Homework 6 - CS348 Spring 2019

Description - In this assignment, you will run and analyze binary classification using ensembles of classification trees.

Getting Started - You should complete the assignment using your own installation of Python 3 and the packages numpy, pandas, matplotlib, and seaborn. Download the assignment from Moodle and unzip the file. This will create a directory with this file, 'HW06.ipynb', and a folder 'data' containing four csv files.

Deliverables - The assignment has a single deliverable: this jupyter notebook file saved as a pdf. Please answer all coding and writing questions in the body of this file. Once all of the answers are complete, download the file by navigating the following menus: File -> Download as -> PDF via LaTeX. Submit the downloaded pdf file on gradescope. Alternatively, you can save the file as a pdf via the following: File -> Print Preview -> Print as pdf.

Data Sets - In this assignment, you will use two datasets, one on handwritten digit classification loaded from the 'data' directory, and the other on breast cancer from the sci-kit learn library.

Academic Honesty Statement - Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is considered cheating. Posting your code to public repositories such as GitHub is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

This academic honesty statement does not restrict you from reading official documentation or using other web resources for understanding the syntax of python, related data science libraries, or properties of distributions.

In [1]:
```
# Do not import any other libraries other than those listed here.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as skl
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load_breast_cancer
```

Problem 1 - Random Forests

In this problem you'll use a Random Forest ensemble model to classify a subset of handwritten digit data with added white-noise.

```
In [2]: x_train = np.loadtxt('data/x_train.csv', delimiter=',')
        y_train = np.loadtxt('data/y_train.csv', delimiter=',')
        x_val = np.loadtxt('data/x_val.csv', delimiter=',')
        y_val = np.loadtxt('data/y_val.csv', delimiter=',')
```

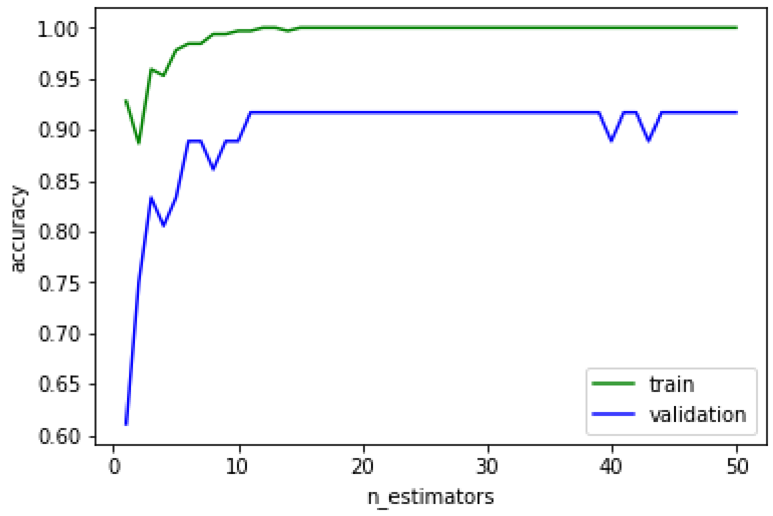
Part 1 (15 points)

For all integer values of `n_estimators` between 1 and 50, using sci-kit learn's `RandomForestClassifier` class with `random_state=0` , train a random forest model between features `x_train` and targets `y_train` . For each model compute the classification accuracy on the training data and on the validation data, `x_val` and `y_val` . Using color to distinguish between training and validation accuracy, plot the results in a 2-d plot with `n_estimators` on the horizontal axis and `accuracy` on the vertical axis.

Which value of `n_estimators` would you select? What are the advantages and disadvantages of using the random forests classifier over the single classification tree? Be specific.

In [7]:
```
# Part 1 Solution

# --- write code here ---
estimators = []
train_acc = []
val_acc = []
for index in range(1, 51):
    estimators.append(index)
    clf = RandomForestClassifier(n_estimators=index, random_state=0)
    clf = clf.fit(x_train, y_train)
    train_acc.append(clf.score(x_train, y_train))
    val_acc.append(clf.score(x_val, y_val))
plt.figure()
plt.plot(estimators, train_acc, color="green", label="train")
plt.plot(estimators, val_acc, color="blue", label="validation")
plt.xlabel("n_estimators")
plt.ylabel("accuracy")
plt.legend(loc="lower right")
plt.show()
```



Part 1 Written Response

I would select n_estimators in range from 44 to 50, since the accuracy is consistent in that range of n_estimators.

Advantages:

1. The process of averaging or combining the results of different decision trees helps to overcome the problem of overfitting.
2. Since it has less varaince than a single decision tree, it will work correctly for a large range of data items than single decision trees.

Disadvantages:

1. Random forests classifier cannot handle both numerical and categorical data.
2. It's time-consuming to construct than decision trees.
3. It's sensitive to noise data.

Part 2 (10 points)

As discussed in class, ensemble methods make predictions by computing a weighted average of the individual predictions from a set of weak classifiers. However, if all of the weak classifiers are identical, the ensemble is equivalent to using a single classifier. How does the random forest algorithm introduce variability into the decision rules of each tree?

Note: Your answer should include two sources of variability.

Part 2 Written Response

By averaging several trees, there is a significantly lower risk of overfitting.

By combining multiple trees. random forest algorithm reduce the chance of stumbling across a classifier that does not perform well because of the relationship between the train and test data.

Part 3 (20 points)

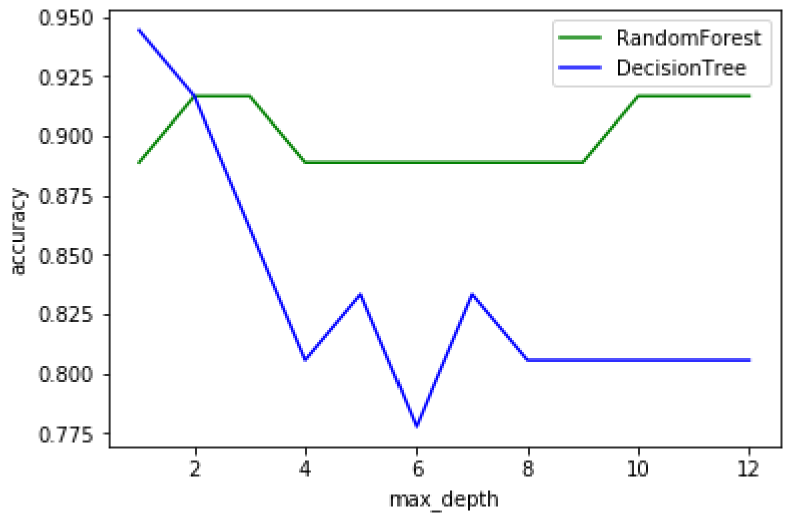
For each value of `max_depth` between 1 and 12, train a random forest and decision tree model using sci-kit learn's `RandomForestClassifier` and `DecisionTreeClassifier` classes using the training data `x_train` and `y_train` with `random_state=0` for both models and `n_estimators=100` for the `RandomForestClassifier`. For each of these models, evaluate the classification accuracy on the validation data `x_val` and `y_val`. Using color to distinguish between the random forest and the decision tree models, plot the results in a 2-d plot with `max_depth` on the horizontal axis and validation accuracy on the vertical axis.

How does the relationship between validation accuracy and `max_depth` differ between the two models? What causes this difference?

Hint: Consider how bagging and the `max_depth` parameter influence the classifier's accuracy in terms of its bias and variance.

In [11]:
```
# Part 3 Solution

# --- write code here ---
depth = []
rf_acc = []
dt_acc = []
for dep in range(1, 13):
    depth.append(dep)
    clf1 = RandomForestClassifier(max_depth=dep, random_state=0, n_estim
ators=100)
    clf2 = DecisionTreeClassifier(max_depth=dep, random_state=0)
    clf1 = clf1.fit(x_train, y_train)
    clf2 = clf2.fit(x_train, y_train)
    rf_acc.append(clf1.score(x_val, y_val))
    dt_acc.append(clf2.score(x_val, y_val))
plt.figure()
plt.plot(depth, rf_acc, label="RandomForest", color="green")
plt.plot(depth, dt_acc, label="DecisionTree", color="blue")
plt.xlabel("max_depth")
plt.ylabel("accuracy")
plt.legend(loc="best")
plt.show()
```



Part 3 Written Response

For RandomForest classifier, the validation accuracy doesn't have a relationship with max_depth, since the plot is a relatively flat horizontal line.

For DecisionTree, the validation accuracy doesn have a relationship with max_depth, as the max_depth increases, the validation accuracy decreases.

By using bagging, the classifier randomly resample to introduce enough diversity to decrease the variance and improved accuracy. Thus RandomForest classifier is not affected by max_depth. However, for DecisionTree classifier, as the max_depth grows, there will be more detailed split rules appear and introduces overfitting problem. Thus, as the max_depth grows, th accuracy of DecisionTree decreases.

Problem 2 - Boosted Classification Trees

In [2]:
```
data = load_breast_cancer()
x = data['data']
y = data['target']
y[y==0] = -1

indeces = np.array([468, 342,  40, 303, 459, 225, 523, 250, 438, 365, 53
8, 330,  35,
       120,  87, 338, 250,  26,  36, 407, 531, 330, 226,  18, 473, 121,
       210, 234, 431, 536, 202, 319, 509, 235, 400, 107, 266, 562,  50,
       209, 507,  24, 280, 178,  90, 171,  44, 208, 536,  74,  41, 162,
       469, 352, 535, 189, 262, 133, 436, 562, 278, 434, 510, 346, 559,
        16, 112,  82, 554, 549, 537, 347, 192, 186,  84, 439, 172, 496,
       522,  25, 388, 474, 410, 529, 478, 225, 298, 323, 170, 308, 161,
       411, 184, 341, 233, 244, 142,  57, 310, 187, 177, 494, 241, 522,
       250, 565, 328, 468, 516, 373, 434, 180, 537, 549, 546,  51, 509,
       390,   7, 416,  28, 544, 252, 115,  55, 388,   4, 275, 144, 237,
       123, 419, 232, 277, 102,  44, 225, 493, 364, 116, 232, 559, 426,
       254, 534, 548, 405, 374, 435, 122, 288, 212, 441, 545, 279, 130,
       366, 447, 260, 568, 328,  96, 524, 368, 269, 418, 280, 137,   0,
        59, 208,  22, 468, 433, 479, 261, 355, 543, 390, 537, 404, 455,
       147, 524, 554, 455, 202, 356, 278, 440, 280,  52, 552, 501,   0,
       177, 505, 107,  58,  51, 504,  47, 102, 267,  10, 247,  86, 215,
       423,  64, 251, 183, 439, 438, 315, 527, 287, 487, 538, 337, 301,
        21,  23,  89, 272, 400, 402, 435, 438, 263, 357, 426, 188, 427,
       324, 318, 388, 357, 108, 536, 393, 467,   6, 283, 205,  32,  58,
       516, 170,  46,  69, 522, 351,   9, 480, 341, 428, 439, 299, 206,
       347, 421, 113, 400, 129,  40, 371, 243, 327, 382,  73, 450, 472,
        95,  92, 284, 502, 556, 414, 159, 360,  96, 167,  45, 455, 435,
       118,  78, 359, 256,  80, 250, 206,  25, 259, 528, 145, 174, 388,
       355, 119, 512, 376, 365, 230, 389,  84, 376, 174, 178, 486, 278,
       138, 432, 511, 374,  24, 181, 140, 207,  50,  52, 566, 186, 435,
       551, 240, 406, 233, 234, 148, 123, 144, 522,  32, 547, 223, 426,
        43, 453, 343, 443,  45, 413, 315, 265,  81, 267, 148, 114, 379,
       365, 456, 204,  29, 198, 270,  20,  71, 441, 475, 187, 187, 121,
       210, 255, 513,  49, 203, 131, 211, 436, 479, 568, 297, 172, 444,
       298,   7, 345, 539, 407, 370, 235,   9, 307, 140, 433, 423,  63,
       556, 129, 363, 137,  97,  26, 126, 311, 294, 349, 253, 326, 323,
       334, 245, 128, 175, 568, 182, 245, 273, 519, 491, 481, 170, 438,
       353, 531, 252,  73, 437, 408, 209, 246, 248,  12, 525, 423, 476,
       543,  16, 414, 491, 404, 417, 122, 135, 354, 207, 323,  93, 261,
       260,  77, 154, 276, 257, 546,  67, 309,  96, 227, 393, 461, 323,
       418, 333, 109, 108,  57, 222, 275, 413, 110, 549, 290, 524, 470,
       298,   0, 317, 455, 492, 191, 132, 277, 525, 282, 525, 486, 320,
       324, 377, 178, 494, 382, 477, 180, 392, 254, 454, 113, 168, 176,
       219, 234, 194, 356, 145, 494, 189, 253, 208, 417, 117, 253, 521,
        46, 389, 466, 123, 153, 495, 241, 398, 522, 245, 499, 563, 308,
        21,   5, 119,   7, 488, 178, 557, 368, 244, 379, 223,  45, 500,
        90, 291, 428, 418, 183, 394, 410, 521, 458, 496, 441, 190,  61,
       558, 133,  80,  43, 272, 190, 334, 158,  46, 532,   4,  46, 136,
       530,  93,  64, 559, 506,  65, 177, 241, 118, 249])

split_point = int(x.shape[0]/2)

x_train = x[indeces][:split_point]
y_train = y[indeces][:split_point]
```

```
x_val = x[indeces][split_point:]
y_val = y[indeces][split_point:]
```

Part 1 (45 points)

Using sci-kit learn's `DecisionTreeClassifier` class as a "weak learner", implement the AdaBoost algorithm as described in the "Example Algorithm (Discrete AdaBoost)" section of https://en.wikipedia.org/wiki/AdaBoost (https://en.wikipedia.org/wiki/AdaBoost). Your implementation should modify the `fit` and `predict` methods of the `AdaBoostTrees` class below. Your implementation can use any built-in methods from the `DecisionTreeClassifier` class in sci-kit learn, but cannot use any of the classes or methods in `sklearn.ensembles`.

Once you have implemented the `fit` and `predict` methods for the `AdaBoostTrees` class, compute the classification accuracy on the validation data using the default values of `n_estimators=10` and `max_depth=1`.

Note 1: You should not modify the `__init__` method for the `AdaBoostTrees` implementation, but you will need to use the 4 initialized attributes in the `fit` and `predict` methods.

Note 2: The class labels in `y_train` have been coded as `[-1, 1]` to be consistent with the description in the wikipedia article. This differs from our standard convention of coding class labels as `[0, 1]`.

Hint: You may want to inspect the documentation of the initialization and method arguments for the `DecisionTreeClassifier` object. This documentation can be found at https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html).

In [16]:
```
# Part 1 Solution

class AdaBoostTrees(object):

#   Note: You will need to remove the "pass" statements

    def __init__(self, n_estimators=10, max_depth=1):

#       self.n_estimators : The number of decision tree classifiers used
in the ensemble.
#       self.max_depth    : The max_depth setting for each of the Decisi
onTree objects.
#       self.trees        : A list containing all of the DecisionTree ob
jects.
#       self.tree_weights : A list containing the weight of each of the
 trained DecisionTree objects.

        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.trees = []
        self.tree_weights = []

    def fit(self, x_train, y_train):

#       This method takes as input two numpy arrays, x_train and y_trai
n,
#       and modifies the AdaBoostTrees object in-place. This method impl
ements
#       the AdaBoost algorithm using sci-kit learn's DecisionTreeClassif
ier
#       as the "weak learner".

#       --- Insert code here ---
        w = np.ones(len(x_train)) / len(x_train)
        for i in range(self.n_estimators):
            clf = DecisionTreeClassifier(max_depth=self.max_depth)
            clf = clf.fit(x_train, y_train, sample_weight=w)
            pred = clf.predict(x_train)

            eps = w.dot(pred!=y_train)
            alpha = (np.log(1-eps) - np.log(eps)) / 2

            w = w*np.exp(-alpha * y_train * pred)
            w = w / w.sum()

            self.trees.append(clf)
            self.tree_weights.append(alpha)

    def predict(self, x):

#       This method takes as input a numpy array of size (N, D) and
#       returns a numpy array of predictions of size (N). Calling this
#       method requires that fit() has already been run on the training
 data.

#       --- Insert code here ---
```

```
        y_pred = np.zeros(len(x))
        for (clf, alpha) in zip(self.trees, self.tree_weights):
            y_pred = y_pred + alpha * clf.predict(x)
        y_pred = np.sign(y_pred)

        return y_pred
```

In [21]:
```
# Part 1 Solution

# --- Insert code here ---
model = AdaBoostTrees(n_estimators=10, max_depth=1);
model.fit(x_val, y_val)
pred = model.predict(x_val)

corr = 0
for i in range(len(pred)):
    if pred[i]==y_val[i]:
        corr += 1
print(corr/len(pred))
```

```
0.9894736842105263
```

Part 2 (10 points)

Both the `RandomForestClassifier` and our `AdaBoostTrees` models implement an ensemble of decision tree classifiers. How do these methods differ, and what are the advantages and disadvantages of each. Describe a situation where you would prefer a Random Forest classification method over the Adaptive Boosting approach you implemented in part 1.

Part 2 Written Response

Random forests classifier method inherently discriminate against overfitting and removes stochastic elements by balancing out the entroy, however, adaboost method doesn't.

RandomForestClassifierAdvantages:

1. The process of averaging or combining the results of different decision trees helps to overcome the problem of overfitting.
2. Since it has less varaince than a single decision tree, it will work correctly for a large range of data items than single decision trees.

RandomForestClassifierDisadvantages:

1. Random forests classifier cannot handle both numerical and categorical data.
2. It's time-consuming to construct than decision trees.
3. It's sensitive to noise data.

AdaBoostTreeAdvantages:

1. Very simple to implement.
2. Feature selection on very large sets of features.
3. Does feature selection resulting in relatively simple classifier.
4. Fairly good generalization. AdaBoostTreeDisadvantages:
5. Suboptimal solution.
6. Sensitive to noisy data and outliers.

```
In [ ]:
```