ALBUKHARY INTERNATIONAL UNIVERSITY

Artificial Intelligence

LAB TWELVE

Compare Machine Learning Algorithms

It is important to compare the performance of multiple different machine learning algorithms consistently. In this chapter you will discover how you can create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. You can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. After completing this lesson you will know:

1. How to formulate an experiment to directly compare machine learning algorithms.
2. A reusable template for evaluating the performance of multiple algorithms on one dataset.
3. How to report and visualize the results when comparing algorithm performance.

Choose The Best Machine Learning Model

When you work on a machine learning project, you often end up with multiple good models to choose from. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. You need to be able to use these estimates to choose one or two best models from the suite of models that you have created. When you have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two algorithm to finalize. A way to do this is to use visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies. In the next section you will discover exactly how you can do that in Python with scikit-learn.

Compare Machine Learning Algorithms Consistently

The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data. You can achieve this by forcing each algorithm to be evaluated on a consistent test harness. In the example below six different classification algorithms are compared on a single dataset:

- Logistic Regression.
- Linear Discriminant Analysis.
- k-Nearest Neighbors.
- Classification and Regression Trees.
- Naive Bayes.
- Support Vector Machines.

The dataset is the Pima Indians onset of diabetes problem. The problem has two classes and eight numeric input variables of varying scales. The 10-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to

In [ ]:

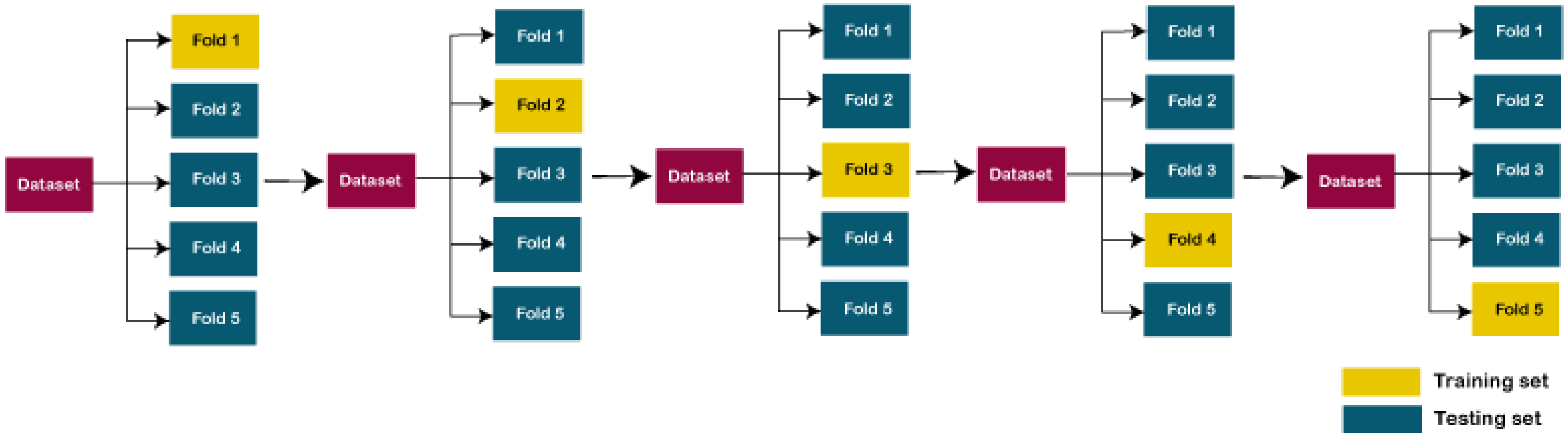
```python
# Compare Algorithms
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
# load dataset
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# prepare models
models = []
models.append(('LR', LogisticRegression(solver='lbfgs', max_iter=1000)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=7)
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    msg = "{}: {}; ({})".format(name, cv_results.mean(), cv_results.std())
    print(msg)

# boxplot algorithm comparison
fig = pyplot.figure()
```

```
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

K-fold Cross Validation

Cross validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts (e.g. k = 5 or k = 10). Each split of the data is called a fold. The algorithm is trained on k-1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross validation you end up with k different performance scores that you can summarize using a mean and a standard deviation.

Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1^{st} iteration, the first fold is reserved for test the model, and rest are used to train the model. On 2^{nd} iteration, the second fold is used to test the model, and rest are used to train the model. This process will continue until each fold is not used for the test fold. Consider the below diagram:



The result is a more reliable estimate of the performance of the algorithm on new data. It is more accurate because the algorithm is trained and evaluated multiple times on different data. The choice of k must allow the size of each test partition to be large enough to be a reasonable sample of the problem, whilst allowing enough repetitions of the train-test evaluation of the algorithm to provide a fair estimate of the algorithms performance on unseen data. For modest sized datasets in the thousands or tens of thousands of records, k values of 3,

```
In [ ]:
```