Satvinder Singh Panesar | UBit name: satvinde | Person No: 5024 8888

Project 3: Classification Algorithms

1. Nearest Neighbor

Flow:

To classify an unknown record: –

Compute distance to other training records

```
distance = np.subtract(feature, another_feature)
distance = [pow(x, 2) for x in distance]
distance = math.sqrt(sum(distance))
```

Fig: Python code snippet for computing distance (Euclidian)

Identify k nearest neighbors

Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

```
predictions.append(stats.mode(votes))
```

Fig: Python code snippet for getting majority vote

Preprocessing:

Normalizing data to reduce value dominance.

```
# normalizing data
means = {}
std_devs = {}

for i in range(0, no_of_columns - 1):
    if i not in columns_with_categorical_data:
        means[i] = np.mean(features[:,i])
        std_devs[i] = np.std(features[:,i])

for feature_index, feature in enumerate(features):
    for attr_index, ele in enumerate(feature):
        if attr_index not in columns_with_categorical_data and attr_index < no_of_columns - 1:
            temp = (ele - means[attr_index])/std_devs[attr_index]
            feature[attr_index] = temp
        features[feature_index] = feature
```

Fig: Python code snippet for data normalization

Converting categorical to numerical values

```
# pre processing
for ele_index, ele in enumerate(features):
    for val_index, val in enumerate(ele):
        try:
            isinstance(float(val), float)
        except:
            if val in string_to_number_map:
                ele[val_index] = string_to_number_map[val]
            else:
                length = len(string_to_number_map.keys())
                ele[val_index] = length
                string_to_number_map[val] = length
    features[ele_index] = ele
features = [[float(y) for y in x] for x in features]
```

Fig: Python code snippet for categorical value conversion

Parameter setting:

If k is too small, sensitive to noise points

If k is too large, neighborhood may include points from other classes

Evaluating algorithm (Same for all below algorithms):

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

accuracy = round((a+d)/(a+b+c+d), 3)

precision = round(a/(a+c), 3)

recall = round(a/(a+b), 3)

f_measure = round((2*a)/(2*a+b+c), 3)

Observations:

We use 10-fold cross validation

k = 5 (Number of neighbors)

====>project3_dataset1.txt

Fold: 1			
Accuracy:0.965	Precision: 0.957	Recall: 0.957	F-measure: 0.957
Fold: 2			
Accuracy:0.965	Precision: 0.941	Recall: 0.941	F-measure: 0.941
Fold: 3			
Accuracy:0.965	Precision: 0.929	Recall: 0.929	F-measure: 0.929
Fold: 4			
Accuracy:0.965	Precision: 1.0	Recall: 0.905	F-measure: 0.95
Fold: 5			
Accuracy:0.912	Precision: 1.0	Recall: 0.75	F-measure: 0.857
Fold: 6			
Accuracy:0.982	Precision: 1.0	Recall: 0.963	F-measure: 0.981
Fold: 7			
Accuracy:1.0	Precision: 1.0	Recall: 1.0	F-measure: 1.0
Fold: 8			
Accuracy:0.982	Precision: 1.0	Recall: 0.933	F-measure: 0.966
Fold: 9			
Accuracy:0.947	Precision: 1.0	Recall: 0.903	F-measure: 0.949
Fold: 10			
Accuracy:0.982	Precision: 0.957	Recall: 1.0	F-measure: 0.978

==>Average metric values:
Accuracy:0.966
Precision: 0.978
Recall: 0.928
F-measure: 0.951

====>project3_dataset2.txt

Fold: 1			
Accuracy:0.596	Precision: 0.556	Recall: 0.476	F-measure: 0.513

Fold: 2			
Accuracy:0.766	Precision: 0.556	Recall: 0.769	F-measure: 0.645
Fold: 3			
Accuracy:0.63	Precision: 0.6	Recall: 0.316	F-measure: 0.414
Fold: 4			
Accuracy:0.717	Precision: 0.583	Recall: 0.467	F-measure: 0.519
Fold: 5			
Accuracy:0.565	Precision: 0.5	Recall: 0.1	F-measure: 0.167
Fold: 6			
Accuracy:0.587	Precision: 0.444	Recall: 0.222	F-measure: 0.296
Fold: 7			
Accuracy:0.739	Precision: 0.556	Recall: 0.385	F-measure: 0.455
Fold: 8			
Accuracy:0.696	Precision: 0.286	Recall: 0.182	F-measure: 0.222
Fold: 9			
Accuracy:0.522	Precision: 0.312	Recall: 0.312	F-measure: 0.312
Fold: 10			
Accuracy:0.674	Precision: 0.462	Recall: 0.429	F-measure: 0.444

==>Average metric values:
Accuracy:0.649
Precision: 0.485
Recall: 0.366
F-measure: 0.399

Pros:

Easy to implement.

Really good performance for continuous values.

Cons:

Sensitive to initialization and outliers.

Comments:

Lazy learners i.e. don't build model explicitly.

Works better with continuous values as distance calculation not appropriate for categorical values.

2. Decision trees

Flow:

Starting at root of decision tree traverse until leaf is found, which in turn is the predicted label.

If data has records that belong to single class, form the leaf node.

If data has records that belong to more than one class, use an attribute and split the data into smaller subsets.

Recursively apply procedure to each subset.

In our approach, "1." Is the root node, adding "1" gives left child and adding "2" gives right child.

```
build_tree_2(left_set, no_of_features_to_use, get_entropy(left_set), columns_with_categorical_data, id+"1", 
build_tree_2(right_set, no_of_features_to_use, get_entropy(right_set), columns_with_categorical_data, id+"2"
```

Fig: Left and right children

Finding the best attribute to split:

Split the records based on an attribute test that optimizes certain criterion. (gain in our case).

First we need to find entropy which measures impurity:

$$Entropy(t) = -\sum_j p(j \mid t) \log p(j \mid t)$$

```
def get_entropy(data):
    if len(data) == 0:
        return 0
    no_of_columns = len(data[0])
    no_of_objects = len(data)
    no_of_ones = len([x for x in data if x[no_of_columns-1] == 1])
    no_of_zeros = no_of_objects - no_of_ones
    prob_of_one = no_of_ones / no_of_objects
    prob_of_zero = no_of_zeros / no_of_objects
    if prob_of_one == 0:
        return round(-(prob_of_zero * math.log(prob_of_zero, 2)), 3)
    elif prob_of_zero == 0:
        return round(-(prob_of_one * math.log(prob_of_one, 2)), 3)
    else:
        return round(-(prob_of_one * math.log(prob_of_one, 2))-(prob_of_zero * math.log(prob_of_zero, 2)), 3)
```

Fig: Python code snippet for entropy calculation

Max when all records are uniformly distributed.

Min when all records belong to same class.

Then compute gain:

$$GAIN_{split} = Measure(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Measure(i) \right)$$

```
entropy_div1 = get_entropy(div1)
entropy_div2 = get_entropy(div2)
info = (len(div1)/no_of_objects*entropy_div1) + (len(div2)/no_of_objects*entropy_div2)
info_gained = data_entropy - info
```

Fig: Python code snippet for calculating gain

Splitting the data:

For continuous attributes: Binary Decision: (A < v) or (A ≥ v)

For nominal attributes: Two- way split: (A == v) or (A! = v)

Preprocessing:

Same as nearest neighbor we convert categorical attributes into nominal attributes

Post processing or avoiding overfitting:

Allow decision tree to grow fully.

Trim nodes in a bottom up fashion until training accuracy is not 1.

```python
if int(accuracy) == 1:
    # post pruning
    while True:
        leaf_nodes = [x for x in tree_map if tree_map[x] == 1 or tree_map[x] == 0]
        level = [len(x) for x in leaf_nodes]
        max_level = level.index(max(level))
        node_to_prune = leaf_nodes[max_level]
        parent_node = node_to_prune[0:len(node_to_prune)-1]
        if parent_node == "1.":
            break
        leaf_node_value = tree_map[node_to_prune]
        del tree_map[node_to_prune]
        tree_map[parent_node] = leaf_node_value

        predictions = []

        for data in local_test_data:
            predictions.append(predict_using_decision_tree(data, tree_map, columns_with_categorical_data))

        accuracy, precision, recall, f_measure, miss_classification_rate = evaluate(train_labels, predictions)

        if accuracy < 1:
            break
```

Fig: Python code snippet for post pruning of decision trees

Observations:

We use 10-fold cross validation.

====>project3_dataset1.txt

Fold: 1			
Accuracy:0.877	Precision: 0.864	Recall: 0.826	F-measure: 0.844
Fold: 2			
Accuracy:0.895	Precision: 0.789	Recall: 0.882	F-measure: 0.833
Fold: 3			
Accuracy:1.0	Precision: 1.0	Recall: 1.0	F-measure: 1.0
Fold: 4			
Accuracy:0.947	Precision: 0.909	Recall: 0.952	F-measure: 0.93
Fold: 5			
Accuracy:0.86	Precision: 0.8	Recall: 0.8	F-measure: 0.8
Fold: 6			
Accuracy:0.965	Precision: 1.0	Recall: 0.926	F-measure: 0.962
Fold: 7			
Accuracy:0.93	Precision: 0.875	Recall: 0.955	F-measure: 0.913
Fold: 8			
Accuracy:0.965	Precision: 0.933	Recall: 0.933	F-measure: 0.933
Fold: 9			
Accuracy:0.877	Precision: 0.929	Recall: 0.839	F-measure: 0.881
Fold: 10			
Accuracy:0.893	Precision: 0.864	Recall: 0.864	F-measure: 0.864

==>Average metric values:
Accuracy:0.921
Precision: 0.896
Recall: 0.898
F-measure: 0.896

====>project3_dataset2.txt

Fold: 1			
Accuracy:0.532	Precision: 0.474	Recall: 0.429	F-measure: 0.45

Fold: 2			
Accuracy:0.702	Precision: 0.474	Recall: 0.692	F-measure: 0.562
Fold: 3			
Accuracy:0.522	Precision: 0.412	Recall: 0.368	F-measure: 0.389
Fold: 4			
Accuracy:0.522	Precision: 0.316	Recall: 0.4	F-measure: 0.353
Fold: 5			
Accuracy:0.565	Precision: 0.5	Recall: 0.4	F-measure: 0.444
Fold: 6			
Accuracy:0.543	Precision: 0.412	Recall: 0.389	F-measure: 0.4
Fold: 7			
Accuracy:0.587	Precision: 0.286	Recall: 0.308	F-measure: 0.296
Fold: 8			
Accuracy:0.739	Precision: 0.455	Recall: 0.455	F-measure: 0.455
Fold: 9			
Accuracy:0.609	Precision: 0.429	Recall: 0.375	F-measure: 0.4
Fold: 10			
Accuracy:0.674	Precision: 0.467	Recall: 0.5	F-measure: 0.483

==>Average metric values:
Accuracy:0.6
Precision: 0.422
Recall: 0.432
F-measure: 0.423

Pros:

Simple to interpret.

Unlike nearest neighbor, builds an explicit model.

Unlike nearest neighbor, no need for any initialization.

Good for small dataset.

Cons:

Becomes too large and complicated for large datasets.

Over fitting issues.

Comments:

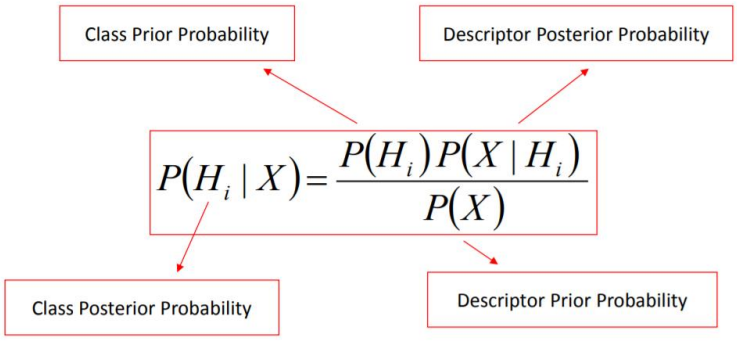
Nearest neighbor has better accuracy and this is mostly due to overfitting.

Notice that we are although post pruning the decision tree we stop when training accuracy is less than 1.

Maybe if we continue to prune until accuracy doesn’t reduce we may get a better accuracy.

3. Naïve Bayes

Flow:



We choose the outcome with highest class posterior probability.

Class prior probability is class prior probability that X belongs to a particular class.

```
# stores probability of 0 and 1
class_prior_probabilities = []

for ele in set(train_labels):
    class_prior_probabilities.append(len([x for x in train_labels if x == ele])/len(train_labels))
```

Fig: Python code snippet for calculating class prior probabilities

Descriptor prior probability is prior probability of X.

Descriptor posterior probability is posterior probability of X given Hi.

```
for ele_index, ele in enumerate(data):
    if ele_index not in columns_with_categorical_data:
        temp = mean_std_dev[ele_index]
        temp = temp[label_index]
        # representing random variables as a normal distibution and using pdf
        intermediate_probabilities.append(scipy.stats.norm(temp[0], temp[1]).pdf(ele))
    else:
        num = len([x for x in train_data if x[ele_index] == ele and x[-1] == label])
        intermediate_probabilities.append(num/den)
posterior_probability = np.product(intermediate_probabilities)
```

Fig: Python code snippet for calculating descriptor posterior probabilities

For continuous attributes, we assume distribution to be normal and use pdf to calculate probability.

For above step we need to store mean and standard deviation of all labels in a list.

```
# stores [mean, std] for 0 and 1 of all columns
mean_std_dev = []
for i in range(0, len(train_data[0]) -1):
    temp = []
    for label in labels:
        data = [x[i] for x in train_data if x[-1] == label]
        temp.append([np.mean(data), np.std(data)])
    mean_std_dev.append(temp)
```

Fig: Python code snippet for storing mean and standard deviation

For categorical attributes, we find total and actual count to get probability.

```
num = len([x for x in train_data if x[ele_index] == ele and x[-1] == label])
intermediate_probabilities.append(num/den)
```

Fig: Python code snippet for calculating descriptor posterior probability for categorical attributes

Preprocessing:

Like nearest neighbor we again convert categorical into numerical data.

Note: We have avoided zero probability problem using Laplacian correction.

Observations:

We use 10-fold cross validation.

====>project3_dataset1.txt

Fold: 1			
Accuracy:0.947	Precision: 0.917	Recall: 0.957	F-measure: 0.936
Fold: 2			
Accuracy:0.93	Precision: 0.882	Recall: 0.882	F-measure: 0.882
Fold: 3			
Accuracy:0.965	Precision: 0.875	Recall: 1.0	F-measure: 0.933
Fold: 4			
Accuracy:0.912	Precision: 0.9	Recall: 0.857	F-measure: 0.878
Fold: 5			
Accuracy:0.895	Precision: 0.889	Recall: 0.8	F-measure: 0.842
Fold: 6			
Accuracy:0.965	Precision: 1.0	Recall: 0.926	F-measure: 0.962
Fold: 7			
Accuracy:0.965	Precision: 0.955	Recall: 0.955	F-measure: 0.955
Fold: 8			
Accuracy:0.965	Precision: 0.933	Recall: 0.933	F-measure: 0.933
Fold: 9			
Accuracy:0.912	Precision: 0.964	Recall: 0.871	F-measure: 0.915
Fold: 10			
Accuracy:0.893	Precision: 0.864	Recall: 0.864	F-measure: 0.864

==>Average metric values:
Accuracy:0.935
Precision: 0.918
Recall: 0.904
F-measure: 0.91

====>project3_dataset2.txt

Fold: 1			
Accuracy:0.66	Precision: 0.6	Recall: 0.714	F-measure: 0.652
Fold: 2			
Accuracy:0.745	Precision: 0.524	Recall: 0.846	F-measure: 0.647
Fold: 3			
Accuracy:0.783	Precision: 0.765	Recall: 0.684	F-measure: 0.722
Fold: 4			
Accuracy:0.674	Precision: 0.5	Recall: 0.6	F-measure: 0.545
Fold: 5			
Accuracy:0.63	Precision: 0.6	Recall: 0.45	F-measure: 0.514
Fold: 6			
Accuracy:0.63	Precision: 0.526	Recall: 0.556	F-measure: 0.541

Fold: 7			
Accuracy:0.848	Precision: 0.75	Recall: 0.692	F-measure: 0.72
Fold: 8			
Accuracy:0.761	Precision: 0.5	Recall: 0.545	F-measure: 0.522
Fold: 9			
Accuracy:0.609	Precision: 0.444	Recall: 0.5	F-measure: 0.471
Fold: 10			
Accuracy:0.696	Precision: 0.5	Recall: 0.571	F-measure: 0.533

==>Average metric values:
Accuracy:0.704
Precision: 0.571
Recall: 0.616
F-measure: 0.587

Pros:

Unlike decision trees, can work with large dataset.

Yields optimal classifiers. (if attributes are independent)

Unlike nearest neighbor, no need for any initialization.

Cons:

Assumes attribute independence.

Comments:

Works best with independent attributes as indicated by dataset2.

4. Random Forest

Flow:

Choose number of trees to grow.

Choose number of features used to calculate the best split at each node.

For each tree:

- Choose a training set by choosing N times (N is the number of training examples) with replacement from the training set

```
# sampling with replacement
indices = np.random.choice(len(train_data), int(len(train_data) / no_of_decision_trees), replace = True)
for index in indices:
    data.append(train_data[index])
```

Fig: Python code snippet for sampling with replacement

- For each node, randomly choose m features and calculate the best split

```
no_of_columns = len(train_data[0])

# select columns randomly from train data
columns_selected = random.sample(range(0, no_of_columns - 1), no_of_features_to_use)
```

Fig: Python code snippet for choosing features

- Fully grown and not pruned

Use majority voting among all the trees.

Random Decision Tree:

At each node, an un-used feature is chosen randomly:

- A discrete feature is un-used if it has never been chosen previously on a given decision path starting from the root to the current node.
- A continuous feature can be chosen multiple times on the same decision path, but each time a different threshold value is chosen

We stop when one of the following happens:

- A node becomes too small (<= 3 examples).
- Or the total height of the tree exceeds some limits, such as the total number of features.

```
# if training samples are small or height of tree > number of features
if len(local_train_data) <= 3 or len(id) > no_of_columns:
    try:
        tree_map[id] = stats.mode(local_train_data[:,len(local_train_data[0])-1])
    except:
        tree_map[id] = random.choice(local_train_data[:,len(local_train_data[0])-1])
    return
```

Fig: Python code snippet for decision tree conditions

Preprocessing:

Like nearest neighbor, we replace categorical with numerical values.

Parameter setting:

A higher number of tress will reduce variability but increase runtime.

Number of features used to calculate the best split at each node is typically 20%.

Observations:

We use 10-fold cross validation

Number of decision trees = 5

Number of features randomly selected at each node = 20% (ceil) of total number of features

====>project3_dataset1.txt

Fold: 1			
Accuracy:0.947	Precision: 0.917	Recall: 0.957	F-measure: 0.936
Fold: 2			
Accuracy:0.93	Precision: 0.882	Recall: 0.882	F-measure: 0.882
Fold: 3			
Accuracy:0.965	Precision: 1.0	Recall: 0.857	F-measure: 0.923
Fold: 4			
Accuracy:0.947	Precision: 0.909	Recall: 0.952	F-measure: 0.93
Fold: 5			
Accuracy:0.947	Precision: 1.0	Recall: 0.85	F-measure: 0.919
Fold: 6			
Accuracy:0.93	Precision: 0.96	Recall: 0.889	F-measure: 0.923
Fold: 7			
Accuracy:0.965	Precision: 0.917	Recall: 1.0	F-measure: 0.957
Fold: 8			
Accuracy:0.965	Precision: 1.0	Recall: 0.867	F-measure: 0.929

Fold: 9			
Accuracy:0.947	Precision: 0.967	Recall: 0.935	F-measure: 0.951
Fold: 10			
Accuracy:0.946	Precision: 1.0	Recall: 0.864	F-measure: 0.927

==>Average metric values:
Accuracy:0.949
Precision: 0.955
Recall: 0.905
F-measure: 0.928

====>project3_dataset2.txt

Fold: 1			
Accuracy:0.468	Precision: 0.3	Recall: 0.143	F-measure: 0.194
Fold: 2			
Accuracy:0.723	Precision: 0.5	Recall: 0.462	F-measure: 0.48
Fold: 3			
Accuracy:0.609	Precision: 0.6	Recall: 0.158	F-measure: 0.25
Fold: 4			
Accuracy:0.609	Precision: 0.4	Recall: 0.4	F-measure: 0.4
Fold: 5			
Accuracy:0.587	Precision: 0.6	Recall: 0.15	F-measure: 0.24
Fold: 6			
Accuracy:0.522	Precision: 0.333	Recall: 0.222	F-measure: 0.267
Fold: 7			
Accuracy:0.717	Precision: 0.5	Recall: 0.385	F-measure: 0.435
Fold: 8			
Accuracy:0.717	Precision: 0.417	Recall: 0.455	F-measure: 0.435
Fold: 9			
Accuracy:0.587	Precision: 0.364	Recall: 0.25	F-measure: 0.296
Fold: 10			
Accuracy:0.674	Precision: 0.444	Recall: 0.286	F-measure: 0.348

==>Average metric values:
Accuracy:0.621
Precision: 0.446
Recall: 0.291
F-measure: 0.334

Note: When you re execute the code, values may change as random samples are taken always.

Pros:

Solves overfitting by majority vote.

Can handle large number of features as we choose limited number of random features at each node.

Cons:

Slow to execute.

Complex to implement.

Comments:

Overfitting reduced in comparison to decision trees.

Variability reduced due to many trees.

5. Boosting

Flow:

Boost a set of weak learners to a strong learner.

Make records currently misclassified more important.

Initially, set uniform weights on all the records.

```
weights = []
weight = 1/len(train_data)
for i in range(0, len(train_data)):
    weights.append(weight)
```

Fig: Python code snippet for weight vector

At each round

• Create a bootstrap sample based on the weights

• Train a classifier on the sample and apply it on the original training set

• Records that are wrongly classified will have their weights increased

Calculating error:

$$\varepsilon_i = \frac{\sum_{j=1}^{N} w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^{N} w_j}$$

Calculating importance of each classifier:

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

```
alpha = (1/2) * math.log((1-error)/error)
```

Fig: Python code snippet for alpha calculations

• Records that are classified correctly will have their weights decreased

Update the weights of the data points for the next iteration. If $k_m(x_i)$ is a miss, set

$$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m} = w_i^{(m)} \sqrt{\frac{1-e_m}{e_m}}$$

otherwise

$$w_i^{(m+1)} = w_i^{(m)} e^{-\alpha_m} = w_i^{(m)} \sqrt{\frac{e_m}{1-e_m}}$$

```
# updating weights
for index, weight in enumerate(weights):
    if index in indices:
        position = indices.index(index)
        if predictions[position] != local_train_labels[position]:
            weights[index] = weights[index] * math.sqrt((1-error)/error)
        else:
            weights[index] = weights[index] * math.sqrt(error/(1-error))
```

Fig: Python code snippet for weight updating

• If the error rate is higher than 50%, start over

Final prediction is weighted average of all the classifiers with weight representing the training accuracy.

$$C^*(x) = \arg\max_y \sum_{i=1}^{K} \alpha_i \delta(C_i(x) = y)$$

Preprocessing:

Like nearest neighbor, we replace categorical with numerical values.

Parameter Setting:

Need to set number of decision trees.

Observations:

We use 10-fold cross validation

Number of decision trees = 5

====>project3_dataset1.txt

Fold: 1			
Accuracy:0.965	Precision: 0.957	Recall: 0.957	F-measure: 0.957
Fold: 2			
Accuracy:0.982	Precision: 1.0	Recall: 0.941	F-measure: 0.97
Fold: 3			
Accuracy:0.965	Precision: 1.0	Recall: 0.857	F-measure: 0.923
Fold: 4			
Accuracy:0.965	Precision: 1.0	Recall: 0.905	F-measure: 0.95
Fold: 5			
Accuracy:0.93	Precision: 0.944	Recall: 0.85	F-measure: 0.895
Fold: 6			
Accuracy:0.895	Precision: 0.957	Recall: 0.815	F-measure: 0.88
Fold: 7			
Accuracy:0.965	Precision: 0.917	Recall: 1.0	F-measure: 0.957
Fold: 8			
Accuracy:0.982	Precision: 1.0	Recall: 0.933	F-measure: 0.966
Fold: 9			
Accuracy:0.895	Precision: 0.963	Recall: 0.839	F-measure: 0.897
Fold: 10			
Accuracy:0.946	Precision: 0.913	Recall: 0.955	F-measure: 0.933

==>Average metric values:
Accuracy:0.949
Precision: 0.965
Recall: 0.905
F-measure: 0.933

====>project3_dataset2.txt

Fold: 1			
Accuracy:0.596	Precision: 0.571	Recall: 0.381	F-measure: 0.457
Fold: 2			
Accuracy:0.702	Precision: 0.429	Recall: 0.231	F-measure: 0.3
Fold: 3			
Accuracy:0.63	Precision: 0.571	Recall: 0.421	F-measure: 0.485
Fold: 4			
Accuracy:0.674	Precision: 0.5	Recall: 0.333	F-measure: 0.4
Fold: 5			
Accuracy:0.696	Precision: 0.875	Recall: 0.35	F-measure: 0.5
Fold: 6			
Accuracy:0.543	Precision: 0.4	Recall: 0.333	F-measure: 0.364
Fold: 7			
Accuracy:0.63	Precision: 0.375	Recall: 0.462	F-measure: 0.414
Fold: 8			
Accuracy:0.761	Precision: 0.5	Recall: 0.455	F-measure: 0.476
Fold: 9			
Accuracy:0.63	Precision: 0.455	Recall: 0.312	F-measure: 0.37
Fold: 10			
Accuracy:0.761	Precision: 0.615	Recall: 0.571	F-measure: 0.593

==>Average metric values:
Accuracy:0.662
Precision: 0.529
Recall: 0.385
F-measure: 0.436

Note: When you re execute the code, values may change as random samples are taken always.

Pros:

Less initialization parameters than Random Forest.

Cons:

Complex to implement.

Serial execution is slower as compared to parallel execution of random forest.

Comments:

Accuracy better than Random Forest for dataset2.

Comparison of different algorithms:

		Average Accuracy	
		dataset1	**dataset2**
Algorithm	**Nearest Neighbor**	**0.966**	**0.649**
	Decision tree	**0.921**	**0.6**
	Naïve Bayes	**0.935**	**0.704**
	Random Forest	**0.949**	**0.621**
	Boosting	**0.949**	**0.662**

Conclusion:

Nearest neighbor works good for continuous values but does no explicit learning.

Decision tree works good for small datasets and can introduce overfitting.

Naïve Bayes returns class probabilities but assumes attribute independence.

Random Forest reduces variability by majority vote.

Boosting increases performance but increases runtime due to serial nature of execution.

References:

Lecture slides by Professor Gao

http://www.inf.fu-berlin.de/inst/ag-ki/adaboost4.pdf

https://www.fil.ion.ucl.ac.uk/~wpenny/course/bayes.pdf