LINEAR REGRESSION IN SCIKIT

LEARNING OBJECTIVES

- Scikit-learn
- Math basis for linear regression
- Loss function, Residual error, Mean Squared Error (MSE)
- linear_models with regularization in scikit: Ridge, Lasso, Elastic
- Linear regression review: p_values, R-Squared, confidence intervals
- Multicollinearity
- Polynomial regression using scikit-learn

PRE WORK & REVIEW

LAST LESSON REVIEW

- Matplotlib
- Seaborn
- Bokeh, Seaborn, Plot.ly

PROJECTS

- Final Project part I due lesson 8
- Project 2, due lesson 10
- Submission via new github repo ga-students:

LAST SESSION

ANY QUESTIONS FROM LAST CLASS?

SCIKIT-LEARN

SCIKIT-LEARN - OVERVIEW

- Classification, Regression and Clustering algorithms
- support vector machines, random forests, gradient boosting, k-means, and many others DBSCAN
- is designed to interoperate with Python scientific libraries NumPy and SciPy.
- Written in Python, with some core algorithms written in Cython

As of 2016, scikit-learn is under active development and is sponsored by INRIA, Telecom ParisTech

- Excellent and extensive algorithm implementation
- Simple conceptual API
- Fantastic documentation
- Super Efficient workflow

SCIKIT-LEARN - OVERVIEW

- Classification: Identifying to which category an object belongs to.
  - Applications: Spam detection, Image recognition.
  - Algorithms: SVM, nearest neighbors, random forest, ...
- Regression: Predicting a continuous-valued attribute associated with an object.
  - Applications: Drug response, Stock prices.
  - Algorithms: SVR, ridge regression, Lasso, ...
- Clustering: Automatic grouping of similar objects into sets.
  - Applications: Customer segmentation, Grouping experiment outcomes
  - Algorithms: k-Means, spectral clustering, mean-shift, ...

SCIKIT-LEARN - OVERVIEW

- Dimensionality reduction: Reducing the number of random variables to consider.
  - Applications: Visualization, Increased efficiency
  - Algorithms: PCA, feature selection, non-negative matrix factorization.
- Model selection: Comparing, validating and choosing parameters and models.
  - Goal: Improved accuracy via parameter tuning
  - Modules: grid search, cross validation, metrics.
- Preprocessing: Feature extraction and normalization.
  - Application: Transforming input data such as text for use with machine learning algorithms.
  - Modules: preprocessing, feature extraction.

EXTENSION - ECOSYSTEM

- Extensions and associated projects

SCIKIT-LEARN - API

Could not be simpler

1. select a model for instance

```
regr = linear_model.LinearRegression( some model tuning params, the metr
```

or

2. Train the model to the data

```
regr.fit(X, y)
```

3. Predict on new data

```
y_hat = regr.predict(Some New Data)
```

4. Score

```
regr.score(X,y)
```

SCIKIT-LEARN - DOCUMENTATION

Documentation of scikit-learn 0.17

Linear models

THE UNDERLYING MATH

LOSS FUNCTION

ORDINARY LEAST SQUARES (OLS)

Given X and y you want to find the best function f that minimizes

$$\|y - f(X)\|^2$$

Ordinary least squares (OLS) is the simplest and thus most common estimator. It is conceptually simple and computationally straightforward.

OLS estimates are commonly used to analyze both experimental and observational data. The OLS method minimizes the sum of squared residuals, and leads to a closed-form expression for the estimated value of the unknown parameter.

The estimator is unbiased and consistent if the errors have finite variance and are uncorrelated with the regressors

OLS - 1 DIMENSION

Let's say we have n samples:

- Variable $x = [x_1, \ldots, x_n]$
- Outcome $y = [y_1, \ldots, y_n]$

We want to find the *best* a and b such that for all x_i and y_i

$$y_i = a * x_i + b$$

Would work perfectly of the points (x_i, y_i) were on a line. But they are not!

OLS - 1 DIMENSION

So even for the best a and b possible we'll end up with some estimation
$\hat{y}_i = a * x_i + b$

- $\hat{y}_i \neq y_i$
-
  $$\|y - \hat{y}\| > \varepsilon$$

RESIDUALS

The residuals are

$$\varepsilon_i = y_i - \hat{y}_i$$
$$\varepsilon_i = y_i - (a * x_i + b)$$

for $i = [1, \ldots, n]$

The residuals are the **distance** between the **true** outcomes y_i and their estimates $\hat{y}_i$

We want to minimize the distance

LOSS FUNCTION

We do so by minimizing the L^2 norm of the residuals

$$||y - \hat{y}|| = ||y - (ax + b)||$$

$$||y - \hat{y}||^2 = \sum_{i=0}^{n} [y_i - (a * x_i + b)]^2$$

NORMS

$\mathbf{L^2}$ NORM

$$||x|| = \sqrt{x_1^2 + ... + x_n^2}$$

$\mathbf{L^1}$ NORM

$$|x| = |x_1| + ... + |x_n|$$

$\mathbf{L^\infty}$ NORM

$$|x|_\infty = max[|x_1|, ..., |x_n|]$$

LOSS FUNCTION

$$L(a, b) = ||y - \hat{y}||^2 = \sum_{i=0}^{n} [y_i - (a * x_i + b)]^2$$

- **Convex** function
- to find the minimum, set the derivative over a and b to 0
- that gives us 2 linear equations in a and b that we can solve

Wikipedia example)

MULTI DIMENSIONS

- m target $y = [y_1, \ldots, y_n]$
- n different variables $X = [(x_{i,j})]$

X is an (m by n) matrix

You want to find the (n+1) weights $\beta = [\beta_0, \beta_1, \ldots., \beta_n]$ that minimizes

$$L(\beta) = ||y - X\beta||$$

The closed form solution is

$$\hat{\beta} = (X^T.X)^{-1}X^Ty$$

LET'S PYTHON THAT: NOTEBOOK - LINEAR REGRESSION ALGEBRA

- 4 samples

```
input = np.array([ [1, 6], [2, 5], [3, 7], [4, 10] ])
```

- Matrix of predictors

```
X = np.matrix([np.ones(m), input[:,0]]).T
```

- Outcome vector

```
y = np.matrix(input[:,1]).T
```

- estimated regression weights $\hat{\beta} = (X^T.X)^{-1}X^Ty$

```
beta_hat = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```

- and plot it

AND NOW WITH SCIKIT

=> Notebook: L6 Linear Regression Scikit

MEAN SQUARED ERROR

A classic metric to assess your prediction is the Mean Squared Error (MSE) defined as

$$MSE = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Go back to the notebook and calculate the MSE for both the Ozone ~ Temp and Ozone ~ Wind

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_true, y_pred)
```

Other available metrics

R SQUARED AND MULTI COLINEARITY

Wind and Temp are negatively correlated

```
df.corr() gives a -0.49 correlation for Wind vs Temp
```

Let's compare Ozone ~ Wind, Ozone ~ Temp and Ozone ~ Wind + Temp

- MSE
- R^2

```
sklearn.metrics.r2_score
```

- p-values

```
from sklearn import feature_selection, linear_
pvals = feature_selection.f_regression(X, y)[1
```

- the model's coefficients

Conclusion?

RIDGE

LASSO

ELASTIC

THE LOSS FUNCTION

The loss function for linear regression is defined as

$$Loss(w) = \min_{w} \| Xw - y \|_2^2$$

- w are the coefficients aka weights
- X is the matrix of inputs
- $\hat{y} = Xw$
- y is the predictor
- $\|x\|_2$ is the L^2 Norm $\|x\|_2^2 = \sum x_i^2$

RIDGE

By introducing a *Regularization* term in the loss function, we change the model

RIDGE

$$Loss(w) = \min_{w} \| Xw - y \|_2^2 + \alpha \|w\|_2^2$$

- α is typically a penalty on the complexity of the regression model's weights.
- Makes the coefficients more robust to collinearity.
- Imposes a sort of **Ockham's razor** constraint on the coefficients.

RIDGE

Notebook - L6 OLS vs Ridge

```
clf = Ridge()
```

Use: X, y, w = make_regression(n_samples=100, n_features=3, coef=True, random_state=88)

LASSO

Instead of using the L^2 norm for the regularization term (Ridge) we use the L^1 norm (sum of absolute value)

$$Loss(w) = \min_{w} \frac{1}{2n} \| Xw - y \|_2^2 + \alpha |w|_1$$

- Reduces the number of weights

ELASTIC

Combination of Ridge (L^2) and Lasso (L^1).

The loss function is in this case

$$Loss(w) = \min_w \frac{1}{2n} \parallel Xw - y \parallel_2^2 + \alpha\rho|w|_1 + \frac{\alpha(1-\rho)}{2} \|w\|_2^2$$

with 2 parameters α and ρ

- Reduced number of weights
- Better robustness to multicolinearity - Less Variance

INFLUENCE OF REGULARIZATION IN RIDGE

- Set the α from 10^{-6} to 10^{+6}
- Evolution of the estimated weights values
- Evolution of Mean Squared Error of the weights

Notebook - L6 Ridge coefficients as a function of the L2 regularization

POLYNOMIAL REGRESSION USING SCIKIT-LEARN

- Load the polynomial.csv data

```
X = df.x.reshape(-1, 1)
y = df.y
```

- plot it
- fit a simple regression and plot the y_hat. What's the MSE?
- fit a regression with x and x^2 as features. What's the MSE?

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(2)
X2 = poly.fit_transform(X)
```

- Same thing with a 16 order polynomial. MSE? What's the problem?

Solution

LESSON REVIEW

LESSON REVIEW

- math basis for linear regression
- Loss function, Residual error, Mean Squared Error (MSE)
- Linear regression review: p_values, R-Squared
- Multicollinearity
- linear_models in scikit: Ridge, Lasso, Elastic
- Polynomial regression using scikit-learn

BEFORE NEXT CLASS

5 QUESTIONS ABOUT TODAY

EXIT TICKET