School of Data and Computer Science, Sun Yat-sen University

# Random Forest

18308045 谷正阳

June 13, 2021

## Contents

# 1 Importance function

This function is used to evaluate how good a split of dataset is. The basic idea is that if a label in each subset after a split dominates that subset, the split is good.

## 1.1 Purity function

To evaluate whether a label dominates a subset, we need to build a special function. Since we do binary classification here, we just input the probability of one label in the subset into the function. The return value should be high if the probability is close to 0 or 1. Otherwise, the value should be low. There are two common functions to evaluate the purity of a set, one of which is entropy, the other of which is negtive Gini index. Entropy has the form of

$$B(q) = -(q\log(q) + (1-q)\log(1-q)), \tag{1}$$

while negtive Gini index has the form of

$$Neg\_Gini(p) = -(1-(p^2+(1-p^2))). \tag{2}$$

Based on these two purity functions, we build three importance functions.

## 1.2 Information gain

Information gain is to calculate the gain of entropy after the split. It has the form of

$$Gain(A) = B(\frac{p}{p+n}) - \sum_{v=1}^{V}\frac{p_v+n_v}{p+n}B(\frac{p_v}{p_v+n_v}). \tag{3}$$

Because after the discretization every attributes' domains are binary, and $p, p+n$ are the same when comparing splits, the actual equation used here is

$$Neg\_Remainder\_Mul\_N(A) = (p_0+n_0)(-B(\frac{p_0}{p_0+n_0})) + (p_1+n_1)(-B(\frac{p_1}{p_1+n_1})). \tag{4}$$

The function NEG_REMAINDER_MUL_N below takes all the features as inputs and returns an array containing all their $Neg\_Remainder\_Mul\_N$.

```
def NEG_B(q):
    I_q = 1 - q
    return xlogy(q, q) + xlogy(I_q, I_q)

def NEG_REMAINDER_MUL_N(S, F):
    label = S[:, -1:]
```

```
    S1 = S[:, F]

    p1 = np.sum(S1 & label)
    N1 = np.sum(S1)

    p0 = np.sum(label) - p1
    N0 = S.shape[0] - N1

    return N1 * NEG_B(p1 / N1) + N0 * NEG_B(p0 / N0)
```

### 1.3 Information gain ratio

The information gain3 has a weakness, which is that if the domain of an attribute is too big, the second term $\sum_{v=1}^{V} \frac{p_v+n_v}{p+n} B(\frac{p_v}{p_v+n_v})$ can be really small. Therefore, the information gain prefer those attributes with larger dommain. The information gain ratio has the form of

$$Gain\_ratio(A) = \frac{Gain(A)}{-\sum_{v=1}^{V} \frac{p_v+n_v}{p+n} \log(\frac{p_v+n_v}{p+n})}, \tag{5}$$

which is actually add a penalty on the information gain3. If the domain of an attribute is large, the $-\sum_{v=1}^{V} \frac{p_v+n_v}{p+n} \log(\frac{p_v+n_v}{p+n})$ will be large, so its gain ratio is small.

However, since all the attributes here have binary domains, the information gain ratio is the same as the information gain.

### 1.4 Negative Gini gain

This has a similar form as the information gain3, since it's actually replace the $\log(p)$ with its approximate substitution $p-1$. This can be calculated faster than information gain since $p-1$ is easier to calculate than $\log(p)$.

```
def NEG_GINI_MINUS_1(p):
    return p ** 2 + (1 - p) ** 2

def NEG_GINI_INDEX_MINUS_1_MUL_N(S, F):
    label = S[:, -1:]
    S1 = S[:, F]

    p1 = np.sum(S1 & label)
    N1 = np.sum(S1)

    p0 = np.sum(label) - p1
```

```
    N0 = S.shape[0] - N1

    return N1 * NEG_GINI_MINUS_1(p1 / N1) + N0 * NEG_GINI_MINUS_1(p0 / N0)
```

# 2 Discretization

Since the data preprocessed in the last part of project are all continuous value. And the decision tree requires those attributes to be discretized. This part is implemented in function read. The function is used to find the best split point to split the domain of an attribute into 2 labels.

## 2.1 The first step: to calculate all split points

To discretize an attibute, firstly, we build an array containing all values appear in the attribute. And we sort it and remove all the duplicate values in it. Now we get a sorted domain of the attributes. Then we can use matrix operation to generate an array containing all split points.

```
mids = X[:, f]
np.sort(mids)
mids = np.unique(mids)
mids = (mids[:-1] + mids[1:]) / 2
```

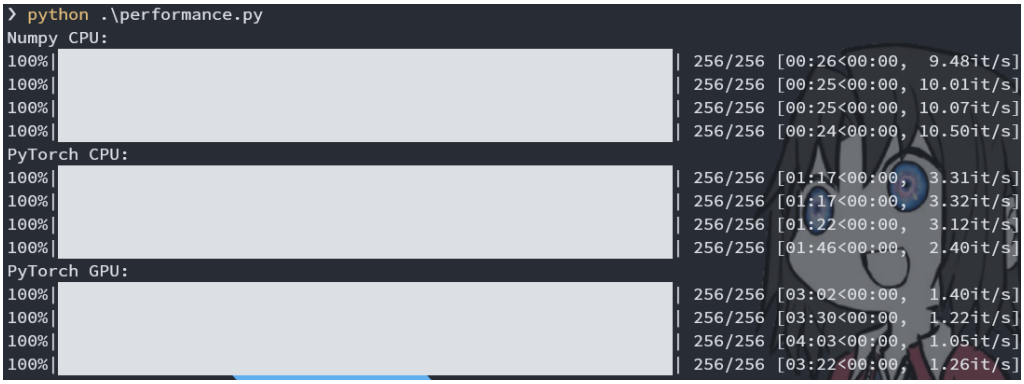
## 2.2 The second step: to choose the best split point

We can get scores of split points using importance function defined above. and choose the split with highest importance.

```
x_best = None
mid_best = None
score_best = -float("inf")
for mid in mids:
    x = (X[:, f:f + 1] >= mid)
    score = IMPORTANCE(np.concatenate((x, Y), axis=1), np.array([0]))
    if score_best < score:
    x_best = x
    mid_best = mid
    score_best = score
X[:, f:f + 1] = x_best
threshold[f] = mid_best
```

### 2.3 Comparison between Numpy version and PyTorch version

Since there are some matrix operation, it seems like the calculation can be speeded up by CUDA which is supported by PyTorch but not supported by NumPy. Therefore, I implement discretization in 2 versions and compare their performance.




The result shows that neither PyTorch with CPU nor PyTorch with GPU is faster than Numpy. The reason may be that there are lots of loop operation which performs worse on PyTorch than on Numpy.

## 3 Experiment

I divide the output into 4 outputs like what I do in the last project. Therefore I need to build 4 random forests in each model.

### 3.1 Different importance functions

I test different importance functions. Since we have discussed that information gain ratio and information gain are the same before, we just test the performance of information gain and negative Gini index gain.

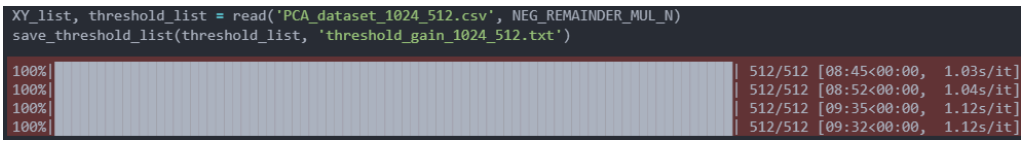



Figure 1: Discretization using information gain

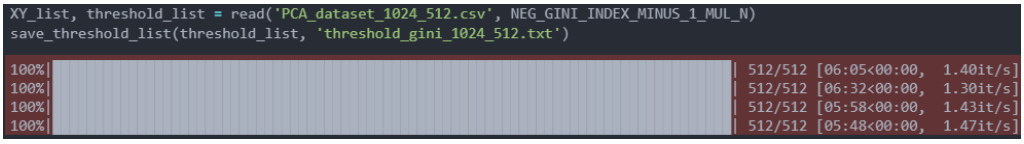



Figure 2: Discretization using negative Gini index gain

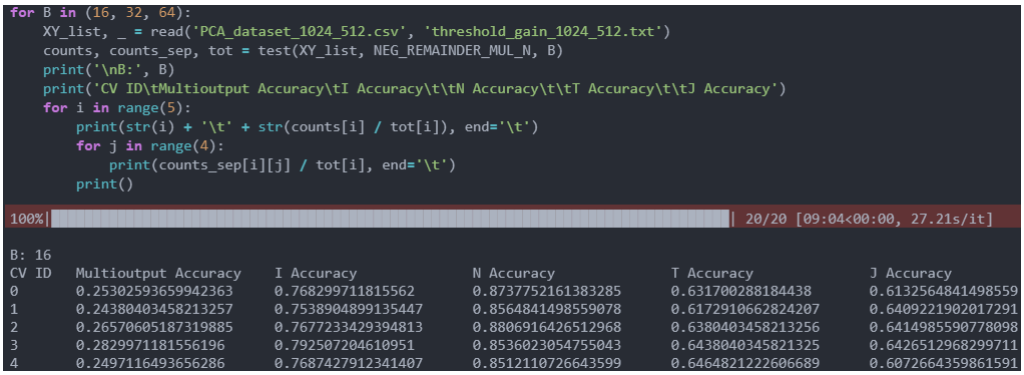



Figure 3: Attributes selection using information gain

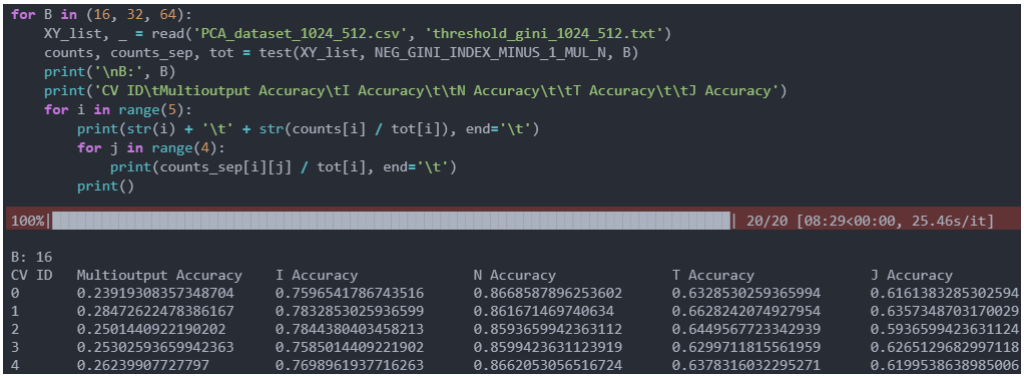



Figure 4: Attributes selection using negative Gini index gain

Their accuracy is almost the same and the time consumption of Gini is a little less than that of gain.

## 3.2 Different types of features

I test several kinds of features as follows.

```
for B in (16, 32, 64):
    XY_list, _ = read('PCA_dataset_512_256.csv', 'threshold_gain_512_256.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

```
100%|| 20/20 [06:53<00:00, 20.70s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.2962536023054755      0.7665706051873199      0.8662824207492795      0.6720461095100865      0.6397694524495677
1       0.2899135446685879      0.7746397694524496      0.8668587896253602      0.7002881844380403      0.631700288184438
2       0.2962536023054755      0.7832853025936599      0.8547550432276657      0.6847262247838617      0.6368876080691642
3       0.26051873198847264     0.7654178674351585      0.8530259365994236      0.6547550432276658      0.6351585014409222
4       0.2773933102652826      0.7675893886966552      0.8696655132641292      0.6660899653979239      0.6280276816608996
```

```
for B in (16, 32, 64):
    XY_list, _ = read('PCA_dataset_1024_512.csv', 'threshold_gain_1024_512.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

```
100%|| 20/20 [09:04<00:00, 27.21s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.25302593659942363     0.768299711815562       0.8737752161383285      0.631700288184438       0.6132564841498559
1       0.24380403458213257     0.7538904899135447      0.8564841498559078      0.6172910662824207      0.6409221902017291
2       0.26570605187319885     0.7677233429394813      0.8806916426512968      0.6380403458213256      0.6414985590778098
3       0.2829971181556196      0.792507204610951       0.8536023054755043      0.6438040345821325      0.6426512968299711
4       0.2497116493656286      0.7687427912341407      0.8512110726643599      0.6464821222606689      0.6072664359861591
```

Figure 5: Tf-Idf processed by feature selection and PCA

```
for B in (16, 32, 64):
    XY_list, _ = read('feature_dataset_512.csv', 'threshold_gain_512.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

```
100%|| 20/20 [20:28<00:00, 61.41s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.33025936599423633     0.768299711815562       0.8547550432276657      0.7279538904899135      0.6922190201729107
1       0.3337175792507205      0.7844380403458213      0.8639769452449567      0.7077809798270893      0.6835734870317003
2       0.35561959654178676     0.7959654178674351      0.8749279538904899      0.7446685878962536      0.699135446685879
3       0.35389048991354466     0.7780979827089337      0.861671469740634       0.7371757925072047      0.6939481268011527
4       0.3460207612456747      0.7612456747404844      0.8575547866205305      0.7352941176470589      0.6966551326412919
```

Figure 6: Tf-Idf processed by feature selection

```
for B in (16, 32, 64):
    XY_list, _ = read('tf_dataset_512.csv', 'threshold_gain_tf_512.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

```
100%|| 20/20 [08:14<00:00, 24.74s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.2760806916426513      0.7631123919308357      0.854178674351585       0.6518731988472622      0.6426512968299711
1       0.24956772334293947     0.768299711815562       0.8714697406340057      0.6282420749279539      0.6103746397694525
2       0.2420749279538905      0.7631123919308357      0.8610951008645533      0.6282420749279539      0.6057636887608069
3       0.2524495677233429      0.7654178674351585      0.8599423631123919      0.6426512968299711      0.5821325648414986
4       0.2520184544405998      0.7854671280276817      0.8615916955017301      0.6113033448673587      0.6026528258362168
```

Figure 7: Tf processed by feature selection

It turns out that TF-Idf processed by feature selection has the best performance.

### 3.3 Different number of features

It seems like the larger the number of features is, the better the decision tree predicts. However, if the number of features is too large, those trival features may greatly affect the random forest. That is because the random forest needs to randomly select attributes before choosing the best attribute. Too many trival features may cause that the features randomly selected are all trival.

```
B = 16
XY_list, _ = read('feature_dataset_16.csv', 'threshold_gain_16.txt')
counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
print('\nB:', B)
print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
for i in range(5):
    print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
    for j in range(4):
        print(counts_sep[i][j] / tot[i], end='\t')
    print()
```

```
100%|| 20/20 [00:42<00:00,  2.13s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.4806916426512968      0.8293948126801153      0.8968299711815562      0.7510086455331412      0.7487031700288185
1       0.4847262247838617      0.8345821325648415      0.8910662824207493      0.7371757925072047      0.738328530259366
2       0.4806916426512968      0.8259365994236312      0.8985590778097983      0.760806916426513       0.7440922190201729
3       0.4818443804034582 3    0.8340057636887608      0.8864553314121038      0.7585014409221902      0.7417867435158502
4       0.47693194925028837     0.8281430219146482      0.8944636678200693      0.751441753171857       0.7341407151095732
```

Figure 8: 16 features

```
B = 16
XY_list, _ = read('feature_dataset_32.csv', 'threshold_gain_32.txt')
counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
print('\nB:', B)
print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
for i in range(5):
    print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
    for j in range(4):
        print(counts_sep[i][j] / tot[i], end='\t')
    print()
```

```
100%|| 20/20 [02:04<00:00,  6.21s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.4933717579250720 4    0.8265129682997118      0.8974063400576369      0.7648414985590778      0.7636887608069164
1       0.500864553314121       0.8213256484149856      0.9123919308357349      0.7665706051873199      0.770028818443804
2       0.49394812680115274     0.8270893371757925      0.8933717579250721      0.770028818443804       0.770028818443804
3       0.4783861671469741      0.8265129682997118      0.9048991354466859      0.7452449567723343      0.7613832853025937
4       0.5103806228373703      0.8408304498269896      0.8863898500576701      0.7710495963091119      0.7820069204152249
```

Figure 9: 32 features

```
B = 16
XY_list, _ = read('feature_dataset_64.csv', 'threshold_gain_64.txt')
counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
print('\nB:', B)
print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
for i in range(5):
    print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
    for j in range(4):
        print(counts_sep[i][j] / tot[i], end='\t')
    print()
```

```
100%|| 20/20 [05:35<00:00, 16.78s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.4812680115273775      0.822478386167147       0.8812680115273775      0.792507204610951       0.7596541786743516
1       0.45878962536023055     0.8190201729106629      0.885878962536023       0.7757925072046109      0.7487031700288185
2       0.46628242074927956     0.8138328530259366      0.8853025936599423      0.7642651296829971      0.7573487031700288
3       0.45936599423631125     0.8097982708933718      0.8829971181556195      0.7694524495677233      0.7504322766570605
4       0.4538638985005767      0.8143021914648212      0.8904267589388697      0.7681660899653979      0.7479815455594002
```

Figure 10: 64 features

```
B = 16
XY_list, _ = read('feature_dataset_128.csv', 'threshold_gain_128.txt')
counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
print('\nB:', B)
print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
for i in range(5):
    print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
    for j in range(4):
        print(counts_sep[i][j] / tot[i], end='\t')
    print()
```

```
100%|| 20/20 [09:06<00:00, 27.33s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.407492795389049       0.7907780979827089      0.8731988472622478      0.7631123919308357      0.72507204610951
1       0.407492795389049       0.7953890489913544      0.8743515850144092      0.770028818443804       0.7296829971181557
2       0.4138328530259366      0.801729106628242       0.8639769452449567      0.7544668587896254      0.7412103746397695
3       0.4034582132564842      0.7827089337175792      0.8812680115273775      0.7648414985590778      0.7210374639769452
4       0.41753171856978083     0.8044982698961938      0.8696655132641292      0.773356401384083       0.7162629757785467
```

Figure 11: 128 features

```
B = 16
XY_list, _ = read('feature_dataset_256.csv', 'threshold_gain_256.txt')
counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
print('\nB:', B)
print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
for i in range(5):
    print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
    for j in range(4):
        print(counts_sep[i][j] / tot[i], end='\t')
    print()
```

```
100%|| 20/20 [18:24<00:00, 55.22s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.3619596541786744      0.7838616714697406      0.8668587896253602      0.7400576368876081      0.7014409221902017
1       0.3631123919308357      0.7706051873198847      0.854178674351585       0.7406340057636888      0.700864553314121
2       0.38270893371757925     0.7798270893371758      0.8731988472622478      0.760806916426513       0.72507204610951
3       0.3746397694524496      0.7890489913544668      0.8599423631123919      0.7475504322766571      0.7112391930835735
4       0.37139561707035756     0.7877739331026529      0.8725490196078431      0.7364475201845444      0.7001153402537486
```

Figure 12: 256 features

```
for B in (16, 32, 64):
    XY_list, _ = read('feature_dataset_512.csv', 'threshold_gain_512.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

```
100%|| 20/20 [20:28<00:00, 61.41s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.33025936599423633     0.768299711815562       0.8547550432276657      0.7279538904899135      0.6922190201729107
1       0.3337175792507205      0.7844380403458213      0.8639769452449567      0.7077809798270893      0.6835734870317003
2       0.35561959654178676     0.7959654178674351      0.8749279538904899      0.7446685878962536      0.699135446685879
3       0.35389048991354466     0.7780979827089337      0.861671469740634       0.7371757925072047      0.6939481268011527
4       0.3460207612456747      0.7612456747404844      0.8575547866205305      0.7352941176470589      0.6966551326412919
```

Figure 13: 512 features

```
for B in (16, 32, 64):
    XY_list, _ = read('feature_dataset_1024.csv', 'threshold_gain_1024.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

```
100%|| 20/20 [36:08<00:00, 108.41s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.3106628242074928      0.7671469740634006      0.8564841498559078      0.7152737752161383      0.6449567723342939
1       0.32507204610951007     0.7711815561959654      0.8530259365994236      0.7141210374639769      0.6760806916426513
2       0.33602305475504324     0.7723342939481268      0.8639769452449567      0.7198847262247838      0.6737752161383286
3       0.32910662824207493     0.7890489913544668      0.8639769452449567      0.7170028818443804      0.6512968299711815
4       0.31430219146482125     0.7635524798154556      0.8708189158016147      0.7370242214532872      0.654555940023068
```

Figure 14: 1024 features

The random forest with 32 features have the best performance.

### 3.4 Different number of trees in a forest

```
for B in (16, 32, 64):
    XY_list, _ = read('feature_dataset_512.csv', 'threshold_gain_512.txt')
    counts, counts_sep, tot = test(XY_list, NEG_REMAINDER_MUL_N, B)
    print('\nB:', B)
    print('CV ID\tMultioutput Accuracy\tI Accuracy\t\tN Accuracy\t\tT Accuracy\t\tJ Accuracy')
    for i in range(5):
        print(str(i) + '\t' + str(counts[i] / tot[i]), end='\t')
        for j in range(4):
            print(counts_sep[i][j] / tot[i], end='\t')
        print()
```

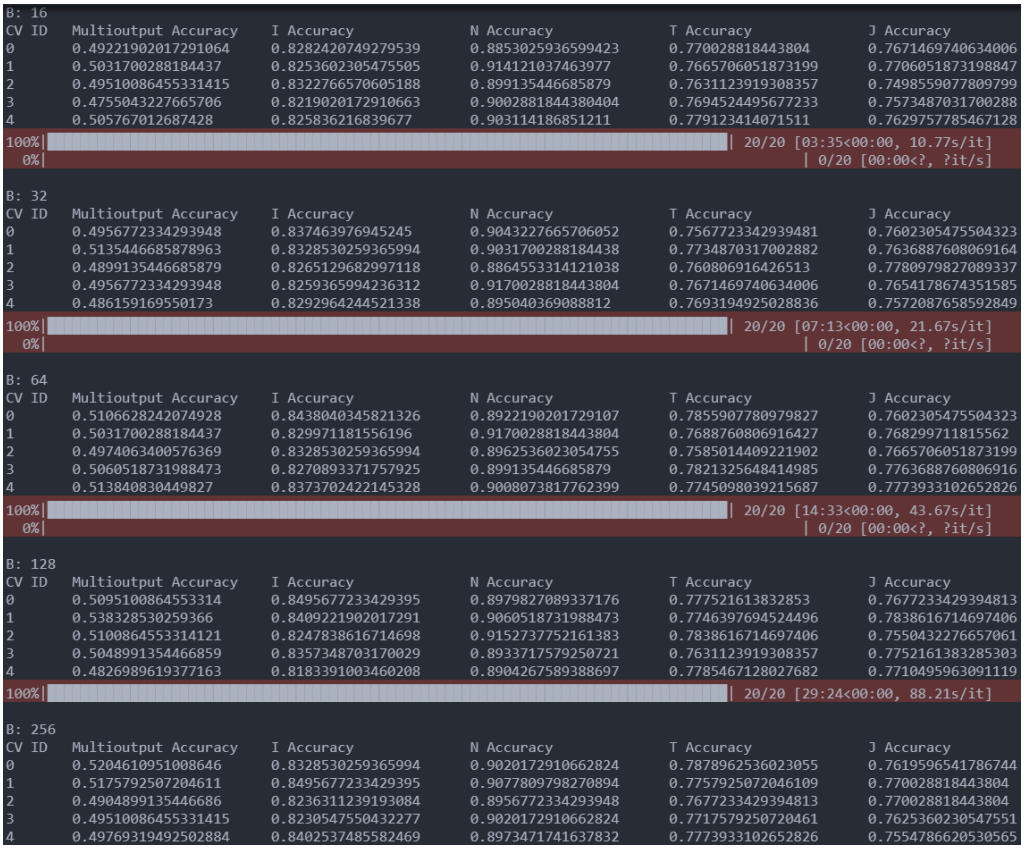
```
100%|| 20/20 [20:28<00:00, 61.41s/it]

B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.33025936599423633     0.768299711815562       0.8547550432276657      0.7279538904899135      0.6922190201729107
1       0.3337175792507205      0.7844380403458213      0.8639769452449567      0.7077809798270893      0.6835734870317003
2       0.35561959654178676     0.7959654178674351      0.8749279538904899      0.7446685878962536      0.699135446685879
3       0.35389048991354466     0.7780979827089337      0.861671469740634       0.7371757925072047      0.6939481268011527
4       0.3460207612456747      0.7612456747404844      0.8575547866205305      0.7352941176470589      0.6966551326412919
100%|| 20/20 [39:35<00:00, 118.79s/it]

B: 32
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.39077809798270896     0.777521613832853       0.8662824207492795      0.7688760806916427      0.7198847262247838
1       0.37521613832853024     0.7671469740634006      0.8680115273775216      0.7544668587896254      0.7354466858789626
2       0.37118155619596543     0.7884726224783861      0.8639769452449567      0.7631123919308357      0.7020172910662824
3       0.3573487031700288      0.7740634005763689      0.8512968299711815      0.7561959654178675      0.7170028818443804
4       0.3731257208765859      0.7687427912341407      0.8621683967704729      0.7525951557093425      0.7145328719723183
100%|| 20/20 [1:18:30<00:00, 235.52s/it]

B: 64
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.3804034582132565      0.7648414985590778      0.8634005763688761      0.7804034582132565      0.723342939481268
1       0.38270893371757925     0.7619596541786744      0.8645533141210374      0.7809798270893372      0.7487031700288185
2       0.4034582132564842      0.7752161383285303      0.8610951008645533      0.8023054755043227      0.7463976945244957
3       0.3982708933717579      0.7890489913544668      0.8587896253602305      0.8051873198847262      0.7187319884726224
4       0.3875432525951557      0.7704728950403691      0.8615916955017301      0.7745098039215687      0.7249134948096886
```

Figure 15: 512 features



```
B: 16
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.49221902017291064     0.8282420749279539      0.8853025936599423      0.770028818443804       0.7671469740634006
1       0.5031700288184437      0.8253602305475505      0.914121037463977       0.7665706051873199      0.7706051873198847
2       0.49510086455331415     0.8322766570605188      0.899135446685879       0.7631123919308357      0.7498559077809799
3       0.4755043227665706      0.8219020172910663      0.9002881844380404      0.7694524495677233      0.7573487031700288
4       0.505767012687428       0.825836216839677       0.903114186851211       0.779123414071511       0.7629757785467128
100%|| 20/20 [03:35<00:00, 10.77s/it]
  0%|| 0/20 [00:00<?, ?it/s]

B: 32
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.4956772334293948      0.837463976945245       0.9043227665706052      0.7567723342939481      0.7602305475504323
1       0.5135446685878963      0.8328530259365994      0.9031700288184438      0.7734870317002882      0.7636887608069164
2       0.4899135446685879      0.8265129682997118      0.8864553314121038      0.760806916426513       0.7780979827089337
3       0.4956772334293948      0.8259365994236312      0.9170028818443804      0.7671469740634006      0.7654178674351585
4       0.486159169550173       0.8292964244521338      0.895040369088812       0.7693194925028836      0.7572087658592849
100%|| 20/20 [07:13<00:00, 21.67s/it]
  0%|| 0/20 [00:00<?, ?it/s]

B: 64
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.5106628242074928      0.8438040345821326      0.8922190201729107      0.7855907780979827      0.7602305475504323
1       0.5031700288184437      0.829971181556196       0.9170028818443804      0.7688760806916427      0.768299711815562
2       0.4974063400576369      0.8328530259365994      0.8962536023054755      0.7585014409221902      0.7665706051873199
3       0.5060518731988473      0.8270893371757925      0.899135446685879       0.7821325648414985      0.7763688760806916
4       0.513840830449827       0.8373702422145328      0.9008073817762399      0.7745098039215687      0.7773933102652826
100%|| 20/20 [14:33<00:00, 43.67s/it]
  0%|| 0/20 [00:00<?, ?it/s]

B: 128
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.5095100864553314      0.8495677233429395      0.8979827089337176      0.777521613832853       0.7677233429394813
1       0.538328530259366       0.8409221902017291      0.9060518731988473      0.7746397694524496      0.7838616714697406
2       0.5100864553314121      0.8247838616714698      0.9152737752161383      0.7838616714697406      0.7550432276657061
3       0.5048991354466859      0.8357348703170029      0.8933717579250721      0.7631123919308357      0.7752161383285303
4       0.4826989619377163      0.8183391003460208      0.8904267589388697      0.7785467128027682      0.7710495963091119
100%|| 20/20 [29:24<00:00, 88.21s/it]

B: 256
CV ID   Multioutput Accuracy    I Accuracy              N Accuracy              T Accuracy              J Accuracy
0       0.5204610951008646      0.8328530259365994      0.9020172910662824      0.7878962536023055      0.7619596541786744
1       0.5175792507204611      0.8495677233429395      0.9077809798270894      0.7757925072046109      0.770028818443804
2       0.4904899135446686      0.8236311239193084      0.8956772334293948      0.7677233429394813      0.770028818443804
3       0.49510086455331415     0.8230547550432277      0.9020172910662824      0.7717579250720461      0.7625360230547551
4       0.49769319492502884     0.8402537485582469      0.8973471741637832      0.7773933102652826      0.7554786620530565
```

Figure 16: 32 features

When the number of features is large, the influence of the number of trees is big. When the number of features is little, the influence of the number of trees is small. It seems like the model falls into a locally optimal solution. And this also support for the explanation of why large number of features can have bad result. Large number of trees can cover the influence caused by frequently selection of trival attributes.

# 4 Conclusions

The result is not good enough compared with SVM. Reasons may be that the model falls into a locally optimal solution, or these types of features may be bad for random forest, or the binary discretization is not good enough.