B. Prabhakar

========================

201505618

========================

Assignment I

confusion matrix,
mean,
Standard Deviation
and
accuracy.

1.a) Iris Data: Random subsampling with 1NN

```
=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==
Trial-1:
Confusion Matrix
+-----------------+-----------------+----------------+-------------+
|                 | Iris-versicolor | Iris-virginica | Iris-setosa |
+=================+=================+================+=============+
| Iris-versicolor | 22              | 0              | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-virginica  | 2               | 20             | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-setosa     | 0               | 0              | 31          |
+-----------------+-----------------+----------------+-------------+
Accuracy: 97.3333333333

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==
Trial-2: -
Confusion Matrix
+-----------------+-----------------+----------------+-------------+
|                 | Iris-versicolor | Iris-virginica | Iris-setosa |
+=================+=================+================+=============+
| Iris-versicolor | 23              | 0              | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-virginica  | 2               | 22             | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-setosa     | 0               | 0              | 28          |
+-----------------+-----------------+----------------+-------------+
Accuracy: 97.3333333333

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==
Trial-3:
Confusion Matrix
+-----------------+-----------------+----------------+-------------+
|                 | Iris-versicolor | Iris-virginica | Iris-setosa |
+=================+=================+================+=============+
| Iris-versicolor | 20              | 0              | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-virginica  | 6               | 23             | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-setosa     | 0               | 0              | 26          |
+-----------------+-----------------+----------------+-------------+
Accuracy: 92.0
```

=*==

Trial-4:

Confusion Matrix

	Iris-versicolor	Iris-virginica	Iris-setosa
Iris-versicolor	24	0	0
Iris-virginica	5	24	0
Iris-setosa	0	0	22

Accuracy: 93.3333333333

=*==

Trial-5:

Confusion Matrix

	Iris-versicolor	Iris-virginica	Iris-setosa
Iris-versicolor	26	0	0
Iris-virginica	7	24	0
Iris-setosa	0	0	18

Accuracy: 90.6666666667

=*==

Trial-6:

Confusion Matrix

	Iris-versicolor	Iris-virginica	Iris-setosa
Iris-versicolor	22	3	0
Iris-virginica	2	25	0
Iris-setosa	0	0	23

Accuracy: 93.3333333333

=*==

Trial-7:

Confusion Matrix

	Iris-versicolor	Iris-virginica	Iris-setosa
Iris-versicolor	25	1	0
Iris-virginica	2	22	0
Iris-setosa	0	0	25

Accuracy: 96.0

=*==

Trial-8:

Confusion Matrix

	Iris-versicolor	Iris-virginica	Iris-setosa
Iris-versicolor	24	1	0
Iris-virginica	2	22	0
Iris-setosa	0	0	26

Accuracy: 96.0

=*==

Trial-9:

Confusion Matrix

	Iris-versicolor	Iris-virginica	Iris-setosa
Iris-versicolor	21	1	0
Iris-virginica	2	23	0
Iris-setosa	0	0	28

Accuracy: 96.0

```
=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==
Trial-10:
Confusion Matrix
+-----------------+-----------------+----------------+-------------+
|                 | Iris-versicolor | Iris-virginica | Iris-setosa |
+=================+=================+================+=============+
| Iris-versicolor | 23              | 1              | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-virginica  | 1               | 20             | 0           |
+-----------------+-----------------+----------------+-------------+
| Iris-setosa     | 0               | 0              | 30          |
+-----------------+-----------------+----------------+-------------+
Accuracy: 97.3333333333

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==

Overall Mean and Standard Deviation(SD)
{'../../KNN/iris.data': ('Mean: 94.9333333333', 'SD: 5.79043195472')}
```

1.b) Iris Data: 3NN with Random subsampling

=*=

Trial-1:

Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	19	0	2
Iris-setosa	0	31	0
Iris-virginica	0	0	23

Accuracy: 97.3333333333

=*=

Trial-2:

Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	23	0	5
Iris-setosa	0	27	0
Iris-virginica	0	0	20

Accuracy: 93.3333333333

=*=

Trial-3:

Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	22	0	3
Iris-setosa	0	26	0
Iris-virginica	0	0	24

Accuracy: 96.0

=*=
Trial-4:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	24	0	2
Iris-setosa	0	26	0
Iris-virginica	0	0	23

Accuracy: 97.3333333333

=*=
Trial-5:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	23	0	3
Iris-setosa	0	20	0
Iris-virginica	1	0	28

Accuracy: 94.6666666667

=*=
Trial-6:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	21	0	5
Iris-setosa	0	26	0
Iris-virginica	0	0	23

Accuracy: 93.3333333333

=*=
Trial-7:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	22	0	5
Iris-setosa	0	23	0
Iris-virginica	0	0	25

Accuracy: 93.3333333333

=*=
Trial-8:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	22	0	3
Iris-setosa	0	28	0
Iris-virginica	0	0	22

Accuracy: 96.0

=*=
Trial-9:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	25	0	2
Iris-setosa	0	25	0
Iris-virginica	0	0	23

Accuracy: 97.3333333333

=*=
Trial-10:
Confusion Matrix

	Iris-versicolor	Iris-setosa	Iris-virginica
Iris-versicolor	21	0	2
Iris-setosa	0	24	0
Iris-virginica	0	0	28

Accuracy: 97.3333333333

=*=
Overall Mean and Standard Deviation(SD)
{'../../KNN/iris.data': ('Mean: 95.6', 'SD: 5.83927088295')}

2.a) Wine Data: 1NN with Random subsampling

=*=

Trial-1:

Confusion Matrix

	2	3	1
2	28	10	1
3	8	13	2
1	0	2	25

Accuracy: 74.1573033708

=*=

Trial-2:

Confusion Matrix

	2	3	1
2	22	9	3
3	8	13	3
1	1	3	27

Accuracy: 69.6629213483

=*=

Trial-3:

Confusion Matrix

	2	3	1
2	28	6	5
3	7	12	5
1	0	1	25

Accuracy: 73.0337078652

=*=

Trial-4:

Confusion Matrix

	2	3	1
2	23	6	3

3	10	14	2
1	0	1	30

Accuracy: 75.2808988764

=*=
Trial-5:
Confusion Matrix

	2	3	1
2	20	9	2
3	12	17	1
1	1	4	23

Accuracy: 67.4157303371

=*=
Trial-6:
Confusion Matrix

	2	3	1
2	20	10	2
3	7	17	0
1	4	3	26

Accuracy: 70.7865168539

=*=
Trial-7:
Confusion Matrix

	2	3	1
2	31	12	0
3	12	9	0
1	1	4	20

Accuracy: 67.4157303371

=*=
Trial-8:
Confusion Matrix

```
+---+----+----+----+
|   | 2  | 3  | 1  |
+===+====+====+====+
| 2 | 19 | 23 | 0  |
+---+----+----+----+
| 3 | 5  | 16 | 2  |
+---+----+----+----+
| 1 | 1  | 2  | 21 |
+---+----+----+----+
Accuracy: 62.9213483146

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-9:
Confusion Matrix
+---+----+----+----+
|   | 2  | 3  | 1  |
+===+====+====+====+
| 2 | 33 | 5  | 3  |
+---+----+----+----+
| 3 | 9  | 12 | 2  |
+---+----+----+----+
| 1 | 0  | 3  | 22 |
+---+----+----+----+
Accuracy: 75.2808988764

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-10:
Confusion Matrix
+---+----+----+----+
|   | 2  | 3  | 1  |
+===+====+====+====+
| 2 | 31 | 5  | 3  |
+---+----+----+----+
| 3 | 9  | 13 | 0  |
+---+----+----+----+
| 1 | 0  | 2  | 26 |
+---+----+----+----+
Accuracy: 78.6516853933

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Overall Mean and Standard Deviation(SD)
{'../../KNN/wine.data': ('Mean: 71.4606741573', 'SD: 4.33448137868')}
```

2.b) Wine Data: 3NN with Random subsampling

```
=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-1:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 24 | 0  | 3  |
+---+----+----+----+
| 2 | 7  | 16 | 14 |
+---+----+----+----+
| 3 | 6  | 1  | 18 |
+---+----+----+----+
Accuracy: 65.1685393258

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-2:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 29 | 0  | 1  |
+---+----+----+----+
| 2 | 4  | 19 | 13 |
+---+----+----+----+
| 3 | 7  | 2  | 14 |
+---+----+----+----+
Accuracy: 69.6629213483

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-3:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 29 | 0  | 3  |
+---+----+----+----+
| 2 | 1  | 13 | 16 |
+---+----+----+----+
| 3 | 3  | 5  | 19 |
+---+----+----+----+
Accuracy: 68.5393258427

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-4:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 35 | 0  | 0  |
+---+----+----+----+
```

```
| 2 | 3  | 18 | 7  |
+---+----+----+----+
| 3 | 9  | 4  | 13 |
+---+----+----+----+
Accuracy: 74.1573033708

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-5:
Confusion Matrix
+---+----+---+----+
|   | 1  | 2 | 3  |
+===+====+===+====+
| 1 | 28 | 0 | 1  |
+---+----+---+----+
| 2 | 5  | 8 | 29 |
+---+----+---+----+
| 3 | 6  | 0 | 12 |
+---+----+---+----+
Accuracy: 53.9325842697

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-6:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 30 | 0  | 2  |
+---+----+----+----+
| 2 | 3  | 15 | 18 |
+---+----+----+----+
| 3 | 2  | 1  | 18 |
+---+----+----+----+
Accuracy: 70.7865168539

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-7:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 33 | 0  | 1  |
+---+----+----+----+
| 2 | 4  | 16 | 14 |
+---+----+----+----+
| 3 | 6  | 1  | 14 |
+---+----+----+----+
Accuracy: 70.7865168539

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-8:
Confusion Matrix
+---+----+----+----+
```

```
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 25 | 0  | 1  |
+---+----+----+----+
| 2 | 5  | 18 | 14 |
+---+----+----+----+
| 3 | 4  | 4  | 18 |
+---+----+----+----+
Accuracy: 68.5393258427

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-9:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 22 | 0  | 2  |
+---+----+----+----+
| 2 | 2  | 19 | 13 |
+---+----+----+----+
| 3 | 7  | 9  | 15 |
+---+----+----+----+
Accuracy: 62.9213483146

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-10:
Confusion Matrix
+---+----+----+----+
|   | 1  | 2  | 3  |
+===+====+====+====+
| 1 | 28 | 0  | 1  |
+---+----+----+----+
| 2 | 2  | 17 | 16 |
+---+----+----+----+
| 3 | 4  | 0  | 21 |
+---+----+----+----+
Accuracy: 74.1573033708

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Overall Mean and Standard Deviation(SD)
{'../../KNN/wine.data': ('Mean: 67.8651685393', 'SD: 4.10810856604')}
```

3.a) Tic-Tac-Toe End game: 1NN with Random subsampling

```
=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-1:
Confusion Matrix
+----------+----------+----------+
|          | negative | positive |
+==========+==========+==========+
| negative | 122      | 52       |
+----------+----------+----------+
| positive | 72       | 233      |
+----------+----------+----------+
Accuracy: 74.1127348643

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-2:
Confusion Matrix
+----------+----------+----------+
|          | negative | positive |
+==========+==========+==========+
| negative | 122      | 42       |
+----------+----------+----------+
| positive | 64       | 251      |
+----------+----------+----------+
Accuracy: 77.8705636743

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-3:
Confusion Matrix
+----------+----------+----------+
|          | negative | positive |
+==========+==========+==========+
| negative | 114      | 57       |
+----------+----------+----------+
| positive | 74       | 234      |
+----------+----------+----------+
Accuracy: 72.6513569937

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-4:
Confusion Matrix
+----------+----------+----------+
|          | negative | positive |
+==========+==========+==========+
| negative | 128      | 43       |
+----------+----------+----------+
| positive | 61       | 247      |
+----------+----------+----------+
Accuracy: 78.2881002088

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
```

Trial-5:
Confusion Matrix

	negative	positive
negative	122	35
positive	96	226

Accuracy: 72.6513569937

=*=
Trial-6:
Confusion Matrix

	negative	positive
negative	124	34
positive	81	240

Accuracy: 75.9916492693

=*=
Trial-7:
Confusion Matrix

	negative	positive
negative	112	48
positive	72	247

Accuracy: 74.9478079332

=*=
Trial-8:
Confusion Matrix

	negative	positive
negative	115	49
positive	70	245

Accuracy: 75.1565762004

=*=
Trial-9:
Confusion Matrix

	negative	positive

```
+==========+==========+==========+
| negative | 137      | 39       |
+----------+----------+----------+
| positive | 71       | 232      |
+----------+----------+----------+
Accuracy: 77.0354906054

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-10:
Confusion Matrix
+----------+----------+----------+
|          | negative | positive |
+==========+==========+==========+
| negative | 103      | 53       |
+----------+----------+----------+
| positive | 65       | 258      |
+----------+----------+----------+
Accuracy: 75.3653444676

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Overall Mean and Standard Deviation(SD)
{'../../KNN/tic-tac-toe.mod.data': ('Mean: 75.4070981211', 'SD:
4.59223571749')}
```

3.b) Tic-Tac-Toe End game: Random subsampling: 3NN

=*=

Trial-1:
Confusion Matrix

	positive	negative
positive	296	7
negative	121	55

Accuracy: 73.2776617954

=*=
Trial-2:
Confusion Matrix

	positive	negative
positive	311	7
negative	95	66

Accuracy: 78.7056367432

=*=
Trial-3:
Confusion Matrix

	positive	negative
positive	315	9
negative	89	66

Accuracy: 79.5407098121

=*=
Trial-4:
Confusion Matrix

	positive	negative
positive	307	7
negative	93	72

Accuracy: 79.1231732777

=*=
Trial-5:
Confusion Matrix

	positive	negative
positive	299	11
negative	112	57

Accuracy: 74.3215031315

=*=
Trial-6:
Confusion Matrix

	positive	negative
positive	308	12
negative	78	81

Accuracy: 81.2108559499

=*=
Trial-7:
Confusion Matrix

	positive	negative
positive	312	8
negative	99	60

Accuracy: 77.6617954071

=*=
Trial-8:
Confusion Matrix

	positive	negative
positive	299	11
negative	102	67

Accuracy: 76.4091858038

=*=
Trial-9:
Confusion Matrix

```
+----------+----------+----------+
|          | positive | negative |
+==========+==========+==========+
| positive | 311      | 3        |
+----------+----------+----------+
| negative | 100      | 65       |
+----------+----------+----------+
Accuracy: 78.496868476

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Trial-10:
Confusion Matrix
+----------+----------+----------+
|          | positive | negative |
+==========+==========+==========+
| positive | 293      | 9        |
+----------+----------+----------+
| negative | 118      | 59       |
+----------+----------+----------+
Accuracy: 73.4864300626

=*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*==*=
Overall Mean and Standard Deviation(SD)
{'../../KNN/tic-tac-toe.mod.data': ('Mean: 77.2233820459', 'SD:
4.72434285894')}
```

4.a) Iris dataset; 1NN with 5-Fold

Trial: 1

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 2

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 3

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 4

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 5

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 6

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 7

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 8

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 9

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 10

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

================

Over all Result:

================

Mean: 99.7777777778

Standard deviation: 8.98000041244

4.b) Iris dataset; 3NN with 5-Fold

Trial: 1

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 2

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-94.4444444444

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 3

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 4

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-91.6666666667

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 5

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-97.2222222222

For Fold-5: Accuracy-100.0

Trial: 6

For Fold-1: Accuracy-94.4444444444

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 7

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-97.2222222222

For Fold-5: Accuracy-100.0

Trial: 8

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 9

For Fold-1: Accuracy-94.4444444444

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 10

For Fold-1: Accuracy-97.2222222222

For Fold-2: Accuracy-97.2222222222

For Fold-3: Accuracy-97.2222222222

For Fold-4: Accuracy-97.2222222222

For Fold-5: Accuracy-100.0

=================

Over all Result:

=================

Mean: 98.3333333333

Standard deviation: 8.85000034875

5.a) **Wine dataset; 1NN with 5-Fold**

Trial: 1

For Fold-1: Accuracy-97.6744186047

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 2

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-97.6744186047

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 3

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.6744186047

For Fold-4: Accuracy-97.6744186047

For Fold-5: Accuracy-0.0

Trial: 4

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.6744186047

For Fold-4: Accuracy-97.6744186047

For Fold-5: Accuracy-100.0

Trial: 5

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 6

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.6744186047

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-0.0

Trial: 7

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-97.6744186047

For Fold-5: Accuracy-100.0

Trial: 8

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-97.6744186047

For Fold-4: Accuracy-97.6744186047

For Fold-5: Accuracy-100.0

Trial: 9

For Fold-1: Accuracy-97.6744186047

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 10

For Fold-1: Accuracy-97.6744186047

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

================

Over all Result:

================

Mean: 95.4418604651

Standard deviation: 8.59014692081

5.b) Wine dataset; 3NN with 5-Fold

Trial: 1

For Fold-1: Accuracy-81.3953488372

For Fold-2: Accuracy-72.0930232558

For Fold-3: Accuracy-88.3720930233

For Fold-4: Accuracy-86.0465116279

For Fold-5: Accuracy-100.0

Trial: 2

For Fold-1: Accuracy-81.3953488372

For Fold-2: Accuracy-79.0697674419

For Fold-3: Accuracy-76.7441860465

For Fold-4: Accuracy-86.0465116279

For Fold-5: Accuracy-100.0

Trial: 3

For Fold-1: Accuracy-83.7209302326

For Fold-2: Accuracy-79.0697674419

For Fold-3: Accuracy-86.0465116279

For Fold-4: Accuracy-72.0930232558

For Fold-5: Accuracy-100.0

Trial: 4

For Fold-1: Accuracy-72.0930232558

For Fold-2: Accuracy-88.3720930233

For Fold-3: Accuracy-79.0697674419

For Fold-4: Accuracy-81.3953488372

For Fold-5: Accuracy-0.0

Trial: 5

For Fold-1: Accuracy-88.3720930233

For Fold-2: Accuracy-79.0697674419

For Fold-3: Accuracy-79.0697674419

For Fold-4: Accuracy-76.7441860465

For Fold-5: Accuracy-100.0

Trial: 6

For Fold-1: Accuracy-79.0697674419

For Fold-2: Accuracy-90.6976744186

For Fold-3: Accuracy-76.7441860465

For Fold-4: Accuracy-74.4186046512

For Fold-5: Accuracy-100.0

Trial: 7

For Fold-1: Accuracy-76.7441860465

For Fold-2: Accuracy-81.3953488372

For Fold-3: Accuracy-76.7441860465

For Fold-4: Accuracy-90.6976744186

For Fold-5: Accuracy-100.0

Trial: 8

For Fold-1: Accuracy-83.7209302326

For Fold-2: Accuracy-72.0930232558

For Fold-3: Accuracy-74.4186046512

For Fold-4: Accuracy-90.6976744186

For Fold-5: Accuracy-100.0

Trial: 9

For Fold-1: Accuracy-86.0465116279

For Fold-2: Accuracy-81.3953488372

For Fold-3: Accuracy-76.7441860465

For Fold-4: Accuracy-83.7209302326

For Fold-5: Accuracy-100.0

Trial: 10

For Fold-1: Accuracy-79.0697674419

For Fold-2: Accuracy-83.7209302326

For Fold-3: Accuracy-76.7441860465

For Fold-4: Accuracy-79.0697674419

For Fold-5: Accuracy-100.0

================

Over all Result:

================

Mean: 82.6046511628

Standard deviation: 7.43467443748

6.a) Tic-Tac-Toe dataset; 3NN with 5-Fold

Trial: 1

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-99.5798319328

For Fold-4: Accuracy-99.5798319328

For Fold-5: Accuracy-100.0

Trial: 2

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 3

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 4

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 5

For Fold-1: Accuracy-99.5798319328

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 6

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 7

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 8

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 9

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-100.0

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

Trial: 10

For Fold-1: Accuracy-100.0

For Fold-2: Accuracy-100.0

For Fold-3: Accuracy-99.5798319328

For Fold-4: Accuracy-100.0

For Fold-5: Accuracy-100.0

================

Over all Result:

================

Mean: 97.9663865546

Standard deviation: 8.81718010051

`6.b)` Tic-Tac-Toe dataset; 3NN with 5-Fold

Trial: 1

For Fold-1: Accuracy-85.2941176471

For Fold-2: Accuracy-77.731092437

For Fold-3: Accuracy-80.6722689076

For Fold-4: Accuracy-79.4117647059

For Fold-5: Accuracy-80.0452763456

Trial: 2

For Fold-1: Accuracy-87.8151260504

For Fold-2: Accuracy-79.8319327731

For Fold-3: Accuracy-82.7731092437

For Fold-4: Accuracy-81.0924369748

For Fold-5: Accuracy-100.0

Trial: 3

For Fold-1: Accuracy-82.3529411765

For Fold-2: Accuracy-82.3529411765

For Fold-3: Accuracy-84.0336134454

For Fold-4: Accuracy-80.2521008403

For Fold-5: Accuracy-100.0

Trial: 4

For Fold-1: Accuracy-85.2941176471

For Fold-2: Accuracy-84.8739495798

For Fold-3: Accuracy-83.1932773109

For Fold-4: Accuracy-83.1932773109

For Fold-5: Accuracy-100.0

Trial: 5

For Fold-1: Accuracy-83.1932773109

For Fold-2: Accuracy-83.1932773109

For Fold-3: Accuracy-84.4537815126

For Fold-4: Accuracy-81.9327731092

For Fold-5: Accuracy-100.0

Trial: 6

For Fold-1: Accuracy-79.4117647059

For Fold-2: Accuracy-83.1932773109

For Fold-3: Accuracy-79.8319327731

For Fold-4: Accuracy-85.7142857143

For Fold-5: Accuracy-100.0

Trial: 7

For Fold-1: Accuracy-85.7142857143

For Fold-2: Accuracy-81.0924369748

For Fold-3: Accuracy-79.8319327731

For Fold-4: Accuracy-81.512605042

For Fold-5: Accuracy-80.603550442

Trial: 8

For Fold-1: Accuracy-79.8319327731

For Fold-2: Accuracy-81.0924369748

For Fold-3: Accuracy-83.6134453782

For Fold-4: Accuracy-83.1932773109

For Fold-5: Accuracy-75.0

Trial: 9

For Fold-1: Accuracy-81.9327731092

For Fold-2: Accuracy-84.8739495798

For Fold-3: Accuracy-84.0336134454

For Fold-4: Accuracy-78.1512605042

For Fold-5: Accuracy76.0239846572

Trial: 10

For Fold-1: Accuracy-84.4537815126

For Fold-2: Accuracy-81.9327731092

For Fold-3: Accuracy-81.0924369748

For Fold-4: Accuracy-80.6722689076

For Fold-5: Accuracy-70.0

=================

Over all Result:

=================

Mean: 75.8823529412

Standard deviation: 6.83021117982

Source code

driver_Random_Sampling.py

```
#!/bin/python

# python driver_Random_Sampling.py <numberOfTrials> <k> <file-1> <feature-
count-1> <file-2> <feature-count-2> ...

import math as m
import sys
import kthNearestNeighbour_randomSampling as knc

res = {};
numberOfTrials = int( sys.argv[1] )
fractionForTrainingSet = 0.5
k = int( sys.argv[2] )

# print len(sys.argv)

for fileNum in range( 3, len(sys.argv)-1, 2 ):
	sum = 0
	squareSum = 0;
	filename = sys.argv[fileNum];
	numberOfFeatures = int(sys.argv[fileNum+1]);

	for i in range( numberOfTrials ):
		print "=*="*23
		print "Trial-" + str(i+1) + ": "
		val = knc.doKthNearestNeighbourClassification( filename,
numberOfFeatures, fractionForTrainingSet, k )
		sum += val
		squareSum += (sum*sum)
		print "Mean: " + str(val*100) + "\n"
	mean = sum*1.0/numberOfTrials;
	res[filename] = ( "Mean: " + str(mean*100), "SD: " +
str(m.sqrt(squareSum/numberOfTrials - mean)) )

print "=*="*23
print "Overall Mean and Standard Deviation(SD)"
print res
```

driver_kFold_Sampling.py

```
#!/bin/python

# python driver_Random_Sampling.py <k> <file-1> <feature-count-1>

import math as m
import sys
import kthNearestNeighbour_kFold as knc

res = {};
k = int( sys.argv[1] )
numberOfFolds = 5

# print len(sys.argv)

filename = sys.argv[2];
numberOfFeatures = int(sys.argv[3]);

res = knc.doKthNearestNeighbourClassification( filename, numberOfFeatures,
numberOfFolds, k )

def getSD(lst):
	total = sum(lst)
	sd = 0
	for i in lst:
		sd += pow( i-total, 2)
	sd /= 1.0*len(lst)

	import math as m
	return m.sqrt(sd)
```

```
print "\n================= "
print "Over all Result: "
print "================= "
print "Mean: " +str( sum(res)*1.0/len(res)*100 )
print "Standard deviation: " +str( getSD(res) )
```

kthNearestNeighbour_kFold.py

```
#!/bin/python

featuresList = [];

def loadDatasetWithNFold( dataset, numberOfFeatures, startIndexForTestData,
endIndexForTestData ):

    trainingSet=[]
    testSet=[]

    endIndexForTestData = endIndexForTestData if endIndexForTestData<len(dataset)
else len(dataset)-1

    for i in range(startIndexForTestData+1):
        trainingSet.append( dataset[i] )
        if dataset[i][-1] not in featuresList:
            featuresList.append( dataset[i][-1] )

    for i in range(startIndexForTestData+1, endIndexForTestData+1):
            testSet.append( dataset[i] )
            if dataset[i][-1] not in featuresList:
             featuresList.append( dataset[i][-1] )

    for i in range(startIndexForTestData+2, len(dataset)):
            trainingSet.append( dataset[i] )
            if dataset[i][-1] not in featuresList:
                featuresList.append( dataset[i][-1] )

    return (trainingSet, testSet)
```

```
def euclideanDist( x, y):
    dist = 0;

    import math;
    for i in range(len(x)-1):
            dist = dist + math.pow( (float(x[i])-float(y[i])), 2 );
    return math.sqrt(dist);



def getDistancesOfKSimilarSets( train, testInstance, k):

    distanes = []
    for i in range(len(train)):
        distanes.append( ( train[i], euclideanDist(train[i], testInstance) ) )

    import operator;
    distanes.sort( key=operator.itemgetter(1) );

    return [ item[0] for item in distanes[0:k] ];



def getPrediction( distOfkSimiarSets ):
        votesForTheClass = { }
        for item in distOfkSimiarSets:
                if item in votesForTheClass.keys():
                        votesForTheClass[ item[-1] ] += 1;
                else:
                        votesForTheClass[ item[-1] ] = 1;
        import operator
```

```
	l = sorted( votesForTheClass.iteritems(), key=operator.itemgetter(1),
reverse=False )



	return l[0][0];



def get_1NN( train, testSet ):
	totalCount = len(testSet)
	OneNN = [];

	import operator;
	for i in range(len(testSet)):
		OneNN.append( getDistancesOfKSimilarSets( train, testSet[i], 1)[0] )

	return OneNN;



def getAuccuracy( train, testSet, k ):
	totalCount = len(testSet)
	correctCount = 0.0;

	for i in range(len(testSet)):
		predition = getPrediction( getDistancesOfKSimilarSets( train, testSet[i], k )
)
		if predition == testSet[i][-1]:
			correctCount+=1;

	return correctCount*1.0/totalCount;
```

```
""" python driver_kFold_Sampling.py <k> <file-name> <feature-count>
"""
def doKthNearestNeighbourClassification( filename, numberOfFeatures, numberOfFolds,
k ):

        auccuracy = 0;
        sum = 0
        squareSum = 0;

        import csv
        import random
        import math as m

        with open(filename, 'rb') as csvfile:
            lines = csv.reader(csvfile)
            dataset = list(lines)
            accuracyList =[]
            for i in range(10):
                    print "--------------"
                    print "Trial: " +str(i+1)
                    random.shuffle( dataset )
                    numberOfLines = len(dataset)
                    foldSize = numberOfLines/(numberOfFolds-1)
                    currentFoldNum = 0;
                sum=0
                    for i in range(0, numberOfLines, foldSize):
                     (trainingSet, testSet) = loadDatasetWithNFold( dataset, \
                                                                  numberOfFeatures, \
                                                                          i, \
                                                                          i+foldSize-1 );
                     currentFoldNum +=1
```

```
                val = getAuccuracy( trainingSet, testSet, k )
                sum += val
                squareSum += (sum*sum)
                print "For Fold-" + str(currentFoldNum) + ": Mean-" + str(val*100)

            mean = sum*1.0/(numberOfFolds);
            accuracyList.append( mean );



        return accuracyList;#!/bin/python

featuresList = [];

def loadDatasetWithNFold( dataset, numberOfFeatures, startIndexForTestData,
endIndexForTestData ):

    trainingSet=[]
    testSet=[]

    endIndexForTestData = endIndexForTestData if endIndexForTestData<len(dataset)
else len(dataset)-1

    for i in range(startIndexForTestData+1):
        trainingSet.append( dataset[i] )
        if dataset[i][-1] not in featuresList:
            featuresList.append( dataset[i][-1] )

    for i in range(startIndexForTestData+1, endIndexForTestData+1):
        testSet.append( dataset[i] )
        if dataset[i][-1] not in featuresList:
            featuresList.append( dataset[i][-1] )
```

```
    for i in range(startIndexForTestData+2, len(dataset)):
        trainingSet.append( dataset[i] )
        if dataset[i][-1] not in featuresList:
            featuresList.append( dataset[i][-1] )

    return (trainingSet, testSet)



def euclideanDist( x, y):
    dist = 0;

    import math;
    for i in range(len(x)-1):
        dist = dist + math.pow( (float(x[i])-float(y[i])), 2 );
    return math.sqrt(dist);



def getDistancesOfKSimilarSets( train, testInstance, k):

    distanes = []
    for i in range(len(train)):
        distanes.append( ( train[i], euclideanDist(train[i], testInstance) ) )

    import operator;
    distanes.sort( key=operator.itemgetter(1) );

    return [ item[0] for item in distanes[0:k] ];



def getPrediction( distOfkSimiarSets ):
        votesForTheClass = { }
```

```
        for item in distOfkSimiarSets:
            if item in votesForTheClass.keys():
                votesForTheClass[ item[-1] ] += 1;
            else:
                votesForTheClass[ item[-1] ] = 1;
        import operator
        l = sorted( votesForTheClass.iteritems(), key=operator.itemgetter(1),
reverse=False )



        return l[0][0];



def get_1NN( train, testSet ):
        totalCount = len(testSet)
        OneNN = [];

        import operator;
        for i in range(len(testSet)):
            OneNN.append( getDistancesOfKSimilarSets( train, testSet[i], 1)[0] )

        return OneNN;



def getAuccuracy( train, testSet, k ):
        totalCount = len(testSet)
        correctCount = 0.0;

        for i in range(len(testSet)):
            predition = getPrediction( getDistancesOfKSimilarSets( train, testSet[i], k )
)
            if predition == testSet[i][-1]:
```

```
                    correctCount+=1;

        return correctCount*1.0/totalCount;




""" python driver_kFold_Sampling.py <k> <file-name> <feature-count>
"""
def doKthNearestNeighbourClassification( filename, numberOfFeatures, numberOfFolds, k ):

        auccuracy = 0;
        sum = 0
        squareSum = 0;

        import csv
        import random
        import math as m

        with open(filename, 'rb') as csvfile:
           lines = csv.reader(csvfile)
           dataset = list(lines)
           accuracyList =[]
           for i in range(10):
                   print "--------------"
                   print "Trial: " +str(i+1)
                   random.shuffle( dataset )
                   numberOfLines = len(dataset)
                   foldSize = numberOfLines/(numberOfFolds-1)
                   currentFoldNum = 0;
               sum=0
```

```
            for i in range(0, numberOfLines, foldSize):
                (trainingSet, testSet) = loadDatasetWithNFold( dataset, \
                                                               numberOfFeatures, \
                                                                       i, \
                                                                       i+foldSize-1 );
                currentFoldNum +=1
                val = getAuccuracy( trainingSet, testSet, k )
                sum += val
                squareSum += (sum*sum)
                print "For Fold-" + str(currentFoldNum) + ": Mean-" + str(val*100)

            mean = sum*1.0/(numberOfFolds);
            accuracyList.append( mean );
    return accuracyList;
```

kthNearestNeighbour_randomSampling.py

```
#!/bin/python

featuresList = [];

def loadDatasetRandSampling(filename, numberOfFeatures, fractionForTrainingSet):
        trainingSet=[]
        testSet=[]

        import csv
        import random
        import math as m
        with open(filename, 'rb') as csvfile:
            lines = csv.reader(csvfile)
            dataset = list(lines)
            lenghtOfTrainingSet = int(m.floor( fractionForTrainingSet * len(dataset) ))
            random.shuffle( dataset )

            for i in range(lenghtOfTrainingSet):
                trainingSet.append( dataset[i] )
                if dataset[i][-1] not in featuresList:
                        featuresList.append( dataset[i][-1] )
            for i in range( lenghtOfTrainingSet, len(dataset) ):
                testSet.append( dataset[i] )
                if dataset[i][-1] not in featuresList:
                        featuresList.append( dataset[i][-1] )

        return (trainingSet, testSet)
```

```
def euclideanDist( x, y):
    dist = 0;

    import math;
    for i in range(len(x)-1):
        dist = dist + math.pow( (float(x[i])-float(y[i])), 2 );
    return math.sqrt(dist);



def getDistancesOfKSimilarSets( train, testInstance, k):

    distanes = []
    for i in range(len(train)):
        distanes.append( ( train[i], euclideanDist(train[i], testInstance) ) )

    import operator;
    distanes.sort( key=operator.itemgetter(1) );

    return [ item[0] for item in distanes[0:k] ];



def getPrediction( distOfkSimiarSets ):
    votesForTheClass = { }
    for item in distOfkSimiarSets:
        if item in votesForTheClass.keys():
            votesForTheClass[ item[-1] ] += 1;
        else:
            votesForTheClass[ item[-1] ] = 1;
    import operator
```

```
        l = sorted( votesForTheClass.iteritems(), key=operator.itemgetter(1),
reverse=False )


        return l[0][0];



def get_1NN( train, testSet ):
        totalCount = len(testSet)
        OneNN = [];

        import operator;
        for i in range(len(testSet)):
                OneNN.append( getDistancesOfKSimilarSets( train, testSet[i], 1)[0] )

        return OneNN;



def getAuccuracy( train, testSet, k ):
        totalCount = len(testSet)
        correctCount = 0.0;

        # Init ConfusionMatrix
        confusionMatrix = { }
        for i in featuresList:
                for j in featuresList:
                        confusionMatrix[ (i,j) ] = 0

        for i in range(len(testSet)):
```

```
            predition = getPrediction( getDistancesOfKSimilarSets( train, testSet[i], k )
)
            if predition == testSet[i][-1]:
                correctCount+=1;
            confusionMatrix[ testSet[i][-1], predition ] += 1

        print "Confusion Matrix"
        from texttable import Texttable
        table=[]
        row=[""]
        row.extend(featuresList)
        table.append(row)
        for i in featuresList:
            row=[i]
            for j in featuresList:
                row.append( confusionMatrix[ (i,j) ])
            table.append(row)
        T=Texttable();
        T.add_rows(table)
        print T.draw();

        return correctCount*1.0/totalCount;



""" python Iris_Solution.py <sizeOfTrainingSet[0-1]> <k> <DataSetFileName>
"""
def doKthNearestNeighbourClassification( filename, numberOfFeatures,
fractionForTrainingSet, k ):

        (trainingSet, testSet) = loadDatasetRandSampling( filename, numberOfFeatures,
fractionForTrainingSet )
```

```
# print "Train set: " +str(len(trainingSet))
# print "Test set: " +str(len(testSet))
return getAuccuracy( trainingSet, testSet, k )
```

tic-tac-toe_preproess.py

```
#!/bin/python

def toDict( string1 ):
	l = string1.replace(", ", ",").split(",")
	d={};
	index=0
	for i in range(len(l)):
		d[ l[i] ] = index
		index+=1
	return d

f = open( "./tic-tac-toe.data", "r")
f1 = open( "./tic-tac-toe.mod.data", "w")

while True:
	l = f.readline()
	if len(l.strip()) == 0:
		break;
	l = l.replace('o,', '0,')
	l = l.replace('x,', '1,')
	l = l.replace('b,', '2,')
	f1.write(l);

f.close();
f1.close();
```

plot_iris.py

```
#!/bin/python

x=[]
y=[]
class1=[]
xyc=[]

import csv
import random
import math as m
import sys
with open(sys.argv[1], 'rb') as csvfile:
	lines = csv.reader(csvfile)
	dataset = list(lines)
	for line in dataset:
		x.append( line[1] )
		y.append( line[3] )
		class1.append( line[4] )
		xyc.append( (line[1], line[3], line[4]) )

lowX, heighX = m.floor(float(min(x))), m.ceil(float(max(x)))
lowY, heighY = m.floor(float(min(y))), m.ceil(float(max(y)))

import numpy as np
x_test = np.arange( lowX, heighX+0.2, 0.2 )
y_test = np.arange( lowY, heighY+0.2, 0.2 )
testSet = []

for i in range(len(x_test)):
```

```
    x_test[i], y_test[i] = float("{0:.1f}".format( x_test[i] )),
float("{0:.1f}".format( y_test[i] ))


for i in x_test:
    for j in y_test:
        testSet.append([i,j])



import kthNearestNeighbour_randomSampling as knn


resultSet = [];
points = {}
for tSet in testSet:
    prediction = knn.getPrediction( knn.getDistancesOfKSimilarSets( zip(x,y,class1),
tSet, 1 ) )
    resultSet.append( tSet )
    a,b = float("{0:.1f}".format( tSet[0] )), float("{0:.1f}".format( tSet[1] ))
    if prediction == "Iris-setosa":
        resultSet[-1].extend( ['r'] )
        points[ (a,b) ] = 'r'
        # print (a,b)
    elif prediction == "Iris-versicolor":
        resultSet[-1].extend( ['y'] )
        points[ (a,b) ] = 'y'
        # print (a,b)
    elif prediction == "Iris-virginica":
        resultSet[-1].extend( ['b'] )
        points[ (a,b) ] = 'b'
        # print (a,b)



from matplotlib import pyplot as ppl
```

```
import matplotlib.patches as mpatches


# Plot data-set from the file
isa = ppl.plot( [ i[0] for i in xyc if  i[-1]=='Iris-setosa' ],
                              [ i[1] for i in xyc if  i[-1]=='Iris-setosa' ], 'r+', label="Iris-setosa" )
ivr = ppl.plot( [ i[0] for i in xyc if  i[-1]=='Iris-versicolor' ],
                              [ i[1] for i in xyc if  i[-1]=='Iris-versicolor' ], 'yo', label="Iris-
versicolor" )
iva = ppl.plot( [ i[0] for i in xyc if  i[-1]=='Iris-virginica' ],
                              [ i[1] for i in xyc if  i[-1]=='Iris-virginica' ], 'b^', label="Iris-
virginica" )
ppl.xlabel("Sepal width")
ppl.ylabel("Petal width")
ppl.legend();



# Find decision bounday
def get_closest_point( list, point ):
          list.sort( key = lambda p : (p[0]-point[0])**2 + (p[0]-point[0])**2 )
          return list[0]


bound_ry_x = [];
bound_ry_y = [];


print x_test
print y_test
for i in x_test:
          for j in y_test:
                    x1, y1 = float("{0:.1f}".format( i )), float("{0:.1f}".format( j ))
                    x1_next, y1_next = float("{0:.1f}".format( i+0.2 )),
float("{0:.1f}".format( j+0.2 ))
                    # print x1, y1
```

```
            if x1_next<=heighX and points[ (x1,y1) ] == 'r' and
points[ (x1_next,y1) ] != 'r':
                bound_ry_x.append( x1+0.1 )
                bound_ry_y.append( y1 )
            elif y1_next<=heighY and points[ (x1,y1) ] == 'r' and points[ (x1,y1_next)
] != 'r':
                bound_ry_x.append( x1 )
                bound_ry_y.append( y1+0.1 )



bound_yb_x = [];
bound_yb_y = [];
y_test = y_test[::-1]
for i in x_test:
    for j in y_test:
        x1, y1 = float("{0:.1f}".format( i )), float("{0:.1f}".format( j ))
        x1_next, y1_next = float("{0:.1f}".format( i+0.2 )),
float("{0:.1f}".format( j-0.2 ))
        if x1_next<=heighX and points[ (x1,y1) ] == 'b' and points[ (x1_next,y1) ]
!= 'b':
            bound_yb_x.append( x1+0.1 )
            bound_yb_y.append( y1 )
        elif y1_next>=lowY and points[ (x1,y1) ] == 'b' and
points[ (x1,y1_next) ] != 'b':
            bound_yb_x.append( x1 )
            bound_yb_y.append( y1-0.1 )



ppl.plot( bound_ry_x, bound_ry_y, "b-");
ppl.plot( bound_yb_x, bound_yb_y, "b-");
ppl.show();
```

Answers to the Questions from Question paper

Answer 1:

Iris	
Data Set Characteristics	Multivariate
Number of Instances	150
Attribute Characteristics	Real
Number of Attributes	4
Associated Tasks	Classification
No of Classes	3 1. Iris Setosa 2. Iris Versicolour 3. Iris Virginica
Wine	
Data Set Characteristics	Multivariate
Number of Instances	48842
Attribute Characteristics	Real
Number of Attributes	13
Associated Tasks	Classification
No of Classes	3
Tic-Tac-Toe Endgame Data Set	
Data Set Characteristics	Multivariate
Number of Instances	958
Attribute Characteristics	Real
Number of Attributes	9
Associated Tasks	Classification

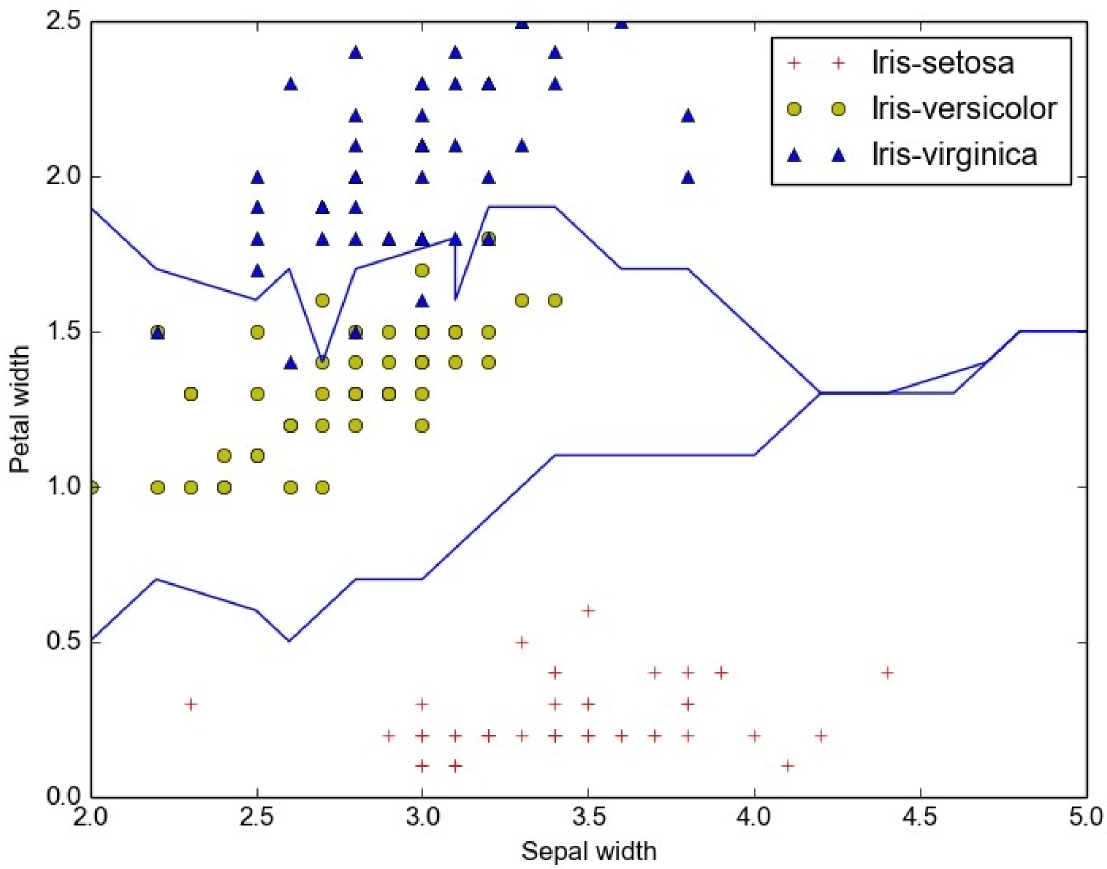
No of Classes	2 1. Positive 2. Negative
In the *Tic-Tac-Toe* dataset all the given values of each of the feature are replaed with a unique integer and then the value of the features that are thus obtained are used as an input to the algorithm(KNN, K-Fold) for permorming the classification.	

Answer 2:

The observations can be found in the page Confusion Matrix, Mean and Standard Deviation.

Observation:
The data-set for which the Random sub-sampling is giving good accuray, the 5-Fold subsampling is giving better accuracy. It is might be because in the 5-Fold all the values in the training set are contributing to the classification process.

Answer 3:



Answer 4:

YES.

The deision boundary of 3-NN will be piece-wise linear.

Reason: In 3-NN the decision boundary is formed by considering the perpendicular bisectors of the imaginary line formed by connecting the transition-points(The meaning of *transition-points* is given in the Question-3 of Assignment-1 doument). Eah of these perpendicular bisectors are linear. Hene we can conlude that the deision boundary of 3-NN is piece-wise linear (each piee is provided by eah of the perpendiular-bisectors).