Data Preparation for Machine Learning

7-Day Crash-Course

Jason Brownlee

MACHINE
LEARNING
MASTERY

Data Preparation for Machine Learning Crash Course

© Copyright 2021 Jason Brownlee. All Rights Reserved.

Edition: v1.2

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started...

Data preparation involves transforming raw data into a form that is more appropriate for modeling. Preparing data may be the most important part of a predictive modeling project and the most time-consuming, although it seems to be the least discussed. Instead, the focus is on machine learning algorithms, whose usage and parameterization has become quite routine. Practical data preparation requires knowledge of data cleaning, feature selection data transforms, dimensionality reduction, and more. In this crash course, you will discover how you can get started and confidently prepare data for a predictive modeling project with Python in seven days. Let's get started.

Who Is This Crash-Course For?

Before we get started, let's make sure you are in the right place. This course is for developers that may know some applied machine learning. Maybe you know how to work through a predictive modeling problem end-to-end, or at least most of the main steps, with popular tools. The lessons in this course do assume a few things about you, such as: **You need to know**:

- You know your way around basic Python for programming.
- You may know some basic NumPy for array manipulation.
- You may know some basic scikit-learn for modeling.

You do NOT need to know:

- You do not need to be a math wiz!
- You do not need to be a machine learning expert!

This crash course will take you from a developer who knows a little machine learning to a developer who can effectively and competently prepare data for a predictive modeling project. Note that this crash course assumes you have a working Python 3 SciPy environment with at least NumPy installed. If you need help with your environment, you can follow the step-by-step tutorial here:

- How to Setup a Python Environment for Machine Learning

Crash-Course Overview

This crash course is broken down into seven lessons. You could complete one lesson per day (recommended) or complete all of the lessons in one day (hardcore). It really depends on the time you have available and your level of enthusiasm. Below is a list of the seven lessons that will get you started and productive with probability for machine learning in Python:

- **Lesson 01**: Importance of Data Preparation.
- **Lesson 02**: Fill Missing Values With Imputation.
- **Lesson 03**: Select Features With RFE.
- **Lesson 04**: Scale Data With Normalization.
- **Lesson 05**: Transform Categories With One Hot Encoding.
- **Lesson 06**: Transform Numbers to Categories With kBins.
- **Lesson 07**: Dimensionality Reduction with PCA.

Each lesson could take you 60 seconds or up to 30 minutes. Take your time and complete the lessons at your own pace. Ask questions and even share your results online. The lessons expect you to go off and find out how to do things. I will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help on and about the best-of-breed tools in Python. (Hint: I have all of the answers directly on this blog; use the search box.) Share your results online, I'll cheer you on!

Hang in there, don't give up!

Lesson 01: Importance of Data Preparation

In this lesson, you will discover the importance of data preparation in predictive modeling with machine learning. Predictive modeling projects involve learning from data. Data refers to examples or cases from the domain that characterize the problem you want to solve. On a predictive modeling project, such as classification or regression, raw data typically cannot be used directly. There are four main reasons why this is the case:

- **Data Types**: Machine learning algorithms require data to be numbers.
- **Data Requirements**: Some machine learning algorithms impose requirements on the data.
- **Data Errors**: Statistical noise and errors in the data may need to be corrected.
- **Data Complexity**: Complex nonlinear relationships may be teased out of the data.

The raw data must be pre-processed prior to being used to fit and evaluate a machine learning model. This step in a predictive modeling project is referred to as *data preparation*. There are common or standard tasks that you may use or explore during the data preparation step in a machine learning project. These tasks include:

- **Data Cleaning**: Identifying and correcting mistakes or errors in the data.
- **Feature Selection**: Identifying those input variables that are most relevant to the task.
- **Data Transforms**: Changing the scale or distribution of variables.
- **Feature Engineering**: Deriving new variables from available data.
- **Dimensionality Reduction**: Creating compact projections of the data.

Each of these tasks is a whole field of study with specialized algorithms.

Your Task

For this lesson, you must list three data preparation algorithms that you know of or may have used before and give a one-line summary for its purpose. One example of a data preparation algorithm is data normalization that scales numerical variables to the range between zero and one.

Next

In the next lesson, you will discover how to fix data that has missing values, called data imputation.

Lesson 02: Fill Missing Values With Imputation

In this lesson, you will discover how to identify and fill missing values in data. Real-world data often has missing values. Data can have missing values for a number of reasons, such as observations that were not recorded and data corruption. Handling missing data is important as many machine learning algorithms do not support data with missing values. Filling missing values with data is called data imputation and a popular approach for data imputation is to calculate a statistical value for each column (such as a mean) and replace all missing values for that column with the statistic.

The horse colic dataset describes medical characteristics of horses with colic and whether they lived or died. It has missing values marked with a question mark '?'. We can load the dataset with the `read_csv()` function and ensure that question mark values are marked as NaN. Once loaded, we can use the `SimpleImputer` class to transform all missing values marked with a NaN value with the mean of the column. The complete example is listed below.

```python
# statistical imputation transform for the horse colic dataset
from numpy import isnan
from pandas import read_csv
from sklearn.impute import SimpleImputer
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/horse-colic.csv'
dataframe = read_csv(url, header=None, na_values='?')
# split into input and output elements
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]
# print total missing
print('Missing: %d' % sum(isnan(X).flatten()))
# define imputer
imputer = SimpleImputer(strategy='mean')
# fit on the dataset
imputer.fit(X)
# transform the dataset
Xtrans = imputer.transform(X)
# print total missing
print('Missing: %d' % sum(isnan(Xtrans).flatten()))
```

Listing 1: Example of imputing missing values.

Your Task

For this lesson, you must run the example and review the number of missing values in the dataset before and after the data imputation transform.

Next

In the next lesson, you will discover how to select the most important features in a dataset.

Lesson 03: Select Features With RFE

In this lesson, you will discover how to select the most important features in a dataset. Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model. Recursive Feature Elimination, or RFE for short, is a popular feature selection algorithm. RFE is popular because it is easy to configure and use and because it is effective at selecting those features (columns) in a training dataset that are more or most relevant in predicting the target variable.

The scikit-learn Python machine learning library provides an implementation of RFE for machine learning. RFE is a transform. To use it, first, the class is configured with the chosen algorithm specified via the `estimator` argument and the number of features to select via the `n_features_to_select` argument. The example below defines a synthetic classification dataset with five redundant input features. RFE is then used to select five features using the decision tree algorithm.

```
# report which features were selected by RFE
from sklearn.datasets import make_classification
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5,
    random_state=1)
# define RFE
rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=5)
# fit RFE
rfe.fit(X, y)
# summarize all features
for i in range(X.shape[1]):
  print('Column: %d, Selected=%s, Rank: %d' % (i, rfe.support_[i], rfe.ranking_[i]))
```

Listing 2: Example of feature selection with RFE.

Your Task

For this lesson, you must run the example and review which features were selected and the relative ranking that each input feature was assigned.

Next

In the next lesson, you will discover how to scale numerical data.

Lesson 04: Scale Data With Normalization

In this lesson, you will discover how to scale numerical data for machine learning. Many machine learning algorithms perform better when numerical input variables are scaled to a standard range. This includes algorithms that use a weighted sum of the input, like linear regression, and algorithms that use distance measures, like k-nearest neighbors.

One of the most popular techniques for scaling numerical data prior to modeling is normalization. Normalization scales each input variable separately to the range 0-1, which is the range for floating-point values where we have the most precision. It requires that you know or are able to accurately estimate the minimum and maximum observable values for each variable. You may be able to estimate these values from your available data. You can normalize your dataset using the scikit-learn object `MinMaxScaler`. The example below defines a synthetic classification dataset, then uses the `MinMaxScaler` to normalize the input variables.

```
# example of normalizing input data
from sklearn.datasets import make_classification
from sklearn.preprocessing import MinMaxScaler
# define dataset
X, y = make_classification(n_samples=1000, n_features=5, n_informative=5, n_redundant=0,
    random_state=1)
# summarize data before the transform
print(X[:3, :])
# define the scaler
trans = MinMaxScaler()
# transform the data
X_norm = trans.fit_transform(X)
# summarize data after the transform
print(X_norm[:3, :])
```

Listing 3: Example of scaling numerical data.

Your Task

For this lesson, you must run the example and report the scale of the input variables both prior to and then after the normalization transform. For bonus points, calculate the minimum and maximum of each variable before and after the transform to confirm it was applied as expected.

Next

In the next lesson, you will discover how to transform categorical variables to numbers.

Lesson 05: Transform Categories With One Hot Encoding

In this lesson, you will discover how to encode categorical input variables as numbers. Machine learning models require all input and output variables to be numeric. This means that if your data contains categorical data, you must encode it to numbers before you can fit and evaluate a model. One of the most popular techniques for transforming categorical variables into numbers is the one hot encoding. Categorical data are variables that contain label values rather than numeric values.

Each label for a categorical variable can be mapped to a unique integer, called an ordinal encoding. Then, a one hot encoding can be applied to the ordinal representation. This is where one new binary variable is added to the dataset for each unique integer value in the variable, and the original categorical variable is removed from the dataset. For example, imagine we have a *color* variable with three categories (*red*, *green*, and *blue*). In this case, three binary variables are needed. A "1" value is placed in the binary variable for the color and "0" values for the other colors.

This one hot encoding transform is available in the scikit-learn Python machine learning library via the `OneHotEncoder` class. The breast cancer dataset contains only categorical input variables. The example below loads the dataset and one hot encodes each of the categorical input variables.

```
# one hot encode the breast cancer dataset
from pandas import read_csv
from sklearn.preprocessing import OneHotEncoder
# define the location of the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/breast-cancer.csv"
# load the dataset
dataset = read_csv(url, header=None)
# retrieve the array of data
data = dataset.values
# separate into input and output columns
X = data[:, :-1].astype(str)
y = data[:, -1].astype(str)
# summarize the raw data
print(X[:3, :])
# define the one hot encoding transform
encoder = OneHotEncoder(sparse=False)
# fit and apply the transform to the input data
X_oe = encoder.fit_transform(X)
# summarize the transformed data
print(X_oe[:3, :])
```

Listing 4: Example of one hot encoding categorical data.

Your Task

For this lesson, you must run the example and report on the raw data before the transform, and the impact on the data after the one hot encoding was applied.

Next

In the next lesson, you will discover how to transform numerical variables into categories.

Lesson 06: Transform Numbers to Categories With kBins

In this lesson, you will discover how to transform numerical variables into categorical variables. Some machine learning algorithms may prefer or require categorical or ordinal input variables, such as some decision tree and rule-based algorithms. This could be caused by outliers in the data, multi-modal distributions, highly exponential distributions, and more. Many machine learning algorithms prefer or perform better when numerical input variables with non-standard distributions are transformed to have a new distribution or an entirely new data type.

One approach is to use the transform of the numerical variable to have a discrete probability distribution where each numerical value is assigned a label and the labels have an ordered (ordinal) relationship. This is called a discretization transform and can improve the performance of some machine learning models for datasets by making the probability distribution of numerical input variables discrete.

The discretization transform is available in the scikit-learn Python machine learning library via the `KBinsDiscretizer` class. It allows you to specify the number of discrete bins to create (`n_bins`), whether the result of the transform will be an ordinal or one hot encoding (`encode`), and the distribution used to divide up the values of the variable (`strategy`), such as '`uniform`'. The example below creates a synthetic input variable with 10 numerical input variables, then encodes each into 10 discrete bins with an ordinal encoding.

```
# discretize numeric input variables
from sklearn.datasets import make_classification
from sklearn.preprocessing import KBinsDiscretizer
# define dataset
X, y = make_classification(n_samples=1000, n_features=5, n_informative=5, n_redundant=0,
    random_state=1)
# summarize data before the transform
print(X[:3, :])
# define the transform
trans = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')
# transform the data
X_discrete = trans.fit_transform(X)
# summarize data after the transform
print(X_discrete[:3, :])
```

Listing 5: Example of transforming numerical to categorical variables.

Your Task

For this lesson, you must run the example and report on the raw data before the transform, and then the effect the transform had on the data. For bonus points, explore alternate configurations of the transform, such as different strategies and number of bins.

Next

In the next lesson, you will discover how to reduce the dimensionality of input data.

Lesson 07: Dimensionality Reduction With PCA

In this lesson, you will discover how to use dimensionality reduction to reduce the number of input variables in a dataset. The number of input variables or features for a dataset is referred to as its dimensionality. Dimensionality reduction refers to techniques that reduce the number of input variables in a dataset. More input features often make a predictive modeling task more challenging to model, more generally referred to as the curse of dimensionality.

Although on high-dimensionality statistics, dimensionality reduction techniques are often used for data visualization, these techniques can be used in applied machine learning to simplify a classification or regression dataset in order to better fit a predictive model. Perhaps the most popular technique for dimensionality reduction in machine learning is Principal Component Analysis, or PCA for short. This is a technique that comes from the field of linear algebra and can be used as a data preparation technique to create a projection of a dataset prior to fitting a model. The resulting dataset, the projection, can then be used as input to train a machine learning model.

The scikit-learn library provides the `PCA` class that can be fit on a dataset and used to transform a training dataset and any additional datasets in the future. The example below creates a synthetic binary classification dataset with 10 input variables then uses PCA to reduce the dimensionality of the dataset to the three most important components.

```
# example of pca for dimensionality reduction
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=3, n_redundant=7,
    random_state=1)
# summarize data before the transform
print(X[:3, :])
# define the transform
trans = PCA(n_components=3)
# transform the data
X_dim = trans.fit_transform(X)
# summarize data after the transform
print(X_dim[:3, :])
```

Listing 6: Example of dimensionality reduction with PCA.

Your Task

For this lesson, you must run the example and report on the structure and form of the raw dataset and the dataset after the transform was applied. For bonus points, explore transforms with different numbers of selected components. This was the final lesson in the mini-course.

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come. You discovered:

- The importance of data preparation in a predictive modeling machine learning project.
- How to mark missing data and impute the missing values using statistical imputation.
- How to remove redundant input variables using recursive feature elimination.
- How to transform input variables with differing scales to a standard range called normalization.
- How to transform categorical input variables to be numbers called one hot encoding.
- How to transform numerical variables into discrete categories called discretization.
- How to use PCA to create a projection of a dataset into a lower number of dimensions.

This is just the beginning of your journey with data preparation. Keep practicing and developing your skills. Take the next step and check out my book on **Data Preparation for Machine Learning**.

How Did You Go With The Crash-Course?

Did you enjoy this crash-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

Take the Next Step