Feature Selection with scikit-learn and Hivemall for House Price Prediction

Treasure Data Workflow provides an easy way to predict continuous values, like a price or energy consumption, using Linear Regression predictor. Machine Learning algorithms can be run as part of your scheduled workflows, using Python custom scripts. This article introduces feature selection using scikit-learn in a Python script, which selects important features and builds a partial query for Hivemall to predict house prices.

Feature selection is a common machine learning technique used to build a simplified model for understanding and to enhance generalization by removing irrelevant or redundant information.

Feature Selection Using Python Custom Scripts

This article describes how to predict house prices using the Boston house pricing data set with Linear Regression predictor. The feature selection from scikit-learn helps identify meaningful attributes from which to create supervised models.

Example Workflow to Predict House Prices

The workflow:

- Splits the training and test data sets
- Selects important features with scikit-learn
- Builds a partial query for Hivemall
- Trains and evaluates the model with selected features on Hivemall

Prerequisites

- Make sure that the custom scripts feature is enabled for your TD account.
- Download and install the TD Toolbelt and the TD Toolbelt Workflow module. For more information, see TD Workflow Quickstart.
- Basic Knowledge of Treasure Workflow's syntax

Run the Example Workflow

1. Download the house price prediction project.
2. From the command line terminal window, change to the to house-price-prediction directory. For example:
3. Cd house-price-prediction
4. Run *data.sh* to ingest training and test data on Treasure Data. The script uses the Boston Housing Dataset with about 506 cases to build the model. The script also creates a database named *boston* and table named *house_prices* to store the data.

```
$ ./data.sh
```

Input

Assume the input table is:

crim double	zn double	indus double	chas int	nox double	rm double	age double	dis double	rad int	tax int	ptratio double	b double	lstat double	medv double
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Where "**medv",** is Median value of owner-occupied homes in $1000's, the target value for regression. "**chas"** and "**rad"** is categorical values, and other features are quantitative.

By default, the top 4 correlated columns with **medv** are used.

- Run the example workflow as follows:

```
$ td workflow push regressor
```

Set credentials for this workflow

export TD_API_KEY=1/xxxxx

export TD_API_SERVER=https://api.treasuredata.com

```
$ td wf secrets --project regressor --set apikey --set endpoint
```

Enter apikey e.g.) X/XXXXXXX, and endpoint e.g.) https://api.treasuredata.com

```
$ td wf start regressor regression-py
```

The predicted results for the price of houses are stored in the *predictions* table.

To view the table:

1. Log into TD Console.
2. Search for the boston database.
3. Locate the *predictions* table

Output

This workflow outputs predicted results

rowid **string**	**predicted_price** **double**
1-10	33.97034232809395
1-121	30.3377696027913
...	...

Review the Workflow Custom Python Script

Review the contents of the directory:

- regression-py.dig - Example workflow for sales prediction and notification to Slack.
- task/__init__.py - Custom Python script with scikit-learn. It selects important features and builds a partial query for Hivemall.

This example uses scikit-learn's SelectFromModel function, which enables selection of features when building a predictive model.

```
$ ./data.sh # Ingest example data to Treasure Datareg = ExtraTreesRegressor()reg = reg.fit(X, y)model =
SelectFromModel(reg, prefit=True)feature_idx = model.get_support()feature_name = df.drop(columns=['medv']).
columns[feature_idx]selected_features = set(feature_name)...(snip)...feature_query = self.
_feature_column_query(selected_features, feature_types=feature_types)
```

In this example we use ExtraTreeRegressor to get feature importance, you can use any other logics such asLassoCV.

In the example code, there is a _create_vectorize_table function for creating a vectorized table to train Hivemall models with Python.

```
self._create_vectorize_table(engine_hive, dbname, "train", "{}_train".format(source_table), feature_query)
```

While you can use this function within the custom Python script instead of exporting the partial query, we recommend that you export queries. Exporting queries gives you the benefit of Digdag parallelization and manageability on TD Console.