Evaluation of Hierarchical Classification

Ameet Deshpande

May 2017

1 Abstract

Two approaches for hierarchical classification are evaluated.
First approach(called **Local Classifier Per Parent**) creates one classifier per parent in the hierarchy tree, while the second(called **Global Classifier**) treats all the leaves as separate classes and does a multi-label classification.
The four models will be evaluated on the following metrics.

- Jaccard Similarity Score
- Hamming Loss(Closer to 0 is more optimal)
- Execution Time

It is worth noting that the machine used is a windows, 4 $GB\ RAM$ machine with an intel $i5$ processor. The dataset that is being used is the Slashdot News data in $ARFF$ Format which can be found here. The training set consists of 3402 samples and the test set consists of 379 samples. The code is written using python and sklearn library.
A problem transformation technique called Label PowerSet is used in all the cases, where if there are $|l|$ labels, then $2^{|l|}$ different labels are created and the model is trained on that. This approach works when there are not too many different label combinations possible as the number of labels will shoot up exponentially.

2 Hierarchical Structure

- Root Node
- Entertainment
  - Games
  - Not Games
- Technology
  - Apple

  - Linux
  - IT
  - Hardware
  - Developers
  - Apache
  - Mobile
  - Technology(Generic)
- Science
- Politics

3 Local Classifier Per Parent

3.1 Support Vector Machine

- Jaccard Similarity Score : 0.525
- Hamming Loss(Zero is optimal) : 0.045
- Execution Time : 0.65 *seconds*

3.2 Random Forest Classifier

- Jaccard Similarity Score : 0.5
- Hamming Loss(Zero is optimal) : 0.049
- Execution Time : 0.7 *seconds*

4 Global Classifier

4.1 Support Vector Machine

- Jaccard Similarity Score : 0.575
- Hamming Loss(Zero is optimal) : 0.049
- Execution Time : 2 *seconds*

4.2 Random Forest Classifier

- Jaccard Similarity Score : 0.594
- Hamming Loss(Zero is optimal) : 0.047
- Execution Time : 1 *second*

5 Analysis

For random forests, the larger the value of *n_estimator* the more stable the results are. For global classifier approach, as there is only one model to train, the value can be made as large as possible with time constraints as the only limit. For the local classifier per parent approach, there will be one model to train per non-leaf node. Thus the running time for fitting the model parameters will be higher and the permissible value for the parameter *n_estimator* will be lower.
The Global Classifier seems to be performing better than the Local Classifier per node in this example as expected as the number of leaves is not very large. Execution time wise, the local classifier approach's highly vectorized code does slightly better than the global approach, but this is not expected to follow if the number of parents increases.
The evaluation metrics for the 4 models show that they perform very similarly and that it is important to fine tune the parameters like the regularization coefficient and number of trees. *GridSearchCV can be used for this*

6 Code

6.1 Global Classifier

```
from functions import load_from_arff
from sklearn.ensemble import RandomForestClassifier
from skmultilearn.problem_transform import LabelPowerset
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import hamming_loss
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
import time

X_train = y_train = X_test = y_test = None

train_filename = "SLASHDOT-F-train.arff"
test_filename = "SLASHDOT-F-test.arff"

[X_train, y_train] = load_from_arff(filename=
    ↪ train_filename, load_sparse=True, labelcount=22)
[X_test, y_test] = load_from_arff(filename=test_filename,
    ↪  load_sparse=True, labelcount=22)

# The matrices are initially in lil_matrix format
# Converting them to compressed row matrix format

X_train = X_train.tocsr()
```

```
y_train = y_train.todense()
X_test = X_test.tocsr()
y_test = y_test.todense()

label_set = set([12,15,0,19,9,17,3,4,7,21,8,20])
label_list = [12,15,0,19,9,17,3,4,7,21,8,20]

y_train = y_train[:,label_list]
y_test = y_test[:,label_list]

start_time = time.process_time()
# classifier = LabelPowerset(RandomForestClassifier(
    ↪ random_state=0, n_estimators=10, n_jobs=-1))

# classifier = RandomForestClassifier(random_state=0,
    ↪ n_estimators=10)
# classifier = BinaryRelevance(classifier = LinearSVC(),
    ↪ require_dense = [False, True])
classifier = LabelPowerset(SGDClassifier(penalty='l2',
    ↪ alpha=0.01))
classifier.fit(X_train, y_train)
y_predicted = classifier.predict(X_test)
total_time = time.process_time() - start_time

print("Total time taken is : "+str(total_time))

print("Jaccard Similarity Score is : "+str(
    ↪ jaccard_similarity_score(y_test, y_predicted)))
print("Hamming Loss is : "+str(hamming_loss(y_test,
    ↪ y_predicted)))
# print("F1 Similarity score is : "+str(f1_score(y_test,
    ↪ y_predicted, average='macro')))
```

6.2 Local Classifier

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import numpy as np
from functions import load_from_arff, fill_bool_array,
    ↪ fit_classifiers
from sklearn.ensemble import RandomForestClassifier
from skmultilearn.problem_transform import LabelPowerset
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import hamming_loss
from sklearn.linear_model import SGDClassifier
```

```
import time
import copy

X_train = y_train = X_test = y_test = None

train_filename = 'SLASHDOT-F-train.arff'
test_filename = 'SLASHDOT-F-test.arff'

[X_train, y_train] = load_from_arff(filename=
    ↪ train_filename,
                                        load_sparse=True,
                                          ↪ labelcount=22)
[X_test, y_test] = load_from_arff(filename=test_filename,
                                      load_sparse=True,
                                        ↪ labelcount=22)

# The matrices are initially in lil_matrix format
# Converting them to compressed row matrix format

X_train = X_train.tocsr()
y_train = y_train.todense()
X_test = X_test.tocsr()
y_test = y_test.todense()

# label_set = set([0, 3, 4, 7, 8, 9, 12, 15, 17, 19, 20,
    ↪ 21])
label_list = [[0, 3, 4, 7, 8, 9, 12, 15, 17, 19, 20, 21],
    ↪  [0, 12], [3, 4, 7, 9, 15, 17, 19, 21]]
inverse_label_list = [None, [3, 4, 7, 8, 9, 15, 17, 19,
    ↪ 20, 21], [0, 8, 12, 20]]

start_time = time.time()

# Initialize the classifiers, one per parent

# classifier = LabelPowerset(RandomForestClassifier(
    ↪ random_state=0, n_estimators=10, n_jobs=-1))
classifier = LabelPowerset(SGDClassifier(penalty='l2',
    ↪ alpha=0.01))
classifiers = []
for i in range(3):
    classifiers.append(copy.deepcopy(classifier))

# Create boolean array for choosing relevant documents
    ↪ for each parent classifer
```

```
bool_shape = y_train.shape[0]
bool_array = [np.zeros(shape=bool_shape, dtype=bool) for
    ↪ i in range(3)]

# Populate the boolean arrays

bool_array = fill_bool_array(bool_array, label_list,
    ↪ y_train)

# Fit the classifiers

classifiers = fit_classifiers(classifiers, bool_array,
    ↪ label_list,
                                X_train, y_train)

# Choose only those test documents that belong to one of
    ↪ the chosen categories

test_bool_array = np.zeros(shape=y_test.shape[0], dtype=
    ↪ bool)
for i in label_list[0]:
    test_bool_array = np.logical_or(test_bool_array, np.
        ↪ array(y_test[:,
                                        i] == 1).flatten())

# Subset the test documents accordingly

X_test = X_test[test_bool_array, :]
y_test = y_test[test_bool_array, :]

# y_predicted will store the predicted results

y_predicted = np.zeros(y_test.shape)

# Perform the first level of classification

y_predicted[:, [0, 1, 8, 20]] = classifiers[0].predict(
    ↪ X_test).todense()

# Based on the first level of classification, subset the
    ↪ documents which are
# predicted to be in any other parent category

test_bool_array = [None, None]
test_bool_array[0] = y_predicted[:, 0] == 1
test_bool_array[1] = y_predicted[:, 1] == 1
```

```
# Further predict categories for the other parent
    ↪ classifiers

for i in range(2):
    temp_y_predicted = np.zeros(shape=y_predicted[
        ↪ test_bool_array[i], :
                                    ].shape)
    temp_y_predicted[:, inverse_label_list[i + 1]] = \
        y_predicted[test_bool_array[i], :][:,
            ↪ inverse_label_list[i + 1]]
    temp_y_predicted[:, label_list[i + 1]] = classifiers[
        ↪ i
            + 1].predict(X_test[test_bool_array[i], :]).
                ↪ todense()
    y_predicted[test_bool_array[i], :] = temp_y_predicted

# Print the total time taken

total_time = time.time() - start_time
print('Total time taken is : ' + str(total_time))

# Print the scores(evaluation metris)

print('Jaccard Similarity Score is : ' \
    + str(jaccard_similarity_score(y_test, y_predicted)))
print('Hamming Loss is : ' + str(hamming_loss(y_test,
    ↪ y_predicted)))
```