Regression metrics optimization

Metrics optimization: our plan

1) **Regression**
   - MSE, (R)MSE, R-squared
   - MAE
   - (R)MSPE, MAPE
   - (R)MSLE
2) **Classification:**
   - Accuracy
   - Logloss
   - AUC
   - Cohen's Kappa

RMSE, MSE, R-squared

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

How do you optimize them?

Just fit the right model!

$$\text{RMSE} = \sqrt{\text{MSE}} \qquad R^2 = 1 - \frac{MSE}{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

RMSE, MSE, R-squared

- **Tree-based**

  XGBoost, LightGBM

  sklearn.RandomForestRegressor

- **Linear models**

  A lot of linear models implemented in siclicar, and most of them are designed to optimize MSE. For example, ordinarily squares, reach regression, regression and so on.

  sklearn.<>Regression

  sklearn.SGDRegressor

  Vowpal Wabbit *(quantile loss)*

- **Neural nets**

  PyTorch, Keras, TF, etc.

Synonyms: L2 loss

Read the docs!

MAE

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

How do you optimize it?

Again, just run the right model!

MAE

- **Tree-based**
  - ~~XGBoost~~, LightGBM
  - sklearn.RandomForestRegressor
- **Linear models**
  - ~~sklearn.<>Regression~~
  - ~~sklearn.SGDRegressor~~
  - Vowpal Wabbit *(quantile loss)*
- **Neural nets**
  - PyTorch, Keras, TF, etc.

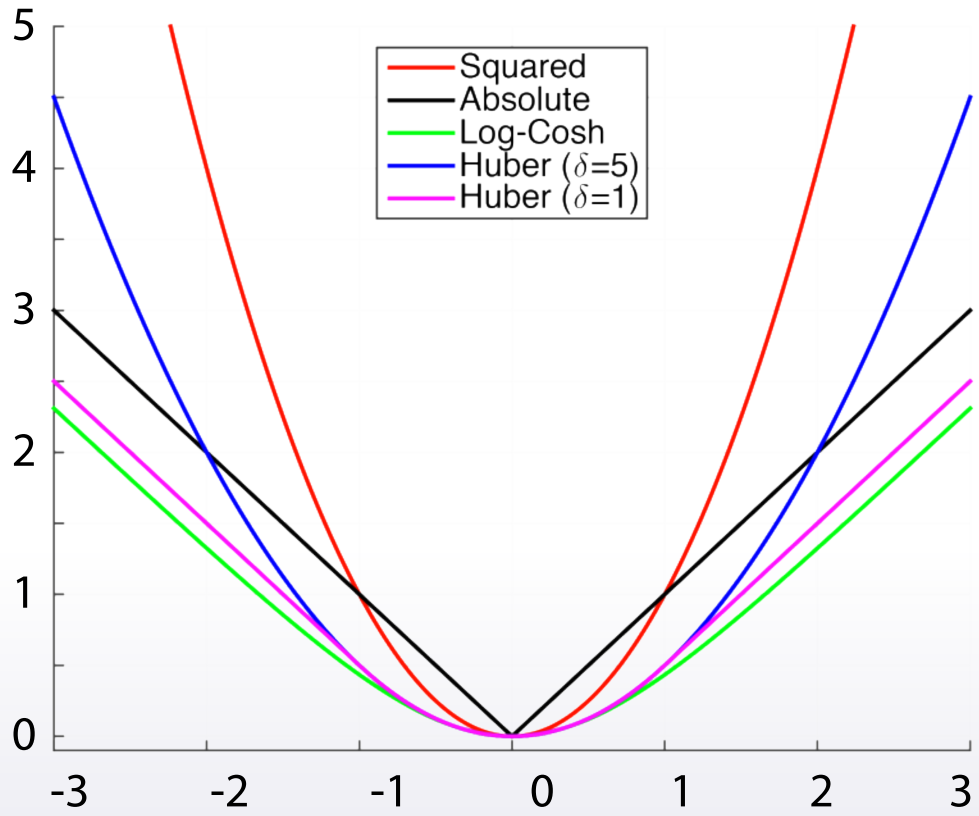
Synonyms: L1, Median regression

Read the docs!

MAE: optimal constant

The most famous one is Huber loss. It's basically a mix between MSE and MAE.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|$$



MSPE and MAPE

$$\text{MSPE} = \frac{100\%}{N} \sum_{i=1}^{N} \left(\frac{y_i - \hat{y}_i}{y_i} \right)^2 \quad \text{MAPE} = \frac{100\%}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

How do you optimize them?

~~Just run the right model!~~

MSPE (MAPE) as weighted MSE (MAE)

But there are several specific approaches that I want to mention.

This approach is based on the fact that MSP is a weighted version of MSE and MAP is a weighted version of MAE. On the right side, we've sen expression for MSP and MAP. The summon denominator just ensures that the weights are summed up to 1, but it's not required.

Sample weights

$$\text{MSPE} = \frac{100\%}{N} \sum_{i=1}^{N} \left(\frac{y_i - \hat{y}_i}{y_i} \right)^2$$

$$w_i = \frac{1/y_i^2}{\sum_{i=1}^{N} 1/y_i^2}$$

$$\text{MAPE} = \frac{100\%}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

$$w_i = \frac{1/y_i}{\sum_{i=1}^{N} 1/y_i}$$

MSPE (MAPE)

- **Use weights for samples (`sample_weights`)**
  - And use MSE (MAE)
  - *Not every library accepts sample weights*
    - `XGBoost`, `LightGBM` accept
    - Neural nets
      - Easy to implement if not supported
- **Resample the train set**
  - `df.sample(weights=sample_weights)`
  - And use *any* model that optimizes MSE (MAE)
  - Usually need to resample many times and average

RMSLE

$$\text{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\log(y_i+1) - \log(\hat{y}_i+1)\right)^2} =$$

$$= \sqrt{MSE\left(\log(y_i+1), \log(\hat{y}_i+1)\right)}$$

Train:

1. Transform target:

$$z_i = \log(y_i+1)$$

2. Fit a model with MSE loss

Test:

- Transform predictions back:

$$\hat{y}_i = \exp(\hat{z}_i) - 1$$

Conclusion

1) **Regression**
   - MSE, (R)MSE, R-squared
   - MAE
   - (R)MSPE, MAPE
   - (R)MSLE
2) **Classification:**
   - Accuracy
   - Logloss
   - AUC
   - Cohen's Kappa