# Machine Learning Documentation

*Release 0.1*

**mansenfranzen**

**Aug 05, 2017**

# Contents

1 Linear Regression 3

2 Transformation 7

Contents:

CHAPTER 1

# Linear Regression

Documentation for the linear regression module.

mlearn.algorithms.linreg.**sum_of_squared_residuals**(*x*, *y*, *beta*)

Calculate the sum of squared residuals for observed $y$ values, given $x$ predictor matrix and provided $\beta$ parameters.

**Parameters**

- **x** (*np.array*) – Corresponds to a matrix $x$ which contains all $x_i$ values (including $x_0 = 1$).
- **y** (*np.array*) – Corresponds to vector $y$ which refers to the observed values.
- **beta** (*np.array*) – Corresponds to vector $\beta$ which contains the parameters to calculate the predicted values.

**Returns** ssr

**Return type** numpy.array

### References

### Notes

$$SSR(\beta) = \sum_{i=1}^{n}(x_i\beta - y_i)^2$$

$$SSR(\beta) = (x\beta - y)^T(x\beta - y)$$

### Examples

```
>>> # The predictor matrix
>>> x = np.array([[1, 11, 104],
                  [1, 15, 99],
                  [1, 22, 89],
                  [1, 27, 88]])
>>> # The observed values vector
>>> y = np.array([[12],
                  [15],
                  [19],
                  [22]])
>>> # The parameters vector
>>> beta_zero = np.zeros((3,1))
>>> sum_of_squared_residuals(x, y, beta_zero)
[ 1214.]
```

mlearn.algorithms.linreg.**gradient_descent**(*x*, *y*, *beta_init=None*, *gamma=0.01*, *max_iter=200*, *threshold=0.01*, *scaling=True*, *regularize=False*)

Numerically estimates the unknown parameters $\beta_i$ for a linear regression model where $x$ refers to the predictor matrix and and $y$ to the observed values vector.

The first derivative of the sum of the squared residuals is used to calculate the gradient for each parameter. In every iteration, the parameter is changed in decreasing direction of the gradient with the given step size $\gamma$. This is done until the maximum iteration amount is reached or the difference of sum of squared residuals between two iterations falls below a given threshold.

**Parameters**

- **x** (*np.array*) – Corresponds to a matrix $x$ which contains all $x_i$ values (including $x_0 = 1$).
- **y** (*np.array*) – Corresponds to vector $y$ which refers to the observed values.
- **beta_init** (*np.array, optional*) – Initial $\beta_i$ values may be provided. Otherwise they are set to zero.
- **gamma** (*float, optional*) – The step size $\gamma$ of the gradient descent. Determines how much parameters change per iteration.
- **max_iter** (*float, optional*) – Sets the maximum number of iterations.
- **threshold** (*float, optional*) – Define the threshold for convergence. If the difference of sum of the squared residuals between two consecutive iterations falls below this value, the gradient descent has converged and the function stops.
- **scaling** (*boolean, optional*) – By default, the predictors are z-transformed. This improves the gradient descent performance because all predictors behave on the same scale.
- **regularize** (*float, optional*) – Apply the regularization term $\lambda$ to the estimation of $\beta$. It can prevent overfitting when $x$ contains a large number of higher order predictors. Increasing $\lambda$ will decrease $\beta_i$ values which causes the decision boundary to be smoother.

**Returns** beta

**Return type** numpy.array

### Notes

$$f(\beta) = \frac{1}{2n}\sum_{i=1}^{n}(x_i\beta - y_i)^2 = \frac{1}{2n}(x\beta - y)^T(x\beta - y) = \frac{1}{2n}SSR(\beta)$$

$$f'(\beta) = \beta - \frac{\gamma}{n}\sum_{i=1}^{n}(x_i\beta_m - y_i)x_i = \beta - \gamma\frac{1}{n}x^T(x\beta - y)$$

$$f'_{reg}(\beta) = \beta(1 - \gamma\frac{\lambda}{n}\beta_{reg}) - \frac{\gamma}{n}x^T(x\beta - y) \text{ where } \beta_{reg} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1_m \end{bmatrix}$$

### References

[4] https://en.wikipedia.org/wiki/Gradient_descent

`mlearn.algorithms.linreg.`**`ordinary_least_squares`**(*x*, *y*, *regularize=False*)

Analytically calculate the unknown parameters $\beta$ for a linear regression model where $x$ refers to the predictor matrix and and $y$ to the observed values vector.

**Parameters**

- **x** (*np.array*) – Corresponds to a matrix $x$ which contains all $x_i$ values (including $x_0 = 1$).
- **y** (*np.array*) – Corresponds to vector $y$ which refers to the observed values.
- **regularize** (*float, optional*) – Apply the regularization term $\lambda$ to the estimation of $\beta$. It can prevent overfitting when $x$ contains a large number of higher order predictors. Increasing $\lambda$ will decrease $\beta_i$ values which causes the decision boundary to be smoother.

**Returns** **beta**

**Return type** numpy.array

### Notes

$$\hat{\beta} = (x^Tx)^{-1}x^Ty$$

Regularization with m predictors: $\hat{\beta} = (x^Tx + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & 1_{m+1} \end{bmatrix})^{-1}x^Ty$

### References

### Examples

```
>>> # The predictor matrix
>>> x = np.array([[1, 11, 104],
                  [1, 15, 99],
                  [1, 22, 89],
                  [1, 27, 88]])
>>> # The observed values vector
>>> y = np.array([[12],
                  [15],
                  [19],
                  [22]])
>>> # The parameters vector
>>> beta_zero = np.zeros((3,1))
>>> sum_of_squared_residuals(x, y, beta_zero)
[ 1214.]
>>> beta_min = ordinary_least_squares(x, y)
>>> sum_of_squared_residuals(x, y, beta_min)
[ 0.14455509]
```

CHAPTER 2

# Transformation

Documentation for the transformation module.

mlearn.algorithms.transform.**standardize**(*x*)

Perform z-transformation to get standard score of input matrix.

**Parameters** **x** (*numpy.array*) – Input matrix to be standardized.

**Returns** **x_stand** – Standardized matrix.

**Return type** numpy.array

### Notes

$$z = \frac{x - \mu}{\sigma}$$

### References

### Examples

```
>>> # The input matrix
>>> x = np.array([[1, 11, 104],
                  [1, 15, 99],
                  [1, 22, 89],
                  [1, 27, 88]])
>>> standardize(x)
[[ 0.         -1.25412576  1.33424877]
 [ 0.         -0.60683505  0.59299945]
 [ 0.          0.52592371 -0.88949918]
 [ 0.          1.3350371  -1.03774904]]
```

# Index


## G

gradient_descent() (in module mlearn.algorithms.linreg), 4

## O

ordinary_least_squares() (in module mlearn.algorithms.linreg), 5

## S

standardize() (in module mlearn.algorithms.transform), 7
sum_of_squared_residuals() (in module mlearn.algorithms.linreg), 3