EML4930/EML6934: Lecture 12

Scikit-Learn: Machine learning and **Regression**

Charles Jekel

November 16, 2017

More with Scikit-Learn

- Saving Python objects with Pickle
- Saving Scikit-Learn models with joblib
- Cross Validation
- Neural Network Example

Saving Python objects with Pickle

```
# This code fits a support vector classifier to the iris dataset
from sklearn import svm
from sklearn import datasets
clf = svm.SVC()
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf.fit(X, y)

# we can use Python's pickle to save the clf object
# pickle writes an object's memory state
import pickle
# save the clf object to 'my_svc_clf.p'
pickle.dump( clf, open('my_svc_clf.p', 'wb'))

# load clf object using
clf = pickle.load( open('my_svc_clf.p', 'rb'))
```

You should use cPickle in Python 2

In Python 2 you should use cPickle which is up to 1000 time faster than pickle. Python 3 automatically uses cPickle (if available).

```python
# importing from cPickle in Python 2
import cPickle as pickle
```

However this import will break Python 3 code

We can use Try: Except: to write a Python X

```
# this code will import pickle on both Python 2 and Python 3
try:
    # first let’s try importing
    import cPickle as pickle
    # if this returns an error, it isn’t shown
    # rather the code from except: is run
except:
    # this only runs if the try code had an error
    import pickle
```

Summary of pickle

- Pickle objects can be hacked
- Don't load a pickled object if you can't trust the source
- Malicous code can be executed when loading a Pickled objected
- You can use Pickle to save any Python object
- If you use Python 2, you should use cPickle as it's 1000 times faster!

Saving Scikit-Learn models with joblib

Scikit-Learn recommends using joblib over pickle to save models as it's more efficient with large numpy arrays.

```
from sklearn import svm
from sklearn import datasets
clf = svm.SVC()
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf.fit(X, y)

from sklearn.externals import joblib
# save the clf object using
joblib.dump(clf, 'filename.pkl')

# load the .pkl file to clf using
clf = joblib.load('filename.pkl')
```

Why do we want to be able to save and open models?

because some models may take a long time to train as we want to used a trained model to predict for the future

Scikit-learn has lots of models

- Naive Bayes
- Linear regression
- Support Vector Machines
- Decision Trees and Random Forests
- Gaussian process prediction

so which model do we use?

Cross Validation for model validation

- Cross Validation (CV) is a tool for model selection on the principle that data collection is expensive (we can't afford to obtain more data points)
- Used to estimate how accurately a predictive model will perform in practice
- There are various CV variations, I'm going to focus on k-fold CV

K-fold Cross Validation

1. Partition the data into k equal sized sets of samples
2. Train the model on $k - 1$ sample sets, validate the model (score) on the single remaining set
3. Iterate k times
4. Final K-fold CV metric is the average score from each iteration

4-fold CV visualized

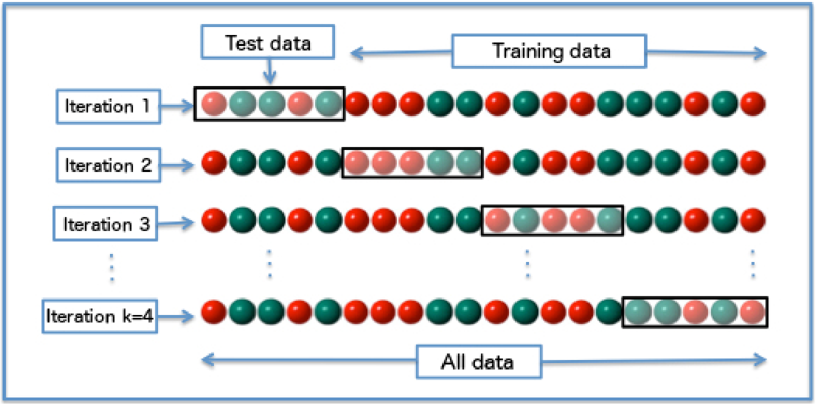

Figure 1: Visualization of a 4-fold CV. CC BY-SA 3.0 author Fabian Flöck

K-fold CV tips

- In practice k can be anywhere between 5-20
- There is no best k
- 10-fold CV is really common
- everyone developes their own preference
- Use for model validation
- Use to approximate the accuracy of the model

Scikit-learn includes a KFold CV iterator

Use KFold as a cross validation iterator.

```
from sklearn import svm
from sklearn import datasets
clf = svm.SVC()
iris = datasets.load_iris()
X, y = iris.data, iris.target

# import the cross validation iterator
from sklearn.model_selection import KFold
# initialize the KFold object for 5 fold CV
kf = KFold(n_splits=5)

for train, test in kf.split(X):
        # train are the indexes of the training set
        # test are the indexes of the test set
        X_test, X_train = X[test], X[train]
        y_test, y_train = y[test], y[train]
```

Train and validate the model within each loop of the iteration

```
score = []
for train, test in kf.split(X):
        # train are the indexes of the training set
        # test are the indexes of the test set
        X_test, X_train = X[test], X[train]
        y_test, y_train = y[test], y[train]

        # fit the model on the train set
        clf.fit(X_train,y_train)

        # score the model on the test set
        score.append(clf.score(X_test, y_test))

print('5 Fold CV avg score =', np.mean(score))
```

Scikit-Learn includes a function to do this in one line

If you have a bunch of models, and different pre-processing the KFold iterator may be more helpful. However if you just want to do a quick CV score you can use the cross_validate function.

```
# import the cross_validate function
from sklearn.model_selection import cross_validate

# get the 10 fold cross validation scores
scores = cross_validate(clf, X, y,
        cv=10, return_train_score=False)

# this creates a dictionary of the scores
print('scores =', scores)

print('10 Fold CV avg score =', np.mean(scores['test_score']))
```

Cross validation and Scikit-learn summary

- There are many model validation tools in Scikit-Learn
- K-fold cross validation is useful for model validation
- KFold is a cross validation model iterator
- Alternatively there is a cross_validate function to get the CV scores
- General 5-fold to 20-fold cross validations are used

scikit-learn comes with a few small standard datasets

Toy datasets for practice

- load_boston Load and return the boston house-prices dataset (regression).
- load_iris Load and return the iris dataset (classification).
- load_diabetes Load and return the diabetes dataset (regression).
- load_digits Load and return the digits dataset (classification).
- load_linnerud Load and return the linnerud dataset (multivariate regression).
- load_wine Load and return the wine dataset (classification).
- load_breast_cancer Load and return the breast cancer wisconsin dataset (classification).

Scaling for zero mean and unit variance

A lot of algorithms are sensitive to scaling (such as SVC) of the design features. Standard scalar mean centers you design features and scales for unit variance.

```
from sklearn.preprocessing import StandardScaler
import numpy as np
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
ss = StandardScaler()
# fit the pre-processing model to the training set
ss.fit(X_train)
# transform the training set
X_train = ss.transform(X_train)
print(X_train)
print('mean = ', np.mean(X_train,axis=0))
print('std = ', np.std(X_train,axis=0))
```

There are a number of example datasets in Documentation

`http://scikit-learn.org/stable/auto_examples/index.html`

Nueral Network on MNIST dataset

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
mnist = fetch_mldata("MNIST original")
# rescale the data, use the traditional train/test split
X, y = mnist.data, mnist.target


# transform X for unit scaling
ss = StandardScaler()
X = ss.fit_transform(X)

# create test train set
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Nueral Network on MNIST dataset

```
# set up my layers
layers = (256, 128, 64, 32, 16, 4)
# there will be two more layers than len(n)
# these are the number of nodes on hidden layers

# load MLP model
mlp = MLPClassifier(hidden_layer_sizes=layers, max_iter=1000, al
                solver='adam', verbose=10, tol=0, random_state=1
                learning_rate='adaptive')
mlp.fit(X_train,y_train)

# score the  model
print(mlp.score(X_test,y_test))

# this only scores about 0.968
# if you want to learn more, check out the TF tutorial
# https://www.tensorflow.org/get_started/mnist/beginners
```