CS109A Introduction to Data Science:

Homework 8: Ensembles: Bagging, Random Forests, and Boosting

Harvard University
Fall 2018
Instructors: Pavlos Protopapas, Kevin Rader

In [1]:
```
#RUN THIS CELL
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-C
HTML(styles)
```
Out[1]:

INSTRUCTIONS

- To submit your assignment follow the instructions given in Canvas (https://canvas.harvard.edu/courses/42693/pages/homework-policies-and-submission-instructions).
- If needed, clarifications will be posted on Piazza.
- This homework can be submitted in pairs.
- If you submit individually but you have worked with someone, please include the name of your **one** partner below.

Name of the person you have worked with goes here:

Learning Objectives

Completing this assignment will demonstrate success at the following objectives:

- Statistical
  - Predict when bagging will help model performance.
  - Identify how Random Forests improve over bagging.
  - Predict when boosting will help model performance.
  - Compare and contrast bagging and boosting.
- Coding

    - Identify and fix problems in poorly written code
- Communication
    - Visually explain a complex concept

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import cross_val_score
from sklearn.utils import resample
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

%matplotlib inline

import seaborn as sns
sns.set(style='whitegrid')
pd.set_option('display.width', 1500)
pd.set_option('display.max_columns', 100)
```

Overview: Higgs Boson Discovery

The discovery of the Higgs boson in July 2012 marked a fundamental breakthrough in particle physics. The Higgs boson particle was discovered through experiments at the Large Hadron Collider at CERN, by colliding beams of protons at high energy. A key challenge in analyzing the results of these experiments is to differentiate between collisions that produce Higgs bosons and collisions that produce only background noise. We shall explore the use of ensemble methods for this classification task.

You are provided with data from Monte-Carlo simulations of collisions of particles in a particle collider experiment. The training set is available in `Higgs_train.csv` and the test set is in `Higgs_test.csv`. Each row in these files corresponds to a particle collision described by 28 features (columns 1-28), of which the first 21 features are kinematic properties measured by the particle detectors in the accelerator, and the remaining features are derived by physicists from the first 21 features. The class label is provided in the last column, with a label of 1 indicating that the collision produces Higgs bosons (signal), and a label of 0 indicating that the collision produces other particles (background).

The data set provided to you is a small subset of the HIGGS data set in the UCI machine learning repository. The following paper contains further details about the data set and the predictors used: Baldi et al., Nature Communications 5, 2014 (https://www.nature.com/articles/ncomms5308).

In [3]:
```
# Load data
data_train = pd.read_csv('data/Higgs_train.csv')
data_test = pd.read_csv('data/Higgs_test.csv')

print(f"{len(data_train)} training samples, {len(data_test)} test samples")
print("\nColumns:")
print(', '.join(data_train.columns))
```

```
5000 training samples, 5000 test samples

Columns:
lepton pT, lepton eta, lepton phi, missing energy magnitude, missing ener
gy phi, jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta,
jet 2 phi, jet 2 b-tag, jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet
4 pt, jet 4 eta, jet 4 phi, jet 4 b-tag, m_jj, m_jjj, m_lv, m_jlv, m_bb,
m_wbb, m_wwbb, class
```

In [4]:
```
display(data_train.head())
display(data_train.describe())
```

	lepton pT	lepton eta	lepton phi	missing energy magnitude	missing energy phi	jet 1 pt	jet 1 eta	jet 1 phi	jet 1 b-tag	jet 2 pt	jet 2 eta	jet 2 phi
0	0.377	-1.5800	-1.7100	0.991	0.114	1.250	0.620	-1.480	2.17	0.754	0.7750	-0.667
1	0.707	0.0876	-0.4000	0.919	-1.230	1.170	-0.553	0.886	2.17	1.300	0.7620	-1.060
2	0.617	0.2660	-1.3500	1.150	1.040	0.955	0.377	-0.148	0.00	1.060	-0.0194	1.110
3	0.851	-0.3810	-0.0713	1.470	-0.795	0.692	0.883	0.497	0.00	1.620	0.1240	1.180
4	0.768	-0.6920	-0.0402	0.615	0.144	0.749	0.397	-0.874	0.00	1.150	0.1270	1.320

	lepton pT	lepton eta	lepton phi	missing energy magnitude	missing energy phi	jet 1 pt	jet 1 eta
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	0.978645	-0.014280	-0.018956	1.005793	0.002528	0.980390	0.025014
std	0.547025	1.011927	0.997945	0.591907	1.003337	0.463677	1.002018
min	0.275000	-2.410000	-1.740000	0.010000	-1.740000	0.170000	-2.920000
25%	0.587000	-0.764250	-0.877500	0.581000	-0.870000	0.676000	-0.659250
50%	0.846000	-0.009305	-0.016050	0.903500	0.001300	0.891000	0.049500
75%	1.220000	0.725500	0.837000	1.300000	0.866000	1.160000	0.716000
max	5.330000	2.430000	1.740000	6.260000	1.740000	4.190000	2.960000

In [5]:
```
# Split into NumPy arrays
X_train = data_train.iloc[:, data_train.columns != 'class'].values
y_train = data_train['class'].values
X_test = data_test.iloc[:, data_test.columns != 'class'].values
y_test = data_test['class'].values
```

Question 1: A Single Model [20 pts]

We start by fitting a basic model we can compare the other models to. We'll pick a decision tree as the base model, because we'll later include random forests and want a fair comparison. We'll tune the decision tree using cross-validation. As usual, we'll be tuning the maximum tree depth; we refer to this parameter as "depth" for simplicity.

Since we will only be using tree-based methods in this homework, we do not need to standardize or normalize the predictors.

1.1: Fit a decision tree model to the training set. Choose a range of tree depths to evaluate. Plot the estimated performance +/- 2 standard deviations for each depth using 5-fold cross validation. Also include the training set performance in your plot, but set the y-axis to focus on the cross-validation performance.

Hint: use `plt.fill_between` to shade the region.

1.2 Select an appropriate depth and justify your choice. Using your cross-validation estimates, report the mean +/- 2 stdev. Then report the classification accuracy on the test set. (Store the training and test accuracies in variables to refer to in a later question.)

1.3 What is the mechanism by which limiting the depth of the tree avoids over-fitting? What is one downside of limiting the tree depth? Your answer should refer to the bias-variance trade-off.

Answers

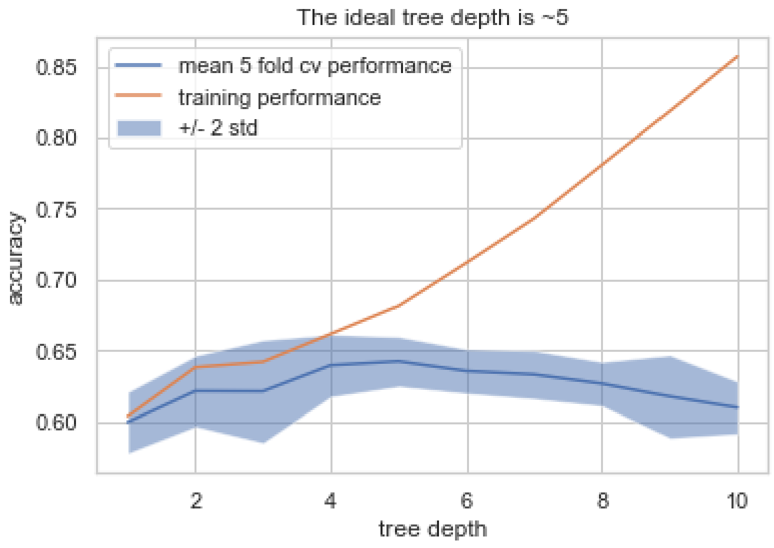
1.1 Fit a decision tree model to the training set. Choose a range of tree depths to evaluate. Plot the estimated performance +/- 2 standard deviations for each depth using 5-fold cross validation. Also include the training set performance in your plot, but set the y-axis to focus on the cross-validation performance.

Hint: use `plt.fill_between` to shade the region.

In [42]:
```python
# your code here
depth = range(1,11)
results = []
training_results = []
for d in depth:
    results.append(cross_val_score(DecisionTreeClassifier(max_depth = d),
                                   data_train.drop('class', axis = 1),
                                   data_train['class'],
                                   cv = 5,
                                   )
                  )
    pred = (DecisionTreeClassifier(max_depth = d)
            .fit(data_train.drop('class', axis = 1), data_train['class'])
            .predict(data_train.drop('class', axis = 1))
           )
    training_results.append(accuracy_score(data_train['class'], pred))
```

In [43]:
```
lower = [np.mean(r)-2*np.std(r) for r in results]
upper = [np.mean(r)+2*np.std(r) for r in results]
mean = [np.mean(r) for r in results]
```

In [44]:
```
plt.plot(depth, mean, label = 'mean 5 fold cv performance')
plt.plot(depth, training_results, label = 'training performance')
plt.fill_between(depth, lower, upper, alpha = .5, label = '+/- 2 std')
plt.xlabel('tree depth')
plt.ylabel('accuracy')
plt.title('The ideal tree depth is ~5')
plt.legend()
plt.show()
```



1.2 Select an appropriate depth and justify your choice. Using your cross-validation estimates, report the mean +/- 2 stdev. Then report the classification accuracy on the test set. (Store the training and test accuracies in variables to refer to in a later question.)

The best performance seems to be at depth 5 because the cross validation score is highest at 0.64. The +/- 2 stdev range is [0.62 , 0.66].

In [46]:
```
train_acc = training_results[4]
print("The training set accuracy is", train_acc)
```

```
The training set accuracy is 0.6812
```

In [45]:
```
# your code here
pred = (DecisionTreeClassifier(max_depth = 5)
             .fit(data_train.drop('class', axis = 1), data_train['class'])
             .predict(data_test.drop('class', axis = 1))
           )
test_acc = accuracy_score(data_test['class'], pred)
print("The test set accuracy is", test_acc)
```

```
The test set accuracy is 0.648
```

1.3 What is the mechanism by which limiting the depth of the tree avoids over-fitting? What is one downside of limiting the tree depth? Your answer should refer to the bias-variance trade-off.

your answer here What is the mechanism by which limiting the depth of the tree avoids over-fitting?

By limiting the depth of the tree, you limit the number of splits the model can apply to your data. This reducees the flexibility of the model, hence increasing bias and reducing variance, which in turn prevents over-fitting.

What is one downside of limiting the tree depth?

A downside of limiting the tree depth is that if your model is overly restricted, it may potentially not capture the true variations in the data (it will be overly biased).

Question 2: Bagging [25 pts]

Bagging is the technique of building the same model on multiple bootstraps from the data and combining each model's prediction to get an overall classification. In this question we build an example by hand and study how the number of bootstrapped datasets impacts the accuracy of the resulting classification.

2.1 Choose a tree depth that will overfit the training set. What evidence leads you to believe that this depth will overfit? Assign your choice to a variable here. (You may want to explore different settings for this value in the problems below.)

2.2 Create 45 bootstrapped replications of the original training data, and fit a decision tree to each. Use the tree depth you just chose in 2.1. Record each tree's prediction. In particular, produce a dataset like those below, where each row is a training (or test) example, each column is one of the trees, and each entry is that tree's prediction for that example. (Labeling the rows and columns is optional.)

Store these results as `bagging_train` and `bagging_test` . Don't worry about visualizing these results yet.

2.3 *Aggregate* all 45 *bootstrapped* models to get a combined prediction for each training and test point: predict a 1 if and only if a majority of the models predict that example to be from class 1. What accuracy does this *bagging* model achieve on the test set? Write an assertion that verifies that this test-set accuracy is at least as good as the accuracy for the model you fit in Question 1.

2.4 We want to know how the number of bootstraps affects our bagging ensemble's performance. Use the `running_predictions` function (given below) to get the model's accuracy score when using only 1,2,3,4,... of the bootstrapped models. Make a plot of training and test set accuracies as a function of number of bootstraps.

On your plot, also include horizontal lines for two baselines:

- the test accuracy of the best model from question 1
- the test accuracy of a single tree with the tree depth you chose in 2.1, trained on the full training set.

2.5 Referring to your graph from 2.4, compare the performance of bagging against the baseline of a single depth-10 tree. Explain the differences you see.

2.6 Bagging and limiting tree depth both affect how much the model overfits. Compare and contrast these two approaches. Your answer should refer to your graph in 2.4 and may duplicate something you said in your answer to 1.5.

2.7: In what ways might our bagging classifier be overfitting the data? In what ways might it be underfitting?

Hints

- Use `resample` from sklearn to easily bootstrap the x and y data.
- use `np.mean` to easily test for majority. If a majority of models vote 1, what does that imply about the mean?

Answers:

2.1 Choose a tree depth that will overfit the training set. What evidence leads you to believe that this depth will overfit? Assign your choice to a variable here. (You may want to explore different settings for this value in the problems below.)

Your answer here
I will choose a depth of 10, because the performance on the training data is much higher than the performance on the cross validation.

```
In [139]: # your code here
overfit_depth = 10
```

2.2 Create 45 bootstrapped replications of the original training data, and fit a decision tree to each. Use the tree depth you just chose in 2.1. Record each tree's prediction. In particular, produce a dataset like those below, where each row is a training (or test) example, each column is one of the trees, and each entry is that tree's prediction for that example. (Labeling the rows and columns is optional.)

Structure of `bagging_train` and `bagging_test`:

`bagging_train`:

|bootstrap model 1's prediction	bootstrap model 2's prediction	...	bootstrap model 45's prediction
training row 1	binary value	binary value	...
training row 2	binary value	binary value	...
...	...	...	...

`bagging_test`:

|bootstrap model 1's prediction	bootstrap model 2's prediction	...	bootstrap model 45's prediction
test row 1	binary value	binary value	...
test row 2	binary value	binary value	...
...	...	...	...

In [140]:
```
# your code here
bootstraps = 45
boot_data_train = [data_train.sample(frac = 1, replace = True) for _ in ra
models = [DecisionTreeClassifier(max_depth=overfit_depth).fit(data.drop('c
predictions_train = [model.predict(data_train.drop('class', axis = 1)) for
predictions_test = [model.predict(data_test.drop('class', axis = 1)) for m
```

In [141]:
```
train_bag_df = pd.DataFrame(predictions_train).T
train_bag_df.head()
```

Out[141]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
1	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0
2	1.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0
3	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0

In [142]:
```
test_bag_df = pd.DataFrame(predictions_test).T
test_bag_df.head()
```

Out[142]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0
1	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
2	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0
3	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0

2.3 Aggregate all 45 bootstrapped models to get a combined prediction for each training and test point: predict a 1 if and only if a majority of the models predict that example to be from class 1. What accuracy does this bagging model achieve on the test set? Write an assertion that verifies that this test-set accuracy is at least as good as the accuracy for the model you fit in Question 1.

In [143]:
```
# your code here
train_pred_bag = train_bag_df.apply(lambda row: int(np.mean(row) > .5), ax
test_pred_bag = test_bag_df.apply(lambda row: int(np.mean(row) > .5), axis
```

In [204]:
```
train_bag_acc = accuracy_score(train_pred_bag, data_train['class'])
train_bag_acc
```

Out[204]:
```
0.9346
```

What accuracy does this bagging model achieve on the test set?

In [144]:
```
test_bag_acc = accuracy_score(test_pred_bag, data_test['class'])
test_bag_acc
```

Out[144]:
```
0.6886
```

In [146]:
```
assert test_bag_acc >= test_acc
```

2.4 We want to know how the number of bootstraps affects our bagging ensemble's performance. Use the running_predictions function (given below) to get the model's accuracy score when using only 1,2,3,4,... of the bootstrapped models. Make a plot of training and test set accuracies as a function of number of bootstraps.

On your plot, also include horizontal lines for two baselines:

the test accuracy of the best model from question 1 the test accuracy of a single tree with the tree depth you chose in 2.1, trained on the full training set.

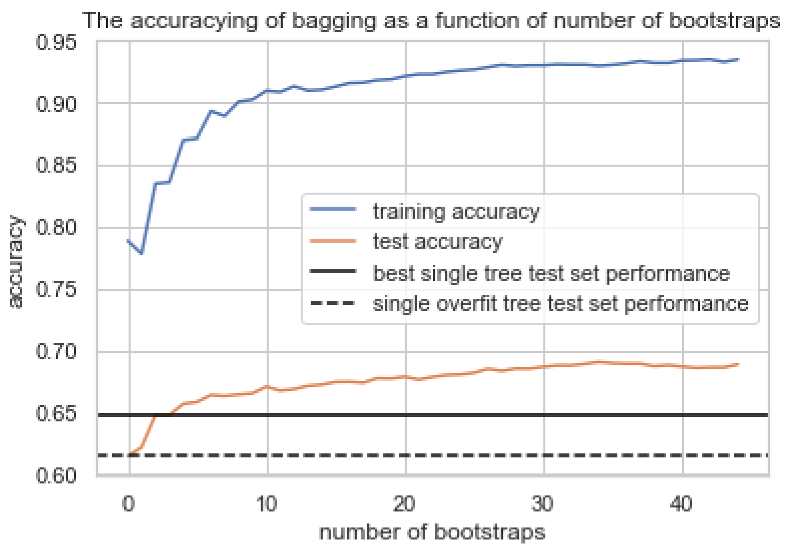
In [154]:
```
def running_predictions(prediction_dataset, targets):
    """A function to predict examples' class via the majority among trees

    Inputs:
      prediction_dataset - a (n_examples by n_sub_models) dataset, where e
          for example i
      targets - the true class labels

    Returns:
      a vector where vec[i] is the model's accuracy when using just the fi
    """

    n_trees = prediction_dataset.shape[1]

    # find the running percentage of models voting 1 as more models are co
    running_percent_1s = np.cumsum(prediction_dataset, axis=1)/np.arange(1

    # predict 1 when the running average is above 0.5
    running_conclusions = running_percent_1s > 0.5

    # check whether the running predictions match the targets
    running_correctnesss = running_conclusions == targets.reshape(-1,1)

    return np.mean(running_correctnesss, axis=0)
    # returns a 1-d series of the accuracy of using the first n trees to p
```

In [158]:
```
# your code here
training_acc_by_boot_num = running_predictions(train_bag_df.values, data_t
test_acc_by_boot_num = running_predictions(test_bag_df.values, data_test['
```

In [202]:
```
# your code here
pred = (DecisionTreeClassifier(max_depth = overfit_depth)
            .fit(data_train.drop('class', axis = 1), data_train['class'])
            .predict(data_test.drop('class', axis = 1))
           )
test_acc_overfit = accuracy_score(data_test['class'], pred)
```

In [203]:
```
pred_train = (DecisionTreeClassifier(max_depth = overfit_depth)
            .fit(data_train.drop('class', axis = 1), data_train['class'])
            .predict(data_train.drop('class', axis = 1))
           )
train_acc_overfit = accuracy_score(data_train['class'], pred_train)
```

In [164]:
```
plt.plot(training_acc_by_boot_num, label = "training accuracy")
plt.plot(test_acc_by_boot_num, label = 'test accuracy')
plt.axhline(test_acc, color = 'black', label = 'best single tree test set
plt.axhline(test_acc_overfit, color = 'black', linestyle = '--',label = 's
plt.xlabel('number of bootstraps')
plt.ylabel('accuracy')
plt.title('The accuracying of bagging as a function of number of bootstrap
plt.legend()
plt.show()
```



2.5 Referring to your graph from 2.4, compare the performance of bagging against the baseline of a single depth-10 tree. Explain the differences you see.

your answer here
The single depth 10 tree performs very similarly to bagging with only one bootstrapped dataset. This is because a single bootstrapped dataset should be statistically similar to the original dataset resulting in a similar model.

As we increase the number of bootstrapped datasets, the bagged model performs much better because averaging the models reduces the variance and, consequently, overfits less.

2.6 Bagging and limiting tree depth both affect how much the model overfits. Compare and contrast these two approaches. Your answer should refer to your graph in 2.4 and may duplicate something you said in your answer to 1.5.

your answer here
Bagging allows one to limit the variance of your model while still allowing for flexibility in predictions. Limiting the tree depth reduces variance, but also severely reduces how expressive our model can be. We can see that the added expressiveness of bagging leads to better performance.

2.7: In what ways might our bagging classifier be overfitting the data? In what ways might it be underfitting?

your answer here
The bagging classifier might overfit the data because it still uses all of the available features for each tree. This leads to the trees being correlated by often making similar splits to our feature space. If each individual tree is overfitting and the trees are correlated, it means the bagged model will also be overfitting.

The bagging classifier might be underfitting if each individual tree is not expressive enought to capture the true variation in the data.

Question 3: Random Forests [15 pts]

Random Forests are closely related to the bagging model we built by hand in question 2. In this question we compare our by-hand results with the results of using `RandomForestClassifier` directly.

3.1 Fit a `RandomForestClassifier` to the original `X_train` data using the same tree depth and number of trees that you used in Question 2.2. Evaluate its accuracy on the test set.

3.2 For each of the decision trees you fit in the bagging process, how many times is each feature used at the top node? How about for each tree in the random forest you just fit? What about the process of training the Random Forest causes this difference? What implication does this observation have on the accuracy of bagging vs Random Forest?

Hint: A decision tree's top feature is stored in `model.tree_.feature[0]`. A random forest object stores its decision trees in its `.estimators_` attribute.

3.3: Make a table of the training and test accuracy for the following models:

- Single tree with best depth chosen by cross-validation (from Question 1)
- A single overfit tree trained on all data (from Question 2, using the depth you chose there)
- Bagging 45 such trees (from Question 2)
- A Random Forest of 45 such trees (from Question 3.1)

(This problem should not require fitting any new models, though you may need to go back and store the accuracies from models you fit previously.)

What is the relative performance of each model on the training set? On the test set? Comment on how these relationships make sense (or don't make sense) in light of how each model treats the bias-variance trade-off.

Answers:

3.1 Fit a `RandomForestClassifier` to the original `X_train` data using the same tree depth and number of trees that you used in Question 2.2. Evaluate its accuracy on the test set.

In [183]:
```
# your code here
model_rf = RandomForestClassifier(max_depth=overfit_depth, n_estimators =
```

In [200]:
```
train_acc_rf = accuracy_score(data_train['class'], model_rf.predict(data_t
train_acc_rf
```

Out[200]:
```
0.9294
```

In [201]:
```
pred_rf = model_rf.predict(data_test.drop('class', axis = 1))
test_acc_rf = accuracy_score(data_test['class'], pred_rf)
test_acc_rf
```

Out[201]:
```
0.6952
```

3.2
For each of the decision trees you fit in the bagging process, how many times is each feature used at the top node?

Hint: A decision tree's top feature is stored in `model.tree_.feature[0]`. A random forest object stores its decision trees in its `.estimators_` attribute.

In [196]:
```
first_feature = [m.tree_.feature[0] for m in models]
print(np.unique(first_feature, return_counts= True))
```

```
(array([25]), array([45]))
```

All of the models use feature 25 for the top node.

How about for each tree in the random forest you just fit?

```
In [197]: first_feature_rf = [m.tree_.feature[0] for m in model_rf.estimators_]
          unique, counts = np.unique(first_feature_rf, return_counts= True)
          dict(zip(unique, counts))
```

```
Out[197]: {0: 4,
           2: 1,
           3: 5,
           5: 2,
           9: 1,
           14: 1,
           17: 1,
           18: 1,
           22: 6,
           24: 5,
           25: 11,
           26: 4,
           27: 3}
```

What about the process of training the Random Forest causes this difference?

Since the random forest uses a selection of features at each split, it leads to more variation in the first split of the tree.

What implication does this observation have on the accuracy of bagging vs Random Forest?

The trees in a random forest model are less correlated, which means that they are less likely to overfit.

3.3

Fill in the following table (ideally in code, but ok to fill in this Markdown cell).

classifier	training accuracy	test accuracy
single tree with best depth chosen by CV	|	
single depth-X tree	|	
bagging 45 depth-X trees	|	
Random Forest of 45 depth-X trees	|	

In [312]:
```
index = ['single tree with best depth chosen by CV',
         'single depth-10 tree',
         'bagging 45 depth-10 trees',
         'Random Forest of 45 depth-10 trees']
results_df = pd.DataFrame({'training accuracy':[train_acc, train_acc_overf
                           'test accuracy': [test_acc, test_acc_overfit, t
results_df.index = index
results_df
```

Out[312]:

	training accuracy	test accuracy
single tree with best depth chosen by CV	0.6812	0.6480
single depth-10 tree	0.8576	0.6208
bagging 45 depth-10 trees	0.9346	0.6886
Random Forest of 45 depth-10 trees	0.9294	0.6952

Question 4: Boosting [15 pts]

In this question we explore a different kind of ensemble method, boosting, where each new model is trained on a dataset weighted towards observations that the current set of models predicts incorrectly.

We'll focus on the AdaBoost flavor of boosting and examine what happens to the ensemble model's accuracy as the algorithm adds more predictors to the ensemble.

4.1 We'll motivate AdaBoost by noticing patterns in the errors that a single classifier makes. Fit `tree1`, a decision tree with depth 3, to the training data. For each predictor, make a plot that compares two distributions: the values of that predictor for examples that `tree1` classifies correctly, and the values of that predictor for examples that `tree1` classifies incorrectly. Do you notice any predictors for which the distributions are clearly different?

4.2 The following code attempts to implement a simplified version of boosting using just two classifiers (described below). However, it has both stylistic and functionality flaws. First, imagine that you are a grader for a Data Science class; write a comment for the student who submitted this code. Then, imagine that you're the TF writing the solutions; make an excellent example implementation. Finally, use your corrected code to compare the performance of `tree1` and the boosted algorithm on both the training and test set.

4.3 Now let's use the sklearn implementation of AdaBoost: Use `AdaBoostClassifier` to fit another ensemble to `X_train`. Use a decision tree of depth 3 as the base learner and a learning rate 0.05, and run the boosting for 800 iterations. Make a plot of the effect of the number of estimators/iterations on the model's train and test accuracy.

Hint: The `staged_score` method provides the accuracy numbers you'll need. You'll need to use `list()` to convert the "generator" it returns into an ordinary list.

4.4 Repeat the plot above for a base learner with depth of (1, 2, 3, 4). What trends do you see in the training and test accuracy?

(It's okay if your code re-fits the depth-3 classifier instead of reusing the results from the previous problem.)

4.5 Based on the plot you just made, what combination of base learner depth and number of iterations seems optimal? Why? How does the performance of this model compare with the performance of the ensembles you considered above?

Answers

4.1

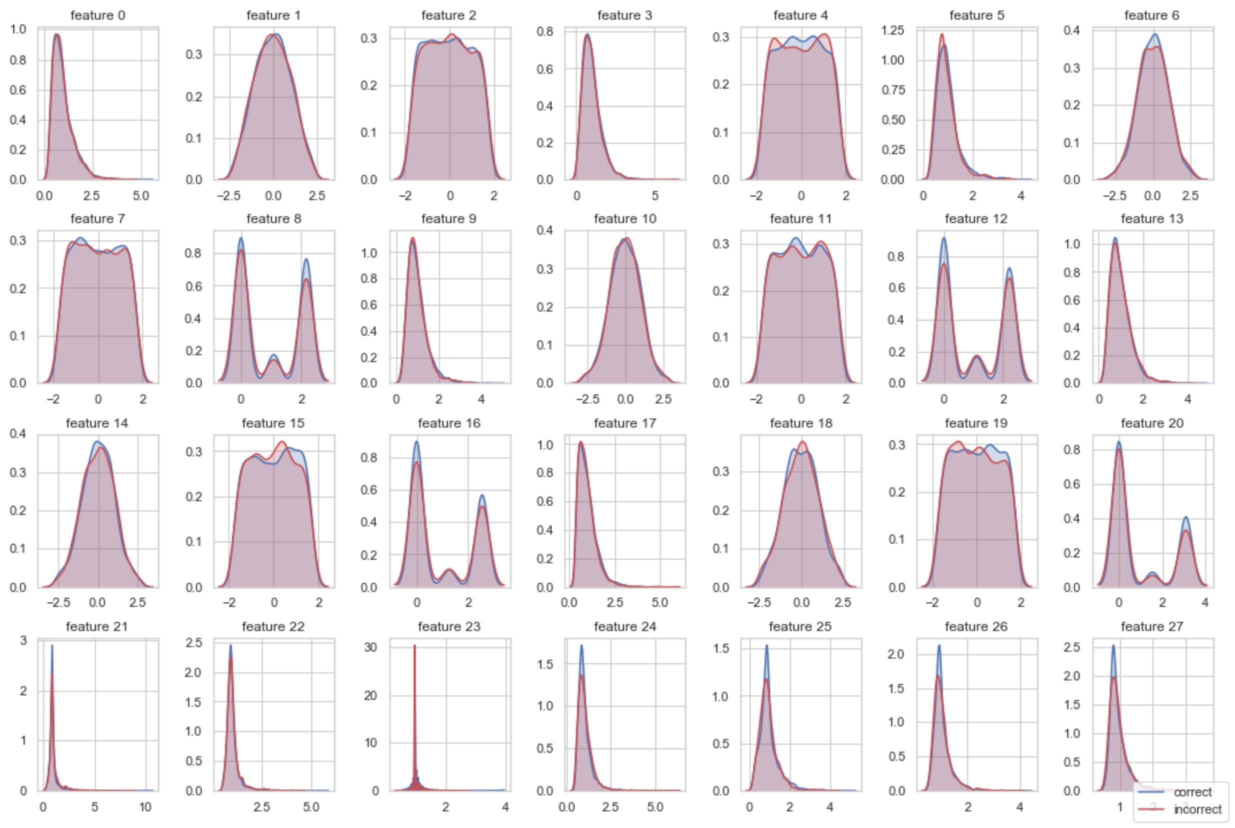
Hints:

- If you have `fig, axs = plt.subplots(...)`, then `axs.ravel()` gives a list of each plot in reading order.
- `sns.kdeplot` (https://seaborn.pydata.org/generated/seaborn.kdeplot.html) takes `ax` and `label` parameters.

In [228]:
```
# your code here
tree1 = DecisionTreeClassifier(max_depth=3).fit(data_train.drop('class', a
data_train['pred_correct'] = data_train['class'] == tree1.predict(data_tra
```

In [249]:
```
# your code here
fig, axes = plt.subplots(4,7, figsize = (15,10))
for idx, ax in enumerate(axes.ravel()):
    sns.kdeplot(data_train[data_train['pred_correct']][data_train.columns[
                shade=True,
                color="b",
                ax = ax,
                legend = False,
                label = 'correct'
               )
    sns.kdeplot(data_train[data_train['pred_correct'].apply(lambda x: not
                shade=True,
                color="r",
                ax = ax,
                legend = False,
                label = 'incorrect'
               )
    ax.set_title("feature {}".format(idx))
    handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc=4)
fig.tight_layout()
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/scipy/
stats/stats.py:1713: FutureWarning: Using a non-tuple sequence for multid
imensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr
[seq]`. In the future this will be interpreted as an array index, `arr[n
p.array(seq)]`, which will result either in an error or a different resul
t.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



your answer here

features 4, 15, 19, 24,25, 26, and 27 seem different.

4.2 The following code attempts to implement a simplified version of boosting using just two classifiers (described below). However, it has both stylistic and functionality flaws. First, imagine that you are a grader for a Data Science class; write a comment for the student who submitted this code. Then, imagine that you're the TF writing the solutions; make an excellent example implementation. Finally, use your corrected code to compare the performance of `tree1` and the boosted algorithm on both the training and test set.

The intended functionality is the following:

1. Fit `tree1` , a decision tree with max depth 3.
2. Construct an array of sample weights. Give a weight of 1 to samples that `tree1` classified correctly, and 2 to samples that `tree1` misclassified.
3. Fit `tree2` , another depth-3 decision tree, using those sample weights.
4. To predict, compute the probabilities that `tree1` and `tree2` each assign to the positive class. Take the average of those two probabilities as the prediction probability.

In [251]:
```
def boostmeup():
    tree = DecisionTreeClassifier(max_depth=3)
    tree1 = tree.fit(X_train, y_train)
    sample_weight = np.ones(len(X_train))
    q = 0
    for idx in range(len(X_train)):
      if tree1.predict([X_train[idx]]) != y_train[idx]:
         sample_weight[idx] = sample_weight[idx] * 2
         q = q + 1
    print("tree1 accuracy:", q / len(X_train))
    tree2 = tree.fit(X_train, y_train, sample_weight=sample_weight)

# Train
    q = 0
    for idx in range(len(X_train)):
        t1p = tree1.predict_proba([X_train[idx]])[0][1]
        t2p = tree2.predict_proba([X_train[idx]])[0][1]
        m = (t1p + t2p) / 2
        if m > .5:
            if y_train[idx] == True:
                q = q + 0
            else:
                q = q + 1
        else:
            if y_train[idx] == True:
                q = q + 1
            else:
                q = 0
    print("Boosted accuracy:", q / len(X_train))

# Test
    q = 0
    for idx in range(len(X_test)):
        t1p = tree1.predict_proba([X_test[idx]])[0][1]
        t2p = tree2.predict_proba([X_test[idx]])[0][1]
        m = (t1p + t2p) / 2
        if m > .5:
            if y_train[idx] == True:
                q = q + 0
            else:
                q = q + 1
        else:
            if y_train[idx] == True:
                q = q + 1
            else:
                q = 0
    print("Boosted accuracy:", q / len(X_test))

boostmeup()
```

```
tree1 accuracy: 0.3582
Boosted accuracy: 0.0008
Boosted accuracy: 0.002
```

Your answer here

Comments:

```
* It would be better if you enapsulate your code better. Make an obj
ect for your boosted model with a fit and predict method
* You should either vectorize your functions or use list comprehensi
ons - fewer "for" loops!
* The way you're counting correct values is wrong
* you're fitting the same tree model twice, not two different trees
```

In [279]:

```python
# your code here
class BoostedTree(object):
    def __init__(self, max_depth = 3):
        self.max_depth = max_depth

    def fit(self, x, y):
        self.tree1 = DecisionTreeClassifier(max_depth=self.max_depth).fit(
        pred = self.tree1.predict(x)
        sample_weight = np.array(pred != y).astype(int) + 1 # 1 if correct
        self.tree2 = DecisionTreeClassifier(max_depth=self.max_depth).fit(

    def predict(self, x):
        tree1_prob = self.tree1.predict(x)
        tree2_prob = self.tree2.predict(x)
        boosted_prob = np.array(tree1_prob + tree2_prob)/2
        return (boosted_prob >= .5).astype(int)
```

In [280]:

```python
bt = BoostedTree()
bt.fit(X_train, y_train)
```

In [297]:

```python
tree_acc_train = accuracy_score(bt.tree1.predict(X_train), y_train)
print("basic tree training accuracy:", tree_acc_train)

tree_acc_test = accuracy_score(bt.tree1.predict(X_test), y_test)
print("basic tree test accuracy:", tree_acc_test)

boost_acc_train = accuracy_score(bt.predict(X_train), y_train)
print("boosting training accuracy:", boost_acc_train)

boost_acc_test = accuracy_score(bt.predict(X_test), y_test)
print("boosting test accuracy:", boost_acc_test)
```

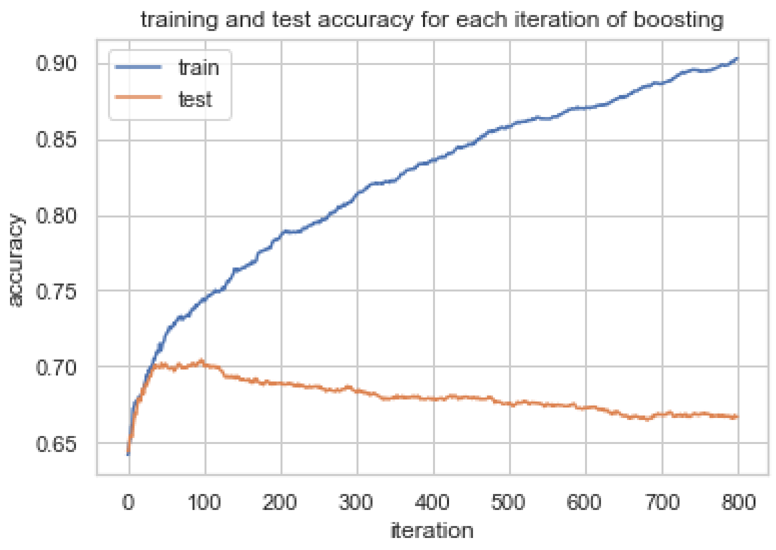
```
basic tree training accuracy: 0.6418
basic tree test accuracy: 0.6442
boosting training accuracy: 0.6264
boosting test accuracy: 0.6316
```

4.3 Now let's use the sklearn implementation of AdaBoost: Use AdaBoostClassifier to fit another ensemble to X_train. Use a decision tree of depth 3 as the base learner and a learning rate 0.05, and run the boosting for 800 iterations. Make a plot of the effect of the number of estimators/iterations on the model's train and test accuracy.

Hint: The staged_score method provides the accuracy numbers you'll need. You'll need to use list() to convert the "generator" it returns into an ordinary list.

In [306]:
```
# your code here
adaboost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_de
                              learning_rate=0.05,
                              n_estimators = 800
                             )
adaboost.fit(X_train, y_train)

train_scores = list(adaboost.staged_score(X_train, y_train))
test_scores = list(adaboost.staged_score(X_test, y_test))

plt.plot(np.arange(len(train_scores)), train_scores, label = 'train')
plt.plot(np.arange(len(test_scores)), test_scores, label = 'test')
plt.title('training and test accuracy for each iteration of boosting')
plt.xlabel('iteration')
plt.ylabel('accuracy')
plt.legend()
```

Out[306]:
```
<matplotlib.legend.Legend at 0x124b42748>
```

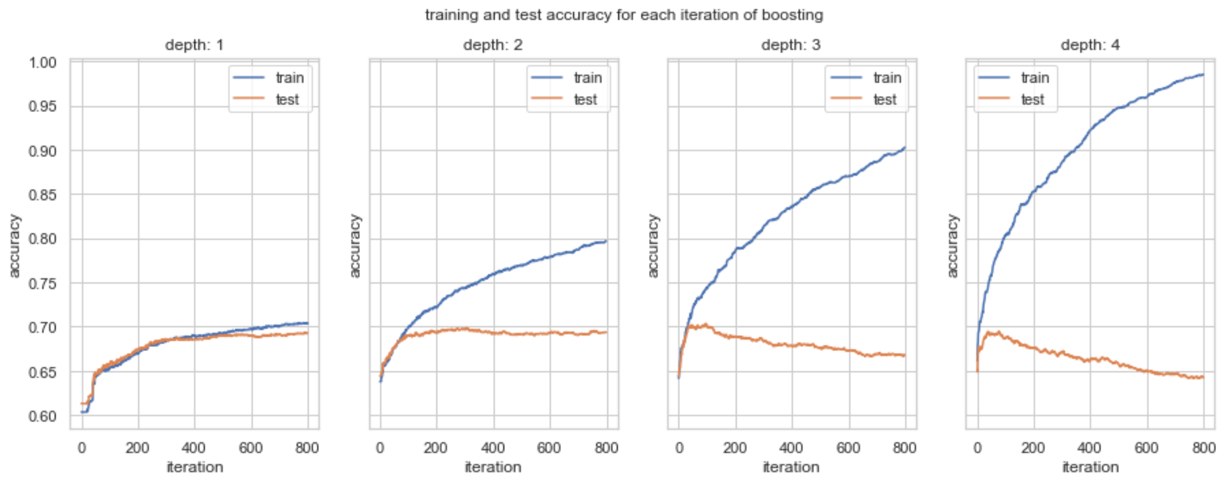


4.4 Repeat the plot above for a base learner with depth of (1, 2, 3, 4). What trends do you see in the training and test accuracy?

(It's okay if your code re-fits the depth-3 classifier instead of reusing the results from the previous problem.)

In [311]:
```
# your code here
depths = [1,2,3,4]
fig, axes = plt.subplots(1, 4, figsize = (15,5), sharex= True, sharey=True
for idx, ax in enumerate(axes):
    adaboost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(ma
                                  learning_rate=0.05,
                                  n_estimators = 800
                                 )
    adaboost.fit(X_train, y_train)

    train_scores = list(adaboost.staged_score(X_train, y_train))
    test_scores = list(adaboost.staged_score(X_test, y_test))

    ax.plot(np.arange(len(train_scores)),train_scores, label = 'train')
    ax.plot(np.arange(len(test_scores)),test_scores, label = 'test')
    ax.set_xlabel('iteration')
    ax.set_ylabel('accuracy')
    ax.set_title("depth: {}".format(depths[idx]))
    ax.legend()
fig.suptitle('training and test accuracy for each iteration of boosting')
fig.show()
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/matplo
tlib/figure.py:457: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



Your answer here

4.5

Based on the plot you just made, what combination of base learner depth and number of iterations seems optimal?

It looks like the highest test set accuracy is achieved with depth 2 and 300 iterations.

Why?

When you boost models, it's easy to overfit so it's important to optimize for the test set, rather than training set.

How does the performance of this model compare with the performance of the ensembles you considered above?

This model performs better than all ensembles we've trained thus far.

Your answer here

Question 5: Understanding [15 pts]

This question is an overall test of your knowledge of this homework's material. You may need to refer to lecture notes and other material outside this homework to answer these questions.

5.1 How do boosting and bagging relate: what is common to both, and what is unique to each?

5.2 Reflect on the overall performance of all of the different classifiers you have seen throughout this assignment. Which performed best? Why do you think that may have happened?

5.3 What is the impact of having too many trees in boosting and in bagging? In which instance is it worse to have too many trees?

5.4 Which technique, boosting or bagging, is better suited to parallelization, where you could have multiple computers working on a problem at the same time?

5.5 Which of these techniques can be extended to regression tasks? How?

Answers:

5.1 How do boosting and bagging relate: what is common to both, and what is unique to each?

Your answer here

They both take a weighted average of the predictions of multiple models. Bagging generates the ensemble by bootstrapping your data. Boosting generages the ensembles by fiting models to the residuals of the past models.

5.2 Reflect on the overall performance of all of the different classifiers you have seen throughout this assignment. Which performed best? Why do you think that may have happened?

The boosted model performed best. Thought this is not always the case, boosting often outperforms bagging. This is true because bagging tries to reduce the variance of complex classifiers, while boosting optimizes the performance direclty. This makes it easier find the exact point when test set performance is optimal.

5.3 What is the impact of having too many trees in boosting and in bagging? In which instance is it worse to have too many trees?

If you have too many trees, you risk overfitting your data. This is more of a problem in boosting than in bagging.

5.4 Which technique, boosting or bagging, is better suited to parallelization, where you could have multiple computers working on a problem at the same time?

Bagging can be parallelized because each model is fit independently, but boosting cannot be easily parallelized because the process is iterative.

5.5 Which of these techniques can be extended to regression tasks? How?

Both of these models can be exteneded to regression tasks.

Bagging can be extended to regression by using a regression model as the base classifer. You can then either average the predictions of the ensemble.

Boosting can also be extended to regression by using a regression model as the base classifier. After fitting a model and making a prediction, you can fit the next model to the residuals. After you've reached a desired number of iterations, you can weight the predictions of all the models.

Question 6: Explaining Complex Concepts Clearly [10 pts]

One of the core skills of a data scientist is to be able to explain complex concepts clearly. To practice this skill, you'll make a short presentation of one of the approaches we have recently studied.

Choose one of the following topics:

- Decision Trees
- Random Forests
- Bagging
- Boosting
- Simple Neural Nets (like the MLP we saw in Homework 6)
- (other topics are possible, but get staff approval first)

Make 3 slides explaining the concept.

- Focus on **clear explanations**, NOT aesthetic beauty. Photos of pen-and-paper sketches are fine if they're legible.
- For your audience, choose **future CS109A students**.
- You may take inspiration from anywhere, but explain in **your own words** and **make your own illustrations**.

Submit your slides as a PDF and the source format (`.pptx` , Google Slides, etc.)

NOTE: If you would be okay with us using your slides for future classes (with attribution, of course), please include a note to that effect. This will not affect your grade either way.