SciComp with Py

From Random Trees To Random Forests

Vladimir Kulyukin
Department of Computer Science
Utah State University

Basic Idea

- In many situations, it is hard to rely on a single classifier
- Even when we have several classifiers of the same type (e.g., decision trees), there are cases when it makes sense to use all of them rather than just one to get more reliable results
- Why? Because classifiers may be trained on different data or may be trained with different parameters
- Even if the classifiers are of the same type, they may be different, e.g., trained on different attributes

Ensemble Learning

Ensemble learning refers to using multiple models of the same or different type to solve the same problem and choosing the solution by voting.

Ensemble Learning Techniques

- ***Bagging*** (aka bootstrap aggregating): models of the same type built on random samples drawn from the same data
- ***Boosting***: every subsequent model in an ensemble focuses (boosts) on the attributes misclassified by the previous model
- ***Bucket of models***: train several models and pick the one that works best on the test data
- ***Stacking***: train models on some train data, run them simultaneously on the test data, combine the results

Random Forest

Random forest is a collection of decision trees trained on different, randomly selected subsets of a given data set. Random forests are a type of ensemble learning. Decision trees in a forest may be trained on different subsets of a given set of attributes.

Background on Random Forest

- Random forests are a sensible choice for any linear (and even non-linear) prediction problems
- It is a relatively new ML method (started in the 1990's)
- A rule of thumb in data science: when you can't think of an algorithm for a data science problem, try random forests

How Does Random Forests Work?

- We create a bunch of decision trees (hence a forest): the trees are created by randomly choosing training data subsets and/or randomly choosing subsets of attributes (hence the term *random forest*)
- To classify a new object, we take each tree from the forest and classify the new object with it (each tree votes on the class of the object)
- The forest chooses the class of the object on the basis of the majority of votes

Advantages of Random Forests

- Random forests can handle large data sets
- Random forests effectively estimate missing data (i.e., large data sets with many data items missing)
- Random forests can be and have been applied to unlabeled data for clustering

Disadvantages of Random Forests

- Random forests are much better at classification than they are at regression (curve fitting)
- To many data scientists, random forests feel like black boxes (they are black boxes for all practical purposes), because they have very little control over what the learned model does

Implementing Random Forests from Scratch

Review: Digits Dataset

- Digits is one of the sklearn image datasets
- This dataset consists of 1797 images of handwritten characters
- Each image is 8 x 8, i.e., has 64 pixels.
- In numpy terms, each image is a 64-element array of floats
- The target vector for this dataset is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Review: Digits Dataset

```
from sklearn.datasets import load_digits


digits_data = load_digits()
data_items = digits_data.data
target = digits_data.target
```

source in digits_decision_tree.py

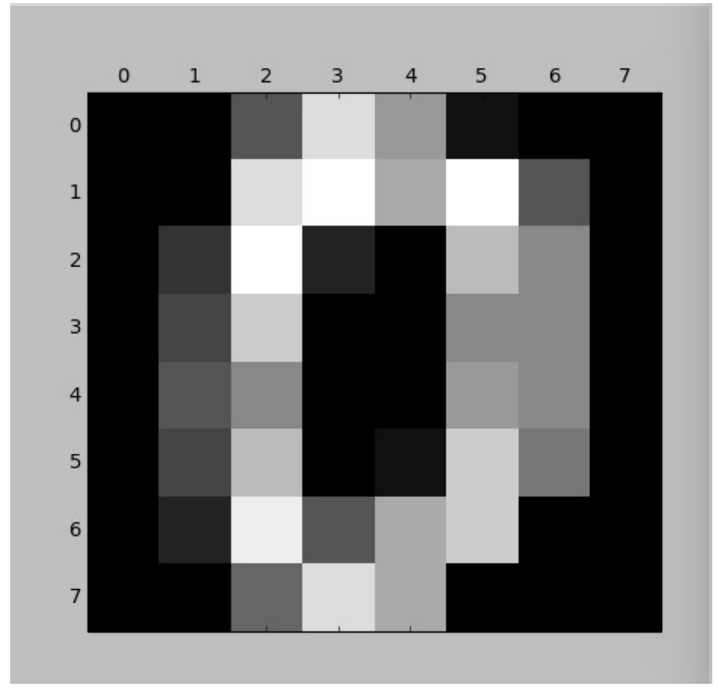
Review: Digits Dataset

```
print(data_items[0])

plt.gray()

plt.matshow(digits_data.images[0])

plt.show()
```




```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15.
10. 15.  5. 0.  0.  3. 15.  2.  0. 11.  8.  0.  0.
4. 12.  0.  0.  8. 8.  0.  0.  5.  8.  0.  0.  9.  8.
0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5.
10. 12.  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
```

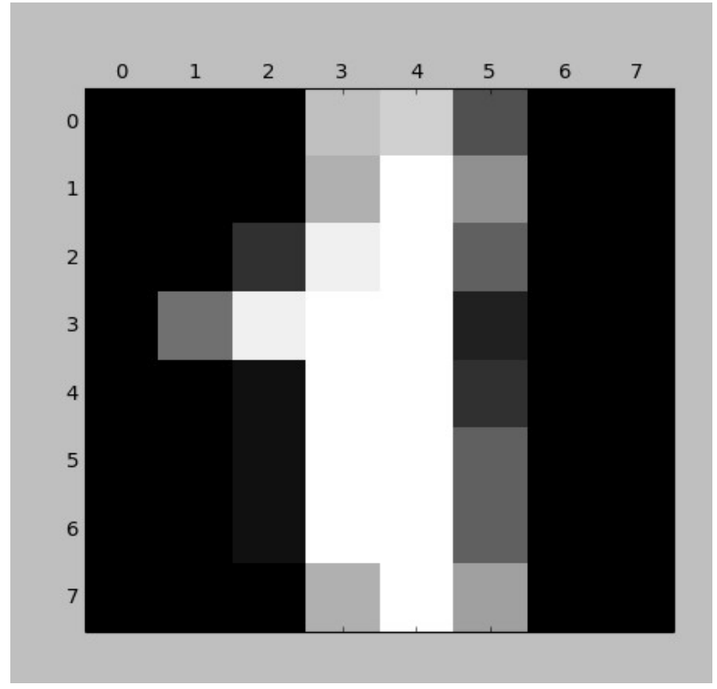
Review: Digits Dataset

```
print(data_items[1])

plt.gray()

plt.matshow(digits_data.images[1])

plt.show()
```




```
[ 0.  0.  0. 12. 13.  5.  0.  0.  0.  0.  0. 11.
16.  9.  0. 0.  0.  0.  3. 15. 16.  6.  0.  0.  0.  7.
15. 16. 16.  2. 0.  0.  0.  0.  1. 16. 16.  3.  0.
0.  0.  0.  1. 16. 16. 6.  0.  0.  0.  0.  1. 16. 16.
 6.  0.  0.  0.  0.  0. 11. 16. 10.  0.  0.]
```

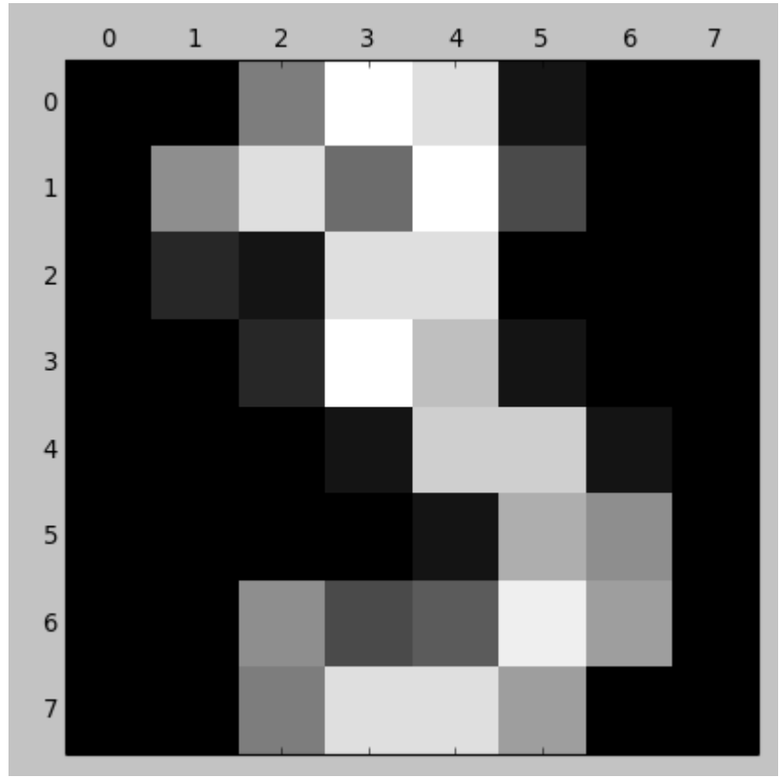
Review: Digits Dataset

```
print(data_items[3])

plt.gray()

plt.matshow(digits_data.images[3])

plt.show()
```




```
[ 0.  0.  7. 15. 13.  1.  0.  0.  0.  8. 13.  6. 15.
  4.  0.  0.  0.  2.  1. 13. 13.  0.  0.  0.  0.  0.
  2. 15. 11.  1.  0.  0.  0.  0.  0.  1. 12. 12.  1.  0.
  0.  0.  0.  0.  1. 10.  8.  0.  0.  0.  8.  4.  5.
 14.  9.  0.  0.  0.  7. 13. 13.  9.  0.  0.]
```

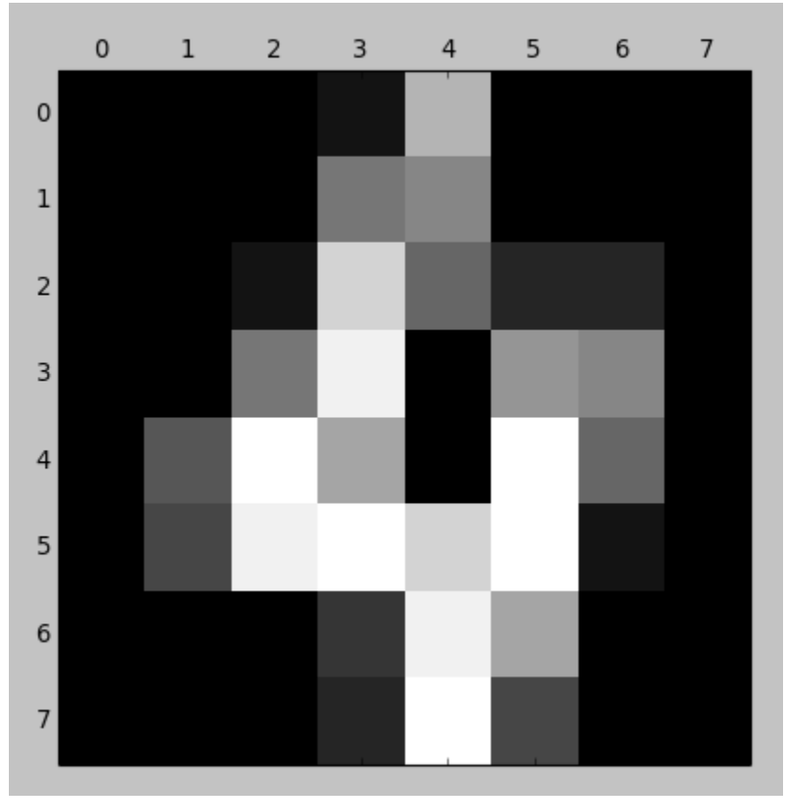
Review: Digits Dataset

```
print(data_items[4])

plt.gray()

plt.matshow(digits_data.images[4])

plt.show()
```




```
[ 0.  0.  0.  1. 11.  0.  0.  0.  0.  0.  0.  7.  8.
  0.  0.  0.  0.  0.  1. 13.  6.  2.  2.  0.  0.  0.  7.
 15.  0.  9.  8.  0.  0.  5. 16. 10.  0. 16.  6.  0.
  0.  4. 15. 16. 13. 16.  1.  0.  0.  0.  0.  3. 15.
 10.  0.  0.  0.  0.  0.  2. 16.  4.  0.  0.]
```

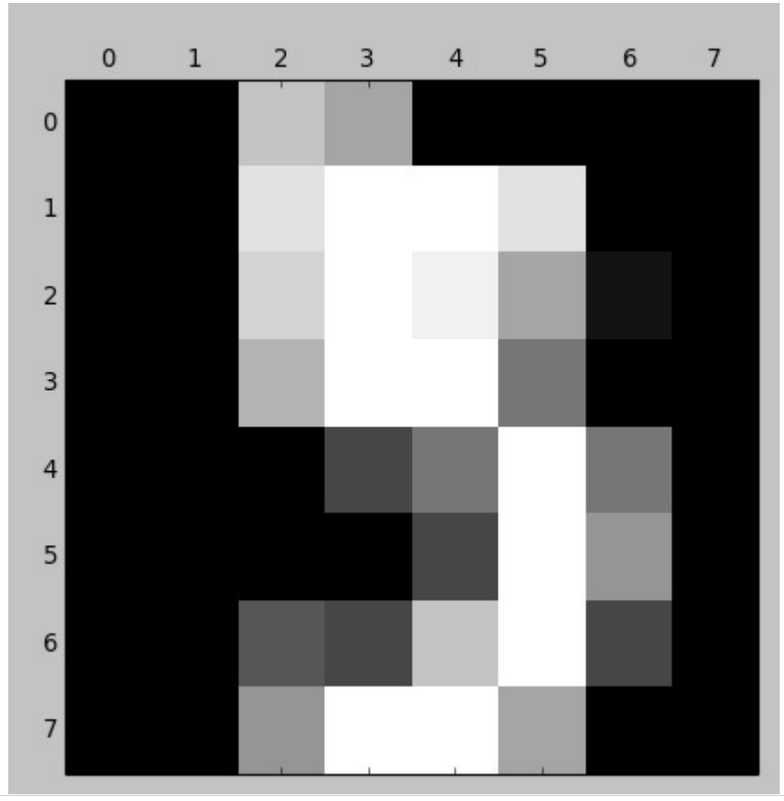
Review: Digits Dataset

```
print(data_items[5])

plt.gray()

plt.matshow(digits_data.images[5])

plt.show()
```




```
[ 0.  0. 12. 10.  0.  0.  0.  0.  0.  0. 14. 16.
16. 14.  0.  0.  0.  0. 13. 16. 15. 10.  1.  0.  0.
 0. 11. 16. 16.  7.  0.  0.  0.  0.  0.  4.  7. 16.
 7.  0.  0.  0.  0.  0.  4. 16.  9.  0.  0.  0.  5.
 4. 12. 16.  4.  0.  0.  0.  9. 16. 16. 10.  0.  0.]
```

Problem

Let's write a program that classifies digit images from the digits dataset with a random forest. Your program should take two command line parameters: the number of decision trees in the random forest and the number of classification experiments to run with the random forest. For each experiment, it should display a vector of results from all decision trees on a given digit and the digit's target.

sample call: $ python digits_random_forest.py -num_trees 30 -num_exps 100

# of trees in a forest

# of classification experiments

Solution

this is a decision tree classifier

list of decision trees

available test sizes

randomly choose proportion of data for testing

```
def create_random_forest(classifier, n, data_items, data_target):
    random_forest = [ ]
    test_sizes = (0.20, 0.25, 0.30, 0.35, 0.40)
    for i in xrange(n):
        tsize = random.choice(test_sizes)
        train_data, _, train_target, _= \
                train_test_split(data_items, data_target, test_size=tsize,
                                 random_state=random.randint(0, 1000))
        dtr = tree.DecisionTreeClassifier(random_state=random.randint(0, 1000))
        random_forest.append(dtr.fit(train_data, train_target))
    return random_forest
```

source in digits_random_forest.py

Solution

classify data_item with each decision tree in this random forest

```
def classify_with_random_forest(rand_forest, data_item):
    return [dt.predict(data_item)[0] for dt in rand_forest]

def random_forest_experiment(rand_forest, data_items, data_target):
    i = random.randint(0, data_items.shape[0] - 1)
    rf_classifications = classify_with_random_forest(rand_forest, data_items[i])
    return rf_classifications, data_target[i]

def run_random_forest_experiments(rand_forest, data_items, data_target, n):
    for _ in xrange(n):
        print(random_forest_experiment(rand_forest, data_items, data_target))
```

source in digits_random_forest.py

Solution

```
digits_data = load_digits()
digits_items = digits_data.data
digits_target = digits_data.target
clf = tree.DecisionTreeClassifier(random_state=0)

if __name__ == '__main__':
    import argparse
    ap = argparse.ArgumentParser()
    ap.add_argument('-num_trees', '--num_trees', help='number of trees',
                    type=int)
    ap.add_argument('-num_exps', '--num_exps', help='number of experiments',
                    type=int)
    args = vars(ap.parse_args())
    rf = create_random_forest(args['num_trees'], digits_items, digits_target)
    print(run_random_forest_experiments(rf, digits_items,
                                        digits_target,
                                        args['num_exps']))
```

grow a random forest
and run experiments with it

source in digits_random_forest.py

Test Run

$ python digits_random_forest.py -num_trees 10 -num_exps 10

```
([5, 5, 5, 5, 5, 5, 5, 5, 5, 5], 5)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 9)
([8, 8, 8, 8, 8, 8, 8, 8, 8, 8], 8)
([4, 0, 6, 4, 0, 6, 0, 6, 6, 6], 6)
([5, 5, 5, 5, 5, 5, 5, 5, 5, 5], 5)
([9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 9)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([8, 8, 2, 8, 8, 8, 1, 8, 8, 8], 8)
([9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 9)
```

Problem

Let's write a program that classifies iris flowers with a random forest. Your program should take two command line parameters: the number of decision trees in the random forest and the number of classification experiments to run with the random forest. For each experiment, it should display a vector of results from all decision trees on a given digit and the digit's target.

sample call: $ python iris_random_forest.py -num_trees 30 -num_exps 100

Solution

```
iris_data = load_iris()
iris_items = iris_data.data
iris_target = iris_data.target
clf = tree.DecisionTreeClassifier(random_state=0)

if __name__ == '__main__':
    import argparse
    ap = argparse.ArgumentParser()
    ap.add_argument('-num_trees', '--num_trees', help='number of trees',
                    type=int)
    ap.add_argument('-num_exps', '--num_exps', help='number of experiments',
                    type=int)
    args = vars(ap.parse_args())
    rf = create_random_forest(args['num_trees'], iris_items, iris_target)
    print(run_random_forest_experiments(rf, iris_items, iris_target,
                                        args['num_exps']))
```

source in iris_random_forest.py

Test Run

$ python iris_random_forest.py -num_trees 10 -num_exps 10

```
([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 0)
([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], 1)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], 1)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], 1)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], 2)
```

Using RandomForest from SKLEARN.ENSEMBLE

Problem

Let's write a program that classifies digits from the digits dataset with a random forest by using RandomForestClassifier from sklearn.ensemble. Your program should take two command line parameters: the number of random forests and the number of classification experiments to run with the random forest. For each experiment, it should display a vector of results from all decision trees on a given digit and the digit's target.

sample call: $ python digits_random_forest2.py -num_rfs 30 -num_exps 100

# of random forests

# of classification experiments

Solution

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import random
import sys

iris_data = load_iris()
iris_items = iris_data.data
iris_target = iris_data.target

def create_random_forests(n, data_items, data_target):
    random_forests = [ ]
    test_sizes = (0.20, 0.25, 0.30, 0.35, 0.40)
    for i in xrange(n):
        tsize = random.choice(test_sizes)
        train_data, _, train_target, _ = \
                train_test_split(data_items, data_target, test_size=tsize,
                                 random_state=random.randint(0, 1000))
        clf = RandomForestClassifier(n_estimators=random.randint(10, 20),
                                 random_state=random.randint(0, 1000))
        random_forests.append(clf.fit(train_data, train_target))
    return random_forests
```

Create a random forest classifier

source in iris_random_forest2_.py

Solution

```
def classify_with_random_forests(rand_forests, data_item):
    return [(rf.predict(data_item)[0], len(rf.estimators_)) for rf in rand_forests]

def random_forest_experiment(rand_forests, data_items, data_target):
    i = random.randint(0, data_items.shape[0] - 1)
    rf_classifications = classify_with_random_forests(rand_forests, data_items[i])
    return rf_classifications, data_target[i]

def run_random_forest_experiments(rand_forests, data_items, data_target, n):
    for _ in xrange(n):
        print(random_forest_experiment(rand_forests, data_items, data_target))
```

source in iris_random_forest2_.py

Solution

```
if __name__ == '__main__':
    import argparse
    ap = argparse.ArgumentParser()
    ap.add_argument('-num_rfs', '--num_rfs', help='number of random forests',
                    type=int)
    ap.add_argument('-num_exps', '--num_exps', help='number of experiments',
                    type=int)
    args = vars(ap.parse_args())
    rfs = create_random_forests(args['num_rfs'], iris_items, iris_target)
    print(run_random_forest_experiments(rfs, iris_items, iris_target,
                                        args['num_exps']))
```

source in iris_random_forest2_.py

Test Run

$ python iris_random_forest2.py -num_rfs 20 -num_exps 10

([(0, 11), (0, 15), (0, 14), (0, 16), (0, 19), (0, 16), (0, 15), (0, 13), (0, 20), (0, 20)], 0)

([(0, 11), (0, 15), (0, 14), (0, 16), (0, 19), (0, 16), (0, 15), (0, 13), (0, 20), (0, 20)], 0)

([(2, 11), (2, 15), (2, 14), (2, 16), (2, 19), (2, 16), (2, 15), (2, 13), (2, 20), (2, 20)], 2)

([(2, 11), (2, 15), (2, 14), (2, 16), (2, 19), (2, 16), (2, 15), (2, 13), (2, 20), (2, 20)], 2)

([(0, 11), (0, 15), (0, 14), (0, 16), (0, 19), (0, 16), (0, 15), (0, 13), (0, 20), (0, 20)], 0)

([(1, 11), (1, 15), (1, 14), (1, 16), (1, 19), (1, 16), (1, 15), (1, 13), (1, 20), (1, 20)], 1)

([(0, 11), (0, 15), (0, 14), (0, 16), (0, 19), (0, 16), (0, 15), (0, 13), (0, 20), (0, 20)], 0)

([(2, 11), (2, 15), (2, 14), (2, 16), (2, 19), (2, 16), (2, 15), (2, 13), (2, 20), (2, 20)], 2)

([(0, 11), (0, 15), (0, 14), (0, 16), (0, 19), (0, 16), (0, 15), (0, 13), (0, 20), (0, 20)], 0)

([(2, 11), (2, 15), (2, 14), (2, 16), (2, 19), (2, 16), (2, 15), (2, 13), (2, 20), (2, 20)], 2)

Test Run

$ python digits_random_forest2.py -num_rfs 10 -num_exps 10

([(0, 12), (0, 18), (0, 14), (0, 20), (0, 20), (0, 15), (0, 16), (0, 13), (0, 19), (0, 14)], 0)

([(5, 12), (5, 18), (5, 14), (5, 20), (5, 20), (5, 15), (5, 16), (5, 13), (5, 19), (5, 14)], 5)

([(9, 12), (9, 18), (9, 14), (9, 20), (9, 20), (9, 15), (9, 16), (9, 13), (9, 19), (9, 14)], 9)

([(0, 12), (0, 18), (0, 14), (0, 20), (0, 20), (0, 15), (0, 16), (0, 13), (0, 19), (0, 14)], 0)

([(5, 12), (5, 18), (5, 14), (5, 20), (5, 20), (5, 15), (5, 16), (5, 13), (5, 19), (5, 14)], 5)

([(7, 12), (7, 18), (7, 14), (7, 20), (7, 20), (7, 15), (7, 16), (7, 13), (7, 19), (7, 14)], 7)

([(6, 12), (6, 18), (6, 14), (6, 20), (6, 20), (6, 15), (6, 16), (6, 13), (6, 19), (6, 14)], 6)

([(8, 12), (8, 18), (8, 14), (8, 20), (8, 20), (8, 15), (8, 16), (1, 13), (8, 19), (8, 14)], 8)

([(0, 12), (0, 18), (0, 14), (0, 20), (0, 20), (0, 15), (0, 16), (0, 13), (0, 19), (0, 14)], 0)

([(9, 12), (9, 18), (9, 14), (9, 20), (9, 20), (9, 15), (9, 16), (9, 13), (9, 19), (9, 14)], 9)

Decision Tree on IRIS Data Set

```
>>> compute_cm_cr(dtr, 0.3)
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        18
          1       0.88      1.00      0.94        15
          2       1.00      0.83      0.91        12

avg / total       0.96      0.96      0.95        45


Confusion matrix:
[[18  0  0]
 [ 0 15  0]
 [ 0  2 10]]
```

Classification Report for IRIS

Confusion Matrix for IRIS

Decision Tree on Digits Dataset

	precision	recall	f1-score	support
0	1.00	0.95	0.98	44
1	0.71	0.78	0.74	58
2	0.91	0.83	0.87	48
3	0.77	0.78	0.77	59
4	0.76	0.84	0.80	57
5	0.91	0.88	0.90	49
6	0.94	0.85	0.89	59
7	0.85	0.87	0.86	52
8	0.73	0.77	0.75	47
9	0.82	0.81	0.81	67
avg / total	0.84	0.83	0.83	540

Confusion matrix:

```
[[42  0  0  0  1  0  0  0  1  0]
 [ 0 45  0  3  3  0  0  1  3  3]
 [ 0  4 40  0  0  0  1  0  3  0]
 [ 0  0  1 46  2  1  0  1  3  5]
 [ 0  3  0  0 48  1  1  2  0  2]
 [ 0  0  1  0  2 43  1  1  1  0]
 [ 0  1  0  0  5  1 50  0  1  1]
 [ 0  2  0  3  2  0  0 45  0  0]
 [ 0  5  2  2  0  0  0  1 36  1]
 [ 0  3  0  6  0  1  0  2  1 54]]
```

Testing Random Forests on Digits and IRIS Data Sets

$ python digits_random_forest.py -num_trees 10 -num_exps 1000
0.982
$ python digits_random_forest2.py -num_rfs 10 -num_exps 1000
acc = 0.998

$ python iris_random_forest.py -num_items 10 -num_exps 1000
acc = 0.983

$ python iris_random_forest2.py -num_rfs 10 -num_exps 1000
acc = 0.983

Conclusions

- Random forests improve accuracy over single decision trees on digits data set
- Random forests slightly improve accuracy over single decision trees on IRIS data set