Classification
scikit-learn

Artificial Intelligence @ Allegheny College

Janyl Jumadinova

September 29–October 1, 2021

scikit-learn

- Popular Python machine learning library
- Designed to be a well documented and approachable for non-specialist
- Built on top of NumPy and SciPy
- `scikit-learn` can be easily installed with pip or conda

```
pip install scikit-learn
conda install scikit-learn
```

Data representation in scikit-learn

- Training dataset is described by a pair of matrices, one for the input data and one for the output.
- Most commonly used data formats are a NumPy `ndarray` or a Pandas `DataFrame / Series`.

Data representation in scikit-learn

- Training dataset is described by a pair of matrices, one for the input data and one for the output.
- Most commonly used data formats are a NumPy `ndarray` or a Pandas `DataFrame / Series`.
- Each row of these matrices corresponds to one sample of the dataset.
- Each column represents a quantitative piece of information that is used to describe each sample (called "features").

Data representation in scikit-learn

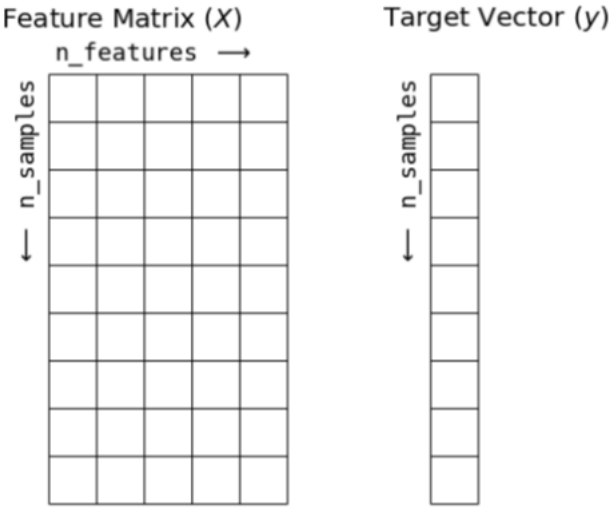


image credit: James Bourbeau

Features in scikit-learn

`feature` **Module**

`https://scikit-image.org/docs/dev/api/skimage.feature.html`

Local Binary Pattern Feature Extraction

Introduced by Ojala et. al in "Multiresolution Gray Scale and Rotation Invariant Texture Classificationwith Local Binary Patterns"

1. Check whether the points surrounding the central point are greater than or less than the central point $\rightarrow$ get LBP codes (stored as array).
2. Calculate a histogram of LBP codes as a feature vector.

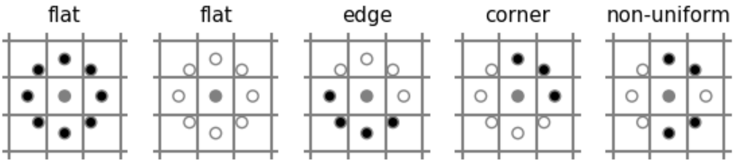


image credit:

https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html

Local Binary Pattern Feature Extraction

- Example: The histogram of the LBP outcome is used as a measure to classify textures.

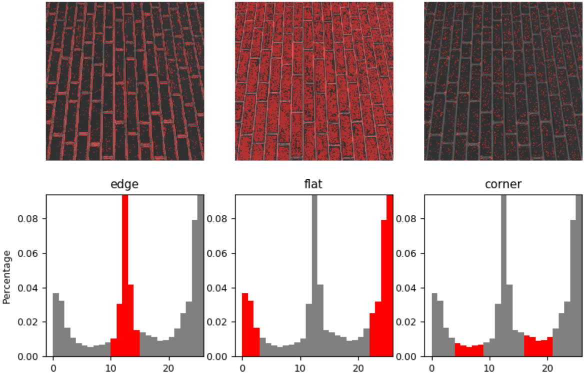


image credit:
https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html

Estimators in scikit-learn

- Algorithms are implemented as **estimator** classes in `scikit-learn`.
- Each estimator in scikit-learn is extensively documented (e.g. the KNeighborsClassifier documentation) with API documentation, user guides, and example usages.
- A **model** is an instance of one of these estimator classes

Training a model

fit then predict

```
# Fit the model
model.fit(X, y)

# Get model predictions
y_pred = model.predict(X)
```

Decision Tree in scikit-learn

```python
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=2)
clf.fit(X, y)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

image credit: James Bourbeau

Model performance metrics

- Many commonly used performance metrics are built into the metrics subpackage in scikit-learn.
- However, a user-defined scoring function can be created using the `sklearn.metrics.make_scorer` function.

```
# Classification metrics
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score, f1_score, log_loss)
# Regression metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
y_pred = [0, 2, 1, 3, 1]
y_true = [0, 1, 1, 3, 2]
```

```
accuracy_score(y_true, y_pred)
```

```
0.6
```

```
mean_squared_error(y_true, y_pred)
```

```
0.4
```

Separate training and testing sets

- scikit-learn has a convenient train_test_split function that randomly splits a dataset into a testing and training set.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=2)

print(f'X.shape = {X.shape}')
print(f'X_test.shape = {X_test.shape}')
print(f'X_train.shape = {X_train.shape}')
```

```
X.shape = (150, 2)
X_test.shape = (30, 2)
X_train.shape = (120, 2)
```

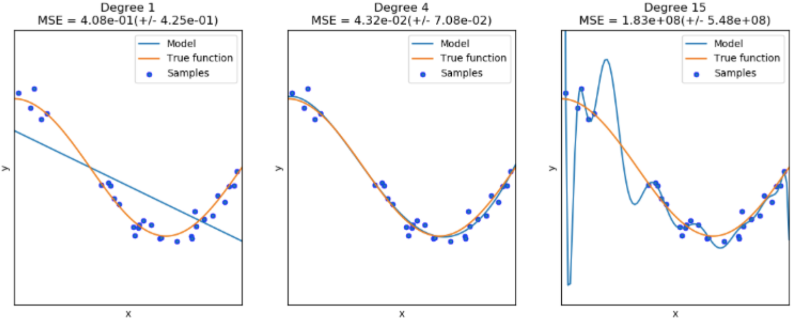
image credit: James Bourbeau

Model selection - hyperparameter optimization

- Model **hyperparameter** values (parameters whose values are set before the learning process begins) can be used to avoid under- and over-fitting.

Model selection - hyperparameter optimization

- Model **hyperparameter** values (parameters whose values are set before the learning process begins) can be used to avoid under- and over-fitting.
- **Under-fitting** - model isn't sufficiently complex enough to properly model the dataset at hand.
- **Over-fitting** - model is too complex and begins to learn the noise in the training dataset.



k-fold cross validation

- **Cross-validation** is a resampling procedure used to evaluate machine learning models on a limited data sample.
- It uses a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.
- The parameter **k** refers to the number of groups that a given data sample is to be split into.

k-fold cross validation

1. Shuffle the dataset randomly.
2. Split the dataset into k groups.
3. For each unique group:
   3.1. Take the group as a hold out or test data set.
   3.2. Take the remaining groups as a training data set.
   3.3. Fit a model on the training set and evaluate it on the test set.
   3.4. Retain the evaluation score and discard the model.
4. Summarize the skill of the model using the sample of model evaluation scores.

k-fold cross validation

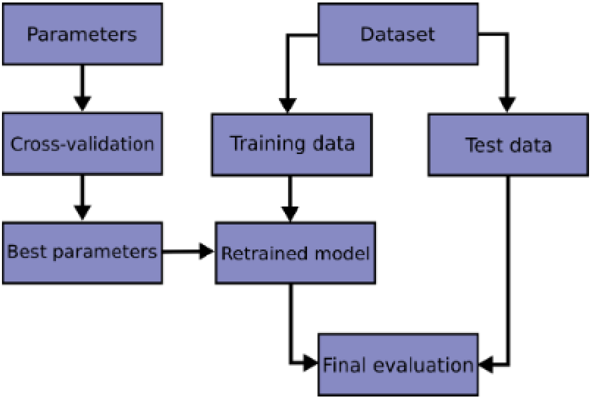


image credit:

`https://scikit-learn.org/stable/modules/cross_validation.html`

k-fold cross validation

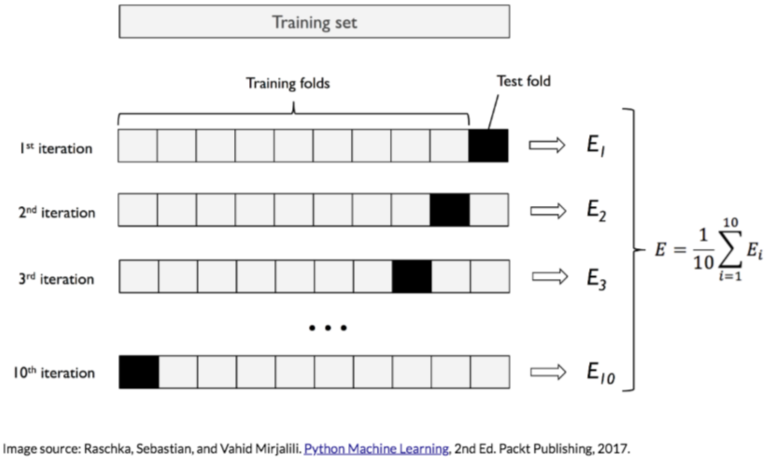


image credit:

James Bourbeau

Cross Validation in scikit-learn

```
from sklearn.model_selection import cross_validate

clf = DecisionTreeClassifier(max_depth=2)
scores = cross_validate(clf, X_train, y_train,
                        scoring='accuracy', cv=10,
                        return_train_score=True)

print(scores.keys())
test_scores = scores['test_score']
train_scores = scores['train_score']
print(test_scores)
print(train_scores)

print('\n10-fold CV scores:')
print(f'training score = {np.mean(train_scores)} +/- {np.std(train_scores)}')
print(f'validation score = {np.mean(test_scores)} +/- {np.std(test_scores)}')
```

```
dict_keys(['fit_time', 'score_time', 'test_score', 'train_score'])
[0.78571429 0.64285714 0.83333333 0.66666667 1.         0.91666667
 0.54545455 0.72727273 0.81818182 0.72727273]
[0.79245283 0.79245283 0.76851852 0.80555556 0.75       0.77777778
 0.79816514 0.79816514 0.78899083 0.79816514]

10-fold CV scores:
training score = 0.787024375076132 +/- 0.016054059411612778
validation score = 0.7663419913419914 +/- 0.12718955265834164
```

image credit:

James Bourbeau