```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import StratifiedKFold, train_test_split
from sklearn import metrics

def cv_error(clf, X, y, k=5):
    """
        Splits the data, X and y, into k-folds. Trains a classifier on K-1
            folds and
        testing on the remaining fold. Calculates the cross validation train
            and test
        error for classifier clf by averaging the performance across folds.
        Input:
          clf- an instance of any classifier
          X- (n,d) array of feature vectors, where n is # of examples
             and d is # of features
          y- (n,) array of binary labels {1,-1}
          k- int, # of folds (default=5)

        Returns: average train and test error across the k folds as np.float64
    """
    skf = StratifiedKFold(y, k)
    train_scores, test_scores = [], []
    for train, test in skf:
        X_train, X_test, y_train, y_test = X[train], X[test], y[train], y[test]
        clf.fit(X_train, y_train)
        train_scores.append(clf.score(X_train, y_train))
        test_scores.append(clf.score(X_test, y_test))

    return 1 - np.array(train_scores).mean(), 1 - np.array(test_scores).mean()


def plot_error(X, y):
    '''
    Plots the variation of 5-fold cross-validation error w.r.t. maximum
    depth of the decision tree
        X- (n, d) array of feature vectors, where n is # of examples
             and d is # of features
        y- (n, ) array of labels corresponding to X
    '''

    # ---------- make your implementation here---------------
    train_error = []
    test_error = []
    max_depth = list(range(1, 21))
    for i in range(1,21):
        clf = DecisionTreeClassifier(criterion="entropy", max_depth=i)
        result = cv_error(clf, X, y, 5)
        train_error.append(result[0])
        test_error.append(result[1])
    plt.plot(max_depth, train_error, label="train error")
    plt.plot(max_depth, test_error, label="test error")
    plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)
```

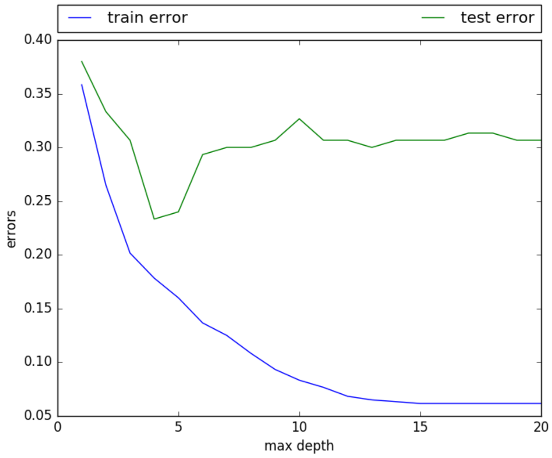
```
    plt.ylabel('errors')
    plt.xlabel('max depth')
    plt.savefig('max_depth_vs_error.png')
    #plt.show()
    # ------------------------------------------------------

def majority_vote(pred_list):
    '''
    Given a list of m (n, ) arrays, each (n, ) array containing the predictions
        for
    same set of n examples (using different classifiers), return a (n, ) array
    containing majority vote prediction of the m (n, ) arrays
    Input:
        pred_list- a list of m (n, ) arays
    Output:
        y_pred- (n, ) array containing majority vote prediction using pred_list
    '''
    y_pred = []
    for i in range(len(pred_list[0])):
        lst = [row[i] for row in pred_list]
        y_pred.append(max(set(lst), key = lst.count))

    return y_pred


def bagging_ensemble(X_train, y_train, X_test, y_test, m = None, n_clf = 10):
    '''
    Returns accuracy on the test set X_test with corresponding labels y_test
    using a bagging ensemble classifier with n_clf decision trees trained with
    training examples X_train and training labels y_train.
    Input:
        X_train- (n_train, d) array of training feature vectors, where n_train
            is # of examples and d is # of features
        y_train- (n_train, ) array of labels corresponding to X_train
        X_test- (n_test, d) array of testing feature vectors, where n_test is
            # of examples and d is # of features
        y_test- (n_test, ) array of labels corresponding to X_test
        m - int, # of features to consider when looking for the best split
        n_clf- # of decision tree classifiers in the bagging ensemble, default
            value of n_clf is 10
    Output:
        Accuracy of the bagging ensemble classifier on X_test
    '''

    # ---------- make your implementation here---------------
    bagging_ratio = 0.6
    total_size = y_train.shape[0]
    bagging_size = int(bagging_ratio * total_size)
    bagsX = []
    bagsY = []
    for i in range(0, n_clf):
        tempx = []
        tempy = []
        for j in range(0, bagging_size):
            rand = np.random.randint(0, total_size)
            tempx.append(X_train[rand, :])
```

```python
            tempy.append(y_train[rand])
        bagsX.append(tempx)
        bagsY.append(tempy)
    clfs = []
    accuracy = 0
    for i in range(0, n_clf):
        clfs.append(DecisionTreeClassifier(max_features=m));
    for i in range(0, n_clf):
        clfs[i].fit(bagsX[i], bagsY[i])
    preds = []
    for i in range(0, n_clf):
        temp = clfs[i].predict(X_test)
        preds.append(temp)
    y_pred = majority_vote(preds)
    count = 0
    for i in range(0, y_test.shape[0]):
        if y_pred[i] == y_test[i]:
            count = count + 1
    accuracy = count / y_test.shape[0]
    return accuracy
    # -------------------------------------------------------


def random_forest(X_train, y_train, X_test, y_test, m, n_clf = 10):
    '''
    Returns accuracy on the test set X_test with corresponding labels y_test
    using a random forest classifier with n_clf decision trees trained with
    training examples X_train and training labels y_train.
    Input:
        X_train- (n_train, d) array of training feature vectors, where n_train
            is # of examples and d is # of features
        y_train- (n_train, ) array of labels corresponding to X_train
        X_test- (n_test, d) array of testing feature vectors, where n_test is
            # of examples and d is # of features
        y_test- (n_test, ) array of labels corresponding to X_test
        n_clf- # decision tree classifiers in the random forest, default
            value of n_clf is 10
    Output:
        Accuracy of the random forest classifier on X_test
    '''

    # ---------- make your implementation here---------------
    accuracy = bagging_ensemble(X_train, y_train, X_test, y_test, m, n_clf)
    return accuracy
    # -------------------------------------------------------


def plot_histograms(random_forest_scores, bagging_scores):
    '''
    Plots histogram of values in random_forest_scores and bagging_scores
    overlayed on top of each other
    Input:
        random_forest_scores- a list containing accuracy values for random
            forest classifier
        for 100 different train and test set splits
        bagging_scores- a list containing accuracy values for bagging ensemble
```

```
            classifier
        using decision trees for the same 100 different train and test set
            splits
        as random_forest_scores
    '''
    bins = np.linspace(0.8, 1.0, 100)
    plt.figure()
    plt.hist(random_forest_scores, bins, alpha=0.5, label='random forest')
    plt.hist(bagging_scores, bins, alpha=0.5, label='bagging')
    plt.xlabel('Accuracy')
    plt.ylabel('Frequency')
    plt.legend(loc='upper left')
    plt.savefig('histogram.png')
    print("all done")
    #plt.show()


def q1a():
    # Load Iris dataset
    iris = load_iris()
    X, y = iris.data[:,:2], iris.target
    # PLot cross-validation error vs max_depth
    plot_error(X, y)


def q1b():
    # Load digits dataset
    digits = load_digits(4)
    X, y = digits.data, digits.target

    # Calculate accuracy of bagging ensemble and random forest for 100 random
        train/test splits
    # Analyze how the performance of bagging & random forest changes with m
    results1, results2 = [], []
    for j in range(0, 64, 2):
        print (j / 64 * 100, "%")
        bagging_scores, random_forest_scores = [], []
        for i in range(100):
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                = 0.2)
            random_forest_scores.append(random_forest(X_train, y_train, X_test,
                y_test, j+1))
            bagging_scores.append(bagging_ensemble(X_train, y_train, X_test,
                y_test))
        results1.append(np.median(np.array(random_forest_scores)))
        results2.append(np.median(np.array(bagging_scores)))

    plt.figure()
    plt.plot(range(0, 64, 2), results1, '--', label = 'random forest')
    plt.plot(range(0, 64, 2), results2, '--', label = 'bagging')
    plt.xlabel('m')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.savefig('second_plot.png')
    #plt.show()
```

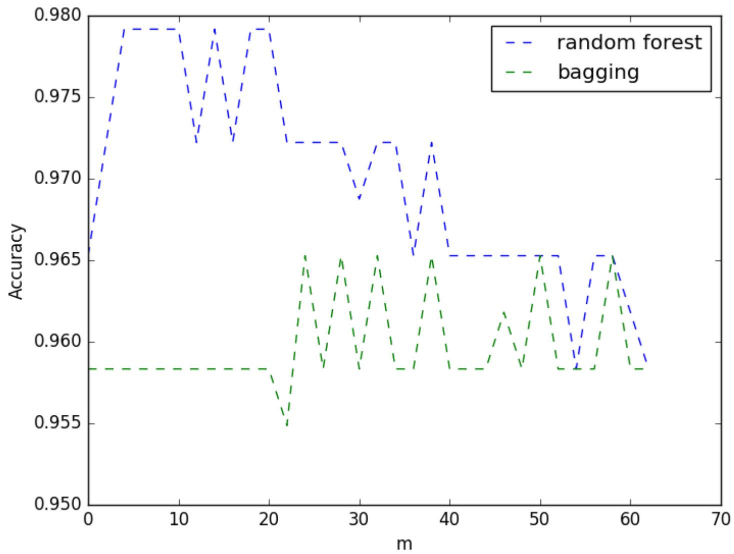
```
    print("second plot finished, on to histogram")

    #Plot histogram of performances for m = 8
    bagging_scores, random_forest_scores = [], []
    for i in range(100):
        print(i, "%")
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
            )
        random_forest_scores.append(random_forest(X_train, y_train, X_test,
            y_test,8))
        bagging_scores.append(bagging_ensemble(X_train, y_train, X_test, y_test
            ))
    plot_histograms(random_forest_scores, bagging_scores)

if __name__ == '__main__':
    q1a()
    q1b()
```

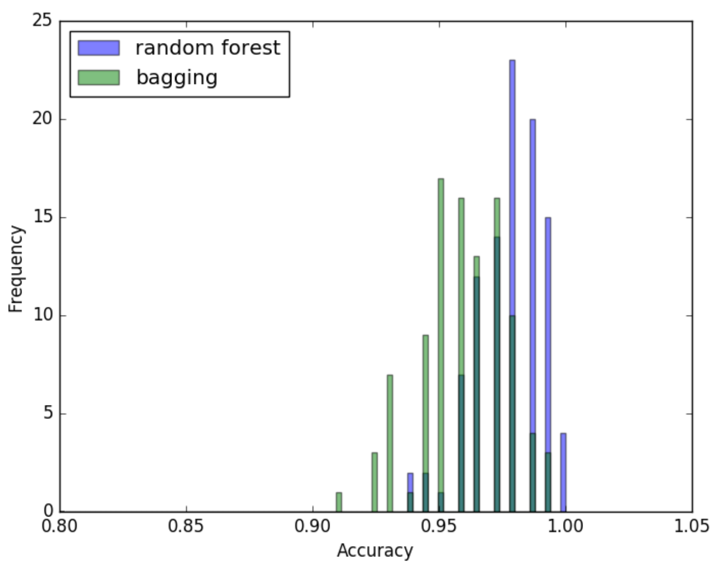
Problem 1




Looks like a depth of 4 works pretty well

Problem 2



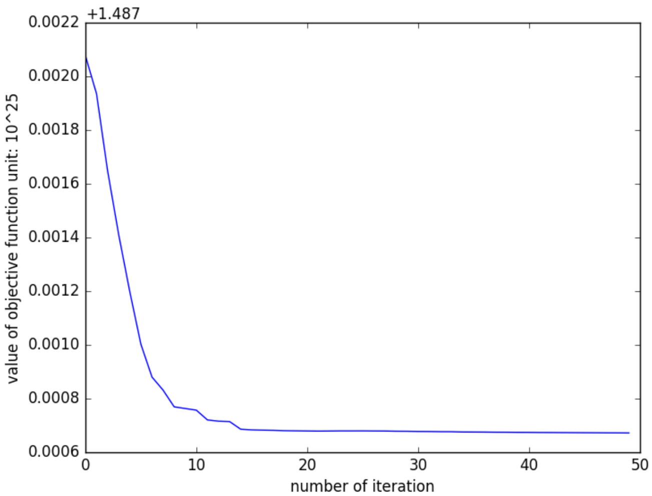

m = 15 is ok, as m becomes larger, the performance of random forest actually decreases.

```python
from __future__ import division
from scipy.ndimage import imread
import numpy as np
from matplotlib import pyplot as plt

def compute_distances_using_numpy(X, ClusterMeans, n, k):
    X_row_norms = np.linalg.norm(X) **2
    M_row_norms = np.linalg.norm(ClusterMeans) **2
    D = (np.outer(X_row_norms, np.ones(k)) + np.outer(np.ones(n), M_row_norms)
        - 2 * np.dot(X, ClusterMeans.T))
    return D

# Load the mandrill image as an NxNx3 array. Values range from 0.0 to 255.0.
mandrill = imread('mandrill.png', mode='RGB').astype(float)

N = int(mandrill.shape[0])

M = 2
k = 64

# Store each MxM block of the image as a row vector of X
X = np.zeros((N**2//M**2, 3*M**2))
for i in range(N//M):
    for j in range(N//M):
        X[i*N//M+j,:] = mandrill[i*M:(i+1)*M,j*M:(j+1)*M,:].reshape(3*M**2)


# TODO: Implement k-means and cluster the rows of X, then reconstruct the
# compressed image using the cluster center for each block, as specified in
# the homework description.
size = X.shape[1]
n = X.shape[0]
cluster_pool = np.zeros([k, size])
cluster_pool_save = np.zeros([k, size])
blank = []
points_pool = []
points_pool_save = []

#prepare the initial random clusters
for i in range(0, k):
    rand = np.random.randint(0, n)
    cluster_pool[i, :] = X[rand, :]

#initialize point pool
for i in range(0, k):
    temp = []
    points_pool.append(temp)
    blank.append(temp)

count = 0
objective = []
iteration = []

#while not np.array_equal(cluster_pool, cluster_pool_save):
while count < 50:
```

```
        #associate each point with a cluster_pool
        print("count: ", count)
        iteration.append(count)

        distance = compute_distances_using_numpy(X, cluster_pool, n, k)

        points_pool = blank

        sum_distance = 0
        for i in range(0, n):
            min_val = float('inf')
            min_index = 0
            for j in range(0, k):
                if distance[i][j] < min_val:
                    min_val = distance[i][j]
                    min_index = j
            sum_distance = sum_distance + min_val * min_val
            points_pool[min_index].append(i)

        print("value of objective function: ", sum_distance)
        objective.append(sum_distance / (10**25))

        #update cluster using the mean of all the associated points
        cluster_pool_save = np.array(cluster_pool)
        for i in range(0, k):
            total = np.zeros([1, size])
            totalnumber = len(points_pool[i])
            if totalnumber != 0:
                for j in range(0, totalnumber):
                    index = points_pool[i][j]
                    total = total + X[index, :]
                total = total / totalnumber
                cluster_pool[i, :] = total

        count = count + 1

    for i in range(0, k):
        total = np.zeros([1, size])
        for j in range(0, len(points_pool[i])):
            index = points_pool[i][j]
            X[index, :] = cluster_pool[i, :]

    mandrill_cpy = np.array(mandrill)

    for i in range(N//M):
        for j in range(N//M):
            mandrill_cpy[i*M:(i+1)*M,j*M:(j+1)*M,:] = X[i*N//M+j,:].reshape(M,M,3)

    difference = np.array(mandrill)
    for i in range(mandrill.shape[0]):
        for j in range(mandrill.shape[1]):
            for k in range(mandrill.shape[2]):
                difference[i][j][k] = mandrill[i][j][k] - mandrill_cpy[i][j][k] +
                    128

    sum_abs = 0
```

```
for i in range(0, N):
    for j in range(0, N):
        for k in range(0, 3):
            sum_abs = sum_abs + np.abs(mandrill_cpy[i][j][k] - mandrill[i][j][k
                ])
absolute_error = sum_abs / (255 * 3 * N * N)
print("relative absolute error: ", absolute_error)

# To show a color image using matplotlib, you have to restrict the color
# color intensity values to between 0.0 and 1.0. For example,
plt.plot(iteration, objective)
plt.ylabel('value of objective function unit: 10^25')
plt.xlabel('number of iteration')
plt.savefig('error_curve.png')
plt.show()

plt.imshow(mandrill/255)
plt.savefig('original.png')
#plt.show()
plt.imshow(mandrill_cpy/255)
plt.savefig('compressed.png')
#plt.show()
plt.imshow(difference/255)
plt.savefig('difference.png')
#plt.show()
```

Problem 3




the nose is more clear than other part. The part with more contrast will be better preserved.

compression ratio = 6.3%

absolute error is 0.158

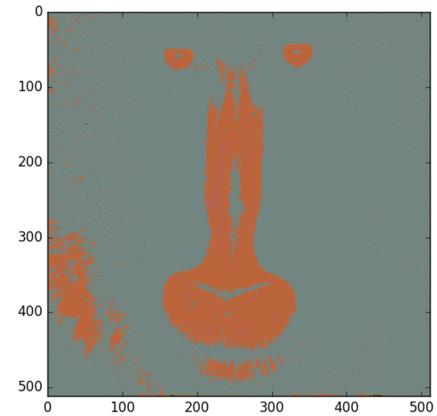
n = 512.0;

M = 2;

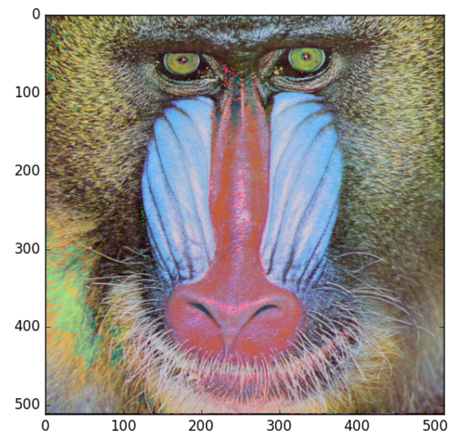
k = 64;

$$\frac{k}{n^2}+\frac{\mathrm{Log}[2, k]}{24\,M^2}$$

0.0627441




compressed




difference

Problem 4.

original → 24 bits per pixel, 8 bits /color

image → Size $N \times N$

$M, N, k$

for $M$, each $M \times M$ block can be considered as 1 pixel

Now the resolution is $(\frac{N}{M})^2$ actually

For each block we need to store which cluster it belongs to which takes $\log_2 k$ bits.

And we need to store the color for clusters, $k \times 24$ bits/pixel which is $24k$ bits.

All together ⇒ $\frac{24k + \log_2 k \times (\frac{N}{M})^2}{N^2} = \frac{24k}{N^2} + \frac{\log_2 k}{M^2}$ bits/pixel

Compression ratio $= (\frac{24k}{N^2} + \frac{\log_2 k}{M^2}) / 24 = \frac{k}{N^2} + \frac{\log_2 k}{24M^2}$

Problem 5

| | | | | | |
|---|---|---|---|---|---|
| 42 | new | KevinRabideau | 0.66090 | 5 | Fri, 11 Nov 2016 23:57:39 (-1.6h) |
| 43 | ↓19 | Sahil Misra | 0.66531 | 2 | Wed, 05 Oct 2016 22:02:47 (-0h) |
| 44 | ↓19 | GuangyuWang | 0.66364 | 1 | Mon, 31 Oct 2016 22:16:57 |
| 45 | ↓19 | cneeruko | 0.66333 | 8 | Wed, 09 Nov 2016 18:13:13 (-35.6d) |
| 46 | ↓19 | Redmond Xia | 0.66298 | 1 | Thu, 03 Nov 2016 02:35:25 |
| 47 | new | minkyan | 0.66220 | 12 | Mon, 14 Nov 2016 02:46:08 (-2.6h) |
| **48** | new | **TianhaoZhou** | **0.66137** | **5** | **Mon, 14 Nov 2016 06:57:15** |

**Your Best Entry ↑**

You improved on your best score by 0.00363.

You just moved up 2 positions on the leaderboard. Tweet this!

| | | | | | |
|---|---|---|---|---|---|
| 49 | new | wyjin | 0.66057 | 8 | Mon, 14 Nov 2016 02:32:50 |
| 50 | new | amang | 0.65958 | 13 | Mon, 14 Nov 2016 04:50:50 (-0.1h) |
| 51 | new | chengyqi | 0.65773 | 18 | Sun, 13 Nov 2016 23:53:58 (-2.2d) |
| 52 | new | yqzeng | 0.65648 | 9 | Sun, 13 Nov 2016 00:19:56 (-0.3h) |
| 53 | new | PavanShah | 0.65634 | 5 | Sun, 13 Nov 2016 05:52:50 (-0.4h) |
| 54 | new | DuncanFairbanks | 0.65493 | 5 | Mon, 14 Nov 2016 05:30:54 (-0.6h) |