How can I use scaling and log transforming together?

Asked 4 years, 7 months ago Modified 11 months ago Viewed 22k times

8

I'm creating a regular linear regression model to establish a baseline before moving on to more advanced techniques. I scaled my data as below:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std=pd.DataFrame(sc.fit_transform(X_train), columns=data.columns)
X_test_std=pd.DataFrame(sc.transform(X_test), columns=data.columns)
```

However, the variables mostly have an extreme skew (right tail), but I can't figure out how to apply a log transform on them.

Would I apply the log transform to variables in both the `X_train` and `X_test` datasets? Do I need to do this before applying the scaling? I just can't think through the right way to go about this in terms of applying predictions to the `X_test` set. Any ideas?

python data-transformation scales

Share Cite Improve this question

Follow

edited Dec 29, 2022 at 12:25

Saurav Maheshkar

245 1 8

asked Apr 11, 2019 at 14:47

slim88

123 1 2 6

4 Scaling and then applying the log would result in errors since any values below the sample mean result in negative values post transform. Log, then scale. – Demetri Pananos Feb 11, 2020 at 18:12

3 Answers

Sorted by: Highest score (default)

10

You can form a pipeline and apply standard scaling and log transformation subsequently. In this way, you can just train your pipelined regressor on the train data and then use it on the test data. For every input, the pipelined regressor will standardize and log transform the input before making the prediction.

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import FunctionTransformer
from imblearn.pipeline import Pipeline

def log_transform(x):
    print(x)
    return np.log(x + 1)


scaler = StandardScaler()
transformer = FunctionTransformer(log_transform)
pipe = Pipeline(steps=[('scaler', scaler), ('transformer', transformer),
('regressor', your_regressor)], memory='sklearn_tmp_memory')

pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Share Cite Improve this answer Follow

answered Jan 24, 2021 at 14:01

Panagiotis Koromilas
101 1 3

7 You can't do log on negative data.. – Ferus Feb 6, 2021 at 11:22

2

To apply the log transform you would use numpy. Numpy as a dependency of scikit-learn and pandas so it will already be installed.

```
import numpy as np

X_train = np.log(X_train)
X_test = np.log(X_test)
```

You may also be interested in applying that transformation earlier in your pipeline before splitting data into training and test sets.

```
# Assumes X and y have already been defined

import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

X = np.log(X)

X_train, X_test, y_train, y_test = train_test_split(X, y)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```

Share Cite Improve this answer Follow

answered Apr 11, 2019 at 20:44

Oliver Rice
121 2

---

0

I had the same issue, with the additional inconvenience of only wanting to apply the transforms to a subset of my features.

My solution is essentially the same as Panagiotis Koromilas's, with these key changes:

1. You can specify a subset of columns to transform
2. The log is applied before StandardScaler(). StandardScaler() typically results in ~half your values being below 0, and it's not possible to take the log of a negative value.
3. The inbuilt numpy function np.log1p is used. This allows you to easily pickle the model & pipeline with `joblib.dump()` and use it elsewhere without needing to make your custom `log_transform()` function available to the code importing the pickled model.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

# these columns will have the scaling and transforms applied to them
COLS_TO_TRANSFORM = ['col1', 'col2']

# set up the log transformer
log_transform = ColumnTransformer(
    transformers=[
        ('log', FunctionTransformer(np.log1p), COLS_TO_TRANSFORM),
    ],
    verbose_feature_names_out=False, # if True, "log_" will be prefixed to the
column names that have been transformed
    remainder='passthrough' # this allows columns not being transformed to pass
through unchanged
)

this ensures that the transform outputs a DataFrame, so that the column names
are available for the next step.
log_transform.set_output(transform='pandas')

# set up whatever other scaling you want
scale = ColumnTransformer(
    transformers=[
        ('scale', StandardScaler(), COLS_TO_TRANSFORM),
    ],
    verbose_feature_names_out=False,
    remainder='passthrough'
)

scale.set_output(transform='pandas')

# put it all together
model = Pipeline(steps=[
```
```

```
    ("log", log_transform),
    ("scale", scale),
    ("regressor", LogisticRegression())])
```

PS:

`set_output()` is a new addition in scikit-learn 1.2.0. Before this it was quite awkward to preserve column names when using ColumnTransformer. More detail

Share Cite Improve this answer Follow

edited Dec 29, 2022 at 10:02

answered Dec 28, 2022 at 15:11

Femke

1 1