Linear Regression

Machine Learning Practice

Dr. Ashish Tendulkar

IIT Madras

How to build baseline regression model?

`DummyRegressor` helps in creating a baseline for regression.

```python
from sklearn.dummy import DummyRegressor

dummy_regr = DummyRegressor(strategy="mean")
dummy_regr.fit(X_train, y_train)
dummy_regr.predict(X_test)
dummy_regr.score(X_test, y_test)
```

- It makes a prediction as specified by the strategy.
- Strategy is based on some statistical property of the training set or user specified value.

Strategy `mean` `median` `quantile` `constant`

$$\begin{tikzcd} \texttt{quantile} & \texttt{constant} \\ \texttt{quantile} \arrow[u] & \texttt{constant} \arrow[u] \end{tikzcd}$$

How is Linear Regression model trained?

Step 1: Instantiate object of a suitable linear regression estimator from one of the following two options

Normal equation

```
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
```

Iterative optimization

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor()
```

Step 2: Call fit method on linear regression object with training feature matrix and label vector as arguments.

```
# Model training with feature matrix X_train and
# label vector or matrix y_train
linear_regressor.fit(X_train, y_train)
```

Works for both single and multi-output regression.

SGDRegressor Estimator

SGDRegressor Estimator

- Implements stochastic gradient descent
- Use for large training set up (> 10k samples)
- Provides greater control on optimization process through provision for hyperparameter settings.

- `loss= 'squared error'`
- `loss = 'huber'`

- `penalty = 'l1'`
- `penalty = 'l2'`
- `penalty = 'elasticnet'`

SGDRegressor

- `learning_rate = 'constant'`
- `learning_rate = 'optimal'`
- `learning_rate = 'invscaling'`
- `learning_rate = 'adaptive'`

- `early_stopping = 'True'`
- `early_stopping = 'False'`

It's a good idea to use a random seed of your choice while instantiating SGDRegressor object. It helps us get reproducible results.

Set `random_state` to seed of your choice.

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(random_state=42)
```

Note: In the rest of the presentation, we won't set the random seed for sake of brevity. However while coding, always set the random seed in the constructor.

How to perform feature scaling for SGDRegressor?

SGD is sensitive to feature scaling, so it is highly recommended to scale input feature matrix.

```python
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

sgd = Pipeline([
                ('feature_scaling', StandardScaler()),
                ('sgd_regressor', SGDRegressor())])

sgd.fit(X_train, y_train)
```

Note

- Feature scaling is not needed for word frequencies and indicator features as they have intrinsic scale.
- Features extracted using PCA should be scaled by some constant c such that the average L2 norm of the training data equals one.

How to shuffle training data after each epoch in SGDRegressor?

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(shuffle=True)
```

How to use set learning rate in SGDRegreesor?

- `learning_rate = 'constant'`
- `learning_rate = 'invscaling'`
- `learning_rate = 'adaptive'`

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(random_state=42)
```

What is the default setting?

- `learning_rate = 'invscaling'`
- `eta0 = 1e-2`
- `power_t = 0.25`

Learning rate reduces after every iteration:
eta = eta0 / pow(t, power_t)

Note: You can make changes to these parameters to speed up or slow down the training process.

How to use set constant learning rate ?

- `learning_rate = 'constant'`

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(learning_rate='constant',
                                eta0=1e-2)
```

Constant learning rate `eta0 = 1e-2` is used throughout the training.

How to set adaptive learning rate?

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(learning_rate='adaptive',
                                eta0=1e-2)
```

- The learning rate is kept to initial value as long as the training loss decreases.
- When the stopping criterion is reached, the learning rate is divided by 5, and the training loop continues.
- The algorithm stops when the learning rate goes below 10^{-6}.

How to set #epochs in SGDRegreesor?

Set max_iter to desired #epochs. The default value is 1000.

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(max_iter=100)
```

Remember one epoch is one full pass over the training data.

Practical tip

SGD converges after observing approximately 10^6 training samples. Thus, a reasonable first guess for the number of iterations for n sampled training set is

$$\text{max\_iter} = \text{np.ceil}(10^6/n)$$

How to set stopping criteria in SGDRegreesor?

Option #1 `tol`, `n_iter_no_change`, `max_iter.`

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(loss='squared_error',
                                max_iter=500,
                                tol=1e-3,
                                n_iter_no_change=5)
```

The SGDRegreesor stops

- when the training loss does not improve (loss > best_loss - `tol)` for `n_iter_no_change` consecutive epochs
- else after a maximum number of iteration `max_iter.`

How to set stopping criteria in SGDRegreesor?

Option #2 **`early_stopping,`** **`validation_fraction`**

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(loss='squared_error',
                                early_stopping=True
                                max_iter=500,
                                tol=1e-3,
                                validation_fraction=0.2,
                                n_iter_no_change=5)
```

Set aside **`validation_fraction`** percentage records from training set as validation set. Use **`score`** method to obtain validation score.

The SGDRegreesor stops when

- validation score does not improve by at least **`tol`** for **`n_iter_no_change`** consecutive epochs.
- else after a maximum number of iteration **`max_iter.`**

How to use different loss functions in SGDRegreesor?

Set **`loss`** parameter to one of the supported values

`'squared_error'` {studied in this course}

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(loss='squared_error')
```

It also supports other losses as documented in sklearn API

How to use averaged SGD?

Averaged SGD updates the weight vector to average of weights from previous updates.

Option #1: Averaging across all updates `average=True`

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(average=True)
```

Option #2: Set `average` to int value.

Averaging begins once the total number of samples seen reaches `average`

Setting `average=10` starts averaging after seeing 10 samples

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(average=10)
```

Averaged SGD works best with a larger number of features and a higher eta0

How do we initialize SGD with weight vector of the previous run?

Set `warm_start = TRUE`

while instantiating object of SGDRegressor

```python
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(warm_start=True)
```

By default `warm_start = False`

How to monitor SGD loss iteration after iteration?

Make use of `warm_start = TRUE`

```python
sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty=None, learning_rate="constant", eta0=0.0005)

for epoch in range(1000):
    sgd_reg.fit(X_train, y_train)  # continues where it left off
    y_val_predict = sgd_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_predict)
```

Model inspection

How to access the weights of trained Linear Regression model?

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \mathbf{w}^T \mathbf{x}$$

The weights $w_1, w_2, \ldots, w_m$ are stored in coef_ class variable.

```
linear_regressor.coef_
```

The intercept w_0 is stored in intercept_ class variable.

```
linear_regressor.intercept_
```

Note: These code snippets works for both LinearRegression and SGDRegressor, and for that matter to all regression estimators that we will study in this module. Why?

All of them are estimators.

Model inference

How to make predictions on new data in Linear Regression model?

Step 1: Arrange data for prediction in a feature matrix of shape (#samples, #features) or in sparse matrix format.

Step 2: Call predict method on linear regression object with feature matrix as an argument.

```
# Predict labels for feature matrix X_test
linear_regressor.predict(X_test)
```

Same code works for all regression estimators.

Model evaluation

General steps in model evaluation

STEP 1: Split data into train and test

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

STEP 2: Fit linear regression estimator on training set.

STEP 3: Calculate training error (a.k.a. empirical error)

STEP 4: Calculate test error (a.k.a. generalization error)

Compare training and test errors

How to evaluate trained Linear Regression model?

Using score method on linear regression object:

```
# Evaluation on the eval set with
# 1. feature matrix
# 2. label vector or matrix (single/multi-output)
linear_regressor.score(X_test, y_test)
```

The score returns R^2 or coefficient of determination

$$R^2 = (1 - \frac{u}{v})$$

residual sum of squares:

$$u = (\mathbf{Xw} - \mathbf{y})^T(\mathbf{Xw} - \mathbf{y})$$

Sum of squared error (actual and predicted label)

total sum of square

Sum of squared error (actual and mean predicted label)

$$v = (\mathbf{y} - \hat{\mathbf{y}}_{\mathbf{mean}})^T(\mathbf{y} - \hat{\mathbf{y}}_{\mathbf{mean}})$$

The score returns R^2 or coefficient of determination

$$R^2 = \left(1 - \frac{u}{v}\right)$$

When?

- The best possible score is 1.0.

u, sum of squared error = 0

- A constant model that always predicts the expected value of y, would get a score of 0.0.

$u = v$

- The score can be negative (because the model can be arbitrarily worse).

Evaluation metrics

sklearn provides a bunch of regression metrics to evaluate performance of the trained estimator on the evaluation set.

`mean_absolute_error`

```
from sklearn.metrics import mean_absolute_error
eval_score = mean_absolute_error(y_test, y_predicted)
```

`mean_squarred_error`

```
from sklearn.metrics import mean_squarred_error
eval_score = mean_squarred_error(y_test, y_predicted)
```

`r2_score` Same as output of **`score`**

```
from sklearn.metrics import r2_score
eval_score = r2_score(y_test, y_predicted)
```

These metrics can also be used in multi-output regression setup.

`mean_squared_log_error`

```
from sklearn.metrics import mean_squared_log_error
eval_score = mean_squared_log_error(y_test, y_predicted)
```

- Useful for targets with exponential growths like population, sales growth etc,
- Penalizes under-estimation heavier than the over-estimation.

`mean_absolute_percentage_error`

```
from sklearn.metrics import mean_absolute_percentage_error
eval_score = mean_absolute_percentage_error(y_test, y_predicted)
```

- Sensitive to relative error.

`median_absolute_error`

```
from sklearn.metrics import median_absolute_error
eval_score = median_absolute_error(y_test, y_predicted)
```

- Robust to outliers

How to evaluate regression model on worst case error?

Use metrics **`max_error`**

Worst case error on train set can be calculated as follows:

```
from sklearn.metrics import max_error
train_error = max_error(y_train, y_predicted)
```

Worst case error on test set can be calculated as follows:

```
from sklearn.metrics import max_error
test_error = max_error(y_test, y_predicted)
```

This metrics can, however, be used only for single output regression. It does not support multi-output regression.

Scores and Errors

- Score is a metric for which higher value is better.
- Error is a metric for which lower value is better.

Convert error metric to score metric by adding neg_ suffix.

Function	**Scoring**
metrics.mean_absolute_error	neg_mean_absolute_error
metrics.mean_squared_error	neg_mean_squared_error
metrics.mean_squared_error	neg_root_mean_squared_error
metrics.mean_squared_log_error	neg_mean_squared_log_error
metrics.median_absolute_error	neg_median_absolute_error

In case, we get comparable performance on train and test with this split, is this performance guaranteed on other splits too?

- Is test set sufficiently large?
  - In case it is small, the test error obtained may be unstable and would not reflect the true test error on large test set.
- What is the chance that the easiest examples were kept aside as test by chance?
  - This if happens would lead to optimistic estimation of the true test error.

We use cross validation for robust performance evaluation.

Cross-validation performs robust evaluation of model performance

- by repeated splitting and
- providing many training and test errors

This enables us to estimate variability in generalization performance of the model.

sklearn implements the following cross validation iterators

`KFold`

`RepeatedKfold`

`LeaveOneOut`

`ShuffleSplit`

How to obtain cross validated performance measure using KFold?

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import linear_regression

lin_reg = linear_regression()
score = cross_val_score(lin_reg, X, y, cv=5)
```

- Uses KFold cross validation iterator, that divides training data into 5 folds.
- In each run, it uses 4 folds for training and 1 for evaluation.

Alternate way of writing the same thing

```python
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import linear_regression

lin_reg = linear_regression()
kfold_cv = KFold(n_splits=5, random_state=42)
score = cross_val_score(lin_reg, X, y, cv=kfold_cv)
```

How to obtain cross validated performance measure using LeaveOneOut?

```python
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import linear_regression

lin_reg = linear_regression()
loocv = LeaveOneOut()
score = cross_val_score(lin_reg, X, y, cv=loocv)
```

which is same as

```python
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import linear_regression

lin_reg = linear_regression()
n = X.shape[0]
kfold_cv = KFold(n_splits=n)
score = cross_val_score(lin_reg, X, y, cv=kfold_cv)
```

How to obtain cross validated performance measure using ShuffleSplit?

```
from sklearn.linear_model import linear_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

lin_reg = linear_regression()
shuffle_split = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
score = cross_val_score(lin_reg, X, y, cv=shuffle_split)
```

It is also called random permutation based cross validation strategy.

- Generates user defined number of train/test splits.
- It is robust to class distribution.

In each iteration, it shuffles order of data samples and then splits it into train and test.

How to specify a performance measure in **`cross_val_score`**

```python
from sklearn.linear_model import linear_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

lin_reg = linear_regression()
shuffle_split = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
score = cross_val_score(lin_reg, X, y, cv=shuffle_split,
                        scoring='neg_mean_absolute_error')
```

`scoring` parameter can be set to one of the scoring schemes implemented in sklearn as follows

`max_error` **`r2`**

`neg_mean_absolute_error` **`neg_mean_squared_error`**

`neg_mean_squared_log_error` **`neg_median_absolute_error`**

`neg_root_mean_squared_error`

How to obtain test scores from different folds?

```python
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=40, test_size=0.3, random_state=0)
cv_results = cross_validate(
    regressor, data, target, cv=cv, scoring="neg_mean_absolute_error")
```

The results are stored in python dictionary with the following keys:

`fit_time`

`score_time`

`test_score`

`estimator` (optional)

`train_score` (optional)

How to obtain trained estimators and scores on training data during cross validation?

- For trained estimator, set `return_estimator = True`
- For scores on training set, set `return_train_score = True`

```python
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=40, test_size=0.3,
                  random_state=0)
cv_results = cross_validate(
    regressor, data, target,
    cv=cv, scoring="neg_mean_absolute_error",
    return_train_score=True,
    return_estimator=True)
```

The estimators can be accessed through `estimator` key of the dictionary returned by `cross_validate`

How to evaluate multiple metrics of regression in cross validation set up?

```python
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=40, test_size=0.3,
                  random_state=0)
cv_results = cross_validate(
    regressor, data, target,
    cv=cv,
    scoring=["neg_mean_absolute_error", "neg_mean_squared_error"]
    return_train_score=True,
    return_estimator=True)
```

`cross_validate` allows us to specify multiple scoring metrics

unlike `cross_val_score`

How to study effect of #samples on training and test errors?

STEP 1: Instantiate an object of learning_curve class with estimator, training data, size, cross validation strategy and scoring scheme as arguments.

```python
from sklearn.model_selection import learning_curve

results = learning_curve(
    lin_reg, X_train, y_train, train_sizes=train_sizes, cv=cv,
    scoring="neg_mean_absolute_error")
train_size, train_scores, test_scores = results[:3]
# Convert the scores into errors
train_errors, test_errors = -train_scores, -test_scores
```

STEP 2: Plot training and test scores as function of the size of training sets. And make assessment about model fitment: under/overfitting or right fit.

Underfitting/Overfitting diagnosis

STEP 1: Fit linear models with different number of features.

STEP 2: For each model, obtain training and test errors.

STEP 3: Plot #features vs error graph - one each for training and test errors.

STEP 4: Examine the graphs to detect under/overfitting.

We can replace #features with any other tunable hyperparameter to do this diagnosis for setting that hyperparameter to the appropriate value.