Chapter 1. Introduction

This is not so much an instructional manual, but rather notes, tables, and examples for machine learning. It was created by the author as an additional resource during training, meant to be distributed as a physical notebook. Participants (who favor the physical characteristics of dead-tree material) could add their own notes and thoughts and have a valuable reference of curated examples.

We will walk through classification with structured data. Other common machine learning applications include predicting a continuous value (regression), creating clusters, or trying to reduce dimensionality, among others. This book does not discuss deep learning techniques. While those techniques work well for unstructured data, most recommend the techniques in this book for structured data.

We assume knowledge and familiarity with Python. Learning how to manipulate data using the pandas library is useful. We have many examples using pandas, and it is an excellent tool for dealing with structured data. However, some of the indexing operations may be confusing if you are not familiar with numpy. Full coverage of pandas could be a book in itself.

Libraries Used

This book uses many libraries. This can be a good thing and a bad thing. Some of these libraries may be hard to install or conflict with other library versions. Do not feel like you need to install all of these libraries. Use “JIT installation” and only install the libraries that you want to use as you need them.

```
>>> import autosklearn, catboost,
category_encoders, dtreeviz, eli5, fancyimpute,
fastai, featuretools, glmnet_py, graphviz,
hdbscan, imblearn, janitor, lime, matplotlib,
missingno, mlxtend, numpy, pandas, pdpbox, phate,
pydotplus, rfpimp, scikitplot, scipy, seaborn,
shap, sklearn, statsmodels, tpot, treeinterpreter,
umap, xgbfir, xgboost, yellowbrick

>>> for lib in [
...     autosklearn,
...       catboost,
...       category_encoders,
...       dtreeviz,
...       eli5,
...       fancyimpute,
...       fastai,
...       featuretools,
...       glmnet_py,
...       graphviz,
...       hdbscan,
...       imblearn,
...       lime,
...       janitor,
...       matplotlib,
...       missingno,
...       mlxtend,
...       numpy,
...       pandas,
...       pandas_profiling,
...       pdpbox,
...       phate,
...       pydotplus,
...       rfpimp,
...       scikitplot,
...       scipy,
...       seaborn,
...       shap,
...       sklearn,
...       statsmodels,
...       tpot,
...       treeinterpreter,
...       umap,
...       xgbfir,
...       xgboost,
...       yellowbrick,
... ]:
...       try:
...           print(lib.__name__, lib.__version__)
...       except:
...           print("Missing", lib.__name__)
catboost 0.11.1
category_encoders 2.0.0
Missing dtreeviz
eli5 0.8.2
fancyimpute 0.4.2
fastai 1.0.28
featuretools 0.4.0
Missing glmnet_py
graphviz 0.10.1
hdbscan 0.8.22
imblearn 0.4.3
janitor 0.16.6
Missing lime
matplotlib 2.2.3
missingno 0.4.1
mlxtend 0.14.0
numpy 1.15.2
pandas 0.23.4
Missing pandas_profiling
pdpbox 0.2.0
```

```
phate 0.4.2
Missing pydotplus
rfpimp
scikitplot 0.3.7
scipy 1.1.0
seaborn 0.9.0
shap 0.25.2
sklearn 0.21.1
statsmodels 0.9.0
tpot 0.9.5
treeinterpreter 0.1.0
umap 0.3.8
xgboost 0.81
yellowbrick 0.9
```

NOTE

Most of these libraries are easily installed with `pip` or `conda`. With `fastai` I need to use `pip install --no-deps fastai`. The `umap` library is installed with `pip install umap-learn`. The `janitor` library is installed with `pip install pyjanitor`. The `autosklearn` library is installed with `pip install auto-sklearn`.

I usually use Jupyter for doing an analysis. You can use other notebook tools as well. Note that some, like Google Colab, have preinstalled many of the libraries (though they may be outdated versions).

There are two main options for installing libraries in Python. One is to use `pip` (an acronym for Pip Installs Python), a tool that comes with Python. The other option is to use Anaconda. We will introduce both.

Installation with Pip

Before using `pip`, we will create a sandbox environment to install our libraries into. This is called a virtual environment named `env`:

```
$ python -m venv env
```

NOTE

On Macintosh and Linux, use `python`; on Windows, use `python3`. If Windows doesn’t recognize that from the command prompt, you may need to reinstall or fix your install and make sure you check the “Add Python to my PATH” checkbox.

Then you activate the environment so that when you install libraries, they go in the sandbox environment and not in the global Python installation. As many of these libraries change and are updated, it is best to lock down versions on a per-project basis so you know that your code will run.

Here is how we activate the virtual environment on Linux and Macintosh:

```
$ source env/bin/activate
```

You will notice that the prompt is updated, indicating that we are using the virtual environment:

```
(env) $ which python
env/bin/python
```

On Windows, you will need to activate the environment by running this command:

```
C:> env\Scripts\activate.bat
```

Again, you will notice that the prompt is updated, indicating that we are using the virtual environment:

```
(env) C:> where python
env\Scripts\python.exe
```

On all platforms, you can install packages using `pip`. To install pandas, type:

```
(env) $ pip install pandas
```

Some of the package names are different than the library names. You can search for packages using:

```
(env) $ pip search libraryname
```

Once you have your packages installed, you can create a file with all of the versions of the packages using `pip`:

```
(env) $ pip freeze > requirements.txt
```

With this `requirements.txt` file you can easily install the packages into a new virtual environment:

```
(other_env) $ pip install -r requirements.txt
```

Installation with Conda

The `conda` tool comes with Anaconda and lets us create environments and install packages.

To create an environment named `env`, run:

```
$ conda create --name env python=3.6
```

To activate this environment, run:

```
$ conda activate env
```

This will update the prompt on both Unix and Windows systems. Now you can search for packages using:

```
(env) $ conda search libraryname
```

To install a package, like pandas, run:

```
(env) $ conda install pandas
```

To create a file with the package requirements in it, run:

```
(env) $ conda env export > environment.yml
```

To install these requirements in a new environment, run:

```
(other_env) $ conda create -f environment.yml
```