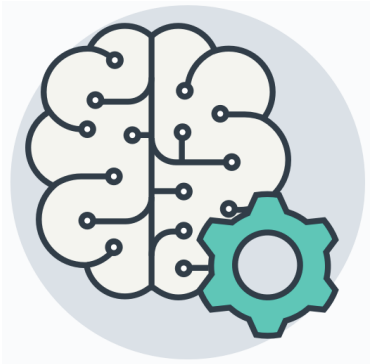
Aprendizado de Máquina

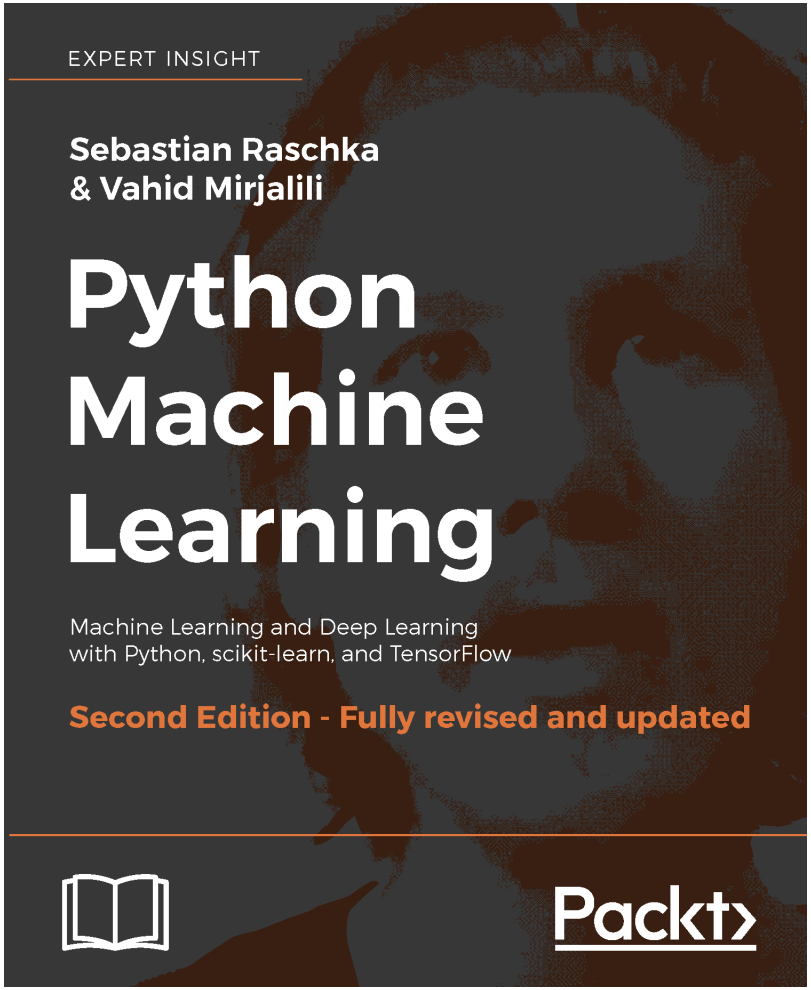
Pre-processamento de dados



Prof. Regis Pires Magalhães

regismagalhaes@ufc.br - http://bit.ly/ufcregis

Main Reference



Python Machine Learning

Chapter 4 - Building Good Training Sets – Data Preprocessing

Data Preprocessing

- Removing and imputing missing values from the dataset;
- Getting categorical data into shape for machine learning algorithms;
- Selecting relevant features for the model construction.

Missing Data

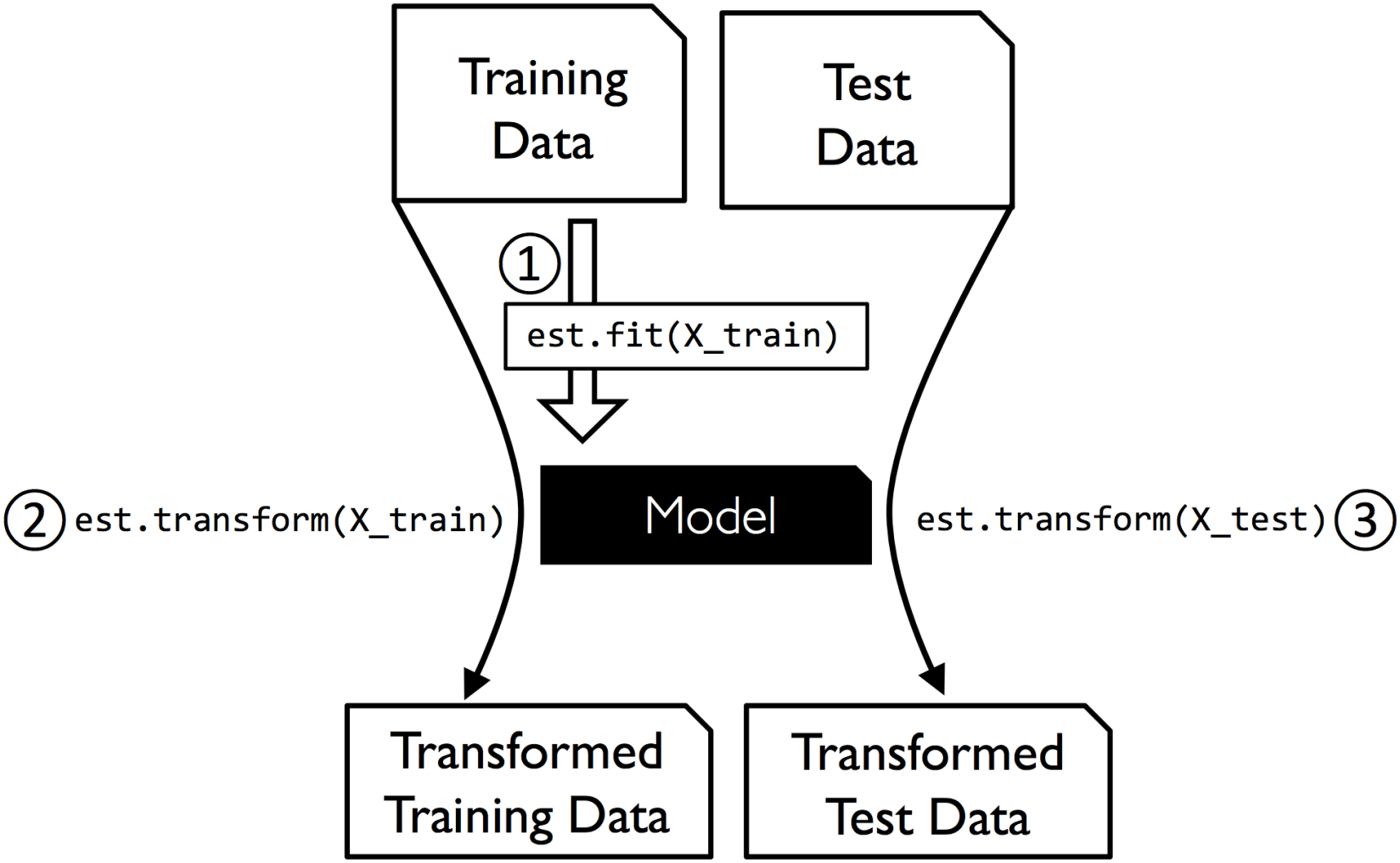
- Blank spaces in our data table
- Placeholder strings such as NaN, which stands for not a number
- NULL (a commonly used indicator of unknown values in relational databases).

Imputing missing values

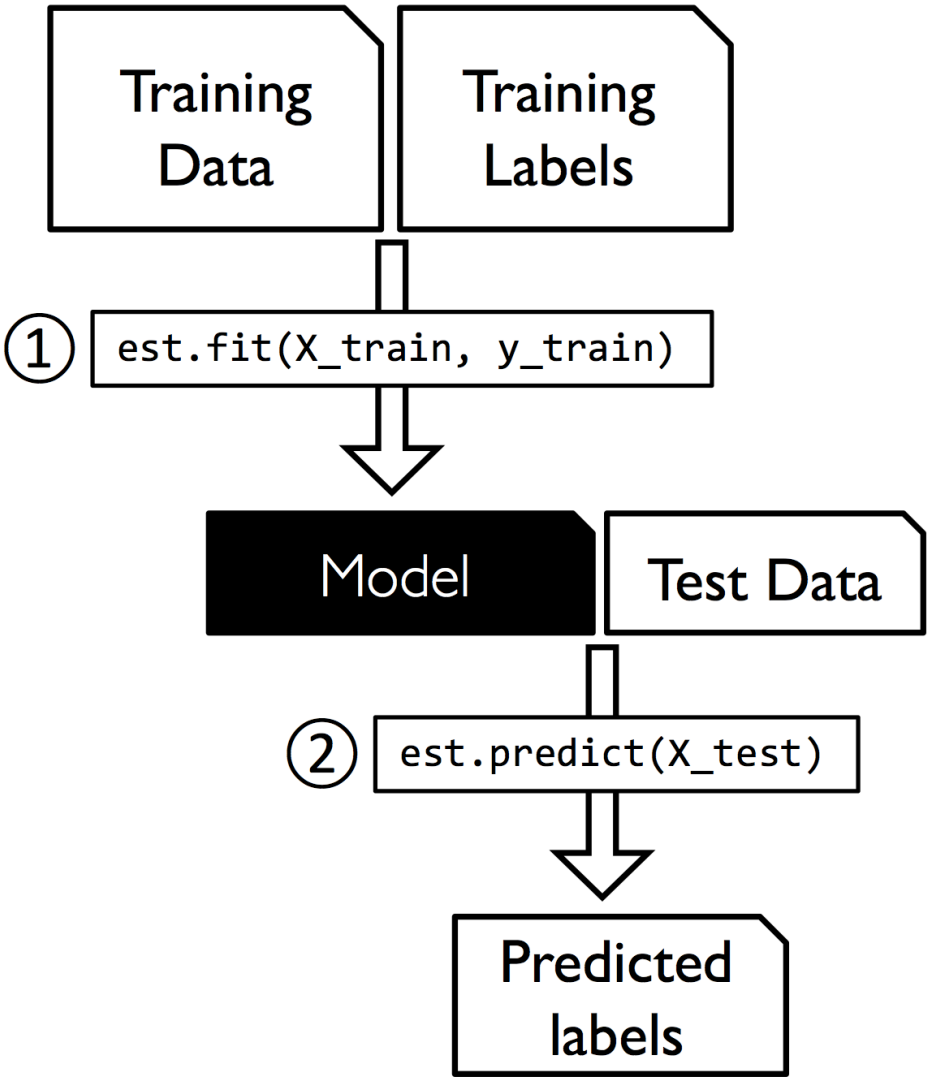
- One of the most common interpolation techniques is mean imputation.
  - We simply replace the missing value with the mean value of the entire feature column.

- Other options for the strategy parameter are **median** or **most_frequent**, where the latter replaces the missing values with the most frequent values.

Scikit-learn transform API



Scikit-learn estimator API



Selecting meaningful features

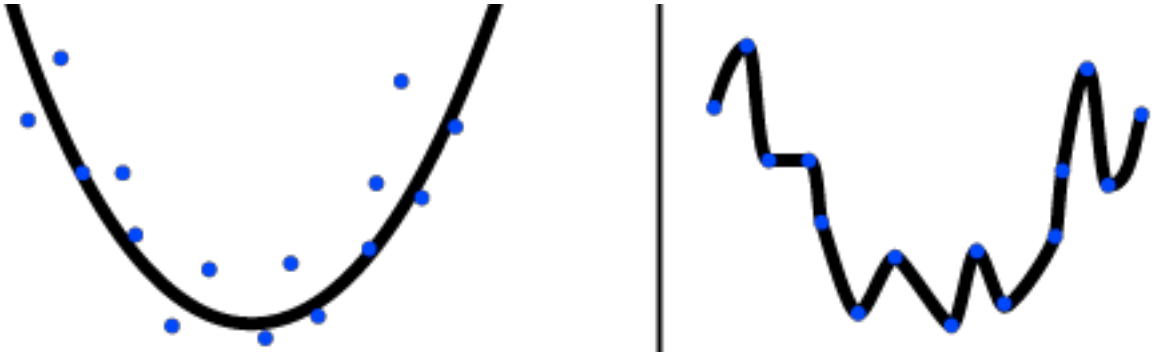
- If we notice that a model performs much better on a training dataset than on the test dataset, this observation is a strong indicator of **overfitting**.
  - The model fits the parameters too closely with regard to the particular observations in the training dataset, but does not generalize well to new data
  - We say the model has a high variance.

Solutions to reduce the generalization error

- Collect more training data
  - often not applicable
- Introduce a penalty for complexity
  - Via regularization
- Choose a simpler model with fewer parameters
- Reduce the dimensionality of the data
  - Via feature selection, which leads to simpler models by requiring fewer parameters to be fitted to the data.
  - Via feature extraction - we derive information from the feature set to construct a new feature subspace.
    - Dimensionality Reduction - techniques to compress a dataset onto a lower-dimensional feature subspace.

Regularization

- By adding a penalty to the Cost Function, the values of the parameters would decrease and thus the overfitted model gradually starts to smooth out depending on the magnitude of the penalty added.



After Regularization | Overfitted Model

L2 regularization

- Approach to reduce the complexity of a model by penalizing large individual weights, where we defined the L2 norm of our weight vector w as follows:

$$L2: \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$

L1 norm

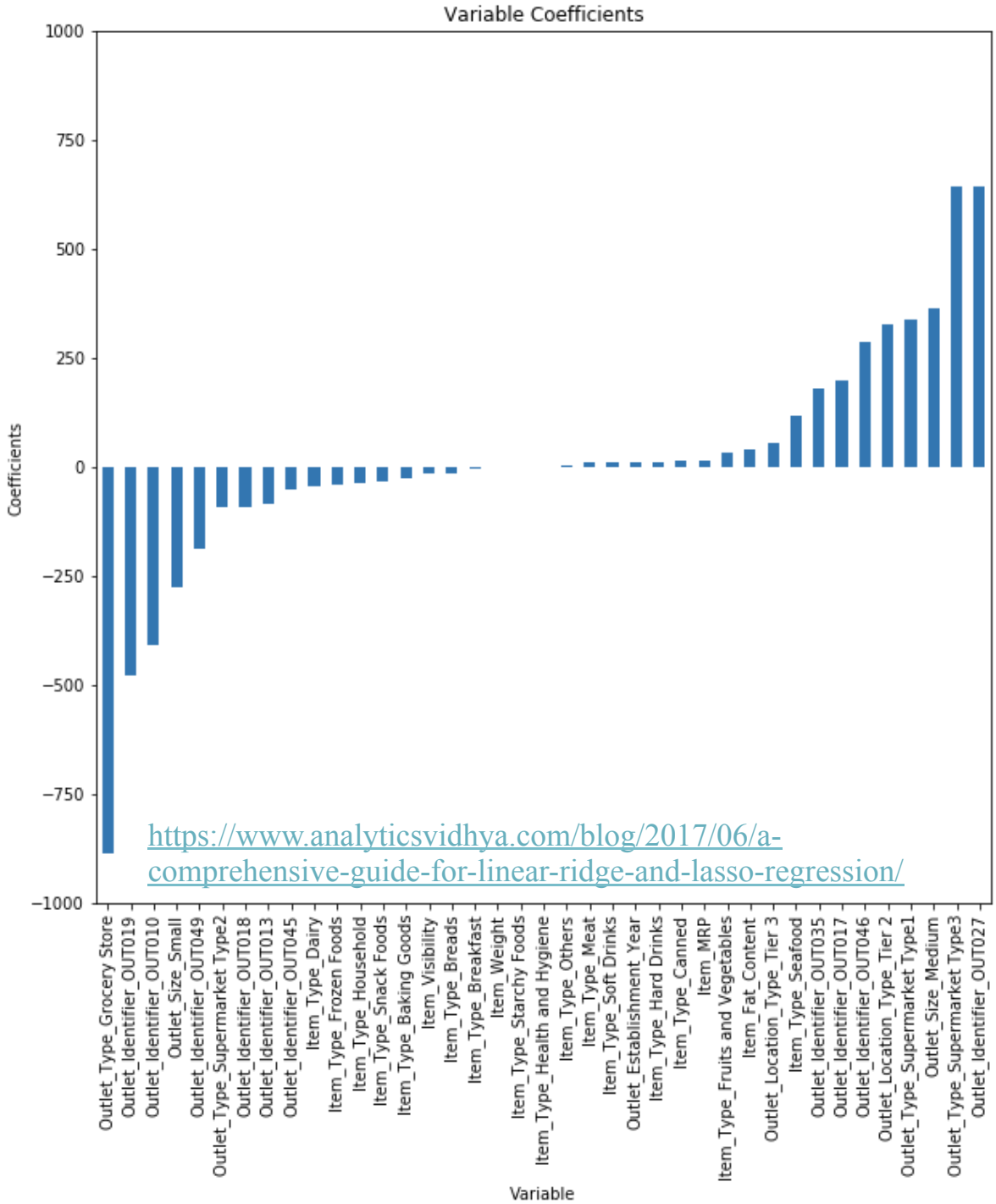
$$L1: \|\boldsymbol{w}\|_1 = \sum_{j=1}^{m} |w_j|$$

- For **L1 regularization**, we simply replace the square of the weights by the sum of the absolute values of the weights.
- L2 will not yield sparse models and all coefficients are shrunk by the same factor (none are eliminated).

L1 regularization

$$L1 : \|\boldsymbol{w}\|_1 = \sum_{j=1}^{m} |w_j|$$

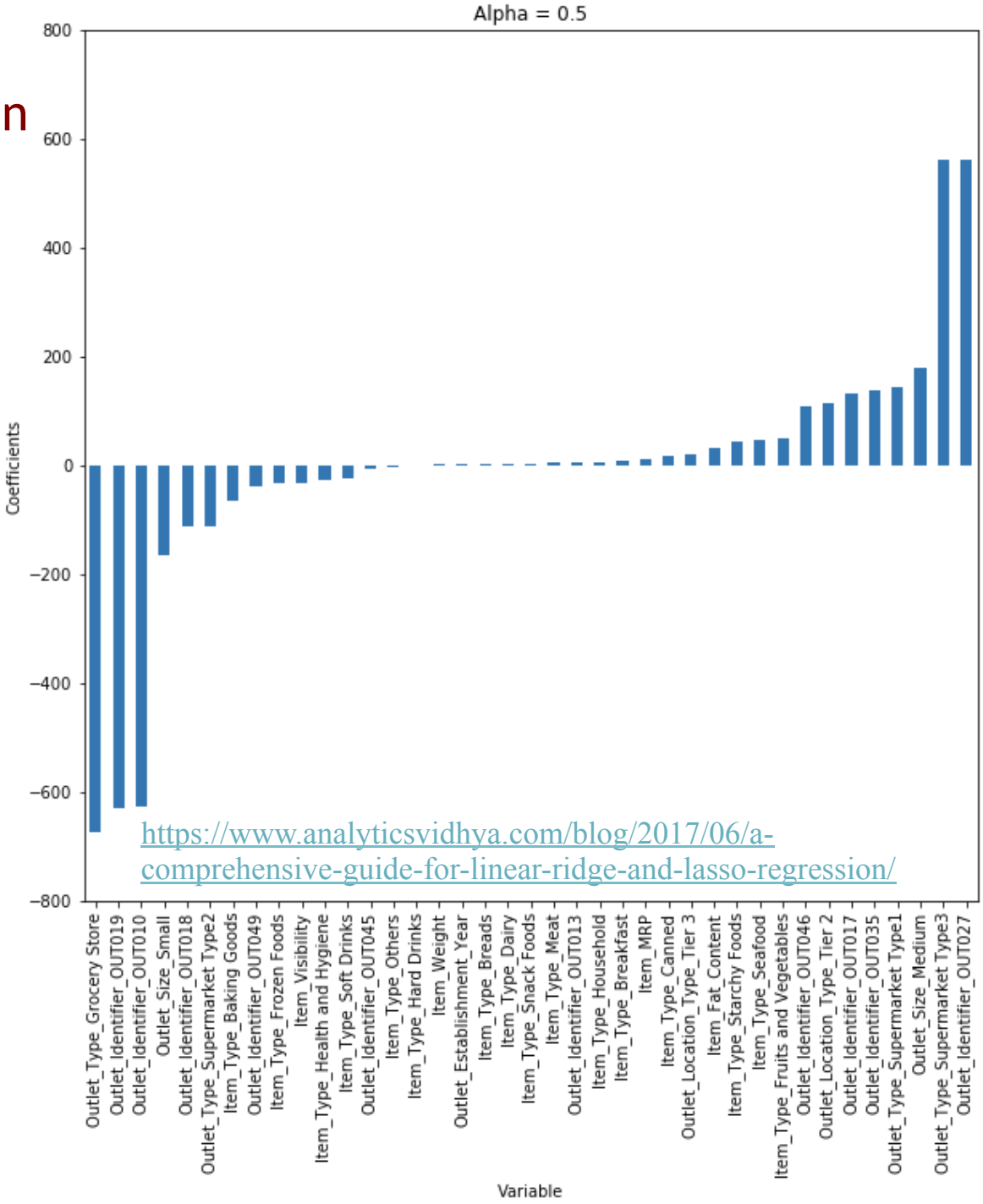
- L1 regularization usually yields sparse feature vectors; most feature weights will be zero.
- Sparsity can be useful in practice if we have a high-dimensional dataset with many features that are irrelevant, especially cases where we have more irrelevant dimensions than samples.
- In this sense, L1 regularization can be understood as a technique for feature selection.
- Some coefficients can become zero and they are eliminated.



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

L2 regularization

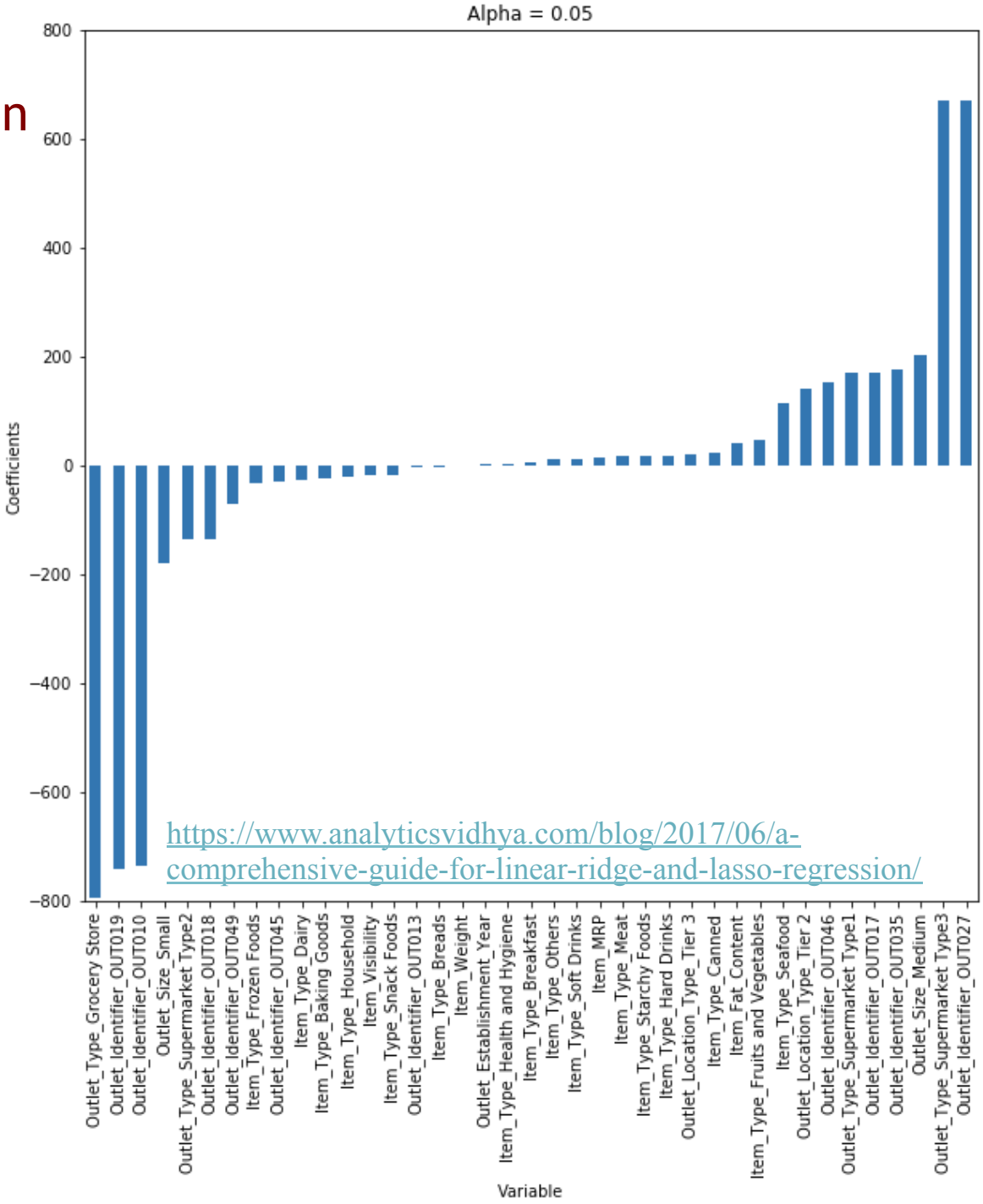
Ridge Regression



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

L2 regularization

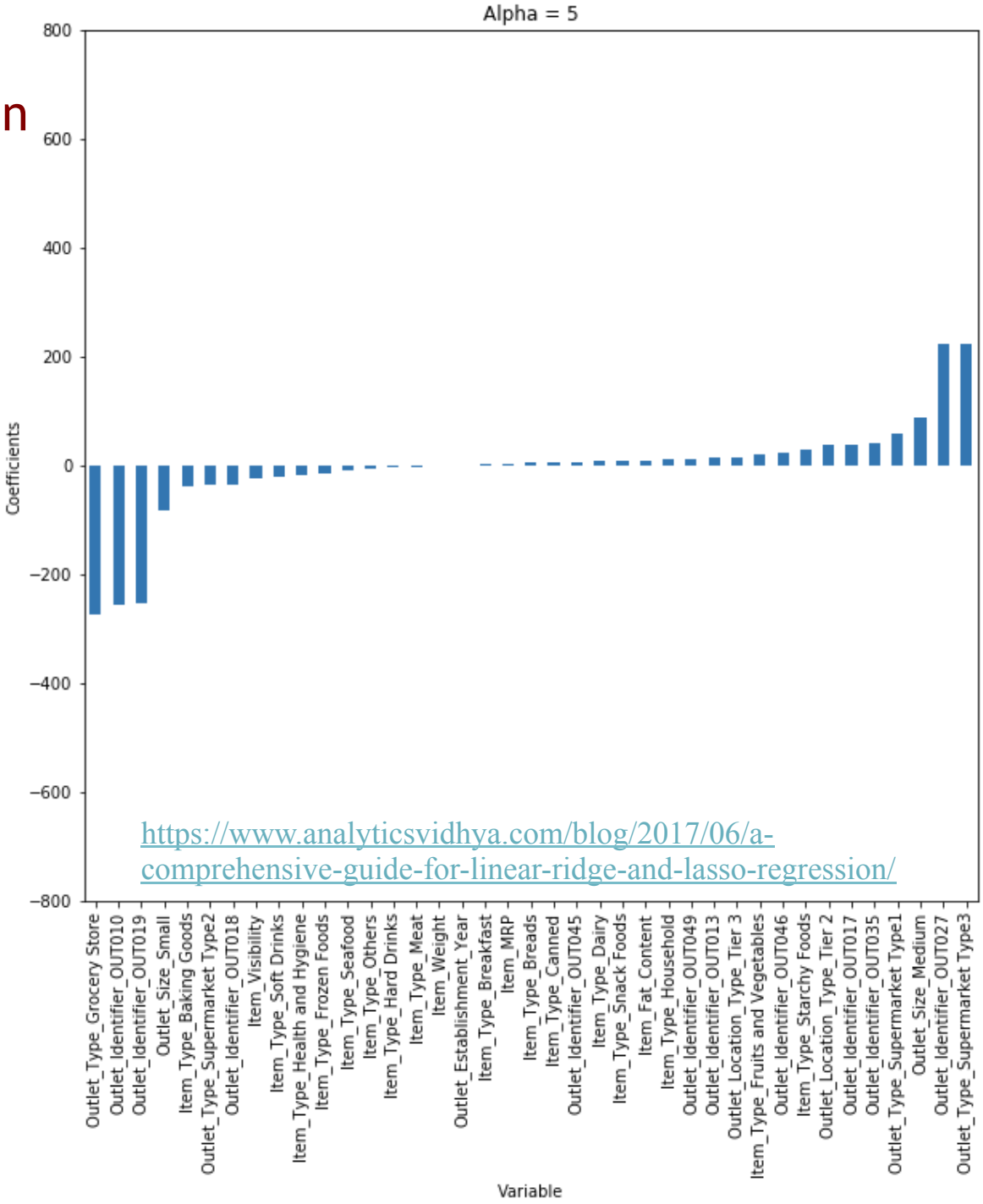
Ridge Regression



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

L2 regularization

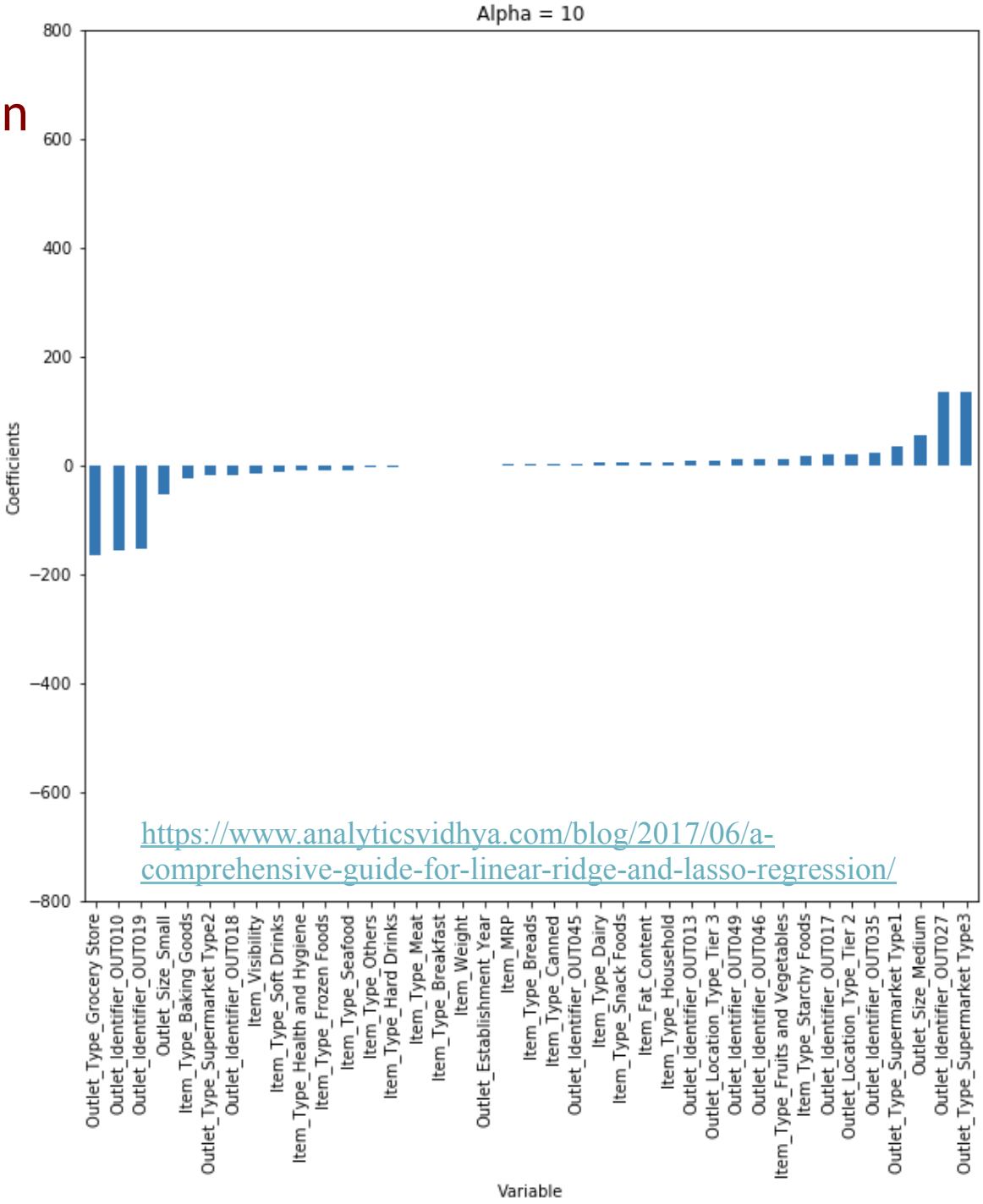
Ridge Regression



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

L2 regularization

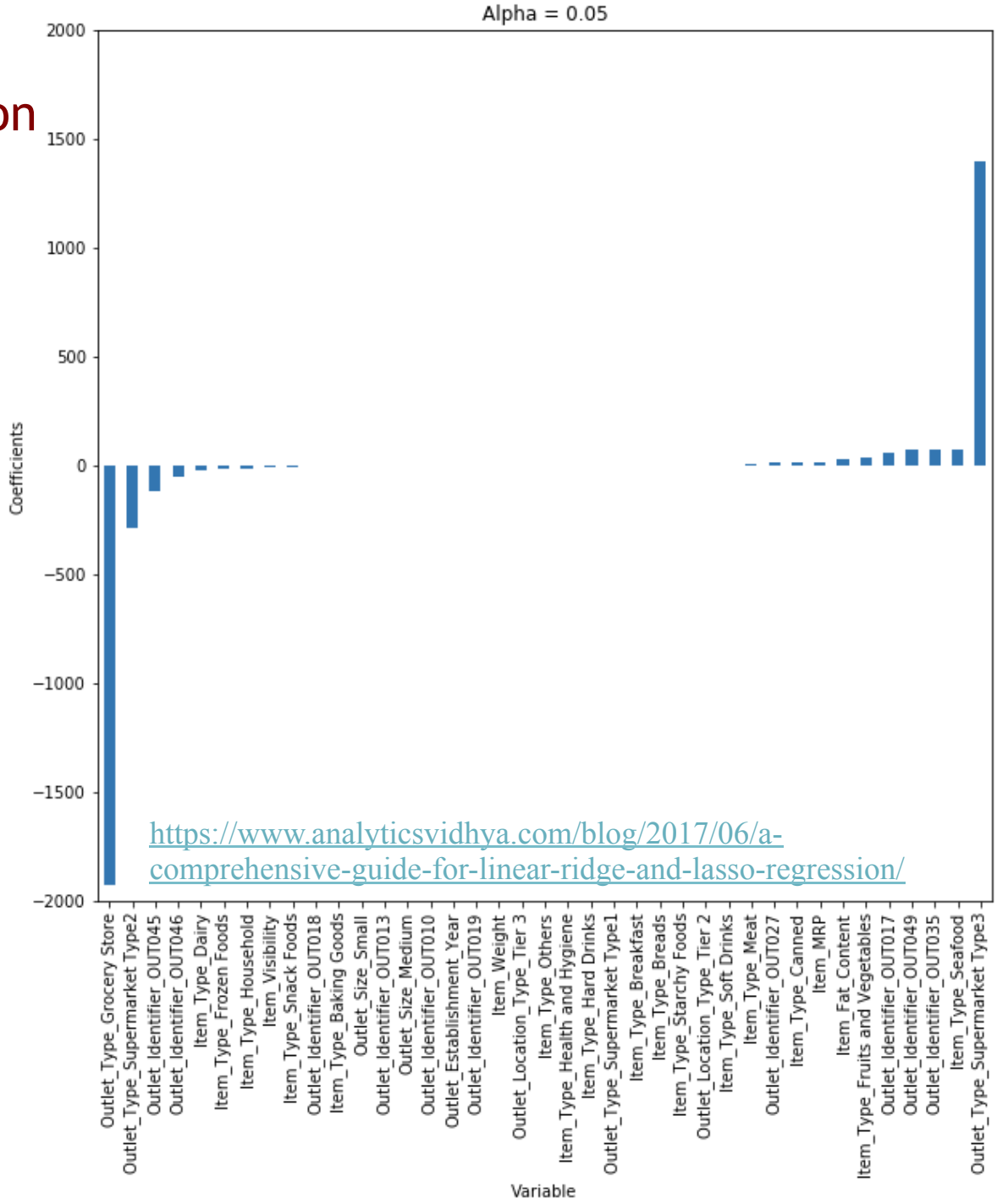
Ridge Regression



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

L1 regularization

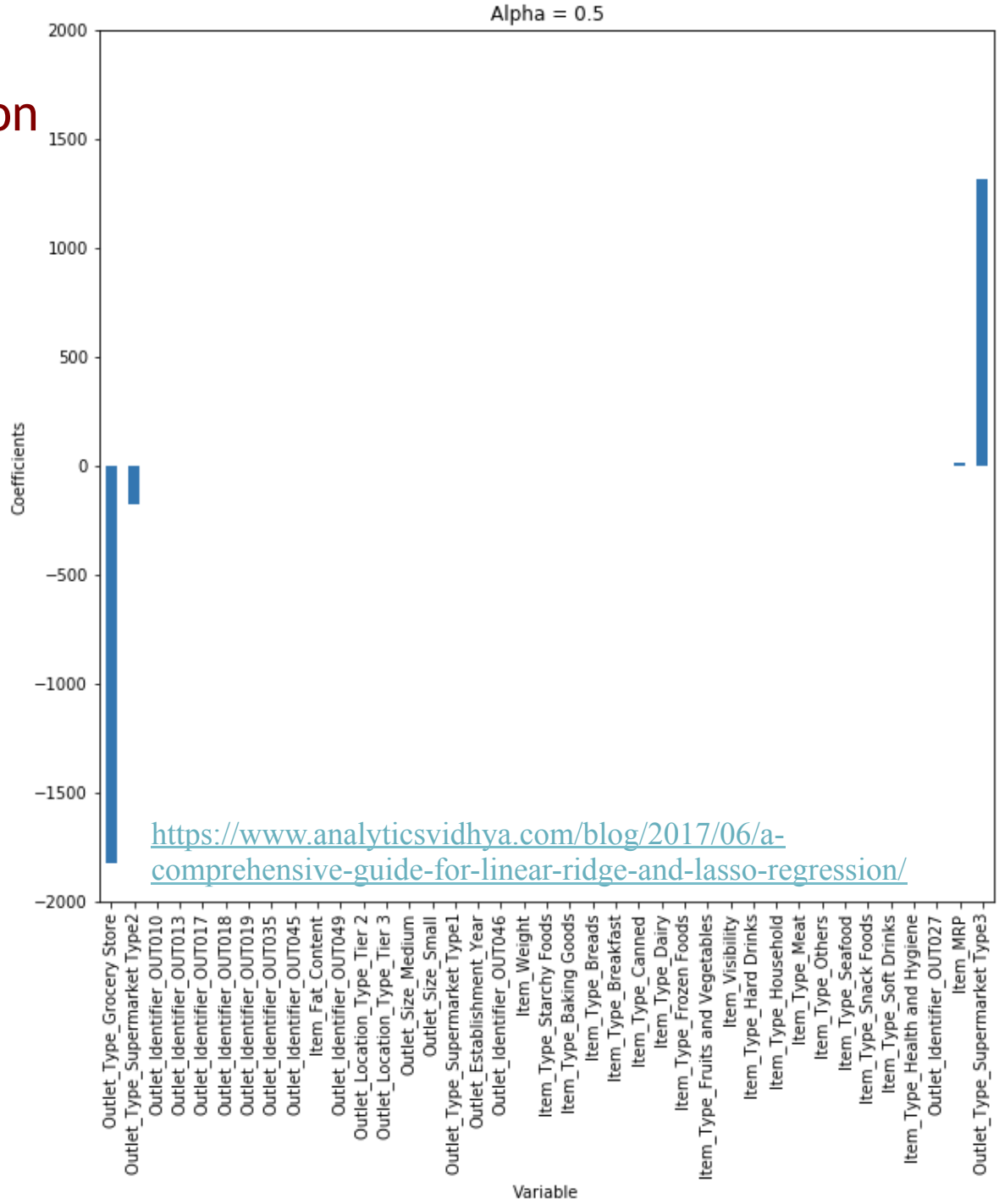
Lasso Regression



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

L1 regularization

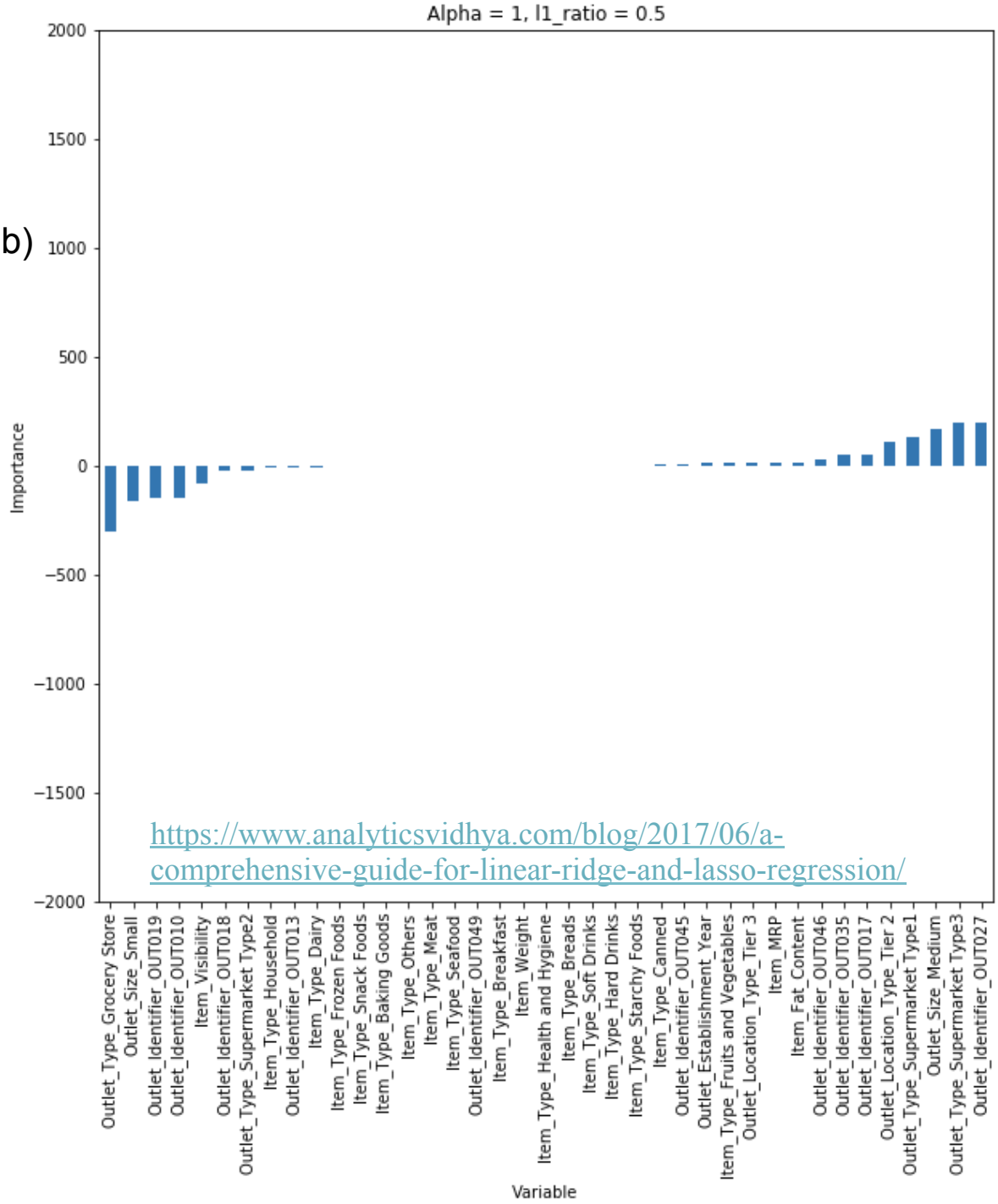
Lasso Regression



https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

Elastic Net Regression

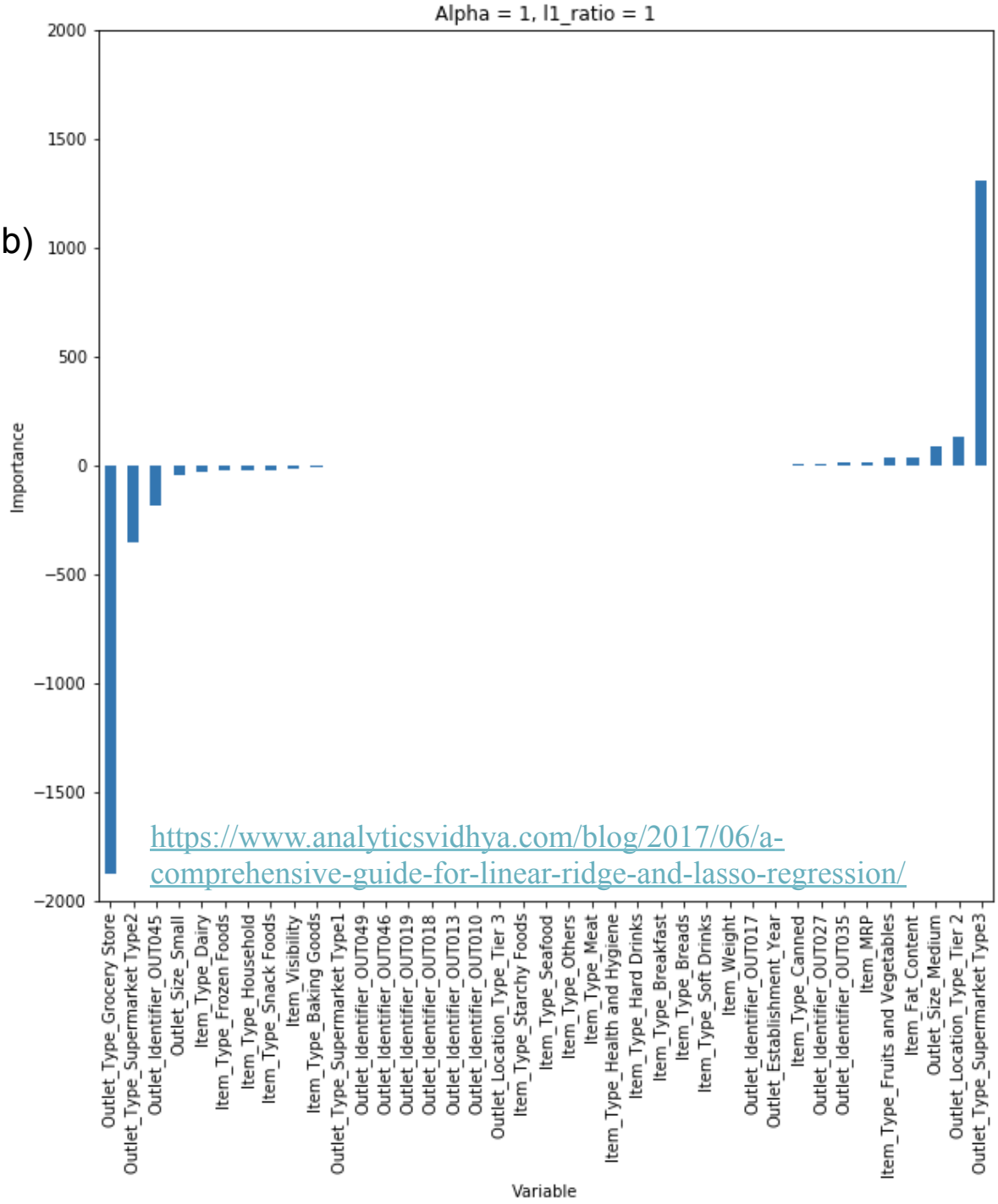
Alpha = a + b
l1_ratio = a / (a+b)



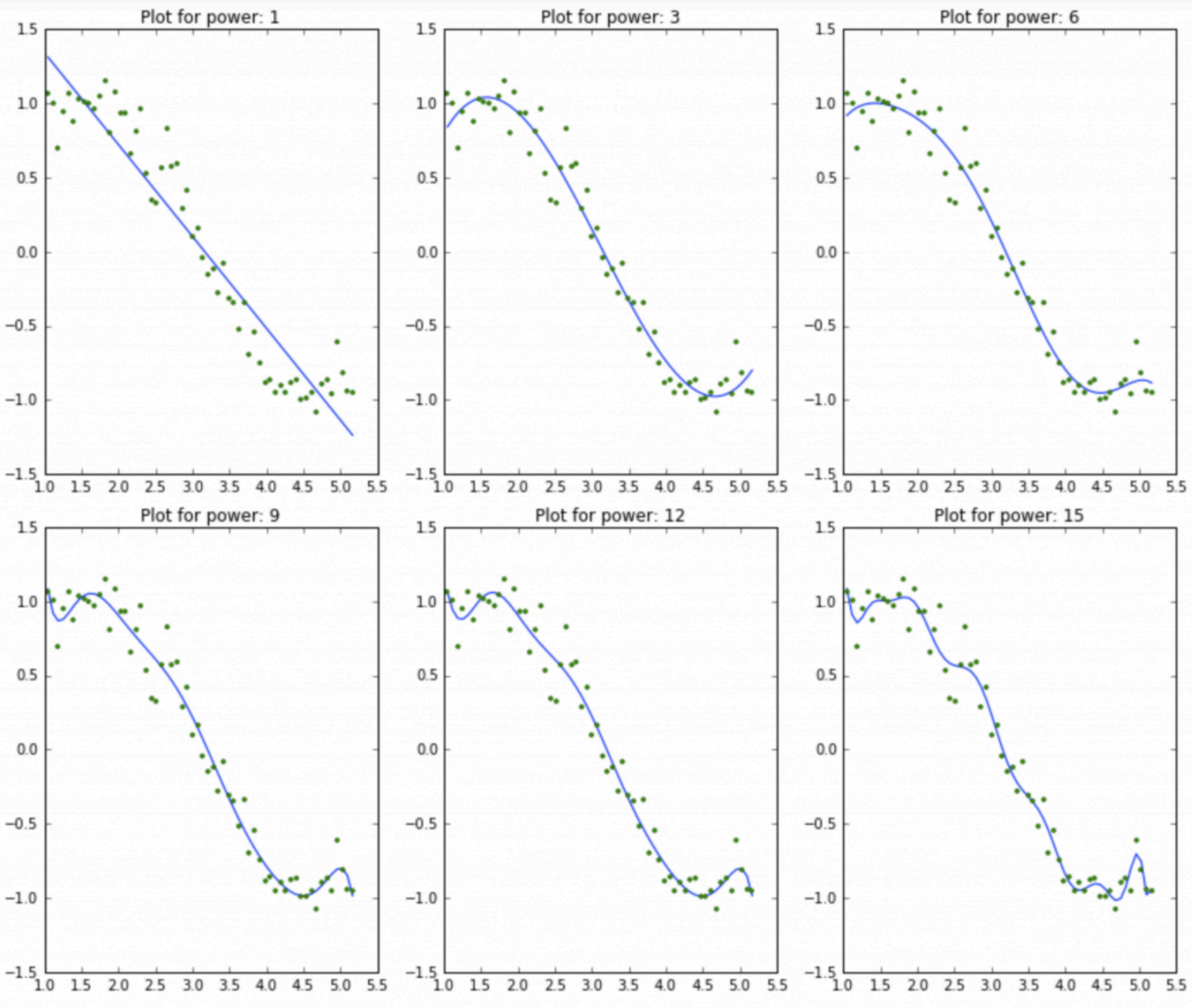
https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

Elastic Net Regression

Alpha = a + b
l1_ratio = a / (a+b)



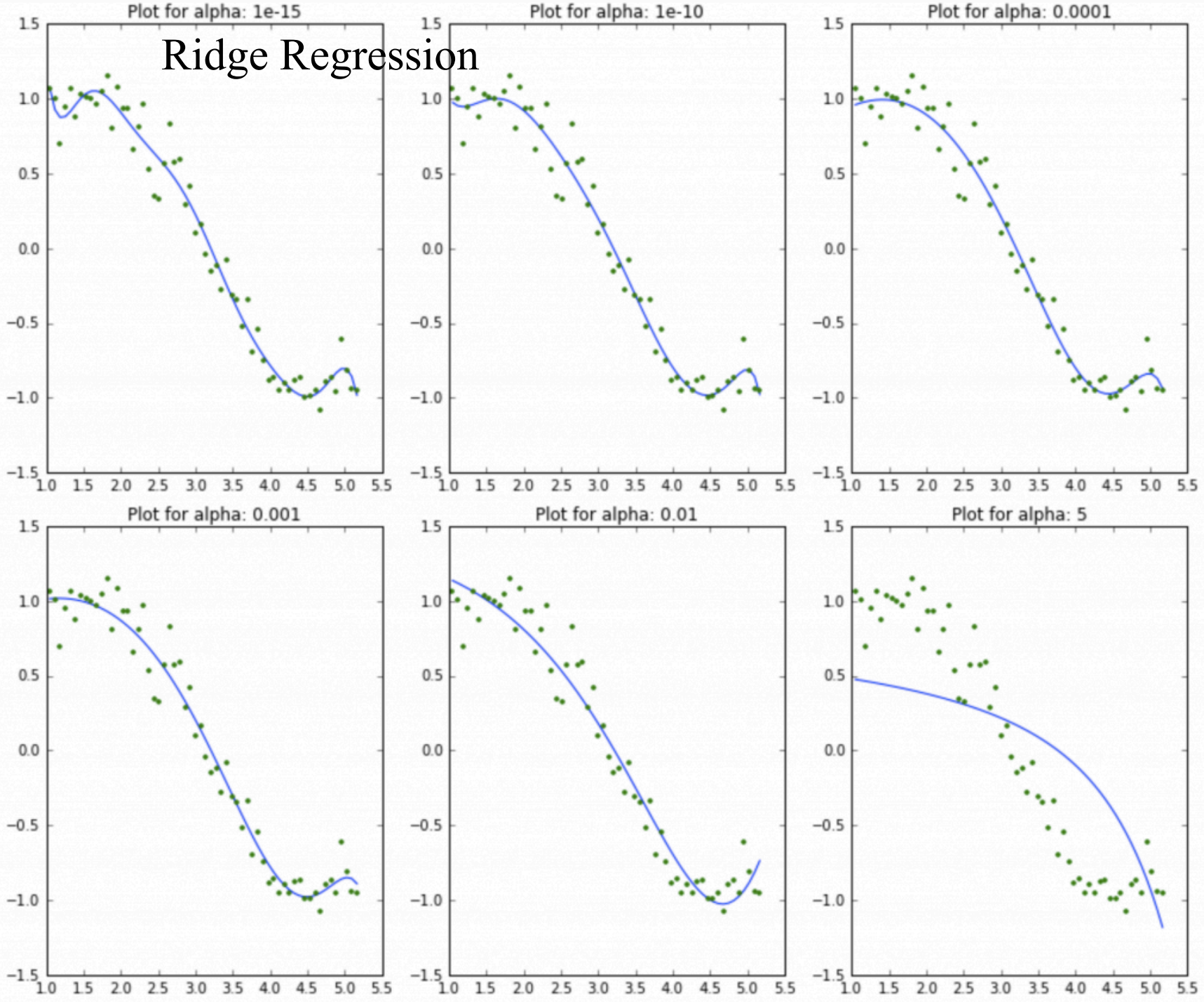
https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/


Plot for power: 1
Plot for power: 3
Plot for power: 6
Plot for power: 9
Plot for power: 12
Plot for power: 15

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	c
model_pow_1	3.3	2	-0.62	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_2	3.3	1.9	-0.58	-0.006	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_3	1.1	-1.1	3	-1.3	0.14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_4	1.1	-0.27	1.7	-0.53	-0.036	0.014	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_5	1	3	-5.1	4.7	-1.9	0.33	-0.021	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_6	0.99	-2.8	9.5	-9.7	5.2	-1.6	0.23	-0.014	NaN	NaN	NaN	NaN	NaN	N
model_pow_7	0.93	19	-56	69	-45	17	-3.5	0.4	-0.019	NaN	NaN	NaN	NaN	N
model_pow_8	0.92	43	-1.4e+02	1.8e+02	-1.3e+02	58	-15	2.4	-0.21	0.0077	NaN	NaN	NaN	N
model_pow_9	0.87	1.7e+02	-6.1e+02	9.6e+02	-8.5e+02	4.6e+02	-1.6e+02	37	-5.2	0.42	-0.015	NaN	NaN	N
model_pow_10	0.87	1.4e+02	-4.9e+02	7.3e+02	-6e+02	2.9e+02	-87	15	-0.81	-0.14	0.026	-0.0013	NaN	N
model_pow_11	0.87	-75	5.1e+02	-1.3e+03	1.9e+03	-1.6e+03	9.1e+02	-3.5e+02	91	-16	1.8	-0.12	0.0034	N
model_pow_12	0.87	-3.4e+02	1.9e+03	-4.4e+03	6e+03	-5.2e+03	3.1e+03	-1.3e+03	3.8e+02	-80	12	-1.1	0.062	-I
model_pow_13	0.86	3.2e+03	-1.8e+04	4.5e+04	-6.7e+04	6.6e+04	-4.6e+04	2.3e+04	-8.5e+03	2.3e+03	-4.5e+02	62	-5.7	0
model_pow_14	0.79	2.4e+04	-1.4e+05	3.8e+05	-6.1e+05	6.6e+05	-5e+05	2.8e+05	-1.2e+05	3.7e+04	-8.5e+03	1.5e+03	-1.8e+02	1
model_pow_15	0.7	-3.6e+04	2.4e+05	-7.5e+05	1.4e+06	-1.7e+06	1.5e+06	-1e+06	5e+05	-1.9e+05	5.4e+04	-1.2e+04	1.9e+03	-:

https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/

Ridge Regression

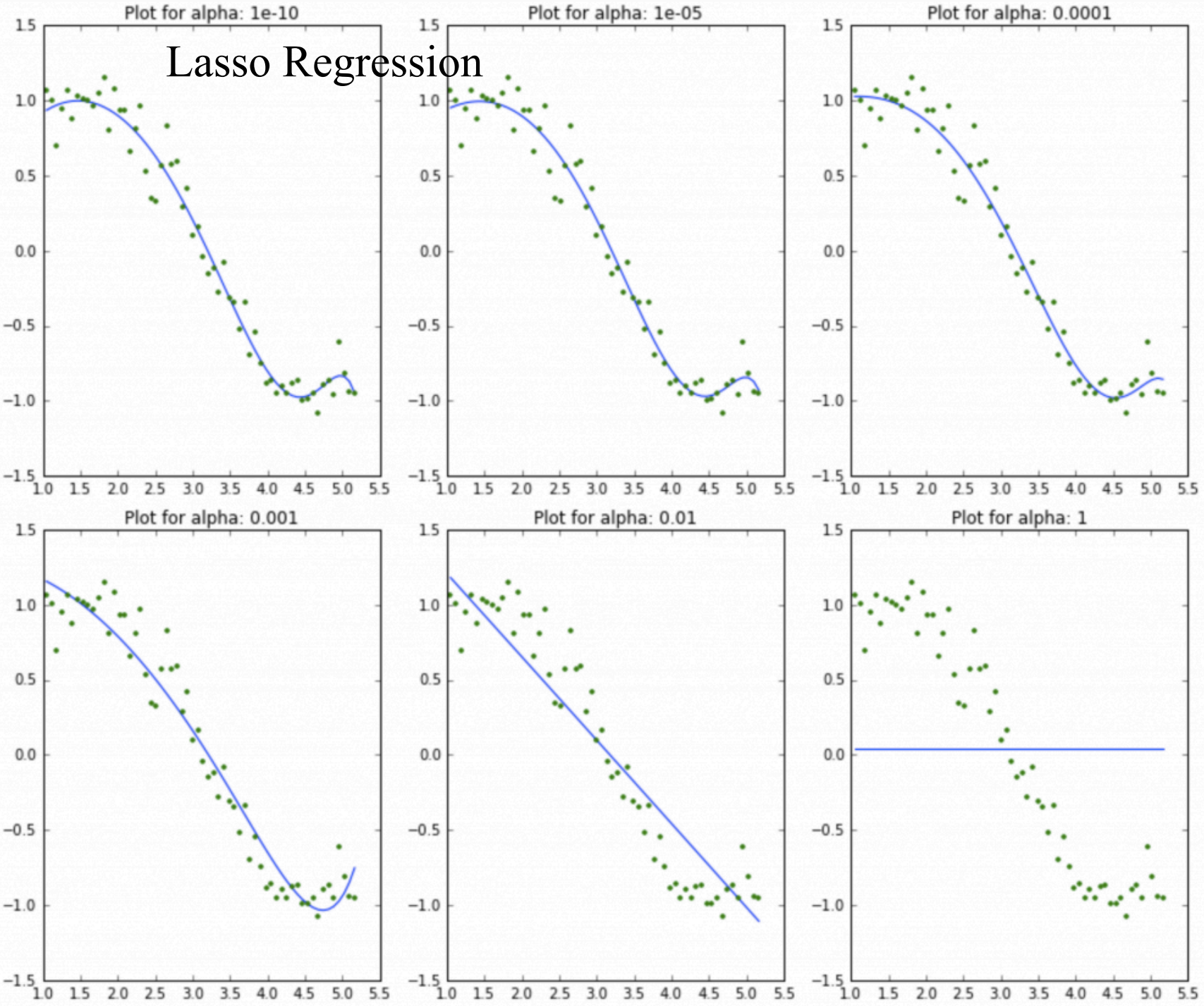


Ridge Regression

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	coe
alpha_1e-15	0.87	95	-3e+02	3.8e+02	-2.4e+02	66	0.96	-4.8	0.64	0.15	-0.026	-0.0054	0.00086	0.0
alpha_1e-10	0.92	11	-29	31	-15	2.9	0.17	-0.091	-0.011	0.002	0.00064	2.4e-05	-2e-05	-4.2
alpha_1e-08	0.95	1.3	-1.5	1.7	-0.68	0.039	0.016	0.00016	-0.00036	-5.4e-05	-2.9e-07	1.1e-06	1.9e-07	2e-
alpha_0.0001	0.96	0.56	0.55	-0.13	-0.026	-0.0028	-0.00011	4.1e-05	1.5e-05	3.7e-06	7.4e-07	1.3e-07	1.9e-08	1.9
alpha_0.001	1	0.82	0.31	-0.087	-0.02	-0.0028	-0.00022	1.8e-05	1.2e-05	3.4e-06	7.3e-07	1.3e-07	1.9e-08	1.7
alpha_0.01	1.4	1.3	-0.088	-0.052	-0.01	-0.0014	-0.00013	7.2e-07	4.1e-06	1.3e-06	3e-07	5.6e-08	9e-09	1.1
alpha_1	5.6	0.97	-0.14	-0.019	-0.003	-0.00047	-7e-05	-9.9e-06	-1.3e-06	-1.4e-07	-9.3e-09	1.3e-09	7.8e-10	2.4
alpha_5	14	0.55	-0.059	-0.0085	-0.0014	-0.00024	-4.1e-05	-6.9e-06	-1.1e-06	-1.9e-07	-3.1e-08	-5.1e-09	-8.2e-10	-1.3
alpha_10	18	0.4	-0.037	-0.0055	-0.00095	-0.00017	-3e-05	-5.2e-06	-9.2e-07	-1.6e-07	-2.9e-08	-5.1e-09	-9.1e-10	-1.6
alpha_20	23	0.28	-0.022	-0.0034	-0.0006	-0.00011	-2e-05	-3.6e-06	-6.6e-07	-1.2e-07	-2.2e-08	-4e-09	-7.5e-10	-1.4

https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/

Lasso Regression



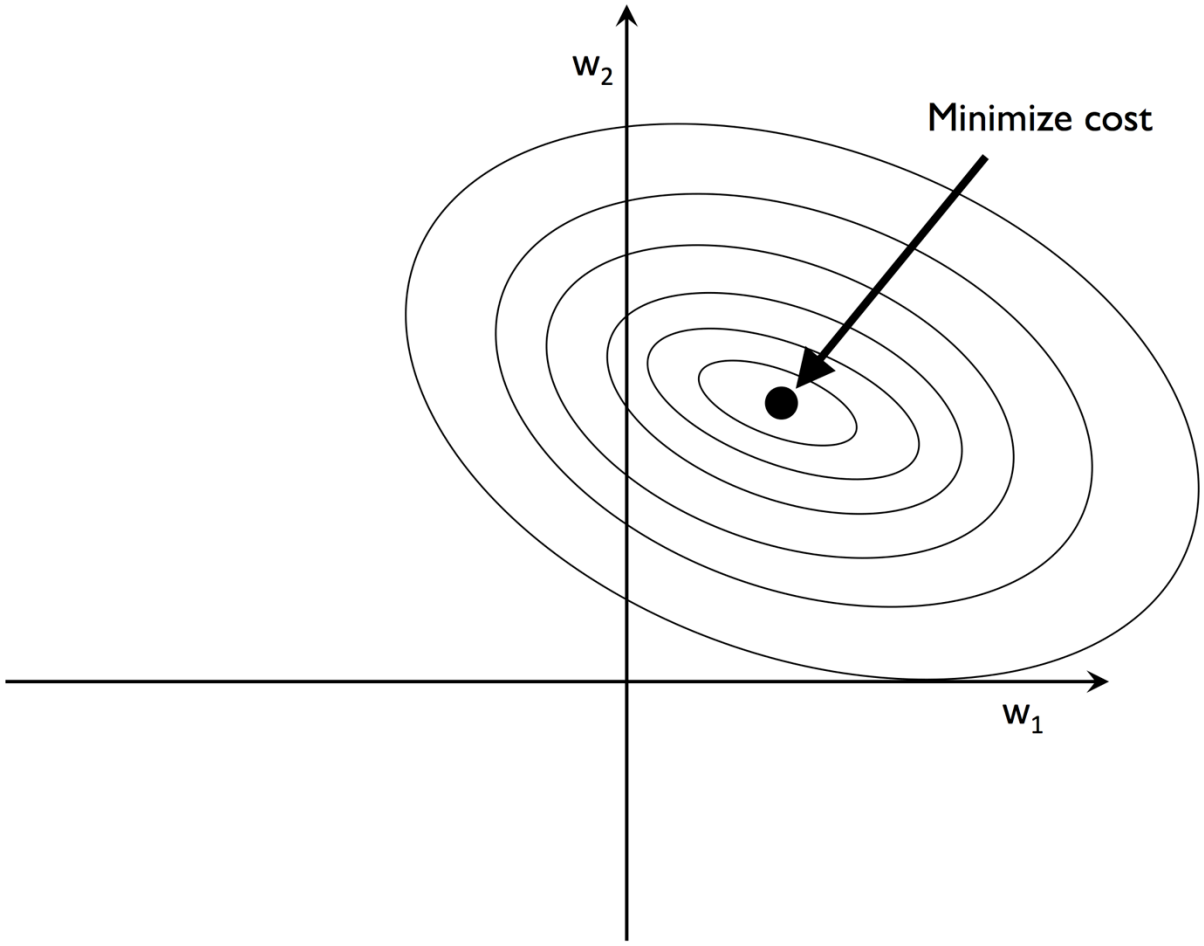
Lasso Regression

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	coe
alpha_1e-15	0.96	0.22	1.1	-0.37	0.00089	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4
alpha_1e-10	0.96	0.22	1.1	-0.37	0.00088	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4
alpha_1e-08	0.96	0.22	1.1	-0.37	0.00077	0.0016	-0.00011	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.3
alpha_1e-05	0.96	0.5	0.6	-0.13	-0.038	-0	0	0	0	7.7e-06	1e-06	7.7e-08	0	0
alpha_0.0001	1	0.9	0.17	-0	-0.048	-0	-0	0	0	9.5e-06	5.1e-07	0	0	0
alpha_0.001	1.7	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0	1.5e-08	7.5
alpha_0.01	3.6	1.8	-0.55	-0.00056	-0	-0	-0	-0	-0	-0	-0	0	0	0
alpha_1	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
alpha_5	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
alpha_10	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0

HIGH SPARSITY

https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/

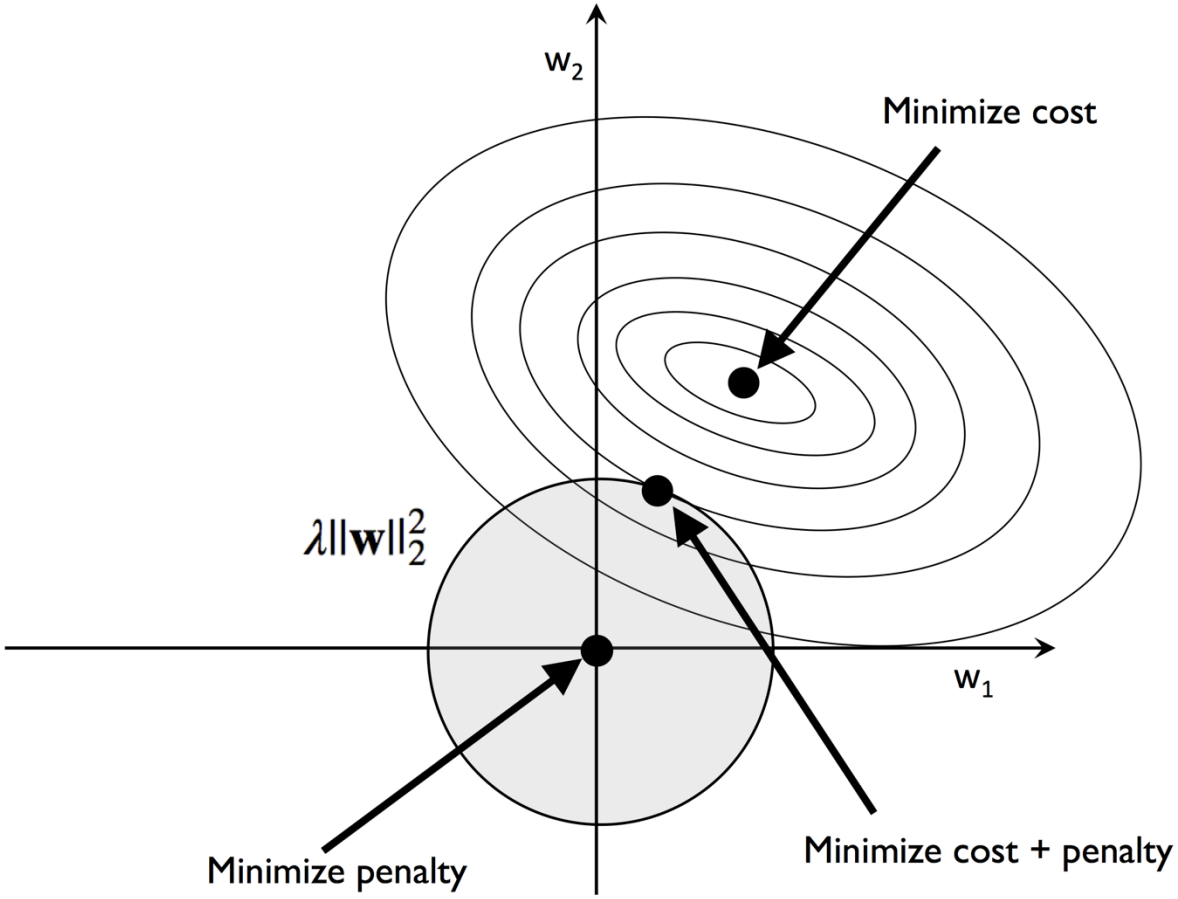
L1 and L2 regularization as penalties against model complexity



Contours of a convex cost function for two weight coefficients w_1 and w_2.

Here, we consider the Sum of Squared Errors (SSE) cost function for Adaline, since it is spherical and easier to draw than the cost function of logistic regression.

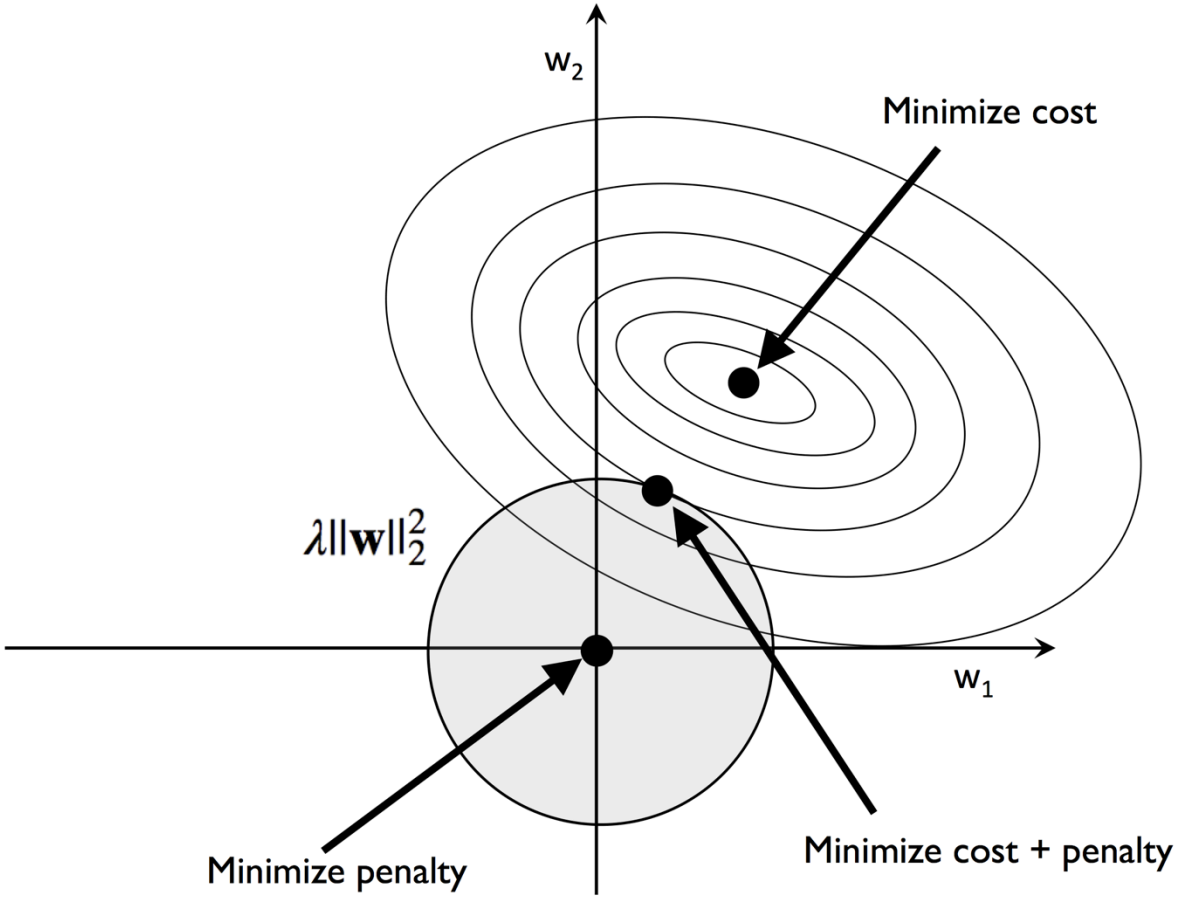
L2 regularization as penalty against model complexity



Regularization adds a penalty term to the cost function to encourage smaller weights; or in other words, **we penalize large weights**.

By increasing the regularization strength via the regularization parameter λ , we shrink the weights towards zero and decrease the dependence of our model on the training data.

L2 regularization as penalty against model complexity



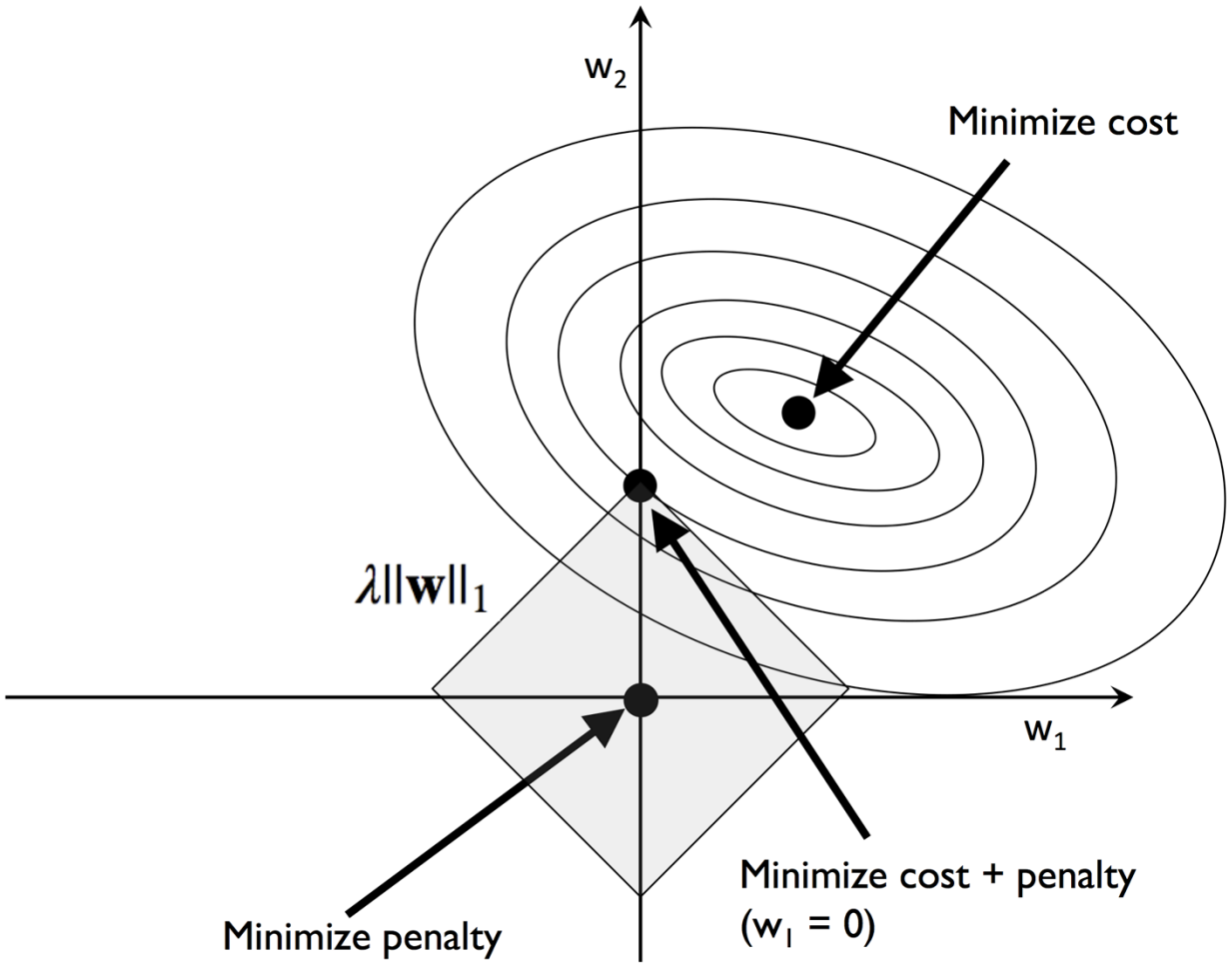
Goal:
Choose the point where the L2 ball intersects with the contours of the unpenalized cost function.

The larger the value of the regularization parameter λ gets, the faster the penalized cost grows, which leads to a narrower L2 ball.

Regularization goal

- Minimize the sum of:
  - unpenalized cost + the penalty term
    - which can be understood as:
      - adding bias
      - preferring a simpler model
    - to reduce the variance in the absence of sufficient training data to fit the model.

Sparse solutions with L1 regularization



The contour of the cost function touches the L1 diamond at $w_1 = 0$.

The intersection between the ellipses of the cost function and the boundary of the L1 diamond — is located on the axes, which encourages sparsity.

Ridge Regression

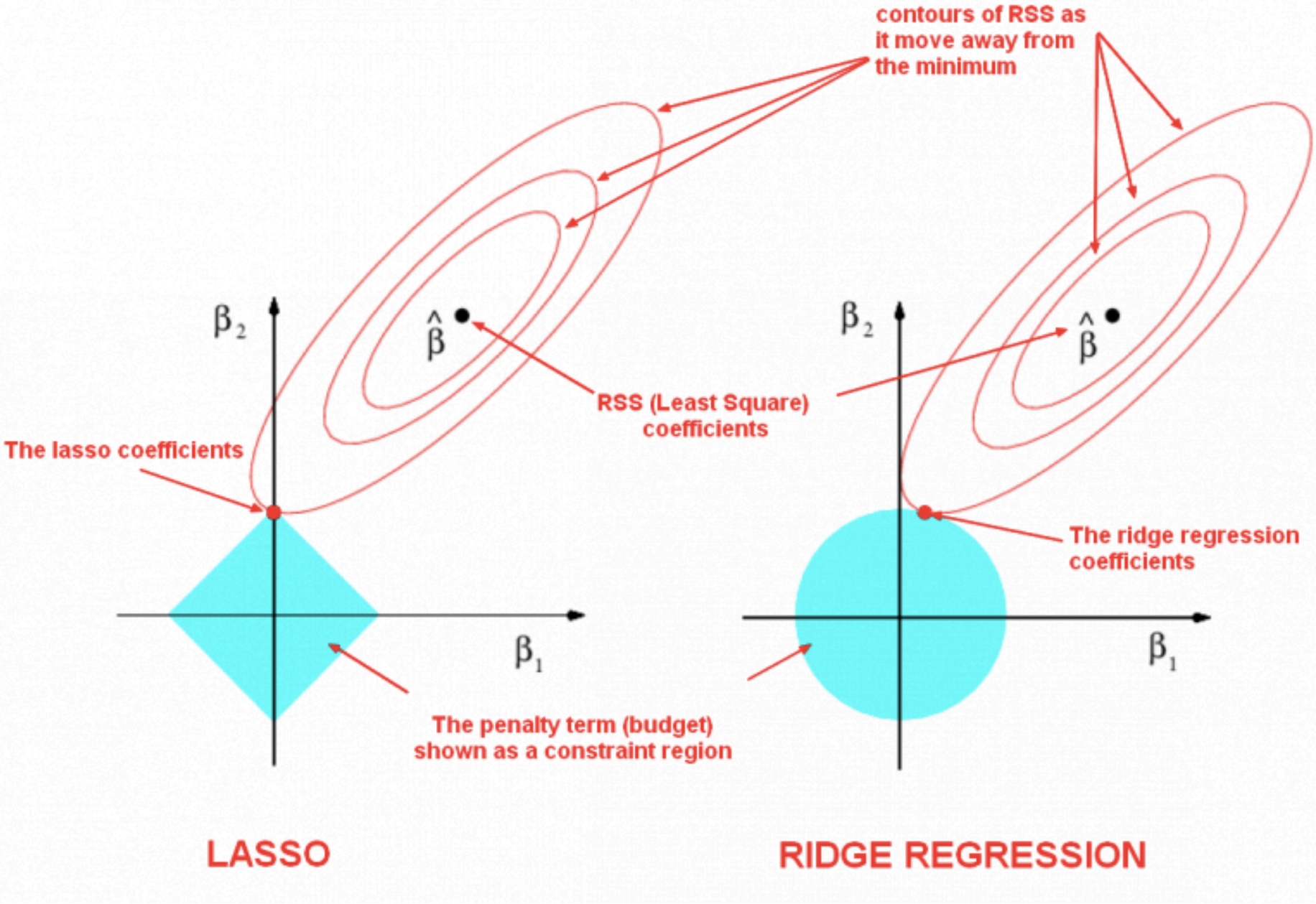
- It performs 'L2 regularization'.
  - It adds a factor of square of the magnitude of coefficients in the optimisation objective.
- Objective = RSS + λ * (sum of square of coefficients)

```
from sklearn.linear_model import Ridge
import numpy as np

n_samples, n_features = 10, 5
np.random.seed(0)
y = np.random.randn(n_samples)
X = np.random.randn(n_samples, n_features)
clf = Ridge(alpha=1.0)
clf.fit(X, y)
```

Lasso Regression

- Least Absolute Shrinkage and Selection Operator
- It performs 'L1 regularization'.
  - It adds a factor of sum of absolute value of coefficients in the optimisation objective.
- Objective = RSS + λ * (sum of absolute value of coefficients)

```
from sklearn import linear_model

clf = linear_model.Lasso(alpha=0.1)
clf.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
print(clf.coef_)
print(clf.intercept_)
```


contours of RSS as
it move away from
the minimum
β_2
$\hat{\beta}$
RSS (Least Square)
coefficients
The lasso coefficients
β_1
The penalty term (budget)
shown as a constraint region
β_2
$\hat{\beta}$
The ridge regression
coefficients
β_1
LASSO
RIDGE REGRESSION

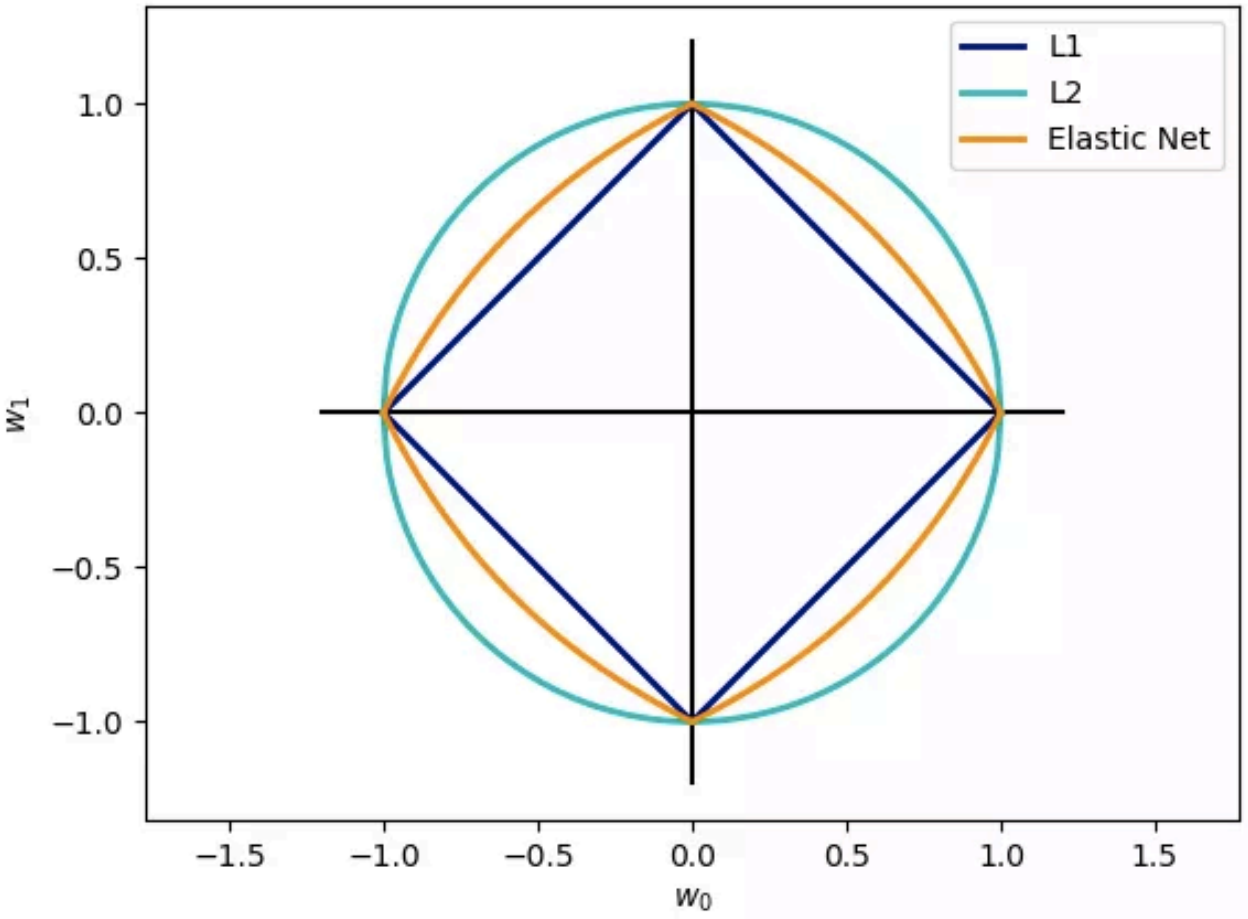
Elastic nets

- combine L1 & L2 methods.
- A penalty is applied to the sum of the absolute values and to the sum of the squared values.

$$ERROR_{L1L2=}ERROR + \lambda((1-\alpha)\sum_{0}^{N}\beta_i^2 + \alpha\sum_{0}^{N}|\beta_i|)$$

- Lambda is a shared penalization parameter while alpha sets the ratio between L1 and L2 regularization in the Elastic Net Regularization.
- Hence, we expect a hybrid behavior between L1 and L2 regularization.

Regularization



Scikit-learn:

penalty:

- str, ‘none’, ‘l2’, ‘l1’, or ‘elasticnet’
- Defaults to ‘l2’ which is the standard regularizer for linear SVM models.
- ‘l1’ and ‘elasticnet’ might bring sparsity to the model (feature selection) not achievable with ‘l2’.

Applications

- Ridge
  - In majority of the cases, it is used to prevent overfitting.
  - Since it includes all the features, it is not very useful in case of exorbitantly high features, say in millions, as it will pose computational challenges.

Applications

- Lasso
  - Since it provides sparse solutions, it is generally the model of choice (or some variant of this concept) for modelling cases where the features are in millions or more.
  - In such a case, getting a sparse solution is of great computational advantage as the features with zero coefficients can simply be ignored.

L_p norm

Sequential feature selection algorithms

- Family of greedy search algorithms that are used to reduce an initial d-dimensional feature space to a k-dimensional feature subspace where k<d.
- Motivation:
  - Automatically select a subset of features that are most relevant to the problem.
    - improve computational efficiency
    - reduce the generalization error of the model
- Useful for algorithms that don't support regularization.

Sequential Backward Selection (SBS)

- Classic sequential feature selection algorithm which aims to reduce the dimensionality of the initial feature subspace with a minimum decay in performance of the classifier to improve upon computational efficiency.
- In certain cases, SBS can even improve the predictive power of the model if a model suffers from overfitting.
- Greedy algorithms make locally optimal choices at each stage of a combinatorial search problem and generally yield a suboptimal solution to the problem.

Sequential Backward Selection (SBS)

- SBS sequentially removes features from the full feature subset until the new feature subspace contains the desired number of features.
- In order to determine which feature is to be removed at each stage, we need to define the criterion function J that we want to minimize.
- the criterion function can simply be the difference in performance of the classifier before and after the removal of a particular feature.
- At each stage we eliminate the feature that causes the least performance loss after removal.
- SBS algorithm has not been implemented in scikit-learn yet.

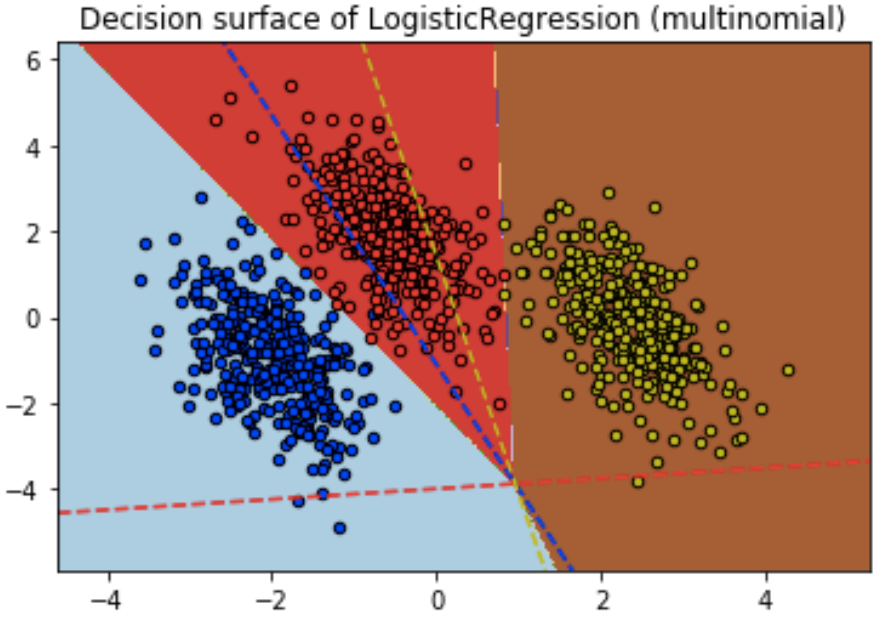
Feature selection algorithms in scikit-learn

- There are many more feature selection algorithms available via scikit-learn.
- Those include:
  - recursive backward elimination based on feature weights
  - tree-based methods to select features by importance
  - univariate statistical tests
- http://scikit-learn.org/stable/modules/feature_selection.html

Multiclass Logistic Regression

- sklearn.linear_model.LogisticRegression
  - `penalty : str, 'l1' or 'l2', default: 'l2'`
  - `solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}`
  - `multi_class : str, {'ovr', 'multinomial'}, default: 'ovr'`
    - The training algorithm uses the one-vs-rest (OvR) scheme if ‘multi_class’ is set to ‘ovr’.
    - Uses the cross-entropy loss if ‘multi_class’ is set to ‘multinomial’.
    - If the option chosen is ‘ovr’, then a binary problem is fit for each label.
    - Else the loss minimised is the multinomial loss fit across the entire probability distribution.
      - Does not work for liblinear solver.

Multinomial and One-vs-Rest Logistic Regression



Stochastic Gradient Descent (SGD)

- sklearn.linear_model.SGDClassifier
  - penalty : str, ‘none’, ‘l2’, ‘l1’, or ‘elasticnet’
    - squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net).
  - loss : str, default: ‘hinge’
    - ‘hinge’ gives a linear SVM.
    - Options: ‘hinge’, ‘log’, ‘modified_huber’, ‘squared_hinge’, ‘perceptron’,
    - or a regression loss: ‘squared_loss’, ‘huber’, ‘epsilon_insensitive’, or ‘squared_epsilon_insensitive’.

Multiclass Logistic Regression

```
Wine Dataset – multi_class: 'ovr' – penalty: 'l1'

X.shape:(178, 13)
np.unique(y):[1 2 3]
accuracy_score:1.0
intercept_:
[-1.26336366 -1.21618634 -2.37024246]
coef_:
[[ 1.24570162  0.18080713  0.74353603 -1.16136393  0.          0.
   1.17018229  0.          0.          0.          0.          0.54694578
   2.5105225 ]
 [-1.53703752 -0.38783998 -0.99522581  0.36515494 -0.05971764  0.
   0.6679069   0.          0.         -1.93444244  1.23318828  0.
  -2.23120752]
 [ 0.13569598  0.16855897  0.35714641  0.          0.          0.
  -2.43805494  0.          0.          1.56356741 -0.81860375 -0.4927206
   0.        ]]
```

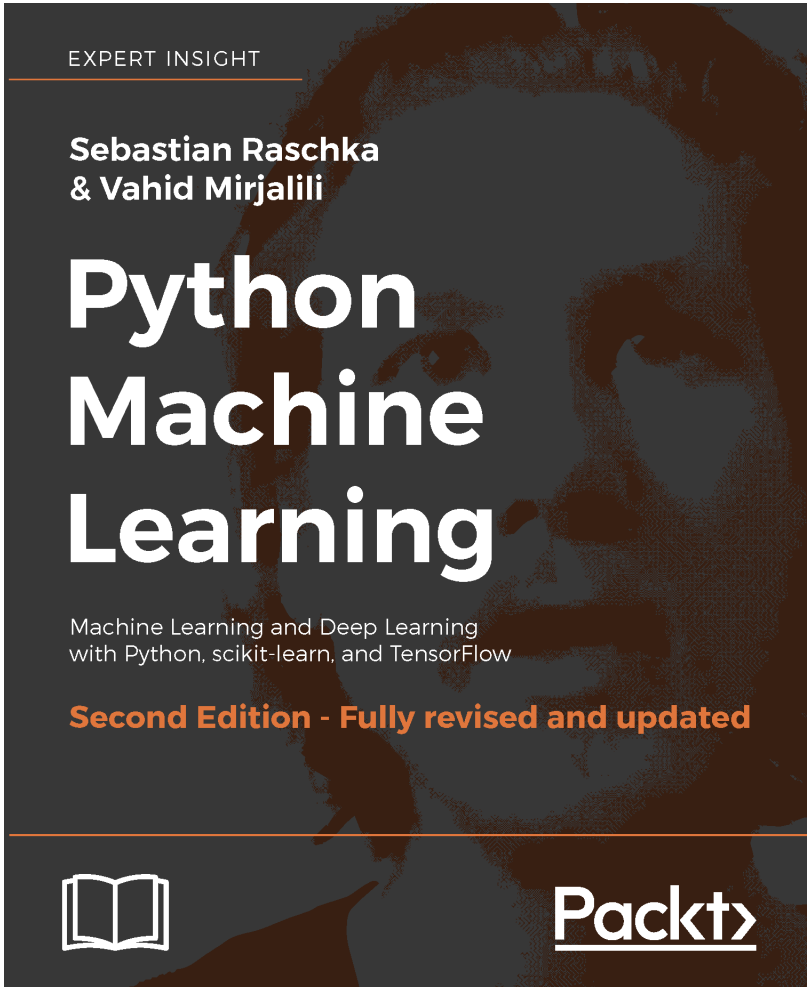
Multiclass Logistic Regression

```
Wine Dataset – multi_class: 'ovr' – penalty: 'l2'

X.shape:(178, 13)
np.unique(y):[1 2 3]
accuracy_score:1.0
intercept_:
[-1.33509136 -0.96973542 -2.04898785]
coef_:
[[ 1.27713853  0.38210274  0.8015599  -1.30842842  0.22782837  0.23101419
   0.90234371 -0.08423823  0.01462196 -0.0312838   0.02796323  0.71703048
   1.79262118]
 [-1.45395571 -0.620303   -1.05445248  0.67148394 -0.29048951  0.18277571
   0.51163918  0.10789643  0.08199321 -1.61228834  0.88800662  0.1659356
  -1.73246957]
 [ 0.38965148  0.4083047   0.40211468  0.26242969  0.15288658 -0.20064653
  -1.38792256 -0.06305419 -0.28440345  1.2553389  -0.93849662 -0.83821807
   0.13754706]]
```

Multiclass Logistic Regression

```
Wine Dataset – multi_class='multinomial', solver='newton-cg', penalty='l2'

X.shape:(178, 13)
np.unique(y):[1 2 3]
accuracy_score:1.0
intercept_:
[ 0.36862533  0.78036204 -1.14898737]
coef_:
[[ 0.78792639  0.24089338  0.4473384  -0.73489829  0.1113343   0.22607273
   0.60652205 -0.14847572  0.24131476  0.14140959  0.10670518  0.58061102
   0.98240089]
 [-0.95393153 -0.43668273 -0.76795841  0.50646632 -0.17622299  0.07110642
   0.38809799  0.09343743  0.11353057 -0.91231356  0.61090399  0.16780023
  -1.10002567]
 [ 0.16600514  0.19578935  0.32062001  0.22843197  0.06488869 -0.29717916
  -0.99462004  0.05503829 -0.35484533  0.77090397 -0.71760916 -0.74841125
   0.11762478]]
```

Main Reference



Python Machine Learning

Chapter 4 - Building Good Training Sets – Data Preprocessing

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regismagalhaes@ufc.br

UFC