Introduction to Machine Learning Program Assignment #1

Members:
0510002 袁鈺勛
0510020 方鈺豪
0510022 劉孟寰
0510023 李佳任
0086043 陳以嫣

(1).

What environments the members are using

Ans:

For both Iris dataset and googleplay dataset, we use python to implement Machine Learning.

Through the process, we install two packages, SKlearn and Pandas.

SKlearn package is for implement the decision tree model and random forest, and for scoring accuracy, precision and recall.

Pandas package is for the reading the CSV file.

Both packages are useful in preprocessing the data.

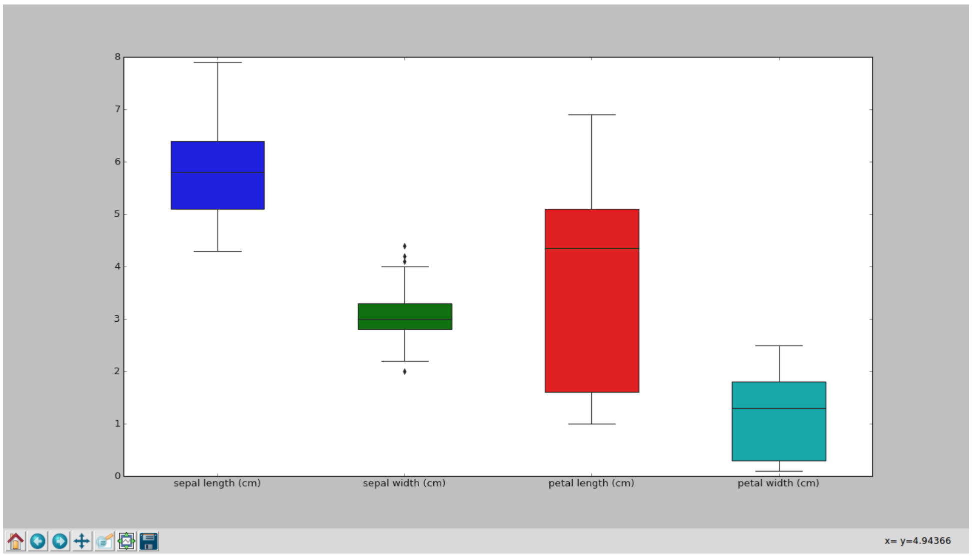
(2)

Basic statistics visualization of the data

Ans:

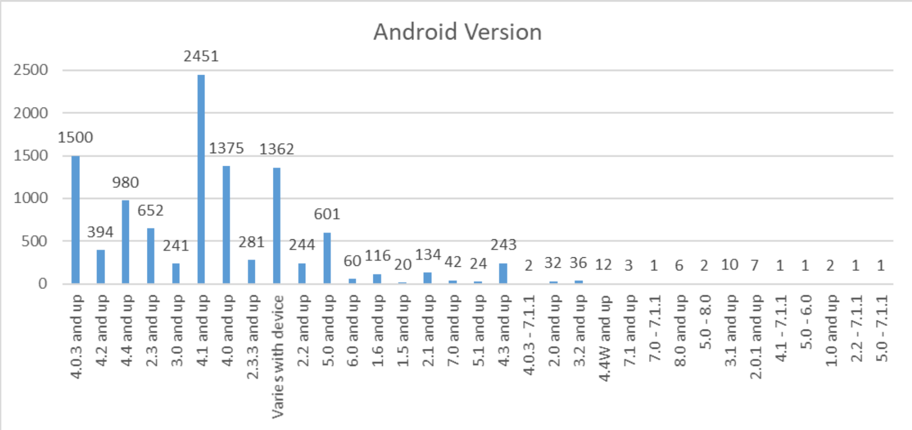
Iris dataset:

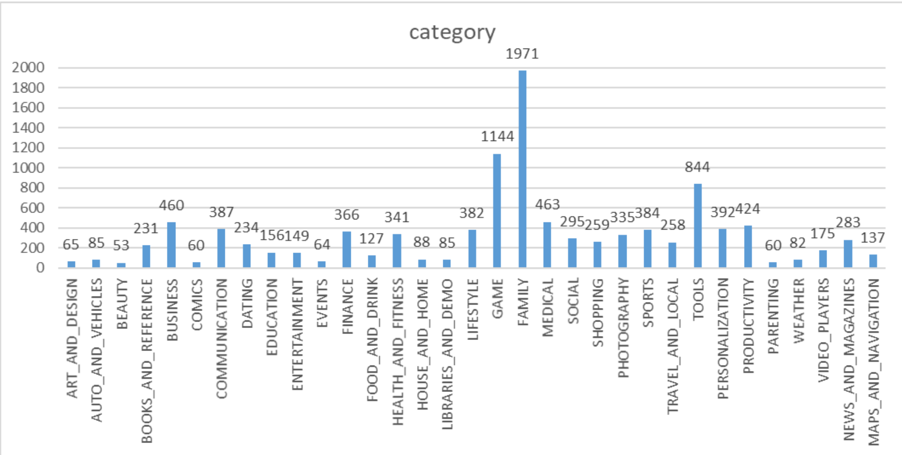
We use the boxplot to present basic statistic visualization.

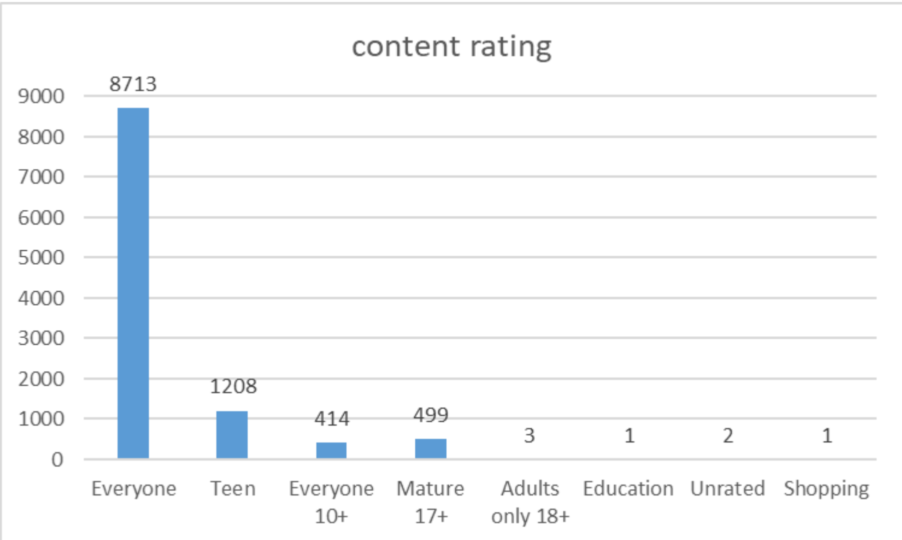


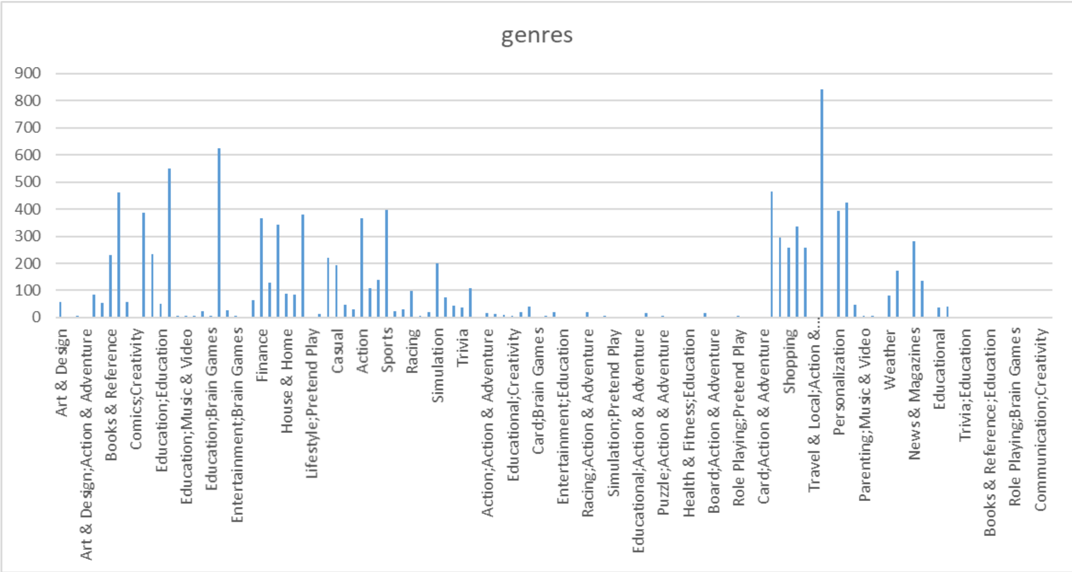
Googleplay dataset:

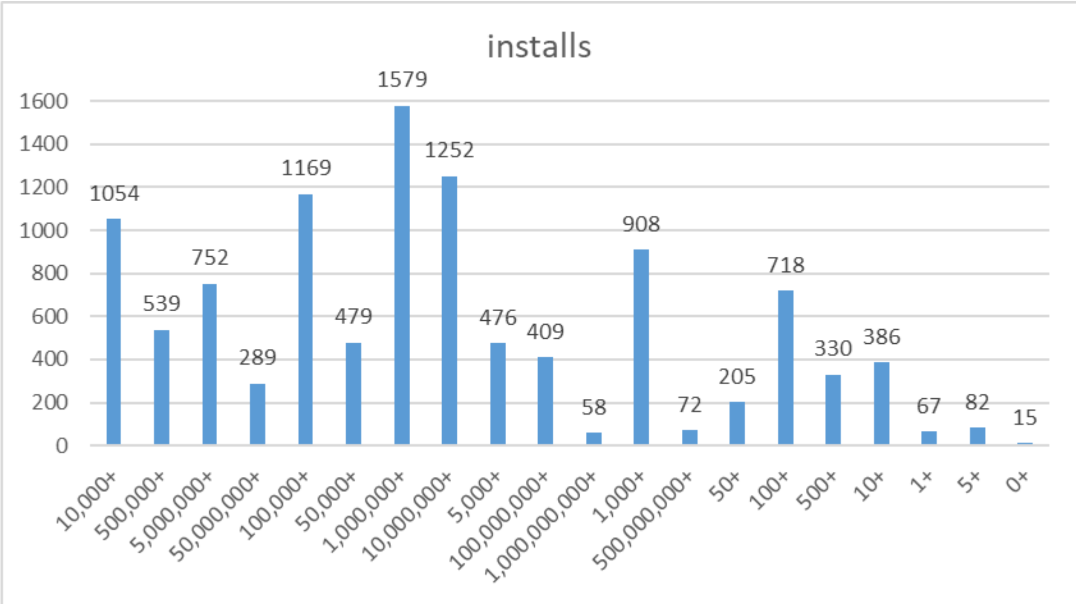
For the features which are not in numeral form, such as Android version, category, content rating,genres,installs and type, we use the bar plot to show the number of each classification in every features.

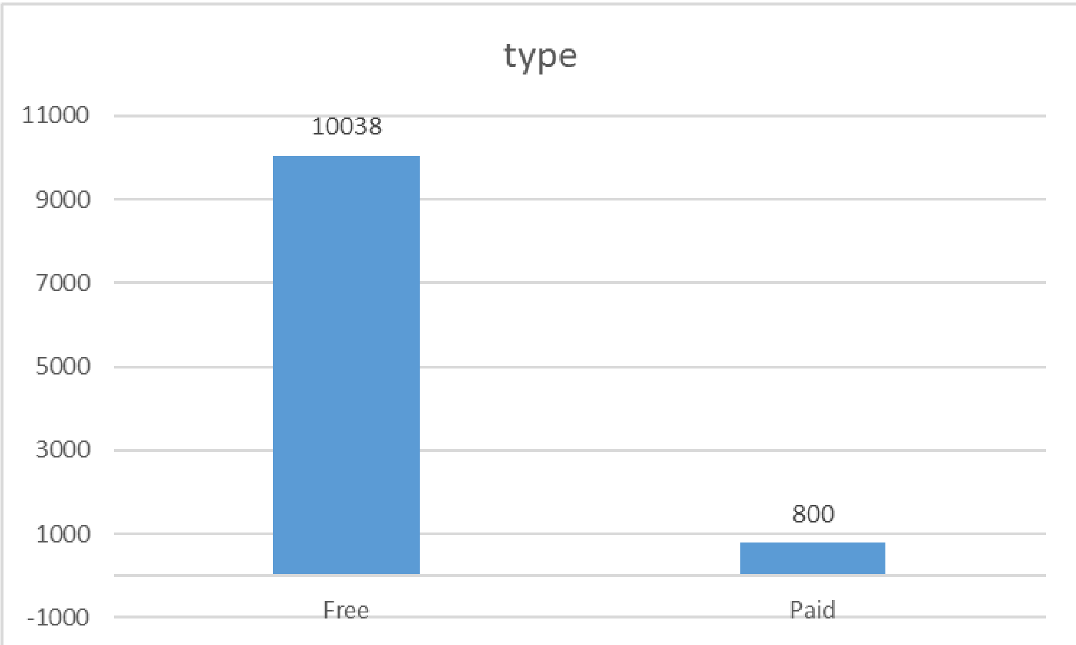




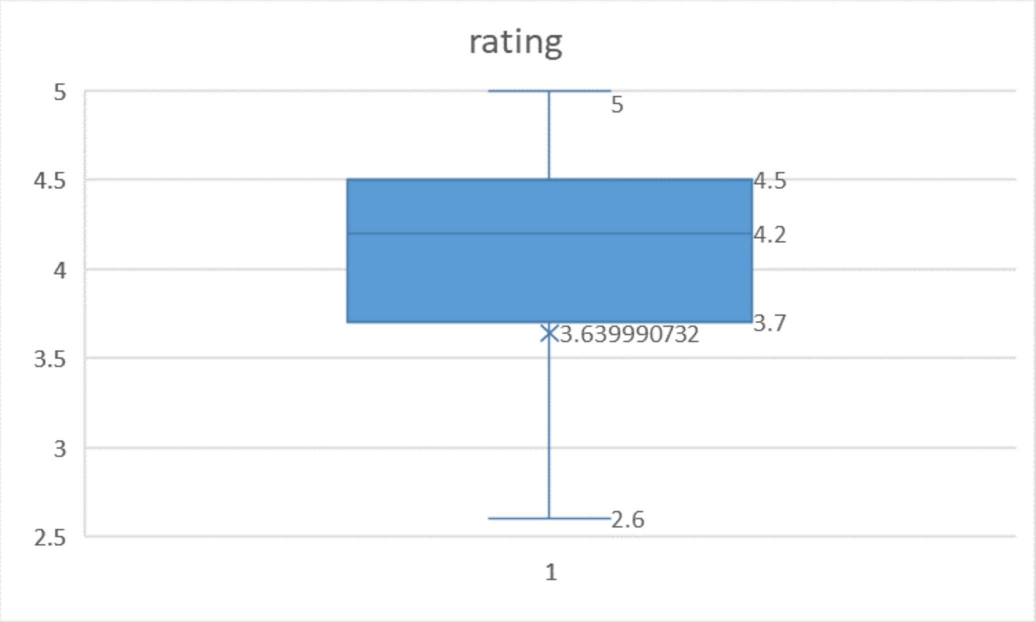


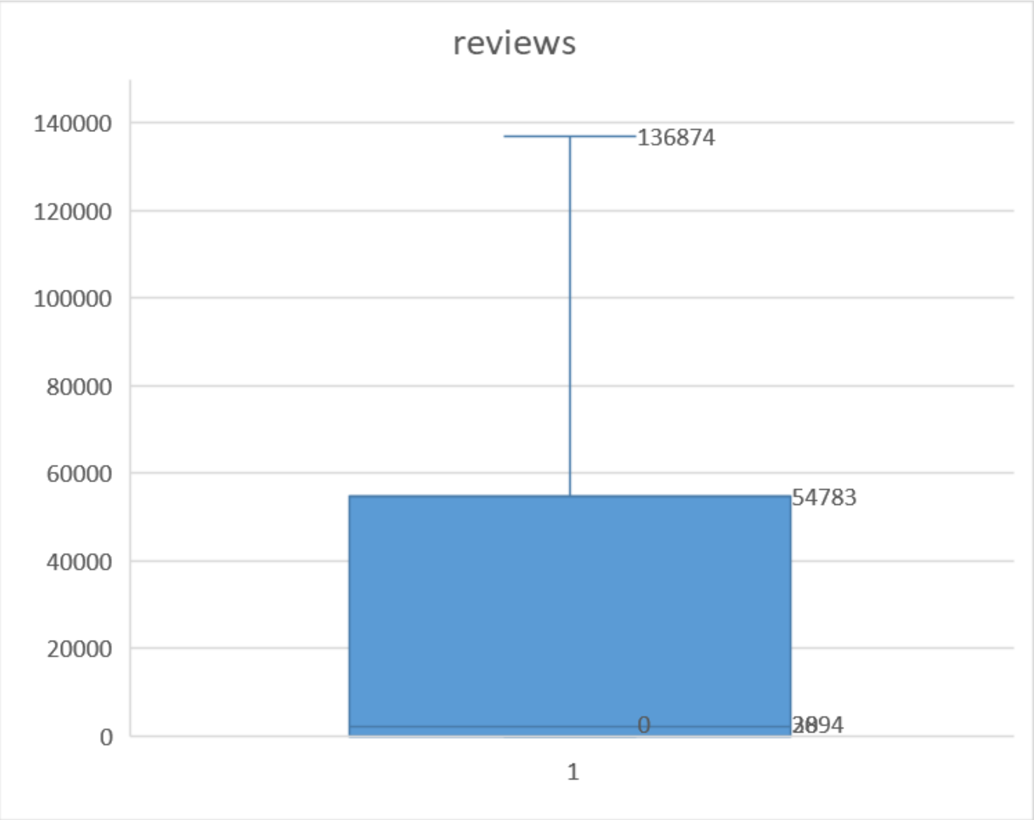

genres


installs
1054
539
752
289
1169
479
1579
1252
476
409
58
908
72
205
718
330
386
67
82
15
10,000+
500,000+
5,000,000+
50,000,000+
100,000+
50,000+
1,000,000+
10,000,000+
5,000+
100,000,000+
1,000,000,000+
1,000+
500,000,000+
50+
100+
500+
10+
1+
5+
0+


type
10038
800
Free
Paid

For the features which are in numeral form, such as reviews and rating, we use the box plot to present basic statistic visualization.





(Ps: Because the distribution of data of the reviews is too wide, the minimum and Q1 are almost invisible.)

(3)

Data preprocessing methods

Ans:

Iris dataset:

```
# Preprocess data
    input_data.replace(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], [0, 1, 2], inplace = True)
    iris_data = input_data.drop("class", axis = 1)
    iris_target = input_data['class']
```

We use data.replace to transfer Iris category into integers and then split the data into features and targets base on columns

Googleplay dataset:

```
# Transfer the data to make it readable for DecisionTreeClassifier
    le = preprocessing.LabelEncoder()
    input_data['Category'] = le.fit_transform(input_data['Category'])
    input_data['Type'] = le.fit_transform(input_data['Type'])
    input_data['Content Rating'] = le.fit_transform(input_data['Content Rating'])
    input_data['Android Ver'] = le.fit_transform(input_data['Android Ver'])
    input_data.replace(['0', '0+', '1+', '5+', '10+', '50+'], 0, inplace = True)
    input_data.replace(['100+', '500+', '1,000+', '5,000+', '10,000+'], 1, inplace = True)
    input_data.replace(['50,000+', '100,000+', '500,000+', '1,000,000+', '5,000,000+'], 2,
inplace = True)
    input_data.replace(['10,000,000+', '50,000,000+', '100,000,000+', '500,000,000+',
'1,000,000,000+'], 3, inplace = True)

    #get "Installs" column and set it as target
    target = input_data['Installs']
    data = input_data.drop("Installs", axis = 1)
```

Because there are too many labels in each column, we use LabelEncoder to encode string-type label into integers, also we split the data into features and targets base on columns.

By the way, there’s too many kinds of ‘Installs’, which is our target, which will cause our accuracy pretty low, so we accumulated part of them. That is, there are 20 kinds of Installs first, after our preprocessing, there only left four, whick will make our prediction more reasonable and ideal.

(4)

How do you generate decision tree and random forest models

Ans:

For both dataset, we use the same methods:

```
# Get tree's prediction
def getTreePrediction(train, test, target):
    clf = tree.DecisionTreeClassifier()
    plot = clf.fit(train, target) # Generate the tree
    predict = clf.predict(test) # Generate prediction
    return plot, predict
# End

# Get forests' prediction
def getForestPrediction(train, test, target):
    predict = []
    for i in range(11):
        clf = tree.DecisionTreeClassifier()
        n_features = np.random.randint(4, size = np.random.randint(low = 2, high = 5)) # Generate random features
        clf.fit(train.iloc[:, n_features], target) # Generate the tree
        predict.append(clf.predict(test.iloc[:, n_features]))
    return getFinalPrediction(predict)
# End
```

For decision tree, we simply used a package in scikit-learn, ’DecisionTreeClassifier’ to build a decision tree, we can get the tree by feed the classifier with our features and targets.

For random forest, we define a function to randomly select some features in our total features to build a tree, and use for loop in range(k) times to build k trees in our random forest.

By the way, for Iris dataset, we built trees in the forest with 2~4 features and the result is pretty ideal, but that does not do the same performance for Googleplay dataset because the data is too large, if we choose too less features to buil a tree, it may be really useless and misleading, so for Googleplay dataset, we randomly select 4~6 features to build a tree in our random forest.

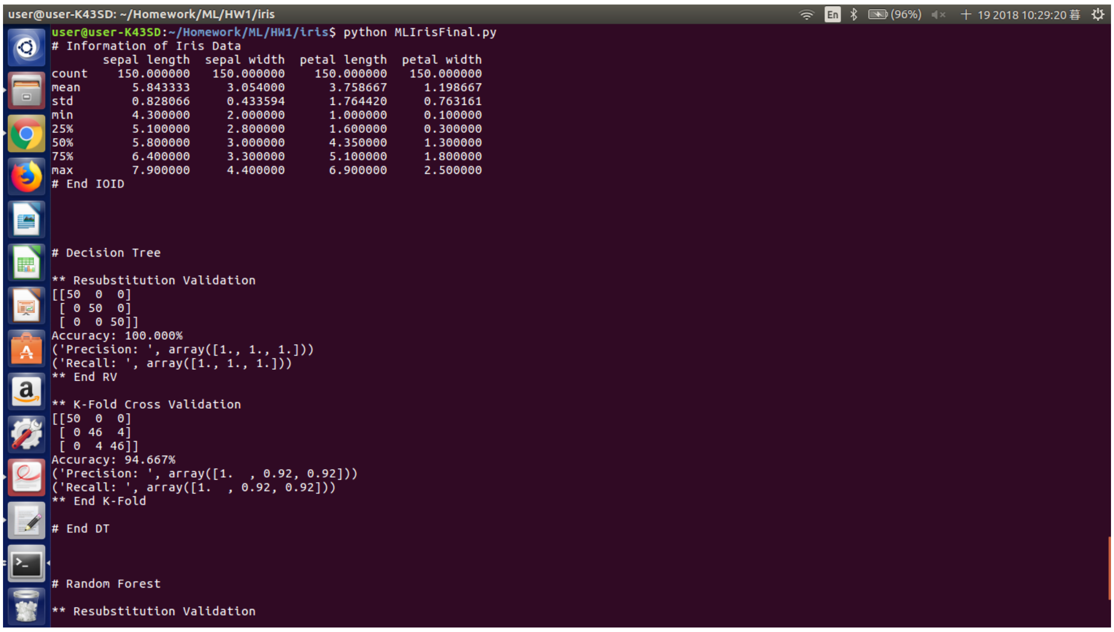
(5)

Performance

Ans:

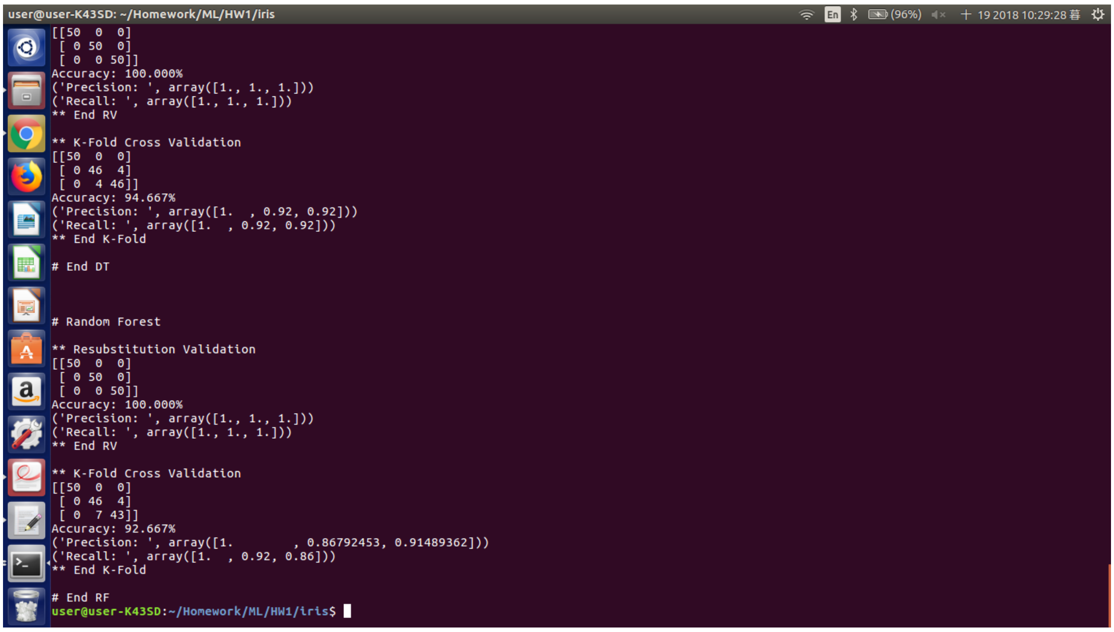
Iris:

1.Decision tree



```
user@user-K43SD:~/Homework/ML/HW1/iris$ python MLIrisFinal.py
# Information of Iris Data
       sepal length  sepal width  petal length  petal width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
# End IOID



# Decision Tree

** Resubstitution Validation
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
Accuracy: 100.000%
('Precision: ', array([1., 1., 1.]))
('Recall: ', array([1., 1., 1.]))
** End RV

** K-Fold Cross Validation
[[50  0  0]
 [ 0 46  4]
 [ 0  4 46]]
Accuracy: 94.667%
('Precision: ', array([1.  , 0.92, 0.92]))
('Recall: ', array([1.  , 0.92, 0.92]))
** End K-Fold

# End DT



# Random Forest

** Resubstitution Validation
```

2.Random forest

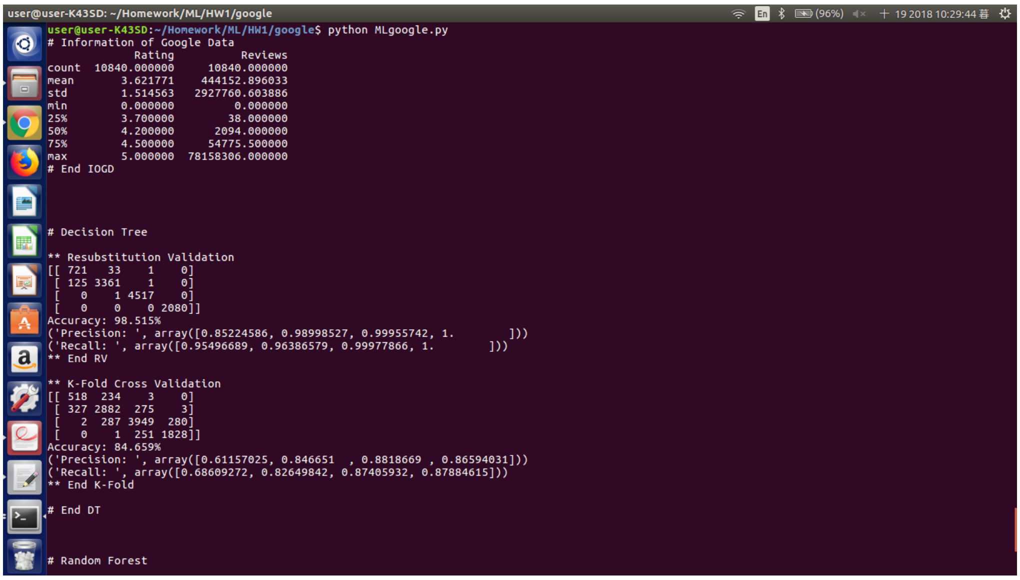


```
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
Accuracy: 100.000%
('Precision: ', array([1., 1., 1.]))
('Recall: ', array([1., 1., 1.]))
** End RV

** K-Fold Cross Validation
[[50  0  0]
 [ 0 46  4]
 [ 0  4 46]]
Accuracy: 94.667%
('Precision: ', array([1.  , 0.92, 0.92]))
('Recall: ', array([1.  , 0.92, 0.92]))
** End K-Fold

# End DT



# Random Forest

** Resubstitution Validation
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
Accuracy: 100.000%
('Precision: ', array([1., 1., 1.]))
('Recall: ', array([1., 1., 1.]))
** End RV

** K-Fold Cross Validation
[[50  0  0]
 [ 0 46  4]
 [ 0  7 43]]
Accuracy: 92.667%
('Precision: ', array([1.        , 0.86792453, 0.91489362]))
('Recall: ', array([1.  , 0.92, 0.86]))
** End K-Fold

# End RF
user@user-K43SD:~/Homework/ML/HW1/iris$
```

For Iris dataset, both decision tree and random forest predictions are ideal, all accuracies, precisions, and recalls are very high. We thought that’s because the dataset is small and the features are petty high related to our targets.

Googleplay

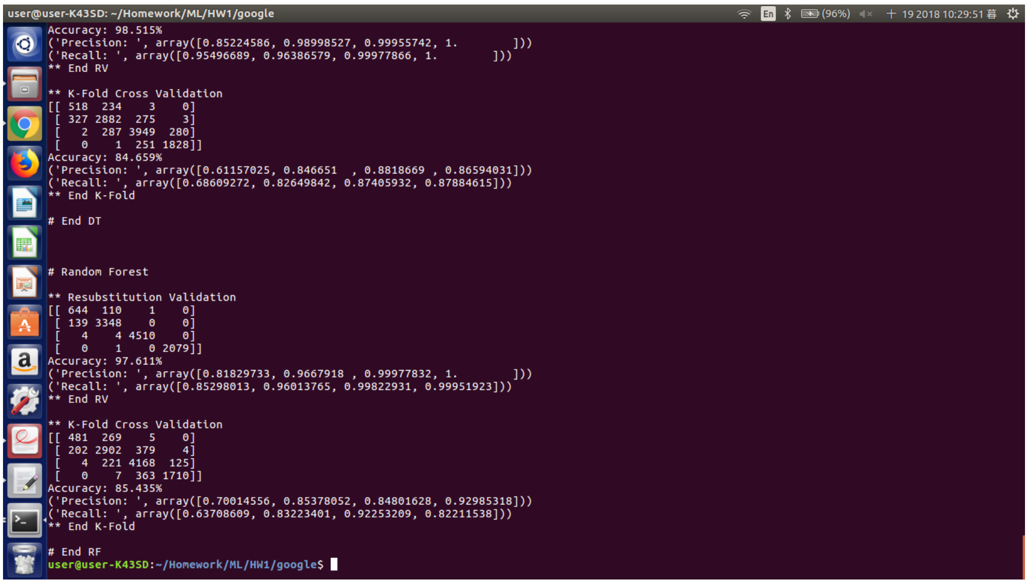
1.Decision tree



```
user@user-K43SD:~/Homework/ML/HW1/google$ python MLgoogle.py
# Information of Google Data
              Rating          Reviews
count  10840.000000     10840.000000
mean       3.621771    444152.896033
std        1.514563   2927760.603886
min        0.000000         0.000000
25%        3.700000        38.000000
50%        4.200000      2094.000000
75%        4.500000     54775.500000
max        5.000000  78158306.000000
# End IOGD



# Decision Tree

** Resubstitution Validation
[[ 721   33    1    0]
 [ 125 3361    1    0]
 [   0    1 4517    0]
 [   0    0    0 2080]]
Accuracy: 98.515%
('Precision: ', array([0.85224586, 0.98998527, 0.99955742, 1.        ]))
('Recall: ', array([0.95496689, 0.96386579, 0.99977866, 1.        ]))
** End RV

** K-Fold Cross Validation
[[ 518  234    3    0]
 [ 327 2882  275    3]
 [   2  287 3949  280]
 [   0    1  251 1828]]
Accuracy: 84.659%
('Precision: ', array([0.61157025, 0.846651  , 0.8818669 , 0.86594031]))
('Recall: ', array([0.68609272, 0.82649842, 0.87405932, 0.87884615]))
** End K-Fold

# End DT



# Random Forest
```

Resubstitution part is about 98% accuracy and each fold(average) in K-fold validation is about 80~90% accuracy, which is better than we thought because the dataset is too large and features are not so related to our target. Also, reducing the category of our target from 20→4(mentioned in preprocessing step) really helps because we first did not do this and get pretty low accuracy.

2.random forest:



```
Accuracy: 98.515%
('Precision: ', array([0.85224586, 0.98998527, 0.99955742, 1.        ]))
('Recall: ', array([0.95496689, 0.96386579, 0.99977866, 1.        ]))
** End RV

** K-Fold Cross Validation
[[ 518  234    3    0]
 [ 327 2882  275    3]
 [   2  287 3949  280]
 [   0    1  251 1828]]
Accuracy: 84.659%
('Precision: ', array([0.61157025, 0.846651  , 0.8818669 , 0.86594031]))
('Recall: ', array([0.68609272, 0.82649842, 0.87405932, 0.87884615]))
** End K-Fold

# End DT



# Random Forest

** Resubstitution Validation
[[ 644  110    1    0]
 [ 139 3348    0    0]
 [   4    4 4510    0]
 [   0    1    0 2079]]
Accuracy: 97.611%
('Precision: ', array([0.81829733, 0.9667918 , 0.99977832, 1.        ]))
('Recall: ', array([0.85298013, 0.96013765, 0.99822931, 0.99951923]))
** End RV

** K-Fold Cross Validation
[[ 481  269    5    0]
 [ 202 2902  379    4]
 [   4  221 4168  125]
 [   0    7  363 1710]]
Accuracy: 85.435%
('Precision: ', array([0.70014556, 0.85378052, 0.84801628, 0.92985318]))
('Recall: ', array([0.63708609, 0.83223401, 0.92253209, 0.82211538]))
** End K-Fold

# End RF
user@user-K43SD:~/Homework/ML/HW1/google$
```

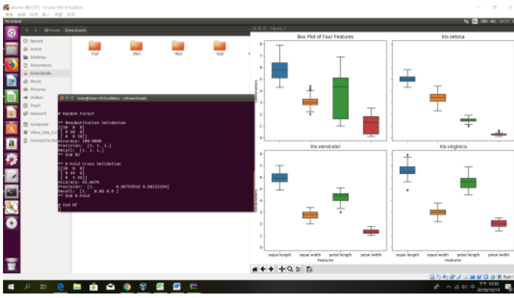
Resubstitution part is about 97% accuracy and each fold(average) in K-fold validation is about 80~90% accuracy, that means our random forest does not help a lot for our accuracy. Maybe it’s becease we drop some important features in some trees in the random forest and make that tree misleading, but the accuracy is still pretty high,so it’s accetable.

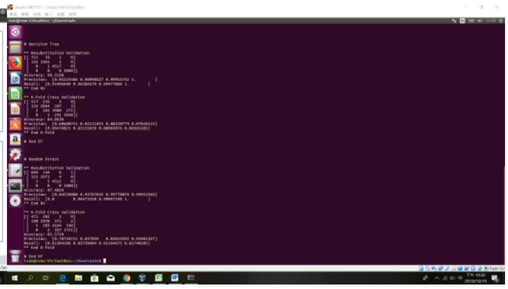
(6)Conclusion

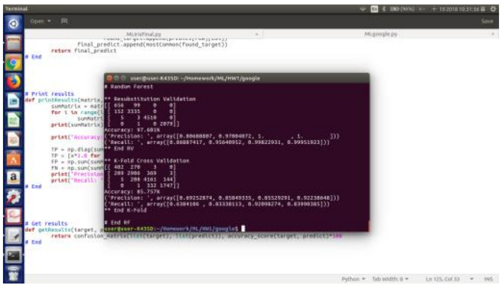
In this project we face a lot of problem, especially for googleplay dataset, so we experiences many times to get the conclusion below:

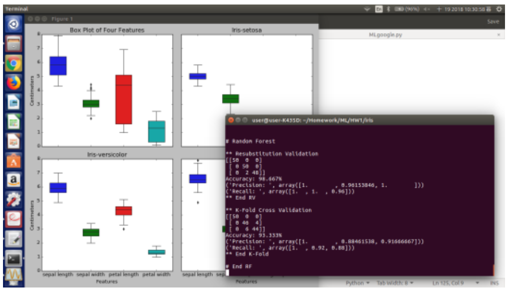
1. The dataset is too large and our target is too diverse. So finally we merge some sorts of our target to decrease the diversity to make the result more reasonable.
2. Before we build a random forest, we can test the relativity between each two features(target) to choose a better target to predict, also we can think how to avoid building useless tree in the random forest by this step.
3. When the dataset is large, we should not put too less features to the classifier to build a tree, it may cause low fidelity.
4. When doing k-fold validation, we should keep a appropriate rate betwwn training data and testing data, if the rate is disordered(like1:1), the accuracy will be pretty bad because of training too less and testing too much.
5. In this project we've learn a lot of knowledge and skills about maching learning and we believe that we can do much better next time.

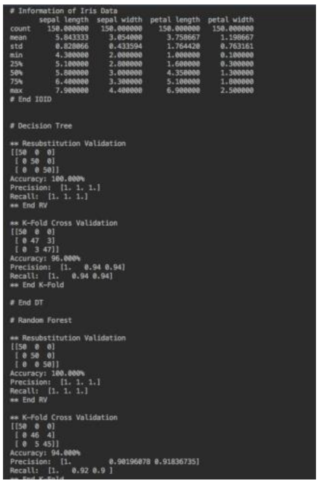
截圖：



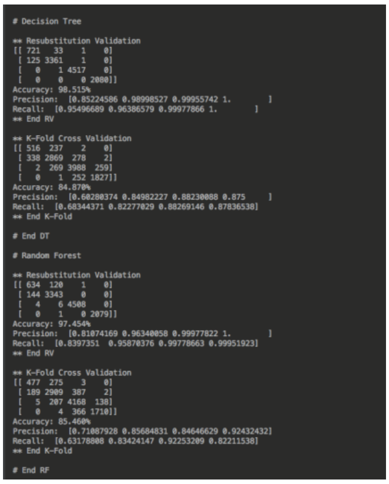








```
# Information of Iris Data
       sepal length  sepal width  petal length  petal width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
# End IOID


# Decision Tree

** Resubstitution Validation
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
Accuracy: 100.000%
Precision:  [1. 1. 1.]
Recall:  [1. 1. 1.]
** End RV

** K-Fold Cross Validation
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
Accuracy: 96.000%
Precision:  [1.   0.94 0.94]
Recall:  [1.   0.94 0.94]
** End K-Fold

# End DT

# Random Forest

** Resubstitution Validation
[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
Accuracy: 100.000%
Precision:  [1. 1. 1.]
Recall:  [1. 1. 1.]
** End RV

** K-Fold Cross Validation
[[50  0  0]
 [ 0 46  4]
 [ 0  5 45]]
Accuracy: 94.000%
Precision:  [1.         0.90196078 0.91836735]
Recall:  [1.   0.92 0.9 ]
```



```
# Decision Tree

** Resubstitution Validation
[[ 721   33    1    0]
 [ 125 3361    1    0]
 [   0    1 4517    0]
 [   0    0    0 2080]]
Accuracy: 98.515%
Precision:  [0.85224586 0.98998527 0.99955742 1.        ]
Recall:  [0.95496689 0.96386579 0.99977866 1.        ]
** End RV

** K-Fold Cross Validation
[[ 516  237    2    0]
 [ 338 2869  278    2]
 [   2  269 3988  259]
 [   0    1  252 1827]]
Accuracy: 84.870%
Precision:  [0.60280374 0.84982227 0.88230088 0.875     ]
Recall:  [0.68344371 0.82277029 0.88269146 0.87836538]
** End K-Fold

# End DT

# Random Forest

** Resubstitution Validation
[[ 634  120    1    0]
 [ 144 3343    0    0]
 [   4    6 4508    0]
 [   0    1    0 2079]]
Accuracy: 97.454%
Precision:  [0.81074169 0.96340058 0.99977822 1.        ]
Recall:  [0.8397351  0.95870376 0.99778663 0.99951923]
** End RV

** K-Fold Cross Validation
[[ 477  275    3    0]
 [ 189 2909  387    2]
 [   5  207 4168  138]
 [   0    4  366 1710]]
Accuracy: 85.460%
Precision:  [0.71087928 0.85684831 0.84646629 0.92432432]
Recall:  [0.63178808 0.83424147 0.92253209 0.82211538]
** End K-Fold

# End RF
```