Cross Validation

Need for cross validation?

1. We wish to know how well a ML model is likely to perform in production.

2. Model’s performance in training is no guarantee production performance

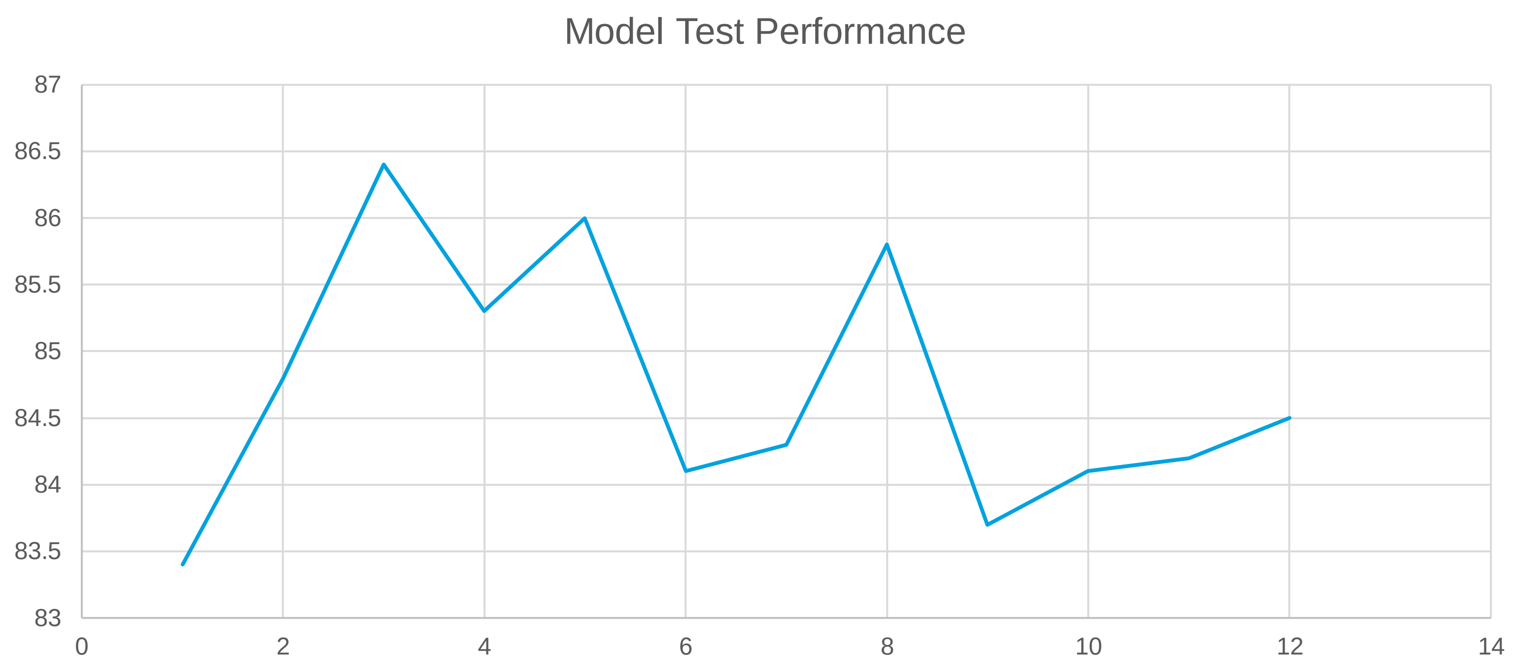
3. To estimate the model production score, hold a part of the sample data out of training phase. We call it test data which represents the universe

4. Usually the available data is not sufficient to split into training and test set and expect the two to represent the universe

5. Hence the model error on test data may not be good estimate of the model error in the universe

6. In the absence of large data sets, a number of techniques can be employed to estimate the model error in production

7. One of the techniques is cross validation



1. Car_MPG prediction accuracy trained on 70% and tested on 30%
2. For each run, we get different accuracy scores
3. Simple training/test a.k.a validation approach gives varying results with every run (random state not set)
4. Thus we cannot rely on one round of testing as we would have by chance got the split that gives max score first time
5. To get a more realistic estimate, we have to use more reliable techniques such as cross validation

What is cross validation?

1. Cross-validation is a technique to evaluate / validate a machine learning model and estimate its performance on unseen data

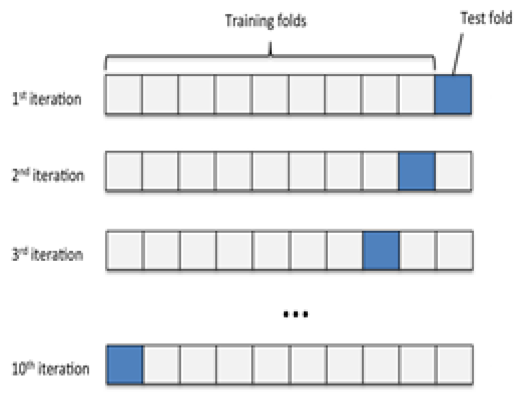
2. The techniques creates and validates given model multiple times

3. The number of times it does so, is dependent on the value selected by the user of the technique. Usually expressed as “K” which is always an integer

4. The sequence of steps used is iterated through as many times as K

5. The process begins by dividing the original data into K parts / folds using random function

Cross validation procedure



1. Shuffle the dataset randomly

2. Split the dataset into k folds

3. For each distinct fold:
   a. Keep the fold data separate / hold out data set
   b. Use the remaining folds as a single training data set
   c. Fit the model on the training set and evaluate it on the test set
   d. Retain the evaluation score and discard the model
   e. Loop back

4. The steps 3.a to 3.e will be executed K times

5. Summarize the scores and average it by dividing the sum by K.

6. Analyze the average score, the dispersion to assess the likely performance of the model in the unseen data (production data / universe)

greatlearning
Learning for Life

Implementing K Fold cross validation

Visual understanding (example based on scikitlearn guide)

a) from numpy import array
b) from sklearn.model_selection import KFold
c) data = array([10,20,30,40,50,60,70,80,90,100])
d) kfold = KFold(5, True)
e) for train, test in kfold.split(data):
f)   print('train: %s, test: %s' % (data[train], data[test]))

Training Data	Test Data
[10 20 30 40 50 60 80 90]	[70 100]
[10 40 50 60 70 80 90 100]	[20 30]
[10 20 30 40 50 70 90 100]	[60 80]
[10 20 30 50 60 70 80 100]	[40 90]
[20 30 40 60 70 80 90 100]	[10 50]

Note : We cannot have K > number of data points…. Why?

Ref: Kfold_introduction.ipynb

Configuring the K

1. K is an integral number. Minimum value of K has to be 2. There will be two iterations in this case

2. Max value of K can be the number of data points. This is also known as Leave One Out Cross Validation or LOOCV

3. Whatever the value of K chosen, the resulting training and test data should be representative of the unseen data as much as possible

4. There is not formula to decide the K but K = 10 is usually considered good

5. Too large a K, means less variance across the training sets thus limit the model differences across iterations

6. For a sample size (N) of n, and K = k, number of records (r) per fold = n/k.

Evaluating the model in an iteration

1. In each iteration, the model is trained on K -1 number of folds and evaluated on the left out fold.

2. The MSE or Mean Squared Error is thus calculated on the left out fold

3. Since the procedure is repeated K times, we will have K MSEs. Total up all the MSE and divide by K to get the overall expected MSE

CVk = (sum(MSEi) for I = 1 to K) / K

Some salient features of K-fold

1. Each record / data point in the sample data before creating the Kfolds, is assigned to a single fold and stays in that fold for the duration of the procedure.

2. This means that each data point is used once in hold-out set and K-1 times in training

3. When hyper parameters are to be tweaked, split the original data into two. Keep one part aside. Use the other to do the Kfold validation. Once the optimal hyperparameters are found, assess the model on the test data

4. Any data transformation done on the whole set outside the loop, may lead to data leakage and overfitting

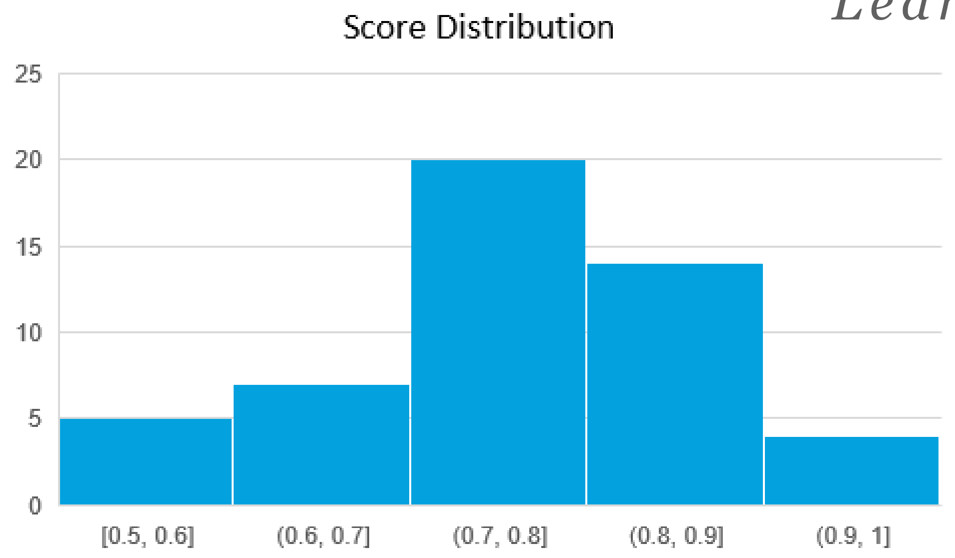
Implementing K Fold cross validation

K fold in Pima Indian Classification

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import numpy as np

filename = 'pima-indians-diabetes.data'
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]

num_folds = 50
seed = 7

kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Kfold_logistic.ipynb

Mean	0.770166667
Standard Error	0.015172553
Median	0.8
Mode	0.8
Standard Deviation	0.107286148
Sample Variance	0.011510317
Kurtosis	0.2246995
Skewness	-0.267638887
Range	0.5
Minimum	0.5
Maximum	1
Sum	38.50833335
Count	50
Confidence Level(95.0%)	0.030490386



1. Distribution of the scores on 50 iterations
2. Model accuracy is likely to be in 0.77 – 0.03 to 0.77 + 0.03 i.e. 0.74 – 0.80 at 95% confidence level

Leave One Out Cross validation (LOOCV) procedure

1. In this method, a single observation (x1, y1) is used for the validation set and the remaining (x2,y2)…. (xn, yn) make up the training set

2. The statistical model is fit on the n-1 training examples

3. The statistical model prediction yhat is made for the excluded observation using x1.

4. MSE1 = (yhat – y)^2 for the excluded point

5. The MSE is unbiased but is poor estimate because it is highly variable

6. We can repeat this by keeping every data point for test one at a time and using rest for training

Ref: LOOCV_Introduction.ipynb

Leave One Out Cross validation (LOOCV) procedure

7. Repeating this approach n times we get MSE1, MSE2,…. MSEn
8. CV(n) = sum(MSEi) for i=1 to n / n
9. LOOCV is a special case of Kfold validation with K = n

```
# scikit-learn k-fold cross-validation
from numpy import array
from sklearn.model_selection import LeaveOneOut
# data sample
data = array([10,20,30,40,50,60,70,80,90,100])
# prepare cross validation
loocv = LeaveOneOut()
# enumerate splits
for train, test in loocv.split(data):
		print('train: %s, test: %s' % (data[train], data[test]))
```

```
train: [ 20 30 40 50 60 70 80 90 100], test: [10]
train: [ 10 30 40 50 60 70 80 90 100], test: [20]
train: [ 10 20 40 50 60 70 80 90 100], test: [30]
train: [ 10 20 30 50 60 70 80 90 100], test: [40]
train: [ 10 20 30 40 60 70 80 90 100], test: [50]
train: [ 10 20 30 40 50 70 80 90 100], test: [60]
train: [ 10 20 30 40 50 60 80 90 100], test: [70]
train: [ 10 20 30 40 50 60 70 90 100], test: [80]
train: [ 10 20 30 40 50 60 70 80 100], test: [90]
train: [10 20 30 40 50 60 70 80 90], test: [100]
```

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
import numpy as np

# prepare cross validation
loocv = LeaveOneOut()
model = LogisticRegression()


filename = 'pima-indians-diabetes.data'
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]


results = cross_val_score(model, X, Y, cv=loocv)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 76.953% (42.113%)    (Compare the standard deviation with KFOLD which is much less)
```

Bias Variance trade-off and cross validation

1. Every machine learning model are impacted by bias, variance and random errors. These errors are analysed in the context of the test environment. Refer to the attached document

2. A simple validation approach consists of a training set and test set where training set is used to build the model and test set is used to validate the model

3. Suppose you create the training and test set in 50:50 ratio using random function multiple times and build and validate the model each time.

4. Since the model is created based on a subset of the data, it is likely to be impacted by bias error and the test errors are likely to vary across iterations

5. The simple validation approach is likely to suffer from both bias and variance errors and the degree of each type of error depends on the ratio of the split

Bias Variance tradeoff and cross validation

6. LOOCV, which gets 90% of the data and multiple iterations, is likely to commit less bias errors. However, since only 10% of the data is left for testing, the variance errors will be high

7. K-Fold cross validation, which lies between simple validation and LOOCV approach, can give more moderate error rates.

8. By selecting right K, we can minimize both the bias and variance errors. The overall error rate is likely to be the model performance in the production

Bootstrap Sampling

1. Also known as sampling with replacement, is a sampling technique used when the amount of data is limited

2. A random function is used to create the sample from the original dataset. That a record has already been picked earlier for the sample is immaterial.

3. Within a sample, there may be repeating records. Could be duplicates or more but two sample sets are unlikely to be 100% same.

4. The records not picked up in an iteration will serve the purpose of test data and test data will always have unique records

5. Suppose we have 10 data points in the dataset. We can create multiple sample sets each with 10 data points or less and corresponding test data.

6. The number of samples created maybe 10, more than 10 or less than 10

7. The more the samples we create from a small size data, more likely we will have samples that are very similar in terms of the data points

Bootstrap Sampling

```
from sklearn.utils import resample
import numpy as np

# load dataset
data = [10,20,30,40,50,60,70,80,90,100]   # original data with 10 data points

# configure bootstrap
n_iterations = 50              # Number of bootstrap samples to create = 50
n_size = int(len(data) * 1)    # picking only 50 % of the given data in every bootstrap sample

# run bootstrap
stats = list()
for i in range(n_iterations):
        # prepare train and test sets
        train = resample(data, n_samples=n_size)  # Sampling with replacement
        test = np.array([x for x in data if x not in train])  # picking rest of the data not considered in sample
        print("Train_data ->", train, " " , "Test_data ->", test)
```

Bootstrap sample -

1. Train_data -> [30, 30, 30, 90, 20, 80, 30, 80, 40, 10] Test_data -> [50 60 70 100]
2. Train_data -> [50, 80, 20, 20, 40, 40, 50, 100, 70, 70] Test_data -> [10 30 60 90]
3. Train_data -> [10, 80, 40, 80, 90, 100, 30, 80, 90, 20] Test_data -> [50 60 70]
4. Train_data -> [30, 30, 40, 70, 70, 50, 100, 100, 70, 20] Test_data -> [10 60 80 90]
5. Train_data -> [90, 30, 100, 40, 10, 30, 30, 50, 10, 30] Test_data -> [20 60 70 80]
6. Train_data -> [30, 40, 30, 20, 80, 20, 10, 30, 50, 40] Test_data -> [60 70 90 100]
7. Train_data -> [70, 30, 60, 30, 80, 100, 40, 20, 70, 70] Test_data -> [10 50 90]
8. Train_data -> [100, 30, 70, 100, 90, 90, 10, 60, 60, 70] Test_data -> [20 40 50 80]
9. Train_data -> [40, 40, 30, 90, 90, 30, 90, 10, 60, 100] Test_data -> [20 50 70 80]
10. Train_data -> [10, 90, 70, 70, 30, 20, 70, 70, 90, 40] Test_data -> [50 60 80 100]
11. Train_data -> [100, 40, 100, 90, 100, 50, 30, 50, 20, 40] Test_data -> [10 60 70 80]
12. Train_data -> [50, 20, 10, 50, 50, 20, 60, 20, 40, 100] Test_data -> [30 70 80 90]
13. Train_data -> [10, 20, 10, 40, 60, 30, 20, 30, 80, 80] Test_data -> [50 70 90 100]
14. Train_data -> [50, 90, 50, 60, 50, 90, 40, 30, 40, 50] Test_data -> [10 20 70 80 100]
15. Train_data -> [100, 70, 100, 70, 100, 70, 90, 20, 20, 60] Test_data -> [10 30 40 50 80]
16. Train_data -> [80, 30, 100, 60, 40, 20, 100, 70, 20, 60] Test_data -> [10 50 90]
17. Train_data -> [20, 100, 50, 70, 50, 10, 50, 60, 30, 70] Test_data -> [40 80 90]
18. Train_data -> [10, 60, 90, 40, 50, 100, 50, 50, 20, 80] Test_data -> [30 70]
19. Train_data -> [20, 30, 80, 10, 100, 80, 60, 90, 50, 80] Test_data -> [40 70]
20. Train_data -> [10, 20, 70, 10, 80, 60, 50, 20, 20, 10] Test_data -> [30 40 90 100]
21. Train_data -> [90, 70, 50, 100, 20, 60, 60, 90, 60, 70] Test_data -> [10 30 40 80]
22. Train_data -> [80, 20, 100, 10, 80, 30, 100, 20, 60, 100] Test_data -> [40 50 70 90]
23. Train_data -> [10, 90, 20, 90, 50, 80, 30, 100, 10, 80] Test_data -> [40 60 70]
24. Train_data -> [100, 30, 30, 70, 90, 30, 30, 90, 100, 10] Test_data -> [20 40 50 60 80]
25. Train_data -> [20, 10, 40, 20, 20, 40, 20, 90, 100, 50] Test_data -> [30 60 70 80]
26. Train_data -> [20, 100, 50, 60, 80, 70, 90, 20, 90, 40] Test_data -> [10 30]
27. Train_data -> [70, 20, 100, 20, 40, 60, 30, 80, 80, 70] Test_data -> [10 50 90]

Bootstrap sample (contd…) -

28. Train_data -> [70, 50, 80, 60, 100, 60, 40, 80, 70, 100]   Test_data -> [10 20 30 90]
29. Train_data -> [40, 90, 80, 20, 10, 70, 10, 80, 90, 60]   Test_data -> [30 50 100]
30. Train_data -> [50, 80, 90, 90, 80, 80, 80, 20, 30, 90]   Test_data -> [10 40 60 70 100]
31. Train_data -> [90, 40, 40, 80, 20, 80, 90, 30, 50, 90]   Test_data -> [10 60 70 100]
32. Train_data -> [40, 50, 100, 100, 70, 60, 40, 100, 50, 10]   Test_data -> [20 30 80 90]
33. Train_data -> [30, 20, 30, 70, 20, 20, 30, 80, 40, 70]   Test_data -> [10 50 60 90 100]
34. Train_data -> [10, 10, 70, 50, 60, 40, 70, 70, 100, 10]   Test_data -> [20 30 80 90]
35. Train_data -> [10, 40, 50, 100, 30, 100, 20, 10, 80, 70]   Test_data -> [60 90]
36. Train_data -> [90, 30, 10, 70, 50, 40, 30, 100, 20, 40]   Test_data -> [60 80]
37. Train_data -> [40, 50, 90, 100, 30, 100, 90, 30, 50, 70]   Test_data -> [10 20 60 80]
38. Train_data -> [30, 10, 20, 70, 60, 90, 90, 30, 70, 60]   Test_data -> [40 50 80 100]
39. Train_data -> [100, 60, 90, 20, 100, 70, 20, 50, 70, 100]   Test_data -> [10 30 40 80]
40. Train_data -> [50, 90, 30, 60, 40, 80, 20, 10, 40, 90]   Test_data -> [70 100]
41. Train_data -> [10, 30, 20, 10, 80, 60, 20, 40, 20, 70]   Test_data -> [50 90 100]
42. Train_data -> [90, 100, 60, 80, 10, 90, 20, 30, 30, 30]   Test_data -> [40 50 70]
43. Train_data -> [60, 70, 70, 100, 20, 30, 20, 50, 60, 70]   Test_data -> [10 40 80 90]
44. Train_data -> [20, 10, 50, 30, 90, 50, 100, 80, 40, 100]   Test_data -> [60 70]
45. Train_data -> [10, 30, 10, 40, 80, 60, 20, 40, 60, 30]   Test_data -> [50 70 90 100]
46. Train_data -> [60, 100, 70, 70, 70, 20, 30, 90, 90, 20]   Test_data -> [10 40 50 80]
47. Train_data -> [40, 80, 80, 20, 80, 90, 50, 30, 30, 80]   Test_data -> [10 60 70 100]
48. Train_data -> [10, 50, 60, 70, 100, 60, 30, 80, 100, 70]   Test_data -> [20 40 90]
49. Train_data -> [80, 20, 40, 100, 10, 90, 50, 40, 90, 20]   Test_data -> [30 60 70]
50. Train_data -> [10, 70, 60, 50, 50, 100, 40, 50, 80, 50]   Test_data -> [20 30 90]

Bootstrap Sampling

1. With the bootstrap samples available, we can create models on the training and test and average out the scores over all the runs. Refer to the attachment

2. When we create and test our model on bootstrapped data set, each iteration will give a performance score

3. When we increase the number of iterations to a large number and plot the frequency curve for the performance scores, one will notice the performance scores tend to follow normal distribution.

4. For very large number of iterations, the distribution becomes almost normal. This is known as the Central Limit Theorem.

5. Central Limit Theorem states “the sampling distribution of the sample means approaches a normal distribution as the sample size gets larger — *no matter what the shape of the population distribution*.

```python
from pandas import read_csv
from sklearn.utils import resample
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from matplotlib import pyplot
import numpy as np

# load dataset
data = read_csv('pima-indians-diabetes.data', header=None)
values = data.values

# configure bootstrap
n_iterations = 1000              # Number of bootstrap samples to create
n_size = int(len(data) * 0.50)    # picking only 50 % of the given data in every bootstrap sample

# run bootstrap
stats = list()
for i in range(n_iterations):
        # prepare train and test sets
        train = resample(values, n_samples=n_size)  # Sampling with replacement
        test = np.array([x for x in values if x.tolist() not in train.tolist()])  # picking rest of the data not considered in sample
    # fit model
        model = DecisionTreeClassifier()
        model.fit(train[:,:-1], train[:,-1])
    # evaluate model
        predictions = model.predict(test[:,:-1])
        score = accuracy_score(test[:,-1], predictions)    # caution, overall accuracy score can mislead when classes are imbalanced
        print(score)
        stats.append(score)
```

Bootstrapping_Confidence_Level.ipy

Model performance measures

1. We never give point estimates for model's performance in production. We always give range estimates. For e.g. model accuracy is likely to be in the range of 80% - 95%

2. Range estimates indicate lack of surety. Which means less than 100% sure. Hence, range estimates need to be backed up by confidence level. How confident are we in the range.

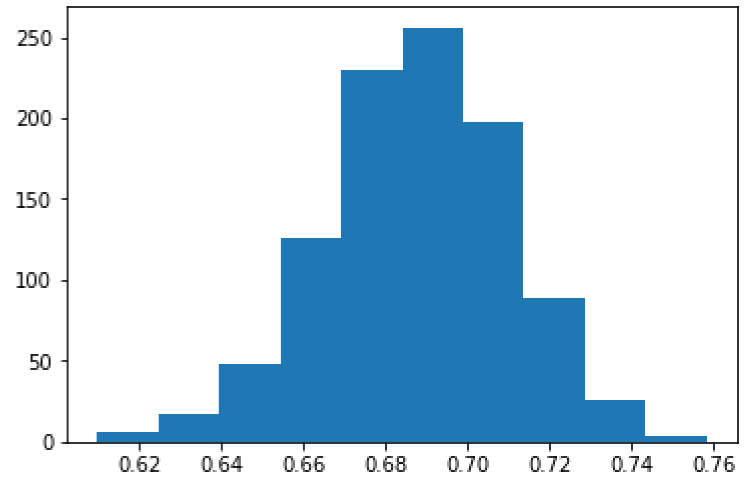
3. Larger the range more confident we are. But range cannot be too large. The model becomes unreliable!

4. The general practice is to quote the range at 95% confidence level. If you are working in life critical projects, one needs to be 99.99999% confident

5. The range can be estimated through K-Fold validation or Bootstrap sampling

Model Accuracy estimates using BootStrapping

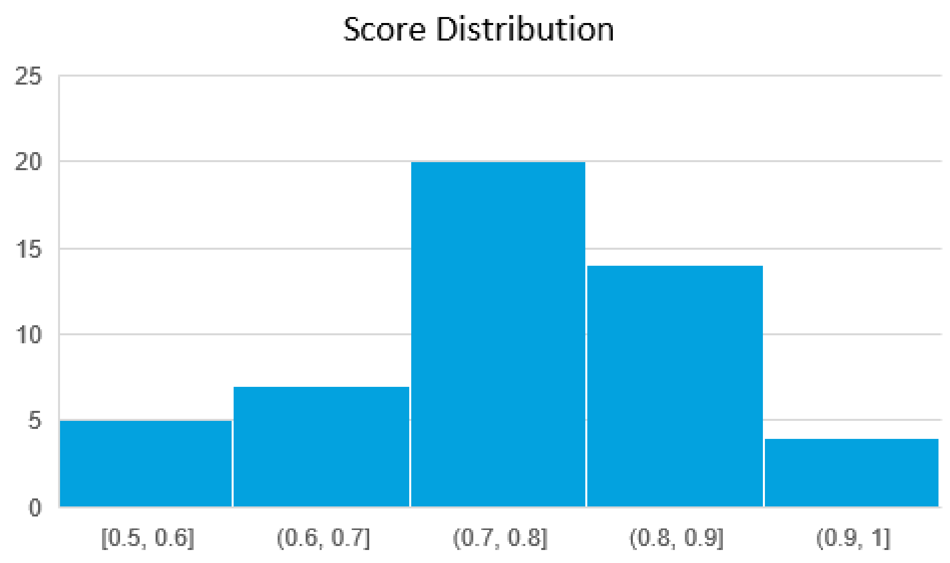
```
# plot scores
pyplot.hist(stats)
pyplot.show()
# confidence intervals
alpha = 0.95                          # for 95% confidence
p = ((1.0-alpha)/2.0) * 100          # tail regions on right and left .25 on each side indicated by P value (border)
lower = max(0.0, np.percentile(stats, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, np.percentile(stats, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```



```
95.0 confidence interval 64.2% and 73.0%
```

Model Accuracy estimates using Kfold Validation

Mean	0.770166667
Standard Error	0.015172553
Median	0.8
Mode	0.8
Standard Deviation	0.107286148
Sample Variance	0.011510317
Kurtosis	0.2246995
Skewness	-0.267638887
Range	0.5
Minimum	0.5
Maximum	1
Sum	38.50833335
Count	50
Confidence Level(95.0%)	0.030490386



1. Model accuracy is likely to be in the range of 0.74 – 0.80 at 95% confidence level