How to Use the ColumnTransformer for Data Preparation

by **Jason Brownlee** on December 31, 2020 in **Data Preparation** 67

You must prepare your raw data using data transforms prior to fitting a machine learning model.

This is required to ensure that you best expose the structure of your predictive modeling problem to the learning algorithms.

Applying data transforms like scaling or encoding categorical variables is straightforward when all input variables are the same type. It can be challenging when you have a dataset with mixed types and you want to selectively apply data transforms to some, but not all, input features.

Thankfully, the scikit-learn Python machine learning library provides the **ColumnTransformer** that allows you to selectively apply data transforms to different columns in your dataset.

In this tutorial, you will discover how to use the ColumnTransformer to selectively apply data transforms to columns in a dataset with mixed data types.

After completing this tutorial, you will know:

- The challenge of using data transformations with datasets that have mixed data types.
- How to define, fit, and use the ColumnTransformer to selectively apply data transforms to columns.
- How to work through a real dataset with mixed data types and use the ColumnTransformer to apply different transforms to categorical and numerical data columns.

Kick-start your project with my new book Data Preparation for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Update Dec/2020**: Fixed small typo in API example.

Use the ColumnTransformer for Numerical and Categorical Data in Python
Photo by Kari, some rights reserved.

Tutorial Overview

This tutorial is divided into three parts; they are:

1. Challenge of Transforming Different Data Types
2. How to use the ColumnTransformer
3. Data Preparation for the Abalone Regression Dataset

Challenge of Transforming Different Data Types

It is important to prepare data prior to modeling.

This may involve replacing missing values, scaling numerical values, and one hot encoding categorical data.

Data transforms can be performed using the scikit-learn library; for example, the SimpleImputer class can be used to replace missing values, the MinMaxScaler class can be used to scale numerical values, and the OneHotEncoder can be used to encode categorical variables.

For example:

```
...
# prepare transform
scaler = MinMaxScaler()
# fit transform on training data
scaler.fit(train_X)
# transform training data
train_X = scaler.transform(train_X)
```

Sequences of different transforms can also be chained together using the Pipeline, such as imputing missing values, then scaling numerical values.

For example:

```
...
# define pipeline
pipeline = Pipeline(steps=[('i', SimpleImputer(strategy='median')), ('s', MinMaxScaler())])
# transform training data
train_X = pipeline.fit_transform(train_X)
```

It is very common to want to perform different data preparation techniques on different columns in your input data.

For example, you may want to impute missing numerical values with a median value, then scale the values and impute missing categorical values using the most frequent value and one hot encode the categories.

Traditionally, this would require you to separate the numerical and categorical data and then manually apply the transforms on those groups of features before combining the columns back together in order to fit and evaluate a model.

Now, you can use the ColumnTransformer to perform this operation for you.

How to use the ColumnTransformer

The ColumnTransformer is a class in the scikit-learn Python machine learning library that allows you to selectively apply data preparation transforms.

For example, it allows you to apply a specific transform or sequence of transforms to just the numerical columns, and a separate sequence of transforms to just the categorical columns.

To use the ColumnTransformer, you must specify a list of transformers.

Each transformer is a three-element tuple that defines the name of the transformer, the transform to apply, and the column indices to apply it to. For example:

- (Name, Object, Columns)

For example, the ColumnTransformer below applies a OneHotEncoder to columns 0 and 1.

```
...
transformer = ColumnTransformer(transformers=[('cat', OneHotEncoder(), [0, 1])])
```

The example below applies a SimpleImputer with median imputing for numerical columns 0 and 1, and SimpleImputer with most frequent imputing to categorical columns 2 and 3.

```
...
t = [('num', SimpleImputer(strategy='median'), [0, 1]), ('cat', SimpleImputer(strategy='most_fre
transformer = ColumnTransformer(transformers=t)
```

Any columns not specified in the list of "*transformers*" are dropped from the dataset by default; this can be changed by setting the "*remainder*" argument.

Setting *remainder='passthrough'* will mean that all columns not specified in the list of "*transformers*" will be passed through without transformation, instead of being dropped.

For example, if columns 0 and 1 were numerical and columns 2 and 3 were categorical and we wanted to just transform the categorical data and pass through the numerical columns unchanged, we could define the ColumnTransformer as follows:

```
sformer = ColumnTransformer(transformers=[('cat', OneHotEncoder(), [2, 3])], remainder='passthrough
```

Once the transformer is defined, it can be used to transform a dataset.

For example:

```
...
transformer = ColumnTransformer(transformers=[('cat', OneHotEncoder(), [0, 1])])
# transform training data
train_X = transformer.fit_transform(train_X)
```

A ColumnTransformer can also be used in a Pipeline to selectively prepare the columns of your dataset before fitting a model on the transformed data.

This is the most likely use case as it ensures that the transforms are performed automatically on the raw data when fitting the model and when making predictions, such as when evaluating the model on a test dataset via cross-validation or making predictions on new data in the future.

For example:

```
...
# define model
model = LogisticRegression()
# define transform
transformer = ColumnTransformer(transformers=[('cat', OneHotEncoder(), [0, 1])])
# define pipeline
pipeline = Pipeline(steps=[('t', transformer), ('m',model)])
# fit the pipeline on the transformed data
pipeline.fit(train_X, train_y)
# make predictions
yhat = pipeline.predict(test_X)
```

Now that we are familiar with how to configure and use the ColumnTransformer in general, let's look at a worked example.

Data Preparation for the Abalone Regression Dataset

The abalone dataset is a standard machine learning problem that involves predicting the age of an abalone given measurements of an abalone.

You can download the dataset and learn more about it here:

- Download Abalone Dataset (abalone.csv)
- Learn More About the Abalone Dataset (abalone.names)

The dataset has 4,177 examples, 8 input variables, and the target variable is an integer.

A naive model can achieve a mean absolute error (MAE) of about 2.363 (std 0.092) by predicting the mean value, evaluated via 10-fold cross-validation.

We can model this as a regression predictive modeling problem with a support vector machine model (SVR).

Reviewing the data, you can see the first few rows as follows:

```
M,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15
M,0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,7
F,0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,9
M,0.44,0.365,0.125,0.516,0.2155,0.114,0.155,10
I,0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,7
...
```

We can see that the first column is categorical and the remainder of the columns are numerical.

We may want to one hot encode the first column and normalize the remaining numerical columns, and this can be achieved using the ColumnTransformer.

First, we need to load the dataset. We can load the dataset directly from the URL using the read_csv() Pandas function, then split the data into two data frames: one for input and one for the output.

The complete example of loading the dataset is listed below.

```
# load the dataset
from pandas import read_csv
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
# split into inputs and outputs
last_ix = len(dataframe.columns) - 1
X, y = dataframe.drop(last_ix, axis=1), dataframe[last_ix]
print(X.shape, y.shape)
```

Note: if you have trouble loading the dataset from a URL, you can download the CSV file with the name '*abalone.csv*' and place it in the same directory as your Python file and change the call to *read_csv()* as follows:

```
...
dataframe = read_csv('abalone.csv', header=None)
```

Running the example, we can see that the dataset is loaded correctly and split into eight input columns and one target column.

```
(4177, 8) (4177,)
```

Next, we can use the *select_dtypes()* function to select the column indexes that match different data types.

We are interested in a list of columns that are numerical columns marked as '*float64*' or '*int64*' in Pandas, and a list of categorical columns, marked as '*object*' or '*bool*' type in Pandas.

```
...
# determine categorical and numerical features
numerical_ix = X.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = X.select_dtypes(include=['object', 'bool']).columns
```

We can then use these lists in the ColumnTransformer to one hot encode the categorical variables, which should just be the first column.

We can also use the list of numerical columns to normalize the remaining data.

```
...
# define the data preparation for the columns
t = [('cat', OneHotEncoder(), categorical_ix), ('num', MinMaxScaler(), numerical_ix)]
col_transform = ColumnTransformer(transformers=t)
```

Next, we can define our SVR model and define a Pipeline that first uses the ColumnTransformer, then fits the model on the prepared dataset.

```
...
# define the model
model = SVR(kernel='rbf',gamma='scale',C=100)
# define the data preparation and modeling pipeline
pipeline = Pipeline(steps=[('prep',col_transform), ('m', model)])
```

Finally, we can evaluate the model using 10-fold cross-validation and calculate the mean absolute error, averaged across all 10 evaluations of the pipeline.

```
...
# define the model cross-validation configuration
cv = KFold(n_splits=10, shuffle=True, random_state=1)
# evaluate the pipeline using cross validation and calculate MAE
scores = cross_val_score(pipeline, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# convert MAE scores to positive values
scores = absolute(scores)
# summarize the model performance
print('MAE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Tying this all together, the complete example is listed below.

```
# example of using the ColumnTransformer for the Abalone dataset
from numpy import mean
from numpy import std
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR

# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
# split into inputs and outputs
last_ix = len(dataframe.columns) - 1
X, y = dataframe.drop(last_ix, axis=1), dataframe[last_ix]
print(X.shape, y.shape)
# determine categorical and numerical features
numerical_ix = X.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = X.select_dtypes(include=['object', 'bool']).columns
# define the data preparation for the columns
t = [('cat', OneHotEncoder(), categorical_ix), ('num', MinMaxScaler(), numerical_ix)]
col_transform = ColumnTransformer(transformers=t)
# define the model
model = SVR(kernel='rbf',gamma='scale',C=100)
# define the data preparation and modeling pipeline
pipeline = Pipeline(steps=[('prep',col_transform), ('m', model)])
# define the model cross-validation configuration
cv = KFold(n_splits=10, shuffle=True, random_state=1)
# evaluate the pipeline using cross validation and calculate MAE
scores = cross_val_score(pipeline, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# convert MAE scores to positive values
scores = absolute(scores)
# summarize the model performance
print('MAE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Running the example evaluates the data preparation pipeline using 10-fold cross-validation.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we achieve an average MAE of about 1.4, which is better than the baseline score of 2.3.

```
(4177, 8) (4177,)
MAE: 1.465 (0.047)
```

You now have a template for using the ColumnTransformer on a dataset with mixed data types that you can use and adapt for your own projects in the future.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

API

- sklearn.compose.ColumnTransformer API.
- pandas.read_csv API.
- sklearn.impute.SimpleImputer API.
- sklearn.preprocessing.OneHotEncoder API.
- sklearn.preprocessing.MinMaxScaler API
- sklearn.pipeline.Pipeline API.

Summary

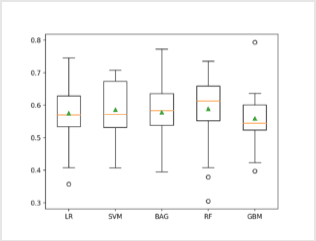
In this tutorial, you discovered how to use the ColumnTransformer to selectively apply data transforms to columns in datasets with mixed data types.

Specifically, you learned:

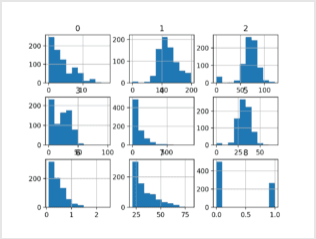
- The challenge of using data transformations with datasets that have mixed data types.
- How to define, fit, and use the ColumnTransformer to selectively apply data transforms to columns.
- How to work through a real dataset with mixed data types and use the ColumnTransformer to apply different transforms to categorical and numerical data columns.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

More On This Topic

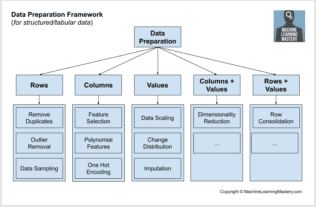


Develop a Model for the Imbalanced Classification of…



How to Selectively Scale Numerical Input Variables…

Imbalanced Classification with the Adult Income Dataset



Framework for Data Preparation Techniques in Machine…

Best Results for Standard Machine Learning Datasets

How to Avoid Data Leakage When Performing Data Preparation

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ TensorFlow 2 Tutorial: Get Started in Deep Learning with tf.keras

A Gentle Introduction to Imbalanced Classification ›

67 Responses to *How to Use the ColumnTransformer for Data Preparation*

Venkatesh Gandi January 20, 2020 at 7:06 am #

REPLY

Wow this function with the pipeline seems to be magical. I really wanted to know what pipelines can do(the power of pipelines). I have seen the documentation of pipeline(), it's not as simple as you explain 🙂 I really likes your way of explanation. Have you written any blogs on introduction to pipelines which can start with simple and explain complicated pipelines. if so, please share the links.

if not, Can you please share some references where we can learn more about the sklearn pipeline?

Jason Brownlee January 20, 2020 at 8:47 am #

REPLY

Thanks.

Yes, maybe here:
https://machinelearningmastery.com/automate-machine-learning-workflows-pipelines-python-scikit-learn/

rahul malik April 22, 2020 at 8:18 am #

REPLY

I have followed your example , I am having a dataset with 3 columns as feature and 1 as label and all are categorical, no numeric values. I have printed X,y and seems they are correct but out is coming as MAE: nan (nan). Can you please suggest what wrong I am doing.

Jason Brownlee April 22, 2020 at 10:12 am #

REPLY

Perhaps check if you have a nan in your input data?

Grzegorz Kępisty April 30, 2020 at 4:47 pm #

REPLY

Very useful utility from sklearn!

I was wondering how to get the full list of transformations that can be applied to ColumnTransformer and the best reference I found is below:
https://scikit-learn.org/stable/modules/preprocessing.html#
Do you know maybe some broader source for this topic?

Regards!

Jason Brownlee May 1, 2020 at 6:32 am #

REPLY

That is a great start.

No, but I have a ton of tutorials on this theme written and scheduled. Get ready!

ashar138 May 19, 2020 at 2:57 pm #

REPLY

Love this. Makes you realize how ColumnTransformer can make your life radically easy.
But what I didn't get is are the transformations applied in series ?

Cause I want to use an Imputer first and then normalize/onehot it.

```
transformers =[ (imputer cat) , (imputer num) , (normalize num), (onehot cat)]
```

I assume this can be done ?

Jason Brownlee May 20, 2020 at 6:19 am #

REPLY

It sure does!

Yes, you can apply different sequences to different subsets of features.

ashar138 May 20, 2020 at 5:58 pm #

REPLY

Turns out you can\t :\
it throws error when dealing with NaN values, even though the imputer is the first transformation.
Need to use Imputer first and then Minmax AGAIN

Jason Brownlee May 21, 2020 at 6:12 am #

REPLY

I give an example of imputing missing values and transforming sequentially using the columntransformer here:
https://machinelearningmastery.com/results-for-standard-classification-and-regression-machine-learning-datasets/

Specifically the Auto Imports example.

ashar138 May 26, 2020 at 1:12 pm #

Okay I am a little confused now..

So what we do is we create a Pipeline, feed it to a ColumnTransformer and we feed that to yet another Pipeline.

But then why do we even need Pipeline ? For example,

```
trans1 = ColumnTransformer([('catimp',
SimpleImputer(strategy='most_frequent'), cat_var),
('enc', OneHotEncoder(handle_unknown='ignore'), cat_var),
('imp', SimpleImputer(strategy= 'median'),num_var )],
remainder='passthrough')
```

steps = [('c', Pipeline(steps=[('s',SimpleImputer(strategy='most_frequent')),
('oe',OneHotEncoder(handle_unknown='ignore'))]), cat_var),
('n', SimpleImputer(strategy='median'), num_var)]

```
trans2 = ColumnTransformer(transformers=steps,
remainder='passthrough')
```

Why are trans1 and trans2 different ?

Jason Brownlee May 26, 2020 at 1:24 pm #

Good question. You don't have to use it, it's just another tool we have available.

Each group of features can be prepared with a pipeline.

We can also have a master pipeline with data prep and modeling – then use cross-validation to evaluate it.

ashar138 May 26, 2020 at 1:45 pm #

I see.. Think I need more time with it..

Actually the example I shared below produces 2 different results. The one without the pipeline gives a NaN found error. Perhaps the Pipeline is needed to execute in a sequence while ColumnTransformer doesn't do that ?

Another issue I observed was while doing transformations of train and valid datasets. The resultant train dataset returned a `scipy.sparse.csr.csr_matrix` while the valid data just returned an ndarray.

I reduced the `sparse_threshold` but that results in a feature mismatch while predicting on the validation dataset.

Anyways I have bothered you enough already, I will figure it out somehow 🙂

Jason Brownlee May 27, 2020 at 7:41 am #

Transforms like one hot encoding will create a sparse matrix by default, you can set an argument to force them to create a dense matrix instead.

Manideep May 27, 2020 at 4:45 am #

REPLY

if i have to use simple imputer and onehotencoder both for a set of categorical columns.could u please tell me what should i do?

Jason Brownlee May 27, 2020 at 8:02 am #

REPLY

Impute then encode in a pipeline.

MS July 1, 2020 at 1:47 am #

REPLY

numerical_x = reduced_df.select_dtypes(include=['int64', 'float64']).columns
categorical_x = reduced_df.select_dtypes(include=['object', 'bool']).columns

t = [(('le',LabelEncoder(),categorical_x),('ohe',OneHotEncoder(),categorical_x),
('catimp', SimpleImputer(strategy='most_frequent'),categorical_x)),
(('num',SimpleImputer(strategy='median'),numerical_x),('sts',StandardScaler(), numerical_x))]

col_transform = ColumnTransformer(transformers=t)
dt= DecisionTreeClassifier()
pl= Pipeline(steps=[('prep',col_transform), ('dt', dt)])
pl.fit(reduced_df,y_train)
pl.score(reduced_df,y_train)

the above code gives this error=>

names, transformers, _ = zip(*self.transformers)
ValueError: not enough values to unpack (expected 3, got 2)

can u please help me

REPLY

Jason Brownlee July 1, 2020 at 5:54 am #

I'm sorry to hear that, this may help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

REPLY

Volkan Yurtseven July 1, 2020 at 5:50 am #

In the example just after the paragraph "This is the most likely use case as it ensures that the transforms are performed automatically on the raw data when fitting the model and when making predictions, such as when evaluating the model on a test dataset via cross-validation or making predictions on new data in the future.." you say
model.fit(train_X, train_y),
but i think it should be:
pipeline.fit(train_X, train_y)

REPLY

Jason Brownlee July 1, 2020 at 5:59 am #

Agreed. Fixed, thank you!

REPLY

Navish July 11, 2020 at 12:04 am #

Hi,
Thanks for the excellent article.
Had three clarifications:

1. So if I ever have to do more than just transform values in preprocessing (like impute & then transform), I need to put each individual combination in a Pipeline? These cannot directly go into the column transformer?

2. I assume we can do multiple splits of the dataframe too? Not just 2? Like if we want to LabelEncode some cat columns, 'most_frequent' impute & OneHotEncode other cat columns, impute 'median' for a few num columns and impute 'mean' for others followed by scaling them?

3. Lastly, if you want to drop certain columns – is it recommended to do so at first outside of any pipeline/transformer?

Jason Brownlee July 11, 2020 at 6:17 am #

REPLY

Correct, a pipeline.

Yes, any subsets you like.

Hmmm, good question. Probably – off the cuff that makes sense.

Navish July 12, 2020 at 4:06 am #

REPLY

Thanks for the reply.

I was reading the entire syntax of Pipeline & ColumnTransformer.
There is an explicit drop option in there too, where you can specify specific columns to be dropped. Might actually be better to use that, just to make it easier for validation/test/production data preprocessing.
This way you can use the 'pass_through' option while dropping columns you want.

Jason Brownlee July 12, 2020 at 6:00 am #

REPLY

Agreed!

Nalla July 27, 2020 at 2:13 am #

REPLY

Hi Jason,

I have some clarifications.

1. If I am using SMOTE and if i prefer using imblearn pipeline.
Is there a way in imblearn pipeline to do a imputer or should we rely on sklearn pipeline

2. How to integrate imblearn pipeline and sklearn or is it advisable to do so?

Thanks,
Nalla

Jason Brownlee July 27, 2020 at 5:50 am #

REPLY

Yes, you can use any data prep you like in the imbalanced learn pipeline.

Use the imbalanced learn pipeline just like you would an sklearn pipeline.

Nalla July 29, 2020 at 1:56 am #

REPLY

Thanks much for the clarification but I would also like to know something in detail

What is the difference or similarity between the sklearn and imblearn pipeline then?

As far as I know – SMOTE can be performed only in imblearn pipeline it cannot be done in sklearn
But how do other functionality work?
Could you give an example to illustrate both of them (similarly and dissimilarity)- if you have any.

And also the possibility of integrating the above two pipelines (i.e)
imblearn and sklearn?
Is the above possible? Like say if I notice an imbalance in dataset so I have decided to go with imblearn pipeline – any means of adding up sklearn pipeline for Imputer or scaling after doing SMOTE in imblearn.

Jason Brownlee July 29, 2020 at 5:54 am #

REPLY

The imblearn pipeline allows you to use data sampling methods as well – e.g. methods that change the number of rows.

That is the only notable difference.

Binyamin August 23, 2020 at 11:39 pm #

REPLY

Hi

I was looking to apply a scaler and one hot encoding in place so that I don't lose the index of my data frame . I came across your blog demonstrating the column transformer and, this was exactly what i needed. however there seems to be one drawback to this method, you lose the names of the features. Column transformer returns an sparse matrix scaled and encoded.

Is there a way to retrieve to column names and recreate a data frame? This would help when it comes to model interpretation

Thanks for your time, your blogs are amazing

Jason Brownlee August 24, 2020 at 6:25 am #

REPLY

Thanks!

Hold the names of the features in a separate array. No need to have that information in your model.

Diego September 13, 2020 at 2:34 am #

REPLY

Dear Jason,

Let's say we have a dataset with 5 features.
We apply column_transformer only to 2nd a 3rd columns, and passthrough the rest of them.
After applying the pipeline, the new order of columns will be: 2,3,1,4,5 (first the ones that have been transformed and then the rest of them)
At the end, the dataset ends up having a different column order, why ? Is there any way to avoid this ?

Thanks a lot

Jason Brownlee September 13, 2020 at 6:09 am #

REPLY

Does it matter? You only need the prediction output from the model. The row order is unchanged.

Diego September 14, 2020 at 12:42 am #

REPLY

Good point actually 🙂

I think it only matters if one of the steps is RFE or RFECV and we want to know what features were selected.

Thanks!!

Jason Brownlee September 14, 2020 at 6:50 am #

REPLY

Sorry Diego, perhaps I'm missing something. How would it matter?

Diego September 15, 2020 at 4:40 am #

Hi Jason,
Sorry may be I was not clear.

Let's say we have features 1,2,3,4 and 5 as part of a Pipeline
Step 1 is MinMaxScaler for features 4 and 5 only (using column_transformer)
Step 2 is RFE or RFECV
Step 3 is a model

After step 1, new column order is 4,5,1,2,3
If step 2 (RFE) says that it kept only first two columns: which two columns were selected ?

4 and 5 or 1 and 2?
I think 4 and 5, since at that point columns were re-ordered.

It happened to me and couldn't believe RFE had seleted those features, but when I thought they were re-ordered, then it made sense.

I hope it is more clear now.
Thanks a lot for your help

Jason Brownlee September 15, 2020 at 5:29 am #

I don't see any problem.

It does not matter what RFE selected, as you are evaluating the modeling pipeline. You are answering the question "what is the performance of this pipeline" not "what features would RFE select". If you want to know the latter, you can study that in isolation.

You allow RFE to choose features based on training data, just like allow the model to fit internal parameters based on the training data. You don't ask "what are the vales of the coefficients internal to the model" because it doesn't really matter when you're focused on model skill.

That all being said, you can access the RFE model in the pipeline or keep a reference to it and report the features that were selected from a single run.

Richard March 10, 2021 at 5:22 am #

REPLY

Hi, one occasion where it matters is with LightGBM and categorical features. Using the sklearn API with LightGBM, the categorical features are specified as a parameter to .fit(). Since the DataFrame is casted to a numpy array during transformation (with for instance StandardScaler()), it is practical to specify categorical features with a list of int. Reordering of columns then makes for a "hard to find" bug.

Diego September 16, 2020 at 11:33 pm #

REPLY

Hi Jason,

Yes. You're right. I should compare an entire pipeline Vs. another one, no matter what variables were selected.
From that perspective it makes sense.

Thanks for your explanation and dedication!!

Jason Brownlee September 17, 2020 at 6:47 am #

REPLY

You're welcome.

Keone November 18, 2020 at 11:37 am #

REPLY

Is there a way to inverse_transform via ColumnTransformer?

Jason Brownlee November 18, 2020 at 1:08 pm #

REPLY

Good question.

It does not look like it:
https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html

Zineb December 30, 2020 at 8:54 pm #

REPLY

Hi Jason,

In the second example of the section , I think you want pipeline.fit_transform instead of scaler.fit_transform !

Zineb December 30, 2020 at 8:56 pm #

REPLY

the section "Challenge of Transforming Different Data Types"

Jason Brownlee December 31, 2020 at 5:25 am #

REPLY

Thanks! Fixed.

SULAIMAN KHAN February 15, 2021 at 10:38 pm #

REPLY

model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64)
##############
InvalidArgumentError: 2 root error(s) found.
(0) Invalid argument: indices[55,1] = 170355 is not in [0, 5000)
[[node sequential_3/embedding_3/embedding_lookup (defined at :9)]]

(1) Invalid argument: indices[55,1] = 170355 is not in [0, 5000)
[[node sequential_3/embedding_3/embedding_lookup (defined at :9)]]
[[Adam/Adam/update/AssignSubVariableOp/_35]]
0 successful operations.
0 derived errors ignored. [Op:__inference_train_function_12567]

Errors may have originated from an input operation.
Input Source operations connected to node sequential_3/embedding_3/embedding_lookup:
sequential_3/embedding_3/embedding_lookup/11365 (defined at /usr/lib/python3.6/contextlib.py:81)

Input Source operations connected to node sequential_3/embedding_3/embedding_lookup:
sequential_3/embedding_3/embedding_lookup/11365 (defined at /usr/lib/python3.6/contextlib.py:81)

Function call stack:
train_function -> train_function

How to fix this error?

Jason Brownlee February 16, 2021 at 6:06 am #

REPLY

Sorry to hear that, these tips may help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

SULAIMAN KHAN February 15, 2021 at 10:41 pm #

REPLY

I combined all my numeric columns and categorical columns. they are giving error in Loss function.

Jason Brownlee February 16, 2021 at 6:06 am #

REPLY

Perhaps inspect the output of your data preparation step to confirm the data has the shape and values that you expect?

Leen April 17, 2021 at 2:07 pm #

REPLY

Hello Jason,

Great tutorial. Can we make Column Transformer part of param_grid ? I want to transform some columns using categorical encoding mechanisms, but there are several of interest: Target Encoding, Weights of Evidence encoding, catboost encoding, etc. Therefore, I want each encoding mechanism to be a hyper parameter inside grid search param_grid. Do you think its possible ?

Jason Brownlee April 18, 2021 at 5:51 am #

REPLY

Hmm, you might have to run your grid search for loop manually.

JG April 19, 2021 at 6:05 pm #

REPLY

Hi Jason,

I see ColumnTransformer() as a very powerful module to apply globally, but distinguishing transformation to every feature of dataset.
I think it is useful not only inside the Pipeline steps but also as stand alone in order to get an inside analysis of features

thank you very much for this tutorial !.
regards,

Jason Brownlee April 20, 2021 at 5:55 am #

REPLY

Agreed!

Leo May 7, 2021 at 11:48 am #

REPLY

Thanks Jason!

I'm having a hard time applying columnTransform (or pipelines) to a group of: (i) one NLP-based feature (that require count vectorizer or tf-idf fit-transform); and (ii) other simpler features that require standard scaling – any chance you've ever faced a similar problem and know how to solve it?

The count vectorizer (or tf-idf) outputs a sparse matrix (while the rest remain as pandas series/df)… should I use numpy arrays for all of them?

Thanks again!

Jason Brownlee May 8, 2021 at 6:31 am #

REPLY

Perhaps get your columntransformer working with a minimal dataset or pipeline, then add elements back until you discover the cause of the issue.

Leo May 9, 2021 at 6:09 am #

REPLY

Sorted w/ FeatureUnion and ColumnSelector (from mlxtended)!

cat_pipe = Pipeline([('selector', ColumnSelector(cat_features)),
('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
('encoder', OneHotEncoder(handle_unknown='ignore', sparse=False))])

num_pipe = Pipeline([('selector', ColumnSelector(num_features)),
('imputer', SimpleImputer(strategy='mean')),
('scaler', MinMaxScaler())])

text_pipe = Pipeline([('selector', ColumnSelector(text_features,drop_axis=True)),
('tf', TfidfVectorizer(preprocessor=lambda x: x,tokenizer=lambda x: x))])

preprocessor = FeatureUnion(transformer_list=[('cat', cat_pipe),
('num', num_pipe),
('nlp', text_pipe)])

Jason Brownlee May 10, 2021 at 6:18 am #

REPLY

Well done!

JG July 3, 2021 at 9:43 pm #

REPLY

Hi Jason,

thanks for the tutorial!

Always a valuable piece of code to perform other experiments!

my experiments:

Is true that ColumnTransformer() API perform different transformation by column or feature (great!) and there is an argument remainder= that we can set to 'passthrough'. Ok.

But my complain to Sklearn API is that transformations are performed consecutively, so if you decide e.g. to perform column 3 transformation before than 2 …so column 2 is replaced by column 3 transformed.

In addition to that, even if you decide to 'passthrough' a remainder column without transformation, between two alternate columns transformation, the ColumnTransformer() replace previous column with the following transformation and leave column (without transformation) at the end of columns serie …

This behaviour introduce a lot of confusion to the way we would expected to work the ColumnTransformer, and it make you wast your time with problems that can be avoided by a good API design :-((
unless would be some other arguments where you can specify them to avoid this missbehaviour columns alterations.

Anyway, as I said before, thank you to your piece of code you can foreseen this behaviour.

regards,

Jason Brownlee July 4, 2021 at 6:03 am #

REPLY

Agreed.

It might be easier to stack ColumnTransformers into a pipeline and perform one subset/operation at a time in sequence.

Vishwanath reddy July 22, 2021 at 4:43 am #

REPLY

Thank you very much
It was very helpful

Jason Brownlee July 22, 2021 at 5:38 am #

REPLY

You're welcome.

Cyanide Lancer December 22, 2021 at 9:48 pm #

REPLY

Just a basic question, why haven't you considered splitting the dataset into train and test for features and labels?

Bobby K May 8, 2022 at 1:13 pm #

REPLY

Good stuff!

You manage to strike a nice balance between complexity and usability and the result is easy to read, easy to follow example that opens the door to a larger field to be explored at once's own pace.

James Carmichael May 9, 2022 at 11:03 am #

REPLY

Thank you for the feedback and support Bobby!

Giovanna April 30, 2023 at 1:33 am #

REPLY

Hi,

I've been researching high and low for an answer and haven't been lucky yet. I'm training a pipeline that consists of data transformation pipelines for both numerical and categorical, added to a column transformer pipeline:

num_pipeline = Pipeline([
('std_scaler', StandardScaler()),
('imputer', IterativeImputer(random_state=seed, max_iter=100,
estimator=DecisionTreeRegressor(max_features='sqrt',
random_state=seed)))])

cat_pipeline = Pipeline([('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),
('encoding', OneHotEncoder(handle_unknown='ignore', sparse=False))])

data_pipeline = ColumnTransformer([
('numerical', num_pipeline, num_feats),
('categorical', cat_pipeline, cat_feats)])

The issue that I'm facing is that I will fit_transform this data_pipeline to my training data and save this trained pipeline with joblib dump to use it for transforming with .transform() the validation data and also for inference in production. So far so good, however, when my training data is large-ish the saved file is very large and takes a long time to load (example: trained with 1.4M rows, saved pipeline is 49GB).

I'm wondering what I could be doing to make this saved trained data transformation pipeline lighter. Any tips?

Pascal August 7, 2023 at 10:20 pm #

REPLY

Hi, thanks for writing these examples down.
In the last line of your code example: I am not 100% sure, but is it OK to take

std(scores)

here?

This is because the values were previously set to absolute. The distances between the values can therefore be different due to the "zero border". So if I had a value of -0.1 before and the mean is 0.1, then the distance = 0.2; but after I apply absolute(), then the value is now 0.1 and has a distance of 0 to 0.1. This affects the standard deviation, doesn't it?

So you would have to take the standard deviation from the scores BEFORE applying absolute(), or am I wrong?

James Carmichael August 8, 2023 at 10:22 am #

REPLY

Hi Pascal…The following may help clarify how to use std():

https://numpy.org/doc/stable/reference/generated/numpy.std.html

Pascal August 10, 2023 at 7:14 pm #

REPLY

This does not answer my question… see this:

series of values before taking the absolute values:
0,1 -0,1 0,5 -0,2 -0,5 -0,7 0,8
std here: 0,530498418

series of values before taking the absolute values:
0,1 0,1 0,5 0,2 0,5 0,7 0,8
std here: 0,285356919

Leave a Reply

Name (required)

Email (will not be published) (required)

SUBMIT COMMENT

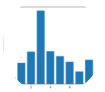
Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

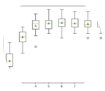
Picked for you:

How to Choose a Feature Selection Method For Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)



How to Calculate Feature Importance With Python



Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Loving the Tutorials?

The Data Preparation EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE