hw5-25912237

November 9, 2015

All code can be very easily ran with python, you just have to import the classes from the inner `tree` folder.

```
In [112]: import scipy as sp
          import scipy.io
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt

          from numpy.random import choice
          from tree.tree import DecisionTree, RandomForest
          from sklearn.preprocessing import Imputer
          from sklearn.preprocessing import LabelEncoder
          from sklearn.cross_validation import train_test_split
```

```
In [42]: feature_list = ['pain', 'private', 'bank', 'money',
                         'drug', 'spam', 'prescription',
                         'creative', 'height', 'featured',
                         'differ', 'width', 'other', 'energy',
                         'business', 'message', 'volumes', 'revision',
                         'path', 'meter', 'memo', 'planning', 'pleased',
                         'record', 'out' ,';','$','#','!','(','[','&']
         feature_dict = dict(zip(range(len(feature_list)), feature_list))
```

1 See Below

2 See Below

3 Spam

```
In [4]: spam = scipy.io.loadmat("spam-dataset/spam_data.mat")
```

```
In [5]: spam_raw_train_X = spam['training_data']
        spam_raw_train_y = spam['training_labels'].reshape((5172,))
        spam_raw_test_X = spam['test_data']
        shuff = choice(len(spam_raw_train_X), len(spam_raw_train_X), False)
        spam_raw_train_X = spam['training_data'][shuff]
        spam_raw_train_y = spam['training_labels'].reshape((5172,))[shuff]
        spam_raw_test_X = spam['test_data']
```

```
In [8]: spamTX, spamTestX, spamTy, spamTesty = train_test_split(
            spam_raw_train_X, spam_raw_train_y, test_size=0.2)
```

```
In [9]: print(len(spamTX), len(spamTestX))
```

```
4137 1035
```

3.1 a.

No extra features were used for either decision trees or random forest with spam.

3.2 Spam Decision Tree

```
In [ ]: %autoreload
        dt = DecisionTree({
                "max_depth":100, # this is > max depth possible
                "min_points": 2
            })
        dt.train(spamTX, spamTy)
        scored = dt.score(spamTestX, spamTesty)
```

3.3 b.

```
In [44]: print("Decision Tree Validation Error Rate:", sum(scored)/len(scored))
```

```
Decision Tree Validation Error Rate: 0.811594202899
```

```
In [45]: spam_out = pd.DataFrame(dt.predict(spam_raw_test_X)).reset_index()
         spam_out.columns = ['Id', 'Category']
         spam_out.Id = spam_out.Id + 1
         spam_out.to_csv("dt_spam_prediction.csv", index=False)
```

3.4 c.

```
In [48]: dt.get_splits(spamTX[0].reshape(1, len(spamTX[0])), feature_list)
         print("Actual:", spamTy[0])
```

```
!  <=  0.0
meter  <=  0.0
(  <=  0.0
volumes  <=  0.0
&  <=  0.0
pain  <=  0.0
;  <=  0.0
[  <=  0.0
prescription  <=  0.0
energy  <=  1.0
path  <=  0.0
memo  <=  0.0
bank  <=  1.0
spam  <=  0.0
differ  <=  0.0
drug  <=  0.0
planning  <=  0.0
#  <=  1.0
revision  <=  0.0
pleased  <=  0.0
$  <=  6.0
business  <=  0.0
message  <=  0.0
featured  <=  0.0
width  <=  0.0
other  <=  2.0
```

```
out  <=  2.0
other  <=  1.0
bank  <=  0.0
out  <=  0.0
money  <=  1.0
$  <=  2.0
$  <=  1.0
money  <=  0.0
$  <=  0.0
private  <=  0.0
other  <=  0.0
#  >  0.0
End of tree, outputting label 0
Actual: 0
```

3.5 Spam Random Forest

```
In [ ]: %autoreload
        rf = RandomForest({
                'ntrees': 50,
                },{
                "max_depth": 100,
                "min_points": 2,
                'random_subset': True,
                'subset_size': 8
                })
        print(rf)
        rf.train(spamTX, spamTy)
        scored = rf.score(spamTestX, spamTesty)
```

3.6 b. (part 2)

```
In [79]: print("Random Forest Validation Error Rate:", sum(scored)/len(scored))
```

```
Random Forest. 50 Trees
Training tree number 0
Training tree number 20
Training tree number 40
trained all trees
Random Forest Validation Error Rate: 0.833816425121
```

```
In [80]: spam_out = pd.DataFrame(rf.predict(spam_raw_test_X)).reset_index()
         spam_out.columns = ['Id', 'Category']
         spam_out.Id = spam_out.Id + 1
         spam_out.to_csv("rf_spam_prediction.csv", index=False)
```

3.7 d.

```
In [86]: from collections import Counter
         for (feat, val), count in Counter(rf.get_splits()).most_common(25):
             print(feature_dict[feat], "split on", val, "with ", count, "trees")
```

```
money split on 0.0 with  50 trees
( split on 0.0 with  50 trees
[ split on 0.0 with  50 trees
```

```
spam split on 0.0 with  50 trees
& split on 0.0 with  50 trees
prescription split on 0.0 with  50 trees
volumes split on 0.0 with  50 trees
$ split on 1.0 with  50 trees
other split on 0.0 with  50 trees
revision split on 0.0 with  50 trees
energy split on 0.0 with  50 trees
pain split on 0.0 with  50 trees
business split on 0.0 with  50 trees
meter split on 0.0 with  50 trees
out split on 0.0 with  50 trees
message split on 0.0 with  50 trees
; split on 0.0 with  50 trees
$ split on 0.0 with  50 trees
# split on 0.0 with  50 trees
! split on 0.0 with  50 trees
drug split on 0.0 with  49 trees
differ split on 0.0 with  49 trees
( split on 1.0 with  49 trees
path split on 0.0 with  49 trees
memo split on 0.0 with  49 trees
```

Best Random Forest Spam Submission on Kaggle: 0.78108

4 Census

```
In [87]: census = pd.read_csv("census-dataset/data.csv", na_values=["?"])
         census_test = pd.read_csv("census-dataset/test_data.csv", na_values=["?"])
         categoricals = ["workclass", "education", "marital-status",
                         "occupation", "relationship", "race", "sex",
                         "native-country"]
         non_categoricals = list(
             set(census.columns
                ).difference(set(categoricals)).difference(set(["label"])))
         # all we're doing is getting all non-categorical variables into a list

In [88]: print(set(census[non_categoricals].columns
                  ).difference(
                 set(census_test[non_categoricals].columns)))
         print(
             set(pd.get_dummies(census[categoricals]).columns).difference(
             set(pd.get_dummies(census_test[categoricals]).columns))
         )# we can see that we get an extra variable in the census,
         # that's obviously problematic and will throw off our computations
         # I'm going to manually add that column to our test set

set()
{'native-country_Holand-Netherlands'}

In [113]: census_X = pd.concat(
                [census[non_categoricals],
                 pd.get_dummies(census[categoricals])],
                axis=1)
```

```
        census_y = census.label.values
        census_X = census_X.reindex_axis(sorted(census_X.columns), axis=1)
```

```
In [114]: temp = pd.get_dummies(census_test[categoricals])
          temp['native-country_Holand-Netherlands'] = 0
          census_test_X = pd.concat([census_test[non_categoricals], temp], axis=1)
          census_test_X = census_test_X.reindex_axis(sorted(census_test_X.columns), axis=1)
```

```
In [91]: censusTX, censusTestX, censusTy, censusTesty = train_test_split(
             census_X.values, census_y, test_size=.35)
```

```
In [92]: print(set(census_X.columns).difference(set(census_test_X.columns)))
```

```
set()
```

```
In [93]: print(censusTX.shape, censusTy.shape)
         print(censusTestX.shape)
```

```
(21270, 105) (21270,)
(11454, 105)
```

5 4.

5.1 a.

The pre-processing I performed was just 'dummifying' the variables. I just convert them into columns for the respective categorical feature. So that every possible value in relationship becomes its own [sparse] feature column. I only handled categorical variables by creating a new column (essentially treating them as a different category).

5.2 Census Decision Tree

```
In [ ]: %autoreload
        dt2 = DecisionTree({
                "max_depth":1000,
                "min_points": 15
            })
        dt2.train(censusTX, censusTy)
        scored = dt2.score(censusTestX, censusTesty)
```

5.3 b.

```
In [103]: print("Best Decision Tree Validation Error Rate:", sum(scored)/len(scored))
```

```
Best Decision Tree Validation Error Rate: 0.845468831849
```

```
In [104]: census_out = pd.DataFrame(dt2.predict(census_test_X.values)).reset_index()
          census_out.columns = ['Id', 'Category']
          census_out.Id = census_out.Id + 1
          census_out.to_csv("dt_census_prediction.csv", index=False)
```

```
In [105]: feature_dict = dict(zip(range(len(census_X.columns)), census_X.columns))
```

5.4 c.

```
In [106]: dt.get_splits(censusTX[0].reshape(1, len(censusTX[0])), feature_dict)
          print("Actual:", spamTy[0])
```

```
marital-status_Widowed  <=  0.0
education_Some-college  <=  0.0
native-country_Cambodia  <=  0.0
education_Masters  <=  0.0
native-country_China  <=  0.0
age  >  0.0
End of tree, outputting label 1
Actual: 0
```

5.5 Census Random Forest

```
In [107]: %autoreload
          rf2 = RandomForest({
                  'ntrees': 50,
                  },{
                  "max_depth": 100,
                  "min_points": 15,
                  'random_subset': True,
                  'subset_size': 15
                  })
          print(rf2)
```

```
Random Forest. 50 Trees
```

```
In [108]: rf2.train(censusTX, censusTy)
```

```
Training tree number 0
Training tree number 20
Training tree number 40
trained all trees
```

5.6 b. (part 2)

```
In [109]: scored = rf2.score(censusTestX, censusTesty)
          print("Best Random Forest Validation Error Rate:", sum(scored)/len(scored))
```

```
Best Random Forest Validation Error Rate: 0.858302776323
```

Best Random Forest Kaggle: 0.85395

5.7 d.

```
In [110]: from collections import Counter
          for (feat, val), count in Counter(rf2.get_splits()).most_common(30):
              print(feature_dict[feat], "split on", val, "with count", count)
```

```
age split on 28.0 with count 50
occupation_Sales split on 0.0 with count 50
education_Doctorate split on 0.0 with count 50
education-num split on 11.0 with count 50
age split on 24.0 with count 50
```

```
native-country_Cuba split on 0.0 with count 50
education_12th split on 0.0 with count 50
education-num split on 10.0 with count 50
relationship_Unmarried split on 0.0 with count 50
education_Some-college split on 0.0 with count 50
age split on 29.0 with count 50
workclass_Self-emp-not-inc split on 0.0 with count 50
education_Prof-school split on 0.0 with count 50
relationship_Other-relative split on 0.0 with count 50
hours-per-week split on 48.0 with count 50
occupation_Machine-op-inspct split on 0.0 with count 50
capital-loss split on 0.0 with count 50
age split on 36.0 with count 50
relationship_Husband split on 0.0 with count 50
occupation_Farming-fishing split on 0.0 with count 50
workclass_State-gov split on 0.0 with count 50
occupation_Craft-repair split on 0.0 with count 50
workclass_Self-emp-inc split on 0.0 with count 50
marital-status_Separated split on 0.0 with count 50
race_Black split on 0.0 with count 50
education_Assoc-voc split on 0.0 with count 50
age split on 46.0 with count 50
education-num split on 12.0 with count 50
race_Amer-Indian-Eskimo split on 0.0 with count 50
native-country_Mexico split on 0.0 with count 50
```

```
In [111]: census_out = pd.DataFrame(rf2.predict(census_test_X.values)).reset_index()
          census_out.columns = ['Id', 'Category']
          census_out.Id = census_out.Id + 1
          census_out.to_csv("rf_census_prediction.csv", index=False)
```

6 5.

6.1 a.

The decision tree techniques I used were pretty straightforward. I implemented stopping criteria which you can see as the parameters passed into the creation criteria for the decision tree (and as the second parameter dictionary in the Random Forest). The two stopping criteria were: 1. Total depth 2. Total number of values required for a split

I experimented with feature selection in the decision trees and got approximately the same kind of results. I didn't implement heuristics for faster training or complex decision boundaries, pruning, or adaboost. I didn't use cross validation.

6.2 b.

My random forest implementation is a straightforward extension of the decision tree classifier mentioned above except that bagging is automatic as was feature selection as a random subset of features of a parameterized length (that you can see as passed into the random forest class creation).