graphlab.linear_regression.create

`graphlab.linear_regression.` `create`*(dataset, target, features=None, l2_penalty=0.01, l1_penalty=0.0, solver='auto', feature_rescaling=True, convergence_threshold=0.01, step_size=1.0, lbfgs_memory_level=11, max_iterations=10, validation_set='auto', verbose=True)*

Create a `LinearRegression` to predict a scalar target variable as a linear function of one or more features. In addition to standard numeric and categorical types, features can also be extracted automatically from list- or dictionary-type SFrame columns.

The linear regression module can be used for ridge regression, Lasso, and elastic net regression (see References for more detail on these methods). By default, this model has an l2 regularization weight of 0.01.

Parameters:

dataset : SFrame

The dataset to use for training the model.

target : string

Name of the column containing the target variable.

features : list[string], optional

Names of the columns containing features. 'None' (the default) indicates that all columns except the target variable should be used as features.

The features are columns in the input SFrame that can be of the following types:

- *Numeric*: values of numeric type integer or float.
- *Categorical*: values of type string.
- *Array*: list of numeric (integer or float) values. Each list element is treated as a separate feature in the model.
- *Dictionary*: key-value pairs with numeric (integer or float) values Each key of a dictionary is treated as a separate feature and the value in the dictionary corresponds to the value of the feature. Dictionaries are ideal for representing sparse data.

Columns of type *list* are not supported. Convert such feature columns to type array if all entries in the list are of numeric types. If the lists contain data of mixed types, separate them out into different columns.

l2_penalty : float, optional

Weight on the l2-regularizer of the model. The larger this weight, the more the model coefficients shrink toward 0. This introduces bias into the model but decreases variance, potentially leading to better predictions. The default value is 0.01; setting this parameter to 0 corresponds to unregularized linear regression. See the ridge regression reference for more detail.

l1_penalty : float, optional

Weight on l1 regularization of the model. Like the l2 penalty, the higher the l1 penalty, the more the estimated coefficients shrink toward 0. The l1 penalty, however, completely zeros out sufficiently small coefficients, automatically indicating features that are not useful for the model. The default weight of 0 prevents any features from being discarded. See the LASSO regression reference for more detail.

solver : string, optional

Solver to use for training the model. See the references for more detail on each solver.

- *auto (default)*: automatically chooses the best solver for the data and model parameters.
- *newton*: Newton-Raphson
- *lbfgs*: limited memory BFGS
- *gd*: gradient descent
- *fista*: accelerated gradient descent

The model is trained using a carefully engineered collection of methods that are automatically picked based on the input data. The `newton` method works best for datasets with plenty of examples and few features (long datasets). Limited memory BFGS (`lbfgs`) is a robust solver for wide datasets (i.e datasets with many coefficients). `fista` is the default solver for l1-regularized linear regression. Gradient-descent (GD) is another well tuned method that can work really well on l1-regularized problems. The solvers are all automatically tuned and the default options should function well. See the solver options guide for setting additional parameters for each of the solvers.

feature_rescaling : boolean, optional

Feature rescaling is an important pre-processing step that ensures that all features are on the same scale. An l2-norm rescaling is performed to make sure that all features are of the same norm. Categorical features are also rescaled by rescaling the dummy variables that are used to represent them. The coefficients are returned in original scale of the problem. This process is particularly useful when features vary widely in their ranges.

validation_set : SFrame, optional

A dataset for monitoring the model's generalization performance. For each row of the progress table, the chosen metrics are computed for both the provided training dataset and the validation_set. The format of this SFrame must be the same as the training set. By default this argument is set to 'auto' and a validation set is automatically sampled and used for progress printing. If validation_set is set to None, then no additional metrics are computed. The default value is 'auto'.

convergence_threshold : float, optional

Convergence is tested using variation in the training objective. The variation in the training objective is calculated using the difference between the objective values between two steps. Consider reducing this below the default value (0.01) for a more accurately trained model. Beware of overfitting (i.e a model that works well only on the training data) if this parameter is set to a very low value.

lbfgs_memory_level : int, optional

The L-BFGS algorithm keeps track of gradient information from the previous `lbfgs_memory_level` iterations. The storage requirement for each of these gradients is the `num_coefficients` in the problem. Increasing the `lbfgs_memory_level` can help improve the quality of the model trained. Setting this to more than `max_iterations` has the same effect as setting it to `max_iterations`.

max_iterations : int, optional

The maximum number of allowed passes through the data. More passes over the data can result in a more accurately trained model. Consider increasing this (the default value is 10) if the training accuracy is low and the *Grad-Norm* in the display is large.

step_size : float, optional (fista only)

The starting step size to use for the `fista` and `gd` solvers. The default is set to 1.0, this is an aggressive setting. If the first iteration takes a considerable amount of time, reducing this parameter may speed up model training.

verbose : bool, optional

If True, print progress updates.

Returns: **out** : LinearRegression

A trained model of type `LinearRegression`.

See also

`LinearRegression`, `graphlab.boosted_trees_regression.BoostedTreesRegression`, `graphlab.regression.create`

Notes

- Categorical variables are encoded by creating dummy variables. For a variable with K categories, the encoding creates $K-1$ dummy variables, while the first category encountered in the data is used as the baseline.
- For prediction and evaluation of linear regression models with sparse dictionary inputs, new keys/columns that were not seen during training are silently ignored.
- Any 'None' values in the data will result in an error being thrown.
- A constant term is automatically added for the model intercept. This term is not regularized.

References

- Hoerl, A.E. and Kennard, R.W. (1970) Ridge regression: Biased Estimation for Nonorthogonal Problems. Technometrics 12(1) pp.55-67
- Tibshirani, R. (1996) Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society. Series B (Methodological) 58(1) pp.267-288.
- Zhu, C., et al. (1997) Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software 23(4) pp.550-560.
- Barzilai, J. and Borwein, J. Two-Point Step Size Gradient Methods. IMA Journal of Numerical Analysis 8(1) pp.141-148.
- Beck, A. and Teboulle, M. (2009) A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. SIAM Journal on Imaging Sciences 2(1) pp.183-202.
- Zhang, T. (2004) Solving large scale linear prediction problems using stochastic gradient descent algorithms. ICML '04: Proceedings of the twenty-first international conference on Machine learning p.116.

Examples

Given an `SFrame` `sf` with a list of columns [`feature_1` ... `feature_K`] denoting features and a target column `target`, we can create a `LinearRegression` as follows:

```
>>> data =  graphlab.SFrame('http://s3.amazonaws.com/dato-
datasets/regression/houses.csv')
```

```
>>> model = graphlab.linear_regression.create(data, target='price',
...                                  features=['bath', 'bedroom', 'size'])
```

For ridge regression, we can set the `l2_penalty` parameter higher (the default is 0.01). For Lasso regression, we set the l1_penalty higher, and for elastic net, we set both to be higher.

```
# Ridge regression
>>> model_ridge = graphlab.linear_regression.create(data, 'price', l2_penalty=0.1)

# Lasso
>>> model_lasso = graphlab.linear_regression.create(data, 'price', l2_penalty=0.,
                                                              l1_penalty=1.0)

# Elastic net regression
>>> model_enet  = graphlab.linear_regression.create(data, 'price', l2_penalty=0.5,
                                                            l1_penalty=0.5)
```