scikit_learn_cheatsheet

December 26, 2025

1 What we're covering in the Scikit-Learn Introduction

This notebook outlines the content convered in the Scikit-Learn Introduction.

It's a quick stop to see all the Scikit-Learn functions and modules for each section outlined.

What we're covering follows the following diagram detailing a Scikit-Learn workflow.

1.1 0. Standard library imports

For all machine learning projects, you'll often see these libraries (Matplotlib, NumPy and pandas) imported at the top.

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

We'll use 2 datasets for demonstration purposes. * `heart_disease` - a classification dataset (predicting whether someone has heart disease or not) * `boston_df` - a regression dataset (predicting the median house prices of cities in Boston)

[3]:
```
# Classification data
heart_disease = pd.read_csv("./data/heart-disease.csv")

# Regression data
from sklearn.datasets import fetch_california_housing
boston = fetch_california_housing() # loads as dictionary
# Convert dictionary to dataframe
boston_df = pd.DataFrame(boston["data"], columns=boston["feature_names"])
boston_df["target"] = pd.Series(boston["target"])
```

1.2 1. Get the data ready

[4]:
```
# Split data into X & y
X = heart_disease.drop("target", axis=1) # use all columns except target
y = heart_disease["target"] # we want to predict y using X
```

[5]:
```
# Split the data into training and test sets
from sklearn.model_selection import train_test_split
# Example use case (requires X & y)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

1.3 2. Pick a model/estimator (to suit your problem)

To pick a model we use the Scikit-Learn machine learning map.

Note: Scikit-Learn refers to machine learning models and algorithms as estimators.

[6]:
```
# Random Forest Classifier (for classification problems)
from sklearn.ensemble import RandomForestClassifier
# Instantiating a Random Forest Classifier (clf short for classifier)
clf = RandomForestClassifier()
```

[7]:
```
# Random Forest Regressor (for regression problems)
from sklearn.ensemble import RandomForestRegressor
# Instantiating a Random Forest Regressor
model = RandomForestRegressor()
```

1.4 3. Fit the model to the data and make a prediction

[8]:
```
# All models/estimators have the fit() function built-in
clf.fit(X_train, y_train)

# Once fit is called, you can make predictions using predict()
y_preds = clf.predict(X_test)

# You can also predict with probabilities (on classification models)
y_probs = clf.predict_proba(X_test)

# View preds/probabilities
y_preds, y_probs
```

[8]:
```
(array([1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0,
        1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
        0, 1, 1, 1, 1, 1, 1, 1, 0, 1]),
 array([[0.25, 0.75],
        [0.4 , 0.6 ],
        [0.99, 0.01],
        [0.  , 1.  ],
        [0.96, 0.04],
        [0.08, 0.92],
        [0.71, 0.29],
        [0.21, 0.79],
        [0.37, 0.63],
```

```
[0.86, 0.14],
[0.98, 0.02],
[0.45, 0.55],
[0.16, 0.84],
[0.5 , 0.5 ],
[1.  , 0.  ],
[0.67, 0.33],
[0.81, 0.19],
[0.71, 0.29],
[0.71, 0.29],
[0.85, 0.15],
[0.85, 0.15],
[0.18, 0.82],
[0.51, 0.49],
[0.47, 0.53],
[0.75, 0.25],
[0.6 , 0.4 ],
[0.2 , 0.8 ],
[0.13, 0.87],
[0.15, 0.85],
[0.97, 0.03],
[0.75, 0.25],
[0.11, 0.89],
[0.26, 0.74],
[0.38, 0.62],
[0.09, 0.91],
[0.17, 0.83],
[0.96, 0.04],
[0.65, 0.35],
[0.48, 0.52],
[0.22, 0.78],
[0.13, 0.87],
[1.  , 0.  ],
[0.07, 0.93],
[0.52, 0.48],
[0.43, 0.57],
[0.31, 0.69],
[0.31, 0.69],
[0.52, 0.48],
[0.79, 0.21],
[0.16, 0.84],
[0.07, 0.93],
[0.91, 0.09],
[0.76, 0.24],
[0.1 , 0.9 ],
[0.92, 0.08],
[0.02, 0.98],
```

```
[0.7 , 0.3 ],
[0.79, 0.21],
[0.35, 0.65],
[0.54, 0.46],
[0.93, 0.07],
[0.47, 0.53],
[0.2 , 0.8 ],
[0.21, 0.79],
[0.02, 0.98],
[0.33, 0.67],
[0.9 , 0.1 ],
[0.23, 0.77],
[0.41, 0.59],
[0.24, 0.76],
[0.05, 0.95],
[0.08, 0.92],
[0.46, 0.54],
[0.17, 0.83],
[1.  , 0.  ],
[0.26, 0.74]]))
```

1.5 4. Evaluate the model

Every Scikit-Learn model has a default metric which is accessible through the `score()` function.

However there are a range of different evaluation metrics you can use depending on the model you're using.

A full list of evaluation metrics can be found in the documentation.

[9]:
```
# All models/estimators have a score() function
clf.score(X_test, y_test)
```

[9]:
```
0.8157894736842105
```

[10]:
```
# Evaluting a model using cross-validation is possible with cross_val_score
from sklearn.model_selection import cross_val_score

# scoring=None means default score() metric is used
print(cross_val_score(estimator=clf,
                      X=X,
                      y=y,
                      cv=5, # use 5-fold cross-validation
                      scoring=None))

# Evaluate a model with a different scoring method
print(cross_val_score(estimator=clf,
                      X=X,
                      y=y,
```

```
                         cv=5, # use 5-fold cross-validation
                         scoring="precision"))
```

```
[0.83606557 0.90163934 0.7704918  0.8        0.78333333]
[0.80555556 0.93548387 0.84375    0.83870968 0.74358974]
```

[11]:
```
# Different classification metrics

# Accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_preds))

# Reciver Operating Characteristic (ROC curve)/Area under curve (AUC)
from sklearn.metrics import roc_curve, roc_auc_score
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_probs[:
 ↪, 1])
print(roc_auc_score(y_test, y_preds))

# Confusion matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_preds))

# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_preds))
```

```
0.8157894736842105
0.8142758142758144
[[28  9]
 [ 5 34]]
              precision    recall  f1-score   support

           0       0.85      0.76      0.80        37
           1       0.79      0.87      0.83        39

    accuracy                           0.82        76
   macro avg       0.82      0.81      0.81        76
weighted avg       0.82      0.82      0.82        76
```

[12]:
```
# Different regression metrics

# Make predictions first
X = boston_df.drop("target", axis=1)
y = boston_df["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = RandomForestRegressor()
```

```
model.fit(X_train, y_train)
y_preds = model.predict(X_test)

# R^2 (pronounced r-squared) or coefficient of determination
from sklearn.metrics import r2_score
print(r2_score(y_test, y_preds))

# Mean absolute error (MAE)
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_preds))

# Mean square error (MSE)
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_preds))
```

```
0.8144610655477476
0.320020089268411
0.23727575850434243
```

1.6 5. Improve through experimentation

Two of the main methods to improve a models baseline metrics (the first evaluation metrics you get).

From a data perspective asks: * Could we collect more data? In machine learning, more data is generally better, as it gives a model more opportunities to learn patterns. * Could we improve our data? This could mean filling in misisng values or finding a better encoding (turning things into numbers) strategy.

From a model perspective asks: * Is there a better model we could use? If you've started out with a simple model, could you use a more complex one? (we saw an example of this when looking at the Scikit-Learn machine learning map, ensemble methods are generally considered more complex models) * Could we improve the current model? If the model you're using performs well straight out of the box, can the **hyperparameters** be tuned to make it even better?

Hyperparameters are like settings on a model you can adjust so some of the ways it uses to find patterns are altered and potentially improved. Adjusting hyperparameters is referred to as hyperparameter tuning.

[13]:
```
# How to find a model's hyperparameters
clf = RandomForestClassifier()
clf.get_params() # returns a list of adjustable hyperparameters
```

[13]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
```

```
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

[14]:
```python
# Example of adjusting hyperparameters by hand

# Split data into X & y
X = heart_disease.drop("target", axis=1) # use all columns except target
y = heart_disease["target"] # we want to predict y using X

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Instantiate two models with different settings
clf_1 = RandomForestClassifier(n_estimators=100)
clf_2 = RandomForestClassifier(n_estimators=200)

# Fit both models on training data
clf_1.fit(X_train, y_train)
clf_2.fit(X_train, y_train)

# Evaluate both models on test data and see which is best
print(clf_1.score(X_test, y_test))
print(clf_2.score(X_test, y_test))
```

```
0.8157894736842105
0.8421052631578947
```

[ ]:
```python
# Example of adjusting hyperparameters computationally (recommended)

from sklearn.model_selection import RandomizedSearchCV

# Define a grid of hyperparameters
grid = {"n_estimators": [10, 100, 200, 500, 1000, 1200],
        "max_depth": [None, 5, 10, 20, 30],
        "max_features": ["auto", "sqrt"],
        "min_samples_split": [2, 4, 6],
        "min_samples_leaf": [1, 2, 4]}
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Set n_jobs to -1 to use all cores (NOTE: n_jobs=-1 is broken as of 8 Dec 2019,␣
 ↪using n_jobs=1 works)
clf = RandomForestClassifier(n_jobs=1)

# Setup RandomizedSearchCV
rs_clf = RandomizedSearchCV(estimator=clf,
                            param_distributions=grid,
                            n_iter=10, # try 10 models total
                            cv=5, # 5-fold cross-validation
                            verbose=2) # print out results

# Fit the RandomizedSearchCV version of clf
rs_clf.fit(X_train, y_train);

# Find the best hyperparameters
print(rs_clf.best_params_)

# Scoring automatically uses the best hyperparameters
rs_clf.score(X_test, y_test)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4,
min_samples_split=6, n_estimators=1000; total time=   2.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4,
min_samples_split=6, n_estimators=1000; total time=   2.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4,
min_samples_split=6, n_estimators=1000; total time=   2.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4,
min_samples_split=6, n_estimators=1000; total time=   2.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4,
min_samples_split=6, n_estimators=1000; total time=   2.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=   2.1s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=   2.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=   2.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=   2.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1000; total time=   2.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=4,
```

```
min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4,
min_samples_split=4, n_estimators=500; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4,
min_samples_split=4, n_estimators=500; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4,
min_samples_split=4, n_estimators=500; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4,
min_samples_split=4, n_estimators=500; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4,
min_samples_split=4, n_estimators=500; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1,
min_samples_split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1,
min_samples_split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1,
min_samples_split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1,
min_samples_split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1,
min_samples_split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1,
```

```
min_samples_split=6, n_estimators=1200; total time=   3.5s
```

1.7 6. Save and reload your trained model

You can save and load a model with `pickle`.

[ ]:
```
# Saving a model with pickle
import pickle

# Save an existing model to file
pickle.dump(rs_clf, open("rs_random_forest_model_1.pkl", "wb"))
```

[ ]:
```
# Load a saved pickle model
loaded_pickle_model = pickle.load(open("rs_random_forest_model_1.pkl", "rb"))

# Evaluate loaded model
loaded_pickle_model.score(X_test, y_test)
```

You can do the same with `joblib`. `joblib` is usually more efficient with numerical data (what our models are).

[ ]:
```
# Saving a model with joblib
from joblib import dump, load

# Save a model to file
dump(rs_clf, filename="gs_random_forest_model_1.joblib")
```

[ ]:
```
# Import a saved joblib model
loaded_joblib_model = load(filename="gs_random_forest_model_1.joblib")
```

[ ]:
```
# Evaluate joblib predictions
loaded_joblib_model.score(X_test, y_test)
```

1.8 7. Putting it all together (not pictured)

We can put a number of different Scikit-Learn functions together using `Pipeline`.

As an example, we'll use `car-sales-extended-missing-data.csv`. Which has missing data as well as non-numeric data. For a machine learning model to work, there can be no missing data or non-numeric values.

The problem we're solving here is predicting a cars sales price given a number of parameters about the car (a regression problem).

[ ]:
```
# Getting data ready
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
```

```
# Modelling
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV

# Setup random seed
import numpy as np
np.random.seed(42)

# Import data and drop the rows with missing labels
data = pd.read_csv("../data/car-sales-extended-missing-data.csv")
data.dropna(subset=["Price"], inplace=True)

# Define different features and transformer pipelines
categorical_features = ["Make", "Colour"]
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant", fill_value="missing")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))])

door_feature = ["Doors"]
door_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant", fill_value=4))])

numeric_features = ["Odometer (KM)"]
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="mean"))
])

# Setup preprocessing steps (fill missing values, then convert to numbers)
preprocessor = ColumnTransformer(
    transformers=[
        ("cat", categorical_transformer, categorical_features),
        ("door", door_transformer, door_feature),
        ("num", numeric_transformer, numeric_features)])

# Create a preprocessing and modelling pipeline
model = Pipeline(steps=[("preprocessor", preprocessor),
                        ("model", RandomForestRegressor())])

# Split data
X = data.drop("Price", axis=1)
y = data["Price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Fit and score the model
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[ ]: