rf_trials

May 26, 2020

[4]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
```

1 Transforming and Splitting Data

[5]:
```
df = pd.read_csv("data/combined_expression.csv")
```

[6]:
```
df.head()
```

[6]:
```
   CELL_LINE_NAME  classification    TSPAN6      TNMD       DPM1     SCYL3  \
0         1240121               5  6.419526  3.182094   9.320548  3.759654
1         1240122               6  7.646494  2.626819  10.153853  3.564755
2         1240123               5  8.319417  3.111183   9.643558  4.757258
3         1240124               1  9.006994  3.028173   9.686700  4.280504
4         1240127               6  7.985676  2.694729  10.676134  4.159685

   C1orf112       FGR       CFH     FUCA2  …   COL15A1   C6orf10   TMEM225  \
0  3.802619  3.215753  4.698729  7.873672  …  3.245454  2.953508  3.543429
1  3.942749  3.290760  3.551675  8.252413  …  2.786709  3.077382  3.728232
2  3.919757  3.602185  3.329644  9.076950  …  3.459089  3.085394  3.462811
3  3.147646  3.188881  3.293807  8.678790  …  2.835403  2.960303  3.415083
4  3.804637  3.481942  3.111261  7.555407  …  2.896523  2.849899  3.480114

     NOTCH4      PBX2      AGER      RNF5    AGPAT1    DFNB59     PRRT1
0  3.352022  4.672310  3.641128  3.135310  3.737072  3.450927  3.168800
1  3.208882  4.586840  3.395654  3.586800  3.519128  3.115323  3.051645
2  3.339030  4.614897  3.395845  3.419193  3.971646  3.729310  3.320022
3  3.290171  4.770123  3.400821  3.383734  3.798107  2.822404  3.297547
4  3.226128  5.832710  3.612179  3.347095  4.457963  5.198524  4.553586

[5 rows x 16383 columns]
```

[7]:
```
X = df.drop(columns=['CELL_LINE_NAME', 'classification'])
y = df['classification']
feat_labels = list(X.columns)
```

[8]:
```
# 20% test, 80% train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

2 Creating the Classifier With All Features

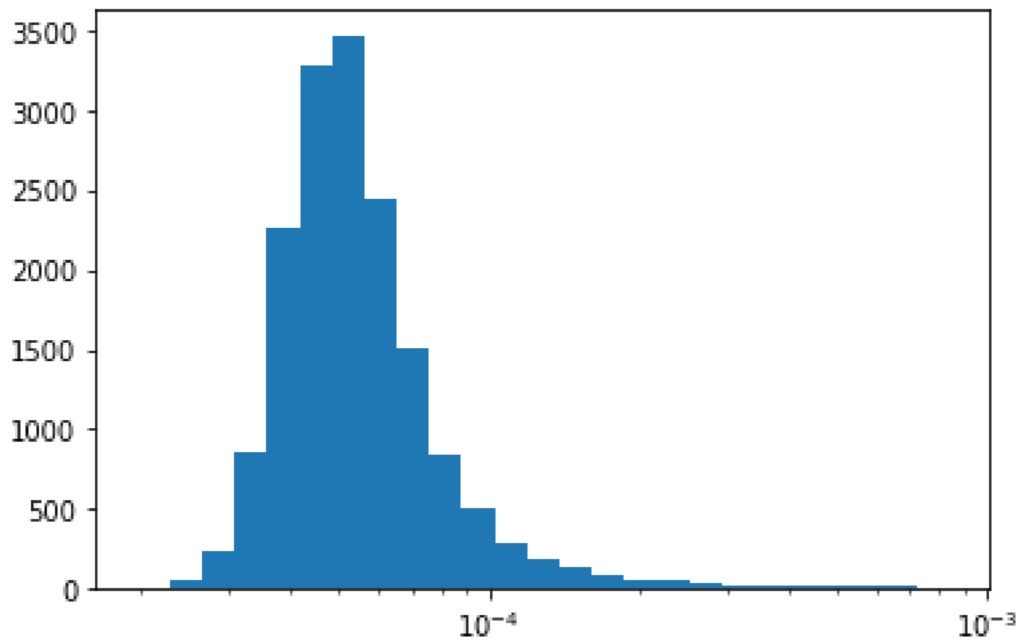
[9]:
```
# create and train the classifier
clf = RandomForestClassifier(n_estimators=X.shape[1], random_state=0, n_jobs=-1)
clf.fit(X_train, y_train)
```

[9]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=16381,
                       n_jobs=-1, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[10]:
```
# associating each feature with its relative importance
feat_importances = list(zip(feat_labels, clf.feature_importances_))
feat_importances.sort(key = lambda x: x[1], reverse=True)
```

[11]:
```
# plotting the feature importances
import matplotlib.pyplot as plt
indices = np.arange(len(feat_importances))
x, y = zip(*feat_importances)

def plot_loghist(x, bins):
    hist, bins = np.histogram(x, bins=bins)
    logbins = np.logspace(np.log10(bins[0]), np.log10(bins[-1]), len(bins))
    plt.hist(x, bins=logbins)
    plt.xscale('log')
plot_loghist(y, 25)
```



3 Testing Various Thresholds

3.1 Defining Functions for Multiple Iterations

[13]:
```
def sfm_transforms(sfm, X_train, X_test):
    X_important_train = sfm.transform(X_train)
    X_important_test = sfm.transform(X_test)
    return X_important_train, X_important_test
```

[15]:
```
def det_accuracy(clf, X_test, y_test):
    y_pred = clf.predict(X_test)
    return accuracy_score(y_test, y_pred)
```

3.2 Testing Multiple Iterations

[72]:
```
hist, bins = np.histogram(y, bins=25)
thresholds = np.logspace(np.log10(bins[0]), np.log10(bins[-1]), len(y))
thresholds = sorted(thresholds, reverse=True)
thresholds = list(filter(lambda x: x >= 8.25e-4, thresholds))
len(thresholds)
```

[72]: 80

```
[73]: thresh_accuracy = list()

j=1
for i in thresholds:
    sfm = SelectFromModel(clf, i)
    sfm.fit(X_train, y_train)
    X_important_train, X_important_test = sfm_transforms(sfm, X_train, X_test)

    clf_important = RandomForestClassifier(n_estimators=X.shape[1],␣
 ↪random_state=0, n_jobs=-1)
    clf_important.fit(X_important_train, y_train)

    thresh_accuracy.append((i, det_accuracy(clf, X_test, y_test)))

    print(f'Iteration: {j}; threshold: {i}')
    j += 1
```

```
Iteration: 1; threshold: 0.0008402505637710409
Iteration: 2; threshold: 0.0008400578348545766
Iteration: 3; threshold: 0.0008398651501444916
Iteration: 4; threshold: 0.0008396725096306465
Iteration: 5; threshold: 0.0008394799133029039
Iteration: 6; threshold: 0.0008392873611511286
Iteration: 7; threshold: 0.0008390948531651882
Iteration: 8; threshold: 0.0008389023893349522
Iteration: 9; threshold: 0.0008387099696502934
Iteration: 10; threshold: 0.0008385175941010836
Iteration: 11; threshold: 0.0008383252626772029
Iteration: 12; threshold: 0.0008381329753685267
Iteration: 13; threshold: 0.0008379407321649397
Iteration: 14; threshold: 0.0008377485330563222
Iteration: 15; threshold: 0.0008375563780325633
Iteration: 16; threshold: 0.0008373642670835479
Iteration: 17; threshold: 0.00083717220019917
Iteration: 18; threshold: 0.0008369801773693199
Iteration: 19; threshold: 0.0008367881985838937
Iteration: 20; threshold: 0.0008365962638327886
Iteration: 21; threshold: 0.0008364043731059048
Iteration: 22; threshold: 0.0008362125263931442
Iteration: 23; threshold: 0.0008360207236844113
Iteration: 24; threshold: 0.0008358289649696127
Iteration: 25; threshold: 0.0008356372502386578
Iteration: 26; threshold: 0.0008354455794814586
Iteration: 27; threshold: 0.0008352539526879264
Iteration: 28; threshold: 0.0008350623698479805
Iteration: 29; threshold: 0.0008348708309515358
Iteration: 30; threshold: 0.0008346793359885166
```

```
Iteration: 31; threshold: 0.0008344878849488421
Iteration: 32; threshold: 0.0008342964778224413
Iteration: 33; threshold: 0.0008341051145992391
Iteration: 34; threshold: 0.0008339137952691661
Iteration: 35; threshold: 0.0008337225198221547
Iteration: 36; threshold: 0.0008335312882481393
Iteration: 37; threshold: 0.0008333401005370568
Iteration: 38; threshold: 0.0008331489566788462
Iteration: 39; threshold: 0.0008329578566634491
Iteration: 40; threshold: 0.000832766800480809
Iteration: 41; threshold: 0.000832575788120873
Iteration: 42; threshold: 0.000832384819573587
Iteration: 43; threshold: 0.0008321938948289047
Iteration: 44; threshold: 0.000832003013876776
Iteration: 45; threshold: 0.0008318121767071593
Iteration: 46; threshold: 0.0008316213833100091
Iteration: 47; threshold: 0.0008314306336752883
Iteration: 48; threshold: 0.000831239927792956
Iteration: 49; threshold: 0.0008310492656529799
Iteration: 50; threshold: 0.0008308586472453243
Iteration: 51; threshold: 0.000830668072559959
Iteration: 52; threshold: 0.0008304775415868556
Iteration: 53; threshold: 0.0008302870543159876
Iteration: 54; threshold: 0.000830096610737331
Iteration: 55; threshold: 0.0008299062108408643
Iteration: 56; threshold: 0.0008297158546165679
Iteration: 57; threshold: 0.0008295255420544247
Iteration: 58; threshold: 0.0008293352731444208
Iteration: 59; threshold: 0.0008291450478765409
Iteration: 60; threshold: 0.0008289548662407785
Iteration: 61; threshold: 0.000828764728227122
Iteration: 62; threshold: 0.0008285746338255693
Iteration: 63; threshold: 0.0008283845830261137
Iteration: 64; threshold: 0.0008281945758187573
Iteration: 65; threshold: 0.0008280046121934992
Iteration: 66; threshold: 0.0008278146921403434
Iteration: 67; threshold: 0.0008276248156492959
Iteration: 68; threshold: 0.000827434982710365
Iteration: 69; threshold: 0.0008272451933135608
Iteration: 70; threshold: 0.0008270554474488961
Iteration: 71; threshold: 0.0008268657451063861
Iteration: 72; threshold: 0.0008266760862760479
Iteration: 73; threshold: 0.000826486470947902
Iteration: 74; threshold: 0.0008262968991119676
Iteration: 75; threshold: 0.0008261073707582724
Iteration: 76; threshold: 0.0008259178858768395
Iteration: 77; threshold: 0.0008257284444577008
Iteration: 78; threshold: 0.000825539046490884
```

```
Iteration: 79; threshold: 0.000825349691966426
Iteration: 80; threshold: 0.0008251603808743588
```

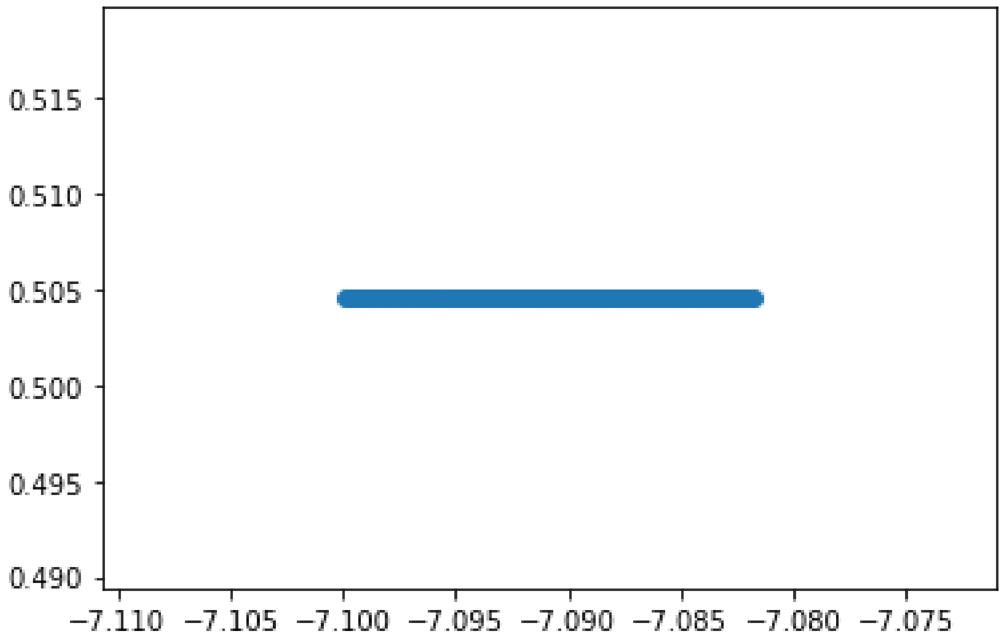
[74]:
```
thresh_accuracy
```

[74]:
```
[(0.0008402505637710409, 0.5045871559633027),
 (0.0008400578348545766, 0.5045871559633027),
 (0.0008398651501444916, 0.5045871559633027),
 (0.0008396725096306465, 0.5045871559633027),
 (0.0008394799133029039, 0.5045871559633027),
 (0.0008392873611511286, 0.5045871559633027),
 (0.0008390948531651882, 0.5045871559633027),
 (0.0008389023893349522, 0.5045871559633027),
 (0.0008387099696502934, 0.5045871559633027),
 (0.0008385175941010836, 0.5045871559633027),
 (0.0008383252626772029, 0.5045871559633027),
 (0.0008381329753685267, 0.5045871559633027),
 (0.0008379407321649397, 0.5045871559633027),
 (0.0008377485330563222, 0.5045871559633027),
 (0.0008375563780325633, 0.5045871559633027),
 (0.0008373642670835479, 0.5045871559633027),
 (0.00083717220019917, 0.5045871559633027),
 (0.0008369801773693199, 0.5045871559633027),
 (0.0008367881985838937, 0.5045871559633027),
 (0.0008365962638327886, 0.5045871559633027),
 (0.0008364043731059048, 0.5045871559633027),
 (0.0008362125263931442, 0.5045871559633027),
 (0.0008360207236844113, 0.5045871559633027),
 (0.0008358289649696127, 0.5045871559633027),
 (0.0008356372502386578, 0.5045871559633027),
 (0.0008354455794814586, 0.5045871559633027),
 (0.0008352539526879264, 0.5045871559633027),
 (0.0008350623698479805, 0.5045871559633027),
 (0.0008348708309515358, 0.5045871559633027),
 (0.0008346793359885166, 0.5045871559633027),
 (0.0008344878849488421, 0.5045871559633027),
 (0.0008342964778224413, 0.5045871559633027),
 (0.0008341051145992391, 0.5045871559633027),
 (0.0008339137952691661, 0.5045871559633027),
 (0.0008337225198221547, 0.5045871559633027),
 (0.0008335312882481393, 0.5045871559633027),
 (0.0008333401005370568, 0.5045871559633027),
 (0.0008331489566788462, 0.5045871559633027),
 (0.0008329578566634491, 0.5045871559633027),
 (0.000832766800480809, 0.5045871559633027),
 (0.000832575788120873, 0.5045871559633027),
 (0.000832384819573587, 0.5045871559633027),
```

```
(0.0008321938948289047, 0.5045871559633027),
(0.000832003013876776, 0.5045871559633027),
(0.0008318121767071593, 0.5045871559633027),
(0.0008316213833100091, 0.5045871559633027),
(0.0008314306336752883, 0.5045871559633027),
(0.000831239927792956, 0.5045871559633027),
(0.0008310492656529799, 0.5045871559633027),
(0.0008308586472453243, 0.5045871559633027),
(0.000830668072559959, 0.5045871559633027),
(0.0008304775415868556, 0.5045871559633027),
(0.0008302870543159876, 0.5045871559633027),
(0.000830096610737331, 0.5045871559633027),
(0.0008299062108408643, 0.5045871559633027),
(0.0008297158546165679, 0.5045871559633027),
(0.0008295255420544247, 0.5045871559633027),
(0.0008293352731444208, 0.5045871559633027),
(0.0008291450478765409, 0.5045871559633027),
(0.0008289548662407785, 0.5045871559633027),
(0.000828764728227122, 0.5045871559633027),
(0.0008285746338255693, 0.5045871559633027),
(0.0008283845830261137, 0.5045871559633027),
(0.0008281945758187573, 0.5045871559633027),
(0.0008280046121934992, 0.5045871559633027),
(0.0008278146921403434, 0.5045871559633027),
(0.0008276248156492959, 0.5045871559633027),
(0.000827434982710365, 0.5045871559633027),
(0.0008272451933135608, 0.5045871559633027),
(0.0008270554474488961, 0.5045871559633027),
(0.0008268657451063861, 0.5045871559633027),
(0.0008266760862760479, 0.5045871559633027),
(0.000826486470947902, 0.5045871559633027),
(0.0008262968991119676, 0.5045871559633027),
(0.0008261073707582724, 0.5045871559633027),
(0.0008259178858768395, 0.5045871559633027),
(0.0008257284444577008, 0.5045871559633027),
(0.000825539046490884, 0.5045871559633027),
(0.000825349691966426, 0.5045871559633027),
(0.0008251603808743588, 0.5045871559633027)]
```

[87]:
```python
testList2 = [(np.log(elem1), elem2) for elem1, elem2 in thresh_accuracy]
plt.scatter(*zip(*testList2))
```

[87]:
```
<matplotlib.collections.PathCollection at 0x1ab24bbe48>
```


0.515
0.510
0.505
0.500
0.495
0.490
−7.110
−7.105
−7.100
−7.095
−7.090
−7.085
−7.080
−7.075

```
[ ]:
```