Machine Learning Algorithm Recipes in scikit-learn

by **Jason Brownlee** on June 20, 2014 in **Python Machine Learning**

Last Updated on August 21, 2019

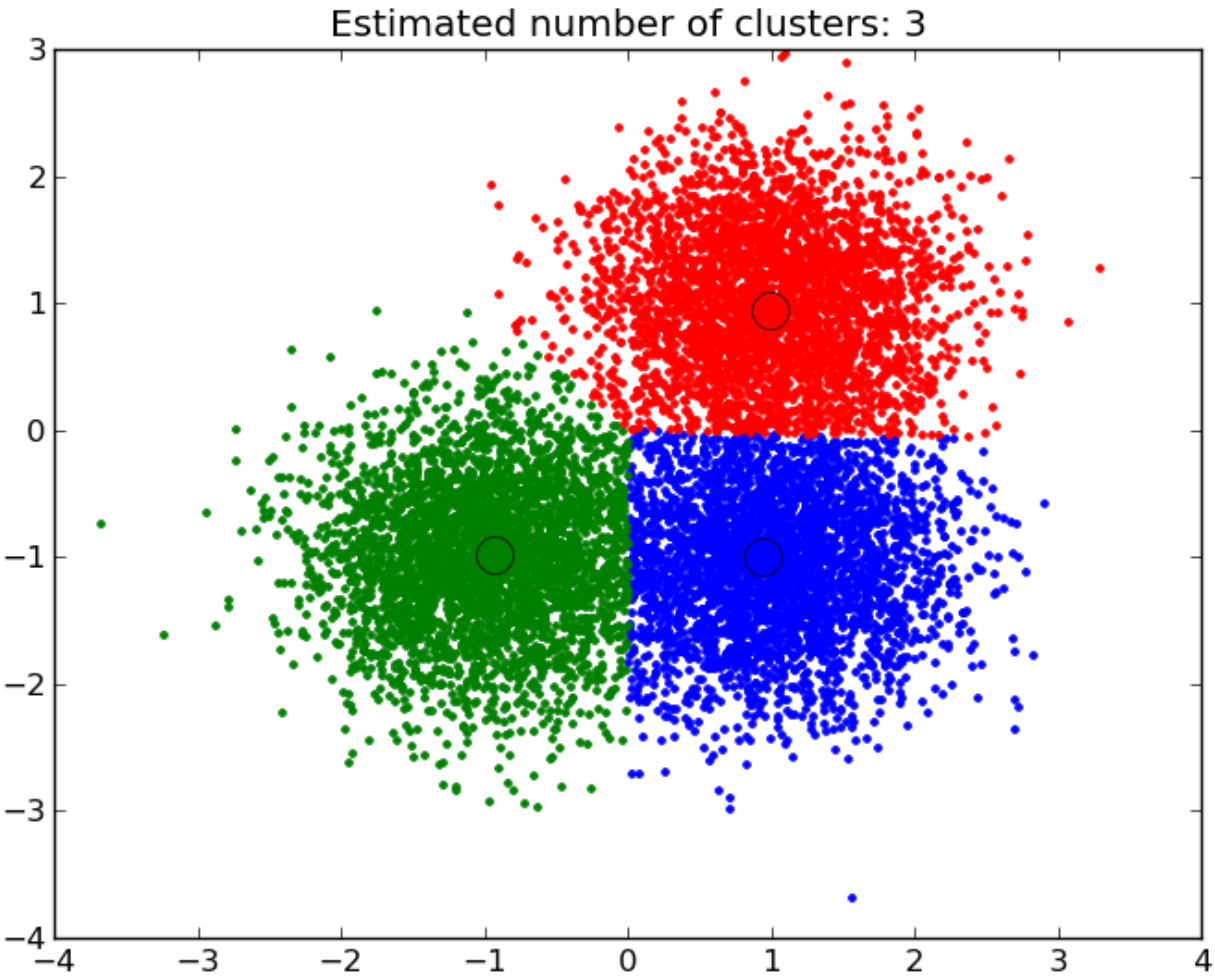
You have to get your hands dirty.

You can read all of the blog posts and watch all the videos in the world, but you're not actually going to start really get machine learning until you start practicing.

The scikit-learn Python library is very easy to get up and running. Nevertheless I see a lot of hesitation from beginners looking get started. In this blog post I want to give a few very simple examples of using scikit-learn for some supervised classification algorithms.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more in my new book, with 16 step-by-step tutorials, 3 projects, and full python code.

Let's get started.



Scikit-Learn Recipes

You don't need to know about and use all of the algorithms in scikit-learn, at least initially, pick one or two (or a handful) and practice with only those.

In this post you will see 5 recipes of supervised classification algorithms applied to small standard datasets that are provided with the scikit-learn library.

The recipes are principled. Each example is:

- **Standalone**: Each code example is a self-contained, complete and executable recipe.
- **Just Code**: The focus of each recipe is on the code with minimal exposition on machine learning theory.
- **Simple**: Recipes present the common use case, which is probably what you are looking to do.
- **Consistent**: All code example are presented consistently and follow the same code pattern and style conventions.

The recipes do not explore the parameters of a given algorithm. They provide a skeleton that you can copy and paste into your file, project or python REPL and start to play with immediately.

These recipes show you that you can get started practicing with scikit-learn right now. Stop putting it off.

Logistic Regression

Logistic regression fits a logistic model to data and makes predictions about the probability of an event (between 0 and 1).

This recipe shows the fitting of a logistic regression model to the iris dataset. Because this is a mutli-class classification problem and logistic regression makes predictions between 0 and 1, a one-vs-all scheme is used (one model per class).

```
# Logistic Regression
from sklearn import datasets
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
# load the iris datasets
dataset = datasets.load_iris()
# fit a logistic regression model to the data
model = LogisticRegression()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

For more information see the API reference for Logistic Regression for details on configuring the algorithm parameters. Also see the Logistic Regression section of the user guide.

Naive Bayes

Naive Bayes uses Bayes Theorem to model the conditional relationship of each attribute to the class variable.

This recipe shows the fitting of an Naive Bayes model to the iris dataset.

```
# Gaussian Naive Bayes
from sklearn import datasets
from sklearn import metrics
```

```
# load the iris datasets
dataset = datasets.load_iris()
# fit a Naive Bayes model to the data
model = GaussianNB()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

For more information see the API reference for the Gaussian Naive Bayes for details on configuring the algorithm parameters. Also see the Naive Bayes section of the user guide.

k-Nearest Neighbor

The k-Nearest Neighbor (kNN) method makes predictions by locating similar cases to a given data instance (using a similarity function) and returning the average or majority of the most similar data instances. The kNN algorithm can be used for classification or regression.

This recipe shows use of the kNN model to make predictions for the iris dataset.

```
# k-Nearest Neighbor
from sklearn import datasets
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
# load iris the datasets
dataset = datasets.load_iris()
# fit a k-nearest neighbor model to the data
model = KNeighborsClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

For more information see the API reference for the k-Nearest Neighbor for details on configuring the algorithm parameters. Also see the k-Nearest Neighbor section of the user guide.

Classification and Regression Trees

Classification and Regression Trees (CART) are constructed from a dataset by making splits that best separate the data for the classes or predictions being made. The CART algorithm can be used for classification or regression.

This recipe shows use of the CART model to make predictions for the iris dataset.

```
# Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
```

```
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

For more information see the API reference for CART for details on configuring the algorithm parameters. Also see the Decision Tree section of the user guide.

Support Vector Machines

Support Vector Machines (SVM) are a method that uses points in a transformed problem space that best separate classes into two groups. Classification for multiple classes is supported by a one-vs-all method. SVM also supports regression by modeling the function with a minimum amount of allowable error.

This recipe shows use of the SVM model to make predictions for the iris dataset.

```
# Support Vector Machine
from sklearn import datasets
from sklearn import metrics
from sklearn.svm import SVC
# load the iris datasets
dataset = datasets.load_iris()
# fit a SVM model to the data
model = SVC()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

For more information see the API reference for SVM for details on configuring the algorithm parameters. Also see the SVM section of the user guide.

Summary

In this post you have seen 5 self-contained recipes demonstrating some of the most popular and powerful supervised classification problems.

Each example is less than 20 lines that you can copy and paste and start using scikit-learn, right now. Stop reading and start practicing. Pick one recipe and run it, then start to play with the parameters and see what effect that has on the results.

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 →

‹ The Best Machine Learning Algorithm

Prepare Data for Machine Learning in Python with Pandas ›

27 Responses to *Machine Learning Algorithm Recipes in scikit-learn*

DR Venugopala Rao Manneni April 7, 2016 at 5:31 pm #

REPLY

Thanks for these Jason. Can you also please give the same for Neural networks (MLP)

Ajinkya June 12, 2016 at 8:48 am #

REPLY

Thanks for this informative tutorial.
Can you please explain how logistic regression is used for classification where more than 2 classes are

involved.?
Thanks

Jason Brownlee June 14, 2016 at 8:14 am #

REPLY

Great question Ajinkya.

Generally, you can take an algorithm designed for binary (two-class) classification and turn it into a multi-class classification algorithm by using the one-vs-all meta algorithm. You create n models, where n is the number of classes. Each model makes a prediction to provide a vector of predictions and the final prediction can be taken as the model for the class that had the highest probability.

This can be used with logistic regression and is very popular with support vector machines.

More on the one-vs-all meta algorithm here:
https://en.wikipedia.org/wiki/Multiclass_classification

Nicolas November 23, 2016 at 1:12 am #

REPLY

Hey

Thank you very much for these helpful examples! I searched a lot until I found this website. You actually saved me a lot of time and nerves with doing an assignment for my ML course at my university 🙂

Keep up the great work!

Jason Brownlee November 23, 2016 at 9:00 am #

REPLY

I'm very glad to hear that Nicolas.

Ash October 24, 2018 at 2:11 am #

REPLY

Hi Jason, How do which algorithm I can use to compare nearest match for a "String" value and then also test its accuracy. e.g. my data has value FR for country but I need FRA, how do I ensure that I predict FRA and provide a accurate predicted match to the end users? Sorry very basic question but new to ML hence the question.

Jason Brownlee October 24, 2018 at 6:31 am #

REPLY

Sorry, I don't have material on string matching/similarity algorithms.

Gill Bates February 11, 2017 at 3:18 am #

REPLY

Dear Jason,
Great job.
Can you please show how to implement other algorithms or “how to catch fish”?
Tks.

Jason Brownlee February 11, 2017 at 5:05 am #

REPLY

Which algorithms Gill?

lalit April 6, 2017 at 9:32 pm #

REPLY

Test data should not be used for training. Here you are using full training data as test data which is wrong

Jason Brownlee April 9, 2017 at 2:39 pm #

REPLY

Yes, I agree. These are just examples on how to fit models in sklearn.

Adi Usman October 20, 2019 at 9:39 am #

REPLY

Thanks for the wonderful beginners’s tutorial. It actually got started. Could you please explain how to interpret the reslts results?

Jason Brownlee October 21, 2019 at 6:12 am #

REPLY

Thanks.

Sure, which part?

Brian Tremaine July 28, 2017 at 3:17 am #

REPLY

Thank you for this tutorial, very helpfull.

I have run the MNIST character recognition using Naive Bayes (GaussianNB) and the results were very poor compared to nearest neighbors. Is the an sklearn function for Bayes that uses priors? I've searched but haven't found anything,

Thanks,
Brian

Jason Brownlee July 28, 2017 at 8:33 am #

REPLY

I would expect that naive Bayes in sklearn would use priors.

The only time priors are dropped is when they add nothing to the equation (e.g. both classes have the same number of obs).

Jarrell R Dunson October 24, 2017 at 6:53 am #

REPLY

Question…I'm trying the code for sklearn.naive_bayes import GaussianNB

but this doesn't seem to work from Python 3.5 or 3.6 …

Is this only to run in Python 2?

Jason Brownlee October 24, 2017 at 3:57 pm #

REPLY

No. It works with py2 and py3.

Perhaps double check your version of sklearn?

Jarrell R Dunson October 25, 2017 at 12:51 am #

REPLY

Thanks… upgraded sklearn, and it works

Jason Brownlee October 25, 2017 at 6:48 am #

REPLY

Glad to hear it!

DG March 1, 2018 at 8:46 am #

REPLY

Thanks for the info, can you post similar examples for cluster analysis or K-means using quantitative and qualitative data?

Jason Brownlee March 1, 2018 at 3:08 pm #

REPLY

Thanks for the suggestion.

Jesús Martínez April 18, 2018 at 1:20 am #

REPLY

Awesome. Scikit-learn is great. Thanks for sharing!

Jason Brownlee April 18, 2018 at 8:10 am #

REPLY

Thanks.

Fredrick Ughimi February 11, 2019 at 4:08 am #

REPLY

Hello Jason, thanks for the time and efforts you put into all this. Very streamlined informative tutorial. More grease.

Jason Brownlee February 11, 2019 at 8:01 am #

REPLY

Thanks.

Jim March 3, 2019 at 9:42 am #

REPLY

Hi Jason,

For logistic regression, I got warnings suggesting that I set both the solver and the multi_class arguments. So I used model = LogisticRegression(solver="newton-cg", multi_class="ovr") and this got rid of them.

Could you share any thoughts on what these two arguments are doing?

Thanks,
Jim

Jason Brownlee March 4, 2019 at 6:55 am #

REPLY

Yes, great question, you can learn more here:
https://machinelearningmastery.com/how-to-fix-futurewarning-messages-in-scikit-learn/

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

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

Picked for you:

Your First Machine Learning Project in Python Step-By-Step

How to Setup Your Python Environment for Machine Learning with Anaconda

Feature Selection For Machine Learning in Python

Python Machine Learning Mini-Course

Save and Load Machine Learning Models in Python with scikit-learn

Loving the Tutorials?

The Machine Learning with Python EBook
is where I keep the ***Really Good*** stuff.

SEE WHAT'S INSIDE