An Introduction to Machine Learning with Python Programming
11 Sep 2023 - 20 Oct 2023

Regularization

Conducted by:
iHUB DivyaSampark, IIT Roorkee
and
Ritvij Bharat Private Limited (RBPL)

Presented by:
Shreyas Shukla

Regularization seeks to solve model issues by:

1. Minimizing model complexity
2. Penalizing the loss function
3. add more bias to reduce model variance
4. optimal penalty hyperparameter

Three main types of Regularization:

- L1 Regularization (LASSO Regression)
- L2 Regularization (Ridge Regression)
- Combining L1 and L2 (Elastic Net)

L1 regularization adds a penalty which is equal to the **absolute value of the magnitude of coefficients.**

- Limits the size of the coefficients.
- Can yield sparse models where some coefficients can become zero.

L1 regularization adds a penalty which is equal to the **absolute value of the magnitude of coefficients.**

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \boxed{\lambda\sum_{j=1}^{p}|\beta_j|}$$

Remember:

- All coefficients are shrunk by the same factor.
- Does not necessarily eliminate coefficients.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$

L2 regularization adds a penalty equal to the **square** of the magnitude of coefficients.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \boxed{\lambda\sum_{j=1}^{p}\beta_j^2}$$

Elastic net combines L1 and L2 with the addition of an alpha parameter deciding the ratio between them:

$$\frac{\sum_{i=1}^{n}(y_i - x_i^J \hat{\beta})^2}{2n} + \lambda \left(\frac{1-\alpha}{2} \sum_{j=1}^{m} \hat{\beta}_j^2 + \alpha \sum_{j=1}^{m} |\hat{\beta}_j| \right)$$

These regularization methods do have a cost:

1. Introduce an additional hyperparameter that needs to be tuned.

2. A multiplier to the penalty to decide the "strength" of the penalty.

Before we dive into coding, let's discuss a few more relevant topics:

- Feature Scaling
- Cross Validation

Feature Scaling

- Some ML models that rely on distance metrics (e.g. KNN) **require** scaling to perform well.

- Benefits:
  1. FS improves the convergence of steepest descent algorithms, which do not possess the property of scale invariance.

  2. If features are on different scales, certain weights may update faster than others since the feature values **xj** play a role in the weight updates.

- Also important in comparing measurements with different units.
-
- Allows direct comparison of model coefficients.

Some rules:

- Must always scale new unseen data before feeding to model.

- Effects direct interpretability of feature coefficients

Easier to compare coefficients to one another, harder to relate back to original unscaled feature.

- Feature scaling benefits:
  - Can lead to great increases in performance.
  - Absolutely necessary for some models.
  - Virtually no “real” downside to scaling features.

- Two main ways to scale features:
  - Standardization: Rescales data to have a mean (**μ**) of 0 and standard deviation (**σ**) of 1
  - Normalization : Rescales all data values to be between 0-1

Standardization:

Rescales data to have a mean (**μ**) of 0 and standard deviation (**σ**) of 1 (unit variance).

$$X_{changed} = \frac{X - \mu}{\sigma}$$

No confusion Please

Standardization also referred to as “Z-score normalization”.

$$X_{changed} = \frac{X - \mu}{\sigma}$$

Normalization: Scales all data values to be between 0 and 1.

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Let’s discuss the fit and transform calls in scaling.

- A .fit() method call simply calculates the necessary statistics (Xmin,Xmax,mean, standard deviation).
- A .transform() call actually scales data and returns the new scaled version of data.
- Previously saw a similar process for polynomial feature conversion.

Very important:

- We only **fit** to training data.
- Calculating statistical information should only come from training data.
- Don’t want to assume prior knowledge of the test set!
- No **Data Leakage** please !!

Using the full data set would cause **data leakage**:

- Calculating statistics from full data leads to some information of the test set leaking into the training process upon transform() conversion.

- Feature scaling process:
    - Perform train test split
    - Fit to training feature data
    - Transform training feature data
    - Transform test feature data

- Do we need to scale the label?
- not necessary nor advised.
- Can negatively impact stochastic gradient descent.
- stochastic gradient descent is advanced topic. Help Yourself !!

- Let’s move to cross-validation!