CS289A_HW05_prob4

March 31, 2017

1 HW05 - Problem 4

Performance evaluation. For each of the 3 datasets, train a decision tree and random forest and report your training and validation accuracies. You should be reporting 12 numbers (3 datasets × 2 classifiers × 2 data splits).

Program overhead:

```
In [2]: %load_ext autoreload
```

```
In [3]: %autoreload 2
```

```
In [4]: import numpy as np
        import decisiontree as dt
        import randomforest as rf
        import HW05_utils as ut
        from matplotlib import pyplot as plt
```

Set the base directory for this homework

```
In [5]: BASE_DIR = "/Users/mitch/Documents/Cal/2_2017_Spring/COMPSCI 289A - Intro t
```

Establish a size for the validation set as a fraction of the total training set

```
In [6]: valfrac = 0.1
```

1.1 ## SPAM

Calculate decision tree and random forest training/validation accuracies for the spam dataset.

Begin by importing data, shuffling, and separating into training and validation sets.

```
In [39]: # Import
         SPAM_PATH = "Data/hw5_spam_dist/spam_data.mat"

         spam_data = ut.load_data(SPAM_PATH,BASE_DIR,'training_data')
         spam_labels = ut.load_data(SPAM_PATH,BASE_DIR,'training_labels').T
         spam_test = ut.load_data(SPAM_PATH,BASE_DIR,'test_data')
```

```
In [40]: # Shuffle
         spamdata, spamlabels = ut.shuffle_data(spam_data,spam_labels)
```

```
In [41]: # Separate
         spamtraindata,spamvaldata = ut.val_partition(spamdata,valfrac)
         spamtrainlabels,spamvallabels = ut.val_partition(spamlabels,valfrac)
```

1.1.1 Spam Decision Tree

Create and train a decision tree classifier using the spam data and labels.

```
In [13]: spam_DTclassifier = dt.DecisionTree(treedepth=28)
```

```
In [14]: spam_DTclassifier.train(spamtraindata,spamtrainlabels)
```

Calculate the validation accuracy using the trained decision tree

```
In [15]: spamDTpredictions = spam_DTclassifier.predict(spamvaldata)
```

```
In [16]: spamDTvalAcc = ut.val_accuracy(spamDTpredictions,spamvallabels)
         print('Validation Accuracy = %.3f%%'%(100*spamDTvalAcc))
```

```
Validation Accuracy = 78.354%
```

Use this procedure to determine the optimal value for the tree depth hyperparameter.

```
In [11]: def get_depth_acc(testtype,traindata,trainlabels,valdata,vallabels,treedep

             # Pick the type of classifier
             if testtype == 'DT':
                 classifier = dt.DecisionTree(treedepth)
             elif testtype == 'RF':
                 classifier = rf.RandomForest(treedepth,ntrees,mfeatures)

             classifier.train(traindata,trainlabels)
             predictions = classifier.predict(valdata)
             valAcc = ut.val_accuracy(predictions,vallabels)

             return valAcc
```

```
In [12]: def testdepths(testtype,traindata,trainlabels,valdata,vallabels,maxdepth,

             depthAccs = np.empty((int(np.ceil(maxdepth/depthstep)),2))
             for depth in range(1,maxdepth+1,depthstep):
                 if depth%5==0:
                     print('Tested up to depth '+str(depth)+'...')
                 Acc = get_depth_acc(testtype,traindata,trainlabels,valdata,vallabe
                 depthAccs[int(np.ceil(depth/depthstep))-1] = np.array([depth,Acc])

             return depthAccs
```

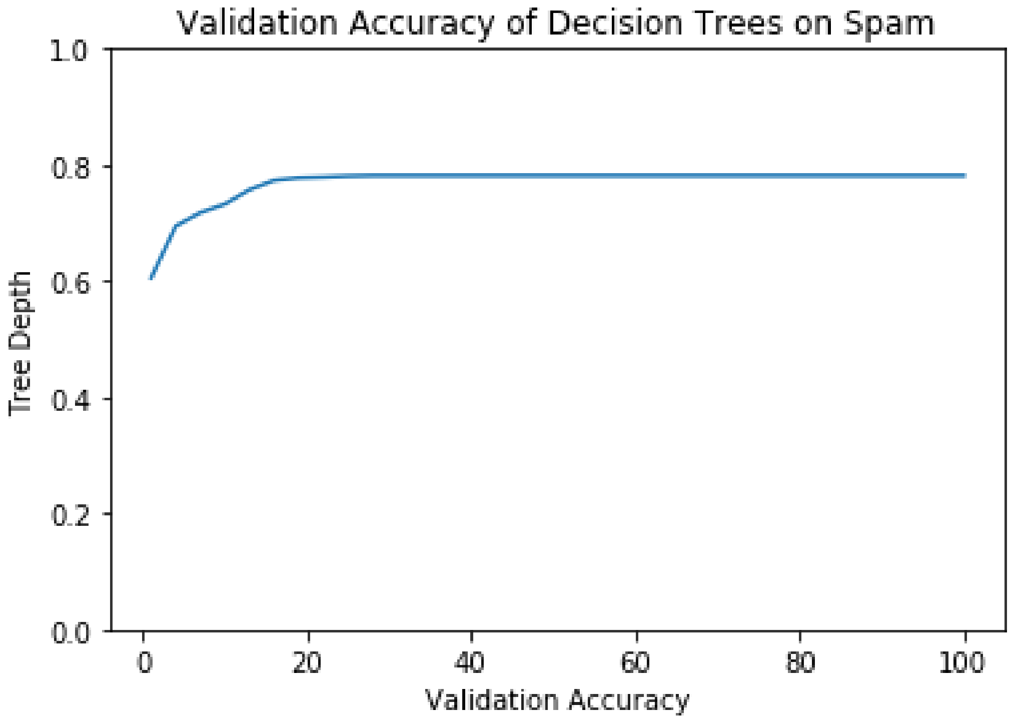
```
In [13]: def plotdepthAccs(depthAccs,title):

             # Plot the Accuracy as a function of tree depth
             fig = plt.figure()
             plt.plot(depthAccs[:,0],depthAccs[:,1])
```

```
            plt.title(title)
            plt.xlabel('Validation Accuracy')
            plt.ylabel('Tree Depth')
            plt.ylim(0,1)
            plt.show()
            am = np.argmax(depthAccs[:,1])
            print('Max accuracy of %.2f%% for depth of %i' %(100*depthAccs[am,1],c
```

```
In [29]: maxdepth=100
         depthstep=3
```

```
In [30]: spamDT_Accs=testdepths('DT',spamtraindata,spamtrainlabels,spamvaldata,spam
```

```
In [31]: plotdepthAccs(spamDT_Accs,'Validation Accuracy of Decision Trees on Spam')
```



```
Max accuracy of 78.14% for depth of 28
```

Make decision tree predictions for the best depth according to test data, and save to a csv file for upload to Kaggle (0-indexing).

```
In [20]: def train_optimal(classifiertype,Accs,data,labels,outfilename):
    # Create (or add to) an output file for the best trained algorithm
    outfile = open(outfilename,'w')
    outfile.write(outfilename+'\n')
    outfile.write((50*'=')+'\n\n')

    # Retrain the classifier on the best depth
    bestdepth = int(Accs[np.argmax(Accs[:,1]),0])
    outfile.write('Optimal Depth: '+str(bestdepth)+'\n')
    if classifiertype == 'DT':
        classifier = dt.DecisionTree(treedepth=bestdepth)
    elif classifiertype == 'RF':
        classifier = rf.RandomForest(treedepth=bestdepth,ntrees=100)
    classifier.train(data,labels)

    outfile.write('\n'+50*'-'+'\n\n\n')
    outfile.close()

    return classifier


def write_Accs(classifier,datasets,datasetslabels,datasetnames,outfilename

    outfile = open(outfilename,'a')
    outfile.write('ACCURACIES\n'+50*'='+'\n\n')
    for ds_i in range(len(datasets)):
        dataset = datasets[ds_i]
        datasetlabels = datasetslabels[ds_i]
        datasetname = datasetnames[ds_i]

        # Write accuracies of an algorithm on dataset(s) to a file
        predictions = classifier.predict(dataset)
        valAcc = ut.val_accuracy(predictions,datasetlabels)
        outfile.write(datasetname+' Accuracy:\t  %0.4f\n' %(100*valAcc))

    outfile.write('\n'+50*'-'+'\n\n\n')
    outfile.close()


def make_kaggle(classifier,testset,kagglecsvfilename,indexing=0):
    # Use this optimal classifier on the test data
    predictions = classifier.predict(testset)
    if indexing == 0:
        ids = np.arange(len(predictions))
    elif indexing == 1:
        ids = np.arange(1,len(predictions)+1)
    predictions_csv = np.concatenate(([ids],[predictions]),axis=0).T
    np.savetxt(kagglecsvfilename,predictions_csv,fmt='%i',delimiter=',',he
In [ ]: spamDToutfilename ='../spamDT_accuracies.txt'
        bestspamDT = train_optimal('DT',np.array([[28,1],[0,0]]),spamdata,spamlabel
        #bestspamDT = train_optimal(spamDT_Accs,spamdata,spamlabels,spamoutfilename
        write_Accs(bestspamDT,
                   [spamtraindata,spamvaldata],
                   [spamtrainlabels,spamvallabels],
                   ['Training','Validation'],
                   spamDToutfilename,
                  )
```

```
In [42]: bestspamDT = train_optimal('DT',np.array([[28,1],[0,0]]),spamdata,spamlabe
         make_kaggle(bestspamDT,spam_test,BASE_DIR+'spam_DT_testpredictions.csv',i
```

1.1.2 Spam Random Forest

Create and train a random forest classifier using the spam data and labels.

```
In [14]: spam_RFclassifier = rf.RandomForest(treedepth=23,ntrees=100)
```

```
In [15]: spam_RFclassifier.train(spamtraindata,spamtrainlabels)
```

Calculate the validation accuracy using the trained decision tree.

```
In [16]: spampredictions = spam_RFclassifier.predict(spamvaldata)
```

```
In [17]: spamvalAcc = ut.val_accuracy(spampredictions,spamvallabels)
         print('Validation Accuracy = %.3f%%'%(100*spamvalAcc))
```

```
Validation Accuracy = 75.992%
```

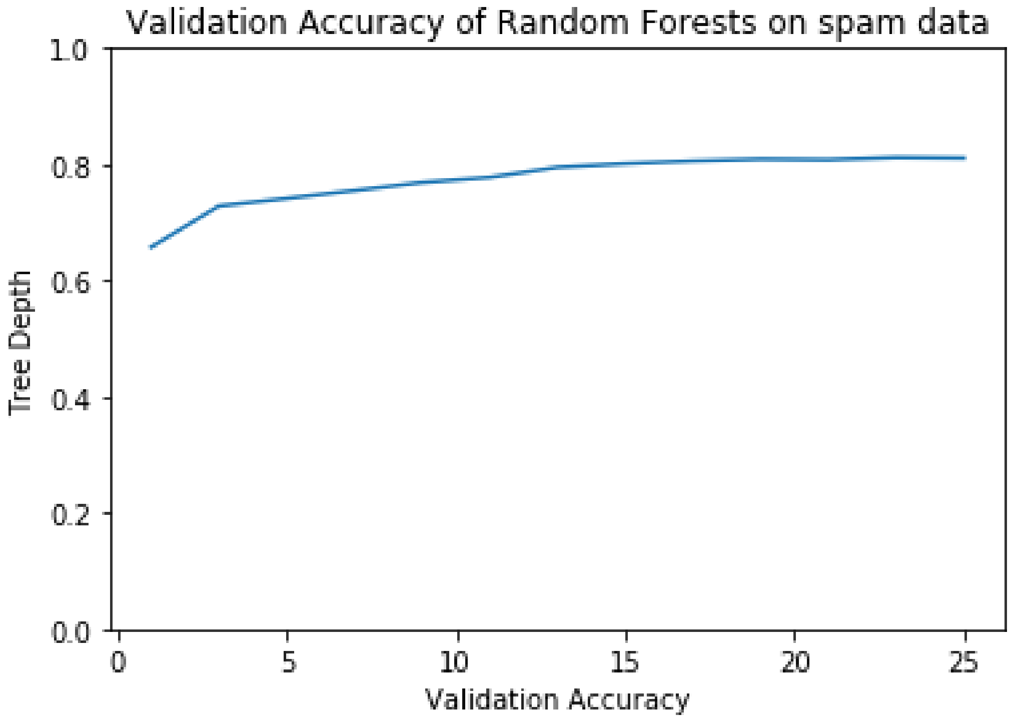
Use this procedure to determine the optimal value for the tree depth hyperparameter. (Re-use the function looping over depths in spam)

```
In [21]: maxdepth = 25
         depthstep = 2
```

```
In [22]: spamRF_Accs=testdepths('RF',spamtraindata,spamtrainlabels,spamvaldata,spam
```

```
Tested up to depth 5
Tested up to depth 15
Tested up to depth 25
```

```
In [23]: plotdepthAccs(spamRF_Accs,'Validation Accuracy of Random Forests on spam
```



```
Max accuracy of 81.18% for depth of 23
```

Make random forest predictions for the best depth according to test data, and save to a csv file for upload to Kaggle (0-indexing).

```
In [114]: spamRFoutfilename ='../spamRF_accuracies.txt'
          bestspamRF = train_optimal('RF',np.array([[23,1],[0,0]]),spamdata,spamlab
          #bestspamRF = train_optimal(spamRF_Accs,spamdata,spamlabels,spamoutfilena
          write_Accs(bestspamRF,
                     [spamtraindata,spamvaldata],
                     [spamtrainlabels,spamvallabels],
                     ['Training','Validation'],
                     spamRFoutfilename,
                    )
```

```
In [24]: #spamRFpredictions_csv = save_optimal(spamRF_Accs,spamdata,spamlabels,spam
         #np.savetxt(BASE_DIR+'spam_RF_testpredictions.csv',spamRFpredictions_csv,f
```

1.2 ## Census

Calculate decision tree and random forest training/validation accuracies for the census dataset.

Begin by importing (preprocessed) data, shuffling, and separating into training and validation sets.

```
In [7]: # Import
        CENSDAT_PATH = "Data/census_traindata_vec.csv"
        CENSLBL_PATH = "Data/census_traindata_lbl.csv"
        CENSTST_PATH = "Data/census_testdata_vec.csv"

        census_data = np.genfromtxt(BASE_DIR+CENSDAT_PATH,delimiter=',')
        census_labels = np.genfromtxt(BASE_DIR+CENSLBL_PATH,delimiter=',')
        census_test = np.genfromtxt(BASE_DIR+CENSTST_PATH,delimiter=',')
```

```
In [8]: # Shuffle
        census_labels = np.reshape(census_labels,(len(census_labels),1))
        censusdata, censuslabels = ut.shuffle_data(census_data,census_labels)
```

```
In [9]: # Separate
        censustraindata,censusvaldata = ut.val_partition(censusdata,valfrac)
        censustrainlabels,censusvallabels = ut.val_partition(censuslabels,valfrac)
```

1.2.1 Census Decision Tree

Create and train a decision tree classifier using the census data and labels

```
In [1]: census_DTclassifier = dt.DecisionTree(treedepth=8)


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-1-e96f23cc0db3> in <module>()
    ----> 1 census_DTclassifier = dt.DecisionTree(treedepth=8)


        NameError: name 'dt' is not defined
```

```
In [29]: census_DTclassifier.train(censustraindata,censustrainlabels)
```

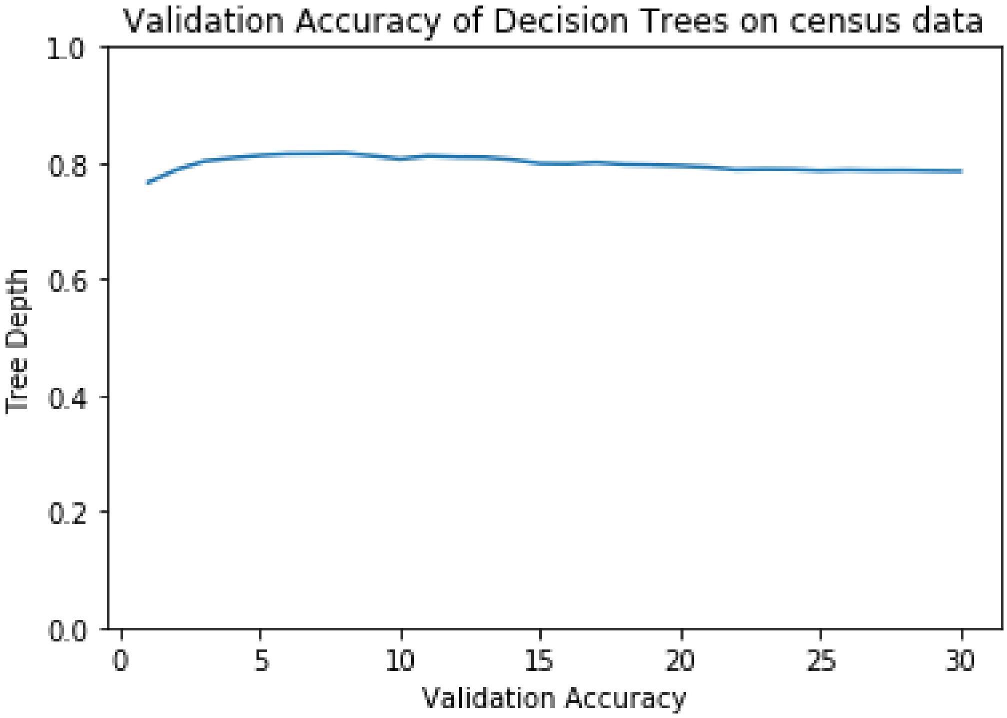
Calculate the validation accuracy using the trained decision tree

```
In [30]: censuspredictions = census_DTclassifier.predict(censusvaldata)
```

```
In [31]: censusvalAcc = ut.val_accuracy(censuspredictions,censusvallabels)
         print('Validation Accuracy = %.3f%%'%(100*censusvalAcc))

Validation Accuracy = 81.601%
```

Use this procedure to determine the optimal value for the tree depth hyperparameter. (Re-use the functions for finding the optimal depth in spam)

```
In [32]: maxdepth = 30
         depthstep = 1

In [33]: censusDT_Accs=testdepths('DT',censustraindata,censustrainlabels,censusvalc

Tested up to depth 5
Tested up to depth 10
Tested up to depth 15
Tested up to depth 20
Tested up to depth 25
Tested up to depth 30


In [34]: plotdepthAccs(censusDT_Accs,'Validation Accuracy of Decision Trees on cens
```



```
Max accuracy of 81.69% for depth of 8
```

Make predictions for the test data, and save to a csv file for upload to Kaggle (1-indexing).

```
In [22]: censusDToutfilename ='../censusDT_accuracies.txt'
         bestcensusDT = train_optimal('DT',np.array([[8,1],[0,0]]),censusdata,censu
         #bestcensusDT = train_optimal(censusDT_Accs,censusdata,censuslabels,census
         write_Accs(bestcensusDT,
                    [censustraindata,censusvaldata],
                    [censustrainlabels,censusvallabels],
                    ['Training','Validation'],
                    censusDToutfilename,
                   )
```

```
In [35]: #censusDTpredictions_csv = save_optimal(censusDT_Accs,censusdata,censuslab
         #np.savetxt(BASE_DIR+'census_DT_testpredictions.csv',censusDTpredictions_c
```

1.2.2 Census Random Forest

Create and train a random forest classifier using the census data and labels.

```
In [25]: census_RFclassifier = rf.RandomForest(treedepth=8,ntrees=100)
```

```
In [26]: census_RFclassifier.train(censustraindata,censustrainlabels)
```

Calculate the validation accuracy using the trained decision tree.

```
In [27]: censuspredictions = census_RFclassifier.predict(censusvaldata)
```

```
In [28]: censusvalAcc = ut.val_accuracy(censuspredictions,censusvallabels)
         print('Validation Accuracy = %.3f%%'%(100*censusvalAcc))
```

```
Validation Accuracy = 82.182%
```

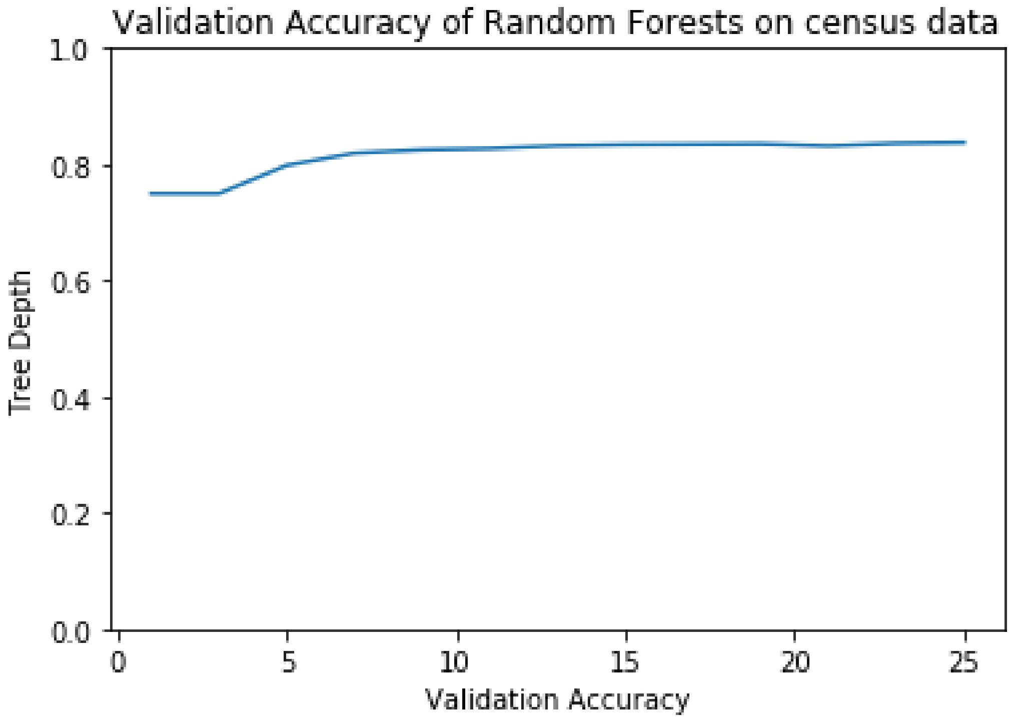
Use this procedure to determine the optimal value for the tree depth hyperparameter. (Re-use the function looping over depths in spam)

```
In [29]: maxdepth = 25
         depthstep = 2
```

```
In [30]: censusRF_Accs=testdepths('RF',censustraindata,censustrainlabels,censusvalc
```

```
Tested up to depth 5...
Tested up to depth 15...
Tested up to depth 25...
```

```
In [31]: plotdepthAccs(censusRF_Accs,'Validation Accuracy of Random Forests on cens
```



```
Max accuracy of 83.77% for depth of 25
```

```
In [15]: censusRFoutfilename ='../censusRF_accuracies.txt'
         bestcensusRF = train_optimal('RF',np.array([[20,1],[0,0]]),censusdata,cens
         #bestcensusRF = train_optimal(censusRF_Accs,censusdata,censuslabels,census
         write_Accs(bestcensusRF,
                    [censustraindata,censusvaldata],
                    [censustrainlabels,censusvallabels],
                    ['Training','Validation'],
                    censusRFoutfilename,
                   )
```

```
Finished training 0 tree(s) out of 100
Finished training 5 tree(s) out of 100
Finished training 10 tree(s) out of 100
Finished training 15 tree(s) out of 100
Finished training 20 tree(s) out of 100
Finished training 25 tree(s) out of 100
Finished training 30 tree(s) out of 100
Finished training 35 tree(s) out of 100
Finished training 40 tree(s) out of 100
Finished training 45 tree(s) out of 100
Finished training 50 tree(s) out of 100
Finished training 55 tree(s) out of 100
Finished training 60 tree(s) out of 100
Finished training 65 tree(s) out of 100
Finished training 70 tree(s) out of 100
Finished training 75 tree(s) out of 100
Finished training 80 tree(s) out of 100
Finished training 85 tree(s) out of 100
Finished training 90 tree(s) out of 100
Finished training 95 tree(s) out of 100
```

```
In [ ]: bestcensusRF = train_optimal('RF',np.array([[20,1],[0,0]]),censusdata,censu
        make_kaggle(bestcensusRF,census_test,BASE_DIR+'census_RF_testpredictions.cs
```

```
Finished training 0 tree(s) out of 100
Finished training 5 tree(s) out of 100
Finished training 10 tree(s) out of 100
Finished training 15 tree(s) out of 100
Finished training 20 tree(s) out of 100
Finished training 25 tree(s) out of 100
Finished training 30 tree(s) out of 100
Finished training 35 tree(s) out of 100
```

1.3 ## Titanic

Calculate decision tree and random forest training/validation accuracies for the Titanic dataset.

Begin by importing (preprocessed) data, shuffling, and separating into training and validation sets.

```
In [16]: # Import
         TITADAT_PATH = "Data/titanic_traindata_vec.csv"
         TITALBL_PATH = "Data/titanic_traindata_lbl.csv"
         TITATST_PATH = "Data/titanic_testdata_vec.csv"

         titanic_data = np.genfromtxt(BASE_DIR+TITADAT_PATH,delimiter=',')
         titanic_labels = np.genfromtxt(BASE_DIR+TITALBL_PATH,delimiter=',')
         titanic_test = np.genfromtxt(BASE_DIR+TITATST_PATH,delimiter=',')
```

```
In [17]: # Shuffle
         titanic_labels = np.reshape(titanic_labels,(len(titanic_labels),1))
         titanicdata, titaniclabels = ut.shuffle_data(titanic_data,titanic_labels)
```

```
In [18]: # Separate
         titanictraindata,titanicvaldata = ut.val_partition(titanicdata,valfrac)
         titanictrainlabels,titanicvallabels = ut.val_partition(titaniclabels,valfr
```

1.3.1 Titanic Decision Tree

Create and train a decision tree classifier using the Titanic data and labels.

```
In [19]: titanic_DTclassifier = dt.DecisionTree(treedepth=8)
```

```
In [20]: titanic_DTclassifier.train(titanictraindata,titanictrainlabels)
```

Calculate the validation accuracy using the trained decision tree

```
In [21]: titanicpredictions = titanic_DTclassifier.predict(titanicvaldata)
```

```
In [22]: titanicvalAcc = ut.val_accuracy(titanicpredictions,titanicvallabels)
         print('Validation Accuracy = %.3f%%'%(100*titanicvalAcc))
```

```
Validation Accuracy = 69.000%
```

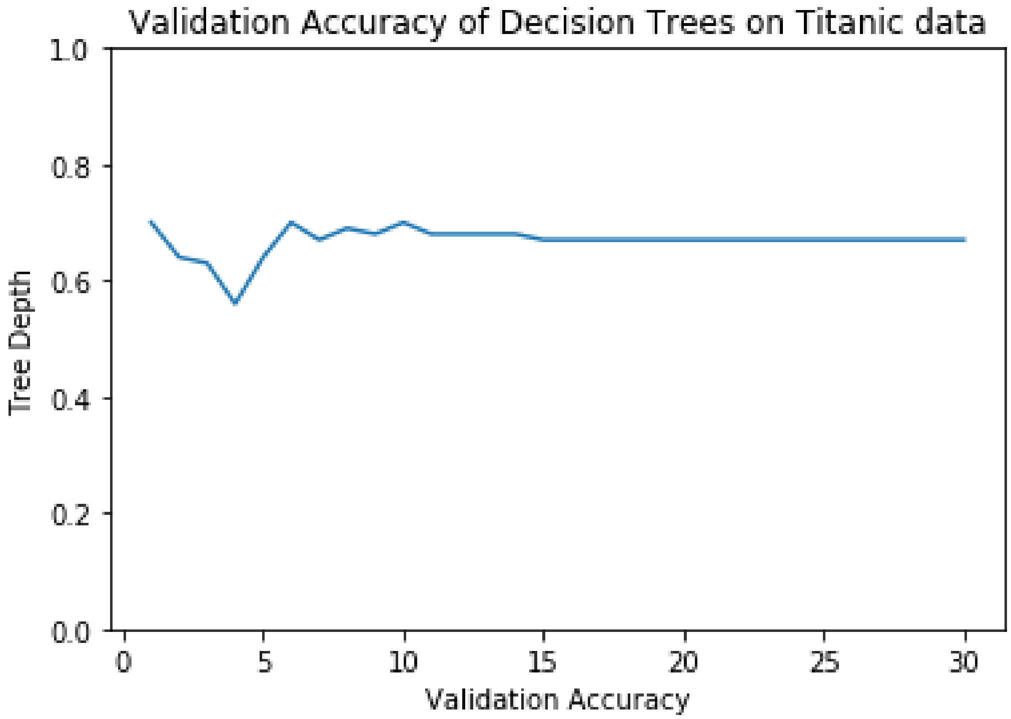
Use this procedure to determine the optimal value for the tree depth hyperparameter. (Re-use the function looping over depths in spam)

```
In [23]: maxdepth = 30
         depthstep = 1
```

```
In [24]: titanicDT_Accs=testdepths('DT',titanictraindata,titanictrainlabels,titanic
```

```
Tested up to depth 5...
Tested up to depth 10...
Tested up to depth 15...
Tested up to depth 20...
Tested up to depth 25...
Tested up to depth 30...
```

```
In [25]: plotdepthAccs(titanicDT_Accs,'Validation Accuracy of Decision Trees on Tit
```



```
Max accuracy of 70.00% for depth of 1
```

```
In [28]: titanicDToutfilename ='../titanicDT_accuracies.txt'
         besttitanicDT = train_optimal('DT',np.array([[6,1],[0,0]]),titanicdata,tit
         #besttitanicDT = train_optimal(titanicDT_Accs,titanicdata,titaniclabels,ti
         write_Accs(besttitanicDT,
                    [titanictraindata,titanicvaldata],
                    [titanictrainlabels,titanicvallabels],
                    ['Training','Validation'],
                    titanicDToutfilename,
                   )
```

```
In [43]: #titanicDTpredictions_csv = save_optimal(titanicRF_Accs,titanicdata,titani
         #np.savetxt(BASE_DIR+'titanic_DT_testpredictions.csv',titanicDTprediction
```

1.3.2 Titanic Random Forest

Create and train a random forest classifier using the Titanic data and labels.

```
In [44]: titanic_RFclassifier = rf.RandomForest(treedepth=8,ntrees=100)
```

```
In [45]: titanic_RFclassifier.train(titanictraindata,titanictrainlabels)
```

Calculate the validation accuracy using the trained decision tree.

```
In [46]: titanicpredictions = titanic_RFclassifier.predict(titanicvaldata)
```

```
In [47]: titanicvalAcc = ut.val_accuracy(titanicpredictions,titanicvallabels)
         print('Validation Accuracy = %.3f%%'%(100*titanicvalAcc))
```

```
Validation Accuracy = 77.000%
```

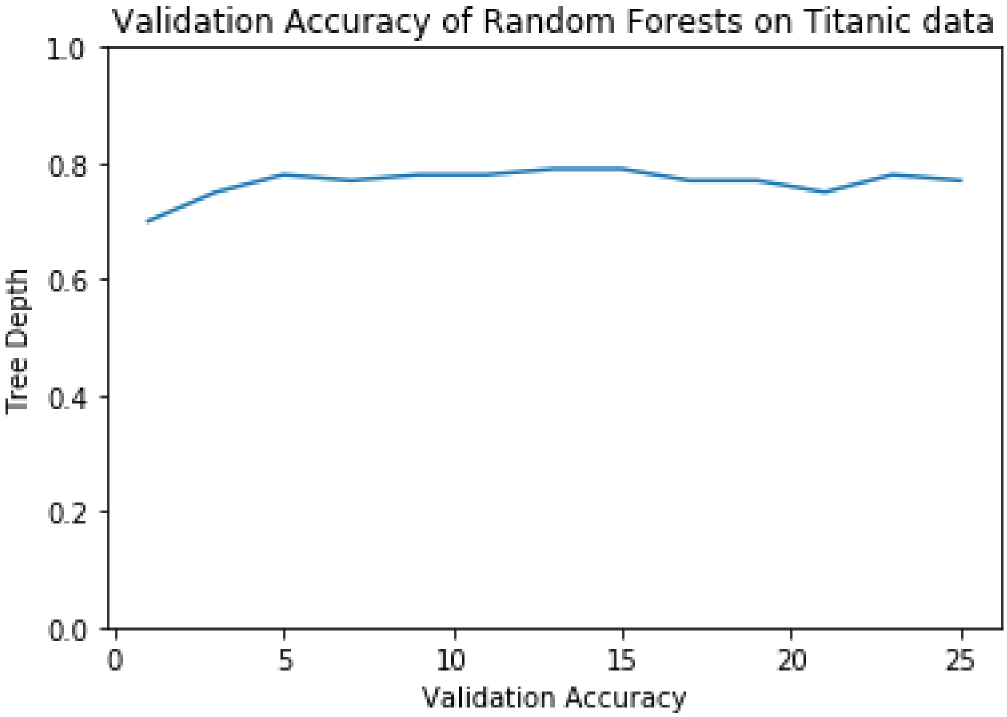
Use this procedure to determine the optimal value for the tree depth hyperparameter. (Re-use the function looping over depths in spam)

```
In [48]: maxdepth = 25
         depthstep = 2
```

```
In [49]: titanicRF_Accs=testdepths('RF',titanictraindata,titanictrainlabels,titanic
```

```
Tested up to depth 5...
Tested up to depth 15...
Tested up to depth 25...
```

```
In [50]: plotdepthAccs(titanicRF_Accs,'Validation Accuracy of Random Forests on Tit
```



```
Max accuracy of 79.00% for depth of 13
```

```
In [29]: titanicRFoutfilename ='../titanicRF_accuracies.txt'
         besttitanicRF = train_optimal('RF',np.array([[13,1],[0,0]]),titanicdata,ti
         #besttitanicDT = train_optimal(titanicDT_Accs,titanicdata,titaniclabels,ti
         write_Accs(besttitanicRF,
                    [titanictraindata,titanicvaldata],
                    [titanictrainlabels,titanicvallabels],
                    ['Training','Validation'],
                    titanicRFoutfilename,
                   )
```

```
Finished training 0 tree(s) out of 100
Finished training 5 tree(s) out of 100
Finished training 10 tree(s) out of 100
Finished training 15 tree(s) out of 100
Finished training 20 tree(s) out of 100
Finished training 25 tree(s) out of 100
Finished training 30 tree(s) out of 100
Finished training 35 tree(s) out of 100
Finished training 40 tree(s) out of 100
Finished training 45 tree(s) out of 100
Finished training 50 tree(s) out of 100
Finished training 55 tree(s) out of 100
Finished training 60 tree(s) out of 100
Finished training 65 tree(s) out of 100
Finished training 70 tree(s) out of 100
Finished training 75 tree(s) out of 100
Finished training 80 tree(s) out of 100
Finished training 85 tree(s) out of 100
Finished training 90 tree(s) out of 100
Finished training 95 tree(s) out of 100
```

```
In [21]: besttitanicRF = train_optimal('RF',np.array([[13,1],[0,0]]),titanicdata,ti
         make_kaggle(besttitanicRF,titanic_test,BASE_DIR+'titanic_RF_testpredictior
```

```
Finished training 0 tree(s) out of 100
Finished training 5 tree(s) out of 100
Finished training 10 tree(s) out of 100
Finished training 15 tree(s) out of 100
Finished training 20 tree(s) out of 100
Finished training 25 tree(s) out of 100
Finished training 30 tree(s) out of 100
Finished training 35 tree(s) out of 100
Finished training 40 tree(s) out of 100
Finished training 45 tree(s) out of 100
Finished training 50 tree(s) out of 100
Finished training 55 tree(s) out of 100
Finished training 60 tree(s) out of 100
Finished training 65 tree(s) out of 100
Finished training 70 tree(s) out of 100
Finished training 75 tree(s) out of 100
Finished training 80 tree(s) out of 100
Finished training 85 tree(s) out of 100
Finished training 90 tree(s) out of 100
Finished training 95 tree(s) out of 100
```

```
In [ ]:
```