Comp hw2

Allen kombasseril
z5232188

Q1

Part a

DecisionTreeClassifier

Dataset	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%
australian	79.86%	81.29%	82.91%	82.02%	82.17%	81.45%	82.03%	83.34%	83.33%	82.33%
balance-scale	75.67%	75.99%	76.98%	77.92%	77.30%	78.10%	77.95%	77.62%	78.09%	77.45%
hypothyroid	99.42%	99.52%	99.20%	99.28%	99.23%	99.31%	99.34%	99.52%	99.52%	99.47%

BernoulliNB with priors

Dataset	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%
australian	79.84%	81.14%	81.28%	81.29%	82.74%	82.16%	82.16%	82.74%	82.74%	82.74%
balance-scale	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%
hypothyroid	92.26%	92.23%	92.23%	92.18%	92.21%	92.23%	92.26%	92.26%	92.26%	92.23%

Part b

(3) most of the 6 models show a learning curve

(5) Some Bernoulli Naive Bayes models are better than Decision Tree models

Part c

BernoulliNB without priors

Dataset	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%
australian	73.62%	79.27%	81.44%	78.98%	78.40%	79.69%	78.52%	79.83%	80.41%	80.41%
balance-scale	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%	46.08%
hypothyroid	83.88%	79.59%	77.44%	74.79%	73.12%	65.05%	53.60%	51.30%	51.09%	50.26%

1) BNB performs better with priors

Q2

Part A:

Training dataset accuracy: 85.64516129032258
Test dataset accuracy: 82.77153558052435

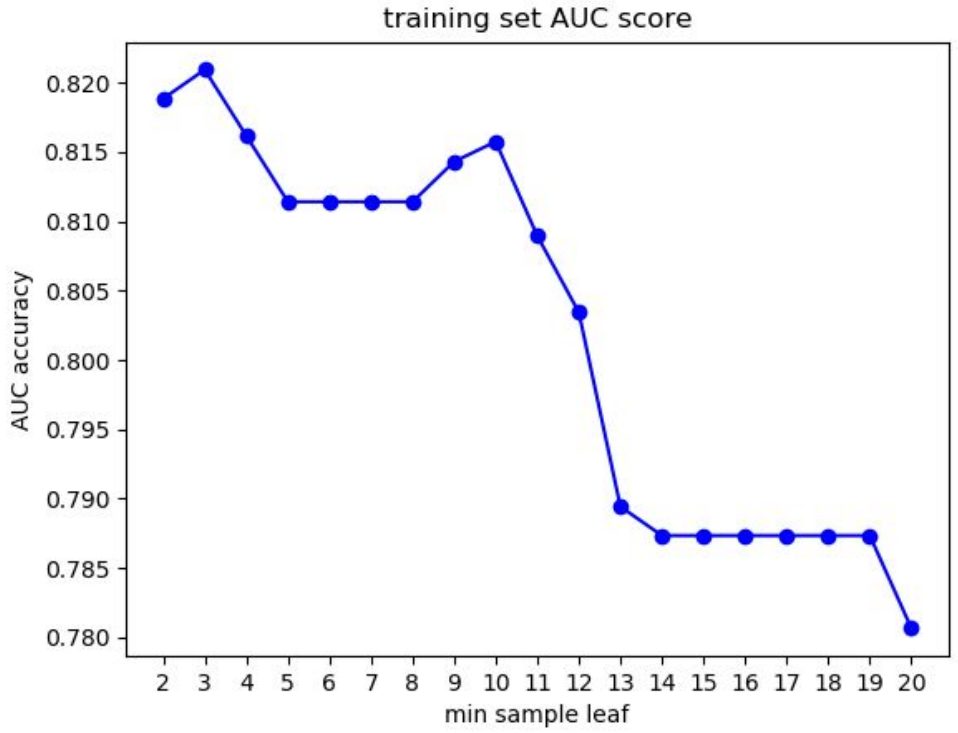
Part B:

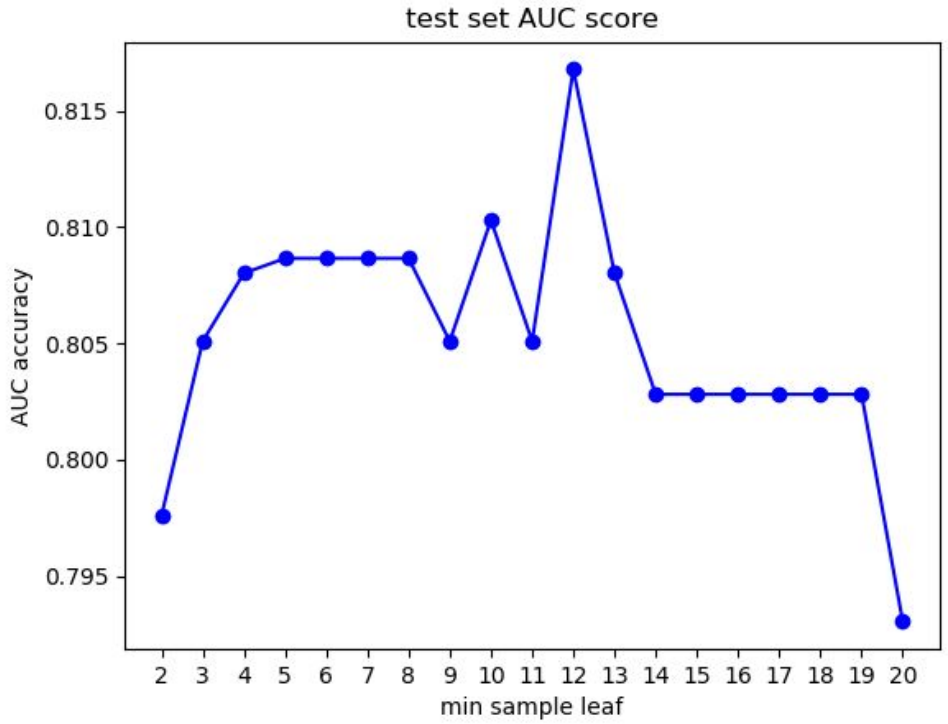
Optimal number of leaves is 3 when comparing the train accuracy

Optimal number of leaves is 12 when comparing the test accuracy:

But i believe we have to choose the model which performs best in the test set,
So optimal number of leaves is 12.

Part C:





Part D

$$p(S = true|G = female, C = 1) = \frac{p(G=female,C=1|S=true).P(S=true)}{p(G=female,C=1|S=true).P(S=true)+p(G=female,C=1|S=False).P(S=false)}$$

$$= \frac{\frac{342}{887} * \frac{45}{342}}{\frac{342}{887}\frac{45}{342} + \frac{77}{545}\frac{545}{887}} = 0.368852459$$

Source code question 1:

COMP9417 19T3 Homework 2: Applying and Implementing Machine Learning

Question 1

Question 1 – Learning curve

The number of data instances required to effectively learn the target function depends on dataset characteristics and the learning algorithm. In this question, you will interpret results of learning three different datasets by two machine learning algorithms trained on varying size of data, from 5% to 90% of the dataset. After training on a fraction of the dataset, the model is tested using the rest of the dataset. For example, if 40% of data is used for training, the remaining 60% of the dataset is used for testing the trained model. The datasets are from different, real-world domains, and vary in size from a few hundred to a couple of thousand instances. For a smoother learning curve, the training is done in a cross validation fashion.

Running the classifiers

1(a). [0.5 mark]

Run the code section in the notebook cells below. This will generate a table of results, which you should copy and paste **WITHOUT MODIFICATION** into you report as your answer for "Question 1(a)".

The output of the code section is a table, which represents the percentage accuracy of classification for the decision tree algorithm. Each of the columns shows accuracy for a model trained on a different fraction of the dataset.

Result interpretation

Answer these questions in your report file. Your answers must be based on the results table you saved in "Question 1(a)".

1(b). [0.5 mark] Refer to Homework2.pdf file.

1(c). [0.5 mark] Refer to Homework2.pdf file.

In [1]:
```
# Code for question 1

import numpy as np
from scipy.io import arff
import pandas as pd
from sklearn.base import TransformerMixin
from sklearn import tree
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV, learning_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
import sys
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# fixed random seed
np.random.seed(1)
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn

def label_enc(labels):
    le = preprocessing.LabelEncoder()
    le.fit(labels)
    return le
```

In [3]:
```
def load_data(path):
    dataset = arff.loadarff(path)
    data = pd.DataFrame(dataset[0])
    attr = np.array(data.columns)
    data = DataFrameImputer().fit_transform(data).values

    # mask categorical features
    masks = []
    for i in range(len(attr)-1):
        if isinstance(attr[i][1],float):
            masks.append(i)
    return data, masks

class DataFrameImputer(TransformerMixin):

    def fit(self, X, y=None):

        self.fill = pd.Series([X[c].value_counts().index[0]
            if X[c].dtype == np.dtype('O') else X[c].mean() for c in X],
            index=X.columns)

        return self

    def transform(self, X, y=None):
        return X.fillna(self.fill)

def get_method_scores(data, method):
    X = data[:,0:data.shape[1]-1]
    lenc = label_enc(data[:,data.shape[1]-1])
    y = lenc.transform(data[:,data.shape[1]-1])
    train_sizes = np.array([0.05, .1, .15, .2, .25, .3, .35, .4, .45, .5])
    _, train_scores, test_scores = learning_curve(method, X, y, cv=5,
                                                  train_sizes=train_sizes,
                                                  scoring=None, shuffle=False, random_state=0,
                                                  error_score=0)
    return test_scores
```

In [4]:
```
def test_method(method, title):
    # load data
    paths = ['australian','balance-scale','hypothyroid']
    scores = []

    for path in paths:
        score = []
        path += '.arff'
        data, masks = load_data(path)

        # training on data with different portions of training data
        score_array = get_method_scores(data, method)
        # we got a [num portions][num folds] array, need to avg them into
        # a list of scores for each portion
        for ar in score_array:
            score.append(np.mean(ar))
        scores.append(score)

    # print the results
    method_name = method.__class__.__name__+' '+title
    header = "{:^75}".format(method_name) + '\n' + '-' * 105  + '\n' + \
    "{:^13} | {:^6} | {:^6} | {:^6} | {:^6} | {:^6} | {:^6} | {:^6} | {:^6} | {:^6} |" \
    .format("Dataset", "5%", "10%", "15%", "20%", "25%", "30%", "35%", "40%", "45%", "50%") + \
     '\n' + '-' * 105

    # print result table
    print(header)
    for i in range(len(scores)):
        print("{:<14}".format(paths[i]),end="")
        for j in range(len(scores[i])):
            print("| {:>6.2%} ".format(scores[i][j]),end="")
        print('|')
    print('\n')

test_method(DecisionTreeClassifier(random_state=0),'')
test_method(BernoulliNB(),'with priors')
test_method(BernoulliNB(fit_prior=False),'without priors')
```

Source code question 2:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import sklearn
from sklearn import tree
from sklearn import metrics
import pydotplus
import graphviz
from IPython.display import Image
from sklearn.model_selection import GridSearchCV


def preprocessing(csvfile): #function for preprocessing

    df=pd.read_csv(csvfile)         #reading csv file
    df = (df - df.min()) / (df.max() - df.min())  #normalizing

    return df


def split_rows(dataframe):        #function to split dataframe
    return dataframe.iloc[0:620],dataframe.iloc[620:887]



def train_test_split(df):             #function to splitting dataset
    X_train, X_test=split_rows(df)        # seperating to test and train
    y_train= X_train.pop('Survived')
    y_test= X_test.pop('Survived')
    y_train= y_train.to_frame()
    y_test = y_test.to_frame()

    return X_train, X_test, y_train, y_test


y=preprocessing('titanic.csv')
print(y)

print("columns",y.index)
```

```
# Get a bool series representing which row satisfies the condition i.e. True for
# row in which value of 'Age' column is more than 30

survived_data= y.loc[y['Survived'] == 1]
not_survived_data=y.loc[y['Survived']==0]
print('survived',survived_data)
print("len of full data",len(y))
print("len of P(s=true)",len(survived_data))
print("len of P(s=false)",len(not_survived_data))
seriesObj_1 = survived_data.loc[survived_data['Sex'] == 1]
seriesObj_11=seriesObj_1.loc[seriesObj_1['Pclass']==0]
seriesObj_2 = not_survived_data.loc[not_survived_data['Sex'] == 1]
seriesObj_21=seriesObj_2.loc[seriesObj_2['Pclass']==0]


print(' G=female,C=1 | S= true ', len(seriesObj_11))
print(' G=female,C=1 | S= false ', len(seriesObj_21))




y.to_csv('preprocessed.csv')


x_train,x_test,y_train,y_test = train_test_split(y)
x_train.to_csv('x_train.csv')
x_test.to_csv('x_test.csv')
y_train.to_csv('y_train.csv')
y_test.to_csv('y_test.csv')
#print(y)

x=pd.concat([x_train,x_test])
y=pd.concat([y_train,y_test])
print(x,y)
```

```
# The decision tree classifier.
clf = tree.DecisionTreeClassifier()
# Training the Decision Tree
clf_train = clf.fit(x_train, y_train)

y_pred_test=clf_train.predict(x_test)
y_pred_train=clf_train.predict(x_train)


print("test Accuracy:",metrics.accuracy_score(y_test, y_pred_test))
print("train accuracry:",metrics.accuracy_score(y_train,y_pred_train))




# Export/Print a decision tree in DOT format.
#print(tree.export_graphviz(clf_train, None))

#Create Dot Data
dot_data = tree.export_graphviz(clf_train, out_file=None, feature_names=list(x_train.columns.values),
                                class_names=['Not survived', 'survived'], rounded=True, filled=True) #Gini deci
#Create Graph from DOT data
graph = pydotplus.graph_from_dot_data(dot_data)




param_grid = { 'min_samples_leaf':list(range(2, 21))}
t=tree.DecisionTreeClassifier()
#print(metrics.SCORERS.keys())

grid_search_tree = GridSearchCV(estimator = t,scoring ='roc_auc' , param_grid = param_grid, cv = 10, n_jobs = -
grid_search_tree.fit(x, y)
print("best parameter",grid_search_tree.best_params_)
best_grid = grid_search_tree.best_estimator_
y_pred=best_grid.predict(x_test)
print("bestscore,",grid_search_tree.best_score_)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("????")
#print(grid_search_tree.cv_results_)
```

```
grid_search_tree = GridSearchCV(estimator = t,scoring ='roc_auc', param_grid = param_grid, cv = 10, n_jobs = -1, verbose = 2)
grid_search_tree.fit(x, y)
print("best parameter",grid_search_tree.best_params_)
best_grid = grid_search_tree.best_estimator_
y_pred=best_grid.predict(x_test)
print("bestscore,",grid_search_tree.best_score_)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("????")
#print(grid_search_tree.cv_results_)

min_leaf=[]
test_accuracy=[]
train_accuracy=[]
best=0

for i in range(2,21):   #?? loop for finding optimal min number of leafs
    # The decision tree classifier.
    dt = tree.DecisionTreeClassifier(min_samples_leaf=i)
    # Training the Decision Tree
    dt_trained = dt.fit(x_train, y_train)

    y_pred_test = dt_trained.predict(x_test)
    y_pred_train= dt_trained.predict(x_train)

    #print(i)

    if (metrics.roc_auc_score(y_test, y_pred_test) > best):
        optimal_leaf = i
        best=metrics.roc_auc_score(y_test, y_pred_test)
    test_accuracy.append(metrics.roc_auc_score(y_test, y_pred_test))
    min_leaf.append(i)
    train_accuracy.append(metrics.roc_auc_score(y_train, y_pred_train))

    print("\n")

print("optimal_leaf:",optimal_leaf)

print("test_accuracy",test_accuracy)
print("train_accuracy",train_accuracy)
```

```
plt.plot(min_leaf, train_accuracy,marker='o', color='blue')
plt.xlabel('min sample leaf')
plt.ylabel('AUC accuracy')
plt.title('training set AUC score')
plt.xticks(np.arange(2, 21, 1.0))
plt.show()
```