Effective Programming Practices for Economists

Data Analysis in Python

Introduction to scikit-learn

Janoś Gabler, Hans-Martin von Gaudecker, and Tim Mensinger

Loading datasets from scikit-learn

- Toy datasets can be found using `sklearn.datasets.load_*`

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
```

- Real world datasets can be downloaded using `sklearn.datasets.fetch_*`

```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

- Some datasets can be generated using `sklearn.datasets.make_*`

```
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=100, n_features=1, noise=0.1)
```

Example: California Housing

```
>>> from sklearn.datasets import fetch_california_housing
>>> housing = fetch_california_housing()
>>> housing.keys()
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])

>>> housing["data"].shape
(20640, 8)

>>> housing["feature_names"]
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']

>>> housing["target"].shape
(20640,)

>>> housing["target_names"]
['MedHouseVal']
```

Re-define the target as 1 if the value is above the 70th-percentile, 0 otherwise:

```
>>> import numpy as np
>>> target = (housing["target"] > np.quantile(housing["target"], q=0.7)).astype(int)
```

Train-test splits

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(
...   housing["data"],
...   target,
...   random_state=1234,
...   test_size=0.3,
... )
>>> X_train.shape
(14448, 8)

>>> y_train.shape
(14448,)

>>> X_test.shape
(6192, 8)

>>> y_test.shape
(6192,)
```

- The function `train_test_split` lets you:
  - select the test set size
  - set `random_state` for reproducibility

Basic scikit-learn steps

- Arrange data into a features matrix / target vector, split into training / test sets
- Choose a class of models by importing the appropriate estimator
- Set hyperparameters by instantiating this class
- Fit the model to your data by calling the `fit()` method on the model instance
- Apply the model to new data using the `predict()` method
- Evaluate the quality of predictions

Running Logistic regression in Sklearn

```
>>> from sklearn.linear_model import LogisticRegression
>>> model = LogisticRegression(
...   fit_intercept=True,
...   penalty=None,
... )
>>> model.fit(X_train, y_train)
>>> y_pred = model.predict(X_test)

>>> y_pred
array([0, 0, 1, ..., 0, 0, 0])

>>> model.score(X_test, y_test)
0.8320413436692506
```

- Use the `LogisticRegression` classifier from `sklearn` to create the model object
- Fit the model to the *training* set to estimate the parameters
- Use the `predict()` method to generate predictions
- Use the `score()` method on the *test* set to assess model quality

Accuracy Score

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\{y_i = \hat{y}_i\}$$

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
0.8320413436692506
```

- Measures the share of correctly predicted data points
- Advantage: Just one number
- Disadvantage: Might not be what you care about

layout: center

Accuracy with imbalanced data

- **Imbalanced data**: Some outcomes occur more frequent than others in the data
- Example: Predicting whether someone has a PhD in a classroom with 49 students and one professor
- Models can "cheat" by predicting majority outcome
- Accuracy would be 98 % but model did not learn anything
- Will need other scores to discover such problems

layout: center

The Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> import pandas as pd
>>> confusion = confusion_matrix(
...     y_test, y_pred, normalize="true"
... )

>>> labels = ["Below 70th", "Above 70th"]
>>> confusion = pd.DataFrame(
...   confusion,
...   columns=labels,
...   index=labels,
... )
>>> confusion
            Below 70th    Above 70th
Below 70th    0.931839      0.068161
Above 70th    0.399678      0.600322
```

- Rows are the true labels
- Columns are the predictions
- Rows sum to 1
- Diagonal elements show the share of correctly classified examples in each category
- Bottom right element: 40 % of observations with true label `"Above 70th"` got misclassified as `"Below 70th"`

layout: center

A note on the different scores

- Think of scores as different summaries of the confusion matrix
- Scores are first calculated for each category
- An aggregation strategy converts them into one score for the entire model
- Only some aggregation strategies work well for imbalanced data

layout: center

Precision Score

```
>>> from sklearn.metrics import precision_score
>>> precision_score(y_test, y_pred, average=None)
array([0.84407702, 0.79137199])
```

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k}$$

- For each class, measures the probability of the predicted positive case actually being truly positive (TP_k)
- FP_k (*false positive*) is the total number of examples classified as label k, but actually from a different class
- Preferred metric when false positive predictions are costly

layout: center

Recall Score

```
>>> from sklearn.metrics import recall_score
>>> recall_score(y_test, y_pred, average=None)
array([0.93183919, 0.60032189])
```

$$\mathrm{Recall}_k = \frac{TP_k}{TP_k + FN_k}$$

- For each class, measures the model's ability to find the positive cases
- FN_k (*false negative*) is the total number of examples actually from class k that were not predicted by the model as such

layout: center

F_1 Score

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_test, y_pred, average=None)
array([0.88578959, 0.68273337])
```

$$F_{1,k} = 2\frac{\text{Precision}_k \cdot \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}$$

- F_1 score is the *harmonic mean* of precision and recall
- For a given class, there is a trade-off in precision and recall
- F_1 balances the two motives
- Good choice if you have no reason to penalize one error more than the another

layout: center

Summary

- Accuracy: share of correct predictions
- Precision: True positives over positive predictions
- Recall: True positives over actual positives
- F_1: Harmonic mean of Precision and Recall

layout: center

Scores with imbalanced data

- Same example with 49 students and one professor
- Models can "cheat" by predicting majority outcome
  - Accurracy: 98 %
  - Precision: 98 % for majority, 0 for minority class
  - Recall: 100 % for majority, 0 for minority class
  - F_1: 99 % for majority, 0 for minority class
- If we just look at scores for majority, we don’t see problems
- Unfortunately that is what you get by default in `sklearn` in the binary case

layout: center

Aggregation Strategies

```
>>> precision_score(
...     y_test,
...     y_pred,
...     average="macro"
... )
0.8177245070078974

>>> precision_score(
...     y_test,
...     y_pred,
...     average="weighted"
... )
0.8282110365957613
```

- `"macro"` strategy takes the simple mean over scores for each class:

  $$\text{Precision}^{(\text{macro})} = \frac{1}{K} \sum_{k=1}^{K} \text{Precision}_k$$

- `"weighted"` strategy weights the scores by the relative sizes of the classes

  $$\text{Precision}^{(\text{weighted})} = \sum_{k=1}^{K} w_k \cdot \text{Precision}_k$$

- Aggregate F_1 score is the harmonic mean of the aggregate precision and recall

layout: center

Sklearn’s Classification Report

```
>>> from sklearn.metrics import classification_report
>>> report = classification_report(
...     y_test,
...     y_pred,
...     target_names=["Below 70th", "Above 70th"],
... )
... print(report)
              precision    recall  f1-score   support

  Below 70th       0.84      0.93      0.89      4328
  Above 70th       0.79      0.60      0.68      1864

    accuracy                           0.83      6192
   macro avg       0.82      0.77      0.78      6192
weighted avg       0.83      0.83      0.82      6192
```

layout: center

Example: Report with imbalanced data

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        49
           1       0.00      0.00      0.00         1

    accuracy                           0.98        50
   macro avg       0.49      0.50      0.49        50
weighted avg       0.96      0.98      0.97        50
```