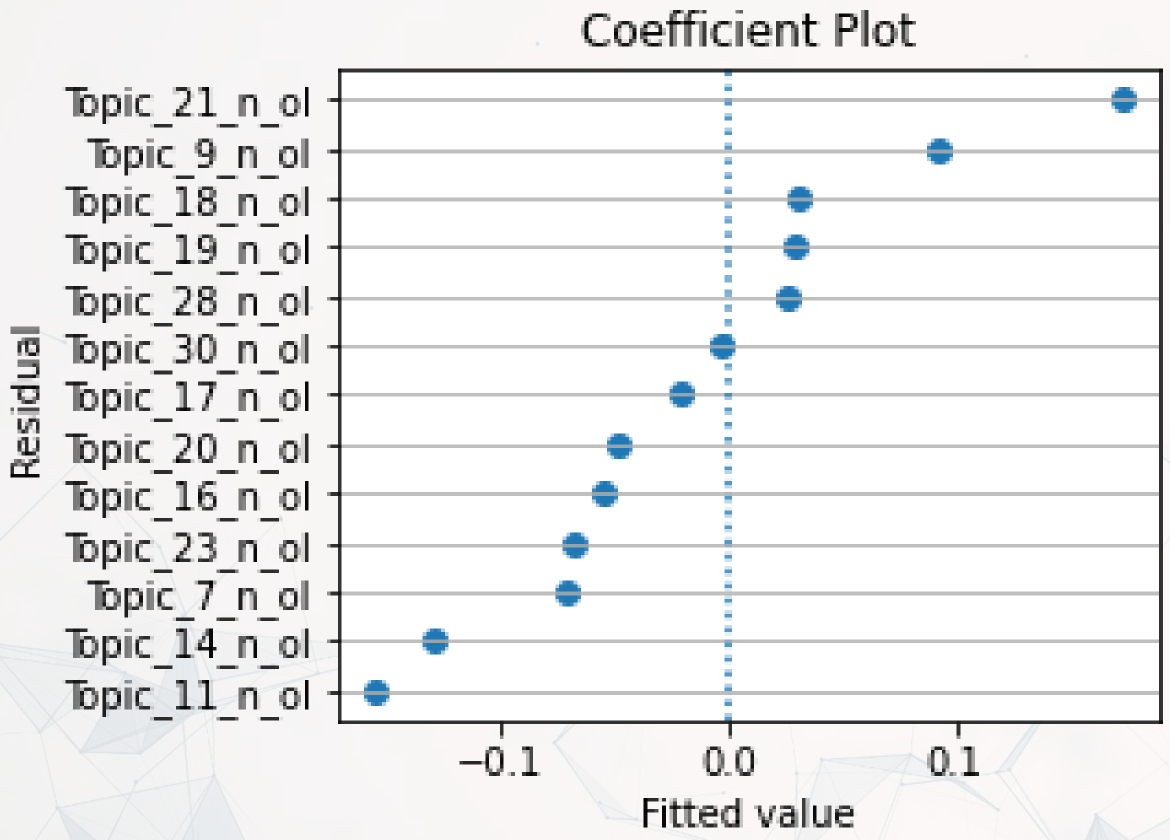
- Validation is where you keep part of the training sample as a hold out sample to evaluate and improve your algorithm against
  - This prevents biasing towards the real hold out sample (the testing sample)
- Cross validation takes this further by making a bunch of validation samples,
- An example of 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! So does glmnet!

10-fold CV LASSO, linear, Python

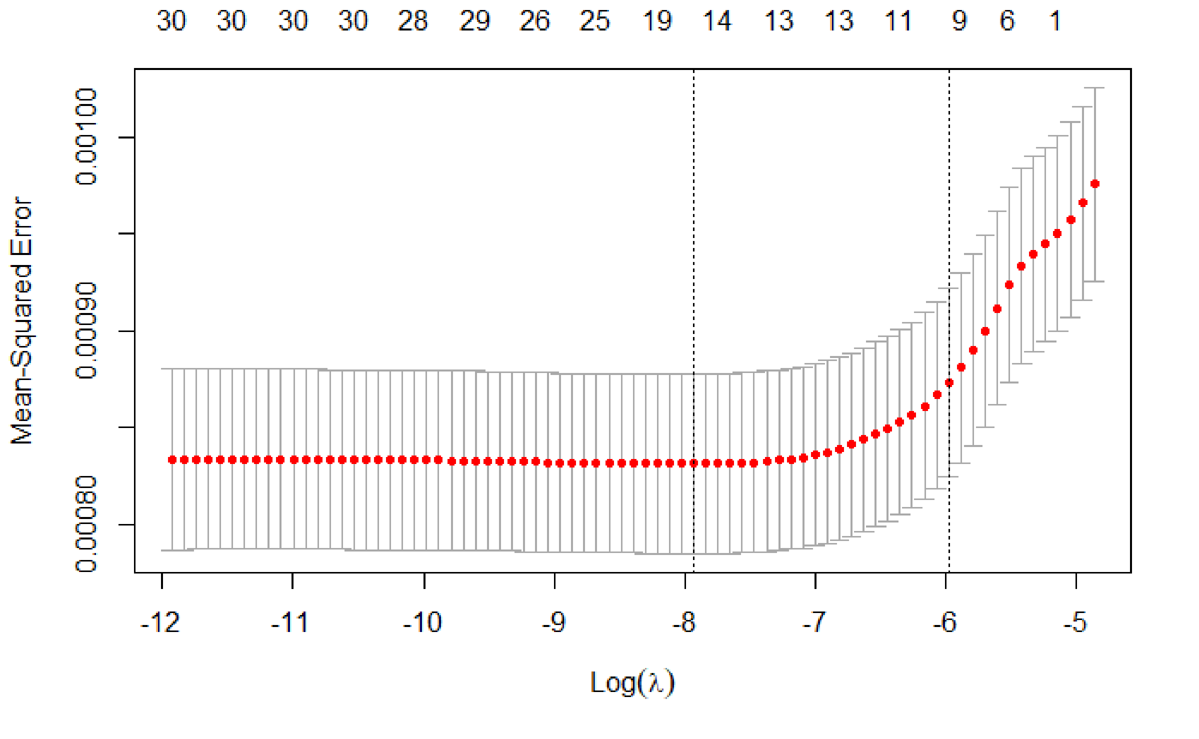
```python
reg_lasso = linear_model.LassoCV(cv=10)
reg_lasso.fit(train_X_linear, np.ravel(train_Y_linear))

coefplot(vars, reg_lasso.coef_)
```



10-fold CV LASSO, linear, R

- To replicate our linear LASSO:

```r
cvfit_lm = cv.glmnet(x=x_lm, y=y_lm, family = "gaussian", alpha = 1, type.measure="mse")
plot(cvfit_lm)
```



> Note: This is optimizing MSE instead of R^2 – glmnet doesn't support R^2!

10-fold CV elastic net, linear, Python

- 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

```python
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))
```

> Note: This does CV over both parameters!

10-fold CV elastic net, linear, R

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

```r
cvfit_en = cv.glmnet(x=x, y=y, family = "binomial", alpha = 0.5, type.measure="auc")
```

> Note: This does CV only over the penalty parameter. You need to build your own grid over the alpha parameter

LASSO for logistic regression

Using python: Setting up to use Scikit-Learn

- Scikit-learn, like many machine learning packages, expects separate data sets or matrices for DVs and IVs
- LASSO, Ridge, and Elastic net are also particular about data format:

> Every input should be normalized to a Z-score! (python-specific requirement)

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

```python
vars = vars_financial + vars_style + vars_topic
scaler_X = preprocessing.StandardScaler()
scaler_X.fit(train[vars])
train_X_logistic = scaler_X.transform(train[vars])
test_X_logistic = scaler_X.transform(test[vars])
```

- `sklearn.preprocessing.StandardScaler()` defaults to transforming to Z-scores
- Applying `.fit()` with data makes it calculate the mean and SD 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!

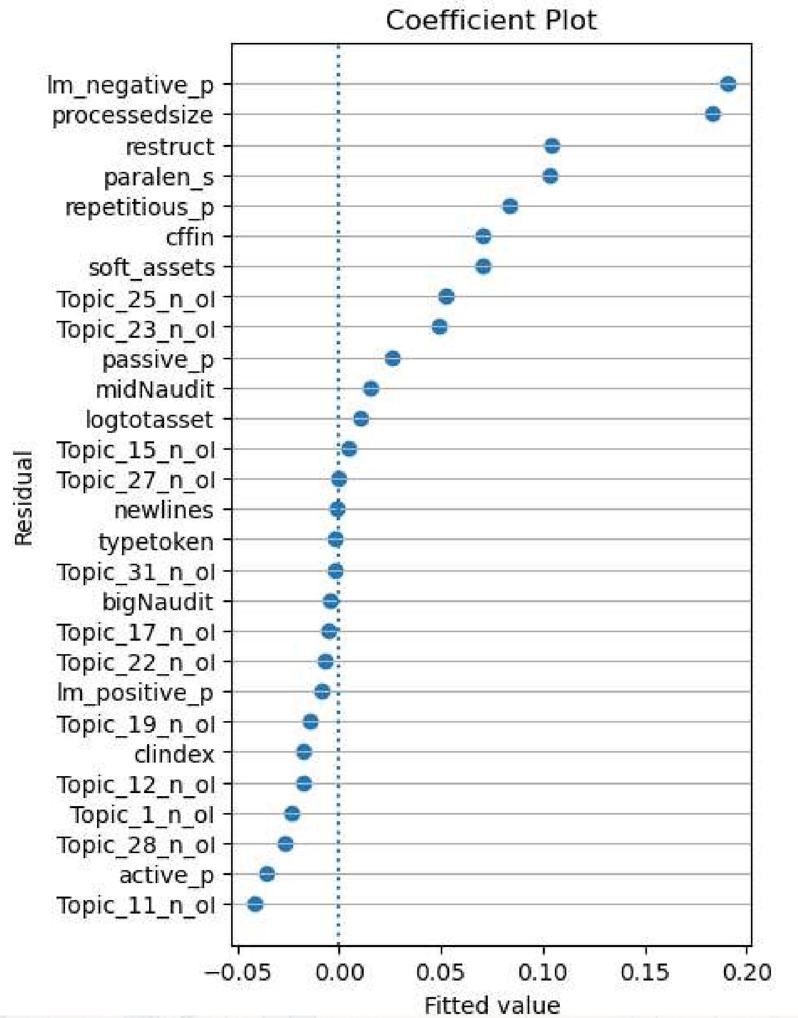
Using Python: Setting up to use Scikit-Learn

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

- Inputs are required to be 2D matrices by `sklearn`
- No scaling need for logistic LASSO, since it is binary

Using Python: Simple LASSO, linear

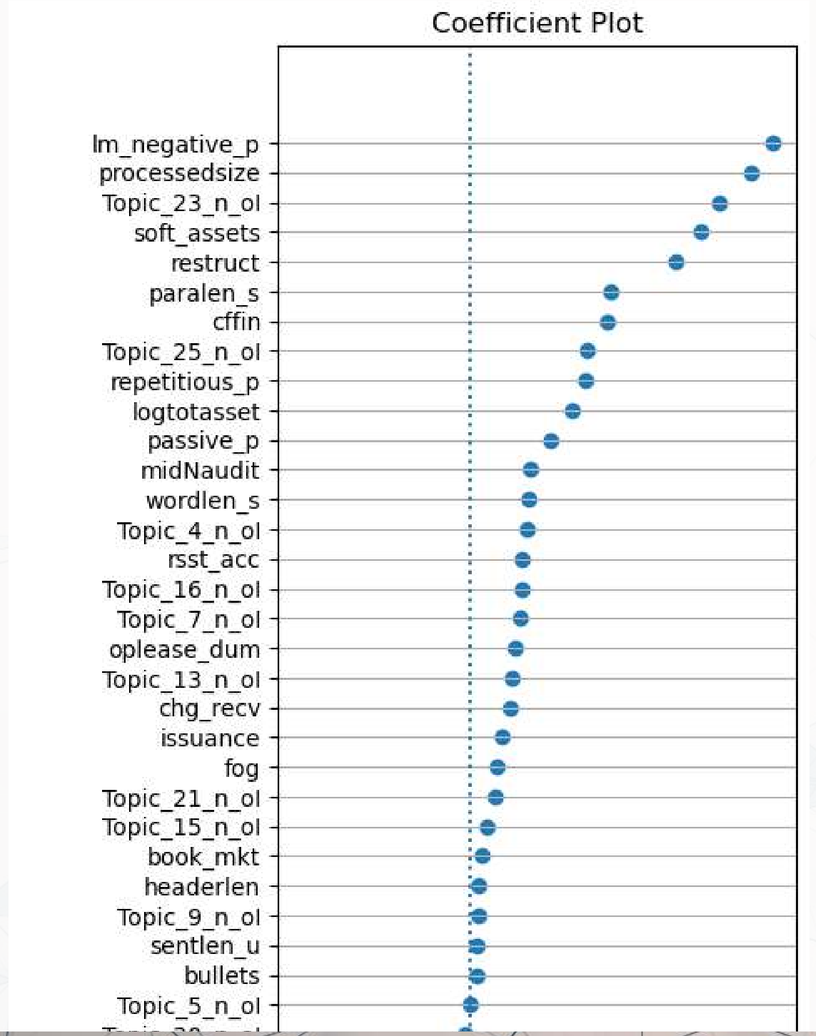
```python
reg_lasso = linear_model.LogisticRegression(penalty='l1', solver='saga', C=0.1)
reg_lasso.fit(train_X_logistic, train_Y_logistic)

coefplot(vars, reg_lasso.coef_)
```



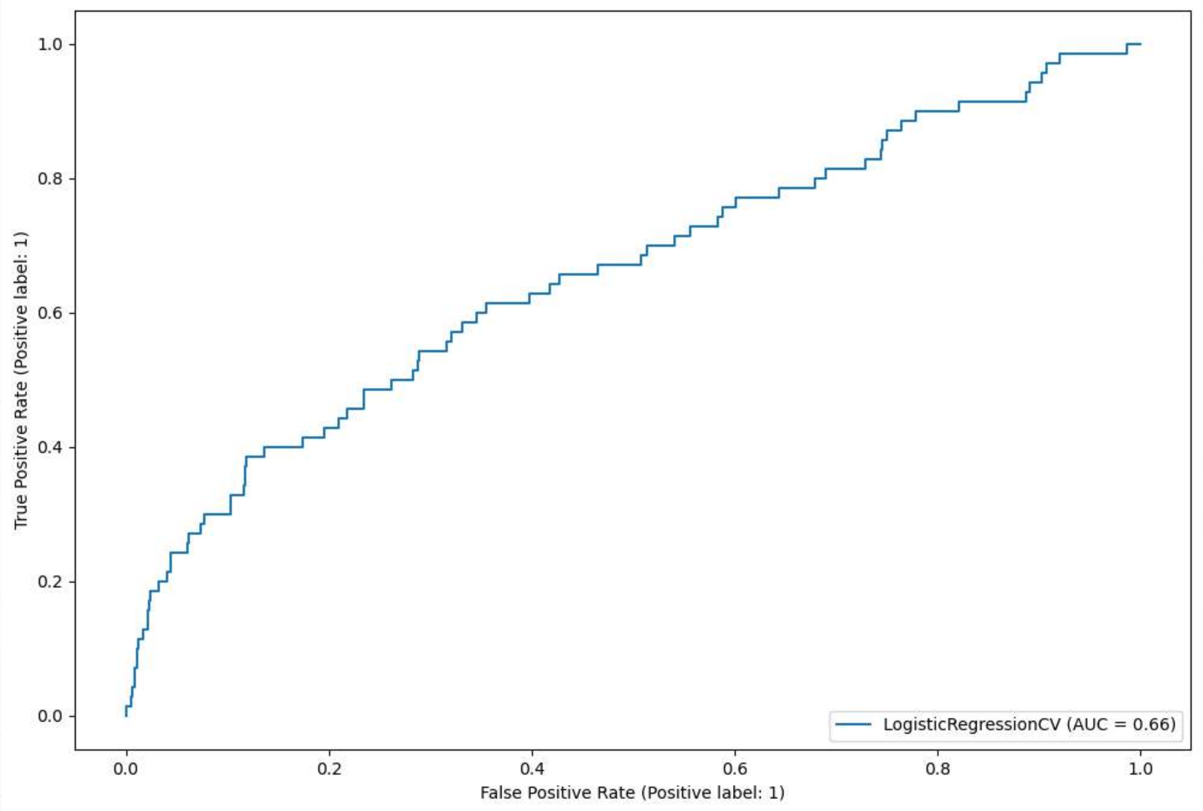
10-fold CV LASSO, linear, Python

```python
reg_lasso = linear_model.LogisticRegressionCV(penalty='l1', solver='saga', Cs=10, cv=5, scoring="roc_auc")
reg_lasso.fit(train_X_logistic, train_Y_logistic)
```

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




```python
display = \
  metrics.RocCurveDisplay.from_estimator(
  reg_lasso, test_X_logistic, test_Y_logist
display.plot()
```

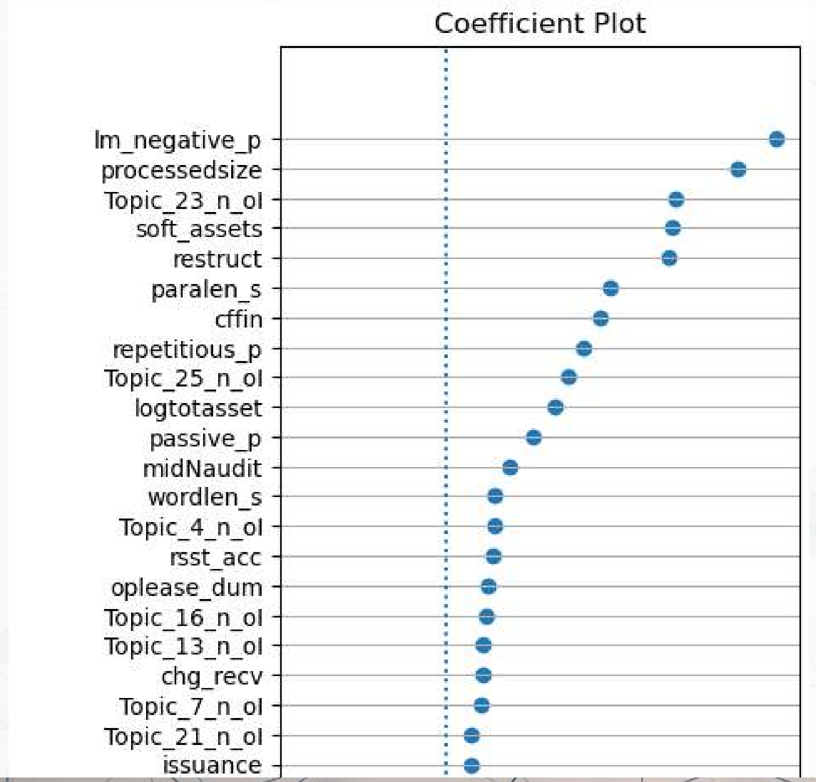


10-fold CV elastic net, linear, Python

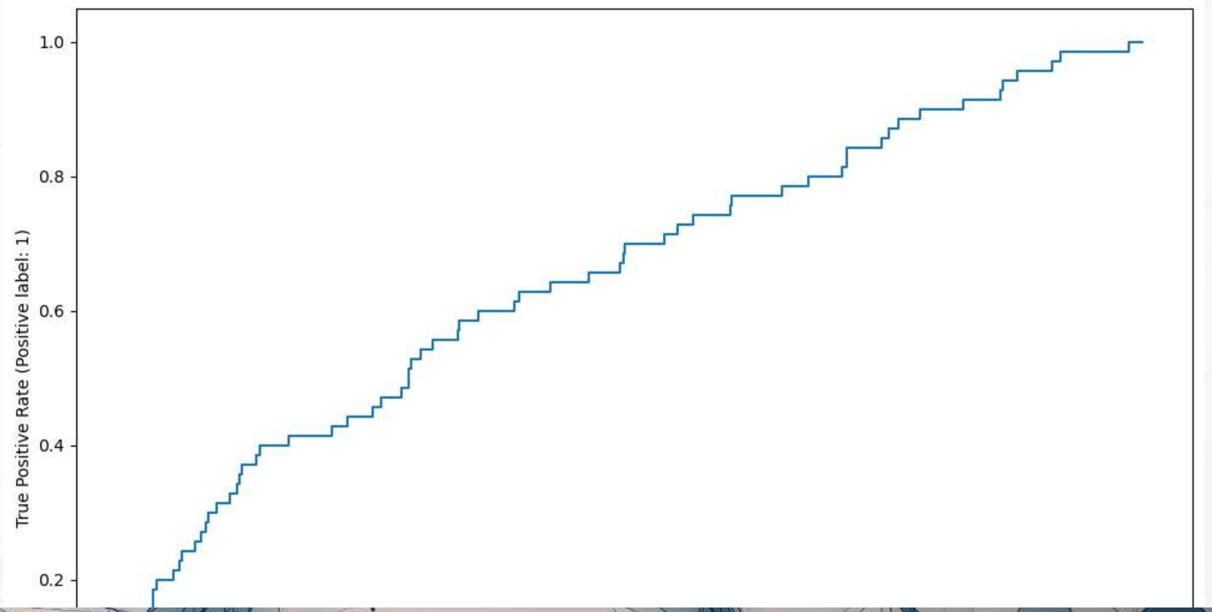
- To replicate our linear LASSO:

```python
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)
```

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

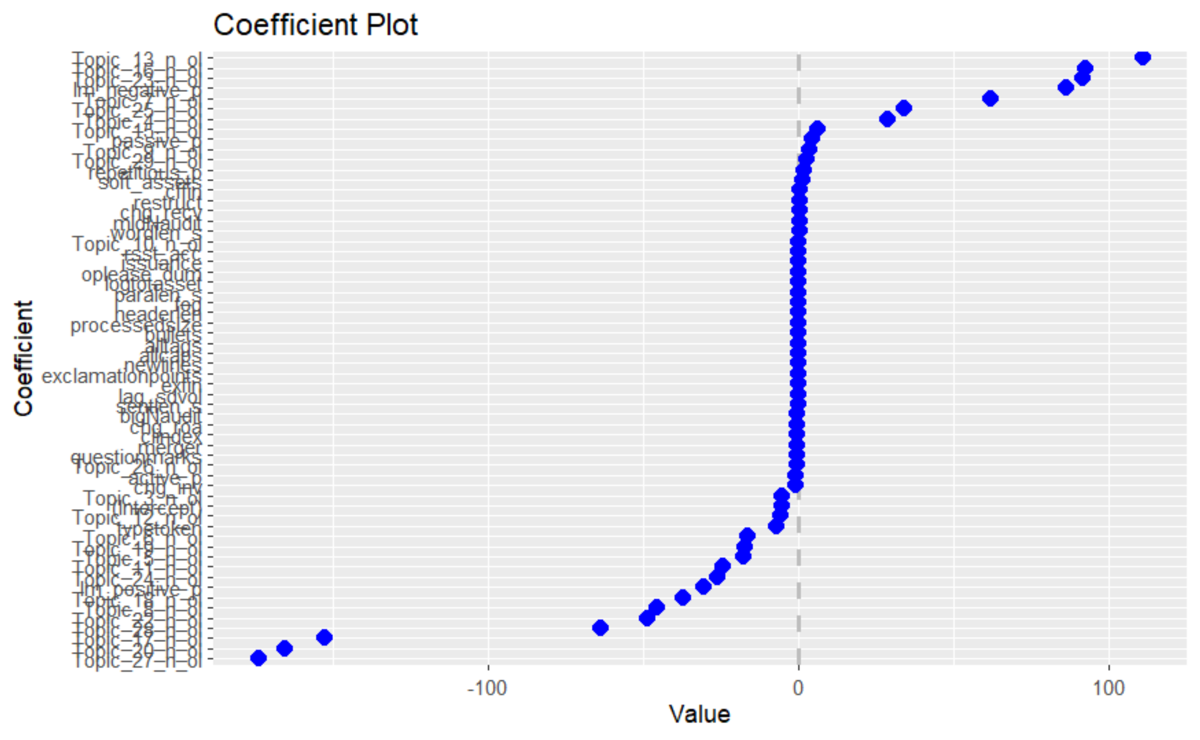



```python
display = \
  metrics.RocCurveDisplay.from_estimator(
  reg_EN, test_X_logistic, test_Y_logistic)
display.plot()
```



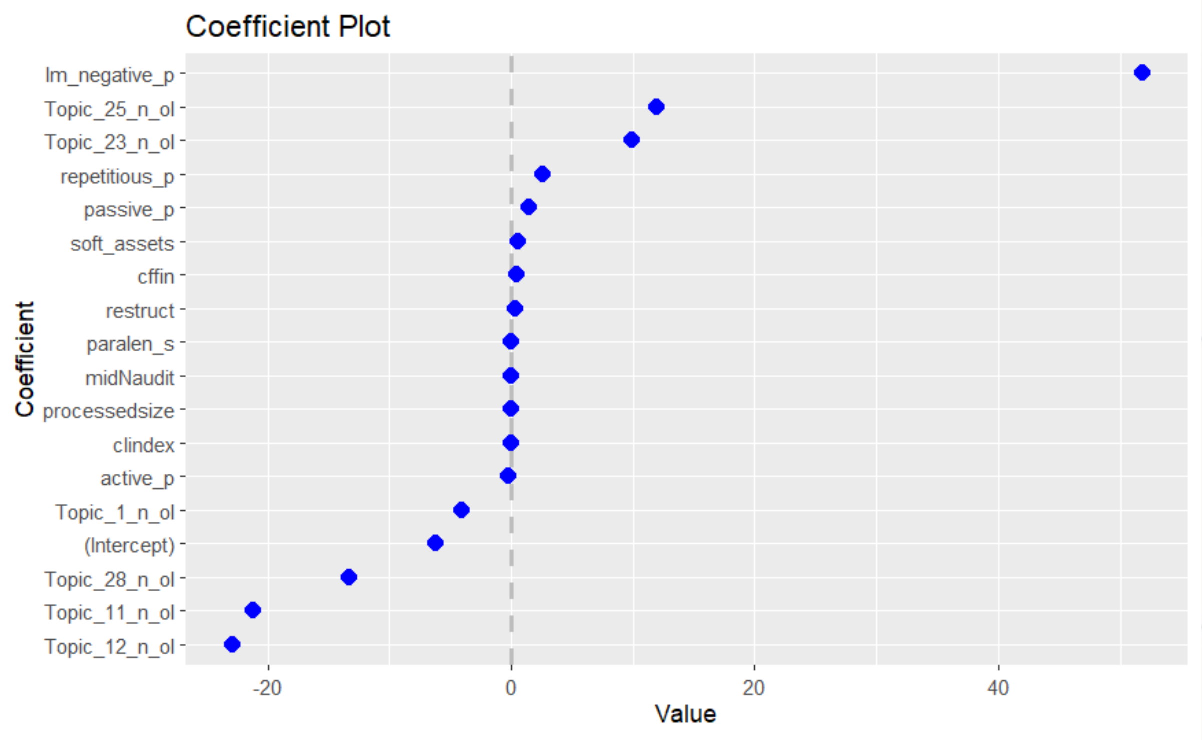
Simple logistic LASSO, R

```r
x <- model.matrix(BCE_eq, data=train)[,-1]  # [,-1] to remove intercept
y <- model.frame(BCE_eq, data=train)[,"Restate_Int"]
fit_LASSO_logit <- glmnet(x=x, y=y,
                          family = "binomial",
                          alpha = 1  # Specifies LASSO.  alpha = 0 is ridge
                         )
coefplot(fit_LASSO_logit, sort='magnitude')
```



10-fold CV LASSO, logistic, R

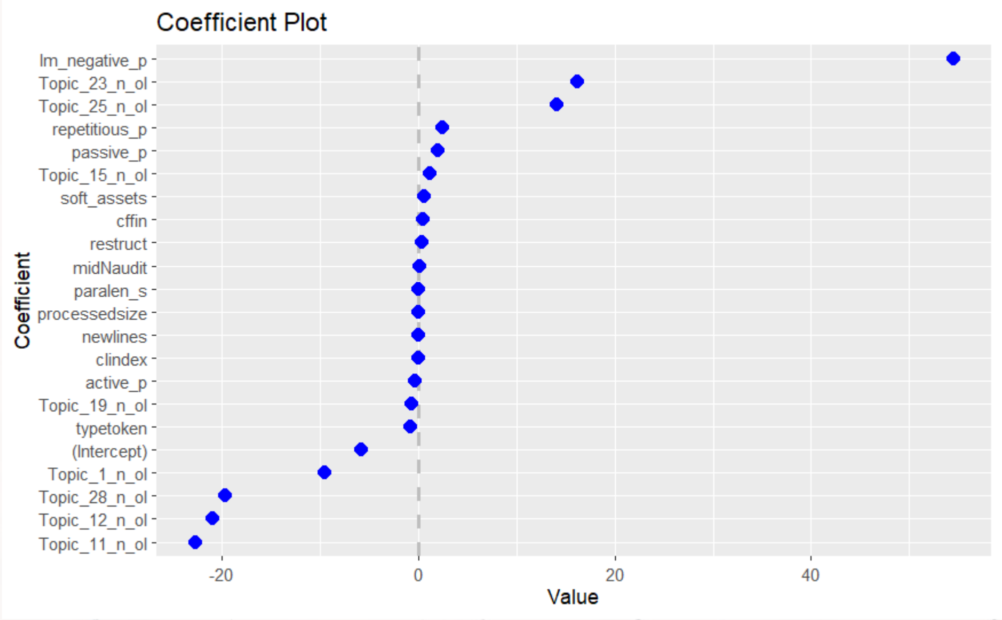
```r
cvfit_logit = cv.glmnet(x=x, y=y, family = "binomial", alpha = 1, type.measure="auc")

coefplot(cvfit_logit, lambda='lambda.min', sort='magnitude')
```



10-fold CV elastic net, logistic, R

- `alpha=1` is LASSO; `alpha=0` is Ridge; `0<alpha<1` is an elastic net

```r
cvfit_en = cv.glmnet(x=x, y=y, family = "binomial", alpha = 0.5, type.measure="auc")

coefplot(cvfit_en, lambda='lambda.min', sort='magnitude')
```



> Note: This does CV only over the penalty parameter. You need to build your own grid over the alpha parameter

Conclusion

Wrap-up

> Econometrics

- R and Stata are both better for this, python is capable but not as simple

> Machine learning regression

- Python is better at this than basic regression
- In some circumstances, these techniques are
  - More econometrically defensible, more robust, and more accurate
- R's glmnet package is more efficient and easier to use
  - But the elastic net implementation is more flexible for CV in Python
- Stata has an interesting implementation in `lassopack`
- For more interesting variants, check out R's `hdm`

Packages used for these slides

Python

- matplotlib
- numpy
- pandas
- scikit-learn
- statsmodels

R

- DT
- downlit
- glmnet
- kableExtra
- knitr
- plotly
- quarto
- reticulate
- revealjs
- tidyverse
- yardstick

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.
- Mullainathan, Sendhil, and Jann Spiess. "Machine learning: an applied econometric approach." Journal of Economic Perspectives 31, no. 2 (2017): 87-106.

Custom code

Replication of R’s coefplot function for use with sklearn’s linear and logistic LASSO

```python
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
```