2017 Fall

CSE 601

Project-3 Report

Xuan Han 50168962

Laiyun Wu 50061617

Ting Zhou 50169020

The First Algorithm: K-Nearest Neighbors

1. Algorithm Flow
    - Predefine k and distance metrics;
    - Compute the distances of each point to all the other points and assign this point to the major class of its k nearest neighbors;
2. Choose Hyperparameters

    For k-nn, there are two hyperparameters need to be determined, which are the number of neighbors k and the distance metrics. Both un-normalized and normalized data are tried to test the results and determine the hyperparameters. After comparing the four performance metrics as shown in Figure1.1-1.4, the hyperparameters are summarized in Table1.1.

Table1.1 Hyperparameters

Data	K	Dist-Metric
Dataset1	13	L2
Dataset2	24	L2

3. Results Analysis

    As shown in Figure1.1-1.4, the performances of normalized data for both dataset1 and dataset2 are better than un-normalized data. The results are from the average of cross validation splitting data into 10 folds. K-nn performs better on dataset1 obviously.

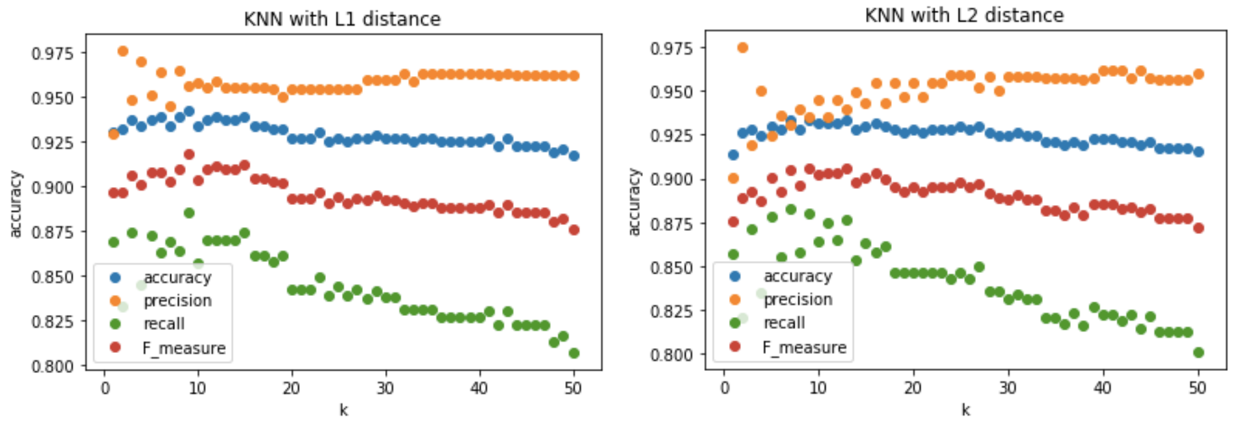


Figure1.1. KNN on Un-Normalized Data (dataset1)

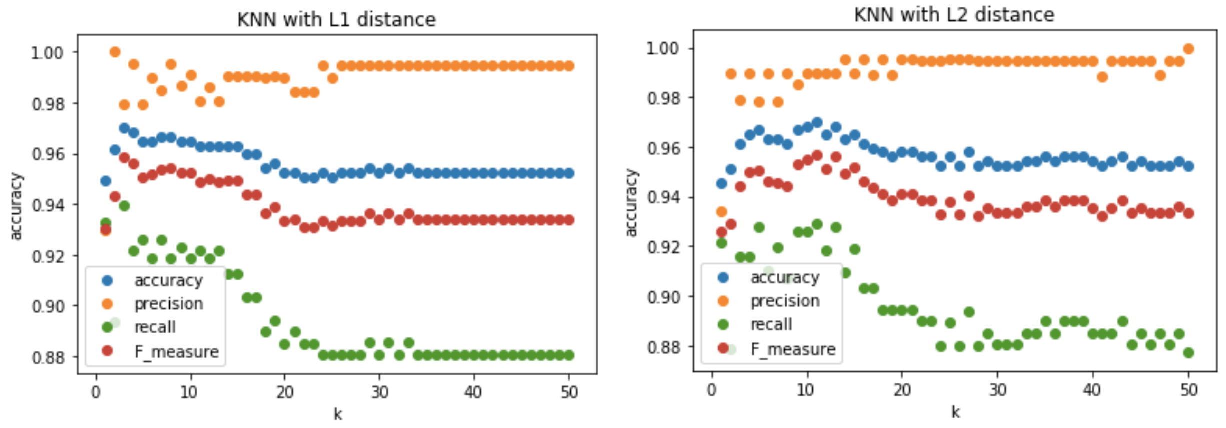
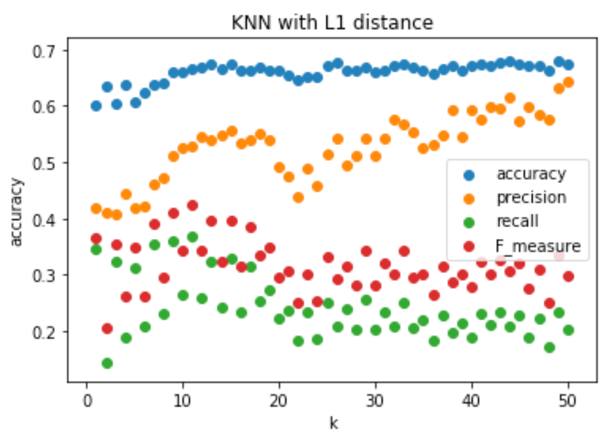


Figure1.2. KNN on Normalized Data (dataset1)



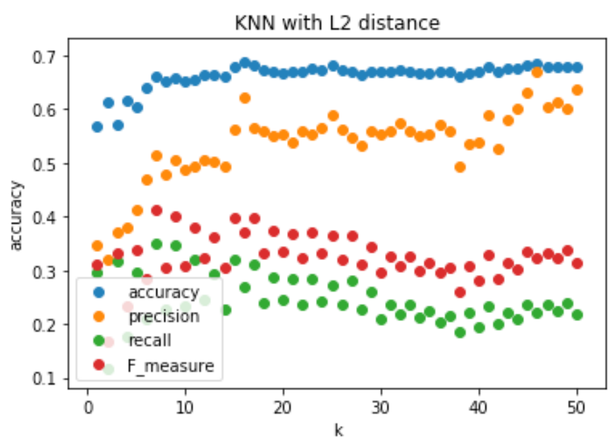
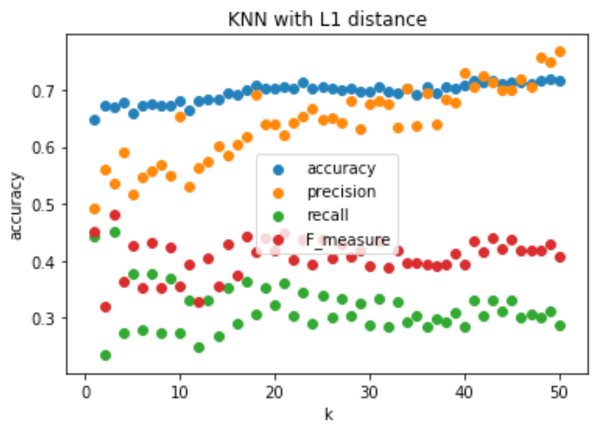


Figure1.1. KNN on Un-Normalized Data (dataset2)



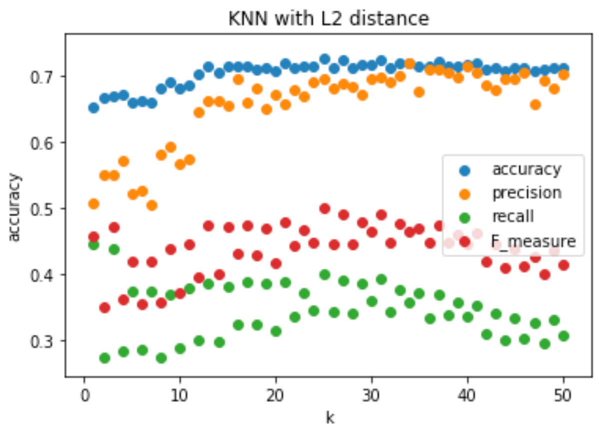


Figure1.4 KNN on Normalized Data (dataset2)

4. Pros and Cons

   Pros:

   - K-nn is simple to implement;
   - Training is very fast O(1), because only need to record the training data;
   - No need to retrain the model if new training dataset is added;

   Cons:

   - Prediction takes O(N), which is not expected in machine learning , because we can accept slow training and want fast prediction;
   - Curse of dimensionality: we need a number of training examples which is k^d where d is the number of dimension of the data;
   - Knn is too easy which is a non-parametric model and has no objective function to converge. This means lot of information of the data cannot be used to do the classification;

The Second Algorithm: Decision Tree (C4.5)

1. Algorithm Flow

- For each feature, find all possible split branches. If the feature is categorical, the possible split branches are the unique value of this feature. If the feature is continuous or numeric, sort the values with respect to the feature and each possible branch of this feature occurs at two sequential values whose labels are different. We can take the average of these two sequential values as a threshold.
- Find the feature with highest information gain and find the split threshold with the highest information gain in this feature.
- Set the node as the threshold determined is the previous step and split the data into child node.
- Recursively call step2-3 until the following conditions to stop splitting the tree:
    1) Stop splitting when the node size is below the predefined minimum leaf size, which used to prevent overfitting;
    2) Stop splitting when the tree depth reaches to the predefined maximum tree depth, which used to prevent overfitting;
    3) Stop splitting when all labels are the same;
    4) Stop splitting when no more information gain;

2. Choose Hyperparameters

   As shown in Figure2.1, the best feature is the 20th for dataset1 and the 8th for dataset2. From the Figure2.2, Table2.1 and Table 2.2, the best tree depth is 4 in order to prevent overfitting. In order to prevent overfitting, We set the minimum leaf size equal to 20.

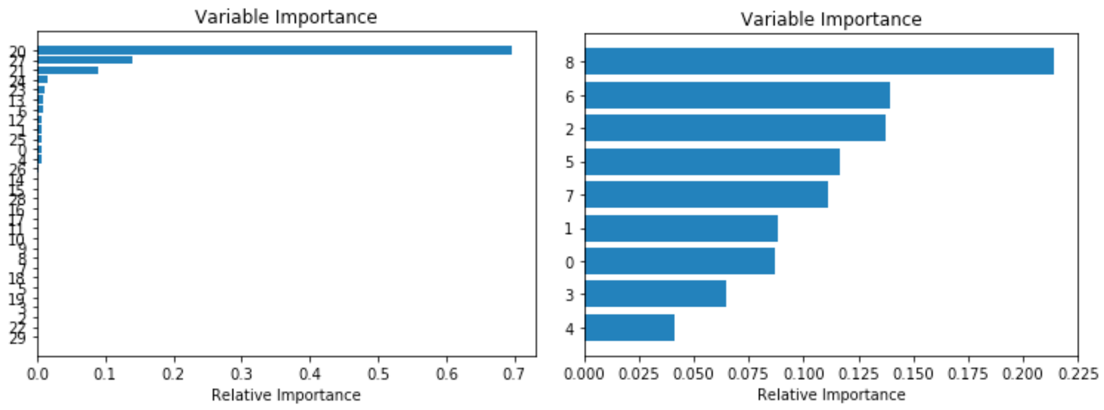


Figure2.1 Variable Importance (left: dataset1, right: dataset2)

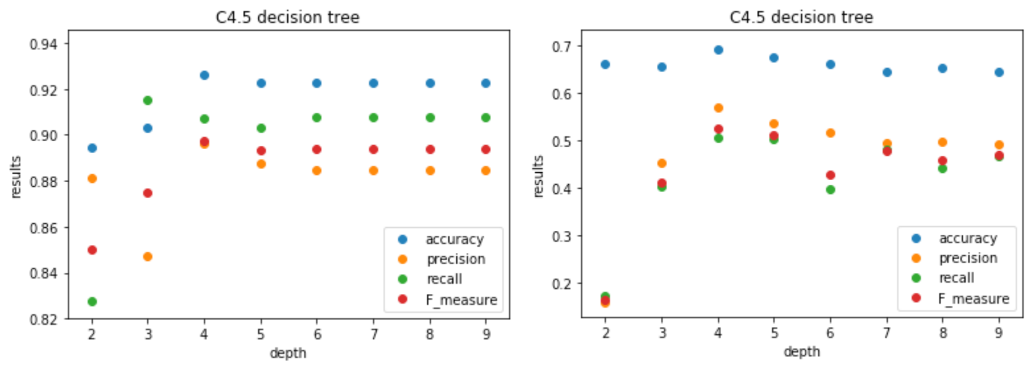


Figure2.2 Results of Four Performance Metrics (left: dataset1, right: dateset2)

3. Results Analysis

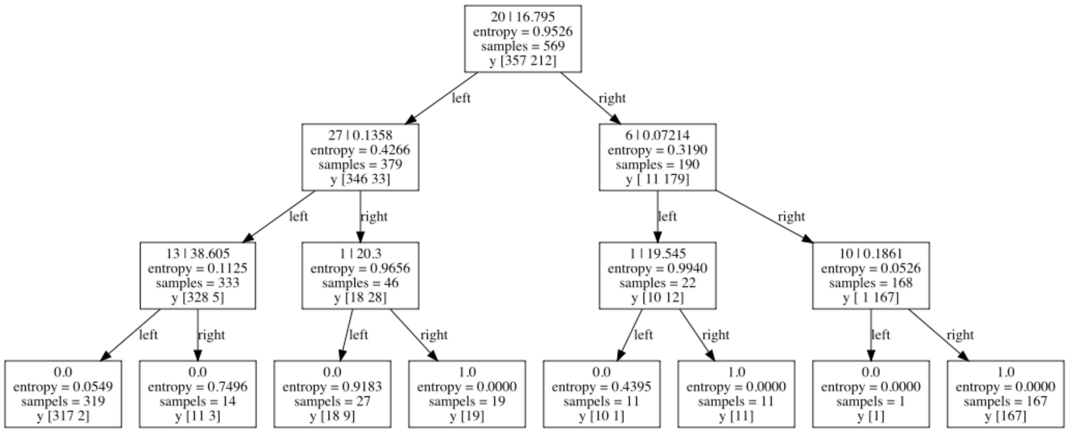
The results for the four performance metrics are summarized in Figure2.2, Table2.1 and Table2.2. The best tree depth is 4 for both dataset1 and dataset2. The decision trees are shown in Figure2.3 and Figure2.4. The results are from the average 10 folds cross-val.

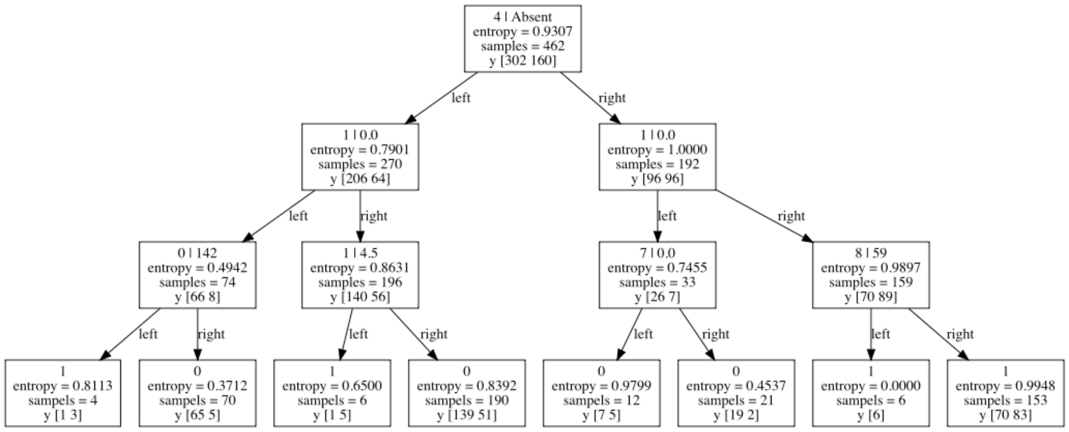
Table2.1 Results of Dataset1

Depth	2	3	4	5	6	7	8	9
Accuracy	0.895	0.903	0.926	0.923	0.923	0.923	0.923	0.923
Precision	0.881	0.847	0.896	0.887	0.884	0.884	0.884	0.884
Recall	0.827	0.915	0.907	0.903	0.908	0.908	0.908	0.908
F-measure	0.850	0.875	0.897	0.894	0.894	0.894	0.894	0.894

Table2.2 Results of Dataset2

Depth	2	3	4	5	6	7	8	9
Accuracy	0.660	0.656	0.691	0.673	0.660	0.645	0.652	0.643
Precision	0.159	0.452	0.570	0.534	0.516	0.493	0.497	0.492
Recall	0.172	0.403	0.506	0.501	0.396	0.481	0.442	0.466
F-measure	0.163	0.411	0.524	0.509	0.427	0.476	0.458	0.470


20 | 16.795 entropy = 0.9526 samples = 569 y [357 212]
left
right
27 | 0.1358 entropy = 0.4266 samples = 379 y [346 33]
6 | 0.07214 entropy = 0.3190 samples = 190 y [11 179]
13 | 38.605 entropy = 0.1125 samples = 333 y [328 5]
1 | 20.3 entropy = 0.9656 samples = 46 y [18 28]
1 | 19.545 entropy = 0.9940 samples = 22 y [10 12]
10 | 0.1861 entropy = 0.0526 samples = 168 y [1 167]
0.0 entropy = 0.0549 sampels = 319 y [317 2]
0.0 entropy = 0.7496 sampels = 14 y [11 3]
0.0 entropy = 0.9183 sampels = 27 y [18 9]
1.0 entropy = 0.0000 sampels = 19 y [19]
0.0 entropy = 0.4395 sampels = 11 y [10 1]
1.0 entropy = 0.0000 sampels = 11 y [11]
0.0 entropy = 0.0000 sampels = 1 y [1]
1.0 entropy = 0.0000 sampels = 167 y [167]
Figure2.3 Decision Tree of Dataset1


4 | Absent entropy = 0.9307 samples = 462 y [302 160]
left
right
1 | 0.0 entropy = 0.7901 samples = 270 y [206 64]
1 | 0.0 entropy = 1.0000 samples = 192 y [96 96]
0 | 142 entropy = 0.4942 samples = 74 y [66 8]
1 | 4.5 entropy = 0.8631 samples = 196 y [140 56]
7 | 0.0 entropy = 0.7455 samples = 33 y [26 7]
8 | 59 entropy = 0.9897 samples = 159 y [70 89]
1 entropy = 0.8113 sampels = 4 y [1 3]
0 entropy = 0.3712 sampels = 70 y [65 5]
1 entropy = 0.6500 sampels = 6 y [1 5]
0 entropy = 0.8392 sampels = 190 y [139 51]
0 entropy = 0.9799 sampels = 12 y [7 5]
0 entropy = 0.4537 sampels = 21 y [19 2]
1 entropy = 0.0000 sampels = 6 y [6]
1 entropy = 0.9948 sampels = 153 y [70 83]
Figure2.4 Decision Tree of Dataset2

4. Pros and Cons

   Pros:

   - The main advantage of decision tree is interpretability which can generate rules helping experts to formalize their knowledge;
   - The training is very efficient and the prediction is very fast;
   - High accuracy for the data can be separated by vertical and horizontal lines;
   - C4.5 can handle both categorical and continuous features;
   - C4.5 can handle noisy and incomplete data;
   - No need to do feature normalization;
   - Can use ensemble of decision tree;

   Cons:

   - Do not work well if the data have smooth boundaries;
   - Do not work well if the data have a lot of un-correlated features;
   - Need to be sensitive to overfitting;
   - High variance and unstable: as a result of the greedy strategy applied by decision tree, the right starting point of the tree can greatly impact the final results;

The Third algorithm: Naïve Bayes

1. Algorithm Flow
    - Compute prior P(y)
    - Compute conditional P(y|x)
    - Compute posterior P(y|x) = P(y|x)*P(y) / P(x)
    - Compare P(y|x)

    $$P(y=1|x) \propto P(x_1|y=1)P(x_2|y=1)\dots P(x_1|y=1)P(y=1)$$
    $$P(y=0|x) \propto P(x_1|y=0)P(x_2|y=0)\dots P(x_1|y=0)P(y=1)$$

2. Choose Hyperparameter:
    There is no hyperparameter in naïve bayes, but if the feature is continuous data we should assume the feature belong to Gaussian distribution and compute the conditional probability.

3. Results Analysis:
    The prediction in dataset1 is okay, because the features in dataset1 is relatively independent and each feature might close to Gaussian distribution. However, the model performs on dataset2 get very poor performance, because the features in dataset2 might be highly correlated and each continuous feature might not be Gaussian distribution. The assumption of Naïve Bayes is very strong, thus dataset2 doesn't satisfy the assumption obviously.

Table 3.1 Performance of Naïve Bayes (Average of 10 Folds)

Metrics	Dataset 1	Dataset 2
Accuracy	0.851	0.658
Precision	0.733	0.504
Recall	0.939	0.106
F-measure	0.819	0.136

4. Pros and Cons
    Pros:
    - Very simple and easy to implement and fast;
    - If the NB conditional independence assumption holds, then it will converge quicker than discriminative models like logistic regression;
    - Even the independence assumption doesn't hold, NB probably also can perform well;
    - Not sensitive to irrelevant features;

    Cons:
    - The assumption of NB is very strong;
    - For continuous features, you need to binning the feature or assume the feature belong to Gaussian distribution for the likelihoods;

The Forth Algorithm: Random Forest

1. Algorithm Flow
   - Predefine B, the number of tree you want to train
   - Given a training set $X = x_1, \dots, x_n$ with responses $Y = y_1, \dots, y_n$, bagging repeatedly (B times) selects n random sample with replacement of the training set. Also storing the out of bagging samples;
   - Using the B bags to train B trees:
     For b = 1,…,B
     1) Sample, with replacement, n training examples from *X, Y*; call these X_b, Y_b;
     2) Train a decision tree f_b on X_b, Y_b; and for the dataset with p features $\sqrt{p}$ (rounded down) features are used to split the tree;
     3) To predict an unlabeled example, using the majority voting among all the trees;
     4) Test the performance of the random forest by observing the out-of-bag error;

2. Choose Hyperparameters
   After many times of trials, we set the maximum depth of each tree to 8 (maxDepth=8) and the minimum leaf size of each tree to 10 (minLeafSize=10) on dataset 1. We set the maximum depth of each tree to 4 (maxDepth=4) and the minimum leaf size of each tree to 10 (minLeafSize=10) on dataset 2. The tree can be more complex than before in section2, because the ensemble model can reduce the variance as well as the bias. In order to determine the best forest size, we tried from 10 to 50 and the results are shown in Figure4.1, Table4.1 and Table4.2. We can see that the best performance occurs when we essemble 50 trees for dataset2.

3. Results Analysis
   Compared with section2 only using single decision, the random forest perform better with increasing the accuracy ~6% and precision ~17% on dataset2 when we use 50 trees, however the recall and F-measure decrease ~10%. Because the ensemble model, random forest, can reduce the variance as well as the bias.

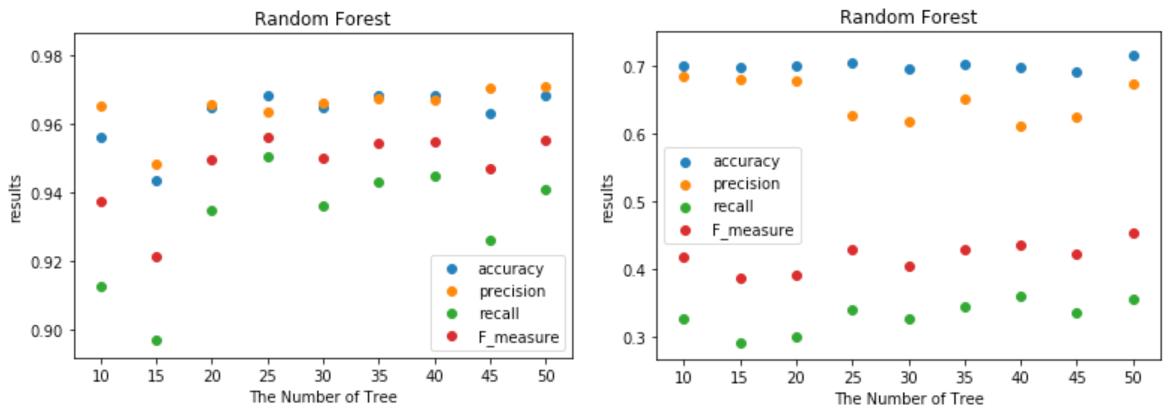


Figure4.1 Results of Four Performance Metrics (left: dataset1, right: dateset2)

Table 4.1 Results of Dataset1

forestSize	10	15	20	25	30	35	40	45	50
Accuracy	0.956	0.943	0.965	0.968	0.965	0.968	0.968	0.963	0.968
Precision	0.965	0.948	0.966	0.963	0.966	0.967	0.967	0.970	0.971
Recall	0.913	0.897	0.935	0.950	0.936	0.943	0.945	0.926	0.941
F-measure	0.938	0.921	0.950	0.956	0.950	0.954	0.955	0.947	0.955

Table 4.2 Results of Dataset2

forestSize	10	15	20	25	30	35	40	45	50
Accuracy	0.699	0.697	0.699	0.703	0.695	0.701	0.697	0.691	0.714
Precision	0.685	0.681	0.677	0.626	0.619	0.651	0.611	0.624	0.673
Recall	0.326	0.291	0.299	0.340	0.328	0.346	0.359	0.335	0.355
F-measure	0.418	0.386	0.391	0.430	0.403	0.428	0.435	0.421	0.454

4. Pros and Cons

Pros:

- It produces a highly accurate classifier;
- It can handle thousands of input features without feature deletion;
- It reduces overfitting and the variance of the model without increasing the bias;
- Training efficiently for very large datasets without cross-validation;

Cons:

- It may not work well when there is non-linear relationship between dependent and independent variables;
- A large number of trees may make the algorithm slow for prediction;

The Fifth Algorithm: Adaboost

1. Algorithm Flow:
   - Predefine M, the number of weak classifier you want;
   - Initially set uniform weights on all the training dataset;
   - For m = 1, … , M:
     1) Create a bootstrap sample based on the weights;
     2) Generate a simple classifier $f_m(X)$, i.e. a tree with depth 1;
     3) Train the simple classifier on the sample and apply it on the original training dataset;
     4) Compute the error rate:
        $$\varepsilon_m = \frac{\sum_{i=1}^{N} w_i 1(y_i \neq f_m(x_i))}{\sum_{i=1}^{N} w_i}$$
        if $\varepsilon_m < 0.5$, continue or start over;
     5) Compute the weight of the weak classifier $f_m(X)$,
        $$\alpha_m = 0.5 * \log\left(\frac{1-\alpha_m}{\alpha_m}\right)$$
     6) Update the example weights:
        The examples are correctly classified will have their weights increased;
        The examples are incorrectly classified will have their weights decreased;
        $$w_i = w_i \exp\left[-\alpha_m y_i f_m(x_i)\right]$$
     7) Normalize the example weights,
        $$w_i = w_i / \sum_{j=1}^{N} w_j$$
     8) Save α_m and $f_m(X)$
   - The prediction is weighted average of all the classifiers,
     $$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$
2. Choose Hyperparameters
   The only hyperparameter in Adaboost is the number of weak classifier M;
   As shown in Figure5.1 and Figure5.2, the model performance

3. Results Analysis
   From the Table 5.1-5.2 and Figure 5.1, we can see that Adaboost perform good on dataset 1, but the performance on dataset 2 is poor although it's better than only use decision tree.

Table 5.1 Results of Dataset 1

#weak classifier	10	20	30	40	50	60	70	80	90	100
Accuracy	0.942	0.956	0.953	0.956	0.963	0.965	0.967	0.963	0.974	0.972
Precision	0.926	0.965	0.953	0.960	0.963	0.977	0.976	0.977	0.982	0.991
Recall	0.913	0.922	0.918	0.926	0.937	0.931	0.936	0.925	0.944	0.935
F-means	0.919	0.940	0.934	0.941	0.949	0.952	0.955	0.949	0.961	0.961

Table 5.2 Results of Dataset 2

#weak classifier	10	20	30	40	50	60	70	80	90	100
Accuracy	0.721	0.703	0.695	0.706	0.706	0.699	0.690	0.684	0.690	0.688
Precision	0.637	0.591	0.578	0.589	0.593	0.583	0.566	0.561	0.572	0.560
Recall	0.497	0.488	0.524	0.513	0.529	0.501	0.483	0.477	0.500	0.502
F-means	0.540	0.519	0.529	0.537	0.549	0.527	0.508	0.495	0.519	0.518

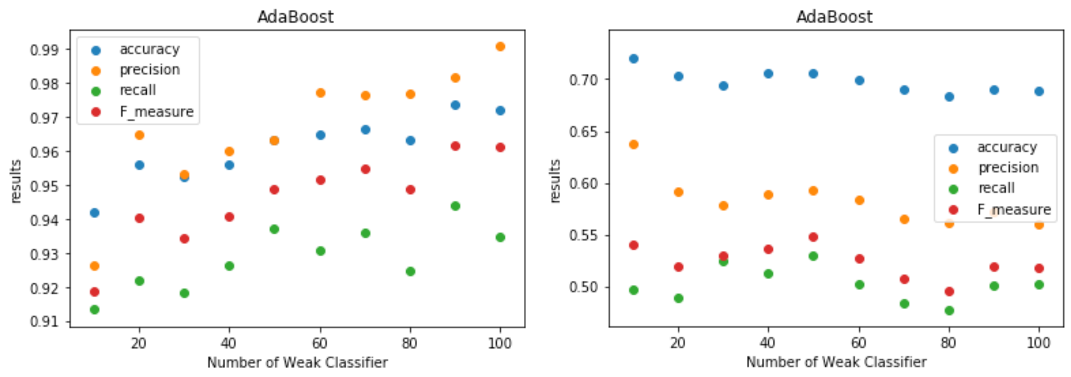


Figure5.1 Results of Four Performance Metrics (left: dataset1, right: dateset2)

4. Pros and Cons

   Pros:

- Adaboost is very powerful even better than random forest;
- Easy to use compared with SVM;
- Adaboost has strong theoretical derivation;
- Prevent overfitting and fairly good generalization;
- Due to Adaboost combines many weak classifier, the training is fast;

   Cons:

- Adaboost can be sensitive to noisy data and outliers and overfit in presence of noise;
- Suboptimal solution for α_m;

Cross Validation:

In this project, we use 10 folds cross validation to test the performance of K-nn, C4.5, Naïve Bayes, Random Forest and Adaboost. We compute the average of the 10 folds to test the performance for specific hyperparameter.

Reference:

http://www.nickgillian.com/wiki/pmwiki.php/GRT/RandomForests
https://en.wikipedia.org/wiki/Random_forest
http://www.robots.ox.ac.uk/~az/lectures/cv/adaboost_matas.pdf